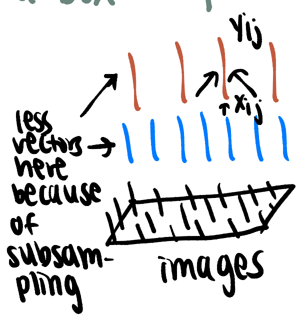
LECTURE 14

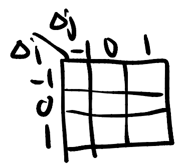
- CNN / conv. Net

each layer corresponds to a box

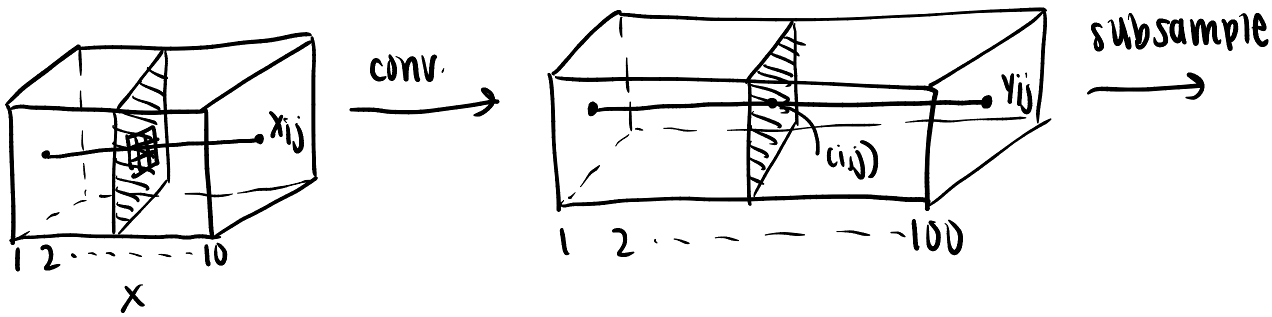
thought vectors compose vectors at lower layer, cover bigger range (more complex)



$$y_{ij} = \sum w_{\Delta i, \Delta j}\, x_{i+\Delta i,\, j+\Delta j} + b + \text{rectify}$$



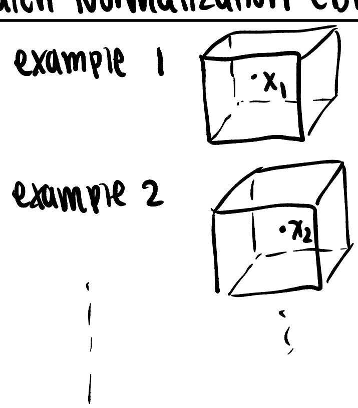
composition



Le Net (80s) MNIST
Dan Net (2010) GPU, bigger model
✓ Alex Net (2012) Image Net (1000 categories, 1 mil. examples)
also a bigger version exists
↳ Amazon Turks

Google Le Net — 1x1, 3x3, 5x5 filters
VGG — 3x3, 16-layer, 19-layer, commonly used as <u>pre-trained</u> features
(use <u>learned weights</u> to get features)
- perceptual
- 64 million parameters

Batch Normalization (BN)



batch $\begin{cases} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_m \end{cases}$ $\Rightarrow$ $\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x_i$ $\Rightarrow$ $\begin{cases} \hat{x}_1 \\ \hat{x}_2 \\ \vdots \\ \hat{x}_i \\ \vdots \\ \hat{x}_m \end{cases}$ $\hat{x}_i = \dfrac{x_i - \bar{x}}{\sigma}$

$$\sigma = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(x_i - \bar{x})^2}$$

insert BN after each weight layer

batch mean = 0
s.d. = 1
avoid covariate shift, stabilize learning

y_1

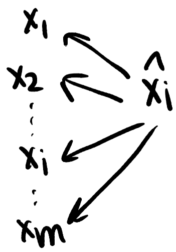
$\vdots$

$\Rightarrow \quad y_i = \beta \hat{x}_i + r$

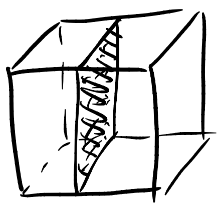
$\vdots$

y_m

back-prop of BN



layer normalization



normalize within the same layer of the <u>same example</u>

Residual Network (Res Net)

conventional CNN:

$$x_{\ell+1} = f_\ell(x_\ell)$$

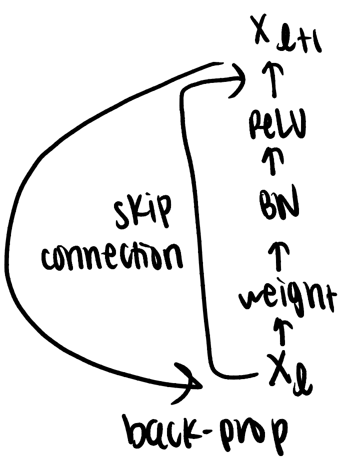
$\downarrow$

weight + BN + ReLU

Res Net:

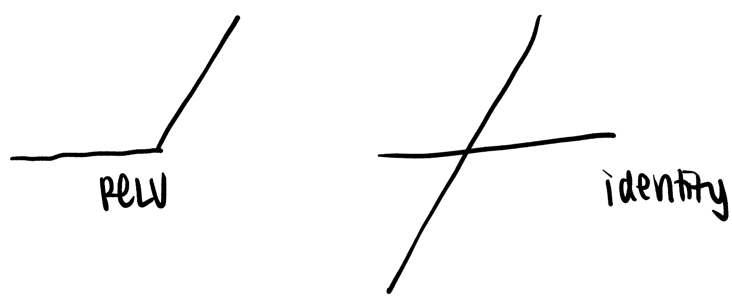
$$x_{\ell+1} = x_\ell + \boxed{f_\ell(x_\ell)}$$

— residual: correction, refinement, update of previous layer

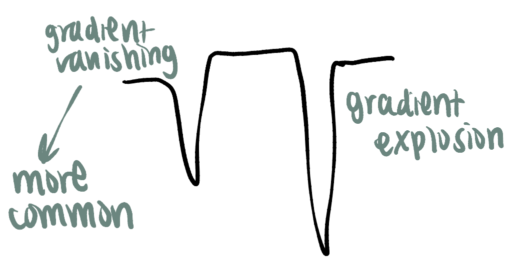


why does this perform better than conventional?

1. ReLU has difficulty to learn identity mapping

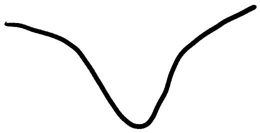


2. Loss function
conventional CNN



- even w/o gradient vanishing, have complex loss function / landscape (sub-optimal local modes)

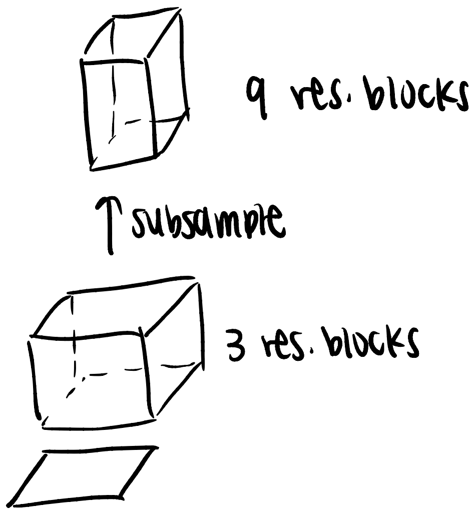
In ResNet



- closer to convex
- loss function better behaved
  ↳ less multi-modal

③ Iterative Algorithm
recurrent network

$x_{t+1} = x_t + f_t(x_t)$
(e.g. gradient descent)



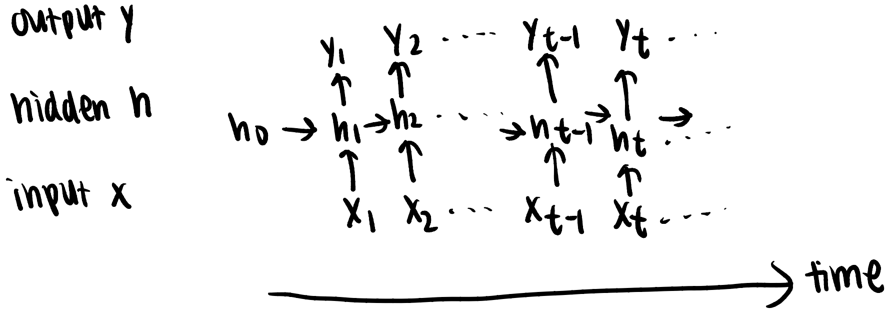
Res 50/200

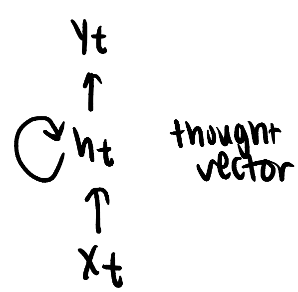
gradient

$$\frac{\partial x_{l+1}}{\partial x_l} = I + \frac{\partial f_l}{\partial x_l}$$

avoid gradient vanishing

Recurrent Neural Network (RNN)





thought process

vanilla RNN

$$h_t = \text{Rectify}\left(W\begin{pmatrix}h_{t-1}\\ x_t\end{pmatrix}\right)$$

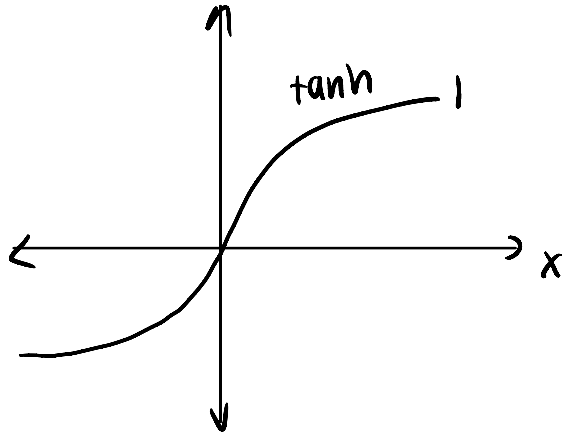
weight concatenate
(generic notation)
differ in each occurrence

$$\begin{bmatrix} W_h & W_x \end{bmatrix} \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}$$

$W_h h_{t-1} + W_x x_t$

Rectify: ReLU sigmoid

$$\tanh = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



$Y_t = \text{Rectify}(W, h_t)$

softmax $\downarrow$ another W

suffer gradient vanishing in back-prop through time
solution: long short term memory (LSTM)

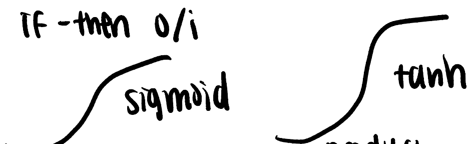
memory cells C_t same dimension as h_t

$$\begin{array}{cccc} f_t & i_t & o_t & \Delta C_t \end{array}$$

$$\begin{bmatrix} f_t & i_t & o_t & \Delta C_t \end{bmatrix} = \text{Rectify}\left(W \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}\right)$$

forget gates, input gates, output gates — IF-then 0/1

update $\rightarrow$ tanh



element wise product

$$C_t = f_t \odot C_{t-1} + i_t \odot \Delta C_t$$

special case $f_t = 1$ $i_t = 1$

$$C_t = C_{t-1} + \Delta C_t \quad \text{(Res Net)}$$

$$h_t = o_t \odot \text{Rectify}(C_t)$$

$$Y_t = \text{Rectify}(W h_t)$$