

# A Note on Machine Learning Methods

Ying Nian Wu, UCLA Statistics, Based on M231B lectures

Updated March 2020

## Contents

<b>1</b>	<b>Ptolemy's Epicycle and Gauss Paradigm</b>	<b>4</b>
1.1	The model . . . . .	4
1.2	Boosting machine . . . . .	6
1.3	Lasso, ridge and kernel machine . . . . .	6
1.4	Neural network . . . . .	6
1.5	Model complexity and regularization . . . . .	6
1.6	Ptolemy or Newton? . . . . .	7
1.7	Euler's linear model . . . . .	7
1.8	Laplace's estimating equation . . . . .	8
1.9	Gauss paradigm . . . . .	8
1.10	Continuing Gauss paradigm . . . . .	9
1.11	Three modes of learning . . . . .	9
1.12	Bayesian, Frequentist, variational . . . . .	11
<b>2</b>	<b>Basics: Linear Models</b>	<b>12</b>
2.1	Linear regression . . . . .	12
2.2	Ridge regression and shrinkage estimator . . . . .	14
2.3	Linear spline . . . . .	14
2.4	Lasso regression . . . . .	15
2.5	Logistic regression . . . . .	17
2.6	Classification and perceptron . . . . .	17
2.7	Loss functions . . . . .	18
2.8	Regularization . . . . .	21
2.9	Gradient descent: learning from errors . . . . .	21
2.10	Iterated reweighed least squares . . . . .	23
2.11	Multivariate or multinomial response . . . . .	24
2.12	Non-linear and non-parametric functions . . . . .	25
<b>3</b>	<b>Model Complexity and Overfitting</b>	<b>26</b>
3.1	Gauss' analysis of least squares . . . . .	26
3.2	Bias and variance tradeoff . . . . .	27
3.3	Stein's estimator . . . . .	27
3.4	Stein's estimator as empirical Bayes . . . . .	29
3.5	Model bias . . . . .	29
3.6	Training and testing errors, overfitting . . . . .	30

3.7	Learning from noise . . . . .	30
3.8	Effective degrees of freedom . . . . .	31
3.9	Classification . . . . .	31
3.10	Learning from coin flipping and Rademacher complexity . . . . .	32
3.11	Growth number and VC-dimension . . . . .	32
3.12	Symmetrization . . . . .	32
<b>4</b>	<b>Kernel Regression</b>	<b>33</b>
4.1	Ridge regression . . . . .	33
4.2	Re-representing ridge regression . . . . .	33
4.3	Kernel trick . . . . .	34
4.4	Representer theorem . . . . .	35
4.5	Kernel regression . . . . .	35
4.6	Reproducing kernel Hilbert space . . . . .	36
4.7	Kernel version of representer theorem . . . . .	36
4.8	Mercer theorem . . . . .	37
4.9	Nearest neighbors interpolation . . . . .	37
<b>5</b>	<b>Gaussian Process</b>	<b>37</b>
5.1	Bayesian interpretation of ridge regression . . . . .	39
5.2	Kernel as covariance . . . . .	39
5.3	Prediction by conditioning . . . . .	39
5.4	Posterior interval . . . . .	40
5.5	Marginal likelihood for hyper-parameter . . . . .	40
<b>6</b>	<b>Kernel SVM</b>	<b>40</b>
6.1	Hinge loss and linear SVM . . . . .	41
6.2	Re-representing hinge loss . . . . .	41
6.3	Primal min-max to dual max-min . . . . .	41
6.4	Dual coordinate descent . . . . .	42
6.5	Kernel trick . . . . .	42
6.6	Support vectors and nearest neighbors template matching . . . . .	43
6.7	Max margin . . . . .	43
6.8	Primal dual . . . . .	44
6.9	Kernel trick . . . . .	45
6.10	Non-separable case and slack variables . . . . .	45
6.11	Primal form with hinge loss . . . . .	46
6.12	Bias and sequential minimal optimization (SMO) . . . . .	46
6.13	Max margin = min distance . . . . .	47
<b>7</b>	<b>Boosting Machines</b>	<b>47</b>
7.1	Classification and regression trees (CART) . . . . .	47
7.2	Random forest . . . . .	49
7.3	Adaboost . . . . .	50
7.4	Gradient boosting . . . . .	52
7.5	Extreme gradient boosting (XGB) as iterated reweighed least squares . . . . .	54

<b>8</b>	<b>Neural Networks</b>	<b>54</b>
8.1	Two layer perceptron . . . . .	55
8.2	Implicit regularization by gradient descent . . . . .	57
8.3	Connection to kernel machine . . . . .	58
8.4	Connection to Gaussian process . . . . .	58
8.5	Multi-layer network . . . . .	58
8.6	Multi-layer back-propagation . . . . .	59
8.7	Stochastic gradient descent (SGD) . . . . .	60
8.8	Convolutional neural networks (CNN, ConvNet) . . . . .	62
8.9	Batch normalization . . . . .	64
8.10	Residual net . . . . .	65
8.11	Recurrent neural networks (RNN), LSTM, GRU . . . . .	66
8.12	Encoder-decoder, thought vector . . . . .	68
8.13	Memory and attention, query and key . . . . .	69
8.14	Word2vec for semantic embedding . . . . .	69
8.15	Self-attention, Transformer, GPT, BERT . . . . .	70
<b>9</b>	<b>Representation Learning with Deep Nets</b>	<b>70</b>
9.1	Three lessons of deep learning . . . . .	70
9.2	Unsupervised learning . . . . .	71
9.3	Thought vector, auto-encoding, embedding, disentangling . . . . .	71
9.4	Kullback-Leibler divergence . . . . .	71
9.5	Decoder . . . . .	73
9.6	Generative adversarial networks (GAN) . . . . .	73
9.7	Variational auto-encoder (VAE) as alternating projection . . . . .	74
9.8	VAE and EM . . . . .	77
9.9	Flow-based model . . . . .	77
<b>10</b>	<b>Representation Learning without Deep Nets</b>	<b>78</b>
10.1	Factor analysis and generalizations . . . . .	78
10.2	K-means as one-hot vector representation . . . . .	79
10.3	Spectral embedding and clustering . . . . .	79
10.4	Multi-dimensional scaling as mirroring . . . . .	79
10.5	t-Stochastic neighborhood embedding (tSNE) as energy-based model . . . . .	80
10.6	Local linear embedding (LLE) . . . . .	81
10.7	Topic model and latent Dirichelet allocation (LDA) . . . . .	81
<b>11</b>	<b>Reinforcement Learning</b>	<b>81</b>
11.1	Alpha Go . . . . .	82
11.2	Alpha Go Zero . . . . .	84
11.3	Atari by Q learning . . . . .	85
11.4	Markov decision process (MDP) . . . . .	86
11.5	Model-based planning: play out in imagination . . . . .	87
11.6	Model-free learning: play out in real life . . . . .	87
11.7	Temporal difference bootstrap . . . . .	87
11.8	Q learning . . . . .	87
11.9	REINFORCE, MLE, re-parametrization . . . . .	88
11.10	Policy gradient: actor and critic . . . . .	89

11.11	Partially observed MDP (POMDP)	91
11.12	Multi-agent reinforcement learning	91
11.13	Inverse reinforcement learning (IRL)	91
11.14	Energy-based model	91
<b>12</b>	<b>Causal Learning</b>	<b>92</b>
12.1	Counterfactual, potential outcome	92
12.2	Matching and propensity score	93
12.3	Structural equations, causal diagram	93
12.4	Training data and testing data	93
12.5	Do calculus, back door and front door adjustment	94
<b>13</b>	<b>Background: Convex Optimization and Duality</b>	<b>94</b>
13.1	von Neumann minimax theorem	94
13.2	Constrained optimization and Lagrange multipliers	96
13.3	Legendre-Fenchel convex conjugate	99

## Preface: “no time to be brief”

**Credits:** Most of the figures in the note are taken from the web, and the credits and copyrights belong to the original authors. Materials presented in the note are based on the original papers and related books. References are being added.

This note is about machine learning methods.

The physicist Wolfgang Pauli apologized in a letter for “no time to be brief”. This also applies to this note, which should be further compactified.

## 1 Ptolemy’s Epicycle and Gauss Paradigm

We begin with an ancient example of learning to highlight the methods and issues in machine learning.

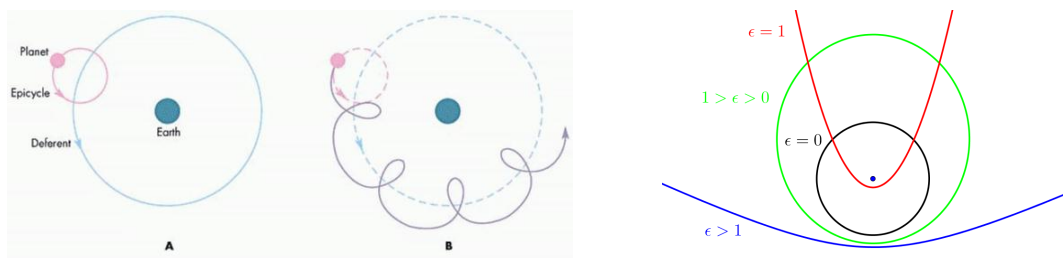


Figure 1: Left: epicycle model, with Earth at the center. Right: Newtonian model, with Sun at the center.

### 1.1 The model

Let  $(x(t), y(t))$  be the position of a planet at time  $t$ , with Earth being the origin  $(0, 0)$ . Suppose we observe  $(x_i, y_i)$  at times  $t_i$ ,  $i = 1, \dots, n$ . We may consider  $(t_i)$  as the input, and  $(x_i, y_i)$  as the output. We want to learn a model that can predict the position of the planet at a future time. Of course we also hope to understand the physics of planetary motion.

## A single circle

The simplest model is a circular trajectory,

$$x(t) = r \cos(\omega t), y(t) = r \sin(\omega t),$$

where  $r$  is the radius and  $\omega$  is the angular speed.

We can learn the model by minimizing the following least squares loss:

$$L(r, \omega) = \sum_{i=1}^n [(x_i - r \cos(\omega t_i))^2 + (y_i - r \sin(\omega t_i))^2].$$

We may use gradient descent to find a local minimum of  $(r, \omega)$ . If  $\omega$  is given, the estimation of  $r$  becomes a least squares linear regression, where  $r$  is the regression coefficient ( $\beta$  in modern notation) and  $(\cos(\omega t), \sin(\omega t))$  are predictors ( $x$  in modern notation). We may search  $\omega$  over a regular grid of points and, for each  $\omega$ , estimate  $r$  by least squares. We then choose  $\omega$  that gives us the minimal value of  $L$ .

## Complex number

We can also write the model using a complex number. Let  $z(t) = x(t) + iy(t)$  and  $z_i = x_i + iy_i$ . Then the model becomes

$$z(t) = r e^{i\omega t},$$

and

$$L(r, \omega) = \sum_{i=1}^n |z_i - r e^{i\omega t_i}|^2.$$

## Circle on top of circle

Ptolemy found out that a single circle did not fit the data well enough. He then assumed that the planet is moving around a center in a circular motion, and this center itself is moving around the Earth in a circular motion. This is an epicycle model that can be written as

$$z(t) = r_1 e^{i\omega_1 t} + r_2 e^{i\omega_2 t},$$

where  $r_1 e^{i\omega_1 t}$  is the original circle, and  $r_2 e^{i\omega_2 t}$  is the circle on top of the original circle. If two circles are not enough, we can add the third circle. In general, we may consider a model

$$z(t) = \sum_{k=1}^d r_k e^{i\omega_k t},$$

where  $d$  (for dimensionality or degrees of freedom) is the number of circles, and defines the complexity of the model.

This is a stroke of genius. It is a precursor to Fourier analysis, which shows that with enough circles, we can fit any trajectory.

The above model is by all means a good model, as good as any model we can find in machine learning literature. It is flexible enough to fit all the curves. It has a clear geometric meaning.

## 1.2 Boosting machine

If we add one circle at a time as was done by Ptolemy, we are actually doing boosting, which is an important learning method based on function expansion. We may call  $e^{i\omega_k t}$  a basis function or a base learner.

We may proceed as follows. Suppose the current model is  $z_{m-1}(t) = \sum_{k=1}^{m-1} r_k e^{i\omega_k t}$ . We want to add a new basis function to make it

$$z_m(t) = z_{m-1}(t) + r_m e^{i\omega_m t}.$$

We can find  $(r_m, \omega_m)$  by minimizing

$$\begin{aligned} L(r_m, \omega_m) &= \sum_{i=1}^n |z_i - z_{m-1}(t_i) - r_m e^{i\omega_m t_i}|^2 \\ &= \sum_{i=1}^n |\tilde{z}_i - r_m e^{i\omega_m t_i}|^2, \end{aligned}$$

where  $\tilde{z}_i = z_i - z_{m-1}(t_i)$ . Then learning  $(r_m, \omega_m)$  is the same as learning a single circle. This method is also called matching pursuit or forward selection.

## 1.3 Lasso, ridge and kernel machine

We may also fix a dense set of  $\{\omega_k, k = 1, \dots, d\}$ , such as in the discrete Fourier analysis. Then this set of basis functions can fit any curve with zero error. This will overfit the data and the learned model will absorb all the noise in the data. As a result, it won't perform well in prediction. We may consider regularization

$$L(r, \omega) = \sum_{i=1}^n \left| z_i - \sum_{k=1}^d r_k e^{i\omega_k t_i} \right|^2 + \lambda \sum_{k=1}^d \rho(r_k),$$

where  $r = (r_k, k = 1, \dots, d)$ , with  $\omega = (\omega_k, k = 1, \dots, d)$  assumed given.  $\rho(r_k)$  is the regularization function.

For instance, we may let  $\rho(r_k) = |r_k|$  which gives us the Lasso, which tends to select a relatively small number of  $(r_k)$  to be non-zeros, and set the other  $(r_k)$  to be zeros. This is related to the boosting machine, except that boosting machine is based on greedy forward selection, whereas the Lasso is based on  $\ell_1$  regularization which is more principled.

Or we may use  $\rho(r_k) = r_k^2$ , which gives us the ridge regression that tends to shrink  $(r_k)$  toward 0. We may also use  $\rho(r_k) = \omega_k r_k^2$ , so that we penalize high frequency components more to force the trajectory to be smooth.

The kernel machine is related to the ridge regression that penalizes  $r_k^2$ . We may let  $h(t) = (e^{i\omega_k t}, k = 1, \dots, d)^\top$ . Then  $z(t) = h(t)^\top \beta$ , where  $\beta = (r_k, k = 1, \dots, d)^\top$ . For computations, we only need to know  $K(t, t') = \langle h(t), h(t') \rangle$ , without dealing with  $h(t)$  and  $\beta$  directly.

## 1.4 Neural network

The circle on top of circle scheme is similar to the perceptron on top of perceptron scheme of neural networks.  $e^{i\omega_k t}$  is called a feature or hidden variable. In neural networks, the features or variables are also learned in addition to their coefficients. This corresponds to learning  $\omega = (\omega_k)$  instead of designing them.

## 1.5 Model complexity and regularization

We know that Ptolemy's model was replaced by Newton's model, because the latter has far fewer parameters and can be learned from a much smaller number of observations. It can also extrapolate to a vast range.

In science, we prefer simple models. As von Neumann said, “Give me three parameters I can fit an elephant, give me four, I can wiggle its trunk.” In fact, “adding an epicycle” is synonymous to bad science.

However, in machine learning, we usually use models that are similar to Ptolemy’s because of the lack of domain knowledge. Such models tend to have a lot of parameters and sometimes are even “over-parametrized”, meaning that the number of parameters exceeds the number of training examples. Such models are either explicitly or implicitly regularized to avoid overfitting. In Lasso and ridge regression, we use explicit regularization. In boosting, we use implicit regularization by early stopping. For neural networks, the gradient descent tends to get into a local mode near the random initialization, which is also a form of implicit regularization.

## 1.6 Ptolemy or Newton?

Ptolemy’s model is actually better than Newton’s model in general, because the former is more universal, and the latter is more specialized to our universe. If our universe is more complex, we may have to use Ptolemy’s model or even more complex models such as kernel machines or neural networks. Which model is better depends on what data the model is intended to explain. For instance, for the stock market, we do not expect a model as simple as Newton’s can fit well. On the other hand, the current kernel machines or neural networks may be too complex for explaining the experiences of a biological intelligent system, which may actually use models similar to Ptolemy’s.

## 1.7 Euler’s linear model

In the above discussion, we used the modern notation and ideas in statistics and machine learning. The least squares, Lasso, ridge regression, etc. were invented nearly two thousand years after Ptolemy’s genius idea.

But indeed we can treat the origin of statistics and machine learning to astronomy. The model is Newtonian mechanics (even though modern machine learning models are more of the style of Ptolemy). The data are those obtained by the observatories. Mathematicians such as Gauss worked for such observatories.

Euler studied the orbit of Jupiter around the Sun. The motion of Jupiter is influenced by the Saturn. By Taylor expansion or perturbation analysis, Euler obtained the following equation:

$$\varphi = \eta - 23525'' \sin q + 168'' \sin 2q^3 2'' \sin 2w - 257'' \sin(w - q) - 243'' \sin(2w - p) + m'' - x'' \sin q + y'' \sin 2q - z'' \sin(w - p) - u(\alpha + 360v + p) \cos(w - p) + Nu'' - 11405k'' \cos q + (1/600)k'' \cos 2q,$$

where  $(\varphi, \eta, q, w, p, N, v)$  are observed and vary from observation to observation, and  $(x, y, m, z, \alpha, k, n, u)$  are unknown parameters. Euler had 75 observations, and by treating  $u\alpha$  as  $\gamma$  he had 7 unknown parameters. In other words, he had 75 equations and 7 unknowns.

Let us translate the above equation into more familiar notation in modern statistics. Let

$$y_i = \varphi_i, \quad \beta = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{pmatrix} = \begin{pmatrix} x \\ y \\ \vdots \\ u \end{pmatrix}, \quad \text{and} \quad x_i = \begin{pmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{pmatrix} = \begin{pmatrix} \sin q_i \\ \sin 2q_i \\ \vdots \\ \sin(w - p) \end{pmatrix}.$$

Then the equation can be written as

$$y_i = x_i^\top \beta = x_{i1}\beta_1 + x_{i2}\beta_2 + \cdots + x_{ip}\beta_p \quad \text{for } i = 1, \dots, n = 75.$$

The following are some observations: (1)  $y_i$  is linear in  $\beta$ , but it can be non-linear in the original variables  $(q, w, p, N, u)$ . (2) The model is known to be correct *a priori*. While (1) is common in linear models, (2) is rare in machine learning.

Euler did not go very far in solving the above problem.

## 1.8 Laplace's estimating equation

Laplace proposed the method of combination of equations, e.g., we combine the 75 equations into 7 equations, so that we can solve for the 7 unknowns. Specifically we solve  $\beta$  from the following estimating equations:

$$\sum_{i=1}^n w_{ik} y_i = \sum_{i=1}^n w_{ik} \sum_{j=1}^p x_{ij} \beta_j, \quad k = 1, \dots, p,$$

where  $(w_{ij})$  is a set of pre-designed weights. Laplace designed a special set of weights. But he did not give a general principle on how to design the weights.

## 1.9 Gauss paradigm

Gauss (and Legendre) proposed to estimate  $\beta$  by least squares, i.e., minimizing the loss function

$$\mathcal{L}(\beta) = \sum_{i=1}^n \left( y_i - \sum_{j=1}^p x_{ij} \beta_j \right)^2.$$

The above loss function can be minimized in closed form by solving the linear equation  $L'(\beta) = 0$ . This leads to the following estimating equation:

$$\frac{\partial}{\partial \beta_k} \mathcal{L}(\beta) = -2 \sum_{i=1}^n x_{ik} \left( y_i - \sum_{j=1}^p x_{ij} \beta_j \right) = 0, \quad k = 1, \dots, p.$$

This estimating equation corresponds to Laplace's estimating equation with  $w_{ik} = x_{ik}$ .

Gauss did three things that set the paradigm for statistics and machine learning.

- (1) Gauss started with a loss function. In machine learning, most of the methods start from loss functions.
- (2) Gauss motivated the loss function by a probabilistic formulation. He assumed that

$$y_i = \sum_{j=1}^p x_{ij} \beta_j + \varepsilon_i, \quad \varepsilon_i \sim N(0, \sigma^2),$$

independently for  $i = 1, \dots, n$ . Assuming a prior distribution  $p(\beta)$ , the posterior distribution is

$$\begin{aligned} p(\beta | (x_i, y_i), i = 1, \dots, n) &\propto p(\beta) \prod_{i=1}^n p(y_i | x_i, \beta) \\ &\propto \exp \left[ -\frac{1}{2\sigma^2} \sum_{i=1}^n \left( y_i - \sum_{j=1}^p x_{ij} \beta_j \right)^2 + \log p(\beta) \right]. \end{aligned}$$

Assuming a uniform  $p(\beta)$ , then maximizing  $p(\beta | (x_i, y_i), i = 1, \dots, n)$  is equivalent to minimizing the loss function  $\mathcal{L}(\beta)$ .  $\prod_{i=1}^n p(y_i | x_i, \beta)$  is called likelihood. The least squares estimate is also the maximum likelihood estimate.

(3) Gauss analyzed the property of the least squares estimator  $\hat{\beta}_{LS}$ , which is a function of  $(x_i, y_i), i = 1, \dots, n$ . He used a Frequentist thinking, even though the loss function is motivated by Bayesian thinking. Specifically, we assume  $(x_i, y_i) \sim p(x, y)$  independently, where  $p(x, y)$  is the joint distribution so that the conditional distribution  $p(y|x)$  is such that  $y_i = N(x_i^\top \beta_{\text{true}}, \sigma^2)$ , where  $\beta_{\text{true}}$  is the true value of  $\beta$ . If we believe Newtonian mechanics, then such a  $\beta_{\text{true}}$  does exist. In Frequentist thinking, we assume  $\beta_{\text{true}}$  is fixed but unknown (whereas in Bayesian thinking, we treat  $\beta$  as a random variable). The Frequentist thinking is



based on hypothetic repeated sampling, i.e., suppose 100 astronomers collected their own data  $(x_i, y_i), i = 1, \dots, n$  independently, and each calculates his or her  $\hat{\beta}_{LS}$  based on least squares, then we have 100 estimated  $\hat{\beta}_{LS}$ . We can show that  $\hat{\beta}_{LS}$  fluctuates around  $\beta_{true}$ , i.e.,  $\mathbb{E}(\hat{\beta}_{LS}) = \beta_{true}$ . That is,  $\hat{\beta}_{LS}$  is unbiased. The magnitude of fluctuation, i.e.,  $\text{Var}(\hat{\beta}_{LS})$  can also be calculated.

As to Laplace estimator based on estimating equation, we can also show that it is unbiased. We can also calculate its variance. Gauss proved that the variance of Laplace estimator is minimized at  $w_{ik} = x_{ik}$ , i.e., the least squares estimator achieves the minimal variance among all possible Laplace estimators. This result is called the Gauss-Markov theorem, which claims that the least squares estimator is the best linear unbiased estimator (BLUE).

We may summarize Gauss paradigm as follows.

- (1) A probabilistic model of the data.
- (2) Loss function based on posterior or likelihood.
- (3) Analysis within the probabilistic framework.

It is fair to say that modern statistics and machine learning did not go much beyond Gauss paradigm.

## 1.10 Continuing Gauss paradigm

Gauss assumed that the prior  $p(\beta)$  is a uniform distribution (within a big finite range). We can also assume  $p(\beta) \sim N(0, \tau^2 I_p)$ , where  $I_p$  is the  $p$ -dimensional identity matrix. This leads to the ridge regression, which maximizes the posterior  $p(\beta | (x_i, y_i), i = 1, \dots, n)$ , or equivalently, minimizes the penalized least squares

$$\sum_{i=1}^n \left( y_i - \sum_{j=1}^p x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p \beta_j^2.$$

The penalty term leads to regularization of estimation, which is important for generalization. It is particularly important if we have a lot of variables, such as  $p > n$ . The ridge regression leads to kernel machine.

We may also perform a full Bayesian analysis, i.e., making multiple guesses of  $\beta$  by sampling from the posterior  $p(\beta | (x_i, y_i), i = 1, \dots, n)$ , and when we make prediction, use each of the multiple guesses to predict, and average over the multiple predictions. This leads to the Gaussian process treatment.

For some problems, it may be appropriate to assume that  $\beta$  is sparse, i.e., only a small number of components of  $\beta$  are non-zero. We can use a prior distribution such as a mixture of a point mass at 0 and a normal distribution with a big variance. We can also use the sparsity inducing  $\ell_1$  regularization to minimize

$$\sum_{i=1}^n \left( y_i - \sum_{j=1}^p x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p |\beta_j|.$$

This is the Lasso (Least Absolute Shrinkage and Selection Operator) estimator.

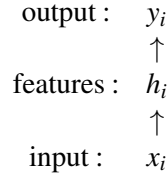
## 1.11 Three modes of learning

### Supervised learning

The table below displays the dataset for supervised learning.  $h_i = (h_{ik}, k = 1, \dots, d)^\top$  is the  $d$ -dimensional vector of features or hidden variables.  $y_i$  follows a linear model on  $h_i$ . Euler's model can be written in this form, where  $x_i$  are the raw input variables  $(\varphi, \eta, q, w, p, N, v)$ , and  $h(x_i)$  are derived from perturbation analysis based on Newtonian mechanism, such as  $(\sin q_i, \sin 2q_i, \dots, \sin(w - p))$ .

	input	features	output
1	$x_1^\top$	$h_1^\top$	$y_1$
2	$x_2^\top$	$h_2^\top$	$y_2$
...			
$n$	$x_n^\top$	$h_n^\top$	$y_n$

The supervised learning can be represented by the diagram below,



where the vector of features  $h_i$  is computed from  $x_i$  via  $h_i = h(x_i)$ .

*Encoder and decoder:* In the above diagram, the transformation  $x_i \rightarrow h_i$  is called an encoder, and the transformation  $h_i \rightarrow y_i$  is called a decoder.

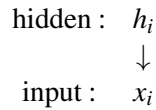
Both classification and regression are about supervised learning because for each input  $x_i$ , an output  $y_i$  is provided as supervision. In regression,  $y_i$  is continuous. In classification,  $y_i$  is categorical. We can represent  $y_i$  by a one-hot vector, i.e., if  $y_i$  denotes the  $k$ -th category, then  $y_i$  is a vector where the  $k$ -th element is 1 and all the other elements are 0.

## Unsupervised learning

In unsupervised learning, the dataset is as below, where  $y_i$  are not provided as supervision.

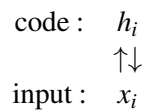
	input	hidden	output
1	$x_1^\top$	$h_1^\top$	?
2	$x_2^\top$	$h_2^\top$	?
...			
$n$	$x_n^\top$	$h_n^\top$	?

In a generative model, the vector  $h_i$  is not a vector of features extracted from the signal  $x_i$ .  $h_i$  is a vector of hidden variables that is used to generate  $x_i$ , as illustrated by the following diagram:



The components of the  $d$ -dimensional  $h_i$  are variably called factors, sources, components or causes. The prototype example is factor analysis or principal component analysis.

*Auto-encoder:*  $h_i$  is also called a code in the auto-encoder illustrated by the following diagram:



The direction from  $h_i$  to  $x_i$  is called the decoder, and the direction from  $x_i$  to  $h_i$  is called the encoder.

*Distributed representation and disentanglement:*  $h_i = (h_{ik}, k = 1, \dots, d)$  is called a distributed representation of  $x_i$ . Usually the components of  $h_i$ ,  $(h_{ik}, k = 1, \dots, d)$ , are assumed to be independent, and  $(h_{ik})$  are said to disentangle the variations in  $x_i$ .

*Embedding:*  $h_i$  can also be considered the coordinates of  $x_i$ , if we embed  $x_i$  in a low-dimensional space, as illustrated by the following diagram:

$$\begin{array}{c} \leftarrow h_i \rightarrow \\ | \\ \leftarrow x_i \rightarrow \end{array}$$

In the training data, we find a  $h_i$  for each  $x_i$ , so that  $\{h_i, i = 1, \dots, n\}$  preserve the relative relations between  $\{x_i, i = 1, \dots, n\}$ . The prototype example of embedding is multi-dimensional scaling, where we want to preserve the Euclidean distances between the examples.

## Reinforcement learning

Reinforcement learning is similar to supervised learning except that the guidance is in the form of reward. Here  $x_i$  is the state.  $y_i$  can be the action taken at this state.  $y_i$  can also be the value of this state, where value is defined as the accumulated reward.

### 1.12 Bayesian, Frequentist, variational

We can write  $(x_i, y_i) \sim p_\theta(x_i, y_i)$  for  $i = 1, \dots, n$ . In supervised learning, we let  $p_\theta(x, y) = p_\theta(y|x)p(x)$ , where we learn  $p_\theta(y|x)$  and we leave  $p(x)$  alone. In unsupervised learning, we do not observe  $y$ , and we model  $p_\theta(x)$  instead.

In the following, we focus on supervised learning. In Bayesian framework, we treat  $\theta$  as a random variable. We assume its marginal distribution to be  $p(\theta)$ . It is called the prior distribution. The learning is based on posterior distribution

$$p(\theta \mid (x_i, y_i), i = 1, \dots, n) \propto p(\theta) \prod_{i=1}^n p(y_i \mid x_i, \theta),$$

where we write  $p_\theta(y|x)$  as  $p(y|x, \theta)$  to emphasize that  $\theta$  is a random variable to be conditioned upon.  $\prod_{i=1}^n p(y_i \mid x_i, \theta)$  is called likelihood.  $l(\theta \mid (x_i, y_i), i = 1, \dots, n) = \sum_{i=1}^n \log p(y_i \mid x_i, \theta)$  is called the log-likelihood.

If we estimate  $\theta$  by maximizing  $p(\theta \mid (x_i, y_i), i = 1, \dots, n)$  over  $\theta$ , we get the so-called Maximum A Posteriori (MAP) estimate.

$$\log p(\theta \mid (x_i, y_i), i = 1, \dots, n) = l(\theta \mid (x_i, y_i), i = 1, \dots, n) + \log p(\theta).$$

If  $p(\theta)$  is uniform within a range, MAP becomes maximum likelihood estimate (MLE). For non-uniform  $p(\theta)$ , MAP is penalized or regularized likelihood.

MAP only captures the maximum or mode of the posterior distribution but misses the uncertainty in the posterior distribution. To capture the uncertainty, we may draw multiple samples  $\theta_m \sim p(\theta \mid (x_i, y_i), i = 1, \dots, n)$  for  $m = 1, \dots, M$ . This can be accomplished by Monte Carlo, such as Markov chain Monte Carlo (MCMC). These  $\theta_m$  are the multiple guesses of  $\theta$ .

The posterior  $p(\theta \mid (x_i, y_i), i = 1, \dots, n)$  is often not tractable in the sense that we cannot calculate the normalizing constant of  $p(\theta) \prod_{i=1}^n p(y_i \mid x_i, \theta)$  to make it a probability distribution. In variational inference, we find a simpler distribution  $q_\phi(\theta)$  to approximate  $p(\theta \mid (x_i, y_i), i = 1, \dots, n)$ , where  $\phi$  is the variational parameter that we choose to minimize the divergence from  $q_\phi(\theta)$  to  $p(\theta \mid (x_i, y_i), i = 1, \dots, n)$ .

For a testing example  $(x_0, y_0)$ , if we know  $\theta$ , we can predict  $y_0$  based on  $p(y_0 | x_0, \theta)$ , where  $\theta$  can be MAP or MLE. But it is better to take into account the uncertainty in  $\theta$  by averaging over multiple guesses, i.e.,

$$\begin{aligned} p(y_0 | x_0, (x_i, y_i), i = 1, \dots, n) &= \int p(y_0 | x_0, \theta) p(\theta | (x_i, y_i), i = 1, \dots, n) \\ &= \mathbb{E}_{p(\theta | (x_i, y_i), i = 1, \dots, n)} [p(y_0 | x_0, \theta)] \approx \frac{1}{M} \sum_{m=1}^M p(y_0 | x_0, \theta_m). \end{aligned}$$

This avoids overfitting, or as the slogan goes: Bayesian does not overfit. For some models such as Gaussian process,  $p(y_0 | x_0, (x_i, y_i), i = 1, \dots, n)$  can be obtained directly without explicitly integrating out  $\theta$ .

The Frequentist thinking treats  $(x_i, y_i) \sim p_{\text{data}}(x, y)$ , where  $p_{\text{data}}$  is an unknown distribution that generates the data. If the model is correct, then there is a true value  $\theta_{\text{true}}$ , so that  $p_{\text{data}}(x, y) = p_{\text{true}}(x, y)$ . We can then study the property of MLE, MAP, or Bayesian prediction within this frequentist framework.

In Gauss paradigm, he used Bayesian to motivate the least squares method, and he justified the least squares method within the Frequentist framework.

## 2 Basics: Linear Models

The linear models are the foundation of most of the machine learning methods. The top layer of support vector machine, adaboost, and neural networks are all linear models. They differ only in terms of how the top layer variables or features are designed or learned.

### 2.1 Linear regression

The following tables illustrate the data frame and the notation of linear regression.

	input	output
1	$x_{11}, x_{12}, \dots, x_{1p}$	$y_1$
2	$x_{21}, x_{22}, \dots, x_{2p}$	$y_2$
...		
$n$	$x_{n1}, x_{n2}, \dots, x_{np}$	$y_n$

For the first table, we can write the model as  $y_i = \sum_{j=1}^p \beta_j x_{ij} + \varepsilon_i$ , and  $\varepsilon_i \sim N(0, \sigma^2)$  independently.

	input	output
1	$x_1^\top$	$y_1$
2	$x_2^\top$	$y_2$
...		
$n$	$x_n^\top$	$y_n$

For the second table,  $x_i = (x_{i1}, \dots, x_{ip})^\top$ , and we can write the model as  $y_i = x_i^\top \beta + \varepsilon_i$ , where  $\beta = (\beta_1, \dots, \beta_p)^\top$ .

input	output
$X = (X_1, X_2, \dots, X_p)$	$Y$

For the third table,  $X_j = (x_{1j}, \dots, x_{nj})^\top$ ,  $Y = (y_1, \dots, y_n)^\top$ , and we can write the model as  $Y = \sum_{j=1}^p X_j \beta_j + \varepsilon$ , or  $Y = X\beta + \varepsilon$ , where  $\varepsilon = (\varepsilon_1, \dots, \varepsilon_n)^\top$ , and  $\varepsilon \sim N(0, \sigma^2 I_n)$ .

### Background: derivatives in matrix form

Suppose  $Y = (y_i)_{m \times 1}$ , and  $X = (x_j)_{n \times 1}$ . Suppose  $Y = h(X)$ . We can define

$$\frac{\partial Y}{\partial X^\top} = \left( \frac{\partial y_i}{\partial x_j} \right)_{m \times n}.$$

To understand the notation, we can treat  $\partial Y = (\partial y_i, i = 1, \dots, m)^\top$  as a column vector, and  $1/\partial X = (1/\partial x_j, j = 1, \dots, n)^\top$  as another column vector. Now we have two vectors of operations, instead of numbers. The product of the elements of the two vectors is understood as composition of the two operators, i.e.,  $\partial y_i (1/\partial x_j) = \partial y_i / \partial x_j$ . Then  $\partial Y / \partial X^\top$  is a squared matrix according to the matrix multiplication rule.

If  $Y = AX$ , then  $y_i = \sum_k a_{ik} x_k$ . Thus  $\partial y_i / \partial x_j = a_{ij}$ . So  $\partial Y / \partial X^\top = A$ .

If  $Y = X^\top S X$ , where  $S$  is symmetric, then  $\partial Y / \partial X = 2SX$ .

If  $S = I$ ,  $Y = |X|^2$ ,  $\partial Y / \partial X = 2X$ .

The chain rule in matrix form is as follows. If  $Y = h(X)$  and  $X = g(Z)$ , then

$$\frac{\partial y_i}{\partial z_j} = \sum_k \frac{\partial y_i}{\partial x_k} \frac{\partial x_k}{\partial z_j}.$$

Thus

$$\frac{\partial Y}{\partial Z^\top} = \frac{\partial Y}{\partial X^\top} \frac{\partial X}{\partial Z^\top}.$$

### Least squares estimator

For general  $(X, Y)$ ,

$$\mathcal{L}(\beta) = |Y - X\beta|^2.$$

Let  $e = Y - X\beta$ , then  $\mathcal{L}(\beta) = |e|^2$ . Applying the chain rule,

$$\frac{\partial \mathcal{L}}{\partial \beta^\top} = \frac{\partial \mathcal{L}}{\partial e^\top} \frac{\partial e}{\partial \beta^\top} = -2e^\top X,$$

hence

$$\mathcal{L}'(\beta) = \frac{\partial \mathcal{L}}{\partial \beta} = -2X^\top (Y - X\beta).$$

Setting  $\mathcal{L}'(\beta) = 0$ , we get the least squares estimator

$$\hat{\beta} = (X^\top X)^{-1} X^\top Y.$$

Geometrically,  $\hat{Y} = X\hat{\beta}$  is the projection of  $Y$  onto the subspace spanned by  $X = (X_1, \dots, X_p)$ , so that  $e = Y - X\hat{\beta}$  is perpendicular to  $X_j$  at  $\hat{\beta}$ , i.e.,  $\langle e, X_j \rangle = X_j^\top e = 0$  for  $j = 1, \dots, p$ , i.e.,  $X^\top (Y - X\hat{\beta}) = 0$ , which leads to the least squares  $\hat{\beta}$ . The projection is

$$\hat{Y} = X\hat{\beta} = X(X^\top X)^{-1} X^\top Y = HY,$$

where the hat matrix  $H = X(X^\top X)^{-1} X^\top$  encodes the projection operation.

## 2.2 Ridge regression and shrinkage estimator

In order to reduce the model bias, we want the number of parameters to be large. However, this will cause overfitting if we continue to use the least squares estimator. We may reduce overfitting by using a biased estimator such as ridge regression.

The ridge regression minimizes

$$\mathcal{L}(\beta) = |Y - X\beta|^2 + \lambda |\beta|^2,$$

for  $\lambda \geq 0$ . Setting

$$\mathcal{L}'(\beta) = -2X^\top(Y - X\beta) + 2\lambda\beta = 0,$$

we have

$$\hat{\beta} = (X^\top X + \lambda I_p)^{-1} X^\top Y.$$

The ridge regression can also be expressed as minimizing the least squares loss  $|Y - X\beta|^2$  subject to  $|\beta|^2 \leq t$  for a certain  $t > 0$ .

More generally, we can minimize

$$\mathcal{L}(\beta) = |Y - X\beta|^2 + \lambda \beta^\top D \beta,$$

for a symmetric matrix  $D$ , and

$$\hat{\beta} = (X^\top X + D)^{-1} X^\top Y.$$

For ridge regression, in the case  $X^\top X = I_p$ , we have

$$\hat{\beta} = \hat{\beta}_{LS} / (1 + \lambda),$$

which is a shrinkage estimator.

## 2.3 Linear spline

In the one-dimensional case, the linear spline model is of the form

$$f(x) = \beta_0 + \sum_{k=1}^d \beta_k (x - \alpha_k)_+,$$

where  $(x - \alpha_k)_+ = \max(0, x - \alpha_k)$ ,  $\alpha_k, k = 1, \dots, d$  are the knots, and  $\beta_k$  is the change of slope at knot  $\alpha_k$ . We can learn the model from the training data  $(x_i, y_i)$ ,  $i = 1, \dots, n$  by ridge regression which minimizes

$$\mathcal{L}(\beta) = \sum_{i=1}^n \left[ y_i - \beta_0 - \sum_{k=1}^d \beta_k (x_i - \alpha_k)_+ \right]^2 + \lambda \sum_{k=1}^d \beta_k^2,$$

where  $\sum_{k=1}^d \beta_k^2$  measures the smoothness of  $f(x)$ .

Let  $x_{ik} = (x_i - \alpha_k)_+$ ,  $x_{i0} = 1$ . The objective function is

$$\mathcal{L}(\beta) = |Y - X\beta|^2 + \beta^\top D \beta,$$

where  $X$  is the  $n \times (p+1)$  matrix,  $D$  is  $(p+1) \times (p+1)$  diagonal matrix, with  $D_{kk} = \lambda$ , except  $D_{11} = 0$  because we do not penalize  $\beta_0$ . Then

$$\hat{\beta} = (X^\top X + D)^{-1} X^\top Y.$$

We may choose  $(\alpha_k)$  to be equally spaced within the range of  $x$ , usually taken to be  $[0, 1]$ . If we use a large number of knots, and if  $\lambda$  is small, then the spline tends to interpolate the observations and overfit the noise. On the other hand, if  $\lambda$  is too big, the model may be close to a horizontal line and may fail to capture the signal.

Modern neural networks can be considered a multivariate generalization of the linear spline. Let  $x$  be a  $p$ -dimensional input. A two layer rectified neural network is of the following form

$$f(x) = \beta_0 + \sum_{k=1}^d \beta_k h_k,$$

$$h_k = \max \left( 0, \alpha_{k0} + \sum_{j=1}^p \alpha_{kj} x_j \right).$$

The model can be considered a multivariate linear spline. We can learn both  $\alpha = (\alpha_{kj}, j = 0, 1, \dots, p, k = 1, \dots, d)$  and  $\beta = (\beta_k, k = 0, 1, \dots, d)$  from training data  $\{(x_i, y_i), i = 1, \dots, n\}$ . We will study this model later.

## 2.4 Lasso regression

The Lasso regression estimates  $\beta$  by

$$\hat{\beta}_\lambda = \arg \min_{\beta} \left[ \frac{1}{2} \|Y - X\beta\|_{\ell_2}^2 + \lambda \|\beta\|_{\ell_1} \right],$$

where  $\|\beta\|_{\ell_1} = \sum_{j=1}^p |\beta_j|$ . Lasso stands for “least absolute shrinkage and selection operator.” There is no closed form solution for general  $p$ .

We do have closed form solution for  $p = 1$ , where  $X$  is an  $n \times 1$  vector,

$$\hat{\beta}_\lambda = \begin{cases} (\langle Y, X \rangle - \lambda) / \|X\|_{\ell_2}^2, & \text{if } \langle Y, X \rangle > \lambda; \\ (\langle Y, X \rangle + \lambda) / \|X\|_{\ell_2}^2, & \text{if } \langle Y, X \rangle < -\lambda; \\ 0 & \text{if } |\langle Y, X \rangle| < \lambda. \end{cases}$$

We can write it as

$$\hat{\beta}_\lambda = \text{sign}(\hat{\beta}) \max(0, |\hat{\beta}| - \lambda / \|X\|_{\ell_2}^2),$$

where  $\hat{\beta} = \langle Y, X \rangle / \|X\|_{\ell_2}^2$  is the least squares estimator. The above transformation from  $\hat{\beta}$  to  $\hat{\beta}_\lambda$  is called soft thresholding.

Compare Lasso with ridge regression in one-dimensional situation, the latter being  $\hat{\beta}_\lambda = \langle Y, X \rangle / (\|X\|_{\ell_2}^2 + \lambda)$ , the behavior of Lasso is richer, including both shrinkage (by subtracting  $\lambda$ ) and selection (via thresholding at  $\lambda$ ).

The reason for the fact that  $\hat{\beta}$  can be zero is that the left and right derivatives of  $|\beta|$  at 0 are not the same, so that the function  $\|Y - X\beta\|_{\ell_2}^2 / 2 + \lambda \|\beta\|_{\ell_1}$  may have a negative left derivative and a positive right derivative at 0, so that 0 can be the minimum. For  $|\beta|^{1+\delta}$  with  $\delta > 0$ , its derivative at 0 is 0, so that  $\hat{\beta}$  cannot be zero in general. For  $|\beta|^{1-\delta}$ , there is a sharp turn at 0, but it is not convex anymore.  $|\beta|$  or piecewise linear function in general is the only choice that has a sharp turn at 0 but is still barely convex.

Thus the Lasso regression prefers sparse  $\beta$ , i.e., only a small number of components of  $\beta$  are non-zero.

### Primal form of Lasso

The primal form of Lasso is  $\min \|Y - X\beta\|_{\ell_2}^2 / 2$  subject to  $\|\beta\|_{\ell_1} \leq t$ . The dual form of Lasso is  $\min \|Y - X\beta\|_{\ell_2}^2 / 2 + \lambda \|\beta\|_{\ell_1}$ . The two forms are equivalent with a one-to-one correspondence between  $t$  and  $\lambda$ . If  $\hat{\beta}_\lambda$

is the solution to the dual form, then it must be the solution to the primal form with  $t = \|\hat{\beta}_\lambda\|_{\ell_1}$ . The reason is that if a different  $\hat{\beta}$  is the solution to the primal form, then  $\hat{\beta}$  is a better solution to the dual form than  $\hat{\beta}_\lambda$ , which results in contradiction.

The primal form also reveals the sparsity inducing property of  $\ell_1$  regularization in that the  $\ell_1$  ball has low-dimensional corners, edges, and faces, but is still barely convex.

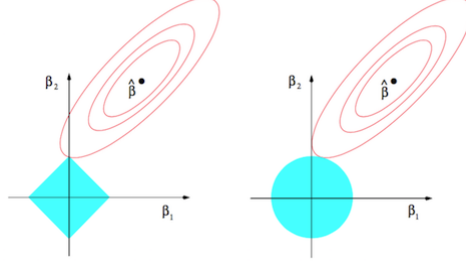


Figure 2: Lasso in primal form.

The above is the well known figure of Lasso. Take the left plot for example. The blue region is  $\|\beta\|_{\ell_1} \leq t$ . The red curves is the contour plot, where each red elliptical circle consists of those  $\beta$  that have the same value of  $\|Y - X\beta\|_{\ell_2}^2$ . The circle on the outside has bigger  $\|Y - X\beta\|_{\ell_2}^2$  than the circle inside. The solution to the problem of  $\min \|Y - X\beta\|_{\ell_2}^2$  subject to  $\|\beta\|_{\ell_1} \leq t$  is where the red circle touches the blue region. Any other points in the blue region will be outside the outer red circle and thus have bigger values of  $\|Y - X\beta\|_{\ell_2}^2$ . The reason that the  $\ell_1$  regularization induces sparsity is that it is likely for the red circle to touch the blue region at a corner, which is a sparse solution. If we use  $\ell_2$  regularization, as is the case with the plot on the right, then the solution is not sparse in general.

### Coordinate descent for Lasso solution path

For multi-dimensional  $X = (X_j, j = 1, \dots, p)$ , we can use the coordinate descent algorithm to compute  $\hat{\beta}_\lambda$ . The algorithm updates one component at a time, i.e., given the current values of  $\beta = (\beta_j, j = 1, \dots, p)$ , let  $R_j = Y - \sum_{k \neq j} X_k \beta_k$ , we can update  $\beta_j = \text{sign}(\hat{\beta}_j) \max(0, |\hat{\beta}_j| - \lambda / \|X_j\|_{\ell_2}^2)$ , where  $\hat{\beta}_j = \langle R_j, X_j \rangle / \|X_j\|_{\ell_2}^2$ .

We can find the solution path of Lasso by starting from a big  $\lambda$  so that all of the estimated  $\beta_j$  are zeros. Then we gradually reduce  $\lambda$ . For each  $\lambda$ , we cycle through  $j = 1, \dots, p$  for coordinate descent until convergence, and then we lower  $\lambda$ . This gives us  $\hat{\beta}(\lambda)$  for the whole range of  $\lambda$ . The whole process is a forward selection process, which sequentially selects new variables and occasionally removes selected variables.

### Least angle regression

In the above algorithm, at any given  $\lambda$ , let  $R = Y - \sum_{j=1}^p X_j \beta_j$ , then  $\hat{\beta}_j = \beta_j + \langle R, X_j \rangle / \|X_j\|_{\ell_2}^2$ . If  $\beta$  is the Lasso solution, then

$$\langle R, X_j \rangle = \begin{cases} \lambda, & \text{if } \beta_j > 0, \\ -\lambda, & \text{if } \beta_j < 0, \\ s\lambda & \text{if } \beta_j = 0. \end{cases}$$

where  $|s| < 1$ . Thus in the above process, for all of those selected  $X_j$ , the algorithm maintains that  $\langle R, X_j \rangle$  to be  $\lambda$  or  $-\lambda$ , for all selected  $X_j$ . If we interpret  $|\langle R, X_j \rangle|$  in terms of the angle between  $R$  and  $X_j$ , then we may call the above process the equal angle regression or the least angle regression (LARS). In fact, the solution



path is piecewise linear, and the LARS computes the linear pieces analytically instead of gradually reducing  $\lambda$  as in coordinate descent.

### Stagewise regression or epsilon-boosting

The stagewise regression iterates the following steps. Given the current  $R = Y - \sum_{j=1}^p X_j \beta_j$ , find  $j$  with the maximal  $|\langle R, X_j \rangle|$ . Then update  $\beta_j \leftarrow \beta_j + \varepsilon \langle R, X_j \rangle$  for a small  $\varepsilon$ . This is similar to the matching pursuit but is much less greedy. Such an update will change  $R$  and reduce  $|\langle R, X_j \rangle|$ , until another  $X_j$  catches up. So overall, the algorithm ensures that all of the selected  $X_j$  to have the same  $|\langle R, X_j \rangle|$ , which is the case with the algorithm in the above two sections. The stagewise regression is also called  $\varepsilon$ -boosting.

We can also view the stagewise regression from the perspective of the primal form of the Lasso problem: minimize  $\|Y - X\beta\|_{\ell_2}^2$  subject to  $\|\beta\|_{\ell_1} \leq t$ . If we relax the constraint by increasing  $t$  to  $t + \Delta t$ , then we want to update  $\beta_j$  with the maximal  $|\langle R, X_j \rangle|$  in order to maximally reducing  $\|Y - X\beta\|_{\ell_2}^2$ .

## 2.5 Logistic regression

Consider a dataset with  $n$  training examples, where  $x_i^\top = (x_{i1}, \dots, x_{ip})$  consists of  $p$  predictors and  $y_i \in \{0, 1\}$  is the outcome or class label.

We assume  $[y_i | x_i, \beta] \sim \text{Bernoulli}(p_i)$ , i.e.,  $\Pr(y_i = 1 | x_i, \beta) = p_i$ , and we assume

$$\text{logit}(p_i) = \log \frac{p_i}{1 - p_i} = s_i = x_i^\top \beta.$$

Then

$$p_i = \text{sigmoid}(s_i) = \frac{e^{s_i}}{1 + e^{s_i}} = \frac{1}{1 + e^{-s_i}},$$

where the sigmoid function is the inverse of the logit function.

## 2.6 Classification and perceptron

For logistic regression, we want to learn  $\beta$  either for the purpose of explanation or understanding, or for the purpose of classification or prediction. In the context of classification, we usually let  $y_i \in \{+1, -1\}$  instead of  $y_i \in \{1, 0\}$ . Those  $x_i$  with  $y_i = +1$  are called positive examples, and those  $x_i$  with  $y_i = -1$  are called negative examples.

We may call  $\beta$  a classifier.  $s_i = x_i^\top \beta = \langle x_i, \beta \rangle$  is the projection of  $x_i$  on the vector  $\beta$ , so the vector  $\beta$  is the direction that reveals the difference between positive  $x_i$  and negative  $x_i$ . Thus  $\beta$  should be aligned with positive  $x_i$  and negatively aligned with negative  $x_i$ , i.e.,  $\beta$  should point from the negative examples to the positive examples.

According to the previous subsection,

$$\Pr(y_i = +1 | x_i, \beta) = \frac{1}{1 + \exp(-s_i)}.$$

A deterministic version of the logistic regression is the perceptron model

$$y_i = \text{sign}(s_i),$$

where  $\text{sign}(s) = +1$  if  $s \geq 0$ , and  $\text{sign}(s) = -1$  if  $s < 0$ .

The perceptron model is inspired by neuroscience. See Figure 3. It can be considered an over-simplified model of a neuron, which takes input  $x_i$ , and emits output  $y_i$ . See Figure 4.

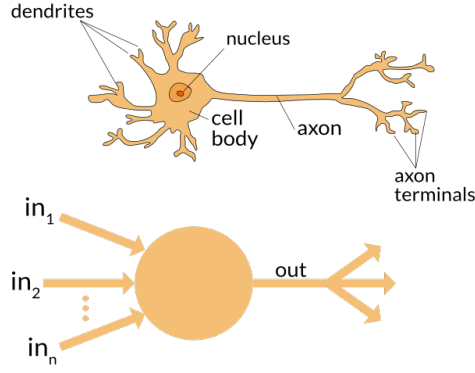


Figure 3: A neuron takes inputs, and generates outputs.

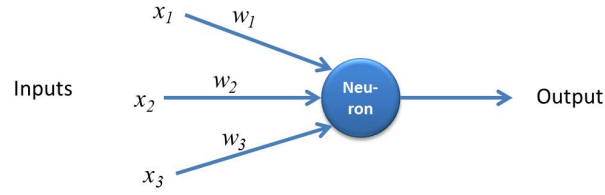


Figure 4: A perceptron is a simple model of a neuron. It computes a weighted sum of the inputs (plus a bias), and outputs the sign of the weighted sum.

The perceptron model can be generalized to neural networks, support vector machines, as well as adaboost, which are three major tools for classification.

Notationally, in machine learning literature, the perceptron is often written as

$$y_i = \text{sign} \left( \sum_{j=1}^p w_j x_{ij} + b \right) = \text{sign}(x_i^\top w + b),$$

where  $w = (w_j, j = 1, \dots, p)^\top$  are the connection weights and  $b$  is the bias term.  $(w, b)$  corresponds to  $\beta$ .

## 2.7 Loss functions

In order to learn  $\beta$  (or  $(w, b)$ ) from the training data  $\{(x_i, y_i), i = 1, \dots, n\}$ , we can minimize the loss function

$$\mathcal{L}(\beta) = \sum_{i=1}^n L(y_i, s_i),$$

where  $L(y_i, s_i)$  is the loss for each training example  $(x_i, y_i)$ . We need to define  $L(y_i, s_i)$ .

### Loss function for least squares regression

For linear regression, we usually use the least squares loss,

$$L(y_i, s_i) = (y_i - s_i)^2.$$

### Loss function for robust linear regression

We may also use the mean absolute value loss,

$$L(y_i, s_i) = |y_i - s_i|,$$

which penalizes large differences between  $y_i$  and  $s_i = x_i^\top \beta$  to a less degree than the least squares loss, thus the estimated  $\beta$  is less affected by the outliers.

### Loss function for logistic regression with 0/1 responses

For logistic regression, we usually maximize the likelihood function, which is

$$\text{Likelihood}(\beta) = \prod_{i=1}^n \Pr(y_i | x_i, \beta).$$

That is, we want to find  $\beta$  to maximize the probability of the observed  $(y_i, i = 1, \dots, n)$  given  $(x_i, i = 1, \dots, n)$ . The maximum likelihood estimate gives the most plausible explanation to the observed data.

For  $y_i \in \{0, 1\}$ ,

$$\Pr(y_i = 1 | x_i, \beta) = \text{sigmoid}(s_i) = \frac{\exp(s_i)}{1 + \exp(s_i)},$$

$$\Pr(y_i = 0 | x_i, \beta) = 1 - \Pr(y_i = 1 | x_i, \beta) = \frac{1}{1 + \exp(s_i)},$$

We can combine the above two equations by

$$\Pr(y_i | x_i, \beta) = \frac{\exp(y_i s_i)}{1 + \exp(s_i)}.$$

The log-likelihood is

$$\text{LogLikelihood}(\beta) = \sum_{i=1}^n \log \Pr(y_i | x_i, \beta) = \sum_{i=1}^n [y_i s_i - \log(1 + \exp(s_i))].$$

We can define the loss function as the negative log-likelihood

$$L(y_i, s_i) = -[y_i s_i + \log(1 + \exp(s_i))].$$

### Loss function for logistic regression with $\pm$ responses

If  $y_i \in \{+1, -1\}$ , we have

$$\Pr(y_i = +1 | x_i, \beta) = \frac{1}{1 + \exp(-s_i)},$$

and

$$\Pr(y_i = -1 | x_i, \beta) = \frac{1}{1 + \exp(s_i)}.$$

Combining them, we have

$$p(y_i | x_i, \beta) = \frac{1}{1 + \exp(-y_i s_i)}.$$

The log-likelihood is

$$\sum_{i=1}^n \log \Pr(y_i | x_i, \beta) = - \sum_{i=1}^n \log(1 + \exp(-y_i s_i)).$$

We define the loss function as the negative log-likelihood. Thus

$$L(y_i, s_i) = \log[1 + \exp(-y_i s_i)].$$

This loss is called the logistic loss.

The least squares loss for linear regression can also be derived from the log-likelihood if we assume the errors follow a normal distribution.

### Loss functions for classification

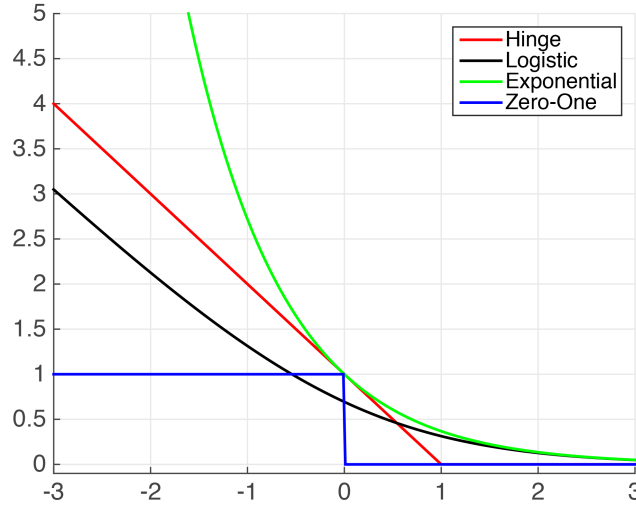


Figure 5: Loss functions for classification. The horizontal axis is  $m_i = y_i x_i^\top \beta$ . The vertical axis is  $L(y_i, x_i^\top \beta)$ . The exponential loss and the hinge loss can be considered approximations to the logistic loss. These loss functions penalize negative  $m_i$ . The more negative  $m_i$  is, the bigger the loss. The loss functions also penalize small positive  $m_i$ , e.g., those  $m_i < 1$ . Such loss functions encourage correct and confident classifications.

The following summarizes several possible choices for the loss function  $L(y_i, s_i)$  for classification. See Figure 5.

$$\begin{aligned} \text{Logistic loss} &= \log(1 + \exp(-y_i s_i)), \\ \text{Exponential loss} &= \exp(-y_i s_i), \\ \text{Hinge loss} &= \max(0, 1 - y_i s_i), \\ \text{Zero-one loss} &= 1(y_i s_i > 0) \end{aligned}$$

Both the exponential and hinge losses can be considered approximations to the logistic loss. The logistic loss is used by logistic regression. The exponential loss is used by adaboost. The hinge loss is used by support vector machines. The zero-one loss is to count the number of mistakes. It is not differentiable and is not used for training.

All the above loss functions are based on  $m_i = y_i s_i$ . We call  $m_i$  the margin for example  $(y_i, x_i)$ . We want  $y_i$  and  $s_i = x_i^\top \beta$  to be of the same sign for correct classification. If  $y_i = +1$ , we want  $s_i = x_i^\top \beta$  to be very positive. If  $y_i = -1$ , we want  $s_i = x_i^\top \beta$  to be very negative. We want the margin  $m_i$  to be as large as possible for confident classification.

## 2.8 Regularization

If we have many parameters, i.e.,  $\beta$  is a high dimensional vector, we need to regularize  $\beta$  and use the following objective function,

$$\mathcal{L}(\beta) = \sum_{i=1}^n L(y_i, s_i) + \lambda \rho(\beta),$$

where  $\rho(\beta)$  is a regularization function and  $\lambda$  is a regularization constant to be carefully selected or tuned.

Popular choices of  $\rho(\beta)$  include: (1)  $\ell_2$  regularization, where  $\rho(\beta) = \|\beta\|_{\ell_2}^2$ , which is the sum of squares of the components of  $\beta$ , giving us the ridge regression; and (2)  $\ell_1$  regularization, where  $\rho(\beta) = \|\beta\|_{\ell_1}$ , which is the sum of the absolute values of the components of  $\beta$ . This gives us the Lasso. Both regularization functions are convex. While  $\ell_2$  induces shrinkage,  $\ell_1$  induces variable selection. Usually we do not penalize the intercept or bias. In machine learning literature, we often write  $\beta$  in terms of  $(w, b)$ , and we only regularize  $w$ , because  $b$  only tells us the overall location of  $\{y_i\}$ , and it does not tell us anything about the relationship between  $y_i$  and  $x_i$ .

## 2.9 Gradient descent: learning from errors

With the loss function  $\mathcal{L}(\beta)$  defined, we can estimate  $\beta$  by minimizing  $\mathcal{L}(\beta)$ . We can use the first order method such as gradient descent. We can also use the second order method such as Newton-Raphson (if we can afford it).

The gradient descent algorithm updates  $\beta$  by

$$\beta_{t+1} = \beta_t - \eta \mathcal{L}'(\beta_t),$$

where  $\eta$  is the step size or learning rate. It may change over time.  $\mathcal{L}'(\beta)$  is the gradient or derivative of  $\mathcal{L}(\beta)$ , which gives us the steepest direction for descent, and

$$\mathcal{L}'(\beta) = \sum_{i=1}^n \frac{\partial}{\partial \beta} L(y_i, s_i) + \lambda \rho'(\beta).$$

For least squares loss,

$$\frac{\partial}{\partial \beta} L(y_i, s_i) = -2(y_i - s_i)x_i = -2e_i x_i,$$

where

$$e_i = y_i - s_i$$

is the error made by the current  $\beta$ .

For logistic regression with  $y_i \in \{0, 1\}$ ,

$$\begin{aligned} \frac{\partial}{\partial \beta} L(y_i, s_i) &= -y_i x_i + \frac{\exp(s_i)}{1 + \exp(s_i)} x_i \\ &= -(y_i - p_i) x_i = -e_i x_i, \end{aligned}$$

where

$$e_i = y_i - p_i$$

is the error made by the current  $\beta$ . If  $y_i = 0$ ,  $p_i$  should be close to 0. If  $y_i = 1$ ,  $p_i$  should be close to 1. Otherwise there will be an error which causes the change in  $\beta$ . The gradient descent algorithm learns from the errors, by incorporating  $e_i x_i$  into  $\beta$ , allowing  $\beta$  memorize those  $x_i$  on which it had made mistakes. In the classification setting, we want  $\beta$  to be a vector that points from negative examples to the positive examples, and the gradient descent algorithm achieves that.

## Newton-Raphson

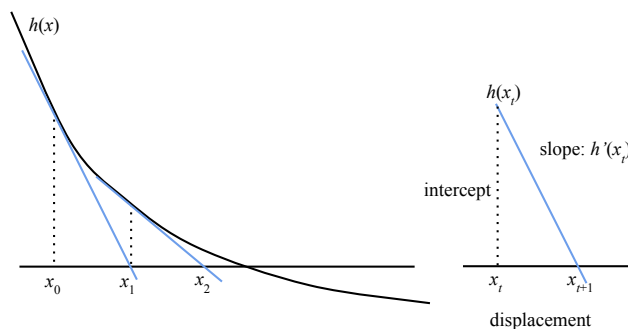


Figure 6: Newton-Raphson

A more efficient method is to update  $\beta$  using Newton-Raphson algorithm. Suppose we want to solve  $h(x) = 0$ . At  $x_t$ , we take the first order Taylor expansion

$$h(x) \doteq h(x_t) + h'(x_t)(x - x_t).$$

Each iteration, we find the root of the linear surrogate function, which is the above first order Taylor expansion,

$$x_{t+1} = x_t - \frac{h(x_t)}{h'(x_t)}.$$

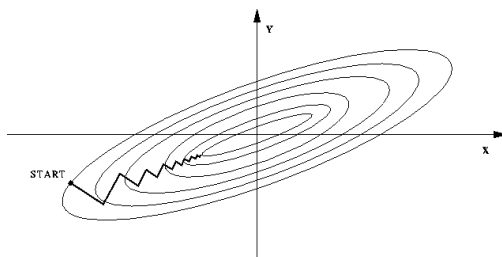


Figure 7: Gradient descent.

Suppose we want to find the mode of  $f(x)$ , we can solve  $f'(x) = 0$ . Using Newton-Raphson, we have

$$x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)}.$$

Each iteration maximizes a quadratic approximation to the original function at  $x_t$ ,

$$f(x) \doteq f(x_t) + f'(x_t)(x - x_t) + \frac{1}{2}f''(x_t)(x - x_t)^2.$$

$f''(x_t)$  is the curvature of  $f$  at  $x_t$ . If the curvature is big, the step size should be small. If the curvature is small, the step size can be made larger.

If the variable is a vector  $x = (x_1, x_2, \dots, x_n)^\top$ , let

$$f'(x) = \left( \frac{\partial f}{\partial x_i} \right)_{n \times 1} \quad f''(x) = \left( \frac{\partial^2 f}{\partial x_i \partial x_j} \right)_{n \times n},$$

$f''(x_t)$  is called the Hessian matrix, we have

$$x_{t+1} = x_t - f''(x_t)^{-1} f'(x_t).$$

$f''(x_t)$  tells us the local shape of  $f$  around  $x_t$ .  $f''(x_t)^{-1} f'(x_t)$  gives us better direction than  $f'(x_t)$  as shown in the above figure. The Newton-Raphson is a second order algorithm.

## 2.10 Iterated reweighed least squares

For maximum likelihood estimate of  $\beta$  in logistic regression, let  $l(\beta)$  be the log-likelihood,

$$l(\beta) = \text{LogLikelihood}(\beta) = \sum_{i=1}^n \log \Pr(y_i | x_i, \beta) = \sum_{i=1}^n [y_i s_i - \log(1 + \exp(s_i))].$$

To find the maximum of  $l(\beta)$ , we first calculate the gradient

$$l'(\beta) = \sum_{i=1}^n \left[ y_i x_i - \frac{e^{x_i^\top \beta}}{1 + e^{x_i^\top \beta}} x_i \right] = \sum_{i=1}^n (y_i - p_i) x_i.$$

The second derivative of the log likelihood function is

$$l''(\beta) = - \sum_{i=1}^n p_i (1 - p_i) x_i x_i^\top.$$

We can update  $\beta$  by

$$\beta^{(t+1)} = \beta^{(t)} + l''(\beta^{(t)})^{-1} l'(\beta^{(t)}).$$

Let  $w_i = p_i(1 - p_i)$ , we can rewrite the update equation as

$$\begin{aligned} \beta^{(t+1)} &= \beta^{(t)} + \left[ \sum_{i=1}^n p_i (1 - p_i) x_i x_i^\top \right]^{-1} (y_i - p_i) x_i \\ &= \left( \sum_{i=1}^n w_i x_i x_i^\top \right)^{-1} \left[ \sum_{i=1}^n w_i x_i x_i^\top \beta^{(t)} + (y_i - p_i) x_i \right] \\ &= \left( \sum_{i=1}^n w_i x_i x_i^\top \right)^{-1} \left[ \sum_{i=1}^n w_i x_i \left( x_i^\top \beta^{(t)} + \frac{y_i - p_i}{w_i} \right) \right]. \end{aligned}$$

Let

$$\hat{y}_i = x_i^\top \beta^{(t)} + \frac{y_i - p_i}{w_i},$$

let  $\tilde{x}_i = x_i \sqrt{w_i}$ ,  $\tilde{y}_i = \hat{y}_i \sqrt{w_i}$ , we can rewrite the equation above as follows:

$$\begin{aligned} \beta^{(t+1)} &= \left( \sum_{i=1}^n w_i x_i x_i^\top \right)^{-1} \left( \sum_{i=1}^n w_i x_i \hat{y}_i \right) \\ &= \left( \sum_{i=1}^n \tilde{x}_i \tilde{x}_i^\top \right)^{-1} \left( \sum_{i=1}^n \tilde{x}_i \tilde{y}_i \right). \end{aligned}$$

The flow is

$$\beta^{(t)} \rightarrow \eta_i = x_i^\top \beta^{(t)} \rightarrow p_i = \sigma(\eta_i) \rightarrow w_i = p_i(1 - p_i) \rightarrow \hat{y}_i = \eta_i + \frac{y_i - p_i}{w_i} \rightarrow \tilde{x}_i = x_i \sqrt{w_i}, \tilde{y}_i = \hat{y}_i \sqrt{w_i} \rightarrow \beta^{(t+1)}.$$

XGboost can be considered a variation of iterated reweighed least squares.

## 2.11 Multivariate or multinomial response

### Regression

For regression,  $y_i$  and  $s_i$  can be multivariate, i.e.,  $y_i = (y_{ik}, k = 1, \dots, K)^\top$  and  $s_i = (s_{ik}, k = 1, \dots, K)^\top$ . We can let

$$s_{ik} = x_i^\top \beta^{(k)}, k = 1, \dots, K,$$

where each  $\beta^{(k)}$  is a  $p$ -dimensional coefficient vector. We can write  $\beta = (\beta^{(k)}, k = 1, \dots, K)$  into a  $p \times K$  coefficient matrix. The loss function or the loss layer is

$$L_i(y_i, s_i) = \frac{1}{2} |y_i - s_i|^2.$$

In this case,

$$-\frac{\partial L_i(y_i, s_i)}{\partial s_i} = y_i - s_i = e_i,$$

i.e., the error.

### Classification

For classification, suppose we want to classify the input  $x_i$  into one of  $K$  categories. We can generalize the binary logistic regression to multinomial logistic regression, by defining

$$s_{ik} = x_i^\top \beta^{(k)}, k = 1, \dots, K,$$

where again each  $\beta^{(k)}$  is a  $p$ -dimensional coefficient vector. We can write  $\beta = (\beta^{(k)}, k = 1, \dots, K)$  into a  $p \times K$  coefficient matrix. The class probability is defined as the soft-max

$$\Pr(y_i = k | x_i, \beta) = \frac{\exp(s_{ik})}{\sum_{k'} \exp(s_{ik'})}.$$

We use the notation  $k'$  to avoid the confusion with the index  $k$  in the numerator.

We may fix  $\beta^{(1)} = 0$  since adding an arbitrary constant vector to all the  $\beta^{(k)}$  does not change class probability. The above model reduces to binary logistic regression if  $K = 2$ .

The above probability is called soft-max. We may also define

$$\hat{y}_i = \arg \max_k s_{ik},$$

which is a hard-max. The hard-max reduces to the perceptron if  $K = 2$ .

Suppose our training dataset is  $(x_i, y_i)$ ,  $i = 1, \dots, n$ , where  $x_i$  is the input vector and  $y_i$  is the output category. Let  $p(y|x, \beta)$  be the probability that the input  $x$  belongs to the category  $y$  according to the above softmax probability. Then we can estimate  $\beta$  by maximizing the log-likelihood

$$\sum_{i=1}^n \log p(y_i | x_i, \beta).$$



We can define the loss to be the negative log-likelihood

$$\mathcal{L}(\beta) = -\sum_{i=1}^n \log p(y_i|x_i, \beta) = \sum_{i=1}^n L_i(y_i, s_i),$$

where  $L_i(y_i, s_i) = -\log p(y_i|x_i, \beta)$ . If  $y_i = k$ , then

$$\begin{aligned} -\frac{\partial L_i(y_i, s_i)}{\partial s_{k'}} &= 1(k' = k) - \frac{\partial}{\partial s_{k'}} \log \sum_{k'=1}^K \exp(s_{ik'}) \\ &= 1(k' = k) - \frac{\exp(s_{ik'})}{\sum_{k'=1}^K \exp(s_{ik'})} \\ &= 1(k' = k) - p_{ik'}. \end{aligned}$$

Let  $Y_i$  be the one-hot vector, i.e., if  $y_i = k$ , then the  $k$ -th element of  $Y_i$  is 1, and 0 otherwise. Let  $p_i = (p_{ik}, k = 1, \dots, K)$  be the vector of predicted probabilities of the  $K$  categories. Then

$$-\frac{\partial L_i(y_i, s_i)}{\partial s_{k'}} = Y_i - p_i = e_i.$$

We can then use chain rule to obtain the derivatives with respect to  $\beta^{(k)}$ , which, again, results in learning from errors. Later we may simply write  $Y_i$  as  $y_i$  with slight abuse of notation.

## Unification

We can unify both regression and classification, so that

$$p(y|s) = \frac{1}{Z(s)} \exp(\langle y, s \rangle),$$

where

$$Z(s) = \sum_y \exp(\langle y, s \rangle)$$

is the normalizing constant. For regression,  $y \sim N(0, \sigma^2 I_K)$ , where  $I_K$  is the  $K$ -dimensional identity matrix, and the sum in the above equation becomes an integral. For classification,  $y$  is one-hot vector.

Then we can show that

$$\frac{\partial \log p(y|s)}{\partial s} = y - \mathbb{E}(y|s),$$

which is the error.

## 2.12 Non-linear and non-parametric functions

We may replace the linear function  $x_i^\top \beta$  by a more flexible non-linear function  $f(x_i)$ . The function  $f(x)$  (here we use  $x$  to denote a generic  $x_i$ ) is sometimes called non-parametric, but it is actually highly parametrized or over-parametrized. In what follows, we shall study the more flexible  $f(x)$ . In support vector machine, adaboost, and neural networks,  $f(x) = h(x)^\top \beta$ , where  $h(x)$  takes different forms in different methods.

### 3 Model Complexity and Overfitting

#### 3.1 Gauss' analysis of least squares

Gauss justified the least squares estimator by proving that among all the linear unbiased estimators, including Laplace's estimating equation, the least squares estimator has the minimal variance.

In Gauss' analysis, he adopted the frequentist framework, by assuming  $Y = X\beta_{\text{true}} + \varepsilon$ , with  $\mathbb{E}(\varepsilon) = 0$  and  $\text{Var}(\varepsilon) = \sigma^2 I_n$ . For least squares estimate  $\hat{\beta}_{\text{LS}}$ ,

$$\hat{\beta}_{\text{LS}} = (X^\top X)^{-1} X^\top Y = (X^\top X)^{-1} X^\top (X\beta_{\text{true}} + \varepsilon) = \beta_{\text{true}} + X^{-1} \varepsilon,$$

therefore

$$\mathbb{E}[\hat{\beta}_{\text{LS}}] = \mathbb{E}[\beta_{\text{true}} + X^{-1} \varepsilon] = \beta_{\text{true}} + X^{-1} \mathbb{E}[\varepsilon] = \beta_{\text{true}},$$

and

$$\begin{aligned} \text{Var}(\hat{\beta}_{\text{LS}}) &= \text{Var}(\beta_{\text{true}} + X^{-1} \varepsilon) = \text{Var}(X^{-1} \varepsilon) \\ &= X^{-1} \text{Var}(\varepsilon) X^{-\top} = \sigma^2 (X^\top X)^{-1}. \end{aligned}$$

For a general linear estimator  $\hat{\beta} = AY$ , where  $A$  may depend on  $X$ . If  $\mathbb{E}(\hat{\beta}) = \beta_{\text{true}}$ , then

$$\mathbb{E}[\hat{\beta}] = AX\beta_{\text{true}} = \beta_{\text{true}} \Rightarrow AX = I_p.$$

Further,

$$\text{Var}(\hat{\beta}) = \text{Var}(AY) = A \text{Var}(\varepsilon) A^\top = \sigma^2 AA^\top.$$

According to Cauchy-Schwartz inequality, for the projection matrix  $H = X(X^\top X)^{-1} X^\top$ , we have  $H^\top = H$  and  $H^2 = H$ , and  $Y^\top Y \geq Y^\top H^\top H Y = Y^\top H Y$ . Therefore

$$\text{Var}(\hat{\beta}) = \sigma^2 AA^\top \geq \sigma^2 AX(X^\top X)^{-1} X^\top A^\top = \sigma^2 (X^\top X)^{-1} = \text{Var}(\hat{\beta}_{\text{LS}}).$$

This proves the Gauss-Markov theorem.

#### Background: expectation and variance in matrix form

Consider a random matrix  $X$ . Suppose  $X$  is  $m \times n$ , and the elements of  $X$  are  $x_{ij}$ ,  $i = 1, \dots, m$  and  $j = 1, \dots, n$ . Usually we write  $X = (x_{ij})_{m \times n}$  or simply  $X = (x_{ij})$ . We define

$$\mathbb{E}(X) = (\mathbb{E}(x_{ij})),$$

i.e., taking expectations element-wise. Let  $A$  be a constant matrix of appropriate dimension, then  $\mathbb{E}(AX) = A\mathbb{E}(X)$ . Let  $B$  be another constant matrix of appropriate dimension, then  $\mathbb{E}(XB) = \mathbb{E}(X)B$ .

The above result can be easily understood if we have iid copies  $X_1, \dots, X_n$ , so that  $\sum_{i=1}^n x_i/n \rightarrow \mathbb{E}(X)$ , and  $\sum_{i=1}^n Ax_i/n \rightarrow \mathbb{E}(AX)$ , but also  $\sum_{i=1}^n Ax_i/n = A \sum_{i=1}^n x_i/n \rightarrow A\mathbb{E}(X)$ . Thus  $\mathbb{E}(AX) = A\mathbb{E}(X)$ .

Let  $X$  be a random vector. Let  $\mu_X = \mathbb{E}(X)$ . We define

$$\text{Var}(X) = \mathbb{E}[(X - \mu_X)(X - \mu_X)^\top].$$

Then the  $(i, j)$ -th element of  $\text{Var}(X)$  is  $\text{Cov}(x_i, x_j)$ . The diagonal elements are  $\text{Var}(x_i)$ .

Let  $A$  be a constant matrix of appropriate dimension, then

$$\text{Var}(AX) = A \text{Var}(X) A^\top.$$

This is because

$$\begin{aligned}
\text{Var}(AX) &= \mathbb{E}[(AX - \mathbb{E}(AX))(AX - \mathbb{E}(AX))^\top] \\
&= \mathbb{E}[(AX - A\mu_X)(AX - A\mu_X)^\top] \\
&= \mathbb{E}[A(X - \mu_X)(X - \mu_X)^\top A^\top] \\
&= A\mathbb{E}[(X - \mu_X)(X - \mu_X)^\top]A^\top \\
&= A\text{Var}(X)A^\top.
\end{aligned}$$

Note that  $A$  does not need to be a square matrix.  $A$  can even be a vector, such as  $a^\top$ , then  $\text{Var}(a^\top X) = a^\top \text{Var}(X)a$ , which is a quadratic form.  $\text{Var}(X)$  is non-negative definite.

### 3.2 Bias and variance tradeoff

While Gauss showed that  $\hat{\beta}_{LS}$  has minimal variance among all the linear unbiased estimators, one may ask whether a biased estimator may be even better than the least squares estimator. For an estimator  $\hat{\beta}$ , let  $\mu = \mathbb{E}(\hat{\beta})$ . The mean squares error can be decomposed into bias and variance terms,

$$\begin{aligned}
\mathbb{E}\|\hat{\beta} - \beta_{\text{true}}\|^2 &= \mathbb{E}\|(\hat{\beta} - \mu) + (\mu - \beta_{\text{true}})\|^2 \\
&= \mathbb{E}\|\hat{\beta} - \mu\|^2 + \|\mu - \beta_{\text{true}}\|^2 = \text{variance} + \text{bias}^2.
\end{aligned}$$

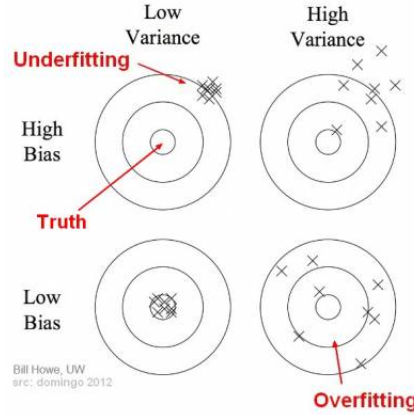


Figure 8: Bias and variance tradeoff.

The above figure illustrates the bias and variance tradeoff. It is possible to introduce some bias so as to reduce the variance.

### 3.3 Stein's estimator

Stein shocked the world of statistics in 1960s by showing that as long as  $p > 2$ , you can always do better than least squares estimator.

For simplicity, let us assume that  $X^\top X = I_p$ , i.e.,  $(X_j, j = 1, \dots, p)$  form an orthonormal basis. Then

$$\hat{\beta}_{LS} = X^\top (X\beta_{\text{true}} + \varepsilon) = \beta_{\text{true}} + X^\top \varepsilon = \beta_{\text{true}} + \delta,$$

where  $\delta = X^\top \varepsilon$  is the model fitting to the noise  $\varepsilon$ . Suppose  $\varepsilon \sim N(0, \sigma^2 I_n)$ , then  $\delta \sim N(0, \sigma^2 I_p)$ , because  $\text{Var}(\delta) = X^\top \text{Var}(\varepsilon) X = \sigma^2 I_p$ . Thus  $\hat{\beta}_{\text{LS}} \sim N(\beta_{\text{true}}, \sigma^2 I_p)$ .

Stein's estimator is defined as

$$\hat{\beta}_{\text{Stein}} = \left(1 - \frac{(p-2)\sigma^2}{\|\hat{\beta}_{\text{LS}}\|^2}\right) \hat{\beta}_{\text{LS}}.$$

Stein proved that for  $p > 2$

$$\mathbb{E}\|\hat{\beta}_{\text{Stein}} - \beta_{\text{true}}\|^2 \leq \mathbb{E}\|\hat{\beta}_{\text{LS}} - \beta_{\text{true}}\|^2.$$

Stein's estimator is based on the observation that  $\|\hat{\beta}_{\text{LS}}\|^2 = \|\beta_{\text{true}}\|^2 + p\sigma^2$ . That is, the vector  $\hat{\beta}_{\text{LS}}$  is longer than the vector  $\beta_{\text{true}}$ . Thus we may want to shrink  $\hat{\beta}_{\text{LS}}$  to bring it closer to  $\beta_{\text{true}}$ . Stein's estimator is an example of shrinkage estimator or ridge regression.

**Stein's Lemma** For  $z \sim N(\mu, \sigma^2)$ ,  $\mathbb{E}[(z - \mu)g(z)] = \sigma^2 \mathbb{E}[g'(z)]$ .

**proof** Integral by parts,

$$\begin{aligned} \mathbb{E}[(z - \mu)g(z)] &= \int (z - \mu)g(z) \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} dz \\ &= -\sigma^2 g(z) \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \Big|_{-\infty}^{\infty} + \sigma^2 \int g'(z) \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} dz \\ &= \sigma^2 \mathbb{E}[g'(z)]. \end{aligned}$$

### Proof of Stein's result

For simplicity, we denote  $\hat{\beta}_{\text{LS}}$  by  $X$ , and denote  $\beta_{\text{true}}$  by  $\theta$ , so that  $X \sim N(\theta, \sigma^2 I_p)$ .

$$\mathbb{E}\|X - \theta\|^2 = \mathbb{E}\left[\sum_{i=1}^p (X_i - \theta_i)^2\right] = p\sigma^2.$$

$$\begin{aligned} \mathbb{E}\left[\left\|\left(1 - \frac{(p-2)\sigma^2}{\|X\|^2}\right)X - \theta\right\|^2\right] &= \mathbb{E}\left[\left\|(X - \theta) - \frac{(p-2)\sigma^2}{\|X\|^2}X\right\|^2\right] \\ &= \mathbb{E}\|X - \theta\|^2 + \mathbb{E}\left[\frac{(p-2)^2\sigma^4}{\|X\|^2}\right] - 2\mathbb{E}\left[\langle X - \theta, \frac{(p-2)\sigma^2}{\|X\|^2}X \rangle\right] = (*). \end{aligned}$$

$\mathbb{E}\left[\langle X - \theta, \frac{(p-2)\sigma^2}{\|X\|^2}X \rangle\right]$  can be simplified as follows

$$\mathbb{E}\left[\sum_{i=1}^p (X_i - \theta_i) \frac{(p-2)\sigma^2}{\|X\|^2} X_i\right] = \mathbb{E}\left[\sum_{i=1}^p (X_i - \theta_i) \frac{(p-2)\sigma^2}{X_i^2 + \sum_{j \neq i}^p X_j^2} X_i\right].$$

Letting  $\frac{(p-2)\sigma^2}{X_i^2 + \sum_{j \neq i}^p X_j^2} X_i = g(x_i)$ , we can use Stein's lemma. Hence

$$\begin{aligned}
\mathbb{E} \left[ \sum_{i=1}^p (X_i - \theta_i) \frac{(p-2)\sigma^2}{X_i^2 + \sum_{j \neq i}^p X_j^2} X_i \right] &= \sigma^2 \sum_{i=1}^p \mathbb{E} \left[ \left( \frac{(p-2)\sigma^2}{X_i^2 + \sum_{j \neq i}^p X_j^2} X_i \right)' \right] \\
&= \sigma^2 \sum_{i=1}^p \mathbb{E} \left[ \frac{(p-2)\sigma^2}{\|X\|^2} - \frac{(p-2)\sigma^2 2X_i^2}{\|X\|^4} \right] \\
&= \sigma^2 \mathbb{E} \left[ \frac{p(p-2)\sigma^2}{\|X\|^2} - \frac{2(p-2)\sigma^2}{\|X\|^2} \right] \\
&= \mathbb{E} \left[ \frac{(p-2)^2 \sigma^4}{\|X\|^2} \right].
\end{aligned}$$

Therefore,

$$\begin{aligned}
(*) &= \mathbb{E} \left[ \|X - \theta\|^2 + \frac{(p-2)^2 \sigma^4}{\|X\|^2} - 2 \frac{(p-2)^2 \sigma^4}{\|X\|^2} \right] \\
&= \mathbb{E} \left[ \|X - \theta\|^2 - \frac{(p-2)^2 \sigma^4}{\|X\|^2} \right] \leq \mathbb{E} [\|X - \theta\|^2] = p\sigma^2.
\end{aligned}$$

### 3.4 Stein's estimator as empirical Bayes

$X \sim N(\theta, \sigma^2 I_p)$  can be written as  $X = (x_i, i = 1, \dots, p)^\top$ , and  $x_i \sim N(\theta_i, \sigma^2)$ . We can adopt a Bayesian treatment, by assuming a prior distribution  $\theta_i \sim N(0, \tau^2)$ . Marginally, if we integrate out  $\theta_i$ , we have  $x_i \sim N(0, \sigma^2 + \tau^2)$ , and the posterior distribution of each  $\theta_i$  is

$$p(\theta_i | X) \sim N \left( x_i \frac{1/\sigma^2}{1/\sigma^2 + 1/\tau^2}, \frac{1}{1/\sigma^2 + 1/\tau^2} \right),$$

and

$$\mathbb{E}(\theta_i | X) = x_i \left( 1 - \frac{\sigma^2}{\sigma^2 + \tau^2} \right).$$

Since  $x_i \sim N(0, \sigma^2 + \tau^2)$ , we have

$$\mathbb{E}(1/\|X\|^2) = 1/[(\sigma^2 + \tau^2)(p-2)].$$

Thus we can estimate  $1/(\sigma^2 + \tau^2)$  by  $(p-2)/\|X\|^2$ . This leads to Stein's estimator.

Gaussian process is a generalization of this idea. The estimation of the hyper-parameters in Gaussian process is similar to the estimation of  $\tau^2$ . It is called empirical Bayes because  $\tau^2$  or  $1/(\sigma^2 + \tau^2)$  is estimated from empirical data. A full Bayesian treatment would put a prior on  $\tau^2$ .

### 3.5 Model bias

In Gauss' analysis, he assumed the model is true, or the Newtonian mechanics is true. According to George Box: "All models are wrong, but some are useful." Thus it is better to assume that the model is wrong or biased.

Suppose  $Y = g + \varepsilon$ , where  $g$  is the vector of ground truth, e.g., the true positions of a planet over time,  $\varepsilon$  is the noise vector. For simplicity, let us also assume that  $X^\top X = I_p$ , i.e.,  $(X_j, j = 1, \dots, p)$  form an orthonormal basis. Then

$$\hat{\beta} = X^\top (g + \varepsilon) = X^\top g + X^\top \varepsilon = \beta^* + \delta,$$

where  $\beta^* = X^\top g$  is the “best” value of  $\beta$ , i.e., the best the model can do to fit the ground truth, and  $\delta = X^\top \varepsilon$  is the model fitting to the noise  $\varepsilon$ .  $\hat{\beta}_j = \beta_j^* + \delta_j$ , for  $j = 1, \dots, p$ .

Let  $\hat{g} = X\beta^*$ , then  $\hat{g}$  is the projection of  $g$  onto the subspace spanned by  $X$ . We call  $|g - \hat{g}|^2$  the model bias.

Let  $\hat{\varepsilon} = X\delta$ , then  $\hat{\varepsilon}$  is the projection of  $\varepsilon$  onto the space  $X$ .

Suppose  $\varepsilon \sim N(0, \sigma^2 I_n)$ , then  $\delta \sim N(0, \sigma^2 I_p)$ , because  $\text{Var}(\delta) = X^\top \text{Var}(\varepsilon)X = \sigma^2 I_p$ , and  $\|X\delta\|^2 = \delta^\top X^\top X \delta = \|\delta\|^2 = p\sigma^2$ .

### 3.6 Training and testing errors, overfitting

The training error is

$$\begin{aligned} \mathbb{E}|Y - \hat{Y}|^2 &= |g - \hat{g}|^2 + \mathbb{E}|\varepsilon - \hat{\varepsilon}|^2 \\ &= |g - \hat{g}|^2 + \mathbb{E}|\varepsilon|^2 - \mathbb{E}|\hat{\varepsilon}|^2 \\ &= |g - \hat{g}|^2 + (n - p)\sigma^2. \end{aligned}$$

Suppose  $X$  is fixed, and the test data is  $\tilde{Y} = g + \tilde{\varepsilon}$ , where  $\tilde{\varepsilon}$  is independent of  $\varepsilon$ . Then the testing error is

$$\begin{aligned} \mathbb{E}|\tilde{Y} - \hat{Y}|^2 &= |g - \hat{g}|^2 + \mathbb{E}|\tilde{\varepsilon} - \hat{\varepsilon}|^2 \\ &= |g - \hat{g}|^2 + \mathbb{E}|\tilde{\varepsilon}|^2 + \mathbb{E}|\hat{\varepsilon}|^2 \\ &= |g - \hat{g}|^2 + (n + p)\sigma^2. \end{aligned}$$

The error of  $\beta^*$  is  $|g - \hat{g}|^2$ .

Thus overfitting is defined as the testing error minus the training error, which is  $2p\sigma^2$ .

We may also interpret overfitting as the error of  $\hat{\beta}$  minus the error of  $\beta^*$ , which is  $p\sigma^2$ . On the training data, the learned  $\hat{\beta}$  does even better than the ground truth  $\beta^*$ . However, on the testing data, the former does worse than the latter.

When we increase the model complexity  $p$ , the training error will keep decreasing. The testing error will decrease and then start to increase, that is, the testing error has a U-shape. See Figure 9.

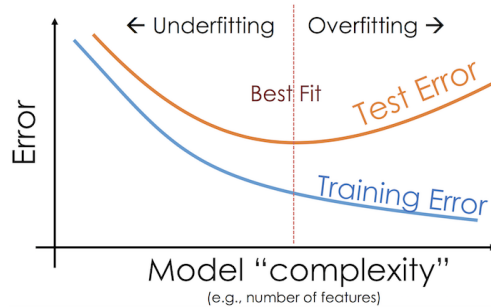


Figure 9: Training and testing errors over model complexity.

### 3.7 Learning from noise

If  $g = 0$ , then the model learns from noise, and it becomes clear how the model overfits. See Figure 10.

The training error is  $\mathbb{E}|\varepsilon - \hat{\varepsilon}|^2$ , and the testing error is  $\mathbb{E}|\tilde{\varepsilon} - \hat{\varepsilon}|^2$ . While  $\hat{\varepsilon}$  is the projection of  $\varepsilon$  onto  $X$ , it is not the projection of  $\tilde{\varepsilon}$  onto  $X$ .

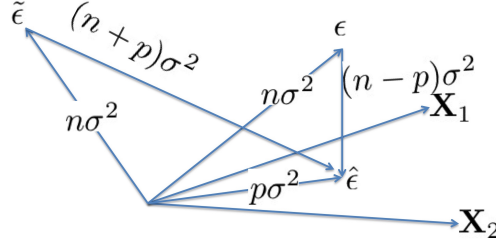


Figure 10: Overfitting due to learning from noise. Training error is  $\mathbb{E}|\epsilon - \hat{\epsilon}|^2 = (n - p)\sigma^2$ , and testing error is  $\mathbb{E}|\tilde{\epsilon} - \hat{\epsilon}|^2 = (n + p)\sigma^2$ .

### 3.8 Effective degrees of freedom

Consider a general estimator  $\hat{\beta}$  such as the ridge estimator. The training error is

$$\mathbb{E}|Y - X\hat{\beta}|^2 = \mathbb{E}|g + \epsilon - X\hat{\beta}|^2.$$

The testing error is

$$\mathbb{E}|\tilde{Y} - X\hat{\beta}|^2 = \mathbb{E}|g + \tilde{\epsilon} - X\hat{\beta}|^2.$$

The overfitting = testing error - training error is

$$2\mathbb{E}\langle \epsilon, X\hat{\beta} \rangle,$$

which is how much  $\hat{\beta}$  absorbs the noise  $\epsilon$ .

Recall for the least squares  $\hat{\beta}$  overfitting is  $2p\sigma^2$ . We can define the effective degrees of freedom of  $\hat{\beta}$  to be

$$\hat{p} = \frac{\mathbb{E}\langle \epsilon, X\hat{\beta} \rangle}{\sigma^2}.$$

With stronger regularization, e.g., bigger  $\lambda$  in the ridge regression, we have smaller  $\hat{p}$  and less overfitting.

For a general estimator, the testing error can be decomposed into model bias, estimation bias, estimation variance, and noise,

$$\mathbb{E}|\tilde{Y} - X\hat{\beta}|^2 = |g - \hat{g}|^2 + |\mathbb{E}(\hat{\beta}) - \beta^*|^2 + \mathbb{E}|\hat{\beta} - \mathbb{E}(\hat{\beta})|^2 + n\sigma^2.$$

As we increase model complexity, the model bias decreases, the estimation bias decreases, but the variance increases.

### 3.9 Classification

#### Error and score

Suppose the classifier is  $\hat{y} = f(x) \in \{-1, +1\}$ . The classification error is  $1(f(x) \neq y)$ , and the classification score is  $f(x)y$ . Clearly score =  $1 - 2 \times$  error. We shall analyze the training score and testing score. Note that here we treat  $f(x)$  as the binary-valued function, unless in other parts of the note.

#### Coin flipping

We shall first study the model fitting of “coin flippings,” i.e., we assume  $y = \epsilon \sim \text{Bernoulli}(1/2)$ , i.e.,  $\Pr(\epsilon = 1) = \Pr(\epsilon = -1) = 1/2$ .

A property of coin flipping is that for any Bernoulli random variable  $Z \sim \text{Bernoulli}(p) \in \{+1, -1\}$ , we have  $\epsilon Z \sim \text{Bernoulli}(1/2)$ .

### 3.10 Learning from coin flipping and Rademacher complexity

For any classifier  $f(x)$ , the testing score is  $\mathbb{E}[\epsilon f(x)] = 0$ .

Suppose our dataset is  $(x_i, y_i), i = 1, \dots, n$  where  $y_i = \epsilon_i \sim \text{Bernoulli}(1/2)$  that follows a coin flipping process. For a classifier  $f(x)$ , the training score is  $\frac{1}{n} \sum_{i=1}^n y_i f(x_i)$ . However, in the training stage, we can select  $f \in \mathcal{F}$ , which is the class of all possible classifiers. For instance, if  $f(x) = \text{sign}(x^\top \beta)$  is a linear classifier, then  $\mathcal{F} = \{f(x) = \text{sign}(x^\top \beta), \forall \beta\}$ . Then the training score of the optimal classifier is

$$R = \mathbb{E} \left[ \max_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \epsilon_i f(x_i) \right].$$

The above is the Rademacher complexity of the class  $\mathcal{F}$ . It is similar to the effective degrees of freedom in regression.

### 3.11 Growth number and VC-dimension

Let  $N = |\{(f(x_1), \dots, f(x_n)), \forall f \in \mathcal{F}\}|$  be the number of all possible predicted sequences  $(f(x_1), \dots, f(x_n))$  if we vary  $f$  in  $\mathcal{F}$ . Clearly  $N \leq 2^n$ .  $N$  is called the growth number. The growth number plays a role in the union bound of the training error, i.e.,

$$\Pr \left( \max_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \epsilon_i f(x_i) > t \right) \leq N \Pr \left( \frac{1}{n} \sum_{i=1}^n \epsilon_i > t \right) \leq N \exp(-2nt^2).$$

The union bound is that  $P(A \cup B) \leq P(A) + P(B)$ . Consider taking an exam  $N$  times and suppose each time the probability of passing the exam is  $p$ . If you take the maximal score, then you increase your chance of passing the exam. But this chance is still bounded by  $Np$ . The exponential term above is due to the concentration inequality.

Recall  $N \leq 2^n$ . The VC-dimension is the largest  $n$ , so that  $N = 2^n$ , i.e., the number of coin flippings that can be perfectly explained by selecting  $f$  from  $\mathcal{F}$ . Recall in the linear regression model, the dimension  $p$  is the number of noises we can perfect explain by our model  $f(x) = x^\top \beta$ . Letting  $p$  be the VC-dimension of the class  $\mathcal{F}$ , one can prove that

$$N \leq \left( \frac{en}{p} \right)^p,$$

i.e., the dimension of the space  $\{(f(x_1), \dots, f(x_n)), \forall f \in \mathcal{F}\}$  is  $p$ .

The Rademacher complexity  $R$  is bounded by the growth number  $N$ .  $R$  is your maximal score if you take the exam  $N$  times.

$$R \leq \sqrt{\frac{2 \log N}{n}}.$$

### 3.12 Symmetrization

In general  $(x, y) \sim p(x, y)$ , where  $[y|x] \sim p(y|x)$ , and  $y$  does not follow Bernoulli(1/2). In the regression setting,  $g$  or  $|g - \hat{g}|^2$  gets cancelled in comparing the training error and testing error, so that what matters is the noise vector  $\epsilon$ . This is also the case with classification, due to the following symmetrization trick.

Suppose the training data is  $(x_i, y_i) \sim p(x, y), i = 1, \dots, n$ . For any  $f$ , consider

$$\frac{1}{n} \sum_{i=1}^n \epsilon_i y_i f(x_i) = \frac{1}{n} \left[ \sum_{i: \epsilon_i = +1} y_i f(x_i) - \sum_{i: \epsilon_i = -1} y_i f(x_i) \right],$$



which can be interpreted as cross-validation version of overfitting, i.e., randomly assign half of the training data as “training”, and the other half as “testing”. However,  $\varepsilon_i y_i \sim \text{Bernoulli}(1/2)$ , thus the cross-validation overfitting has the same distribution as  $\frac{1}{n} \sum_{i=1}^n \varepsilon_i f(x_i)$ , which is about learning from noise.

During testing, suppose that we have  $(\tilde{x}_i, \tilde{y}_i) \sim p(x, y), i = 1, \dots, n$ . For classifier  $f(x)$ , the testing score is  $\frac{1}{n} \sum_{i=1}^n \tilde{y}_i f(\tilde{x}_i)$ . The bound of overfitting can be calculated by

$$\begin{aligned} \mathbb{E} \left[ \frac{1}{n} \sum_{i=1}^n y_i f(x_i) - \frac{1}{n} \sum_{i=1}^n \tilde{y}_i f(\tilde{x}_i) \right] &\leq \mathbb{E} \left[ \max_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n y_i f(x_i) - \frac{1}{n} \sum_{i=1}^n \tilde{y}_i f(\tilde{x}_i) \right] \\ &= \mathbb{E} \left[ \max_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n (y_i f(x_i) - \tilde{y}_i f(\tilde{x}_i)) \varepsilon_i \right], \quad \varepsilon_i \sim \text{Bernoulli}(1/2) \\ &\leq \mathbb{E} \left[ \max_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n y_i \varepsilon_i f(x_i) + \max_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n (-\tilde{y}_i) \varepsilon_i f(\tilde{x}_i) \right] \\ &= 2R. \end{aligned}$$

That is, the overfitting is bounded by  $2R$ .

## 4 Kernel Regression

In kernel machines,  $f(x) = h(x)^\top \beta$ , and  $h(x)$  is a high-dimensional or infinite dimensional vector. It is implicit. We only need to know  $K(x, x') = \langle h(x), h(x') \rangle$ .  $K(x, x')$  is the kernel.

For the beginner, it is best to stick to the form  $h(x)^\top \beta$ . Gradually, one can think more in terms of reproducing kernels.

### 4.1 Ridge regression

In order to reduce the model bias, we want the number of parameters to be large. However, this will cause overfitting if we continue to use the least squares estimator. We may reduce overfitting by using a biased estimator such as ridge regression.

The ridge regression minimizes

$$\mathcal{L}(\beta) = |Y - X\beta|^2 + \lambda |\beta|^2,$$

for  $\lambda \geq 0$ . Similar to the derivation in least square estimator, we have

$$\begin{aligned} \mathcal{L}'(\beta) &= -2X^\top(Y - X\beta) + 2\lambda\beta \\ \mathcal{L}''(\beta) &= 2(X^\top X + \lambda I) \end{aligned}$$

Setting

$$\mathcal{L}'(\beta) = -2X^\top(Y - X\beta) + 2\lambda\beta = 0,$$

we have

$$\hat{\beta} = (X^\top X + \lambda I_p)^{-1} X^\top Y.$$

### 4.2 Re-representing ridge regression

Consider the ridge regression whose loss function is

$$\mathcal{L}(\beta) = |Y - X\beta|^2 + \lambda |\beta|^2,$$

for  $\lambda \geq 0$ . The estimator is

$$\hat{\beta} = (X^\top X + \lambda I_p)^{-1} X^\top Y.$$

The estimated function

$$\hat{f}(x) = x^\top \hat{\beta} = x^\top (X^\top X + \lambda I_p)^{-1} X^\top Y = x^\top X^\top (XX^\top + \lambda I_n)^{-1} Y.$$

The reason

$$(X^\top X + \lambda I_p)^{-1} X^\top = X^\top (XX^\top + \lambda I_n)^{-1}$$

is that

$$X^\top (XX^\top + \lambda I_n) = (X^\top X + \lambda I_p) X^\top.$$

Define

$$K(x, x') = \langle x, x' \rangle$$

to be the linear kernel. Define the matrix  $\mathbf{K} = (\mathbf{K}_{ij} = K(x_i, x_j))$ . Then

$$\mathbf{K} = XX^\top.$$

Let

$$c = (\mathbf{K} + \lambda I_n)^{-1} Y = (c_i, i = 1, \dots, n)^\top.$$

Then

$$\hat{\beta} = X^\top c = \sum_{i=1}^n c_i x_i.$$

This is the representer result, which shows that  $\hat{\beta}$  can be represented as a linear combination of the training examples. Then

$$\hat{f}(x) = x^\top \hat{\beta} = x^\top \sum_{i=1}^n c_i x_i = \sum_{i=1}^n c_i \langle x_i, x \rangle = \sum_{i=1}^n c_i K(x_i, x).$$

### 4.3 Kernel trick

In the above, we can replace  $x$  by  $h(x)$ , where  $h(x) = (h_k(x), k = 1, \dots, d)$  and the dimension  $d$  can be infinite. Then the model becomes

$$f(x) = \sum_{k=1}^d \beta_k h_k(x) = h(x)^\top \beta,$$

where  $h(x) = (h_k(x), k = 1, \dots, d)^\top$  is the hidden vector.

If we define the non-linear kernel

$$K(x, x') = \langle h(x), h(x') \rangle,$$

then all the calculations in the previous subsection go through. Specifically, the matrix becomes  $\mathbf{K} = (\mathbf{K}_{ij} = K(x_i, x_j))$ ,

$$c = (\mathbf{K} + \lambda I_n)^{-1} Y = (c_i, i = 1, \dots, n)^\top.$$

$$\hat{f}(x) = \sum_{i=1}^n c_i K(x_i, x).$$

In order to compute  $\hat{f}$ , we only need to know the kernel  $K$ . We do not need to know the feature function  $h(x)$ .

#### 4.4 Representer theorem

We do not need the magic and trick actually. Let us do it in a more principled way.

For a loss function

$$\sum_{i=1}^n L(y_i, h(x_i)^\top \beta) + \frac{1}{2} \lambda |\beta|^2.$$

The estimator that minimizes the loss function should be in the form of

$$\hat{\beta} = \sum_{i=1}^n c_i h(x_i).$$

That is,  $\hat{\beta}$  lies in the sub-space spanned by  $(h(x_i), i = 1, \dots, n)$ .

We can prove it by contradiction. Suppose the minimizer of the loss function  $\tilde{\beta}$  does not lie in the sub-space spanned by  $(h(x_i), i = 1, \dots, n)$ . Then we can write

$$\tilde{\beta} = \hat{\beta} + \Delta,$$

where

$$\langle \Delta, h(x_i) \rangle = 0, \forall i.$$

Then

$$h(x_i)^\top \tilde{\beta} = h(x_i)^\top (\hat{\beta} + \Delta) = h(x_i)^\top \hat{\beta}.$$

But

$$|\tilde{\beta}|^2 = |\hat{\beta}|^2 + |\Delta|^2.$$

Thus  $\tilde{\beta}$  has bigger loss than  $\hat{\beta}$ , unless  $\Delta = 0$ . This proves the representer theorem.

#### 4.5 Kernel regression

For the penalized least squares loss,

$$L(\beta) = \sum_{i=1}^n \left( y_i - h(x_i)^\top \beta \right)^2 + \lambda |\beta|^2,$$

applying the representer theorem by plugging in  $\beta = \sum_{j=1}^n c_j h(x_j)$ , we have

$$\begin{aligned} L &= \sum_{i=1}^n \left( y_i - h(x_i)^\top \beta \right)^2 + \lambda |\beta|^2 \\ &= \sum_{i=1}^n \left( y_i - h(x_i)^\top \sum_{j=1}^n c_j h(x_j) \right)^2 + \lambda \left| \sum_{i=1}^n c_i h(x_i) \right|^2 \\ &= \sum_{i=1}^n \left( y_i - \sum_{j=1}^n c_j K(x_i, x_j) \right)^2 + \lambda \sum_{i,j} c_i c_j K(x_i, x_j) \\ &= |Y - \mathbf{K}c|^2 + \lambda c^\top \mathbf{K}c. \end{aligned}$$

Taking derivative with respect to  $c$ , and setting it to zero,

$$-\mathbf{K}(Y - \mathbf{K}c) + \lambda \mathbf{K}c = 0,$$

then

$$c = (\mathbf{K} + \lambda I_n)^{-1} Y = (c_i, i = 1, \dots, n)^\top$$

and

$$\hat{f}(x) = \sum_{i=1}^n c_i K(x_i, x).$$

## 4.6 Reproducing kernel Hilbert space

Suppose we have two functions defined on the same set of features,  $f(x) = h(x)^\top \beta$ ,  $g(x) = h(x)^\top \gamma$ . We can define the inner product

$$\langle f, g \rangle_{\mathcal{F}} = \langle \beta, \gamma \rangle.$$

The norm then becomes

$$\|f\|_{\mathcal{F}}^2 = \langle f, f \rangle_{\mathcal{F}} = \langle \beta, \beta \rangle = \|\beta\|^2,$$

and  $\mathcal{F}$  consists of all the functions with finite norm.

The inner product defines a Hilbert space, which is a generalization of Euclidean space, where the notions such as orthogonality and orthogonal decomposition still hold.

The key property is the reproducing property:

$$\begin{aligned} \langle f, K(x', \cdot) \rangle_{\mathcal{F}} &= \langle h(x)^\top \beta, h(x)^\top h(x') \rangle_{\mathcal{F}} \\ &= h(x')^\top \beta = f(x'). \end{aligned}$$

## 4.7 Kernel version of representer theorem

We can define the loss function as

$$\sum_{i=1}^n L(y_i, f(x_i)) + \frac{\lambda}{2} \|f\|_{\mathcal{F}}^2.$$

This becomes non-parametric regression.

The minimizer of the loss function must be of the following form

$$\hat{f}(x) = \sum_{i=1}^n c_i K(x, x_i).$$

We proved this in the last subsection. Now we give a more direct proof. Suppose the minimizer does not lie in the subspace spanned by  $(k(\cdot, x_i), i = 1, \dots, n)$ , then we can write it as

$$\tilde{f}(x) = \sum_{i=1}^n c_i K(x, x_i) + \Delta(x),$$

with

$$\langle \Delta(x), K(x_i, x) \rangle_{\mathcal{F}} = 0.$$

Then on the observed  $x_i$ ,

$$\begin{aligned} \tilde{f}(x_i) &= \langle \tilde{f}(x), K(x, x_i) \rangle_{\mathcal{F}} \\ &= \langle \hat{f}(x) + \Delta(x), K(x, x_i) \rangle_{\mathcal{F}} \\ &= \hat{f}(x_i) + 0 = \hat{f}(x_i). \end{aligned}$$

In the above, we used the reproducing property  $\langle \hat{f}(x), K(x, x_i) \rangle = \hat{f}(x_i)$ . The two estimator have the same values on the observed data and therefore share the same value for the loss term  $\sum_{i=1}^n L(y_i, f(x_i))$ . However,

$$\begin{aligned} \|\tilde{f}\|_{\mathcal{F}}^2 &= \|\hat{f} + \Delta\|_{\mathcal{F}}^2 \\ &= \|\hat{f}\|_{\mathcal{F}}^2 + \|\Delta\|_{\mathcal{F}}^2 \\ &\geq \|\hat{f}\|_{\mathcal{F}}^2. \end{aligned}$$

Therefore, adding any  $\Delta(x)$  will increase our loss function and thus the best estimator should be  $\sum_{i=1}^n c_i K(x, x_i)$ .

With a reproducing kernel  $K$  and the associated norm, we can avoid dealing with  $h(x)$  and  $\beta$  explicitly.

## 4.8 Mercer theorem

For a kernel  $K(x, x')$ , we may consider it an infinite matrix with continuous index  $(x, x')$ . If it is positive definite, we will have the spectral decomposition

$$K(x, x') = \sum_{k=1}^{\infty} \lambda_k q_k(x) q_k(x').$$

Then we can let

$$h_k(x) = q_k(x) \lambda_k^{1/2}; \quad h(x) = (h_k(x), k = 1, \dots, \infty)^\top.$$

The point of Mercer theorem is that as long as the kernel  $K$  is positive definite, then there always exists the feature function  $h$ . But we do not need to know  $h$  explicitly.

A commonly used positive definite kernel is the Gaussian kernel or radial basis function

$$K(x, x') = \exp(-\gamma |x - x'|^2).$$

It should not be confused with the Gaussian distribution. The corresponding  $h_k(x)$  are the Fourier basis, such as those in the Ptolemy epicycle model.

## 4.9 Nearest neighbors interpolation

The kernel model computes the score by

$$s = h(x)^\top \beta = \sum_{i=1}^n c_i K(x_i, x),$$

where  $h(x)$  and  $\beta$  are implicit but useful for theoretical understanding, while  $c_i$  and  $K$  are explicit and are used in computing. The model is an interpolation of the observed examples. For each  $x$ , we can find nearby  $x_i$ , and then average their  $y_i$  values. The kernel model essentially does this in an optimal way. In this sense, the model does not provide much understanding of the data.

## 5 Gaussian Process

The Bayesian interpretation of linear regression assumes a prior distribution on the coefficient vector  $\beta$ . With  $f(x) = h(x)^\top \beta$ ,  $f(x)$  becomes a random function or a stochastic process over the domain of  $x$ . With a Gaussian prior distribution for  $\beta$ ,  $f(x)$  becomes a Gaussian process.

Similar to the kernel version of representer theorem, we can work with the kernel  $K(x, x') = \langle h(x), h(x') \rangle$  directly, where  $K(x, x')$  takes the form of covariance matrix. We do not need to deal with  $h(x)$  and  $\beta$  explicitly, although they help us understand the model.

### Background: multivariate normal

#### Joint distribution

We start from  $Z = (z_1, \dots, z_n)^\top$ , where  $z_i \sim N(0, 1)$  independently. Then  $\mathbb{E}(Z) = 0$ , and  $\text{Var}(Z) = I$ . We denote  $Z \sim N(0, I)$ . The density of  $Z$  is

$$\begin{aligned} f_Z(Z) &= \frac{1}{(2\pi)^{n/2}} \exp \left[ -\frac{1}{2} \sum_i z_i^2 \right] \\ &= \frac{1}{(2\pi)^{n/2}} \exp \left[ -\frac{1}{2} Z^\top Z \right]. \end{aligned}$$

Let  $X = \mu + \Sigma^{1/2}Z$ , then  $Z = \Sigma^{-1/2}(X - \mu)$ , which is a matrix version of standardization. Then

$$\begin{aligned} f_Y(Y) &= \frac{1}{(2\pi)^{n/2}} \exp \left[ -\frac{1}{2}(X - \mu)^\top \Sigma^{-1}(X - \mu) \right] / |\Sigma^{1/2}| \\ &= \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp \left[ -\frac{1}{2}(X - \mu)^\top \Sigma^{-1}(X - \mu) \right]. \end{aligned}$$

Moreover, because  $X = \mu + \Sigma^{1/2}Z$ , we have  $\mathbb{E}(X) = \mu$  and  $\text{Var}(X) = \Sigma$ . We denote  $X \sim N(\mu, \Sigma)$ .

In general, if  $X \sim N(\mu, \Sigma)$ , and  $Y = AX$ , then  $Y \sim N(A\mu, A\Sigma A^\top)$ .  $A$  does not need to be a square matrix.

### Conditional distribution

Assuming  $\mathbb{E}(X) = 0$  (otherwise we can always let  $X \leftarrow X - \mathbb{E}(X)$ ). Let us partition  $X$  into  $(X_1, X_2)$ .

$$\begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \sim N \left( \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix} \right).$$

Let  $\varepsilon = X_2 - AX_1$ . We choose  $A$  to make  $\text{Cov}(\varepsilon, X_1) = 0$ ,

$$\begin{aligned} \text{Cov}(\varepsilon, X_1) &= \text{Cov}(X_2 - AX_1, X_1) \\ &= \text{Cov}(X_2, X_1) - A\text{Cov}(X_1, X_1) \\ &= \Sigma_{21} - A\Sigma_{11} = 0, \end{aligned}$$

so  $A = \Sigma_{21}\Sigma_{11}^{-1}$ , and  $X_2 = AX_1 + \varepsilon$ . This can be considered a regression of  $X_2$  on  $X_1$ . The residual variance is

$$\begin{aligned} \text{Var}(\varepsilon) &= \text{Cov}(\varepsilon, \varepsilon) \\ &= \text{Cov}(X_2 - AX_1, \varepsilon) \\ &= \text{Cov}(X_2, \varepsilon) \\ &= \text{Cov}(X_2, X_2 - AX_1) \\ &= \text{Cov}(X_2, X_2) - \text{Cov}(X_2, AX_1) \\ &= \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}. \end{aligned}$$

Thus

$$\begin{pmatrix} X_1 \\ \varepsilon \end{pmatrix} = \begin{pmatrix} I_1 & 0 \\ -A & I_2 \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \sim N \left( \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \Sigma_{11} & 0 \\ 0 & \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12} \end{pmatrix} \right).$$

So the marginal distribution

$$X_1 \sim N(0, \Sigma_{11}),$$

the mean and the variance of the conditional distribution  $P(X_2|X_1)$  are

$$\begin{aligned} \mathbb{E}(X_2|X_1) &= A\mathbb{E}(X_1) + \mathbb{E}(\varepsilon) \\ &= \Sigma_{21}\Sigma_{11}^{-1}X_1, \end{aligned}$$

$$\begin{aligned} \text{Var}(X_2|X_1) &= \mathbb{E}((X_2 - \mathbb{E}(X_2|X_1))^2|X_1) \\ &= \mathbb{E}(\varepsilon^2|X_1) \\ &= \mathbb{E}(\varepsilon^2) - \mathbb{E}(\varepsilon)^2 \\ &= \text{Var}(\varepsilon), \end{aligned}$$

thereby the conditional distribution

$$[X_2|X_1] \sim N(\Sigma_{21}\Sigma_{11}^{-1}X_1, \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}),$$

which is a multivariate linear regression. The geometric intuition is least squares projection.

## 5.1 Bayesian interpretation of ridge regression

Suppose  $Y = X\beta + \varepsilon$ , where  $\beta \sim N(0, \tau^2 I_p)$ ,  $\varepsilon \sim N(0, \sigma^2 I_n)$ , and  $\varepsilon$  is independent of  $\beta$ . The joint distribution is

$$\begin{bmatrix} Y \\ \beta \end{bmatrix} \sim N \left( \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \tau^2 X X^\top + \sigma^2 I_n & \tau^2 X^\top \\ \tau^2 X^\top & \tau^2 I_p \end{bmatrix} \right).$$

The posterior of  $\beta$  is

$$[\beta | Y, X] = N(\tau^2 X^\top (\tau^2 X X^\top + \sigma^2 I_n)^{-1} Y, \tau^2 I_p - \tau^2 X^\top (\tau^2 X X^\top + \sigma^2 I_n)^{-1} \tau^2 X)$$

The posterior mean is the same as the ridge regression. In fact, we can write

$$\begin{aligned} p(\beta | Y, X) &\propto p(\beta) p(Y | X, \beta) \\ &\propto \exp \left( -\frac{1}{2\tau^2} |\beta|^2 \right) \exp \left( -\frac{1}{2\sigma^2} |Y - X\beta|^2 \right) \\ &= \exp \left( -\frac{1}{2} \left[ \frac{1}{\sigma^2} |Y - X\beta|^2 + \frac{1}{\tau^2} |\beta|^2 \right] \right), \end{aligned}$$

which clearly shares the form of ridge regression.

It is tempting to apply the kernel trick by promoting  $x$  to  $h(x)$ , and replacing  $\langle x, x' \rangle$  by  $\langle h(x), h(x') \rangle = K(x, x')$ . But there is a more direct way where we do not need to deal with  $h(x)$  and  $\beta$  explicitly, and there is no need to explicitly integrating out  $\beta$ .

## 5.2 Kernel as covariance

Assuming  $f(x) = h(x)^\top \beta$ , and  $\beta \sim N(0, \tau^2 I_d)$ ,  $f(x)$  is a stochastic process in the domain of  $x$ , and for any pair of points  $(x, x')$  in the domain,

$$\begin{aligned} \text{Cov}(f(x), f(x')) &= \text{Cov}(h(x)^\top \beta, h(x')^\top \beta) \\ &= \mathbb{E}[h(x)^\top \beta \beta^\top h(x')] \\ &= \tau^2 h(x)^\top h(x') = K(x, x'). \end{aligned}$$

Thus the kernel becomes the covariance matrix.

We may use Gaussian kernel, but it has nothing to do with the word ‘‘Gaussian’’ in the Gaussian process, which comes from the fact that for any  $(x_1, \dots, x_n)$ ,  $(f(x_1), \dots, f(x_n))$  follows a multivariate Gaussian distribution since  $\beta$  follows a Gaussian prior distribution.

## 5.3 Prediction by conditioning

For the training data, we can write:

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \dots \\ f(x_n) \end{bmatrix} + \varepsilon,$$

where  $\varepsilon \sim N(0, \sigma^2 I_n)$ .

Marginally, we have

$$Y \sim N(0, \mathbf{K} + \sigma^2 I_n),$$

where  $\mathbf{K}_{ij} = K(x_i, x_j)$  is an  $n \times n$  matrix. For a testing example  $x_0$ , we have  $\mathbb{E}[f(x_0)] = 0$ ,  $\text{Var}[f(x_0)] = K(x_0, x_0)$ , and  $\text{Cov}(Y, f(x_0)) = K(X, x_0)$ , which is an  $n \times 1$  vector, whose  $i$ -th element is  $K(x_i, x_0)$ . The joint distribution is

$$\begin{bmatrix} Y \\ f(x_0) \end{bmatrix} = \mathbf{N} \left( \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{K} + \sigma^2 I_n & K(X, x_0) \\ K(x_0, X) & K(x_0, x_0) \end{bmatrix}_{(n+1) \times (n+1)} \right).$$

The prediction of  $f(x_0)$  is then based on

$$[f(x_0)|Y, X] \sim \mathbf{N}(K(x_0, X)^\top (\mathbf{K} + \sigma^2 I_n)^{-1} Y, K(x_0, x_0) - K(x_0, X)(\mathbf{K} + \sigma^2 I_n)^{-1} K(x_0, X)^\top),$$

where the posterior mean

$$\hat{f}(x_0) = K(x_0, X)^\top (\mathbf{K} + \sigma^2 I_n)^{-1} Y,$$

which is the same as kernel regression.

## 5.4 Posterior interval

In addition to finding the point estimate as in kernel regression, we also obtain the uncertainty of the estimate with

$$V = K(x_0, x_0) - K(x_0, X)(\mathbf{K} + \sigma^2 I_n)^{-1} K(x_0, X)^\top.$$

We can construct a  $(1 - \alpha)$  posterior interval

$$\hat{f}(x_0) \pm z_{\alpha/2} V^{1/2}.$$

## 5.5 Marginal likelihood for hyper-parameter

The marginal distribution of  $Y \sim \mathbf{N}(0, \mathbf{K}_\gamma + \sigma^2 I_n)$ , where  $\mathbf{K} = (K_{ij} = K(x_i, x_j))$ . There are several hyper-parameters we need to choose for GP. To name a few, we have  $\tau^2$  in the kernel function  $K(x, x') = \tau^2 h(x)^\top h(x')$ ,  $\sigma^2$  in  $P(Y)$  and  $\gamma$ , which is the parameter of the Gaussian kernel  $K(x, x') = \tau^2 \exp(-\gamma|x - x'|^2)$ .

The marginal likelihood of these hyper-parameters is

$$L(\tau^2, \sigma^2, \gamma) = \frac{1}{(2\pi)^{n/2} |\Sigma_{\tau^2, \sigma^2, \gamma}|^{1/2}} \exp \left( -\frac{1}{2} Y^\top \Sigma_{\tau^2, \sigma^2, \gamma}^{-1} Y \right),$$

where  $\Sigma_{\tau^2, \sigma^2, \gamma} = \mathbf{K}_{\tau^2, \gamma} + \sigma^2 I_n$ .

The log-marginal-likelihood for determining  $\tau^2$ ,  $\sigma^2$  and  $\gamma$  is then

$$l(\tau^2, \sigma^2, \gamma) = -\frac{1}{2} Y^\top \Sigma_{\tau^2, \sigma^2, \gamma}^{-1} Y - \frac{1}{2} \log |\Sigma_{\tau^2, \sigma^2, \gamma}|.$$

It is simpler than cross-validation for choosing  $\tau^2$ ,  $\sigma^2$  and  $\gamma$ .

## 6 Kernel SVM

We now study kernel classification, which parallels the kernel regression. Again we begin with the linear classifier and apply the kernel trick, where we use the hinge function as the loss function. Then we explain the original idea of support vector machine as the max margin classifier. Both the hinge and margin formulations involve rewriting the primal problem as min-max, and then changing it to the max-min problem which is the dual problem.



## 6.1 Hinge loss and linear SVM

For classification where  $y_i \in \{+1, -1\}$ , we can classify  $y_i$  by the perceptron model:

$$\hat{y}_i = \text{sign}(x_i^\top \beta).$$

To estimate  $\beta$ , we can modify the loss function of ridge regression  $\mathcal{L}(\beta) = |Y - X\beta|^2 + \lambda|\beta|^2$  to

$$\mathcal{L}(\beta) = \sum_{i=1}^n \max(0, 1 - y_i x_i^\top \beta) + \frac{\lambda}{2} |\beta|^2.$$

This is the objective function of linear SVM.

## 6.2 Re-representing hinge loss

For the hinge loss  $\max(0, 1 - m)$ , it is non-linear in  $m$ , we can represent it as

$$\max(0, 1 - m) = \max_{\alpha \in [0, 1]} \alpha(1 - m),$$

which is linear in  $m$ .

## 6.3 Primal min-max to dual max-min

The optimization problem becomes

$$\hat{\beta} \leftarrow \min_{\beta} \sum_{i=1}^n \max_{\alpha_i \in [0, 1]} \alpha_i(1 - y_i x_i^\top \beta) + \frac{\lambda}{2} |\beta|^2.$$

We can change min max to max min, so that we change the primal problem into the dual problem. For convenience, let  $C = 1/\lambda$  and  $\alpha_i \leftarrow C\alpha_i$ :

$$\begin{aligned} \hat{\beta} &\leftarrow \max_{\alpha_i \in [0, C] \forall i} \min_{\beta} \sum_{i=1}^n \alpha_i(1 - y_i x_i^\top \beta) + \frac{1}{2} |\beta|^2 \\ &= \max_{\alpha_i \in [0, C] \forall i} \min_{\beta} \left[ \frac{1}{2} \left| \beta - \sum_{i=1}^n \alpha_i y_i x_i \right|^2 - \frac{1}{2} \left| \sum_{i=1}^n \alpha_i y_i x_i \right|^2 + \sum_{i=1}^n \alpha_i \right]. \end{aligned}$$

The minimization gives us the representer

$$\hat{\beta} = \sum_{i=1}^n \alpha_i y_i x_i.$$

The dual problem is  $\max_{\alpha_i \in [0, C], \forall i} q(\alpha)$ ,

$$q(\alpha) = \left[ \sum_{i=1}^n \alpha_i - \frac{1}{2} \left| \sum_{i=1}^n \alpha_i y_i x_i \right|^2 \right] = \frac{1}{2} \alpha^\top Q \alpha - \alpha^\top \mathbf{1},$$

where  $\mathbf{1}$  is a vector of 1's.

## 6.4 Dual coordinate descent

The dual problem can be solved by stochastic coordinate descent. At each time step  $t$ , we randomly pick an index  $i$  and compute the optimal one-variable update:

$$\delta^* = \arg \max_{\delta: 0 \leq \alpha_i + \delta \leq C} q(\alpha + \delta e_i),$$

where  $e_i$  is the one-hot vector where the  $i$ -th element is 1.

$$\begin{aligned} q(\alpha + \delta e_i) &= \frac{1}{2}(\alpha + \delta e_i)^\top Q(\alpha + \delta e_i) - (\alpha + \delta e_i)^\top \mathbf{1} \\ &= \frac{Q_{ii}}{2} \delta^2 + \alpha^\top Q_i \delta - \delta, \end{aligned}$$

where  $Q_i$  is the  $i$ -th column of  $Q$ , and  $Q_{ii}$  is the  $i$ -th diagonal element of  $Q$ . By differentiating the above with respect to  $\delta$  and setting it to be zero, we have

$$\delta = \frac{1 - \alpha^\top Q_i}{Q_{ii}}.$$

However, we require  $0 \leq \alpha_i + \delta \leq C$ , so the optimal solution is

$$\delta^* = \max \left( -\alpha_i, \min \left( C - \alpha_i, \frac{1 - \alpha^\top Q_i}{Q_{ii}} \right) \right).$$

## 6.5 Kernel trick

For kernel SVM, we assume feature vector  $h(x)$ , and classify  $y_i$  by

$$\hat{y}_i = \text{sign}(h(x_i)^\top \beta).$$

The loss function is

$$\mathcal{L}(\beta) = \sum_{i=1}^n \max(0, 1 - y_i h(x_i)^\top \beta) + \frac{\lambda}{2} |\beta|^2.$$

The representer is

$$\hat{\beta} = \sum_{i=1}^n \alpha_i y_i h(x_i).$$

The dual problem is  $\max_{\alpha_i \in [0, C], \forall i} q(\alpha)$ ,

$$\begin{aligned} q(\alpha) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j \langle h(x_i), h(x_j) \rangle \\ &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j K(x_i, x_j). \end{aligned}$$

Again the problem can be solved by dual coordinate ascent.

After solving  $\alpha_i$ , we get  $\hat{\beta}$  from the representer, and the estimated function is

$$\begin{aligned} \hat{f}(x) &= h(x)^\top \hat{\beta} \\ &= \sum_{i=1}^n \alpha_i y_i \langle h(x_i), h(x) \rangle \\ &= \sum_{i=1}^n \alpha_i y_i K(x_i, x). \end{aligned}$$

## 6.6 Support vectors and nearest neighbors template matching

For those examples with  $\alpha_i > 0$ , they are called the support vectors, and they determine the classifier, which can be interpreted as a sophisticated version of nearest neighbor template matching. Again we only need to know the kernel function without knowing the feature function  $h$ .

## 6.7 Max margin

Consider the perceptron  $y_i = \text{sign}(x_i^\top \beta)$ , which separates the positive examples and negative examples by projecting the data on vector  $\beta$ , or by a hyperplane that is perpendicular to  $\beta$ . If the positive examples and negative examples are separable, there can be many separating hyperplanes. We want to choose the one with the maximum margin in order to guard against the random fluctuations in the unseen testing examples.

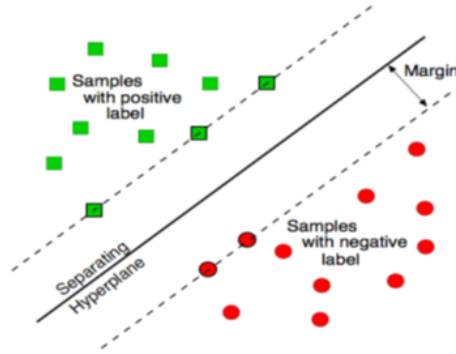


Figure 11: Max margin. We want to separate the positive examples and negative examples by a hyperplane that has the maximum margin.

The idea of support vector machines (SVM) is to find the  $\beta$  so that,

(1) for positive examples  $y_i = +$ ,  $x_i^\top \beta \geq 1$  and,

(2) for negative examples  $y_i = -$ ,  $x_i^\top \beta \leq -1$ .

Here we use +1 and -1, because we can always scale  $\beta$ .

The decision boundary is decided by the training examples that lie on the margin. Those are the support vectors. Let  $u$  be a unit vector that has the same direction as  $\beta$ ,  $u = \frac{\beta}{|\beta|}$ . Suppose  $x_i$  is an example on the margin (i.e., support vector), the projection of  $x_i$  on  $u$  is

$$\langle x_i, u \rangle = \langle x_i, \frac{\beta}{|\beta|} \rangle = \frac{\langle \beta, x_i \rangle}{|\beta|} = \frac{\pm 1}{|\beta|}.$$

So the margin is  $1/|\beta|$ . In order to maximize the margin, we should minimize  $|\beta|$  or  $|\beta|^2$ . Hence, the SVM can be formulated as an optimization problem as follows:

$$\begin{aligned} & \text{minimize} && \frac{1}{2} |\beta|^2, \\ & \text{subject to} && y_i x_i^\top \beta \geq 1, \forall i. \end{aligned}$$

Recall  $x_i^\top \beta$  is the score, and  $y_i x_i^\top \beta$  is the individual margin of observation  $i$ . This is the primal form of SVM.

## 6.8 Primal dual

### Primal as min-max

We first translate the constrained form of the primal into an unconstrained form with Lagrangian

$$L(\beta, \alpha) = \frac{1}{2} |\beta|^2 + \sum_{i=1}^n \alpha_i (1 - y_i x_i^\top \beta),$$

where  $\alpha_i \geq 0$ , and  $\alpha = (\alpha_i, i = 1, \dots, n)$ . This rewrite results in

$$\min_{\beta} \max_{\alpha \geq 0} L(\beta, \alpha).$$

Note that in this unconstrained form, the constraints are automatically satisfied. Otherwise, for the optimal  $\beta$ , if there exists an  $i$  such that  $1 - y_i x_i^\top \beta > 0$ , then we can let  $\alpha_i \rightarrow \infty$  in maximization, which will push  $L(\beta, \alpha) \rightarrow \infty$ . Apparently, such a  $\beta$  cannot be a minimizer. This contradiction proves the statement that constraints are automatically satisfied.

### Dual as max-min

The dual problem is  $\max_{\alpha} \min_{\beta} L(\beta, \alpha)$ . This is analytically more friendly, since for fixed  $\alpha$  and  $\min_{\beta} L$ , we have

$$\begin{aligned} L(\beta, \alpha) &= \frac{1}{2} |\beta|^2 + \sum_{i=1}^n \alpha_i - \left\langle \beta, \sum_{i=1}^n \alpha_i y_i x_i \right\rangle \\ &= \frac{1}{2} \left| \beta - \sum_{i=1}^n \alpha_i y_i x_i \right|^2 - \frac{1}{2} \left| \sum_{i=1}^n \alpha_i y_i x_i \right|^2 + \sum_{i=1}^n \alpha_i, \end{aligned}$$

with the representer

$$\hat{\beta} = \sum_{i=1}^n \alpha_i y_i x_i.$$

That is,  $\beta$  is along the direction from the negative support vectors to the positive support vectors. Substitute  $\hat{\beta}$  back, we have

$$\begin{aligned} q(\alpha) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \left| \sum_{i=1}^n \alpha_i y_i x_i \right|^2 \\ &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle. \end{aligned}$$

$q(\alpha)$  can be maximized by dual coordinate ascent.

The complementary slackness is such that

$$\alpha_i (1 - y_i x_i^\top \beta) = 0$$

for  $i = 1, \dots, n$ . If  $y_i x_i^\top \beta > 1$ , so that  $x_i$  is well classified, then  $\alpha_i = 0$ . If  $\alpha_i > 0$ , then  $y_i x_i^\top \beta = 1$ , i.e.,  $x_i$  is on the boundary and is a support vector. The learned  $\beta = \sum_{i=1}^n \alpha_i y_i x_i$  is decided by the support vectors  $i$  whose  $\alpha_i > 0$ .

## 6.9 Kernel trick

We can play the kernel trick. For each  $x_i$ , we transform it to a high dimensional (or infinite dimensional) feature vector  $h(x_i)$ , and we use  $h(x_i)$  instead of  $x_i$  for classification. Then we need to replace  $\langle x_i, x_j \rangle$  by  $\langle h(x_i), h(x_j) \rangle$ . The kernel trick is such that we do not need to specify  $h()$  explicitly. We only need to define the kernel function

$$K(x_i, x_j) = \langle h(x_i), h(x_j) \rangle.$$

One commonly used kernel function is the Gaussian kernel,

$$K(x, x') = \exp(-\gamma |x - x'|^2)$$

which measures the similarity between  $x$  and  $x'$ . With such a kernel, we can continue to minimize  $q(\alpha)$ . After finding the optimal  $\alpha$ , we have

$$\beta = \sum_{i=1}^n \alpha_i y_i h(x_i).$$

For a testing data  $x$ , we classify it by

$$\begin{aligned} \hat{y} &= \text{sign}(h(x)^\top \beta) \\ &= \text{sign}\left(\sum_{i=1}^n \alpha_i y_i \langle h(x_i), h(x) \rangle\right) \\ &= \text{sign}\left(\sum_{i=1}^n \alpha_i y_i K(x_i, x)\right). \end{aligned}$$

In the above classifier,  $\alpha_i$  are non-zero only for the support vectors. The classifier compares the testing example  $x$  with the support vectors, which serve as exemplars. If  $x$  is more similar to the positive exemplars, we will classify  $x$  to be positive. Otherwise we classify  $x$  to be negative.

## 6.10 Non-separable case and slack variables

The primal form of SVM for the non-separable case is

$$\begin{aligned} &\text{minimize } \frac{1}{2} |\beta|^2 + C \sum_{i=1}^n \xi_i, \\ &\text{subject to } y_i x_i^\top \beta \geq 1 - \xi_i, \text{ and } \xi_i \geq 0, \forall i, \end{aligned}$$

where  $\xi_i$  are slack variables to accommodate the non-separable examples. We want the number of non-zero  $\xi_i$  to be as small as possible, i.e., we penalize  $\sum_i \xi_i$ .

The dual problem in this case is similar to the separable case. We only need to replace the constraint  $\alpha_i \geq 0$  by the box constraint  $\alpha_i \in [0, C], \forall i$ . Specifically, we first translate the constrained primal to unconstrained form, with Lagrangian

$$L(\beta, \alpha, \mu) = \frac{1}{2} |\beta|^2 + C \sum_{i=1}^n \xi_i + \sum_{i=1}^n \alpha_i (1 - \xi_i - y_i x_i^\top \beta) + \sum_{i=1}^n \mu_i (1 - \xi_i),$$

where  $\alpha_i \geq 0$ , and  $\alpha = (\alpha_i, i = 1, \dots, n)$ ,  $\mu_i \geq 0$ , and  $\mu = (\mu_i, i = 1, \dots, n)$ .

Then we have the primal dual transformation:

$$\min_{\beta, \xi_i} \max_{\alpha_i \geq 0, \mu_i \geq 0, \forall i} L(\beta, \alpha, \mu) \rightarrow \max_{\alpha_i \geq 0, \mu_i \geq 0, \forall i} \min_{\beta, \xi_i} L(\beta, \alpha, \mu).$$

That is, we have a dual form

$$\max_{\alpha_i \geq 0, \mu_i \geq 0, \forall i} \min_{\beta, \xi_i} L(\beta, \alpha, \mu) = \frac{1}{2} |\beta|^2 + \sum_{i=1}^n \alpha_i (1 - y_i x_i^\top \beta) + \sum_{i=1}^n (C - \mu_i - \alpha_i) \xi_i.$$

It is easy to see that the term  $\frac{1}{2} |\beta|^2 + \sum_{i=1}^n \alpha_i (1 - y_i x_i^\top \beta)$  is the same as the one in the separable case. For the extra term  $\sum_{i=1}^n (C - \mu_i - \alpha_i) \xi_i$ , it has to be zero to reach the equilibrium. Thus we have

$$0 \leq \alpha_i = C - \mu_i \leq C.$$

That is, for the non-separable case, we solve the same dual form as the separable case with an extra constraint  $\alpha_i \in [0, C] \forall i$ .

### 6.11 Primal form with hinge loss

In order to make  $\xi_i$  as small as possible, if  $y_i x_i^\top \beta \geq 1$ , we can take  $\xi_i = 0$ . Otherwise, we can let  $\xi_i = 1 - y_i x_i^\top \beta$ . Combine the two conditions, the optimal  $\xi_i$  is

$$\xi_i = \max(0, 1 - y_i x_i^\top \beta).$$

Then the primal problem becomes an unconstrained minimization of the loss function

$$\frac{1}{2} |\beta|^2 + C \max(0, 1 - y_i x_i^\top \beta),$$

or let  $C = 1/\lambda$ , the minimization becomes

$$\max(0, 1 - y_i x_i^\top \beta) + \frac{\lambda}{2} |\beta|^2.$$

This is similar to ridge regression with  $\ell_2$  regularization. In general, we can use loss function

$$\sum_{i=1}^n L(y_i x_i^\top \beta) + \frac{\lambda}{2} |\beta|^2,$$

where  $L()$  can be hinge loss or logistic loss.

### 6.12 Bias and sequential minimal optimization (SMO)

With a bias term, SVM solves the optimization problem,

$$\min_{(w, b)} \sum_{i=1}^n \max_{\alpha_i \in [0, C]} \alpha_i (1 - y_i (w^\top x_i + b)) + \frac{1}{2} |w|^2,$$

which is equivalent to

$$\max_{\alpha_i \in [0, C]} \min_{(w, b)} \frac{1}{2} \left| w - \sum_{i=1}^n y_i \alpha_i x_i \right|^2 + \sum_{i=1}^n \alpha_i + \sum_{i=1}^n y_i \alpha_i b - \frac{1}{2} \left| \sum_{i=1}^n \alpha_i y_i x_i \right|^2.$$

The constraint  $\sum_{i=1}^n y_i \alpha_i = 0$  must be satisfied. Otherwise if  $\sum_{i=1}^n y_i \alpha_i > 0$ , then we can let  $b \rightarrow -\infty$ , and if  $\sum_{i=1}^n y_i \alpha_i < 0$ , then  $b \rightarrow \infty$ , so that we will not have minimum. Thus our task is reduced to solving the dual problem,

$$\max_{\alpha_i \in [0, C]} \sum_{i=1}^n \alpha_i - \frac{1}{2} \alpha^\top Q \alpha,$$

subject to  $\sum_{i=1}^n y_i \alpha_i = 0$ . Instead of coordinate ascent, we need to update  $\alpha_i$  in pairs to make sure the constraint is satisfied. This is usually implemented by sequential minimal optimization (SMO).

### 6.13 Max margin = min distance

The primal form of SVM is max margin, and the dual form of SVM is min distance.

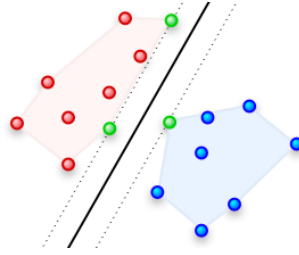


Figure 12: max margin = min distance. In order to find the max marginal separating hyperplane, we can find the minimum distance between the convex hulls of positive and negative examples. The separating hyperplane is in the middle of the minimum distance.

Consider the convex hulls of the positive and negative examples. The margin between the two sets is defined by the minimum distance between two. Let  $X_+ = \sum_{i \in +} c_i x_i$  and  $X_- = \sum_{i \in -} c_i x_i$  ( $c_i \geq 0, \sum_{i \in +} c_i = 1, \sum_{i \in -} c_i = 1$ ) be two points in the positive and negative convex hulls. The margin is  $\min |X_+ - X_-|^2$ .

$$\begin{aligned}
 |X_+ - X_-|^2 &= \left| \sum_{i \in +} c_i x_i - \sum_{i \in -} c_i x_i \right|^2 \\
 &= \left| \sum_i y_i c_i x_i \right|^2 \\
 &= \sum_{i,j} c_i c_j y_i y_j \langle x_i, x_j \rangle, \\
 \text{subject to } &c_i \geq 0, \sum_{i \in +} c_i = 1, \sum_{i \in -} c_i = 1.
 \end{aligned}$$

This problem is essentially the same as the dual problem. After we solve for  $c_i$ , the non-zeros  $c_i$ 's are support vectors, i.e., examples on the boundary.

## 7 Boosting Machines

In trees and boosting machines,  $f(x) = h(x)^\top \beta = \sum_{m=1}^d h_m(x) \beta_m$ , where  $h_m(x)$  are base functions or base learners. In trees,  $h_m(x)$  are indicator functions of rectangle regions. In boosting machines,  $h_m(x)$  are themselves shallow trees. We learn  $h_m(x)$  by greedy search or forward selection.

### 7.1 Classification and regression trees (CART)

Figure 13 illustrates the basic idea of classification and regression tree (CART). The figure on the left shows a simple example of a decision tree, to decide whether a person is male or female based on the height and weight of the person. The classification and regression tree is to learn such a decision rule from the data. The figure on the right illustrates the process of learning, by recursively partitioning the data space. Specifically, for the whole data set, we go through all the variables,  $j = 1, \dots, p$ , and for each variable, we go through all the observed values  $(x_{ij}, i = 1, \dots, n)$ . For each  $t = x_{ij}$ , we consider dividing the data space into two parts based on whether  $x_j \geq t$  or  $x_j < t$ . We choose the best  $j$  and  $t$  according to a certain criterion. Thus we split

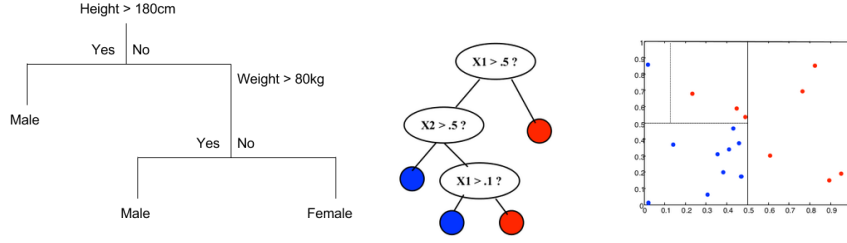


Figure 13: Decision tree. Left: an example of a decision tree that classifies a person as male or female based on height and weight. Right: the process of learning CART is to recursively partitioning each rectangle region into two sub-regions. Each partitioning should increase the purity of the  $y_i$  within each region.

the whole dataset into two subsets. For each subset of data, we repeat what we did for the whole dataset. In this way, we obtain a recursive binary partition of the data space, which corresponds to a binary decision tree. We can design a criterion to determine when to stop this process.

First, what is the criterion to choose  $j$  and  $t$  for binary division? We want to divide the whole region into two regions, so that examples within the same region tend to share the same responses. In other words, the examples within each region display strong purity.

Formally, the process of recursive partitioning eventually divides the whole region into  $M$  sub-regions  $R_m, m = 1, \dots, M$ . For a generic example  $(x, y)$ , we model

$$f(x) = \sum_{m=1}^M c_m \mathbf{1}(x \in R_m),$$

where  $\mathbf{1}()$  is an indicator function, which is 1 if  $x \in R_m$ , and 0 otherwise. This is a piecewise constant function. Suppose  $y_i$  is continuous, and we want to predict  $y$  by  $f(x)$ . We estimate  $f(x)$  by minimizing the least squares loss function

$$\mathcal{L} = \sum_{i=1}^n (y_i - f(x_i))^2.$$

Consider the first step that partitions the whole region into two sub-regions  $R_1$  and  $R_2$ . Let  $\hat{c}_1$  be the average of  $y_i$  with  $x_i \in R_1$ , and let  $\hat{c}_2$  be the average of  $y_i$  with  $x_i \in R_2$ . Then the loss function is

$$\mathcal{L} = \sum_{x_i \in R_1} (y_i - \hat{c}_1)^2 + \sum_{x_i \in R_2} (y_i - \hat{c}_2)^2,$$

which may be considered a measure of purity in the case of regression. We want to find  $j$  and  $t$  to minimize the above  $\mathcal{L}$ . We can repeat this procedure recursively.

To be more concrete, suppose we would like to choose the  $m$ -th region  $R_m = \{x | x_j \leq t\}$  with cutoff position  $x_j$  and cutoff value  $t \in [\min_i x_{ij}, \max_i x_{ij}]$ , we opt to find  $j$  and  $t$  to minimize the risk below

$$x_j, t = \arg \min_{x_j, t} \sum_{i=1}^n (r_i - c_m \mathbf{1}(x \in R_m))^2,$$

where

$$\begin{aligned} r_i &= y_i - \sum_{m'=1}^{m-1} c_{m'} \mathbf{1}(x \in R_{m'}), \\ c_m &= \frac{\sum_{i=1}^n r_i \mathbf{1}(x_i \in R_m)}{\sum_{i=1}^n \mathbf{1}(x_i \in R_m)}, \end{aligned}$$



are the estimation (mean as the best estimator) of  $c_m$  and the  $i$ -th residual of the previous model with  $m - 1$  regions respectively.

As to when to stop, we may consider penalizing the complexity of the tree, e.g., we use the following regularized loss function

$$\mathcal{L} = \sum_{i=1}^n (y_i - f(x_i))^2 + \lambda M,$$

where  $\lambda > 0$  is the cost for adding one region. Then we can stop the partitioning process if the reduction in the loss cannot cover the cost of adding a region.

We may also include some regularization on the partitions. We can further introduce weights for each datapoint in the training set. Then the objective becomes

$$\mathcal{L} = \sum_{i=1}^n w_i (y_i - f(x_i))^2 + \gamma \sum_{m=1}^M |c_m|^2 + \lambda M,$$

and the criterion for selecting partitioning parameters  $j$  and  $t$  also changes accordingly

$$x_{j,t} = \arg \min_{x_{j,t}} \sum_{i=1}^n w_i (r_i - c_m \mathbf{1}(x \in R_m))^2 + \gamma |c_m|^2 + \lambda M,$$

where  $r_i$  is identical to the case without data weights. The estimation to  $c_m$  can still be obtained by taking derivative of  $\mathcal{L}$ , which turns out to be

$$c_m = \frac{\sum_{i=1}^n w_i r_i \mathbf{1}(x_i \in R_m)}{\gamma + \sum_{i=1}^n w_i \mathbf{1}(x_i \in R_m)}.$$

The stop condition is still based on whether  $\mathcal{L}$  decreases or not.

Finally, we consider the binary classification problem. We can simply replace the least square loss in the regression problem with logistic regression loss, then we can directly apply such a tree growing method to it.

For a more generic classification setting, suppose there are  $K$  categories, ideally, we hope that within each region, all the examples belong to the same category. More realistically, let  $\hat{p}_k$  be the proportion of these examples that belong to category  $k$ . Let  $\hat{k}$  be the category that has the most examples, i.e.,  $\hat{k} = \arg \max_k \hat{p}_k$ . We may define the purity in terms of  $\hat{p}_{\hat{k}} = \max_k \hat{p}_k$ . If we stop splitting this region, we will classify the examples in this region to category  $\hat{k}$ . There are other definitions of purity, such as Gini index, which is

$$G = \sum_k \hat{p}_k (1 - \hat{p}_k).$$

## 7.2 Random forest

The idea of random forest is simpler than Adaboost. For the dataset  $\{(x_i, y_i), i = 1, \dots, n\}$ , we create a new dataset by randomly sampling the examples with replacement. For this new dataset, we grow a tree. See Figure 14. For each split, instead of going over all the  $p$  variables  $j$ , we randomly sample  $\sqrt{p}$  variables, and let  $j$  run through this subset of variables to decide on which variable we split. We then repeat this procedure a large number of times to grow a large number of trees, i.e., a forest. In the testing stage, for each input  $x$ , each tree produces a predicted  $y$ . We then take the average of the predicted values in the case of regression, or take the majority vote by the trees in the case of classification. See Figure 15.

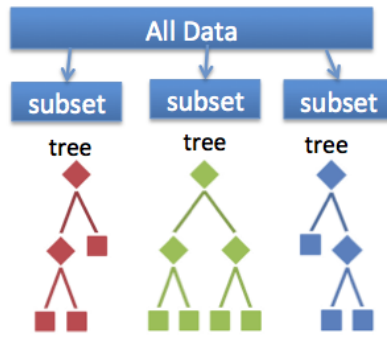


Figure 14: Random forest. We create new datasets by resampling the original training dataset with replacement. For each new dataset, we learn a tree.

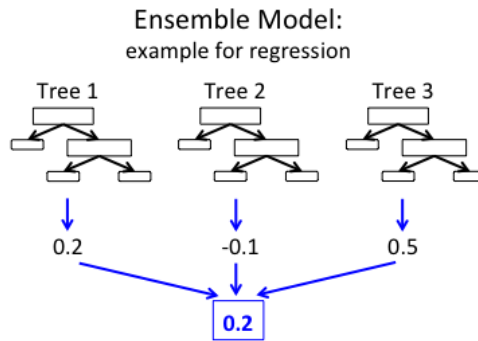


Figure 15: Random forest. The prediction by a random forest is an average of the predictions by the trees in the forest in the case of regression, and is the majority vote in the case of classification.

### 7.3 Adaboost

#### Committee of weak classifiers

Adaboost is a committee machine or ensemble machine for classification, which consists of a number of weak classifiers  $h_k(x_i) \in \{+, -\}$ ,  $k = 1, \dots, d$ . The final classification is a perceptron based on the weak classifiers,

$$y_i = \text{sign} \left( \sum_{k=1}^d \beta_k h_k(x_i) \right),$$

where  $\beta_k$  can be interpreted as the weight of vote of classifier  $k$ .

You may compare the above committee machine to the two-layer neural net we studied before, where  $h_k(x_i)$  plays the role of  $h_{ik}$ .

#### Adding one classifier at a time

In Adaboost, the based learners are weak classifiers. As illustrated by Figure 16, we want to learn a committee to separate the positive examples and negative examples, where the weak classifiers are based on thresholding either  $x_1$  or  $x_2$ . Each iteration adds a new weak classifier, and then assigns bigger weights to

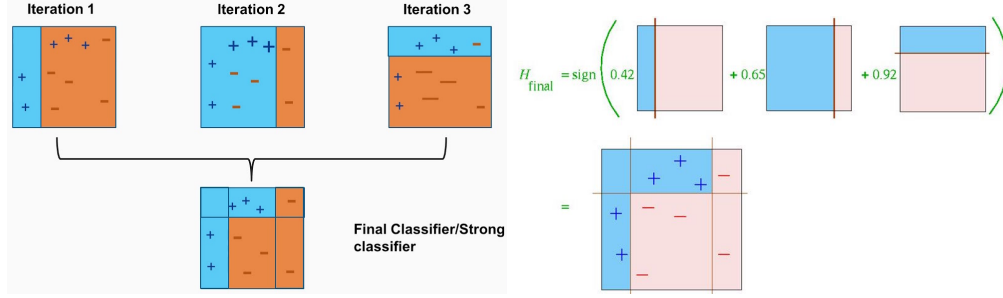


Figure 16: Adaboost. Each iteration adds a new classifier to the committee, and the final committee is a linear combination of the selected classifiers. Each new classifier is trained on reweighed dataset, where those examples that are not classified correctly by the current committee receive bigger weights, which are illustrated by bigger sizes of the + and - signs. The voting weight or coefficient of each added classifier is based on its performance on the reweighed dataset.

the examples that are not classified well by the current committee. Then in the next iteration, we learn a new classifier on the reweighed dataset that focuses on mistakes. Whenever a new classifier is added, it is assigned a voting weight in the committee. The final classifier is a weighted sum of the selected weak classifiers. The Adaboost algorithm is another example of learning from errors.

Even if the classifiers  $\{h_k\}$  may be weak, it is still possible to boost them into a strong classifier.

When training an Adaboost classifier, we sequentially add members to the committee.

Suppose the current committee has  $m - 1$  classifiers,

$$F_{m-1}(x_i) = \sum_{k=1}^{m-1} \beta_k h_k(x_i),$$

and we want to add a new member  $h_m()$  to boost the current classifier to

$$F_m(x_i) = F_{m-1}(x_i) + \beta_m h_m(x_i).$$

We need to select the new classifier  $h_m()$  and assign its weight  $\beta_m$ .

## Exponential loss

The exponential loss is an upper bound of the training error, because

$$1(y_i \neq \text{sign}(f_i)) \leq \exp(-y_i f_i(x_i)).$$

We use this loss to guide the selection of the next weak classifier,

$$\begin{aligned} \mathcal{L}(h_m, \beta_m) &= \sum_{i=1}^n \exp[-y_i (F_{m-1}(x_i) + \beta_m h_m(x_i))] \\ &\propto \sum_{i=1}^n D_i \exp[-\beta_m y_i h_m(x_i)], \end{aligned}$$

where

$$D_i \propto \exp[-(y_i F_{m-1}(x_i))],$$

and we usually normalize  $D_i$  so that  $\sum_{i=1}^n D_i = 1$ .  $\{D_i\}$  is a distribution over the training examples  $\{(x_i, y_i)\}$ . If  $(x_i, y_i)$  is well classified by the current  $F_{m-1}()$ , then  $y_i F_{m-1}(x_i)$  is large, and  $D_i$  is small. Thus the distribution

$\{D_i\}$  focuses on those examples that are not classified well, i.e., it focuses on the errors or mistakes, so that the training algorithm can learn from them.

$y_i h_m(x_i)$  can only be  $+1$  or  $-1$ . If  $y_i h_m(x_i) = 1$ , then  $h_m(x_i) = y_i$ , meaning that  $h_m()$  classifies  $x_i$  correctly. Otherwise if  $y_i h_m(x_i) = -1$ , then  $h_m()$  makes an error on  $x_i$ . Therefore

$$\mathcal{L}(h_m, \beta_m) = \sum_{i: h_m(x_i) = y_i} D_i e^{-\beta} + \sum_{i: h_m(x_i) \neq y_i} D_i e^{\beta} = (1 - \varepsilon) e^{-\beta} + \varepsilon e^{\beta},$$

where  $\varepsilon = \sum_{i: h_m(x_i) \neq y_i} D_i$  is the error of  $h_m()$  on the reweighed or refocused dataset.

Recall that for positive  $a$  and  $b$ ,  $a + b \geq 2\sqrt{ab}$ , because  $a + b - \sqrt{ab} = (\sqrt{a} - \sqrt{b})^2 \geq 0$ , with equality achieved when  $a = b$ , thus for a fixed  $h_m()$ ,

$$(1 - \varepsilon) e^{-\beta} + \varepsilon e^{\beta} \geq 2\varepsilon(1 - \varepsilon),$$

with the equality achieved by

$$(1 - \varepsilon) e^{-\beta} = \varepsilon e^{\beta},$$

so that

$$\beta = \frac{1}{2} \log \frac{1 - \varepsilon}{\varepsilon}.$$

Therefore we want to choose  $h_m()$  to minimize  $\varepsilon(1 - \varepsilon)$ . Since  $\varepsilon \leq 1/2$  (otherwise we can always flip  $h_m()$ ),  $\varepsilon(1 - \varepsilon)$  is an increasing function over  $\varepsilon \in [0, 1/2]$ . Thus we only need to choose  $h_m()$  to minimize the training error  $\varepsilon$ .

## 7.4 Gradient boosting

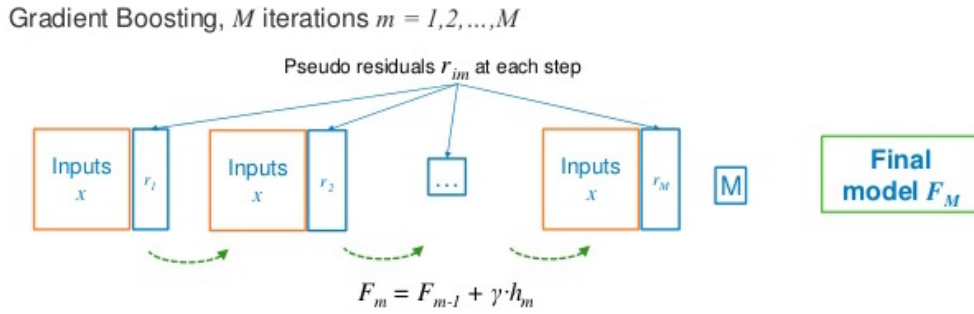


Figure 17: Gradient boosting. Each iteration learns a tree from the errors of the current model. The final model is a linear combination of the learned trees.

We want to learn a function

$$f(x) = \sum_{k=1}^d h_k(x)$$

for the purpose of regression or classification, where  $h_k(x)$  are base functions that are not necessarily classifiers, i.e.,  $h_k(x)$  may return continuous values.

Let  $F_{m-1}(x) = \sum_{k=1}^{m-1} h_k(x)$  be the current function. We want to add a base function  $h_m(x)$  so that

$$F_m(x) = F_{m-1}(x) + h_m(x).$$

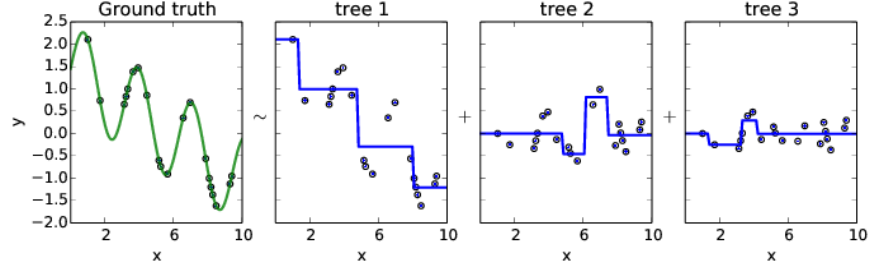


Figure 18: Gradient boosting for regression in one dimensional situation. Each tree is a piecewise constant function.



Figure 19: Gradient boosting for classification in two dimensional situation. Each tree is a piecewise constant function.

For regression, the least squares loss is

$$\mathcal{L} = \sum_{i=1}^n [y_i - (F_{m-1}(x_i) + h_m(x_i))]^2 = \sum_{i=1}^n (r_i - h_m(x_i))^2,$$

where

$$r_i = y_i - F_{m-1}(x_i)$$

is the residual error for observation  $i$ . We can then treat  $\{(x_i, r_i), i = 1, \dots, n\}$  as our new dataset, and learn a regression tree  $h_m(x)$  from this new dataset.

For a general loss function, e.g., the mean absolute deviation,

$$\mathcal{L} = \sum_{i=1}^n |y_i - (F_{m-1}(x_i) + h_m(x_i))|,$$

which penalizes large deviations less than the least squares loss and is thus more robust to outliers, we may learn a new tree  $h_m(x)$  by minimizing  $|r_i - h_m(x_i)|$ . However we need to rewrite the code for learning the tree. It is better to continue to use the code for least squares regression tree.

Let

$$s_i = F_{m-1}(x_i) + h_m(x_i),$$

for  $i = 1, \dots, n$ , and let  $\hat{s}_i = F_{m-1}(x_i)$  be the prediction by  $F_{m-1}$ . Let the loss function be  $\mathcal{L} = \sum_{i=1}^n L(y_i, s_i)$ . In the case of mean absolute deviation,  $\mathcal{L}(y_i, s_i) = |y_i - s_i|$ . Let

$$r_i = -\frac{\partial L(y_i, s_i)}{\partial s_i} \Big|_{\hat{s}_i},$$

Then  $r = (r_i, i = 1, \dots, n)^\top$  is the direction to change  $f = (f_i, i = 1, \dots, n)$  for the steepest descent of  $\mathcal{L}$ .  $r_i$  becomes the residual error in the least squares regression. In general,  $r_i$  means the lack of fitness for example  $i$  because if  $r_i$  is close to 0, there is no need to change  $s_i$  to fit the data. Otherwise we need to make big change in  $f_i$  to fit the data.

We can fit a tree to approximate  $r$  by minimizing

$$\sum_{i=1}^n (r_i - h_m(x_i))^2$$

using the code of least squares regression tree.

## 7.5 Extreme gradient boosting (XGB) as iterated reweighed least squares

A more principled implementation of gradient boosting is to expand  $L(y_i, s_i)$  by the second order Taylor expansion,

$$L(y_i, s_i) \approx L(y_i, \hat{s}_i) + L'(y_i, \hat{s}_i)h_m(x_i) + \frac{1}{2}L''(y_i, \hat{s}_i)h_m(x_i)^2,$$

where  $L'(y, s) = \frac{\partial}{\partial s}L(y, s)$ , and  $L''(y, s) = \frac{\partial^2}{\partial s^2}L(y, s)$ . Let  $r_i = L'(y_i, \hat{s}_i)$  and  $w_i = L''(y_i, \hat{s}_i)$ , then we can write

$$\begin{aligned} L(y_i, s_i) &= \frac{w_i}{2} \left[ h_m(x_i)^2 - 2 \frac{r_i}{w_i} h_m(x_i) \right] + \text{const} \\ &= \frac{w_i}{2} \left[ \frac{r_i}{w_i} - h_m(x_i) \right]^2 + \text{const} \\ &= \frac{w_i}{2} [\tilde{y}_i - h_m(x_i)]^2 + \text{const}. \end{aligned}$$

Thus we can learn  $h_m$  by weighted least squares, with working response  $\tilde{y}_i = r_i/w_i$ , and working weight  $w_i$ . Compared to the original gradient boosting, we take into account the curvature  $w_i$ . The bigger the curvature it, the more important the observation is. We divide  $r_i$  by  $w_i$  because if the curvature is big, then a small step will cause big decrease. This is similar to the Newton-Raphson algorithm for fitting generalized linear models such as logistic regression.

## 8 Neural Networks

In neural networks, we still have  $f(x) = h(x)^\top \beta$ , but  $h(x)$  is not designed in terms of kernels and trees, instead  $h(x)$  is itself expressed as linear models of features at lower layers. Thus a neural network is a linear model on top of linear models, with coordinate-wise non-linear transformation such as the sigmoid in logistic regression. If the non-linear transformation is the so-called rectified linear unit,  $f(x)$  is a piecewise linear function, where the pieces are recursively partitioned, similar to trees.

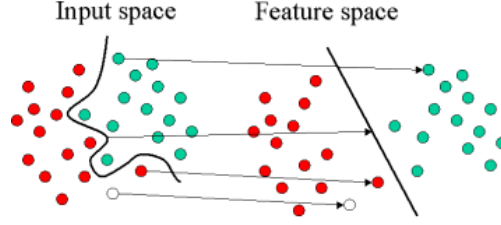


Figure 20: The positive examples and negative examples cannot be separated by a hyperplane in the original space. We can transform each  $x_i$  into a feature  $h_i$ , so that the examples can be separated by a hyperplane in the feature space.

obs	input	hidden	output
1	$X_1^\top$	$h_1^\top$	$y_1$
2	$X_2^\top$	$h_2^\top$	$y_2$
...			
$n$	$X_n^\top$	$h_n^\top$	$y_n$

## 8.1 Two layer perceptron

A perceptron seeks to separate the positive examples and negative examples by projecting them onto a vector  $\beta$ , or in other words, separating them using a hyperplane that is perpendicular to  $\beta$ . If the data are not linearly separable, a perceptron cannot work. We may need to transform the original variables into some other feature space so that the examples can be linearly separated. See Figure 20. One way to solve this problem is to generalize the perceptron into multi-layer perceptron. This structure is also called feedforward neural network. See Figure 21.

The neural network is logistic regression on top of logistic regressions.  $y_i \in \{0, 1\}$  follows a logistic regression on  $h_i = (h_{ik}, k = 1, \dots, d)^\top$ , and each  $h_{ik}$  follows a logistic regression on  $x_i = (x_{ij}, j = 1, \dots, p)^\top$ ,

$$y_i \sim \text{Bernoulli}(p_i),$$

$$p_i = \text{sigmoid}(h_i^\top \beta) = \text{sigmoid}\left(\sum_{k=1}^d \beta_k h_{ik}\right),$$

$$h_{ik} = \text{sigmoid}(x_i^\top \alpha_k) = \text{sigmoid}\left(\sum_{j=1}^p \alpha_{kj} x_{ij}\right).$$

### Rectified linear unit (ReLU)

In modern neural nets, the non-linearity is often achieved through rectified linear units (ReLU)  $\max(0, a)$ . See Figure 22.

$$y_i \sim \text{Bernoulli}(p_i),$$

$$p_i = \text{sigmoid}(h_i^\top \beta) = \text{sigmoid}\left(\sum_{k=1}^d \beta_k h_{ik}\right),$$

$$h_{ik} = \max(x_i^\top \alpha_k, 0) = \max\left(\sum_{j=1}^p \alpha_{kj} x_{ij}, 0\right).$$

Recall the linear spline model  $f(x_i) = \beta_0 + \sum_{j=1}^p \beta_j \max(0, x_i - a_j)$ , see Figure 23, where  $a_j$  are the knots where the spline makes a turn, i.e., a change of slope by  $\beta_j$ . The neural net with ReLU can be viewed as a

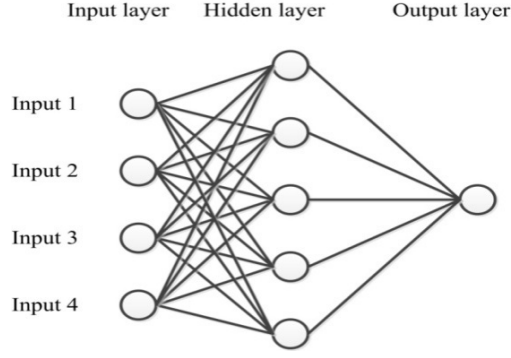


Figure 21: A two layer feedforward neural network. The output follows a logistic regression on the hidden vector. Each component of the hidden vector in turn follows a logistic regression on the input vector.

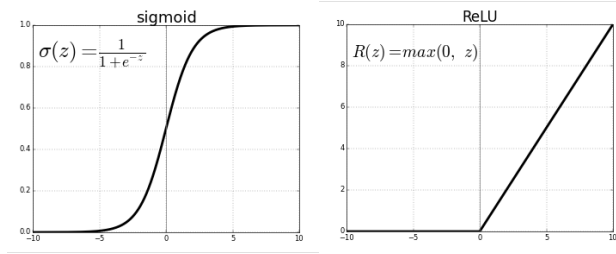


Figure 22: The left is the sigmoid function, and the right is the ReLU function. The sigmoid function saturates at the two ends, causing the gradient to vanish. The ReLU does not saturate for big positive input.

high-dimensional spline or piecewise linear mapping. If there are many layers in the neural net, the number of linear pieces is exponential in the number of layers. It can approximate highly non-linear mapping by patching up the large number of linear pieces.

## Back-propagation

When the non-linear rectification is sigmoid, the log-likelihood is

$$\mathcal{L}(\beta, \alpha) = \sum_{i=1}^n \left\{ y_i \sum_{k=1}^d \beta_k h_{ik} - \log \left[ 1 + \exp \left( \sum_{k=1}^d \beta_k h_{ik} \right) \right] \right\}.$$

This time we use  $\mathcal{L}(\beta, \alpha)$  to denote the log-likelihood function, which is to be maximized.

The gradient is

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \beta} &= \sum_{i=1}^n (y_i - p_i) h_i, \\ \frac{\partial \mathcal{L}}{\partial \alpha_k} &= \frac{\partial \mathcal{L}}{\partial h_k} \frac{\partial h_k}{\partial \alpha_k} = \sum_{i=1}^n (y_i - p_i) \beta_k h_{ik} (1 - h_{ik}) x_i \end{aligned}$$

$\partial \mathcal{L} / \partial \alpha_k$  is calculated by the chain rule. Again the gradient descent learning algorithm learns from the mistake or error  $y_i - p_i$ . The chain rule back-propagates the error to assign the blame to  $\beta$  and  $\alpha$  in order for  $\beta$  and  $\alpha$  to update. If the current network makes a mistake on the  $i$ -th example,  $\beta$  will change to be more



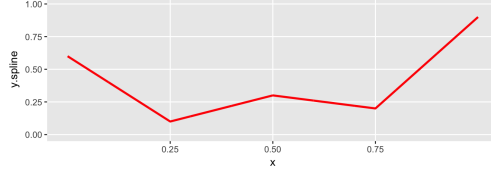


Figure 23: Spline is a continuous piecewise linear function, where at each knot, the spline makes a turn by changing the slope. The neural net can be viewed a high dimensional spline with exponentially many linear pieces.

aligned with  $h_i$ , while each  $\alpha_k$  also changes to be more aligned with  $x_i$ . The amount of change depends on  $\beta_k$  as well as  $h_{ik}(1 - h_{ik})$ , which measures how big a role played by  $x^\top \alpha_k$  in predicting  $y_i$ . If  $x^\top \alpha_k$  plays a big role, then  $\alpha_k$  should receive much blame and change.

For ReLU  $\max(0, a)$ , we should replace  $h_{ik}(1 - h_{ik})$  in the back-propagation by  $1(h_{ik} > 0)$ , which is a binary detector. The sigmoid function saturates at the two ends, where the derivatives are close to zero. This can cause the vanishing gradient problem in back-propagation, so that the network cannot learn from the error. The ReLU function does not saturate for big positive input, which indicates the existence of a certain patten. This help avoids the vanishing gradient problem.

## 8.2 Implicit regularization by gradient descent

For simplicity, let us assume  $x$  is one-dimensional.  $h_k = \max(0, a_k x + b_k)$ , and  $s = f(x) = \sum_{k=1}^d \beta_k h_k$ . As discussed above,  $f(x)$  is a linear spline.

Suppose we initialize at  $(a_k, b_k) \sim p_0(a, b)$  independently for  $k = 1, \dots, d$  and freeze them, and we only learn  $\beta_k$  by gradient descent, starting from  $\beta_k^{(0)} = 0$ . For training examples  $\{x_i, y_i, i = 1, \dots, n\}$ , the loss function is

$$\mathcal{L}(\beta) = \frac{1}{n} \sum_{i=1}^n (y_i - h(x_i)^\top \beta)^2.$$

If  $d \leq n$ , there is a single minimum, and gradient descent converges to this minimum.

When  $d > n$ , the model is over-parametrized, i.e., the number of parameters is greater than the number of training examples. Then there are infinitely many  $\beta$  that gives  $L(\beta) = 0$ . Gradient descent will converge to one such  $\beta$ .

Gradient descent is

$$\Delta \beta \propto \frac{1}{n} \sum_{i=1}^n (y_i - h(x_i)^\top \beta) h(x_i).$$

It will converge to a  $\hat{\beta} = \sum_i c_i h(x_i)$  that satisfies  $y_i = h(x_i)^\top \hat{\beta}$  for all  $i$ . Suppose there is another  $\tilde{\beta}$  that satisfies  $y_i = h(x_i)^\top \tilde{\beta}$  for all  $i$ . Let  $\tilde{\beta} = \hat{\beta} + \Delta$ , then  $\Delta \perp h(x_i)$  for all  $i$ , i.e.,  $\Delta \perp \hat{\beta}$ . Thus  $\|\tilde{\beta}\|^2 = \|\hat{\beta}\|^2 + \|\Delta\|^2$ . Thus gradient descent will converge to  $\hat{\beta}$  with minimal  $\ell_2$  norm. This is implicit regularization, and it underlies the double descent behavior as illustrated by Figure 24. When  $p \leq n$ , there is only one minimum of  $\mathcal{L}(\beta)$ , and there is no regularization, thus the overfitting becomes severe as  $p$  approaches  $n$ . But as  $p > n$ , there are infinitely many  $\beta$  with  $\mathcal{L}(\beta) = 0$ , and gradient descent provides implicit regularization, so that testing error starts to decreases again.

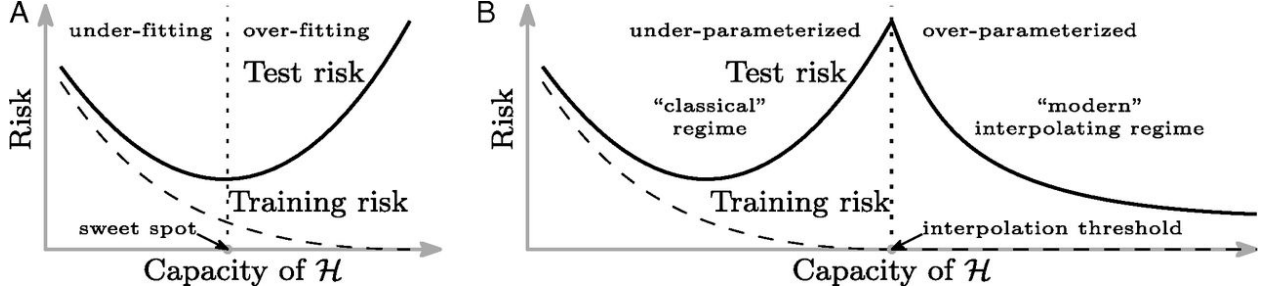


Figure 24: Double descent. Left: testing error for classical statistical model. Right: testing error for modern over-parametrized neural network.

### 8.3 Connection to kernel machine

Suppose we freeze  $(a_k, b_k)$ , and only learn  $\beta$  by gradient descent, then it is equivalent to the kernel machine

$$K(x, x') = \langle h(x), h(x') \rangle = \sum_{k=1}^d h_k(x) h_k(x') = \frac{1}{d} \sum_{k=1}^d \tilde{h}_k(x) \tilde{h}_k(x'),$$

where  $\tilde{h}_k = \sqrt{d} h_k$ . Assuming  $(\sqrt{d} a_k, \sqrt{d} b_k) \sim \tilde{p}_0(a, b)$ , where  $\tilde{p}_0$  is independent of  $d$ , then for large  $d$ , by law of large number,

$$K(x, x') \rightarrow \mathbb{E}_{\tilde{p}_0(a, b)} [\tilde{h}_k(x) \tilde{h}_k(x')].$$

So this model is equivalent to kernel regression. Even if we free  $(a_k, b_k)$  and learn them together with  $\beta_k$ , the learning dynamics is still similar to kernel machine, where, by first-order Taylor expansion,  $h(x) = \frac{\partial}{\partial \theta} f_\theta(x)$ , i.e., the first derivative evaluated at the initialization  $\theta_0$ . The kernel is called neural tangent kernel.

### 8.4 Connection to Gaussian process

We can adopt a Bayesian framework, by assuming a prior distribution  $(a_k, b_k, \beta_k) \sim p(a, b) p(\beta)$  independently. Then

$$f(x) = \sum_{k=1}^d \beta_k h_k(x) = \frac{1}{\sqrt{d}} \sum_{k=1}^d \tilde{\beta}_k h_k(x),$$

where  $\tilde{\beta}_k = \sqrt{d} \beta_k$ . Assuming  $\tilde{\beta} \sim \tilde{p}(\beta)$  independently, where  $\tilde{p}(\beta)$  is independent of  $d$ , and assuming  $\mathbb{E}(\tilde{\beta}) = 0$ , then according to the central limit theorem, as  $d \rightarrow \infty$ ,

$$(f(x_i), i = 1, \dots, n)^\top \sim N(0, K),$$

where  $K_{ij} = K(x_i, x_j)$ , and  $K(x, x') = \mathbb{E}(f(x) f(x'))$  is the kernel. Thus the model is equivalent to the Gaussian process.

### 8.5 Multi-layer network

See Figure 25 for an illustration of multi-layer perceptron or a feedforward neural network. It consists an input layer, an output layer, and multiple hidden layers in between. Each layer can be represented by a vector, which is obtained by multiplying the layer below it by a weight matrix, plus a bias vector, and then transforming each element of the resulting vector by sigmoid or ReLU etc.

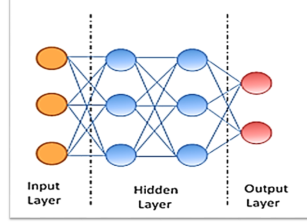


Figure 25: Multi-layer perceptron (two hidden layers here) or feedforward neural network. There is an input layer (or an input vector), an output layer, and multiple hidden layers (or hidden vectors). Each layer is a linear transformation of the layer beneath, followed by an element-wise non-linear transformation.

More formally, the network has the following recursive structure:

$$\begin{aligned} h_l &= f_l(s_l), \\ s_l &= W_l h_{l-1} + b_l, \end{aligned}$$

for  $l = 1, \dots, L$ , where  $l$  denotes the layer, with  $h_0 = X$ , which is the input vector, and  $h_L$  is used to predict  $Y$ , which is the output, based on a log-likelihood function  $L(Y, h_L)$ . Here  $h_L$  corresponds to  $X^\top \beta$  in the previous section.  $W_l$  and  $b_l$  are the weight matrix and bias vector respectively.  $f_l$  is an element-wise transformation, i.e., each component of  $h_l$  is obtained by a non-linear transformation of the corresponding component of  $s_l$ , such that  $h_{lk} = f_l(s_{lk})$  for each  $k$ .

## 8.6 Multi-layer back-propagation

We may write the forward pass as

$$\begin{array}{ccccccc} x & \rightarrow & h_1 & \rightarrow & \dots & \rightarrow & h_{l-1} & \rightarrow & h_l & \dots \rightarrow & L. \\ & & \uparrow & & & & \uparrow & & \uparrow & & \\ & & W_1 & & \dots & & W_{l-1} & & W_l & & \end{array}$$

The back-propagation pass is

$$\begin{array}{ccccccc} x & \leftarrow & h_1 & \leftarrow & \dots & \leftarrow & h_{l-1} & \leftarrow & h_l & \dots \leftarrow & L. \\ & & \downarrow & & & & \downarrow & & \downarrow & & \\ & & W_1 & & \dots & & W_{l-1} & & W_l & & \end{array}$$

This is a process of assigning blame. If the loss function  $L$  finds something wrong, he will blame  $h_L$  and  $W_{L+1}$ . This layer is usually the soft-max layer.  $h_L$  will then blame  $h_{L-1}$  and  $W_L$ , and so on. This process follows the chain rule:

$$\begin{aligned} \frac{\partial L}{\partial h_{l-1}^\top} &= \sum_{k=1}^d \frac{\partial L}{\partial h_{l,k}} \frac{\partial h_{l,k}}{\partial s_{l,k}} \frac{\partial s_{l,k}}{\partial h_{l-1}^\top} \\ &= \sum_{k=1}^d \frac{\partial L}{\partial h_{l,k}} f'_l(s_{l,k}) W_{l,k} \\ &= \frac{\partial L}{\partial h_l^\top} f'_l W_l, \end{aligned}$$

where  $W_{l,k}$  is the  $k$ -th row of  $W_l$ , and  $f'_l = \text{diag}(f'_l(s_{l,k}), k = 1, \dots, d)$ .

$$\begin{aligned}\frac{\partial L}{\partial W_{l,k}} &= \frac{\partial L}{\partial h_{l,k}} \frac{\partial h_{l,k}}{\partial s_{l,k}} \frac{\partial s_{l,k}}{\partial W_{l,k}} \\ &= \frac{\partial L}{\partial h_{l,k}} f'_l(s_{l,k}) h_{l-1}^\top,\end{aligned}$$

thus,

$$\frac{\partial L}{\partial W_l} = f'_l \frac{\partial L}{\partial h_l} h_{l-1}^\top.$$

Similarly,

$$\frac{\partial L}{\partial b_l} = f'_l \frac{\partial L}{\partial h_l}.$$

## 8.7 Stochastic gradient descent (SGD)

### Mini-batch

Let the training data be  $(x_i, y_i)$ ,  $i = 1, \dots, n$ . Let  $L_i(\theta) = L(y_i, x_i; \theta)$  be the loss caused by  $(x_i, y_i)$ . For regression,  $L(y_i, x_i; \theta) = (y_i - f(x_i))^2$ , where  $f(x_i)$  is parametrized by a neural network with parameters  $\theta$ . For classification,  $L(y_i, x_i; \theta) = -\log p(y_i|x_i)$  where  $p(y_i|x_i)$  is modeled by a neural network with parameters  $\theta$ , with a softmax layer at the top. Let  $\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n L_i(\theta)$  be the overall loss averaged over the whole training dataset. The gradient descent is

$$\theta_{t+1} = \theta_t - \eta \mathcal{L}'(\theta_t),$$

where  $\eta$  is the step size or learning rate, and  $\mathcal{L}'(\theta)$  is the gradient.

The above gradient descent may be time consuming because we need to compute  $\mathcal{L}'(\theta) = \frac{1}{n} \sum_{i=1}^n L'_i(\theta)$  by summing over all the examples. If the number of examples is large, the computation can be time consuming. We may use the following stochastic gradient descent algorithm. At each step, we randomly select  $i$  from  $\{1, 2, \dots, n\}$ . Then we update  $\theta$  by

$$\theta_{t+1} = \theta_t - \eta_t L'_i(\theta_t),$$

where  $\eta_t$  is the step size or learning rate, and  $L'_i(\theta_t)$  is the gradient only for the  $i$ -th example. Because  $i$  is randomly selected, the above algorithm is called the stochastic gradient descent algorithm.

Instead of randomly selecting a single example, we may randomly select a mini-batch, and replace  $L'_i(\theta_t)$  by the average of this mini-batch.

In order for the algorithm to converge to a local minimum, we need the following conditions. (1)  $\sum_{t=1}^{\infty} \eta_t = \infty$ . (2)  $\sum_{t=1}^{\infty} \eta_t^2 < \infty$ . The first condition ensures that the algorithm can go the distance toward the minimum. The second condition ensures that the algorithm will not run away from the local minimum once it arrives. One simple example is  $\eta_t = c/t$  for a constant  $c$ . In practice, we need more sophisticated schemes for choosing  $\eta_t$ . For instance, reducing  $\eta_t$  after a certain number of steps, or reducing  $\eta_t$  if the training error stops decreasing.

### Momentum, Adagrad, RMSprop, Adam

The gradient descent algorithm goes downhill in the steepest direction in each step. However, the steepest direction may not be the best direction, as illustrated by Figure 26. In the left figure, the black arrows are the gradient direction. The red arrows are the preferred direction, which is the direction of momentum.

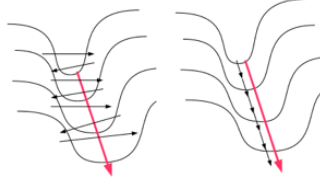


Figure 26: Momentum. The left figure illustrates the original gradient descent algorithm. The black arrow is the gradient direction, and the red arrow is the preferred direction. The right figure illustrates the gradient descent algorithm with momentum, which is along the red arrow.

It is better to move along the direction of the momentum, as illustrated by the right figure. We want to accumulate the momentum, and let it guide the descent. The following is the stochastic gradient descent with momentum:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta_t g_t, \\ \theta_t &= \theta_{t-1} - v_t. \end{aligned}$$

where  $g_t$  is the average gradient computed from the current mini-batch, and  $v_t$  is the momentum or velocity.  $\gamma$  is usually set at .9, for accumulating the momentum, and  $\theta$  is updated based on the momentum.

Adagrad modifies the gradient descent algorithm in another direction. The magnitudes of the components of  $g_t$  may be very uneven, and we need to be adaptive to that. The Adagrad let

$$\begin{aligned} G_t &= G_{t-1} + g_t^2, \\ \theta_{t+1} &= \theta_t - \eta_t \frac{g_t}{\sqrt{G_t + \varepsilon}}, \end{aligned}$$

where  $\varepsilon$  is a small number to avoid dividing by 0. In the above formula,  $g_t^2$  and  $g_t/\sqrt{G_t + \varepsilon}$  denote component-wise square and division.

In Adagrad,  $G_t$  is the sum over all the time steps. It is better to sum over the recent time steps. Adadelta and RMSprop use the following scheme:

$$G_t = \beta G_{t-1} + (1 - \beta) g_t^2,$$

where  $\beta$  can be set at .9. It can be shown that

$$G_t = (1 - \beta)(\beta^{t-1} g_1^2 + \beta^{t-2} g_2^2 + \dots + \beta g_{t-1}^2 + g_t^2),$$

which is a sum over time with decaying weights.

The Adam optimizer combines the idea of RMSprop and the idea of momentum.

$$\begin{aligned} v_t &= \gamma v_{t-1} + (1 - \gamma) g_t, \\ G_t &= \beta G_{t-1} + (1 - \beta) g_t^2, \\ v_t &\leftarrow v_t / (1 - \gamma), G_t \leftarrow G_t / (1 - \beta), \\ \theta_{t+1} &= \theta_t - \eta_t \frac{v_t}{\sqrt{G_t + \varepsilon}}. \end{aligned}$$

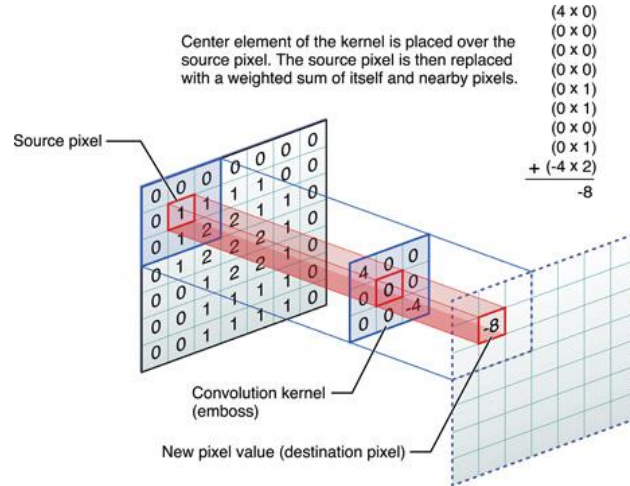


Figure 27: Local weighted summation. Here the filter or kernel is  $3 \times 3$ . It slides over the whole input image or feature map. At each pixel, we compute the weighted sum of the  $3 \times 3$  patch of the input image, where the weights are given by the filter or kernel. This gives us an output image or filtered image or feature map.

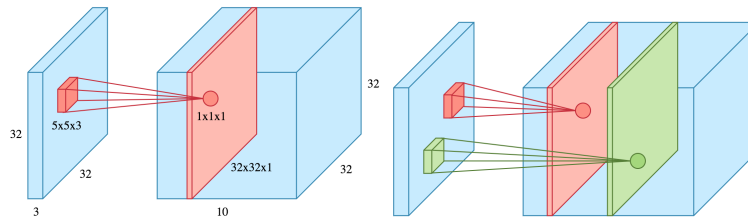


Figure 28: Convolution. The input may consist of multiple channels, illustrated by a rectangle box. Each input feature map is a slice of the box. The local weighted summation in convolution is also over the channels. If the spatial range of the filter is  $3 \times 3$  and the input has 3 channels, then the filter or kernel is  $3 \times 3 \times 3$  box. The spatial range can also be  $1 \times 1$ , then the filter involves weighted summation of the 3 channels at the same pixel. For the input image, there are 3 channels corresponding to 3 colors: red, green, black. For the hidden layers, each layer may consist of hundreds of channels. Each channel is obtained by a filter. For instance, in the figure on the right, the red feature map is obtained by the red filter, and the green feature map is obtained by the green filter.

## 8.8 Convolutional neural networks (CNN, ConvNet)

### Convolution, kernels, filters

In the neural network,  $h_l = f_l(W_l h_{l-1} + b_l)$ , the linear transformation  $s_l = W_l h_{l-1} + b_l$  that maps  $h_{l-1}$  to  $s_l$  can be highly structured. One important structure is the convolutional neural network, where  $W_l$  and  $b_l$  have a convolutional structure.

Specifically, a convolutional layer  $h_l$  is organized into a number of feature maps. Each feature map, also called a channel, is obtained by a filter or a kernel operating on  $h_{l-1}$ , which is also organized into a number of feature maps. Each filter is a local weighted summation, plus a bias term, followed by a non-linear transformation.

Figure 27 explains the local weighted summation. We convolve an input feature map with a  $3 \times 3$  filter to obtain an output feature map. The value of each pixel of the output feature map is obtained by a weighted summation of pixel values of the  $3 \times 3$  patch of the input feature map around this pixel. We apply the

summation around each pixel, using the same weights, to obtain the output feature map.

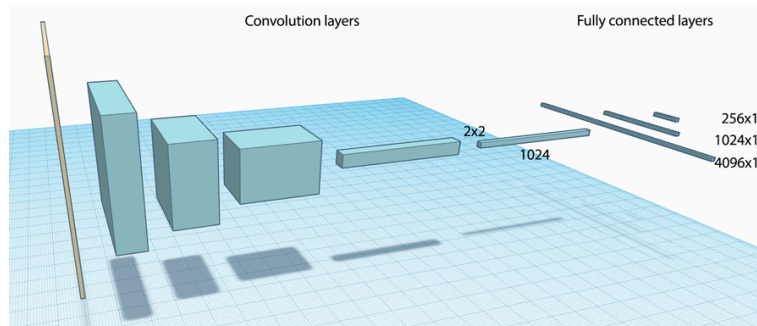


Figure 29: Convolutional neural network (CNN or ConvNet). The input image has 3 channels (R, G, B). Each subsequent layer consists of multiple channels, and is illustrated by a box. Sub-sampling is performed after some layers, so that the box at the higher layers is smaller than the box at the lower layer in the spatial extent. Meanwhile, the box at the higher layer may have more channels than the box at the lower layer, illustrated by the fact that the box at the higher layer is longer than the box at the lower layer. A fully connected layer consists of  $1 \times 1$  feature maps, and is illustrated by a horizontal line.

If there are multiple input feature maps, i.e., multiple input channels, the weighted summation is also over the multiple feature maps. We can apply different filters to the same set of input feature maps. Then we will get different output feature maps. Each output feature map corresponds to one filter. See Figure 28 for illustration.

After the weighted summation, we may also add a bias and apply a non-linear transformation such as sigmoid or ReLU. The filter becomes non-linear.

After obtaining the feature maps in  $h_l$ , we may perform max-pooling, e.g., for each feature map, at each pixel, we replace the value of this pixel by the maximum of the  $3 \times 3$  patch around this pixel. We may also do average pooling, i.e., at each pixel, we replace the value of this pixel by the average of the  $3 \times 3$  patch around this pixel.

The mapping from  $h_l$  to  $h_{l-1}$  may also involve sub-sampling to reduce the size of feature maps. For instance, after obtaining a feature map by a filter, we can partition the feature map into  $2 \times 2$  blocks, and within each block, we only keep the upper left pixel. This will reduce the width and height of the feature map by half.  $h_{l-1}$  may have more channels than  $h_l$ , because each element of  $h_l$  covers a bigger spatial extent than each element of  $h_{l-1}$ , and there are more patterns of bigger spatial extent.

The output feature map may also be  $1 \times 1$ , whose value is a weighted sum of all the elements in  $h_{l-1}$ .  $h_l$  may consist of a number of such  $1 \times 1$  maps. It is called the fully connected layer because each element of  $h_l$  is connected to all the elements in  $h_{l-1}$ .

A convolutional network consists of multiple layers of convolutional and fully connected layers, with max pooling and sub-sampling between the layers. See Figure 29. The network can be learned by back-propagation.

### Alex net, VGG net, inception net

The Alex net is the neural network that achieved the initial breakthrough on object recognition for the ImageNet dataset. It goes through several convolutional layers, followed by fully connected layers. It has 5 convolutional layers and 2 fully connected layers, plus a softmax output layer.

The VGG net is an improvement on the Alex net. There are two versions, VGG16 and VGG19, which consist of 16 hidden layers and 19 hidden layers respectively. The VGG19 has 144 million parameters. The

filters of the VGG nets are all  $3 \times 3$ . Like Alex net, it has two fully connected layers, plus a softmax output layer.

The inception net took its name from the movie “Inception,” which has a line “we need to go deeper.” The network makes extensive use of  $1 \times 1$  filters, i.e., for each feature map in  $h_l$ , each pixel value is a weighted summation of the pixel values of all the feature maps in  $h_{l-1}$  at the same pixel, plus a bias and a non-linear transformation. The  $1 \times 1$  filters serve to fuse the channels in  $h_{l-1}$  at each pixel. The feature maps at each layer of the inception net are obtained by filters of sizes  $1 \times 1$ ,  $3 \times 3$  and  $5 \times 5$ , as well as max pooling.

## Object detection, semantic segmentation, deformable CNN

CNN is commonly used for computer vision. Two prominent tasks are object detection and semantic segmentation. Object detection is to impose bounding boxes on the objects in the image and output the object categories. Semantic segmentation is to segment the image into different regions, with each region corresponding to an object whose category is labeled.

In object detection, we can use CNN to predict the possible bounding boxes and the shifts of the bounding boxes.

In deformable CNN, we can use CNN to predict the deformations of the grids that support the convolutional kernels.

## 8.9 Batch normalization

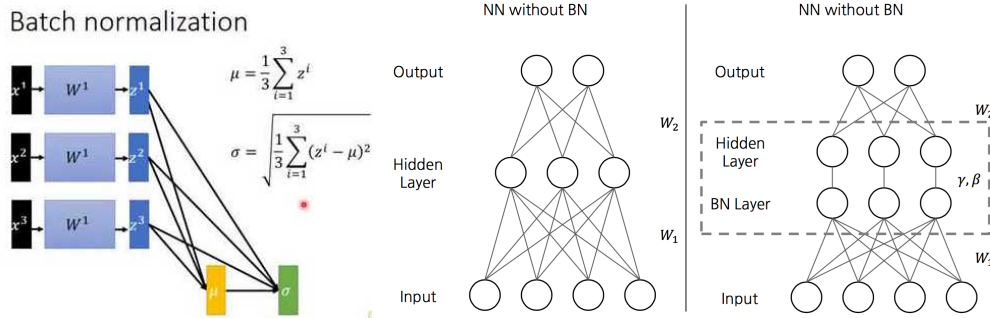


Figure 30: Batch normalization. Suppose we have a batch of 3 examples. For each element of each layer, we compute the mean  $\mu$  and standard deviation  $\sigma$  by pooling over the 3 examples. We then normalize the element for each example, followed by a linear transformation to be learned from the data. In back-propagation, we need to treat  $\mu$  and  $\sigma$  as a layer and compute the derivatives of  $\mu$  and  $\sigma$  with respect to the whole batch. It is as if the whole batch becomes a single example because  $\mu$  and  $\sigma$  are computed from this whole batch.

When training the neural net by back-propagation, the distribution of  $h_l$  keeps changing because the parameters keep changing. This may cause a problem in training. We can stabilize the distribution by batch normalization. That is, between  $h_{l-1}$  and  $h_l$ , we can add a batch normalization layer. For simplicity, let  $x$  be the input to the batch normalization layer, and let  $y$  be the output of the batch normalization layer. For instance,  $x$  is  $h_{l-1}$ , and  $y$  becomes the normalized version of  $h_{l-1}$  to be fed into the layer for computing  $h_l$ .

With slight abuse of notation, we let  $x$  be an element of  $h_{l-1}$ , and we apply the batch normalization for each element of  $h_{l-1}$ . Suppose we have a batch of  $n$  training examples, so that we have  $\{x_i, i = 1, \dots, n\}$ . The



batch normalization layer is defined as follows:

$$\begin{aligned}\mu &= \frac{1}{n} \sum_{i=1}^n x_i; \\ \sigma^2 &= \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2; \\ \hat{x}_i &= \frac{x_i - \mu}{\sigma}; \\ y_i &= \beta + \gamma \hat{x}_i.\end{aligned}$$

This way, we stabilize the distribution of each element of  $h_{l-1}$  during the training process. See Figure 30 for an illustration.

Note that in the batch normalization layer, there are parameters  $\beta$  and  $\gamma$  to be learned. More importantly,  $\mu$  and  $\sigma^2$  are functions of the whole batch. When we do chain rule calculations for back-propagation, we need to compute the derivatives of  $\mu$  and  $\sigma^2$  with respect to all the  $(x_i, i = 1, \dots, n)$ . It is as if the whole batch becomes a single training example.

## 8.10 Residual net

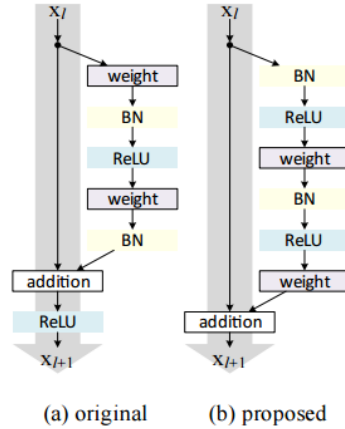


Figure 31: Residual net. The figure on the left is the original version of a residual block. The figure on the right is the revised version, which is in the form of  $x_{l+1} = x_l + F(x_l)$ , where  $F$  consists of two rounds of weighted sum, batch normalization, and ReLU.

A residual block in the residual net is as follows. Let  $x_l$  be the input to the residual block. Let  $x_{l+1}$  be the output of the residual block. Let  $F(x_l)$  be the transformation of  $x_l$  that consists of two rounds of weighted summation, batch normalization, and ReLU. We let

$$x_{l+1} = x_l + F(x_l),$$

as illustrated by the right plot of Figure 32.  $F(x_l)$  models the residual of the mapping from  $x_l$  to  $x_{l+1}$ , on top of the identity mapping. The following are some rationales for such a residual block.

(1) If we model  $x_{l+1} = F(x_l)$ , mathematically  $F(x_l)$  parametrized by a set of weights may be the same as  $x_l + F(x_l)$  parametrized by a different set of weights, computationally it can be more difficult for gradient descent to learn the former than the latter. It can be much easier for stochastic gradient descent to find a

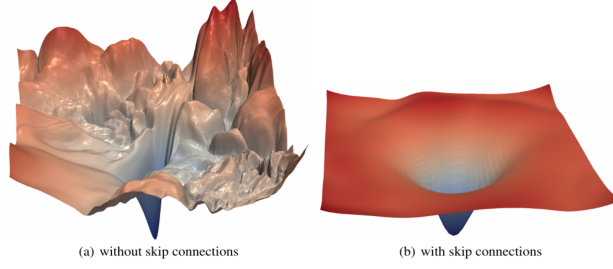


Figure 32: Visualizing loss function. Left: regular network. Right: residual net.

good  $F(x_l)$  in the residual form than in the original form. Specifically, the loss function with residual net parametrization is much smoother.

(2) The mapping  $x_{l+1} = x_l + F(x_l)$  may model an iteration of an iterative algorithm or dynamic process, where  $l$  actually denotes the time step  $t$  instead of the real layer  $l$ . The mapping models the iterative refinement of the same layer over time.

(3) With multiple residual blocks, we implicitly have an ensemble of networks. For instance, consider a simple network  $y = w_2 w_1 x$ , where all the symbols are scalars. If we adopt a residual form  $y = (1 + w_2)(1 + w_1)x$ , we can expand it as  $y = x + w_1 x + w_2 x + w_2 w_1 x$ . Thus the residual form is an ensemble of 4 networks. We may also think of the residual form as an expansion, like the Taylor expansion.

## 8.11 Recurrent neural networks (RNN), LSTM, GRU

### Vanilla RNN

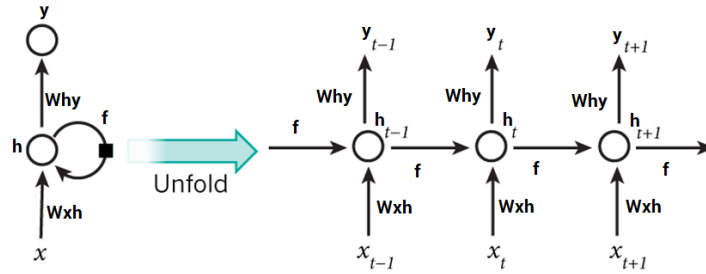


Figure 33: Recurrent neural network. The latent vector summarizes all the information about the past.

Figure 33 shows the vanilla version of the recurrent neural network, which can be written as

$$\begin{aligned} h_t &= f(W(h_{t-1}, x_t)) \\ y_t &= f(W h_t), \end{aligned}$$

where we use  $W$  as a generic notation for weight matrices, and  $f$  as a generic notation for element-wise non-linear transformations. They are different at each occurrence.  $(h_{t-1}, x_t)$  is the concatenated vector. We omit the bias, which can be absorbed into the weight matrix if we add 1 to the vector.

In the case of prediction,  $y_t = x_{t+1}$ . There are different forms of inputs and outputs for different applications of RNN, as illustrated by Figure 34.

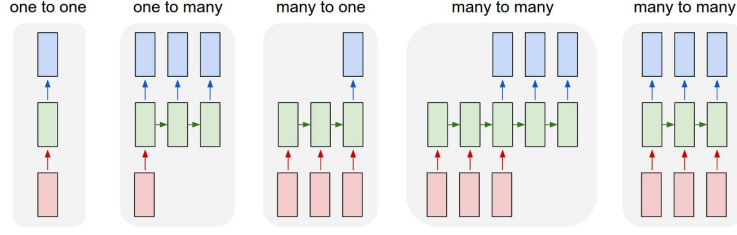


Figure 34: Different forms of inputs and outputs in RNN.

## Long short term memory (LSTM)

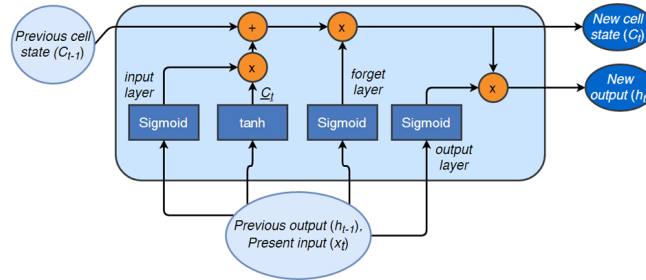


Figure 1. Architecture in a LSTM network

Figure 35: Long short term memory. There is a memory vector, and three gates, forget, input, and output.

The training of RNN is again based on back-propagation, except that we need to back-propagate over time. There is a vanishing (or exploding) gradient problem. The LSTM was designed to overcome this problem by introducing memory cells.

Figure 35 illustrates the architecture of LSTM, which can be written as

$$\begin{aligned} (f_t, i_t, o_t, \Delta c_t) &= f(W(h_{t-1}, x_t)), \\ c_t &= c_{t-1}f_t + \Delta c_t i_t, \\ h_t &= o_t f(c_t), \\ y_t &= f(W h_t). \end{aligned}$$

The gates are like if-then statements in a computer program. They are made continuous by sigmoid  $f$  so that they are differentiable. The residual net is a special case of LSTM.

## Gated recurrent units (GRU)

The gated recurrent unit is a simplification of LSTM, by merging  $c_t$  and  $h_t$ :

$$\begin{aligned} (z_t, r_t) &= f(W(h_{t-1}, x_t)), \\ \tilde{h}_t &= f(W(x_t, r_t h_{t-1})), \\ h_t &= (1 - z_t)h_{t-1} + z_t \tilde{h}_t, \\ y_t &= f(W h_t). \end{aligned}$$

$z_t$  is a gate to decide whether to renew or not, and  $r_t$  is a gate to decide whether to remember or not.

## 8.12 Encoder-decoder, thought vector

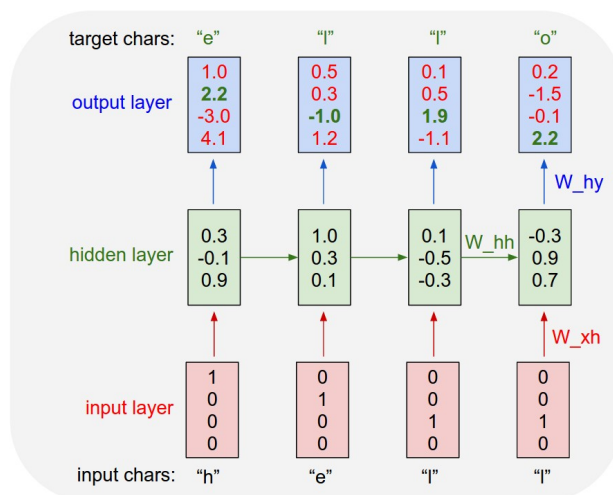


Figure 36: An RNN for sequence generation.

Figure 36 shows an RNN for sequence generation, where  $x_t$  is the current letter, and  $y_t$  is the next letter. Each letter can be represented by a one-hot vector. The RNN maps the one-hot vector into a hidden vector, and the hidden vector is used to generate the next letter by softmax probabilities. We can think of  $x_t \rightarrow h_t$  as the encoder, and  $h_t \rightarrow y_t$  as the decoder. Because  $h_t$  also depends on  $h_{t-1}$ , the  $h_t$  is a thought vector that encodes all the relevant information of the past. That is, we have the following scheme: input  $\rightarrow$  thought vector  $\rightarrow$  output.

### Translation

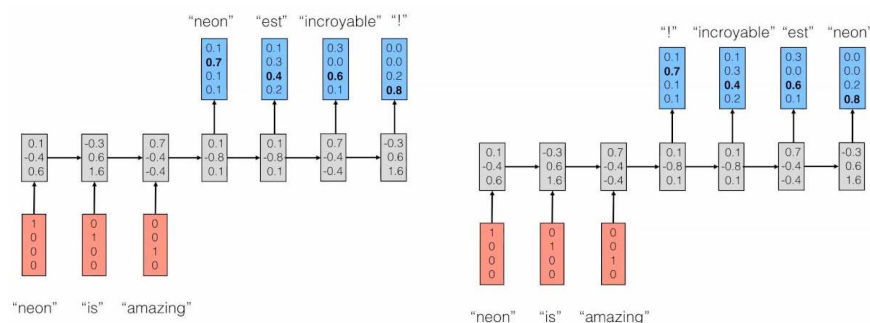


Figure 37: Machine translation. The encoding RNN encodes the input sentence into a thought vector, and the decoding RNN decodes the thought vector into an output sentence.

Figure 37 shows the RNN for machine translation. This time, each word is represented by a one-hot vector. We encode the input sentence into a thought vector by an encoding RNN. The thought vector is fed into a decoding RNN to generate the output sentence.

## Image captioning

Image captioning can be considered a special case of machine translation, where the encoding RNN is replaced by a CNN (e.g., VGG). We can take a certain layer of CNN as the thought vector to be fed into the decoding RNN to generate the caption.

## VQA

The visual question and answering can also be considered a special case of machine translation, where we have an encoder for images and an encoder for questions. We then concatenate the thought vectors of the image and question, and use the concatenated thought vector to generate the answer.

### 8.13 Memory and attention, query and key

For text based QA, e.g., answering questions based on wikipedia, we can encode the sentences in wikipedia into thought vectors. These vectors serve as memory. For an input question, we can encode it into a thought vector. By matching the thought vector of the question to the thought vectors of the memory, we can decide which sentence we want to pay attention to, using a softmax probability distribution over the sentences. The weighted sum of the thought vectors of the sentences weighted by the attention probabilities then becomes a combined thought vector, which, together with the thought vector of the question, is decoded into the answer.

Specifically, let  $m_i, i = 1, \dots, N$  be the vectors in the memory. Let  $Q$  be the question or query. We can let  $(key_i, value_i) = Wm_i$ . Then we can let

$$A = \sum_{i=1}^N attention_i \times value_i,$$

where

$$attention_i = \frac{\exp(\langle Q, key_i \rangle)}{\sum_{i'=1}^N \exp(\langle Q, key_{i'} \rangle)}.$$

We can then let  $A$  emit the answer by soft-max classification.

### 8.14 Word2vec for semantic embedding

Word embedding is at the foundation of modern Natural language processing (NLP). We embed the words into an Euclidean space. Suppose there are  $K$  words in the dictionary. We can represent each word  $x$  as a  $K$ -dimensional one-hot vector. Suppose we want to embed the words in the  $d$ -dimensional space. We can let  $h = Wx$ , so that the  $k$ -th column of  $W$ , i.e.,  $w_k$ , is the embedding of word  $k$ , whose  $x$  is a one-hot vector with  $k$ -th element being 1. This is a linear encoding. We can decode it by a linear soft-max classifier, i.e.,  $p(x|h) \propto \exp(\langle x, \tilde{W}h \rangle)$ , where  $\tilde{W}$  is the decoding matrix, and  $\tilde{W}h$  is the  $K$ -dimension score vector. We may train this model to predict the words within the context of each word. For instance the probability that word  $j$  is within the context of word  $i$  is

$$Q_{ij} = \frac{\exp(\langle w_i, \tilde{w}_j \rangle)}{\sum_k \exp(\langle w_i, \tilde{w}_k \rangle)}.$$

The GloVe (Global Vector) is similar to word2vec, where  $\log Q_{ij} = \langle w_i, \tilde{w}_j \rangle + b_i + \tilde{b}_j$ , where  $b_i$  and  $\tilde{b}_j$  are bias terms. This is similar to matrix factorization in recommender system.

It is likely that we use both vector and symbolic representations. In a certain context and for a short term task, the number of words or symbols can be made very small, so that it may be convenient to use symbols for reasoning. For the long term accumulation of knowledge, the dictionary can be very large, and the vector

representation may be useful to properly represent the semantics of the words. This is like the particle/wave duality in quantum physics, where the one-hot representation is on the side of particle, and the dense vector embedding is on the side of wave.

## 8.15 Self-attention, Transformer, GPT, BERT

For a sentence that consists of words  $x_1, \dots, x_M$ , we can embed the words as  $h_1, \dots, h_M$ , and process them using LSTM or its multi-layer bi-directional versions. A more efficient and flexible sequence is to update the vectors by self-attention mechanism. Specifically, we can let  $(query, key, value)_m = Wh_m$ . Then we update  $h_m$  by  $h_m \leftarrow h_m + \Delta h_m$ , so that

$$\Delta h_m = \sum_{k=1}^M attention_{m \rightarrow k} \times value_k,$$

where

$$attention_{m \rightarrow k} = \frac{\exp(\langle query_m, key_k \rangle)}{\sum_{k'=1}^M \exp(\langle query_m, key_{k'} \rangle)}.$$

which involves soft-max attention based on the matching between query and key. We can also use multiple heads, so that each head has its own system of query, key and value, and then add up multiple heads. The above update can be iterated for multiple steps. Such a system was called Transformer, which was trained for machine translation with an encoder and a decoder, both involving self-attention mechanism. Later the decoder part is adopted by GPT (generative pre-training), and the encoder part is adopted by BERT (bi-directional encoder representations from transformers). BERT is trained by self-supervision task of filling in masked words in the sentences.

## 9 Representation Learning with Deep Nets

Similar to supervised learning such as kernel machines, boosting machines, and neural networks, which are based on  $f(x) = h(x)^\top \beta$ , in representation learning, we represent the input  $x$  by a representation  $h$ . However, without an output  $y$  to predict, we want the hidden vector  $h$  to capture key patterns and structures in  $x$ , and we want  $h$  to be simple, such as being of low dimension, being sparse, or having independent components, or being error resistant. We may think of  $h$  as an encoding of  $x$ , and we want to be able to decode  $x$  from the code  $h$ . We may also think of  $h$  as an embedding of  $x$  in an Euclidean space, and we want  $h$  to mirror the relationship between  $x$ , e.g., the relation between  $(h, h')$  mirrors the relation between  $(x, x')$ , especially for nearby  $x$  and  $x'$ .

### 9.1 Three lessons of deep learning

The following are three lessons offered by deep learning.

- (1) Multi-layer function approximator.
- (2) Vector representation. The vector can be interpreted as a code, a thought vector, or an embedding.
- (3) Learned computation. The inference or sampling algorithm can be approximated by a deep network.

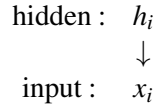
While (1) is most important and enables (2) and (3), (2) can be quite important in itself, and does not have to involve deep network. The representation does not need to be a vector. It can be a matrix or more structured object. (3) may be considered learning from the results of on-line computation, i.e., learning from internal data produced by on-line computation. It is like muscle memory.

## 9.2 Unsupervised learning

In unsupervised learning, the dataset is as below where  $y_i$  are not provided as supervision. In a generative

	input	hidden	output
1	$x_1^\top$	$h_1^\top$	?
2	$x_2^\top$	$h_2^\top$	?
...			
$n$	$x_n^\top$	$h_n^\top$	?

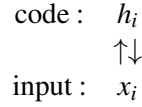
model, the vector  $h_i$  is not a vector of features extracted from the signal  $x_i$ .  $h_i$  is a vector of hidden variables that is used to generate  $x_i$ , as illustrated by the following diagram:



The components of the  $d$ -dimensional  $h_i$  are variably called factors, sources, components or causes.

## 9.3 Thought vector, auto-encoding, embedding, disentangling

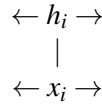
*Auto-encoder:*  $h_i$  is also called a code in the auto-encoder illustrated by the following diagram:



The direction from  $h_i$  to  $x_i$  is called the decoder, and the direction from  $x_i$  to  $h_i$  is called the encoder.

*Distributed representation and disentanglement:*  $h_i = (h_{ik}, k = 1, \dots, d)$  is called a distributed representation of  $x_i$ . Usually the components of  $h_i$ ,  $(h_{ik}, k = 1, \dots, d)$ , are assumed to be independent, and  $(h_{ik})$  are said to disentangle the variations in  $x_i$ .

*Embedding:*  $h_i$  can also be considered the coordinates of  $x_i$ , if we embed  $x_i$  in a low-dimensional space, as illustrated by the following diagram:



In the training data, we find a  $h_i$  for each  $x_i$ , so that  $\{h_i, i = 1, \dots, n\}$  preserve the relative relations between  $\{x_i, i = 1, \dots, n\}$ . The prototype example of embedding is multi-dimensional scaling, where we want to preserve the Euclidean distances between the examples.

## 9.4 Kullback-Leibler divergence

The Kullback-Leibler divergence from  $p$  to  $q$  is defined as

$$\text{KL}(p|q) = \mathbb{E}_p[-\log q(x)] - \mathbb{E}_p[-\log p(x)] = \mathbb{E}_p[\log(p(x)/q(x))].$$

According to Jensen inequality, for the concave function  $\log(x)$ ,

$$\mathbb{E}[\log(x)] \leq \log(\mathbb{E}(x)),$$

so

$$\mathbb{E}_p[\log(q(x)/p(x))] \leq \log \mathbb{E}_p[q(x)/p(x)] = \log \int \left[ \frac{q(x)}{p(x)} p(x) \right] dx = \log 1 = 0.$$

Thus  $\text{KL}(p|q) \geq 0$ .

The KL-divergence is not a metric or distance in that it does not satisfy the triangle inequality.

For conditional distribution, KL involves expectation with respect to the random variable being conditioned on:

$$\text{KL}(p(y|x)|q(y|x)) = \mathbb{E}_{p(x,y)} \left[ \log \frac{p(y|x)}{q(y|x)} \right] = \mathbb{E}_{p(x)} \mathbb{E}_{p(y|x)} \left[ \log \frac{p(y|x)}{q(y|x)} \right].$$

Let  $p(x)$  and  $q(x)$  be the marginal distributions, then

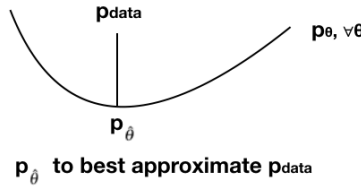
$$\begin{aligned} \text{KL}(p(x,y)|q(x,y)) &= \mathbb{E}_p \left[ \log \frac{p(x,y)}{q(x,y)} \right] \\ &= \mathbb{E}_p \left[ \log \frac{p(x)p(y|x)}{q(x)q(y|x)} \right] \\ &= \mathbb{E}_p \left[ \log \frac{p(x)}{q(x)} \right] + \mathbb{E}_p \left[ \log \frac{p(y|x)}{q(y|x)} \right] \\ &= \text{KL}(p(x)|q(x)) + \text{KL}(p(y|x)|q(y|x)). \end{aligned}$$

Two sources of KL-divergence are: (1) Large deviation, where the KL-divergence plays the role of the exponential rate of probability. (2) Coding theory, where the KL-divergence measures the coding redundancy.

KL divergence is adopted in information theory for redundancy measurement, in probabilistic theory for rate of probability measurement. And in statistics, it is used for maximum likelihood estimation.

### MLE vs KL divergence

Suppose we observe training examples  $x_i \sim p_{\text{data}}(x)$  independently for  $i = 1, \dots, n$ . For big  $n$ , we have  $\frac{1}{n} \sum_{i=1}^n f(x_i) \doteq \mathbb{E}_{p_{\text{data}}} [f(x)]$ . Consider a model  $p_{\theta}(x)$ . We can learn  $\theta$  by maximizing the log-likelihood  $L(\theta) = \frac{1}{n} \sum_{i=1}^n \log p_{\theta}(x_i)$ . As  $n \rightarrow \infty$ ,  $L(\theta) = \frac{1}{n} \sum_{i=1}^n \log p_{\theta}(x_i) \rightarrow \mathbb{E}_{p_{\text{data}}} [\log p_{\theta}(x)]$ . Since  $\text{KL}(p_{\text{data}}|p_{\theta}) = \mathbb{E}_{p_{\text{data}}} [\log p_{\text{data}}(x)] - \mathbb{E}_{p_{\text{data}}} [\log p_{\theta}(x)]$ ,  $\hat{\theta} = \arg \max_{\theta} L(\theta) = \arg \min_{\theta} \text{KL}(p_{\text{data}}|p_{\theta})$ . The maximum likelihood estimate can be seen as the most plausible explanation of the data.



### Inclusive and exclusion KL

For MLE, we minimize  $\text{KL}(p_{\text{data}}|p_{\theta})$ , which is expectation with respect to  $p_{\text{data}}$ .  $p_{\theta}$  seeks to cover all the modes of  $p_{\text{data}}$ . This will cause the mode inclusion behavior and the over-dispersion of the learned  $p_{\theta}$ .

Sometimes we want  $p_{\theta}$  to approximate a given target distribution,  $p_{\text{target}}$ , which we only need to know up to a normalizing constant. We can minimize  $\text{KL}(p_{\theta}|p_{\text{target}})$ , which is expectation with respect to  $p_{\theta}$ . This is variational approximation. Thus  $p_{\theta}$  can ignore some minor modes and only focus on major modes. This will cause the mode exclusion or mode collapsing behavior. Figure 38 illustrates the basic idea.



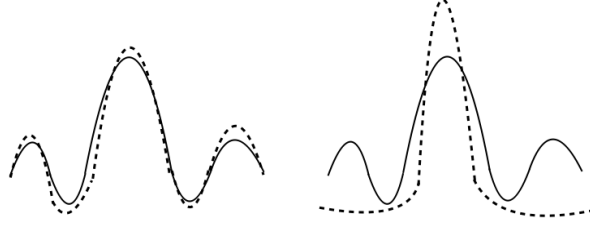


Figure 38: Left:  $\min_{\theta} \text{KL}(p_{\text{data}}|p_{\theta})$  vs Right:  $\min_{\theta} \text{KL}(p_{\theta}|p_{\text{target}})$ . The dotted line is  $p_{\theta}$ .

## 9.5 Decoder

Let  $h$  be the latent variables. We can decode it into an image  $x = g_{\theta}(h)$ , where  $g_{\theta}$  is a neural network with parameters  $\theta$ . This is in the reverse direction from neural network for unsupervised learning, which maps  $x$  to  $h$ . We call the neural network from  $x$  to  $h$  a bottom-up encoder network, and the network from  $h$  to  $x$  the top-down decoder network. While the bottom-up network may be a convolutional network, the top-down network is sometimes called the deconvolutional network.

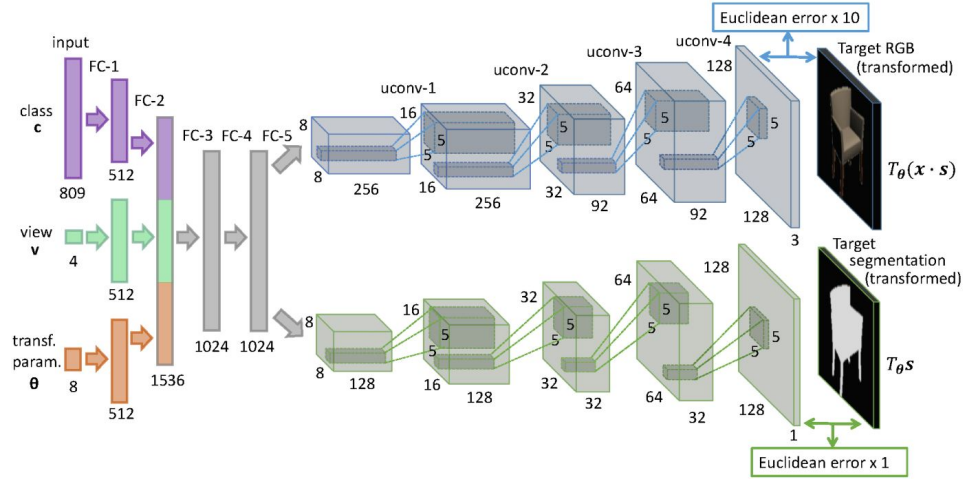


Figure 39: A top-down decoder network, where  $h$  consists of class (one-hot), view and transform parameters of chair.

Figure 39 shows the architecture of a decoder network, which is trained on chair images, where  $h$  consists of class, which is a one-hot vector, view and transform parameters, and  $x$  is the image of the corresponding chair. We can learn the decoder by minimizing  $|x - g_{\theta}(h)|^2$  if we have training data  $\{h_i, x_i\}$ . After training the decoder network, we can interpolate the  $h$  vectors of two different chairs, and generate the interpolated chair by the learned decoder. You can see that the learned network has amazing interpolative ability.

For image processing, such as style transfer, we can again use an encoder and decoder scheme. We can encode the input image into a thought vector. We can also encode the style as another vector. We can then concatenate the two vectors, and feed it into a decoder, to generate the output image.

## 9.6 Generative adversarial networks (GAN)

We can learn the decoder model in unsupervised manner, by treating  $h$  as a latent vector. We can also assume a simple known prior distribution on  $h$ ,  $h \sim p(h)$ . Then the decoder defines a generative model. We call such a model a generator model, which is a non-linear generalization of the factor analysis model.

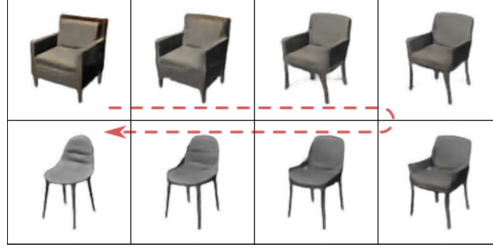


Figure 40: Interpolating the one-hot vectors of two types of chairs.

The model can be learned by generative adversarial networks (GAN), where we pair the generator model  $G$  with a discriminator model  $D$ , where for an image  $x$ ,  $D(x)$  is the probability that  $x$  is a true image instead of a generated image.

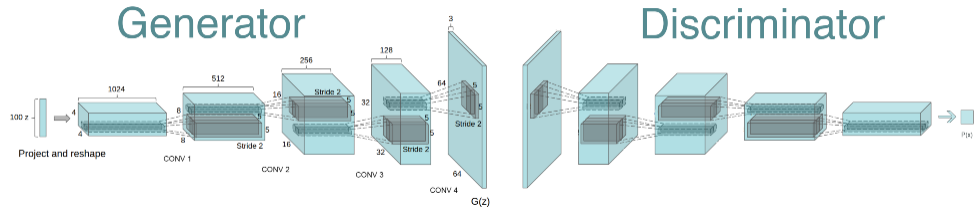


Figure 41: Generative adversarial networks (GAN) consists of a generator and a discriminator.

We can train the pair of  $(G, D)$  by an adversarial, zero-sum game. Specifically, let  $G(h) = g_\theta(h)$  be a generator. Let

$$V(D, G) = \mathbb{E}_{p_{\text{data}}} [\log D(X)] + \mathbb{E}_{h \sim p(h)} [\log(1 - D(G(h)))],$$

where  $\mathbb{E}_{p_{\text{data}}}$  can be approximated by averaging over the observed examples, and  $\mathbb{E}_h$  can be approximated by Monte Carlo average over the faked examples generated by the model.

Thus, we learn  $D$  and  $G$  by  $\min_G \max_D V(D, G)$ .  $V(D, G)$  is the log-likelihood for  $D$ , i.e., the log-probability of the real and faked examples. However,  $V(D, G)$  is not a very convincing objective for  $G$ . In practice, the training of  $G$  is usually modified into maximizing  $\mathbb{E}_{h \sim p(h)} [\log D(G(h))]$  to avoid the vanishing gradient problem.

For a given  $\theta$ , let  $p_\theta$  be the distribution of  $g_\theta(h)$  with  $h \sim p(h)$ . Assuming a perfect discriminator according to the Bayes rule  $D(x) = p_{\text{data}}(x) / (p_{\text{data}}(x) + p_\theta(x))$  (assuming equal numbers of real and fake examples). Then  $\theta$  minimizes Jensen-Shannon

$$\text{JSD}(p_{\text{data}} | p_\theta) = \text{KL}(p_\theta | p_{\text{mix}}) + \text{KL}(p_{\text{data}} | p_{\text{mix}}),$$

where  $p_{\text{mix}} = (p_{\text{data}} + p_\theta) / 2$ . As a result, GAN has mode collapsing behavior.

## 9.7 Variational auto-encoder (VAE) as alternating projection

The generator model is a decoder. We can pair it with an encoder.

The decoder model is  $h \sim p(h) \sim \mathcal{N}(0, I_d)$ , and  $[x|h] \sim p_\theta(x|h) \sim \mathcal{N}(g_\theta(h), \sigma^2 I)$ . It defines a joint distribution  $p_\theta(h, x) = p(h)p_\theta(x|h)$ .

The encoder model is  $[h|x] \sim q_\phi(h|x) \sim \mathcal{N}(\mu_\phi(x), V_\phi(x))$ , where  $V$  is a diagonal matrix. Together with  $p_{\text{data}}(x)$ , we have a joint distribution  $q_\phi(h, x) = p_{\text{data}}(x)q_\phi(h|x)$ .

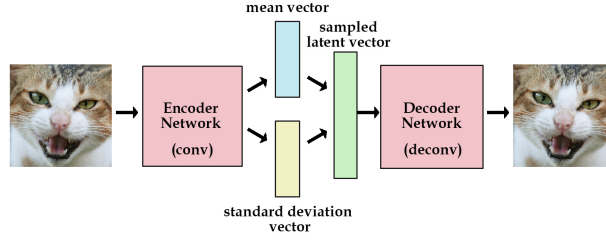


Figure 42: VAE: The decoder is a top-down latent variable model. The encoder approximates the posterior distribution of latent vector given observed image.

We learn both the decoder  $\theta$  and encoder  $\phi$  by

$$\min_{\theta, \phi} \text{KL}(q_{\phi}(h, x) | p_{\theta}(h, x)).$$

The expectation with respect to  $q_{\phi}(h|x)$  can be based on the re-parametrization

$$h = \mu_{\phi}(x) + V_{\phi}^{1/2} z, \quad z \sim \mathcal{N}(0, I_d),$$

so that the expectation is with respect to  $z$ .

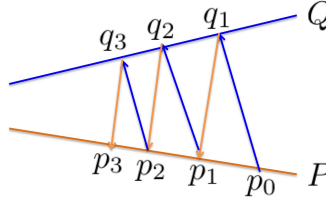


Figure 43: VAE as alternating projection.

Let  $Q = \{q_{\phi}(h, x), \forall \phi\}$  and let  $P = \{p_{\theta}(h, x), \forall \theta\}$ . The VAE problem is to  $\min_{p \in P, q \in Q} \text{KL}(p|q)$ . Starting from  $p_0$ , the VAE iterates the following two steps:

- (1)  $q_{t+1} = \arg \min_{q \in Q} \text{KL}(q|p_t)$ .
- (2)  $p_{t+1} = \arg \min_{p \in P} \text{KL}(q_{t+1}|p)$ .

This is alternating projection. The minimization can also be replaced by gradient descent.

The wake-sleep algorithm is the same as VAE in Step (2). However, in Step (1), it is  $q_{t+1} = \arg \min_{q \in Q} \text{KL}(p_t|q)$ , where we generate dream data from  $p_t$ , and learn  $q_{\phi}(h|X)$  from the dream data.

### Maximum likelihood vs variational, bias and regularization

The VAE objective function

$$\text{KL}(q_{\phi}(h, x) | p_{\theta}(h, x)) = \text{KL}(p_{\text{data}}(x) | p_{\theta}(x)) + \text{KL}(q_{\phi}(h|x) | p_{\theta}(h|x)).$$

The first term  $\text{KL}(p_{\text{data}}(x) | p_{\theta}(x))$  is the objective function of the maximum likelihood. The VAE divergence is an upper bound of the MLE divergence, which is computationally intractable because  $p_{\theta}(x) =$

$\int p_\theta(h, x) dh$  is an intractable integral. The VAE objective function is tractable as long as the encoder or the inference model  $q_\phi(h|x)$  is in closed form.

The accuracy of VAE relative to MLE is determined by the second divergence, which governs the learning of the inference model, i.e., given  $\theta$ ,

$$\hat{\phi}_{\text{VAE}} = \arg \min_{\phi} \text{KL}(q_\phi(h|x) | p_\theta(h|x)).$$

Since  $q_\phi$  is on the left side of KL-divergence, i.e., the exclusive KL, it tends to chase the major mode of  $p_\theta$  while ignoring the minor modes.

While  $q_\phi(h|x)$  seeks to get closer to  $p_\theta(h|x)$  in learning  $\phi$ , in the learning of  $\theta$ , the model  $p_\theta(h|x)$  tends to bias itself from MLE so that  $p_\theta(h|x)$  becomes closer to  $q_\phi(h|x)$ . This bias is actually beneficial to the inference model  $q_\phi$ , and the bias induced by  $q_\phi$  also provides some regularization of the model.

### Variational inference vs maximum a posteriori (MAP)

The maximum a posteriori (MAP) estimate of  $h$  is

$$\hat{h}_{\text{MAP}} = \arg \max_h \log p_\theta(h|x).$$

$q_{\hat{\phi}}(h|x)$  is close to the point mass at  $\hat{h}_{\text{MAP}}$  because of the aforementioned mode chasing behavior. The difference is that  $q_{\hat{\phi}}(h|x)$  is a probability distribution that account for the uncertainty of  $p_\theta(h|x)$ . In fact, the variational divergence is

$$\text{KL}(q_\phi(h|x) | p_\theta(h|x)) = -H(q_\phi(h|x)) - \mathbb{E}_{q_\phi(h|x)}[\log p_\theta(h|x)],$$

where

$$H(q_\phi(h|x)) = -\mathbb{E}_{q_\phi(h|x)}[\log q_\phi(h|x)]$$

is the entropy of  $q_\phi(h|x)$ . The variational approximation is to maximize the posterior  $\mathbb{E}_{q_\phi(h|x)}[\log p_\theta(h|x)]$  like MAP, but it also seeks to maximize the entropy  $H(q_\phi(h|x))$ . For variational inference,  $H(q_\phi(h|x))$  should be in closed form.

In learning  $\theta$ , we should not use  $\hat{h}_{\text{MAP}}$  to infer  $h$  because it does not account for uncertainty.  $\hat{h}_{\text{MAP}}$  amounts to overfitting. It explains away too much of  $x$  so that there is not much left for  $\theta$  to explain. As a result, we cannot learn  $\theta$  accurately, and the learned  $\theta$  cannot generate realistic  $x$  because it can generate realistic  $x$  only for those preferred  $h$  that are similar to  $\hat{h}_{\text{MAP}}$ .

### Local vs global variational

There is a slight ambiguity in  $\text{KL}(q(h|x) | p(h|x))$ . We can interpret it as the KL-divergence with  $x$  fixed, or with  $x$  averaged out by  $p_{\text{data}}$ . In the previous subsection, it is the former. In VAE, it is the latter.

In VAE,  $\mathbb{E}_{p_{\text{data}}}$  is approximated by averaging over the training data  $(x_i, i = 1, \dots, n)$ , so that  $\phi$  is estimated by

$$\min_{\phi} \frac{1}{n} \sum_{i=1}^n \text{KL}(q_\phi(h_i|x_i) | p_\theta(h_i|x_i)).$$

Here  $\phi$  is shared by all  $(x_i, i = 1, \dots, n)$ , and it is the global variational parameter. For each  $x_i$ ,  $q_\phi(h_i|x_i) \sim \text{N}(\mu_\phi(x_i), V_\phi(x_i))$ .

We may also estimate  $(\mu_i, V_i)$  for each  $x_i$  by

$$\min_{\mu_i, V_i} \text{KL}(q_{\mu_i, V_i}(h_i|x_i) | p_\theta(h_i|x_i)),$$

where  $q_{\mu_i, V_i}(h_i|x_i) \sim N(\mu_i, V_i)$ .  $(\mu_i, V_i)$  is called the local variational parameter.

For the local variational parameter, we need an iterative inference algorithm to compute it for each  $x_i$ . This can be time consuming. For the global parameter, we learn an inference network to map  $x_i$  directly to  $(\mu_i, V_i)$ . This amounts to distilling the inference algorithm to the inference network, i.e., the inference network or the inference model is the learned computation.

Traditional variational methods employ local parameters, while VAE uses global parameters by taking advantage of the flexibility of neural nets.

## 9.8 VAE and EM

In the EM algorithm, we assume  $q_\phi(h|x)$  to be infinitely flexible. In the alternating projection of VAE,

- (1)  $q_{t+1} = \arg \min_{q \in Q} \text{KL}(q|p_t)$ , so that  $q_{t+1}(h|x) = p_{\theta_t}(h|x)$ .
- (2)  $p_{t+1} = \arg \min_{p \in P} \text{KL}(q_{t+1}|p)$ , i.e.,

$$\begin{aligned} \theta_{t+1} &= \arg \min_{\theta} \text{KL}(p_{\text{data}} p_{\theta_t}(h|x) | p_{\theta}(h, x)) \\ &= \arg \max_{\theta} \mathbb{E}_{p_{\text{data}}} \mathbb{E}_{p_{\theta_t}(h|x)} [\log p_{\theta}(h, x)]. \end{aligned}$$

The EM algorithm requires that  $p_{\theta}(h|x)$  is in closed form, which means  $p_{\theta}(x)$  should be in closed form.  $q$  is the distribution of data.  $p_{\text{data}}$  is the distribution of observed data  $x$ , and  $q_{\phi}(h|x)$  is the multiple imputation of the missing data  $h$ .  $q_{\phi}(h, x)$  is the distribution of the complete data  $(h, x)$ .

$p$  is the distribution of the model.  $p(h)p_{\theta}(x|h) = p_{\theta}(h, x)$  is the complete-data model, and  $p_{\theta}(x) = \int p_{\theta}(h, x) dh$  is the observed data model.

## 9.9 Flow-based model

Similar to generator model, in the flow-based model, we have  $h \sim N(0, I_D)$ , and  $x = T_{\theta}(h)$ . The difference is that  $h$  has the same dimensionality as  $x$ , which is  $D$ , and  $T_{\theta}$  is a one-to-one differentiable transform. Suppose  $T_{\theta}$  maps a small region  $\Delta h$  around  $h$  to a small region  $\Delta x$  around  $x$ . Then

$$q(h)|\Delta h| = p(x)|\Delta x|,$$

where  $q(h)$  is the density of  $h$ , and  $p(x)$  is the density of  $x$ .  $|\Delta h|$  is the volume of the small region  $\Delta h$ , and  $\Delta x$  is the volume of the small region  $\Delta x$ . To understand this point, you can imagine  $q(h)$  as a cloud of points in the space of  $h$ , and  $p(x)$  is the cloud of points in the space of  $x$ . Then the points in  $\Delta h$  are mapped to the points in  $\Delta x$ , and the number of points in  $\Delta h$  is the same as the number of points in  $\Delta x$ .

$$|\Delta x|/|\Delta h| = |\det(T'_{\theta}(h))|,$$

where  $T_{\theta}(h) = dx/dh^{\top}$  is the  $D \times D$  Jacobian matrix, and  $|\det(T')|$  is the absolute value of the determinant of Jacobian. Locally  $T$  is a linear transformation, and  $T'$  is the matrix of this linear transformation. For any matrix  $A$ , it maps the cube  $[0, 1]^D$  to a parallelogram formed by the column vectors of  $A$ . The volume of this parallelogram is the determinant of  $A$ . Thus the density  $p(x)$  can be obtained in closed form. This is different from the generator model, where the marginal density involves integrating out  $h$ .

We can design  $T$  to consists of a sequence (or a flow) of simple transformations that are invertible and whose Jacobians can be easily computed. For instance, consider a transformation  $x \rightarrow y$ . We can let  $x = (x_1, x_2)$ , where  $x_1$  is  $d_1$ -dimensional, and  $x_2$  is  $d_2 = D - d_1$  dimensional. We can let  $y_1 = x_1$ , and  $y_2 = \mu(x_1) + \sigma(x_1)x_2$ , where  $\mu(x_1)$  is a  $d_2$  dimensional vector and  $\sigma(x_1)$  is a  $d_2$  dimensional diagonal matrix.

The flow-based model can be used as a generative model. It can also be used as an inference model.

## 10 Representation Learning without Deep Nets

### 10.1 Factor analysis and generalizations

The generator model in GAN and VAE can be traced back to the factor analysis model.

#### The model

The model is as follows:

$$x_i = Wh_i + \varepsilon_i,$$

for  $i = 1, \dots, n$ , where  $W$  is a  $p \times d$  dimensional matrix ( $p$  is the dimensionality of  $x_i$  and  $d$  is the dimensionality of  $h_i$ ), and  $\varepsilon_i$  is a  $p$ -dimensional residual vector. The following are the interpretations of  $W$ :

(1) Loading matrix: Let  $W = (w_{jk})_{p \times d}$ .  $x_{ij} \approx \sum_{k=1}^d w_{jk} h_{ik}$ , i.e., each component of  $x_i$ ,  $x_{ij}$ , is a linear combination of the latent factors.  $w_{jk}$  is the loading weight of factor  $k$  on variable  $j$ .

(2) Basis vectors: Let  $W = (W_k, k = 1, \dots, d)$ , where  $W_k$  is the  $k$ -th column of  $W$ .  $x_i \approx \sum_{k=1}^d h_{ik} W_k$ , i.e.,  $x_i$  is a linear superposition of the basis vectors ( $W_k$ ), where  $h_{ik}$  are the coefficients.

(3) Matrix factorization:  $(x_1, \dots, x_n) \approx W(h_1, \dots, h_n)$ , where the  $p \times n$  matrix  $(x_1, \dots, x_n)$  is factorized into the  $p \times d$  matrix  $W$  and the  $d \times n$  matrix  $(h_1, \dots, h_n)$ .

In factor analysis, we have  $h_i \sim N(0, I_d)$ ,  $x_i = Wh_i + \varepsilon_i$ ,  $\varepsilon_i \sim N(0, \sigma^2 I_p)$ , and  $\varepsilon_i$  is independent of  $h_i$ . The dimensionality of  $h_i$ , which is  $d$ , is smaller than the dimensionality of  $x_i$ , which is  $p$ . The factor analysis is very similar to the principal component analysis (PCA), which is a popular tool for dimension reduction. The difference is that in factor analysis, the column vectors of  $W$  do not need to be orthogonal to each other.

The factor analysis model originated from psychology, where  $x_i$  consists of the test scores of student  $i$  on  $p$  subjects.  $h_i$  consists of the verbal intelligence and the analytical intelligence of student  $i$  ( $d = 2$ ). Another example is the decathlon competition, where  $x_i$  consists of the scores of athlete  $i$  on  $p = 10$  sports, and  $h_i$  consists of athlete  $i$ 's speed, strength and endurance ( $d = 3$ ).

#### Generalizing prior assumption: ICA, sparse coding, NMF, recommender, K-means

In independent component analysis, we assume  $p = d$ , and  $h_k \sim p_k$ , which is a heavy tailed distribution. It is also called blind source separation, where  $h_k$  are the sources mixed by  $W$ .

In sparse coding, we assume  $d > p > d_0$ , where  $d_0$  is the number of  $h_k$  that are non-zero.  $W = (W_k, k = 1, \dots, d)$  forms a dictionary, and the sparse  $h$  selects the most meaningful words from this dictionary.

In non-negative matrix factorization, we assume  $h_k \geq 0$ , to emphasize the fact that the parts should add positively.

In a recommender system,  $(x_1, \dots, x_n) \approx W(h_1, \dots, h_n)$ , where  $x_{ij}$  is user  $i$ 's rating of movie  $j$ , and  $x_{ij} = \langle w_j, h_i \rangle$ , where  $w_j$  is the  $j$ -row of  $W$ . We can interpret  $h_i$  as user  $i$ 's desires in various aspects, and  $w_j$  as movie  $j$ 's desirabilities in the corresponding aspects.  $x_i$  are incomplete, but we can complete  $x_i$  by learning  $h_i$  and  $w_j$  from the observed data, so that we can recommend user  $i$  the movies that may receive high ratings from him or her.

If  $h_i$  is a one-hot vector, then  $x_i = Wh_i + \varepsilon_i$  means that  $x_i = W_k + \varepsilon_i$ , where  $k$  is the element of  $h_i$  that is 1, and  $W_k$  is the  $k$ -th column of  $W$ . This is the K-means model.

The sparse vector lies in between one-hot vector and dense vector.

#### Generalizing linear transformation to non-linear transformation

The factor analysis model is  $x = Wh + \varepsilon$  (here we drop the subscript  $i$  for convenience). We can generalize it to  $x = g_\theta(h) + \varepsilon$ , where  $g_\theta$  can be parametrized by a deep network. This is the decoder model or the

generator model in GAN and VAE.

## 10.2 K-means as one-hot vector representation

Let  $(x_i, i = 1, \dots, n)$  be the observed data. The K-means method partition the data into  $K$  different clusters. Each cluster has a mean  $\mu_k, k = 1, \dots, K$ . The K means algorithm iterates the following two steps:

(1) Assignment. We assign each  $x_i$  to a cluster  $k$  by minimizing  $|x_i - \mu_k|^2$  over  $k$ .

(2) Update. For each cluster  $k$ , we compute  $\mu_k$  as the mean of the examples  $x_i$  that belong to cluster  $k$ .

K-means can be considered a special case of encoding, where for each  $x_i$ , the corresponding  $h_i$  is one hot, i.e., only one component of  $h_i$  is 1, and the other components are all 0.

Word2vec is the opposite of K-means, where we encode one-hot vector (word) to a dense vector.

## 10.3 Spectral embedding and clustering

The spectral embedding and clustering method starts from an adjacency matrix

$$A_{ij} = \exp\left(-\frac{1}{2\sigma^2}|x_i - x_j|^2\right),$$

which measures the similarity between  $i$  and  $j$ .  $A$  is an  $n \times n$  matrix with elements  $(A_{ij})$ , where we assume  $A_{ii} = 0$  on the diagonal.  $A_{ij}$  does not care about distant relationship because it is practically zero if  $x_i$  is far away from  $x_j$ .

The spectral embedding is to find  $h_i$  for each  $x_i$ , by minimizing

$$\sum_{i \neq j} A_{ij} |h_i - h_j|^2 = 2\text{trace}(H^\top (D - A)H),$$

where  $D_i = \sum_j A_{ij}$ , and  $D$  is the diagonal matrix whose  $i$ -th diagonal element is  $D_i$ .  $H^\top = (h_i, i = 1, \dots, n)$ . trace is the sum of the diagonal elements.  $H$  can be obtained by the smallest  $d$  eigen vectors of  $D - A$ , where  $d$  is the dimensionality of  $h_i$ .

The basic idea is that the embedding  $(h_i)$  should capture the nearby relations, while ignoring distant relations. If  $A_{ij}$  is big, we want  $h_i$  to be close to  $h_j$ .

The spectral clustering method is similar to spectral embedding. It computes the Laplacian

$$L = D^{-1/2}AD^{-1/2},$$

and finds  $H^\top = (h_i, i = 1, \dots, n)$  as the largest  $d$  eigen vectors of  $L$ . Then perform K-means on  $(h_i, i = 1, \dots, n)$ . Such clustering pays more attention to nearby relations, while ignoring distant relations, because the definition of  $A$ .  $L_{ij} = A_{ij} / \sqrt{D_i D_j}$ , where we normalize the adjacency by  $D$ , i.e., a close friendship to a popular person with a lot of good friends is not really as close as it appears to be.

## 10.4 Multi-dimensional scaling as mirroring

The earliest embedding method is multi-dimensional scaling, which seeks to preserve the Euclidean distance by

$$\min_{\{h_i\}} \sum_{ij} (|x_i - x_j| - |h_i - h_j|)^2.$$

Later embedding methods focus more on preserving the adjacencies of nearby points, i.e., the embedding lets friends stay close, but is not concerned with relations between strangers.

## 10.5 t-Stochastic neighborhood embedding (tSNE) as energy-based model

tSNE seeks to mirror the friendship relationships. For each example  $i$ , we view other examples from its perspective, and define

$$p_{j|i} = \exp\left(-\frac{1}{2\sigma_i^2}|x_i - x_j|^2\right) / \sum_{k \neq i} \exp\left(-\frac{1}{2\sigma_i^2}|x_i - x_k|^2\right).$$

We may choose  $\sigma_i$  so that it distinguishes  $\{x_j, j \neq i\}$  in terms of their relationships to  $x_i$ .

For  $(h_i, i = 1, \dots, n)$  in the embedded space, we mirror  $(x_i, i = 1, \dots, n)$  by defining

$$q_{j|i} = \exp\left(-\frac{1}{2\sigma_i^2}|h_i - h_j|^2\right) / \sum_{k \neq i} \exp\left(-\frac{1}{2\sigma_i^2}|h_i - h_k|^2\right).$$

Both  $p$  and  $q$  are conditional distributions of  $j$  given  $i$ , i.e., if  $i$  feels lonely and wants to call someone, then  $j$  comes to his or her mind with probability  $p_{j|i}$  or  $q_{j|i}$ .

We may optimize the embedding by

$$\min_{\{h_i\}} \text{KL}(p|q) = \min_{\{h_i\}} \sum_{ij} (\log p_{j|i} - \log q_{j|i}).$$

In tSNE, we replace the normal density in  $q_{j|i}$  by the  $t$  distribution or the Cauchy distribution, so that  $h_i$  can be more spread out. We also symmetrize  $p_{j|i}$  by defining  $p_{ij} = (p_{j|i} + p_{i|j})/2n$ , and we define

$$q_{ij} = (1 + |h_i - h_j|^2)^{-1} / \sum_{k \neq l} (1 + |h_k - h_l|^2)^{-1}.$$

We then find  $h = \{h_i\}$  by minimizing

$$L(h) = \text{KL}(p|q) = \sum_{ij} p_{ij} (\log p_{ij} - \log q_{ij}).$$

Now  $p$  and  $q$  are joint distributions of  $(i, j)$ , i.e., if we want to sample a pair of friends, we get  $(i, j)$  with probability  $p_{ij}$  or  $q_{ij}$ .

$L(h)$  is actually the negative log-likelihood, and  $h$  is the parameter.  $p$  serves as the data distribution, and  $q$  is the model distribution.  $q$  is actually an energy-based model of the form

$$q_{\theta}(x) = \frac{1}{Z(\theta)} \exp[f_{\theta}(x)],$$

where  $\theta = h$ , and  $x = (i, j)$ , and

$$f_{\theta}(x) = -\log(1 + |h_i - h_j|^2),$$

It can be shown that

$$-\frac{\partial}{\partial \theta} \text{KL}(p|q_{\theta}) = \mathbb{E}_p \left[ \frac{\partial}{\partial \theta} f_{\theta}(x) \right] - \mathbb{E}_q \left[ \frac{\partial}{\partial \theta} f_{\theta}(x) \right].$$

For tSNE,

$$\frac{\partial f}{\partial h_i} = -2(1 + |h_i - h_j|^2)^{-1}(h_i - h_j).$$

The gradient is

$$\frac{\partial L}{\partial h_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij})(1 + |h_i - h_j|^2)^{-1}(h_i - h_j).$$

The factor 4 is due to the symmetry between  $(i, j)$  and  $(j, i)$ . We can obtain  $(h_i)$  by gradient descent.

The basic idea is that  $(p_{ij})$  captures the nearby relations among  $(x_i)$ , and  $(q_{ij})$  captures the nearby relations among  $(h_i)$ , while allowing  $(h_i)$  to be scattered out according to the heavy-tailed  $t$  distribution. We want to find  $(h_i)$  so that  $Q$  approaches  $P$ .



## 10.6 Local linear embedding (LLE)

LLE seeks to mirror the local linear relationship.

First, for each  $i$ , we find its neighbors  $j$ , and we denote  $j \sim i$ .

Then for each  $i$ , we extract the local linear relationship by minimizing

$$\min_{(w_{ij})} \left| x_i - \sum_{j \sim i} w_{ij} x_j \right|^2.$$

We impose the constraint  $\sum_j w_{ij} = 1$ , so that the learned  $w_{ij}$  is invariant with the shift  $x_i \leftarrow x_i + c$  for a constant  $c$ .

Then we find  $\{h_i\}$  by minimizing

$$\min_{(h_i)} \sum_i \left| h_i - \sum_{j \sim i} w_{ij} h_j \right|^2,$$

which can be minimized by eigen computation.

Unlike tSNE, where the mirroring is in explicitly defined friendship relationship, in LLE, the mirroring is in the form of minimization.

## 10.7 Topic model and latent Dirichlet allocation (LDA)

The topic model is used to analyze documents, e.g., news reports or medical papers, to identify the topics and their key words. Each report may contain multiple topics, and each topic generates a distribution of words. Suppose there are  $K$  topics. For a document, let  $\rho = (\rho_k, k = 1, \dots, K)$  be the probability distribution of topics. Even though  $K$  may be large, only a small number of topics have big probabilities. Suppose  $f_k(x)$  be the distribution of word  $x$  in each topic. Even though the number of words is large, only a small number of words have big probabilities. Then the overall distribution

$$f(x) = \sum_k \rho_k f_k(x),$$

which can be written in the matrix factorization form.

In latent Dirichlet allocation, we assume that  $\rho$  follows a Dirichlet distribution. Given  $\rho$ , the topic follows a multinomial distribution. Given the topic  $k$ , the words follow a multinomial distribution according to  $f_k$ . We assume that the words are unordered in the document, i.e., each document is a bag of words. The model can be written as  $\rho \rightarrow h \rightarrow x$ , where  $h$  is the one-hot hidden vector for topic, and  $x$  is the one-hot vector for word. There are two layers of latent vectors that generate each word in each document, namely  $\rho$  at the document level, and  $h$  at the word level.

The model can be learned by local variational method, with variational inference models  $q(h)$  and  $q(\rho)$ , by minimizing  $\text{KL}(q(h)q(\rho)|p(h, \rho|x))$ , where we assume the inference distributions  $q(h)$  and  $q(\rho)$  to be independent. This leads to the mean field method where we update the variational parameters for  $q(h)$  and  $q(\rho)$  alternatively. It is similar to the Gibbs sampler for sampling  $p(h, \rho|x)$  by alternating the sampling of  $p(h|\rho, x)$  and  $p(\rho|h, x)$ . The sampling in MCMC is replaced by optimizing the variational parameters. For each word  $x$ ,  $q(h)$  tells us the topic. For each document,  $q(\rho)$  tells us the distribution of the topics.

## 11 Reinforcement Learning

The reinforcement learning lies between supervised learning and unsupervised learning. In supervised learning, the representation  $h(x)$  is used to predict  $y$ . In unsupervised learning, the representation  $h$  is used to

encode and mirror  $x$ . In reinforcement learning, the representation  $h$  is used to predict the action or value. In imitation learning, the action is observed, so the learning is supervised. In reinforcement learning, the action is not observed, and the training signal is in the form of cumulated reward or value.

## 11.1 Alpha Go

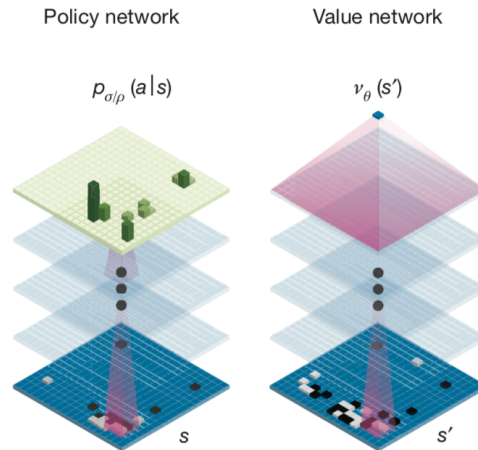


Figure 44: Policy network and value network. The input is a  $19 \times 19$  image. The policy network is a classification network. The value network is a regression network.

Alpha Go is a good starting point to learn reinforcement learning. Let  $s$  be the current state, which is a  $19 \times 19$  image. Let  $a$  be the action, i.e., where to place the stone. There are  $19 \times 19$  choices (although some are forbidden by the rule). We want to decide the action  $a$ .

### Game tree and minimax solution

Starting from the initial state, i.e., an empty board, the black and white players alternatively place black and white stones on the board. This generates a game tree, whose breadth and depth are both big. We can find minimax solution by back-propagation from the leaf nodes, so that each player chooses the moves that maximizes its final result, i.e., a win in this case.

For Go, it is impractical to go through the whole game tree. We may use Monte Carlo tree search (MCTS) to go over the promising moves. But pure MCTS is still quite challenging to implement. We use policy network to reduce the breadth of the tree search, and the value network to reduce the depth of tree search.

### Supervised learning or behavior cloning

We can learn a policy network  $p_{\sigma}(a|s)$ , with parameter  $\sigma$  (stands for supervised) from the data  $\{(s, a)\}$  collected from human players. See Figure 44 for an illustration of the policy network. This is a classification problem and is a supervised learning problem. It is also called behavior cloning. The learning rule is

$$\Delta\sigma \propto \frac{\partial}{\partial\sigma} \log p_{\sigma}(a|s),$$

which maximizes the log-likelihood  $\log p_{\sigma}(a|s)$  over  $\sigma$ .

We can also learn a simpler roll out network to be used in Monte Carlo tree search.

## Reinforcement learning by policy gradient

After learning  $p_\sigma(a|s)$ , we can learn another policy network  $p_\rho(a|s)$  by reinforcement learning. Starting from  $\rho = \sigma$ , we let  $p_\rho$  play against  $p_\sigma$ , until the end. Let  $z \in \{+1, -1\}$  be whether  $\rho$  wins the game. We update  $\rho$  by

$$\Delta\rho \propto \frac{\partial}{\partial\rho} \log p_\rho(a|s)z.$$

The above updating rule is called policy gradient. It is similar to the above maximum likelihood updating rule, except for the following two aspects: (1) the derivative of the log-likelihood is weighted by the reward  $z$ . (2) the action  $a$  is generated by the current policy instead of human expert.

The above updating rule maximizes  $\mathbb{E}_\rho[z]$ . It is called REINFORCE algorithm. We will justify this algorithm later on.

## Value network

After learning  $p_\rho(a|s)$ , we can let  $\rho$  play with itself from a random starting state  $s$ , until we reach the end and get  $z$ . We then update the value network  $v_\theta(s)$  by

$$\Delta\theta \propto \frac{\partial}{\partial\theta} [z - v_\theta(s)]^2.$$

See Figure 44 for an illustration of the value network.

The learned  $v$  can also be used as a baseline for policy gradient

$$\Delta\rho \propto \frac{\partial}{\partial\rho} \log p_\rho(a|s)[z - v_\theta(s)],$$

where  $v_\theta(s)$  serves to reduce the variance of the gradient. Here  $v_\theta$  is the critic, and  $p_\rho$  is the actor.

## Monte Carlo tree search

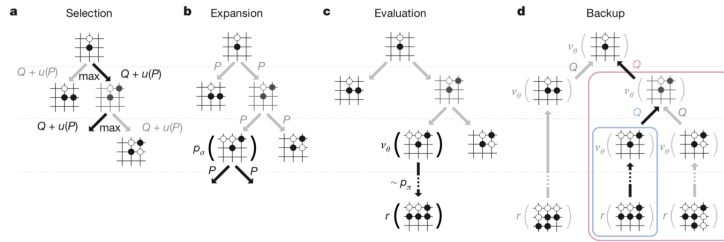


Figure 45: Monte Carlo tree search grows a tree by repeating the following four steps: selection, expansion, evaluation, and backup.

After learning the policy networks  $p_\sigma$ ,  $p_\rho$ , and value network  $v_\theta$ , we can use either the policy network or the value network to play the game. But this will not work well. Instead we use them to help us look ahead and plan the next action using Monte Carlo tree search (MCTS). Specifically, we want to estimate  $Q(s, a)$ , the value of action  $a$  at state  $s$ .

Treating the current state as the root, MCTS grows a tree by repeating the following four steps: selection, expansion, evaluation, and backup. For each node  $s$  of the tree, and each action  $a$  from this node, we record the number of visits  $N(s, a)$  and the action value  $Q(s, a)$ . During each pass of MCTS, starting from the root

state, we go down the tree until we get to a leaf node. At each non-leaf node  $s$ , we choose an action  $a$  that balances  $Q(s, a)$  and  $N(s, a)$ . We want to choose  $a$  with a high  $Q(s, a)$  for exploitation, meanwhile we also want to choose  $a$  with low  $N(s, a)$  for exploration. When we come to a leaf node, we expand the tree using all the possible moves from this node. Then we choose an expanded node, and use a roll out policy to play the game until the end. Finally we backup the roll out result. For all the branches  $(s, a)$  we go through in this pass of MCTS, we increase the visit count  $N(s, a)$  by 1, and increase or decrease the total value of  $(s, a)$  by 1 according to the roll out result.  $Q(s, a)$  is then updated as the current total value divided by the current visit count. MCTS is expected to converge to the minimax solution to the game.

The policy network can help guide the selection step. For each node  $s$ , we select the action according to  $Q(s, a) + Cp_\sigma(a|s)/\sqrt{N(s, a)}$  where the constant  $C$  balances the exploration and exploitation. We use the supervised  $p_\sigma$  instead of reinforcement learning  $p_\rho$ , because  $p_\sigma$  is more diverse.

The value network can help avoid the reliance on roll out. For an expanded node  $s$ , instead of rolling out to the end, and then backup, we can simply backup  $v_\theta(s)$  by adding the total value of a traveled branch  $(s, a)$  the value  $v_\theta(s)$ . The Alpha Go uses a linear combination of roll out result from  $s$  and  $v_\theta(s)$ . Since  $s$  is closer to the end of the game than the root node, the value  $v_\theta(s)$  can be a more precise estimate than the value at the root node.

After many passes of MCTS, at the current root node  $s$ , we choose  $a$  with the maximal  $Q(s, a)$ . We can also choose  $a$  with the maximum  $N(s, a)$ .

The MCTS is a planning process. Its purpose is to select the next move  $a$  from the current root node  $s$ . After we make the move  $a$  in real game play, we can then discard the tree. When we need to choose the next move in real game play, we start to grow another tree.

## 11.2 Alpha Go Zero

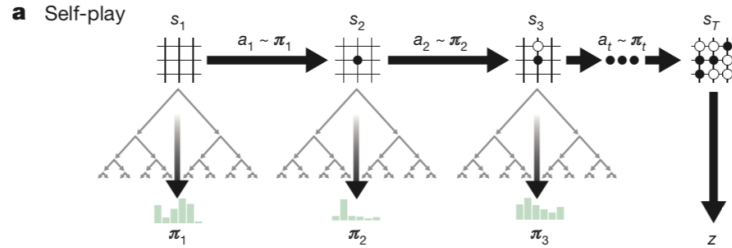


Figure 46: Alpha Go Zero. In self-play, we use MCTS to plan each move, where MCTS is guided by policy and value networks. After the game is over, the game result is used to train the policy and value networks.

In Alpha Go, after learning the policy and value networks, we will not update it, and they are used for MCTS for playing the game. This is a big waste, because when we use MCTS to play the game, the results of the game play should be used to further update the policy and value networks. In fact, the policy and value networks can be learned solely from the games where the moves are planned by MCTS.

In Alpha Go Zero, the learning of  $p_\sigma$ ,  $p_\rho$  and  $v_\theta$  before MCTS is discarded. Instead,  $p$  and  $v$  are learned from the results of the self-play, and  $p$  and  $v$  in turn guides the MCTS in planning each move of the self-play. Specifically, we iterate the following two steps:

(1) Given  $p$  and  $v$ , play a game where each move is planned by MCTS. Specifically, at each state  $s$  of the game, we grow a tree in order to evaluate  $N(s, a)$  and  $Q(s, a)$  (in fact,  $N(s, a)$  and  $Q(s, a)$  are computed for all the nodes in the tree). We let  $\pi(a|s) \propto N(s, a)^{1/\tau}$ , where  $\tau$  is the temperature parameter. We discard the tree and use  $\pi(a|s)$  to select the move  $a$  to play the game. We play the game until the end.

(2) We use  $\pi(a|s)$  to train the policy  $p(a|s)$ . We use the end result  $z \in \{+1, -1\}$  to train the value network  $v(s)$  for all the states of the game.

In Alpha Go Zero, the policy and value networks share the same body, which consists of many residual blocks. Each network has a head. Such a design makes sense because the two networks are consistent with each other.

In Alpha Go and Zero, the policy network is like impulse or habit, and the value network is like gut feeling. They are like emotions. The Monte Carlo tree search is to think through and plan. The reason we have consciousness is that we need to know our emotions (as well as perceptions, i.e., what we see and hear) in order to plan our actions.

### 11.3 Atari by Q learning

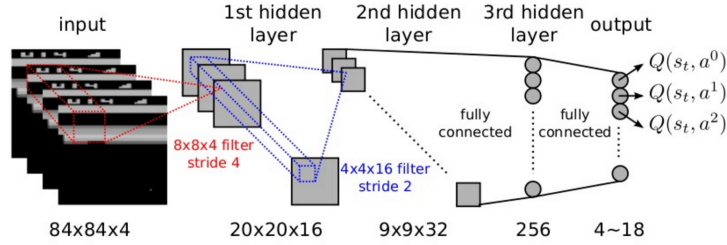


Figure 47: Atari. The action value network  $Q_\theta(s, a)$ .

Atari is a video game. It is different from Go in the following two aspects. (1) In Atari we have immediate rewards in terms of points earned at each step, while in Go the reward is delayed to the end. (2) In Atari, a player does not know the dynamics or the detailed rules that drive the game, although the player can play out the game to observe the change of the state and the reward earned, while in Go, a player knows the rule of game.

We can learn to play Atari by Q learning, by learning the Q function  $Q_\theta(s, a)$  by playing out the game.  $Q(s, a)$  is the cumulative reward to go, i.e., the total increase of score if we take action  $a$  at the current state  $s$ , and then play out the game to the best of our ability. Suppose the cumulative reward to go is  $R$ . Then we can update  $\theta$  by

$$\Delta\theta \propto -\frac{\partial}{\partial\theta}(R - Q_\theta(s, a))^2..$$

In order to obtain  $R$ , we need to play out the game by taking the best action at each step. However, this will be time consuming. After taking the action  $a$ , we get the immediate reward  $r$ , and get to the next state  $s'$ . Then we estimate  $R$  by  $r + \gamma \max_{a'} Q_\theta(s', a')$ , where  $\max_{a'} Q_\theta(s', a')$  is the cumulative reward to go at state  $s'$  if we do our best.  $\gamma$  is the discount factor, usually taken to be close to 1. Then we can use the estimated  $R$  to update  $\theta$ .

The above method of predicting  $R$  is referred to as a bootstrap method, where we use the current model  $Q_\theta(s, a)$  to help predict  $R$ . The difference between the predicted  $R$  and  $Q_\theta(s, a)$  is called temporal difference error.

At each  $s$ , how do we take the action  $a$ ? The greedy policy is to take  $a$  that maximize  $Q_\theta(s, a)$ . Usually we use the  $\epsilon$ -greedy policy, where with  $\epsilon$  probability, we take a random action, and with probability  $1 - \epsilon$ , we take the greedy action.

Q-learning is a model free method, where we do not need to know the dynamics  $s_{t+1} = f(s_t, a_t)$  and the reward function  $r_t = r(s_t, a_t)$ , i.e., we do not know  $f()$  and  $r()$ , but at state  $s_t$ , we can try the action  $a_t$  to observe  $s_{t+1}$  and  $r_t$ .

## 11.4 Markov decision process (MDP)

### Model

The model has two parts.

Part 1: The dynamic model is  $p(s_{t+1}|s_t, a_t)$ . In the deterministic case,  $s_{t+1} = f(s_t, a_t)$ .

Part 2: The reward model is  $p(r_t|s_t, a_t, s_{t+1})$ . In the deterministic case,  $r_t = r(s_t, a_t, s_{t+1})$ , as well as  $\tilde{r}(s_T)$  if there is a terminating state.

The dynamics model and the reward model are Markovian, in that we do not need to know the past history before  $(s_t, a_t)$ .

The trajectory is  $\tau = ((s_t, a_t), t = 0, \dots)$ .

The return is

$$R(\tau) = \sum_{t=0}^{T-1} r(s_t, a_t, s_{t+1}) + \tilde{r}(s_T).$$

For infinite horizon (i.e., no terminal state),

$$R = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}),$$

where  $\gamma$  is the discount factor. It is introduced so that  $R$  is finite. It also makes practical sense because we are more certain about near future and want to collect rewards sooner.

### Policy and value

A policy is  $\pi(a_t|s_t)$ . Again it is Markovian. For deterministic policy,  $a_t = \pi(s_t)$ .

Together with dynamics, the distribution of a trajectory  $\tau$  is

$$p(\tau) = \prod_{t=0}^{T-1} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t).$$

Define the return to go at time  $t$  to be

$$R_t(\tau) = \sum_{k=0}^{\infty} \gamma^k r_{t+k},$$

where  $r_t = r(s_t, a_t, s_{t+1})$ .

The state value is

$$v^\pi(s) = \mathbb{E}_{p(\tau)}[R_t | s_t = s],$$

The state-action value (or quality) is

$$Q^\pi(s, a) = \mathbb{E}_{p(\tau)}[R_t | s_t = s, a_t = a].$$

Usually we write  $\mathbb{E}_{p(\tau)} = \mathbb{E}_\pi$  because the distribution of trajectory depends on  $\pi$ , with the dynamics fixed.

The optimal value is

$$v^*(s) = \max_{\pi} v^\pi(s).$$

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a).$$

A policy  $\pi^*$  is optimal if  $v^{\pi^*}(s) \geq v^\pi(s)$  for all  $\pi$  and all  $s$ .

For MDP, one can always find a deterministic policy to be optimal.

## 11.5 Model-based planning: play out in imagination

We can learn the dynamic model  $p(s_{t+1}|s_t, a_t)$  and the reward model  $p(r_t|s_t, a_t, s_{t+1})$ , and plan the optimal sequence of actions  $(a_t, t = 0, \dots, T-1)$  to maximize the total reward. This is model-based planning or optimal control. It is usually solved by dynamic programming. The value and policy are involved in the calculation of dynamic programming. The Monte Carlo tree search is model-based planning.

In the computation of model-based planning, we may generate trajectories in our mind by imagining playing out a sequence of actions. For instance, the Monte Carlo tree search is computed in the mind, not in real play. After Monte Carlo tree search, we only play the first move in the real game.

## 11.6 Model-free learning: play out in real life

In model-free learning, we do not learn the dynamic and the reward models. Instead we take actions  $a_t$  at  $s_t$  in real life, and observe  $s_{t+1}$  and collect  $r_t$ .

In value-based learning such as Q-learning, we estimate  $Q(s, a)$ , and then use greedy policy that maximize  $Q(s, a)$  in testing.

In policy-based learning such as policy gradient, we roll out the trajectory and update the policy.

Model-based learning requires a small amount of data to learn the dynamic and reward models. Planning does not require more data. The policy and value are computed from simulated data in the mind.

Model-free learning requires more data to estimate the value or policy from real life experience. We need to repeatedly play out the actions in real life and observe the trajectories and rewards.

## 11.7 Temporal difference bootstrap

To reduce the data requirement in model-free learning, we do not need to play out the actions until the end. In order to obtain the total return of the whole trajectory, we may play out for a few steps, and then stop early, and use the current value function to estimate the rest of the return. The value function is expected to be more accurate as we are closer to the end. This is the temporal difference bootstrap estimate. This is a bit like imaginary playing out in model-based method, i.e., at the early stop, we ask ourselves: if we continue to play until the end, what would be the return?

## 11.8 Q learning

The Q-learning is model-free value-based learning using the bootstrap idea. Let  $Q_\theta(s, a)$  be the value network. Let  $s'$  be the state observed after playing  $a$  at state  $s$ . Let  $R$  be the return if we play out as best as we can. Then we can update  $\theta$  by

$$\Delta\theta \propto \frac{\partial}{\partial\theta} [R - Q_\theta(s, a)]^2.$$

We use the bootstrap method to get

$$R = r + \gamma \max_{a'} Q_\theta(s', a').$$

At  $s$ , how should we choose  $a$ ? Usually we use  $\epsilon$ -greedy policy, i.e., with probability  $1 - \epsilon$ , we take  $a = \arg\max_a Q_\theta(s, a)$ , and with probability  $\epsilon$ , we take random action. This is the off-policy approach, i.e., in the learning stage, we do not take the greedy policy according to  $Q$ , because we need to explore other actions. After learning, we can switch to the greedy policy in testing.

Q-learning is essentially dynamic programming, except that  $r$  is observed in real play, i.e., it is empirical version of dynamic programming.

## 11.9 REINFORCE, MLE, re-parametrization

For a policy network  $\pi_\theta(a|s)$ , if we play out this policy in real life, we can obtain a trajectory  $\tau$ , and the distribution of  $\tau$  is

$$p_\theta(\tau) = \prod_t \pi(a_t|s_t) p(s_{t+1}|s_t, a_t).$$

The return is  $R(\tau) = \sum_t \gamma^t r_t$ , where  $r_t = r(s_t, a_t, s_{t+1})$ .

We want to find  $\theta$  to maximize  $\mathbb{E}_{p(\tau)}[R(\tau)]$ .

### REINFORCE

The gradient is

$$\begin{aligned} \frac{\partial}{\partial \theta} \mathbb{E}_{p_\theta(\tau)}[R(\tau)] &= \int R(\tau) \frac{\partial}{\partial \theta} p_\theta(\tau) d\tau \\ &= \int \left[ R(\tau) \frac{\partial}{\partial \theta} \log p_\theta(\tau) \right] p_\theta(\tau) d\tau \\ &= \mathbb{E}_{p_\theta(\tau)} \left[ R(\tau) \frac{\partial}{\partial \theta} \log p_\theta(\tau) \right]. \end{aligned}$$

The stochastic gradient (or empirical gradient) is to sample  $\tau$  by playing out the policy  $\pi_\theta$ , and update  $\theta$  by

$$\Delta \theta \propto R(\tau) \frac{\partial}{\partial \theta} \log p_\theta(\tau).$$

This is the REINFORCE algorithm.

### REINFORCE vs maximum likelihood

If we observe an expert who plays out his policy to generate a trajectory, we can learn  $\theta$  by maximum likelihood, and the stochastic gradient is

$$\Delta \theta \propto \frac{\partial}{\partial \theta} \log p_\theta(\tau).$$

The differences between REINFORCE and maximum likelihood are as follows.

- (1) In REINFORCE,  $\tau$  is generated by the self policy. In MLE,  $\tau$  is generated by the expert.
- (2) In REINFORCE, there is  $R(\tau)$  in the gradient. In MLE, there is no  $R(\tau)$  term.

If we set  $R(\tau) = c$ , a constant, then

$$\mathbb{E}_{p_\theta(\tau)} \left[ c \frac{\partial}{\partial \theta} \log p_\theta(\tau) \right] = \frac{\partial}{\partial \theta} \mathbb{E}_{p_\theta(\tau)}[c] = 0.$$

This means one cannot learn from himself.

### REINFORCE vs re-parametrization

In VAE, the inference model  $q_\phi(h|x)$  is like a policy, where  $x$  is the state, and  $h$  is the action. We want to maximize  $\mathbb{E}_{q_\phi(h|x)}[R(h)]$  for a function  $R(h)$ . We can use REINFORCE algorithm to optimize  $\phi$ . In the parametrization trick, we write  $h = \mu_\phi(x) + V_\phi(x)^{1/2}e$ , where  $e \sim \mathcal{N}(0, I_d)$ , so that  $\mathbb{E}_{q_\phi(h|x)}[R(h)] = \mathbb{E}_e[R(\mu_\phi(x) + V_\phi(x)^{1/2}e)]$ , which can be maximized by gradient ascent.



Mapping the situation to reinforcement learning, if we can reparametrize the policy  $p_\theta(\tau)$  by  $\tau = \pi_\theta(e)$ , where  $e \sim p(e)$  independent of  $\theta$ , and  $\pi$  is now a deterministic function, then we can maximize  $\mathbb{E}_e[R(\pi_\theta(e))]$ , whose gradient is  $\mathbb{E}_e\left[\frac{\partial}{\partial\theta}R(\pi_\theta(e))\right]$ . This is the same as the REINFORCE gradient  $\mathbb{E}_{p_\theta(\tau)}\left[R(\tau)\frac{\partial}{\partial\theta}\log p_\theta(\tau)\right]$ . However,  $\text{Var}_e\left[\frac{\partial}{\partial\theta}R(\pi_\theta(e))\right]$  can be much smaller than  $\text{Var}_{p_\theta(\tau)}\left[R(\tau)\frac{\partial}{\partial\theta}\log p_\theta(\tau)\right]$ . In the former, we directly improve each sampled trajectory  $\tau = \pi_\theta(e)$ . In the latter, however, we improve the probability of those sampled trajectories that have large  $R$ . For instance, suppose you want to learn how to shoot the basket ball. In the reparametrization method, you try to improve each attempt by aiming carefully. In REINFORCE, you randomly shoot the ball blindly, hoping that you may get lucky occasionally, and then you learn from these lucky attempts. As a result, you need to try a lot of times.

## 11.10 Policy gradient: actor and critic

### Independent of dynamics

The gradient is

$$\begin{aligned}\frac{\partial}{\partial\theta}\mathbb{E}_{p_\theta(\tau)}[R(\tau)] &= \mathbb{E}_{p_\theta(\tau)}\left[R(\tau)\frac{\partial}{\partial\theta}\log p_\theta(\tau)\right] \\ &= \mathbb{E}_{p_\theta(\tau)}\left[\left(\sum_{t=0}^{\infty} r_t\right)\frac{\partial}{\partial\theta}\log\left(\prod_{t=0}^{\infty}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t)\right)\right] \\ &= \mathbb{E}_\theta\left[\left(\sum_{t=0}^{\infty} r_t\right)\left(\sum_{t=0}^{\infty}\frac{\partial}{\partial\theta}\log\pi_\theta(a_t|s_t)\right)\right] \\ &= \mathbb{E}_\theta\left[\sum_{t=0}^{\infty}\left(\frac{\partial}{\partial\theta}\log\pi_\theta(a_t|s_t)\sum_{t'=0}^{\infty} r_{t'}\right)\right],\end{aligned}$$

where we ignore the discount factor for simplicity.

### Independent of past

$$\begin{aligned}\frac{\partial}{\partial\theta}\mathbb{E}_\theta[R] &= \mathbb{E}_\theta\left[\sum_{t=0}^{\infty}\left(\frac{\partial}{\partial\theta}\log\pi_\theta(a_t|s_t)\sum_{t'=0}^{\infty} r_{t'}\right)\right] \\ &= \sum_{t=0}^{\infty}\mathbb{E}_\theta\left[\left(\frac{\partial}{\partial\theta}\log\pi_\theta(a_t|s_t)\left(\sum_{t'=0}^{t-1} r_{t'} + \sum_{t'=t}^{\infty} r_{t'}\right)\right)\right] \\ &= \sum_{t=0}^{\infty}\mathbb{E}_\theta\left[\frac{\partial}{\partial\theta}\log\pi_\theta(a_t|s_t)(R_t + C)\right] \quad \text{where } C = \sum_{t'=0}^{t-1} r_{t'} \\ &= \sum_{t=0}^{\infty}\mathbb{E}_\theta\left[\frac{\partial}{\partial\theta}\log\pi_\theta(a_t|s_t)R_t\right] + \sum_{a_t}\pi_\theta(a_t|s_t)\left[\frac{\partial}{\partial\theta}\log\pi_\theta(a_t|s_t)C\right] \\ &= \sum_{t=0}^{\infty}\mathbb{E}_\theta\left[\frac{\partial}{\partial\theta}\log\pi_\theta(a_t|s_t)R_t\right] + \frac{\partial}{\partial\theta}\left[C\sum_{a_t}\pi_\theta(a_t|s_t)\right] \\ &= \sum_{t=0}^{\infty}\mathbb{E}_\theta\left[\frac{\partial}{\partial\theta}\log\pi_\theta(a_t|s_t)R_t\right] + \frac{\partial}{\partial\theta}C \\ &= \sum_{t=0}^{\infty}\mathbb{E}_\theta\left[\frac{\partial}{\partial\theta}\log\pi_\theta(a_t|s_t)R_t\right]\end{aligned}$$

where

$$R_t = \sum_{t'=t}^{\infty} r_{t'}$$

is the reward to go after action  $a_t$ , whereas  $\sum_{t'=0}^{t-1} r_{t'}$  is constant relative to  $a_t$ , because  $a_t$  can only cause change in the future, but it cannot change the past. Therefore we can remove it as a baseline.

### Advantage

$$\frac{\partial}{\partial \theta} \mathbb{E}_{\theta}[R] = \sum_{t=0}^{\infty} \mathbb{E}_{\theta} \left[ \frac{\partial}{\partial \theta} \log \pi_{\theta}(a_t | s_t) (R_t - V(s_t)) \right]$$

for any baseline  $V(s_t)$ , which cannot be changed by  $a_t$ . Here we take

$$V(s_t) = V^{\pi}(s_t) = \mathbb{E}[R_t | s_t],$$

which is the value of  $s_t$ , averaged over all possible actions  $a_t \sim \pi(a|s_t)$ .

$$A_t = R_t - V(s_t)$$

is called the advantage of taking action  $a_t$  relative to other possible actions at  $s_t$ . Subtracting  $V(s_t)$  as a baseline helps reduce the variance of the gradient. Again consider learning a policy to shoot the basket ball. If you are very close to the basket and you hit the basket, it is not a big deal. But if you are beyond the three-pointer arc and you hit the basket, you should learn from such attempts.

When computing the gradient, we fix  $\pi$  at the current policy.

We can learn  $\theta$  by stochastic gradient

$$\Delta \theta \propto \sum_{t=0}^{\infty} \left[ \frac{\partial}{\partial \theta} \log \pi_{\theta}(a_t | s_t) (R_t - V_{\alpha}(s_t)) \right]$$

where we run the policy  $\pi_{\theta}$  to the end. We also need to learn a value network  $V_{\alpha}(s)$  to approximate  $V^{\pi}(s)$ . The above update pushes the policy to favor the above average actions.

The policy  $\pi_{\theta}(a|s)$  is called an actor. The value  $V_{\alpha}(s)$  is called a critic.

### Temporal difference

We may also estimate  $R_t$  based on

$$V^{\pi}(s_t) = \mathbb{E}[R_t] = \mathbb{E} \left[ \sum_{k=0}^n r_{t+k} + V^{\pi}(s_{t+n+1}) \right].$$

Thus we can estimate  $R_t$  without running the policy to the very end, and we estimate  $R_t$  by

$$\hat{R}_t = \sum_{k=0}^n r_{t+k} + V_{\alpha}(s_{t+n+1}),$$

which consists of two parts:

- (1) Monte Carlo unrolling in real life: run the policy  $\pi_{\theta}$  for  $n$  steps to accumulate  $\sum_{k=0}^n r_{t+k}$ .
- (2) Bootstrapping in imagination: prediction the rest of the accumulated reward by the current value network  $V_{\alpha}(s_{t+n+1})$ .

This is a temporal difference scheme.

Thus we can use gradient descent

$$\Delta\theta \propto \sum_{t=0}^{\infty} \left[ \frac{\partial}{\partial\theta} \log \pi_{\theta}(a_t|s_t) \left( \sum_{k=0}^n r_{t+k} + V_{\alpha}(s_{t+n+1}) - V_{\alpha}(s_t) \right) \right].$$

Meanwhile we can update  $\alpha$  by

$$\Delta\alpha \propto \left[ \sum_{k=0}^n r_{t+k} + V_{\alpha}(s_{t+n+1}) - V_{\alpha}(s_t) \right] \frac{\partial V}{\partial\alpha},$$

which seeks to approach the fixed point

$$V^{\pi}(s_t) = \mathbb{E} \left[ \sum_{k=0}^n r_{t+k} + V^{\pi}(s_{t+n+1}) \right]$$

by gradient descent on

$$[\hat{R}_t - V_{\alpha}(s_t)]^2.$$

The above is not a real loss function because the target  $\hat{R}_t = \sum_{k=0}^n r_{t+k} + V_{\alpha}(s_{t+n+1})$  is based on the policy network  $V_{\alpha}$  itself, and this bootstrapped target keeps changing. Without Monte Carlo, it becomes a self-fulfilling prophecy, and there is nothing for the value network to learn.

Q learning is similar to the learning of the value network  $V_{\alpha}(s)$ .

### 11.11 Partially observed MDP (POMDP)

Instead of directly observing  $s$ , we may observe  $o \sim \rho(o|s)$ , which is the observation model. Together with the dynamics, it becomes a hidden Markov model or state space model. We can update the belief  $b_t = p(s_t | o_{\leq t})$  by Bayes rule. In POMDP, we treat the belief as the state. Then it can still be formulated as a MDP.

### 11.12 Multi-agent reinforcement learning

The Alpha Go is a two agent RL. In multi-agent RL, we need to define policy and value for each player. In the cooperative setting, the players have a joint value function to maximize. In the competitive setting, the players play zero-sum or non-zero sum game. In real life, the situation can be both cooperative and competitive.

### 11.13 Inverse reinforcement learning (IRL)

In inverse reinforcement learning, an expert demonstrates  $(s_t, a_t)$ . We can learn a policy  $\pi_{\theta}(a|s)$  by supervised learning from the demonstrations by maximum likelihood. This is called behavior cloning. We can also learn a reward or value function  $r_{\theta}(s, a)$  from demonstrations. This is called inverse reinforcement learning. The value function turns to be more generalizable than the policy.

### 11.14 Energy-based model

An energy-based model is of the following form

$$p_{\theta}(x) = \frac{1}{Z(\theta)} \exp(f_{\theta}(x)),$$

where  $Z(\theta) = \int \exp(f_\theta(x))$  is the normalizing constant. This model originated from statistical mechanics, where  $-f_\theta(x)$  is called the energy function of  $x$ .

Suppose we observe  $x_i \sim p_{\text{data}}(x)$  independently for  $i = 1, \dots, n$ . The log-likelihood is

$$l(\theta) = \frac{1}{n} \sum_{i=1}^n \log p_\theta(x_i),$$

whose gradient is

$$l'(\theta) = \frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial \theta} f_\theta(x_i) - \mathbb{E}_\theta \left[ \frac{\partial}{\partial \theta} f_\theta(x) \right] \rightarrow \mathbb{E}_{p_{\text{data}}} \left[ \frac{\partial}{\partial \theta} f_\theta(x) \right] - \mathbb{E}_\theta \left[ \frac{\partial}{\partial \theta} f_\theta(x) \right],$$

and  $\mathbb{E}_\theta$  can be approximated by Monte Carlo samples from  $p_{\theta_t}(x)$ .

IRL is a good context to introduce the energy-based model, where  $x$  is the action (we make the state implicit for simplicity), and  $f_\theta(x)$  is the reward or value of action  $x$ .

The generator model or the flow model can be considered the policy model or actor, which can generate the samples directly, whereas the energy-based model can be considered a critic or evaluator. Sampling from energy-based model usually requires iterative Markov chain Monte Carlo (MCMC). It is also related to optimal control or planning, where  $f_\theta(x)$  is an objective function.

Suppose we have a policy model  $\pi_\alpha(x)$  such as a flow model, we can train it together with the energy-based model  $p_\theta$  by adversarial contrastive divergence

$$\min_{\theta} \max_{\alpha} [\text{KL}(p_{\text{data}} | p_\theta) - \text{KL}(\pi_\alpha | p_\theta)],$$

where  $\pi_\alpha$  is the actor and  $p_\theta$  is the critic.

## 12 Causal Learning

In supervised learning, we learn to predict the outcome  $y$  based on the input  $x$ . Usually we assume that training data  $(x, y)$  and the testing data  $(x, y)$  come from the same distribution. In causal learning, the testing data may not come from the same distribution as the training data. In this case, can we still learn to predict  $y$  from  $x$ ? This is the problem of causal learning.

### 12.1 Counterfactual, potential outcome

Suppose we want to know whether smoking causes lung cancer. Let  $X \in \{0, 1\}$  be whether a person smokes or not. Let  $Y \in \{0, 1\}$  be whether the person get lung cancer or not. Let  $Y_1$  be the outcome if the person smokes. Let  $Y_0$  be the outcome if the person does not smoke. Then  $Y_1$  vs  $Y_0$  is the causal effect of  $X$  on this person. We may also average over the whole population and define  $\mathbb{E}[Y_1] - \mathbb{E}[Y_0]$  as the population version of the causal effect.

The problem is, we cannot observe both  $Y_1$  and  $Y_0$ . We can only observe either  $Y_1$  or  $Y_0$  in reality based on the value of  $X$ , i.e.,  $Y = Y_X$ .  $Y_1$  and  $Y_0$  are called potential outcomes, and we can speak both of them only in the counterfactual tense.

The causal effect  $\mathbb{E}[Y_1] - \mathbb{E}[Y_0]$  may be different from  $\mathbb{E}[Y|X=1] - \mathbb{E}[Y|X=0]$ , if  $X$  depends on  $(Y_1, Y_0)$ . For instance, if  $Y_1 = 0$  and  $Y_0 = 1$ , we assign  $X = 1$ . If  $Y_1 = 1$  and  $Y_0 = 0$ , we assign  $X = 0$ . Then we may draw the conclusion that smoking may prevent lung cancer. In experimental design,  $X$  is randomized, so that  $X$  does not depend on  $(Y_1, Y_0)$ . In observational studies, there may be hidden variables  $h$  that affect both  $X$  and  $(Y_1, Y_0)$ , and as a result,  $X$  depends on  $(Y_1, Y_0)$ . For instance, if more old people smoke than young

people, then even if smoking has no effect on lung cancer, it may appear that smoking causes lung cancer. Here  $h$  is age.

Both Fisher and Neyman made important contributions to the issue of causality. But Rubin is the first one to formalize the conceptual framework of causal inference based on counterfactual potential outcomes.

## 12.2 Matching and propensity score

In the above example, we may divide people into different age groups, and compare smokers and non-smokers within the same age group. That is, we match on the variable age. In general, we should match on as many variables as we can. We can also just match on propensity score  $\rho(h) = \Pr(X = 1|h)$ , assuming  $h$  includes all the background variables. For people with the same propensity score, then clearly the assignment of  $X$  is randomized, so that we can compare smokers and non-smokers within the same group.

## 12.3 Structural equations, causal diagram

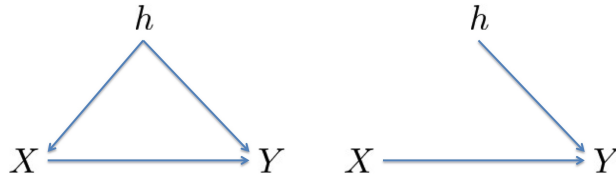


Figure 48: Causal diagram. Left: training. Right: testing. Right is called graph surgery, which is related to counterfactual thinking or intervention.

We can also treat this problem using Pearl's structural equations. (1)  $h = f_h(\epsilon_h)$ ; (2)  $X = f_X(h, \epsilon_X)$ ; and (3)  $Y = f_Y(X, h, \epsilon_Y)$ . These equations can be represented by a causal diagram, where there is an arrow from  $h$  to  $X$ , and there are arrows from both  $h$  and  $X$  to  $Y$ .

Compared to Rubin's potential outcome, each person is defined by  $(\epsilon_h, \epsilon_Y)$ , then  $Y_1 = f_Y(1, h, \epsilon_Y)$ ,  $Y_0 = f_Y(0, h, \epsilon_Y)$ . Then  $X = f_X(h, \epsilon_X)$  may depend on  $(Y_1, Y_0)$ .

We can define  $\mathbb{E}[Y_1]$  as  $\mathbb{E}[Y|X = do(1)] = \mathbb{E}[f_Y(1, h, \epsilon_Y)]$ , and  $\mathbb{E}[Y_0]$  as  $\mathbb{E}[Y|X = do(0)] = \mathbb{E}[f_Y(0, h, \epsilon_Y)]$ . In terms of causal diagram, this means cut the arrow from  $h$  to  $X$ , and set  $X$  to 1 or 0.

## 12.4 Training data and testing data

The training data  $\{X, Y\}$  is generated by

$$h = f_h(\epsilon_h), X = f_X(h, \epsilon_X), Y = f_Y(X, h, \epsilon_Y).$$

The testing data  $\{X, Y\}$  is generated by

$$h = f_h(\epsilon_h), X = x, Y = f_Y(X, h, \epsilon_Y).$$

The testing data is generated by a different computer program than the training data. The conditional distribution for the testing dataset  $[Y|X = x]$  is defined as  $p(y|do(x))$ .

The problem is, given training data, can we predict testing data  $[Y|X = x]$ , i.e.,  $p(y|do(x))$ ?

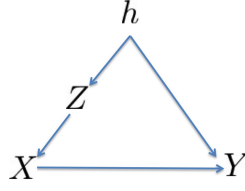


Figure 49: Back door

## 12.5 Do calculus, back door and front door adjustment

For the back-door adjustment,

$$p(y|do(x)) = \sum_z p(y|do(x), z) p(z|do(x)) = \sum_z p(y|x, z) p(z),$$

because  $p(y|do(x), z) = p(y|x, z)$ , and  $p(z|do(x)) = p(z)$ . We cut the arrow from  $z$  to  $x$ .

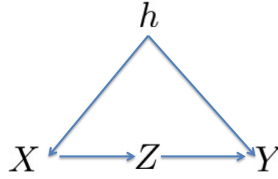


Figure 50: Front door

For the front-door adjustment,

$$\begin{aligned} p(y|do(x)) &= \sum_z p(y|z, do(x)) p(z|do(x)) \\ &= \sum_z p(z|x) p(y|do(z)) \\ &= \sum_z p(z|x) \sum_{x'} p(y|z, x') p(x'), \end{aligned}$$

because  $p(z|do(x)) = p(z|x)$  and  $p(y|z, do(x)) = p(y|do(z))$ , which can be calculated by back door.

The above are part of do-calculus.

## 13 Background: Convex Optimization and Duality

SVM brought convex optimization to machine learning. We will give a systematic treatment to this topic. The easiest starting point is to write the primal problem as a min-max problem, and then change it to the dual max-min problem, which is often easier to solve. We also give other geometric interpretations.

### 13.1 von Neumann minimax theorem

#### von Neumann's game theory

There are two players, X and Y. Assume the loss function is  $F(x, y)$  from X's perspective, where  $x$  is X's action and  $y$  is Y's action. From Y's perspective, his or her loss function is  $-F(x, y)$ . This is a zero sum

game.

For each action  $x$  of  $X$ , the worse case scenario is  $\max_y F(x, y)$ .  $X$  wants to minimize the loss of the worse case, i.e.,  $\min_x \max_y F(x, y)$ . Similarly,  $Y$  wants to  $\max_y \min_x F(x, y)$ . The solution to the game exists if

$$\min_x \max_y F(x, y) = \max_y \min_x F(x, y).$$

### Convex-concave function and saddle point

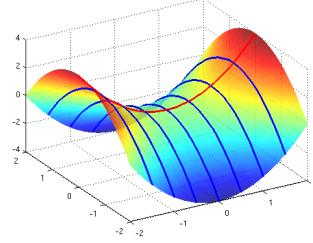


Figure 51: Saddle point of convex-concave function

$F(x, y)$  is a convex-concave function if  $F(x, y)$  is convex in  $x$  given any fixed  $y$ , and  $F(x, y)$  is concave in  $y$  given any fixed  $x$ . For example,  $F(x, y) = x^2 - y^2$  is such a convex-concave function. For convex-concave  $F(x, y)$ , the solution to the above min-max and max-min problem exists, and is the saddle point. If  $(x_*, y_*)$  is the saddle point of  $F(x, y)$ , then  $F(x, y_*) \geq F(x_*, y_*) \geq F(x_*, y)$ . See Figure 51.

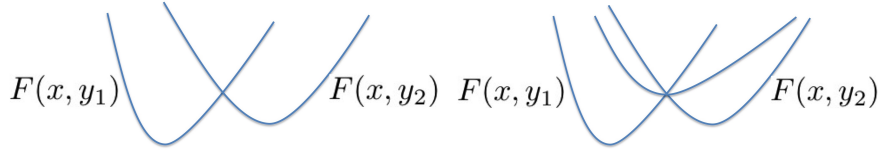


Figure 52: The functions  $F(x, y_i)$  live on different slices of  $y_i$ . We project them onto the same plane. Left:  $\min\text{-max} \geq \max\text{-min}$ . Right: For convex-concave  $F$ ,  $\max\text{-min} \geq \min\text{-max}$ .

The existence of the solution can be understood as follows. For  $F(x, y)$ , we can consider a discrete set of  $y$ , i.e.,  $\{y_1, \dots, y_n\}$ . We can then plot the curves  $F(x, y_i)$  for  $i = 1, \dots, n$ . Each  $F(x, y_i)$  is a function of  $x$ . Then  $\max_i F(x, y_i)$  is the upper envelop of these functions.  $\min_x \max_i F(x, y_i)$  is the minimum of this upper envelop. Now consider  $\min_x F(x, y_i)$ , it is the minimum of the curve  $F(x, y_i)$ . All the  $n$  minima are below the upper envelop.  $\max_i \min_x F(x, y_i)$  is the maximum of these  $n$  minima, and is also below the upper envelop. Thus  $\min_x \min_i F(x, y_i) \geq \max_i \min_x F(x, y_i)$ .

Now we show that the converse is also true for convex-concave function. Consider  $F(x, y_1)$  and  $F(x, y_2)$ . Suppose we let  $y$  move on the line segment from  $y_1$  to  $y_2$ . Suppose  $y = \lambda y_1 + (1 - \lambda) y_2$  for  $\lambda \in [0, 1]$ . Because  $F(x, y)$  is concave in  $y$ , we have

$$F(x, y) \geq \lambda F(x, y_1) + (1 - \lambda) F(x, y_2) \geq \min_{i \in \{1, 2\}} F(x, y_i),$$

where  $\min_{i \in \{1, 2\}} F(x, y_i)$  is the lower envelop. Let  $m(y) = \min_x F(x, y)$  be the minimum of  $F(x, y)$  for each  $y$ . As  $y$  moves from  $y_1$  to  $y_2$ , the minimum must be above the lower envelop. Since  $F(x, y)$  is convex for

each  $y$ , thus at a certain point  $y_0$ , we must have  $m(y_0)$  to be above  $\min_x \max_{i \in \{1,2\}} F(x, y_i)$ , which is on the lower envelop. Thus we have  $\min_x \min_i F(x, y_i) \geq \max_i \min_x F(x, y_i)$ .

Thus we have min-max = max-min.

## 13.2 Constrained optimization and Lagrange multipliers

### Rewrite constrained optimization as min-max

The primal problem is

$$\text{minimize } f(x), \text{ subject to } g(x) \leq 0.$$

We can form the Lagrangian

$$L(x, \lambda) = f(x) + \lambda g(x), \lambda \geq 0.$$

The primal problem then becomes

$$\min_x \max_{\lambda \geq 0} L(x, \lambda).$$

Even though we remove the constraint  $g(x) \leq 0$  in the above min-max problem, the constraint is automatically satisfied. This is because for all  $x$  such that  $g(x) > 0$ , we can let  $\lambda \rightarrow \infty$ , so that  $L(x, \lambda) \rightarrow \infty$ . Thus the solution to the min-max problem must satisfy the constraint  $g(x) \leq 0$ .

### Complementary slackness

If  $(x, \lambda)$  is the solution to the above min-max problem, then we must have

$$\lambda g(x) = 0.$$

This is because  $\lambda g(x) \leq 0$ , since  $g(x) \leq 0$  and  $\lambda \geq 0$ .

If  $g(x) < 0$  strictly, then  $\lambda = 0$ . If  $\lambda > 0$ , then we must have  $g(x) = 0$ . This is the complementary slackness, which is part of the KKT condition.

### Dual problem

The dual problem is  $\min_{\lambda \geq 0} \min_x L(x, \lambda)$ .

Let

$$q(\lambda) = \min_x L(x, \lambda).$$

Then the dual problem becomes  $\max_{\lambda \geq 0} q(\lambda)$ . After we find the optimal  $\lambda_*$ , we can then go back to get the optimal  $x_*$  by retrieving the solution to the above min<sub>x</sub> problem.

The dual problem can be simpler than the primal problem if the dimensionality of  $\lambda$  is much smaller than the dimensionality of  $x$ . The Lagrange multiplier  $\lambda$  may also have a meaningful interpretation.

### Dual is concave

$q(\lambda)$  is a concave function. For any fixed  $x$ ,  $L(x, \lambda)$  is a linear function of  $\lambda$  with slope  $g(x)$  and intercept  $f(x)$ . For all  $x$ , we have a set of lines. For each  $\lambda$ , we take the minimum of all the lines. This gives us the lower envelop illustrated in Figure 53. The lower envelop is concave whether  $f(x)$  and  $g(x)$  are convex or not.

Two important dualities in statistics and machine learning are: (1) The duality between maximum entropy and maximum likelihood in exponential family model. (2) The duality between maximum margin and minimal distance in support vector machine (SVM). In addition to the general geometric interpretations provided here, each specific duality has its own special geometric interpretation.



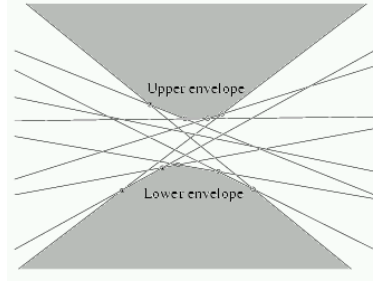


Figure 53: For each  $\lambda$ ,  $q(\lambda)$  is the minimum of all the lines  $\{f(x) + \lambda g(x), \forall x\}$ , thus the function  $q(\lambda)$  lower envelopes these lines, and is a concave function. The dual problem is to maximize  $q(\lambda)$  over  $\lambda \geq 0$ .

### Strong and weak duality

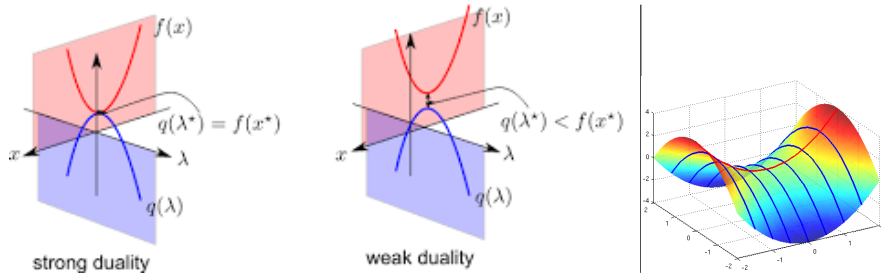


Figure 54:  $q(\lambda)$  lower bounds  $f(x)$ . In the left plot,  $q(\lambda)$  kisses  $f(x)$  at  $(x_*, \lambda_*)$ , and there is strong duality. In the right plot, there is a duality gap between  $q(\lambda)$  and  $f(x)$ , and there is weak duality.

The function

$$q(\lambda) = \min_x L(x, \lambda) \leq \min_{x: g(x) \leq 0} L(x, \lambda) \leq f(x),$$

for  $\lambda \geq 0$ , thus  $q(\lambda)$  lower bounds  $f(x)$ , and  $q(\lambda_*) \leq f(x_*)$ . If  $q(\lambda_*) = f(x_*)$ , i.e.,  $q(\lambda)$  kisses  $f(x)$ , we have strong duality. If  $q(\lambda_*) < f(x_*)$ , there is a duality gap and we have weak duality.

### Contour plot

The situation is illustrated by Figure 55. In the left plot, the minimum of  $f(x)$  is within the constraint  $g(x) < 0$ . Thus  $g(x_*) < 0$  and  $\lambda_* = 0$ . In the right plot, the minimum of  $f(x)$  is outside the constraint  $g(x) \geq 0$ . Thus the constrained minimum is achieved when the contour (level set) of  $f(x)$  touches the contour  $g(x) = 0$  at  $x_*$ , where the two contours are co-tangent, and the gradients of  $f(x)$  and  $g(x)$  at  $x_*$  are of the opposite directions. Thus  $f'(x_*) = -\lambda g'(x_*)$  for some  $\lambda > 0$ , and the derivative of  $f(x) + \lambda g(x)$  is equal to 0 at  $x_*$ , which minimizes  $f(x) + \lambda g(x)$ .

### Range plot

The situation can also be illustrated by Figure 59. Let  $(u = g(x), t = f(x))$  be a point in the space of  $(u, t)$ . The range of  $(u, t)$  is illustrated by the shaded region  $S = \{u = g(x), t = f(x), \forall x\}$ . The minimum of  $t = f(x)$  is taken on the half of the shaded area to the left of the vertical axis, i.e.,  $u = g(x) \leq 0$ . For each  $\lambda$ , the

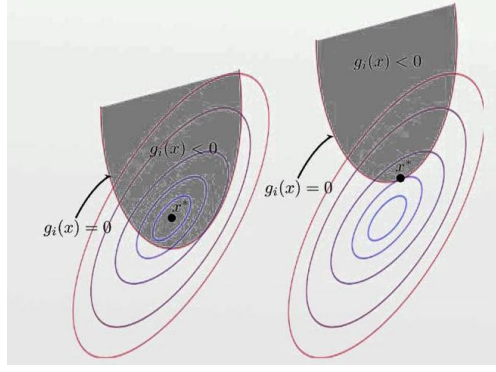


Figure 55: In the left plot, the minimum of  $f(x)$  is within the constrained region  $g(x) \leq 0$ . Thus  $\lambda_* = 0$  and  $g(x_*) < 0$ . In the right plot, the minimum of  $f(x)$  is outside the constrained region, and a contour of  $f(x)$  touches the contour  $g(x) = 0$  at  $x_*$ . The two contours are co-tangent at  $x_*$ , and the gradients of  $f(x)$  and  $g(x)$  at  $x_*$  are of opposite directions, so that  $f'(x_*) = \lambda g'(x_*)$  for a  $\lambda > 0$ .

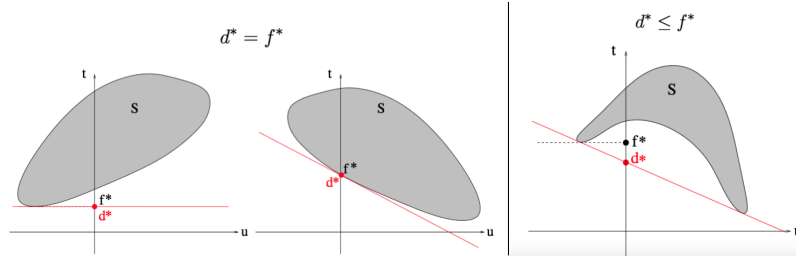


Figure 56: Left: the shaded region is  $S = \{u = g(x), t = f(x)\}$ . The constrained minimum is the lowest point of the left half of the region with  $u \leq 0$ . For each  $\lambda \geq 0$ , the minimum  $q(\lambda)$  of  $f(x) + \lambda g(x) = t + \lambda u$  is the intercept of the supporting line of  $S$  with non-positive slope  $-\lambda$ . The maximum of  $q(\lambda)$  is the highest intercept among all the supporting lines of  $S$ . In the left plot, the minimum is attained for  $u < 0$ , and  $\lambda_* = 0$ . In the right plot, the minimum is attained at  $u = 0$ , and  $\lambda_* < 0$ . Right: The maximum of the dual problem  $d^*$  is less than the minimum of the primal problem  $f^*$ , so we have weak duality.

minimum of  $f(x) + \lambda g(x)$ , which is  $q(\lambda)$ , is the intercept  $c$  of the line  $t = -\lambda u + c$  with a negative slope  $-\lambda$  and the line supports the whole shaded region. We want to find  $\lambda \geq 0$  that achieves the highest intercept, which is  $q(\lambda_*)$ . If the shaded area of  $(t, u)$  is concave, the highest intercept  $q(\lambda_*)$  equals  $f(x_*)$ , and we have the strong duality. If  $u_* = g(x_*) < 0$ , the line is horizontal, and we have  $\lambda_* = 0$ . If  $u_* = g(x_*) = 0$ , we have  $\lambda_* > 0$ , and the line is tangent to the shaded region at  $u = g(x) = 0$ . If the shaded area is not concave, we have weak duality, as illustrated by the right panel of Figure 59.

### Equality constraint

The equality constraint can be similarly treated. Suppose we want to minimize  $f(x)$  subject to  $g(x) = 0$ . The Lagrangian is  $L(x, \lambda) = f(x) + \lambda g(x)$ , where  $\lambda$  can be any real number.

If we want to minimize  $f(x)$  subject to  $g_0(x) = 0$  and  $g_1(x) \leq 0$ . The Lagrangian is  $L(x, \lambda_0, \lambda_1) = f(x) + \lambda_0 g_0(x) + \lambda_1 g_1(x)$ , with  $\lambda_1 \geq 0$ .

### 13.3 Legendre-Fenchel convex conjugate

In constrained optimization, we can rewrite the problem into a min-max problem using the Lagrangian. We can also rewrite an unconstrained minimization problem as a min-max problem, like what we did for the hinge loss.

#### min-max = max-min again

Suppose the primal problem is

$$\min_x [f(x) + \rho(x)],$$

e.g.,  $\rho(x) = |x|^2/2$ , and  $f(x)$  is the loss function such as the hinge loss. We can write

$$f(x) = \max_{\lambda} [\lambda x - f^*(\lambda)].$$

Then the primal problem becomes

$$\min_x \max_{\lambda} [\lambda x - f^*(\lambda) + \rho(x)].$$

Its dual problem is

$$\max_{\lambda} \min_x [\lambda x + \rho(x) - f^*(\lambda)].$$

Let

$$q(\lambda) = \min_x [\lambda x + \rho(x) - f^*(\lambda)],$$

which is particularly simple if  $\rho(x) = |x|^2/2$  and, in such a case, the dual problem is to maximize  $q(\lambda)$ .

#### Supporting lines and upper envelop

A convex function is uniquely defined by its (lower) supporting hyperplanes

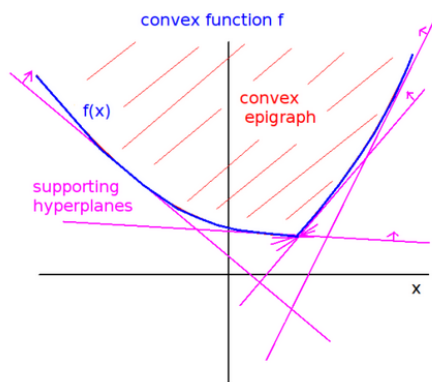


Figure 57: The lower support lines and the upper envelop.

$f^*$  is the Legendre-Fenchel transform or the convex conjugate of  $f$ . For each  $\lambda$ ,  $\lambda x - f^*(\lambda)$  is the lower supporting line of  $f(x)$  with slope  $\lambda$ . As a result,  $f(x)$  is the upper envelop of all the lower supporting lines.

Specifically, for a function  $f(x)$ , let

$$f^*(\lambda) = \max_x [\lambda x - f(x)].$$

Geometrically,  $f^*(\lambda)$  is the maxima gap between  $\lambda x$  and  $f(x)$ , i.e., if we drop the line  $\lambda x$  by  $f^*(\lambda)$ , so that it becomes  $\lambda x - f^*(\lambda)$ , then this line is a supporting line of the function  $f(x)$ . Stated differently, this line is below  $f(x)$ , but touches  $f(x)$  at  $x_*$  that maximizes  $\lambda x - f(x)$ .

We can represent  $f(x)$  as the envelop of its lower supporting lines, i.e.,

$$f(x) = \max_{\lambda} [\lambda x - f^*(\lambda)].$$

For convex  $f$ ,  $f^{**} = f$ . Otherwise,  $f^{**}$  is the lower convex envelop of  $f$ .

### Point-line duality

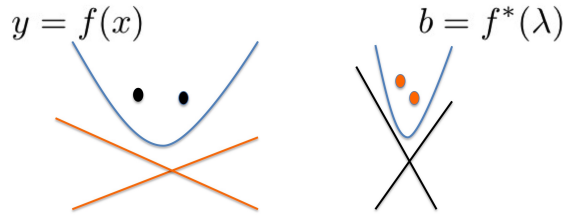


Figure 58: Point line duality. A point in one plane corresponds to a line in another plane, and vice versa.

The same form of calculating lower support line and upper envelop hints at a symmetry or duality. It is the point-line duality. Specifically, for a point  $(\lambda, b)$  in one plane, there is a line  $y = \lambda x - b$  in the plane  $(x, y)$ . Conversely for a point  $(x, y)$ , there is a line  $b = \lambda x - y$  in the plane  $(\lambda, b)$ . Two points forming a line means two lines meet at a point. For a convex function  $y = f(x)$ , any point above  $y = f(x)$  corresponds to a line below  $b = f^*(\lambda)$ . Similarly, for any point  $(\lambda, b)$  above the curve  $b = f^*(\lambda)$ , there is a line below  $y = f(x)$ .

### Complementary regions

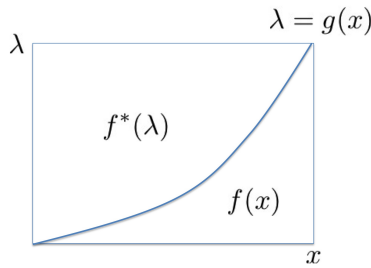


Figure 59: Complementary regions within the rectangle  $[0, x] \times [0, \lambda]$ .  $f(x)$  is the area of the region below  $\lambda = g(x)$ , and  $f^*(\lambda)$  is the area of the region above  $\lambda = g(x)$ .

Another interpretation is based on complementary regions. Consider the curve  $\lambda = g(x)$  within the rectangle  $[0, x] \times [0, \lambda]$ . Let  $f(x) = \int_0^x g(x) dx$  be the area below  $g(x)$  up to  $x$ . Let  $f^*(\lambda) = \int_0^\lambda g^{-1}(\lambda) d\lambda$

be the area above  $g(x)$  up to  $\lambda$ . The total area of the rectangle is  $\lambda x$ . Then  $f(x) = \max_{\lambda} [\lambda x - f^*(\lambda)]$  and  $f^*(\lambda) = \max_x [\lambda x - f(x)]$ .

## References