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

\[ Y_{ij} = \sum w_{oi} x_i + \alpha_j + \beta_j + \text{rectify} \]

LE NET (80s) MNIST
DANNET (2010) GPU, bigger model
✓ ALEX NET (2012) IMAGE NET (1000 categories, 1 mil. examples)
also a bigger version exists
✓ 4 Amazon TURKS

Google LE NET
1x1, 3x3, 5x5 filters
VGG
3x3, 12-layer, 19-layer, commonly used as pre-trained features
(use learned weights to get features)
- perceptual
- 64 million parameters

batch normalization (BN)

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

insert BN after each weight layer

batch mean = 0
s.d. = 1
avoid covariate shift, stabilize learning
\[ y_i = \beta x_i + r \]

Layer normalization

normalize within the same layer of the same example

Residual network (Res Net)

Conventional CNN:
\[ x_{l+1} = f_L(x_L) \]
weight + BN + ReLU

Res Net:
\[ x_{l+1} = x_L + f_L(x_L) \]
residual: correction, refinement, update of previous layer

Skip connection

Back-prop

Why does this perform better than conventional?

1. ReLU has difficulty to learn identity mapping
   - ReLU
   - Identity

2. Loss function
   - Conventional CNN
     - Gradient vanishing
     - Gradient explosion
     - Even w/o gradient vanishing, have complex loss function / landscape (sub-optimal local modes)
In ResNet
- closer to convex
- less function better behaved
- less multi-modal

3. Iterative Algorithm
   recurrent network

   \[ x_{t+1} = x_t + f_t(x_t) \]
   (e.g., gradient descent)

   9 res. blocks

   \[ \uparrow \text{subsample} \]

   3 res. blocks

   Res 50/200

   gradient

   \[ \frac{\partial x_{t+1}}{\partial x_t} = 1 + \frac{\partial f_t}{\partial x_t} \]

   avoid gradient vanishing

Recurrent Neural Network (RNN)

output \( y \)

hidden \( h \)

input \( x \)

\[ y_1, y_2, \ldots, y_{t-1}, y_t, \ldots \]

\[ h_0 \rightarrow h_1 \rightarrow h_2 \rightarrow \cdots \rightarrow h_{t-1} \rightarrow h_t \rightarrow \cdots \]

\[ x_1, x_2, \ldots, x_{t-1}, x_t, \ldots \]

Vanilla RNN

\[ h_t = \text{rectify} \left( W \left( h_{t-1}, x_t \right) \right) \]

weight concatenate

thought vector

\[ y_t \]

thought process

\[ X_t \]
Suffer gradient vanishing in back-prop through time
solution: long short-term memory (LSTM)

\[ Y_t = \text{Rectify}(w_h h_t) \]

\[ \text{Softmax} \]

\[ \text{tanh} \]

\[ \frac{e^x - e^{-x}}{e^x + e^{-x}} \]

\[ w_h h_{t-1} + w_x x_t \]

\[ X_t \]

Memory cells

\[ \text{forget gates} \]
\[ \text{input gates} \]
\[ \text{output gates} \]
\[ \text{update} \]
\[ \text{tanh} \]
\[ \text{sigmoid} \]

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

\[ \text{special case} \quad f_t = 1 \quad i_t = 1 \]

\[ C_t = C_{t-1} + \Delta C_t \quad \text{cres net} \]

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

\[ Y_t = \text{Rectify}(w_h h_t) \]