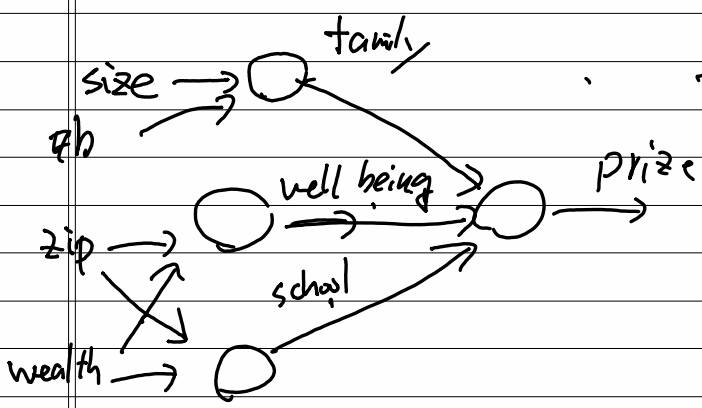
Coursera.

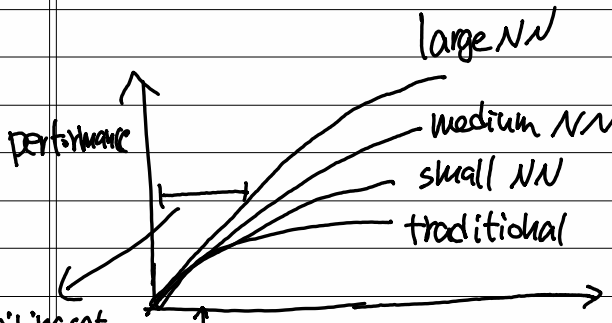


good intuition

but usually full-connected

structured data / unstructured data
table image, video, audio...

DNN outperforms



Data
computation
Algorithm

no clear winner

amount of data (labeled)

m

W2

m training samples $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})\}$

$$X = \begin{bmatrix} | & & | \\ x^{(1)} & \cdots & x^{(m)} \\ | & & | \end{bmatrix} \quad n_x \text{ features}$$

m = # training sample

$X \in \mathbb{R}^{n_x \times m}$

$X.shape = (n_x, m)$

$$Y = \begin{bmatrix} y^{(1)} & \cdots & y^{(m)} \end{bmatrix}$$

$1 \times m$

$Y \in \mathbb{R}$

★ stack in column for easier implementation

Logistic regression

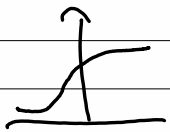
Given X, want $\hat{y} = P(y=1|x)$

$X \in \mathbb{R}^{n_x \times 1}$

parameter: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

sigmoid function σ

$\hat{y} = w^T x + b \Rightarrow \hat{y} = \sigma(w^T x + b)$



$$\sigma(z) = \frac{1}{1+e^{z}}$$

$z \nearrow \infty \quad \sigma(z) = \frac{1}{1+0} = 1$

$z \searrow$ negative $\quad \sigma(z) = \frac{1}{1+\text{big}} = 0$

COST FUNC

cost func

$\hat{y} = \sigma(w^T x + b)$ $\quad \sigma(z) = \frac{1}{1+e^{-z}}$

want $\hat{y}^{(i)} \simeq y^{(i)}$

loss:

$$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

loss func

$$L(\hat{y}, y) = -(y \log(\hat{y}) + (1-y)\log(1-\hat{y}))$$

$y=1 \rightarrow L(\hat{y}, y) = -\log(\hat{y})$ want $-\log(\hat{y})$ ↓

$\log(\hat{y})$ ↑

$\hat{y}$ large

$y=0 \rightarrow L(\hat{y}, y) = -\log(1-\hat{y})$ want $\log(1-\hat{y})$ ↑

$\hat{y}$ small

$\hat{y} = (0, 1)$

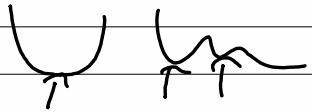
cost func $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(\hat{y}^{(i)}) - (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$$

GRADIENT DESCENT

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y)$$

convex non-convex



$J(w)$

$$w := w - \underbrace{\alpha}_{\text{learning rate}} \frac{dJ(w)}{dw}$$

$J(w,b)$

$$w := w - \alpha \frac{d\,J(w,b)}{dw} \qquad \frac{\partial J(w,b)}{\partial w}$$

$$b := b - \alpha \frac{d\,J(w,b)}{db}$$

(partial derivative symbol

∂ : partial (for multiple var)

d : derivative

DERIVATIVE

$$\text{slope} = \frac{h}{w}$$

$f(a) = 3a$

$$\frac{df(a)}{da} = 3$$

$f(a) = a^2$

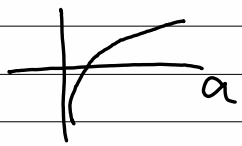
$$\frac{df(a)}{da} = 2a$$

increase a by 0.1

$y \uparrow 2 \times 0.1 = 0.2$

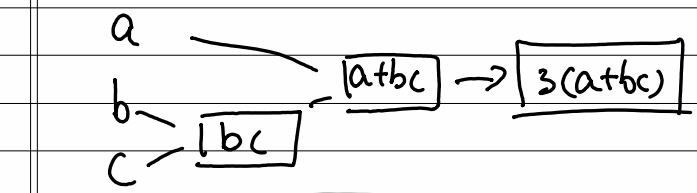
nudge $a \rightarrow$, y increase $2a$

$f(a) = \log(a)$ $\quad \frac{d}{da} f(a) = \frac{1}{a}$

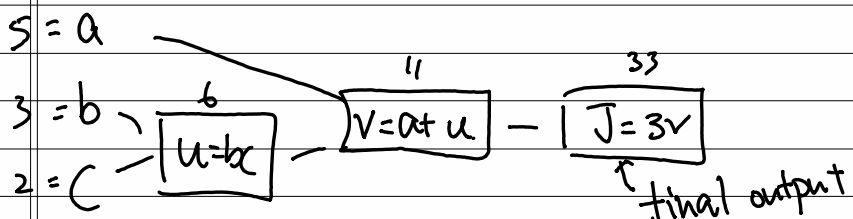


Computation Graph

$J(a,b,c) = 3(a+bc)$



Derivatives with computation graph



$\frac{dJ}{dv} = 3$

$V = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

diff: $0.001 \times 3 = 0.003$

$\frac{dJ}{da} = ?$

$a = 5 \rightarrow 5.001$

$V = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

$\rightarrow \frac{dv}{da} = 1$

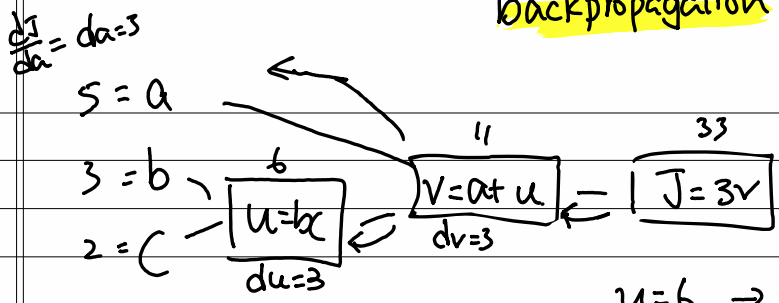
$$\frac{dJ}{da} = \frac{dJ}{dv}\frac{dv}{da}$$ chain rule

Coding convention

$\frac{d\,\text{finaloutputvar}}{d\,\text{var}} \rightarrow$ dvar

- derivative of final output var.

backpropagation



$$\frac{dJ}{du} = \frac{dJ}{dv} \cdot \frac{dv}{du} = 3 \cdot 1 = 3$$

$$\frac{dJ}{db} = \frac{dJ}{dv} \cdot \frac{dv}{du} \cdot \frac{du}{db} = 3 \cdot 1 \cdot 2 = 6$$

$$\frac{dJ}{dc} = \frac{dJ}{du} \cdot \frac{du}{dc} = 3 \cdot 3 = 9$$

$u = 6 \rightarrow 6.001$
$v = 11 \rightarrow 11.001$
$J = 33 \rightarrow 33.003$

$b = 3 \rightarrow 3.001$
$u = 6 \rightarrow 6.002$
$v = 11 \rightarrow 11.002$
$J = 33 \rightarrow 33.006$

Logistic regression gradient descent

$z = w^T x + b$

$\hat{y} = a = \sigma(z)$

single sample — $L(a, y) = -(y \log(a) + (1-y)\log(1-a))$

$z = w_1 x_1 + w_2 x_2 + b \rightarrow a = \sigma(z) \rightarrow L(a, y)$

$dz = \frac{dL}{dz} = \frac{dL(a,y)}{dz} = \frac{dL}{da} \cdot \frac{da}{dz}$

$da = \frac{dL(a,y)}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$

$= a - y$ ← $a(1-a)$ sigmoid derivative

$\frac{dL}{dw_1} = \frac{dL}{dz} \cdot \frac{dz}{dw_1} = x_1 \cdot \frac{dL}{dz}$

m traing example

$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} L(a^{(i)}, y)$

$a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$

to calculate → $dw_1^{(i)}, dw_2^{(i)}, db^{(i)}$

$\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial w_1} L(a^{(i)}, y^{(i)})$

$dw_1^{(i)} = (x^{(i)}, y^{(i)})$

$\frac{\partial}{\partial w_2} J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial w_2} (a^{(i)}, y^{(i)})$

- vectorization

CODE

$$\frac{dL(a,y)}{dz} = a - y$$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(a^{(i)}, y^{(i)}) \qquad \frac{dJ}{dw_1} \quad \frac{dJ}{dw_2} \quad \frac{dJ}{db}$$

for each sample

$dz = a^{(i)} - y^{(i)}$

$dw_1 += x_1 \cdot (a - y)$

$dw_2 += x_2 \cdot (a - y)$

for $i = 1$ to m

$z^{(i)} = w^T x^{(i)} + b$

$a^{(i)} = \sigma(z)$

$J += -[y^{(i)} \log(a^{(i)}) + (1-y^{(i)}) \log(1-a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw_1 += x_1 dz^{(i)}$

$dw_2 += x_2 dz^{(i)}$

$db += dz^{(i)}$

$J /= m$

$dw1 /= m$

$dw2 /= m$

$db /= m$

vectorization / vectorized logistic regression

$z^{(1)} = w^T x^{(1)} + b \quad z^{(2)} = w^T x^{(2)} + b$

$a^{(1)} = \sigma(z^{(1)}) \quad a^{(2)} = \sigma(z^{(2)}) \quad \cdots$

$X \qquad X \in \mathbb{R}^{n \times m}$

$[z^{(1)} \dots z^{(m)}] = w^T X + [b \dots b]$

$1 \times m \quad 1 \times n \quad n \times m \quad 1 \times m$

$= [w^T x^{(1)} + b \quad w^T x^{(2)} + b \quad w^T x^{(3)} + b \dots]$

$z = np.dot(w^T, X) + b$

z scalar

$dz^{(i)} = a^{(i)} - y^{(i)}$

CODE

one iter of gradient descent

$z = w^T x + b$

$= np.dot(w.T, x) + b$

$a = \sigma(z)$

$dz = A - Y$

$dw = \frac{1}{m} X dz^T$

$db = \frac{1}{m} np.sum(dz)$

$w := w - \alpha dw$

$b := b - \alpha db$

Broadcasting

$\begin{bmatrix}1\\2\\3\\4\end{bmatrix} + 100 \rightarrow \begin{bmatrix}1\\2\\3\\4\end{bmatrix} + \begin{bmatrix}100\\100\\100\\100\end{bmatrix} = \ldots$

$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + [100.\ 200\ 300] \rightarrow \begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100. & 200 & 300\\100 & 200 & 300\end{bmatrix} = \ldots$

(m,n) (1,n) ~~~ (m,n) (m,n)

$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100\\200\end{bmatrix} \rightarrow +$

(m,n) (m,2) (m,n) - m,n = ...

(m,n) $\begin{matrix}+\\*\\\div\end{matrix}$ (1,n) ~ (m,n) broadcast

(m,1) ~ (m,n) broadcasting

DEBUG

avoid "rank-1" array.

randn(5) - DON'T USE

randn(5,1) - column vector

randn(1,5) row vector

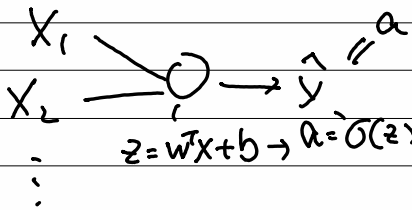
☆ → assert(a.shape = (5,1))

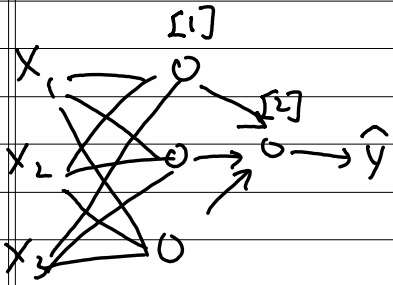
△ (5,).reshape((5,1))

NN

WEEK 3

Logistic regression





$z^{[1]} = w^{[1]} x + b^{[1]} \rightarrow a^{[1]} = \sigma(z^{[1]}) \rightarrow z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$

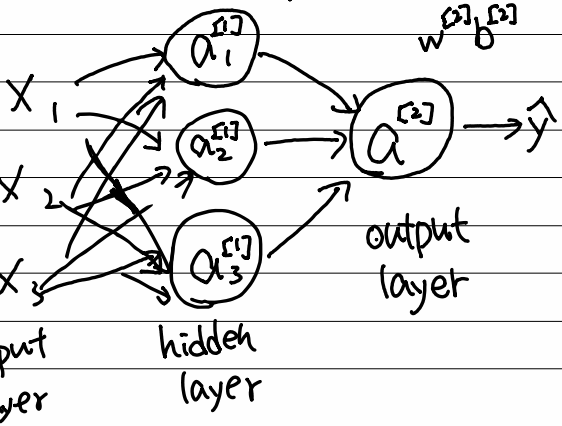
$\rightarrow a^{[2]} = \sigma(z^{[2]})$

$\rightarrow L(a^{[2]}, y)$

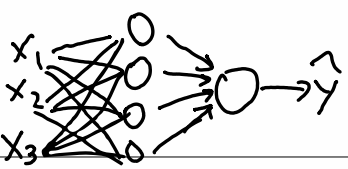
2-layer NN

layer

$a^{[0]} = x$ $\quad a^{[1]}$ $\quad w^{[1]}, b^{[1]}$ $\quad w^{[2]}, b^{[2]}$



$a_i^{[l]}$ — l: layer, i: node in layer



Computing NN output

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$

neuron

vectorize

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

$$\begin{bmatrix} — w_1^{[1]} — \\ — w_2^{[1]} — \\ \vdots \\ — w_4^{[1]} — \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} \cdot x + b_1^{[1]} \\ \vdots \\ w_4^{[1]T} \cdot x + b_4^{[1]} \end{bmatrix}$$

4×3 $\quad 3 \times 1$

$W^{[1]}$ $\qquad b^{[1]}$

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]})$$

CODE

$$z^{[1]} = W^{[1]T} x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]T} a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$

m samples

$$x \longrightarrow a^{[2]} = \hat{y}$$
$$x^{(1)} \longrightarrow a^{[2](1)} = \hat{y}^{(1)}$$
$$\vdots$$
$$x^{(m)} \longrightarrow a^{[2](m)} = \hat{y}^{(m)}$$

for i = 1 to m

$$z^{[1](i)} = W^{[1](i)} x^{(i)} + b^{[1](i)}$$

$\vdots$

vectorize
m training

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$n \times m$.

$$Z^{[1]} = W^{[1]} X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

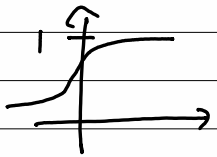
$$A^{[1]} = \begin{bmatrix} | & & | \\ a^{[1](1)} & \cdots & a^{[1](m)} \\ | & & | \end{bmatrix}$$

* different node in layers ↑
→ different training sample

$$z^{[1](1)} = w^{[1]} x^{(1)}$$

ACTIVATION

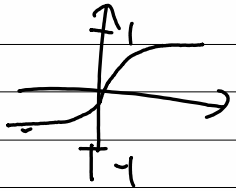
$\sigma = \frac{1}{1+e^{-z}}$



prob only for output [0,1]

$g = \tanh(z)$

$= \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$



almost always better for hidden layers

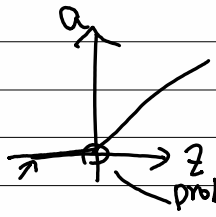
con:

- large z → small gradient → slow

DEFAULT: **ReLU**

Rectified Linear Unit

$a = \max(0, z)$

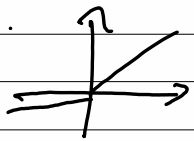


problem in theory only

pro: learn faster!

- bc not affected by small slope
- usually work w/ deep nn w/ $z > 0$

Leaky ReLU



$a = \max(0.01z, z)$

Derivative of Activation Function

$$g(z) = \frac{1}{1+e^{-z}}$$

prime $g'(z)$

$$\frac{dg(z)}{dz} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$

$$= g(z) \cdot (1 - g(z))$$

g prime of z

* Power rule applies when numerator is the power of derivcator

$$\frac{d}{dx} x^n = n \cdot x^{n-1}$$

$$\frac{d}{dx} f(x)^n = \frac{d}{df(x)} f(x)^n \cdot \frac{df(x)}{dx}$$

$$= n \cdot f(x)^{n-1} \cdot f'(x)$$

$$\frac{d}{dx} f(x)^{-1} = -f(x)^{-2} \cdot f'(x)$$

$$= \frac{-f'(x)}{f(x)^2}$$

$$\frac{d}{dz} e^{-z} = -z \cdot e^{-z}$$

$$= -\frac{d}{dz}(1+e^{-z}) \times (1+e^{-z})^{-2}$$

$$= -e^{-z} \cdot (1+e^{-z})^{-2}$$

$$= \frac{-e^{-z}}{(1+e^{-z})^2}$$

$$\frac{d}{dx} \frac{g(x)}{f(x)}$$

$$= \frac{d}{dx} g(x) \cdot f(x)^{-1}$$

---

tanh

$$g(z) = \tanh(z)$$

$$\frac{d}{dz} \tanh(z) = 1 - (\tanh(z))^2$$

ReLU

$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

implement NN

Forward Prop

$$z^{[1]} = W^{[1]} X + b^{[1]}$$
$$A^{[1]} = g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[1]} A^{[1]} + b^{[2]}$$
$$A^{[2]} = g^{[2]}(z^{[2]}) = \sigma(z^{[2]})$$

Back Prop

$$dz^{[2]} = A^{[2]} - Y$$
$$dw^{[2]} = \frac{1}{m} dz^{[2]} A^{[1]T}$$
$$db^{[2]} = \frac{1}{m} np.sum(dz^{[2]}, axis=1, keepdim)$$
$$dz^{[1]} = W^{[1]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$
$$dw^{[1]} = \frac{1}{m} dz^{[1]} X^T$$
$$db^{[1]} = \frac{1}{m} np.sum(dz^{[1]}, axis=1, keepdim=true)$$

Initialization

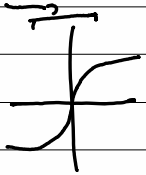
× all $w = 0$
- all hidden unit will be the same

$W^{[1]} = np.random.randn((2,2)) * 0.01$ → in this range

$b^{[1]} = np.zero((2,1))$

⋮

$b^{[2]} = 0$



W3 assignment

planar

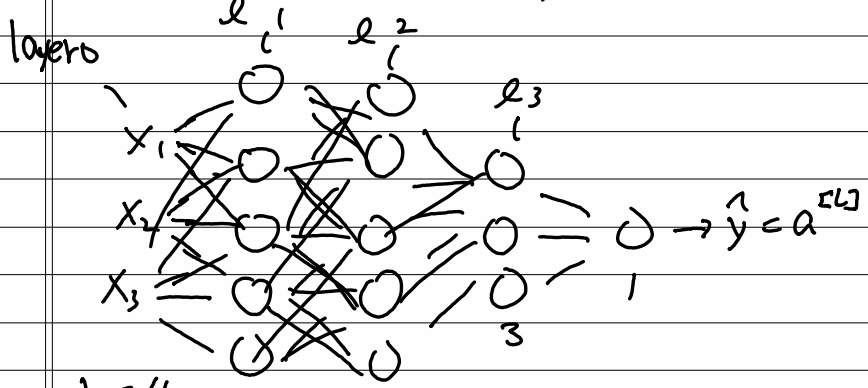
X [x_1 x_2 ...] 2xm

Y [ ] 1xm label

W4

DNN - Deep NN

shallow vs deep



$n^{[l]}$ = # unit in layer l

$n^{[1]}=5$, $n^{[2]}=5$, $n^{[3]}=3$, $n^{[4]}=1$

$n^{[0]} = n_x = 3$

$a^{[l]}$ = activation in layer l

$a^{[l]} = g^{[l]}(z^{[l]})$ $w^{[l]}$ $b^{[l]}$ = weights for $z^{[l]}$

Forward Prop

$$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$$

$$a^{[l]} = g^{[l]}(z^{[l]})$$

$$\Downarrow$$

vectorize

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

$$A^{[l]} = g^{[l]}(Z^{[l]})$$

$$\downarrow$$

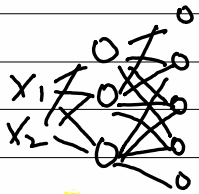
$$\hat{Y} = g(Z^{[4]}) = A^{[4]}$$

$z^{[l]}$

$\{ z^{[l](1)} \cdots z^{[l](m)} \}$

$Z^{[l]}$

DEBUG DNN



$$Z^{[1]} = W^{[1]} \cdot X + b$$

$(3,1)$ $(2,1)$

so, W should be $(3,2)$

$$\Rightarrow (n^{[1]}, 1) = (n^{[1]}, n^{[0]})(n^{[0]}, 1)$$

$$Z^{[2]} = W^{[2]} \cdot a^{[1]} + b^{[2]}$$

$(5,1)$ $(3,1)$

$\Downarrow$

so. $(5,3)$

$$\vdots$$

$$w^{[l]} : (n^{[l]}, n^{[l-1]})$$

$$b^{[l]} : (n^{[l]}, 1)$$

$$dw^{[l]} : (n^{[l]}, n^{[l-1]})$$

$$db^{[l]} : (n^{[l]}, 1)$$

VECTORIZED

$$z^{[1]} = W^{[1]} \cdot x + b^{[1]}$$

$(n^{[1]}, 1)\ (n^{[1]}, n^{[0]})\ (n^{[0]}, 1)\ \ (n^{[1]}, 1)$

$$Z^{[1]} = W^{[1]} \cdot X + b^{[1]}$$

$(n^{[1]}, m)\ (n^{[1]}, n^{[0]})\ (n^{[0]}, m)\ \ (n^{[1]}, 1)$

Python broadcasting

↓

$(n^{[1]}, m)$

$Z^{[l]}, A^{[l]} : (n^{[l]}, m)$

$l = 0,\ A^{[0]} = X : (n^{[0]}, m)$

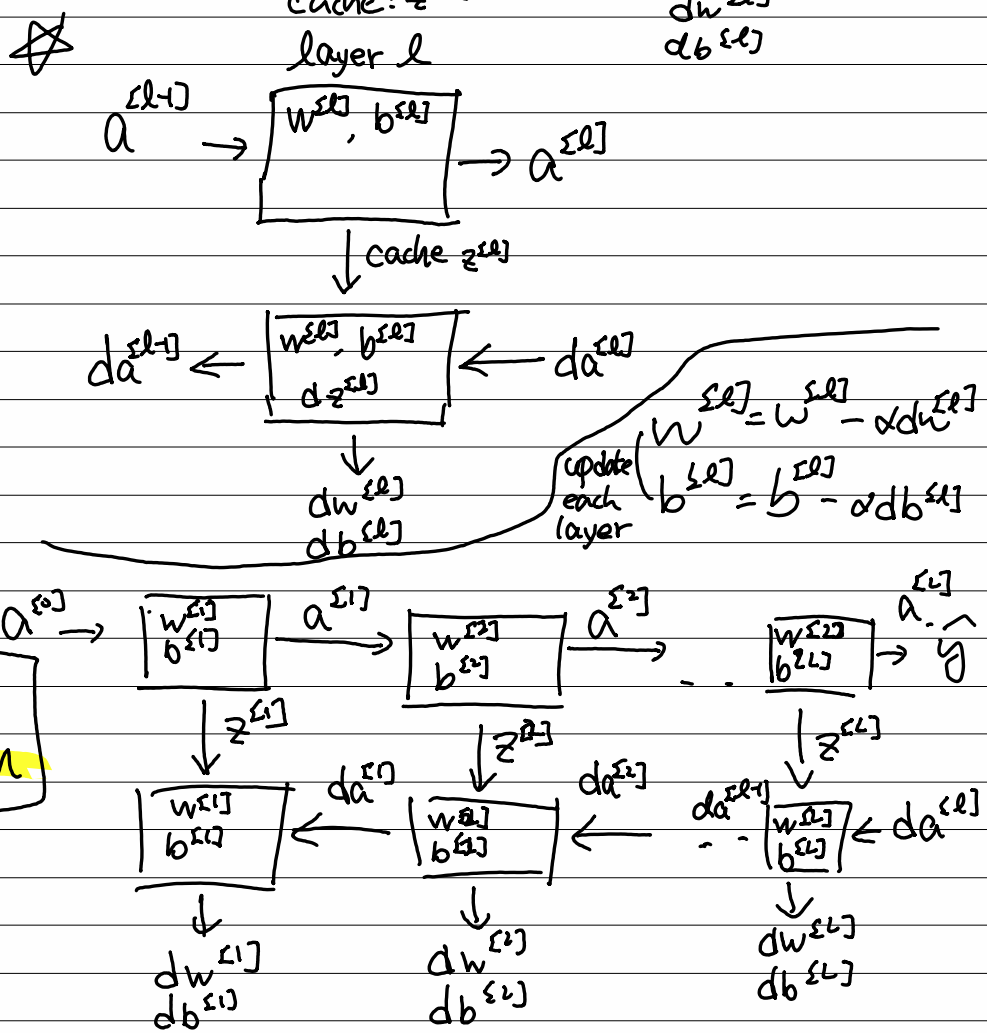
$dZ^{[l]}, dA^{[l]} : (n^{[l]}, m)$

Forward: input $a^{[l-1]}$, output $a^{[l]}$

$z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]}$, cache $z^{[l]}$

$A^{[l]} = g^{[l]}(z^{[l]})$

Backward: input $da^{[l]}$, output $da^{[l-1]}$



IMPLEMENTATION

BACKPROP

- input: $da^{[l]}$

output: $da^{[l-1]}, dw^{[l]}, db^{[l]}$

$$dz^{[l]} = da^{[l]} * g^{[l]'}(z^{[l]})$$

$$dw^{[l]} = dz^{[l]} \cdot a^{[l-1]T}$$

$$db = dz^{[l]}$$

$$da^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$$

$$dz^{[l]} = w^{[l+1]T} dz^{[l+1]} * g^{[l]'}(z^{[l]})$$

$$\Delta da^{[L]} = -\frac{y}{a} + \frac{(1-y)}{(1-a)}$$

$$dA^{[L]} = \left[-\frac{y^{(1)}}{a^{(1)}} + \frac{(1-y^{(1)})}{1-a^{(1)}}, \dots, -\frac{y^{(m)}}{a^{(m)}} + \frac{(1-y^{(m)})}{1-a^{(m)}} \right]$$

HYPERPARAMETER

- learning rate
- iteration
- hidden layer L
- hidden unit $n^{[1]}, n^{[2]}$
- activation func

→ parameters

$W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]} \dots$

look at cost function graph to decide hyperparameter

trial-and-error.

Course 2

Hyperparameter Tuning

COURSE 2

- iterative process
  for hyperparameter tuning

idea → code → exp → idea

DATA

| training | dev | test |

dev: cross-validate
test: for unbias testing

previous

70 / 30 or 60/20/20.

100 ~ 10000 samples

big data

1,000,000 data, 10,000 test might be enough

10,000

→ 98% / 1% / 1%

Mismatch train/test distribution

training: e.g cat on web

dev/test: cat on app

- try making dev/test with same distribut

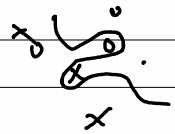
not having test set might be ok

BIAS & VARIANCE

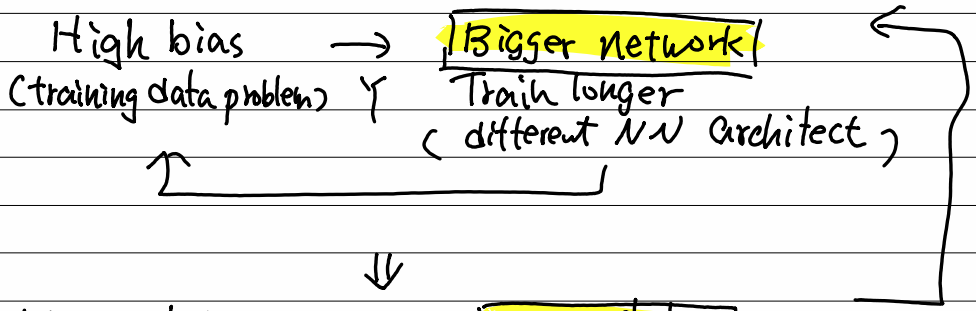
- under fitting — high bias (not fitting well)
- just right
- over fitting — high variance (too flexible)

---

training set error:	1 %	15 %	15 %
dev set error :	11 %	16 %	30 %
	→ over fitting high variance	- under fit high bias	high bias var
			0.5 %
← high bias + high variance			1 %
			low bias var



RECIPE for ML



* traditional method is difficult to improve bias AND variance!

high bias → more training data is not useful

high variance → regularization
more training data.

Regularization

Logistic Regre.. $w \in R^{n_x}$ $b \in R$

$\min_{w,b} J(w,b)$ reλ: regularization para

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_2^2 + \underbrace{\frac{\lambda}{2m} b}_{\text{omit}}$$

L_2 regu $\|w\|_2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$

L_1 regularization $\|w\|_1$ $\frac{\lambda}{2m}\sum_{j=1}^{n_x}|w_j| = \frac{\lambda}{2m}\|w\|_1$

Neural Network

$$J(w^{[1]}, b^{[1]}, \ldots, w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}\|w^{[l]}\|^2$$

square norm

$$\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l-1]}}\sum_{j=1}^{n^{[l]}} (w_{ij}^{[l]})^2 \qquad w: (n^{[l]}, n^{[l-1]})$$

Frobenius form Sum of square norm of matrix

$$dw^{[l]} = (\text{from back prop}) + \frac{\lambda}{m} w^{[l]}$$

$$w^{[l]} = w^{[l]} - \alpha\, dw^{[l]}$$

$$= w^{[l]} - \frac{\alpha\lambda}{m} w^{[l]} - \alpha(\text{from backprop})$$

weight decay

$$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w^{[l]}\|^2$$

$\lambda \uparrow \Rightarrow w \downarrow$ (bcz we penalize high cost)

$\Rightarrow$ simpler NN

Dropout regularization

- dropout random nodes in NN (hidden unit)

INVERTED DROPOUT

eg at layer 3,

keep-drop = 0.8

d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep-prop

a3 = np.multiply(a3, d3)

* ⇒ a3 /= keep-prop

e.g. total 50 units. shut off 10 units (20%)

$z^{[3]} = w^{[3]} \cdot a^{[3]} + b^{[3]}$

↑ bump up by 20%

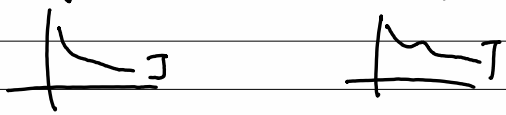
* no drop-out at test time

Why drop-out works?

- intuition → can't rely on one feature, so we spread out weights.
- keep-drop can vary in layers → shrink weights similar to L_2

- Computer Vision uses drop out bc high dimension

- Downside. J might looks weird in the plot

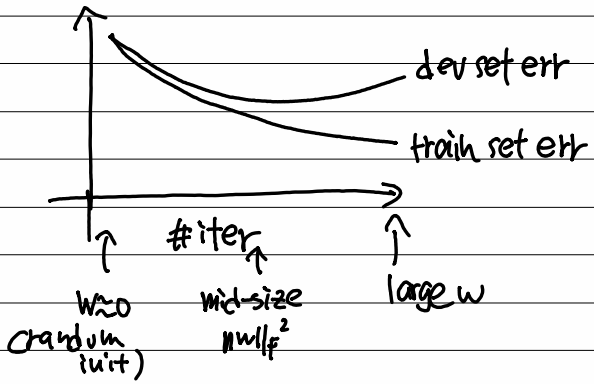


Regularization methods

Data augmentation

- flipping, random crop images. (flipping cats)
- adding distortion to characters etc

Early stopping



- breaks orthogonalization

Normalizing Input:

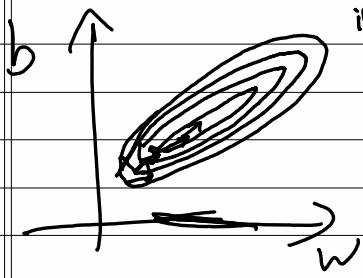
subtract mean
normalize variance

$x = x - \mu$

$x /= \sigma^2$

* use same μ & σ^2 for testing / eval / train sets

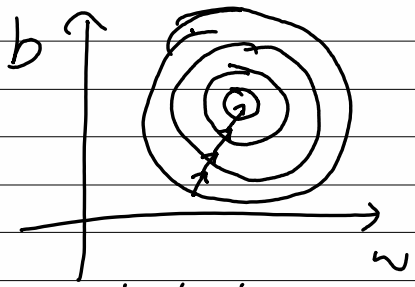
unnormalized



w might be in very diff range

require small learning rate for gradient descent

normalized



better!
safer to take larger step!

vanishing / exploding gradient

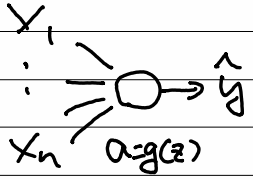
(assuming linear activation) back-prop

$\hat{y} = w^{[L]} \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L-1} X$ - exploding

or

$\hat{y} = w^{[L]} \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}^{L-1} X$ - vanishing

solution (initialization)



$z = w_1 x_1 + \dots + w_n x_n$ ~~$+b$~~

large n → small w_i

$Var(w_i) = \frac{1}{n}$ if ReLU, $Var(w_i) = \frac{2}{n}$

$w^{[l]} = np.random.rand(slope) \times np.sqrt(\frac{2}{n^{[l-1]}})$

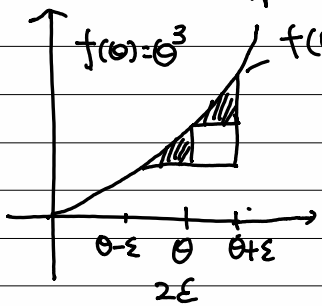
ReLU

tanh : $\sqrt{\frac{1}{n^{[l-1]}}}$

or

Xavier initialization $\sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}}$

Numeric Approx of Gradients



$$\Delta \quad \frac{f(\theta+\epsilon)-f(\theta-\epsilon)}{2\epsilon} \approx g(\theta)$$

$$\frac{(1.01)^3-(0.99)^3}{2\cdot 0.01}=3.0001\approx 3$$

app error $= 0.0001$ ✓

$$f'(\theta)=\lim_{\epsilon\to 0}\frac{f(\theta+\epsilon)-f(\theta-\epsilon)}{2\epsilon} \quad O(\epsilon^2)$$

$$f'(\theta)\lim_{\epsilon\to 0}\frac{f(\theta+\epsilon)-f(\theta)}{\epsilon} \quad \text{err: } O(\epsilon)$$

△ we use **two-sided ϵ** for gradient checking (more accurate) instead of one-sided checking

DEBUG TOOL

Gradient Checking (Grad Check)

$$J(W^{[1]}, b^{[1]} \dots W^{[n]} b^{[n]}) = J(\theta)$$

$$J(dW^{[1]}, db^{[1]} \dots dW^{[n]}, db^{[n]}) = J(d\theta)$$

for each i

$$d\theta_{approx}^{[i]} = \frac{J(\theta_1 \dots \theta_i+\epsilon, \dots) - J(\theta_1 \dots \theta_i-\epsilon \dots)}{2\epsilon}$$

$$\approx d\theta^{[i]} = \frac{\partial J}{\partial \theta_i} \Rightarrow \text{check } d\theta_{app} \overset{?}{\approx} d\theta \; ?$$

$$\frac{\|d\theta_{approx} - d\theta\|_2}{\|d\theta_{approx}\|_2 + \|d\theta\|_2} \approx$$

10^{-7} - great

10^{-5}

10^{-3} - worry! bug!

Grad Check Implementation

- Don't use in training
- look at component to try to identify the bug
- remember regularization
- doesn't work with dropout
  - keep-prop = 1 for debug
- run at random initialization

mini-batch gradient descent

split training set into batches

$X^{\{t\}}, Y^{\{t\}}$

batch v.s. mini-batch
(all) (small batches)

△ notation

$X^{(i)}$

$X^{[l]}$

$X^{\{t\}}$ — curly bracket. training set batch

EXAMPLE

5,000,000 data → 5000 mini-batch, each has 1000 data.

for t = 1 to 5000

forward prop on $X^{\{t\}}$

cost $J^{\{t\}} = \frac{1}{1000} \sum \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \|w^{[l]}\|_F^2$

backprop using $J^{\{t\}}$

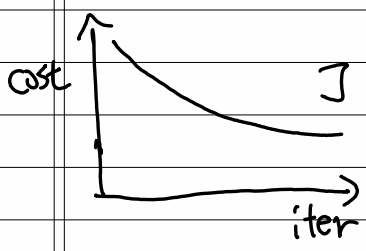
$w^{[l]} = w^{[l]} - \alpha dw^{[l]}$ $b^{[l]} = b^{[l]} - \alpha db^{[l]}$

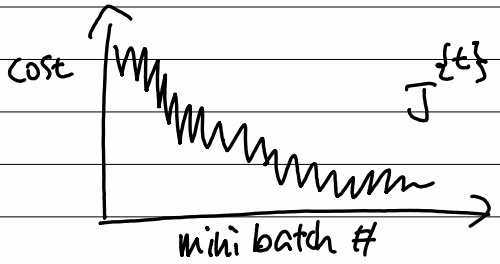
1 epoch — a single pass of 'ALL' data

BATCH — 1 step descent

MINIBATCH — 5000 step descent.

WHY MINI-BATCH WORK?





noisy, but trend down
bc? batch might be difficult

mini-batch size

size = m → batch gradient descent
size = 1 → stochastic gradient descent.

BATCH - too long per iteration
STOCHASTIC - lose the benefit of vectorization

GUIDE:

small traing set:
(~2000) use batch gradient

typical size
64 ~ 512
make sure mini-batch fits CPU/GPU memory.

Exponentially Weighted Average

moving

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

V_t is approx average over $\frac{1}{1-\beta}$ samples

$\beta = 0.9$

e.g. $\beta = 0.9 \rightarrow$ 10 samples

$\beta = 0.98 \rightarrow$ 50 samples

$V_{100} = 0.1\theta_{100} + 0.9V_{99}$

$V_{99} = 0.1\theta_{99} + 0.9V_{98}$

$V_{98} = 0.1\theta_{98} + 0.9V_{97}$

$\vdots$

$V_{100} = 0.1\theta_{100} + 0.9V_{99}$

$= 0.1\theta_{100} + 0.9(0.1\theta_{99} + 0.9V_{98})$

$= 0.1\theta_{100} + 0.1\cdot 0.9\,\theta_{99} + 0.9^2(0.1\theta_{98} + 0.9V_{97})$

$= 0.1\theta_{100} + 0.1\cdot 0.9\,\theta_{99} + 0.1(0.9)^2\theta_{98} + (0.9)^3 V_{97} + \dots$

$(0.9)^{10} \approx 0.35 \approx \frac{1}{e}$ after $\frac{1}{e}$ it drops faster.

$(0.98)^{50} \approx \frac{1}{e}$

Bias Correction in EWA

bias because $V_0 = 0$ $\quad v_t = \beta v_{t-1} + (1-\beta)\theta_t$

incorrect v in the beginning of series

$V_0 = 0$

$v_1 = 0.9v_0 + 0.1\theta_1 = 0.1\theta_1$

$v_2 = 0.9v_1 + 0.1\theta_2$

$= 0.09\theta_1 + 0.1\theta_2$

$\vdots$

Correction

$\frac{v_t}{1-\beta^t}$, correct in the beginning

$t \uparrow$, $\beta^t \sim 0$, correction does not affect v_t.

Gradient Descent with Momentum

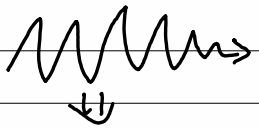
Compute EWA of gradient, then update

on iter t

compute dw db on current mini-batch

$V_{dw} = \beta V_{dw} + (1-\beta) dw$

$V_{db} = \beta V_{db} + (1-\beta) db$

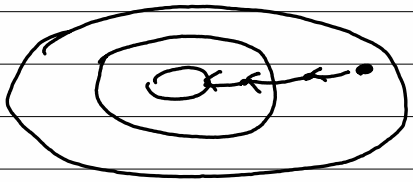


$W = W - \alpha V_{dw}$

$B = B - \alpha V_{db}$.

dampen oscillation

overshoot.



ball rolling.

$V_{dw} = \beta V_{dw} + (1-\beta) dw$

$V_{db} = \beta V_{db} + (1-\beta) db$

acceleration

velocity

friction

hyper parameters

α - learning rate

β - 0.9 usually works

RMSprop (root mean square prop)

On iter t.

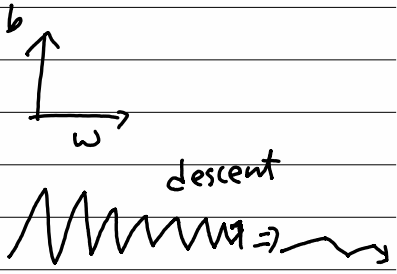
compute dw, db on mini-batch

$S_{dw} = \beta S_{dw} + (1-\beta) dw^2$ - element-wise square

$S_{db} = \beta S_{db} + (1-\beta) db^2$

$w = w - \alpha \frac{dw}{\sqrt{S_{dw}} + \varepsilon}$ (ε: avoid divided by 0)

$b = b - \alpha \frac{db}{\sqrt{S_{db}} + \varepsilon}$



S_{dw} ↓ w ↑

S_{db} ↑ b ↓.

Adam Optimization Algorithm

adaptive moment estimation

momentum + RMSdrop

$V_{dw}=0 \quad S_{dw}=0 \quad V_{db}=0, \; S_{db}=0$

On iter t.

compute dw, db of mini-batch

momentum (

$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dw \quad V_{db} = \beta_1 V_{db} + (1-\beta_1) db$

$S_{dw} = \beta_2 S_{dw} + \beta_2 dw^2 \quad S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$)

RMSprop

bias correction

$$V_{dw}^{corrected} = \frac{V_{dw}}{(1-\beta_1^t)} \qquad V_{db}^{corr} = \frac{V_{db}}{(1-\beta_1^t)}$$

$$S_{dw}^{corrected} = \frac{S_{dw}}{(1-\beta_2^t)} \qquad S_{db}^{corr} = \frac{S_{db}}{(1-\beta_2^t)}$$

RMSprop

$$w = w - \alpha \frac{V_{dw}^{corr}}{\sqrt{S_{dw}^{corr}} + \varepsilon} \qquad b = b - \alpha \frac{V_{db}^{corr}}{\sqrt{S_{db}^{corr}} + \varepsilon}$$

hyperparameter α - tune

β_1 : 0.9 → dw - first momentum

β_2 : 0.999 → dw^2 - second momentum

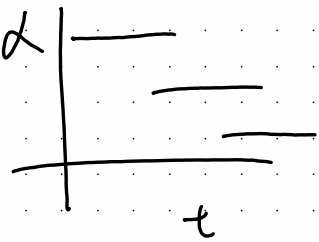
ε : 10^{-8}

Learning Rate Decay

slowly reduce α (avoid overshoot when close to solution)

$$\alpha = \frac{1}{1 + \text{decay rate} \times \text{epoch num}} \alpha_0 \Rightarrow \begin{matrix}\text{para.} \\ \alpha_0 \\ \text{decay rate}\end{matrix}$$

- $\alpha = 0.95^{\text{epoch num}} \cdot \alpha_0$ - exponential decay
- $\alpha = \frac{k}{\sqrt{\text{epoch num}}} \cdot \alpha_0$ or $\frac{k}{\sqrt{t}} \cdot \alpha_0$

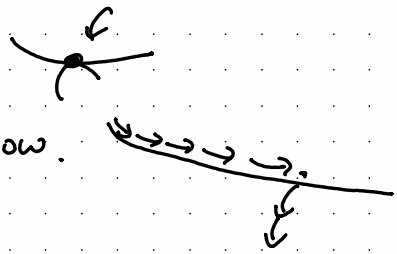


discrete decay

- manual decay (manually update...)

Local optima

most local optima happens at saddle point



- unlikely to stuck in bad local
- plateaus can make learning slow.

Hyperparameter tuning

α . β . $\beta_1, \beta_2, \varepsilon$. ✱ α

- # layers
- # hidden units — second important
- learning rate decay — 3rd
- mini-patch

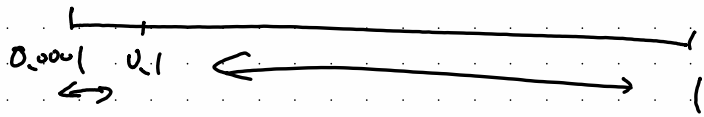
- use "random" search, not grid

- coarse to fine

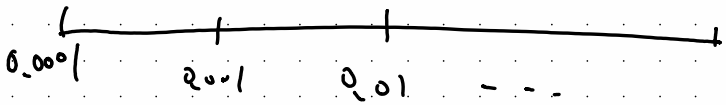
appropriate scale to pick hyper-parameter.

$n^{[l]}$ ⌊ uniform sample works

α.



in log scale



$r = -4 * np.random.rand() \leftarrow r \in [-4, 0]$

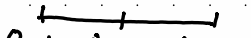
$\alpha = 10^r$

$\in [10^{-4} \dots 10^0]$

Hyperparameter for EWA (exponentially weighted averages)

$\beta = 0.9 \dots 0.999$

$1-\beta = 0.1 \dots 0.001$



$r \in [-3, -1]$

$1-\beta = 10^r$

$\beta = 1 - 10^r$

$\frac{1}{1-\beta}$ ← when $\beta \to 1$.

Small changes have big impact.

?

$\beta = 1 - 10^{(-r-1)}$

Hyperparameter tuning

Babysitting one model ← large model / fewer resource — panda

Training many models in parallel — cavier.

Batch Normalization (single layer)

can we normalize $a^{[l]}$ so that train $w^{[l+1]}$, $b^{[l+1]}$ faster

$z^{(1)} \dots z^{(m)}$

$z^{[l](i)}$

$$\mu = \frac{1}{m}\sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_i (z_i - \mu)^2$$

$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$

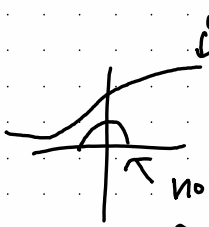
γ, β → learnable hyperparameter

if $\gamma = \sqrt{\sigma^2 + \varepsilon}$

$\beta = \mu$

then $\tilde{z}^{(i)} = z^{(i)}$

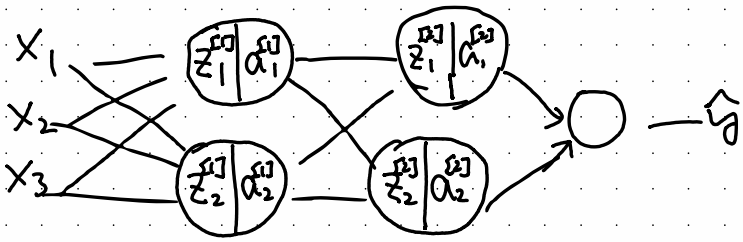
⇒ use $\tilde{z}^{(i)}$ instead of $z^{(i)}$



avoids value squeeze in small area

(by spreading the value)

Batch norm in a NN



$$X \xrightarrow{w^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{BatchNorm}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow a^{[1]} = g^{[1]}(\tilde{z}^{[1]})$$

$$\xrightarrow{w^{[2]}\ b^{[2]}} z^{[2]} \xrightarrow[BN]{\beta^{[2]}\ \gamma^{[2]}} \tilde{z}^{[2]} \rightarrow a^{[2]} \rightarrow \dots$$

Parameters:

$$w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}, \dots w^{[l]}, b^{[l]}$$
$$\beta^{[1]}, \gamma^{[1]} \dots \quad \beta^{[l]}, \gamma^{[l]}$$

this β is different from β in Momentum ADAM etc

(tf.nn.batch-normalization

WORKING with MINI-BATCH

$X^{\{1\}}$ ——

$X^{\{2\}}$ ——

parameter: $w^{[l]}, b^{[l]}, \beta^{[l]}, \gamma^{[l]}$ $(n^{[l]}, 1)$ $(n^{[l]}, 1)$

$$\rightarrow z^{[l]} = w^{[l]} a^{[l-1]} + \cancel{b^{[l]}}$$

Batch-Normalization eliminate

$$\Rightarrow z^{[l]} = w^{[l]} a^{[l-1]}$$

$$\tilde{z}^{[l]} = \gamma^{[l]} z^{[l]}_{norm} + \beta^{[l]}$$

IMPLEMENTATION

for t=1 ... #MINI BATCH
compute forward prop on $X^{\{t\}}$
in each layer, use BN to replace $z^{[l]}$ with $\tilde{z}^{[l]}$
use backprop to compute $dw^{[l]}$, ~~$db^{[l]}$~~, $d\beta^{[l]}$, $d\gamma^{[l]}$
update para $w^{[l]} = w^{[l]} - \alpha dw^{[l]}$

$$\beta^{[l]} = \beta^{[l]} - \alpha d\beta^{[l]}$$

⋮

work with momentum, RMSprop, ADAM etc

Batch norm work bc?.

- learning on shifting of distribution
  'covariance shift' e.g. training on black cat ↓ hard to generalize to yellow cat.

△ Batch norm reduce "cov shift"

△ Batch norm as regularization
- mean/var of each mini-batch differ
- add noise to value $z^{[l]}$ ($\tilde{z}^{[l]}$)
- similar to dropout, it add noise to each hidden layer's activation
→ slight regulation

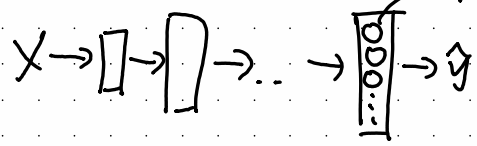
Batch norm at test time

μ, σ^2 for testing: exponentially weighted average across all mini-batch

softmax regression

$p(\text{class } i \mid x)$

$n^{[L]} = \#\text{class}$



e.g. 4 class

$z^{[L]} = w^{[L]} a^{[L-1]} + b^{[L]}$ $\quad (4,1)$

Activation Function

$t = e^{(z^{[L]})}$

$a^{[L]} = \frac{e^{z^{[L]}}}{\sum_{j=1}^{4} t_i}$, $a_i^{[L]} = \frac{t_i}{\sum_{j=1}^{4} t_i}$

hard max

$\begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$

$a^{[L]} = g^{[L]}(z^{[L]})$

$(4,1)$ $\quad (4,1)$

softmax activation

vector → vector.

others output real number.

$C=2 \rightarrow$ soft max reduced to logistic regression

Loss function

$C=4$, $y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ $\hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$

$L(y, \hat{y}) = -\sum_{j=1}^{4} y_j \log \hat{y}_j$

e.g. $-y_2 \log \hat{y}_2$

reduce loss → $\hat{y}_2 \uparrow$

m sample

maximum likelihood estimation

$J(w^{[1]}, b^{[1]}, \dots) = \frac{1}{m} \sum L(\hat{y}^{(i)}, y^{(i)})$

Deep Learning Frameworks

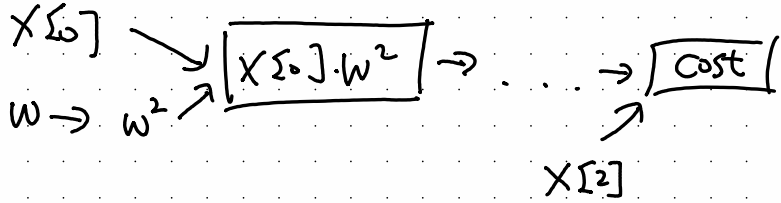
TensorFlow

$(w-5)^2$, $w-5=0$

$J(w) = w^2 - 10w + 25$

& only need to calculate forward prop.

Gradient Tape for back prop



Course 3

STRUCTURING ML Project

Orthogonization

- many potential tools

Fit training set well on cost function
- bigger network
- adam
:

× early stopping

Fit dev set well on cost function
- Regularization
- bigger training set

Fit test set well on cost function
- Bigger dev set

perform well in real-world
- change dev set or cost function

Single Number Value Evaluation

- precision → recognized cat, % of them as cat
- recall → % of actual cat got recognized

Competing.. hard to evaluate

⇒ F1 score ~ average of P & R

$\frac{2}{\frac{1}{P}+\frac{1}{R}}$ "Harmonic Mean"

Satisficing & Optimizing Metric

e.g. maximizing accuracy ← optimizing

subject to running time < 100ms ← satisficing

e.g. wake word / trigger word

'hi siri'

maximize accuracy

s.t. ≤ 1 false positive every 24 hrs.

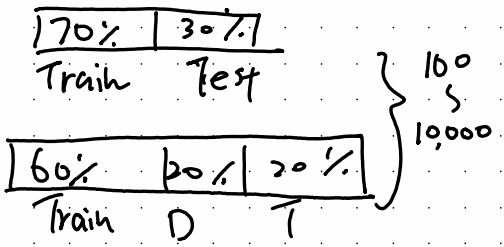
Train / Dev / Test set

Dev / Test should have same distribution

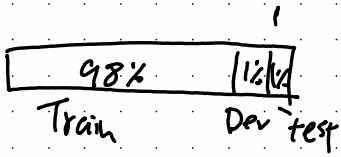
choose Dev/Test that reflect your future target

Size of dev/test sets

Old



1,000,000 data



size of test set ~ big enough to give high confidence on performance

- train | dev ← sometimes not recommanded
- train | test ✗ not good naming...

When to change Dev/Test and Metrics

real data or different from expectation

- metric + dev prefer A
- your user prefer B

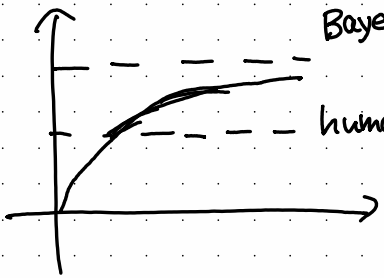
⇒ time to change

e.g. bigger penalty on undesired image

orthogonality

- place target (define metrics)
- shoot at it (change cost func... etc)

human-level performance



→ progress slows when surpass human

if ML is worse
- get more labels
- gain insight from manual analysis
- better analysis of bias / variance

avoidable bias

△ assume human-level error ≈ bayes error (esp. CV tasks)

△ human	1%	7.5%
training error	8%	8%
dev error	10%	10%
	↑ reduce bias	↑ reduce variance

↕ avoidable bias

define human-level error

human-level error as proxy to bayes error.

e.g. human
doctor
group of doctors ← closer to bayes error

theoretical upper bound

human-level error
↕ avoidable bias
training error
↕ variance
dev error

surpassing human-level error

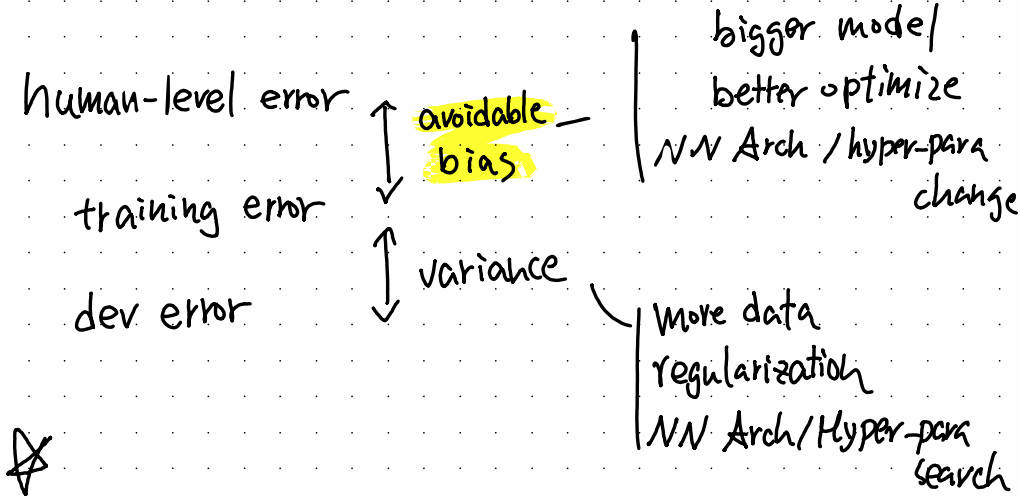
- online ad
- product recommand
- logistic

learned from structure data
not involve perception
lots of data

speech recog
medical data

fundamental assumption of supervised learning

- one can fit training set well
- training set generalizes to dev/test.



Error Ananlysis

cat vs dog. if find mislabelled ones are dog.

Analysis

1. get ~100 mislabeled dev set example
2. count how many are dogs
   - if low, probably not worth it.

⇒ build table for incorrectly labelled dev set.

Image	Dog	big cat	blurry .. ← in
1	✓		
⋮			
		✓	
n			

% ↑

based on percentage, decide what's next step.

Incorrect Labelled Data

- robust at "random" error in training data
- for dev/test, label accuracy might be less important. (based on the err analysis table)

correcting dev/test labels

- apply same process for both dev/test set (ensure same distri)
- consider examples your algorithm got 'right' as well as 'wrong'
- note training and dev/test might come from different distri

Build system quickly then iterate

- set dev/test set early
- build
- bias/var/error analysis
  - esp for new applications

Training / Dev from different distri

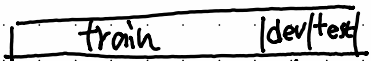
e.g.

cat from webpage ~200,000

web from mobile (blur etc) ~10,000 ← what we care

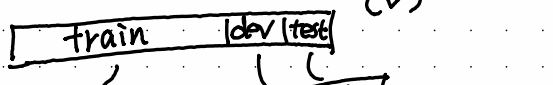
option:

1. 200k+10k = 210k (X)



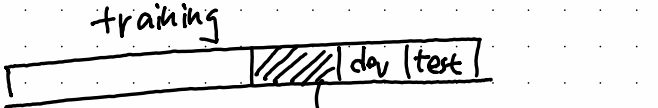
2,500 → $2500 \times \frac{10}{210} = 119$ mobile image.

2. (V)



200k+5k
web
mobile

Bias & variance with mismatched data dist.

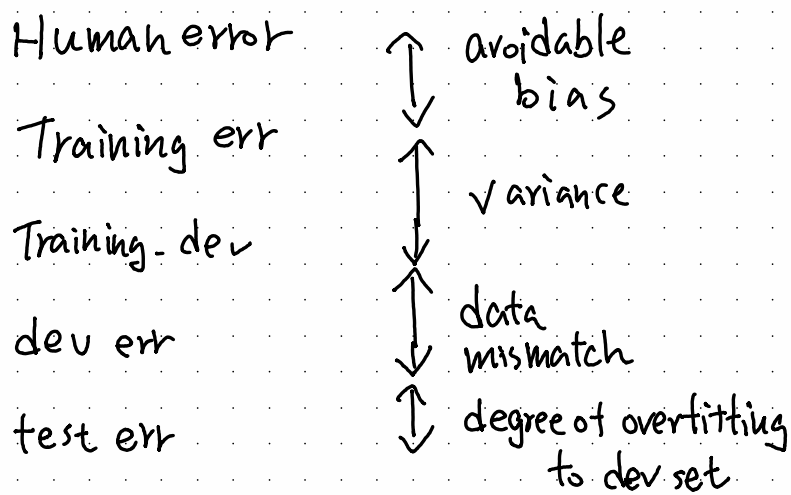


train on this

training-dev set

same dist as training set

human err	0%			
Train	1%	1%	10%	10%
train-dev	9%	1.5%	11%	11%
dev	10%	10%	12%	20%
	variance problem	data mismatch	bias problem	bias + data mismatch



Sometimes dev performs better than training

- e.g. dev/test are of different distribution
- additional label on the dev/training might better understand human performance

Addressing Data mismatching

- manual error analysis to understand the diff between training, dev/test.
- making training data more similar

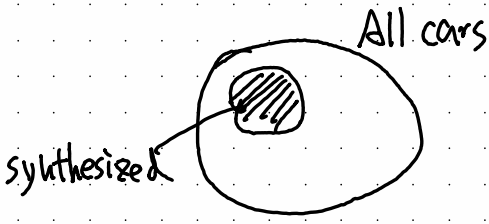
↓

Artificial Data Synthesis

e.g. speech + car noise = synthesised in car speech

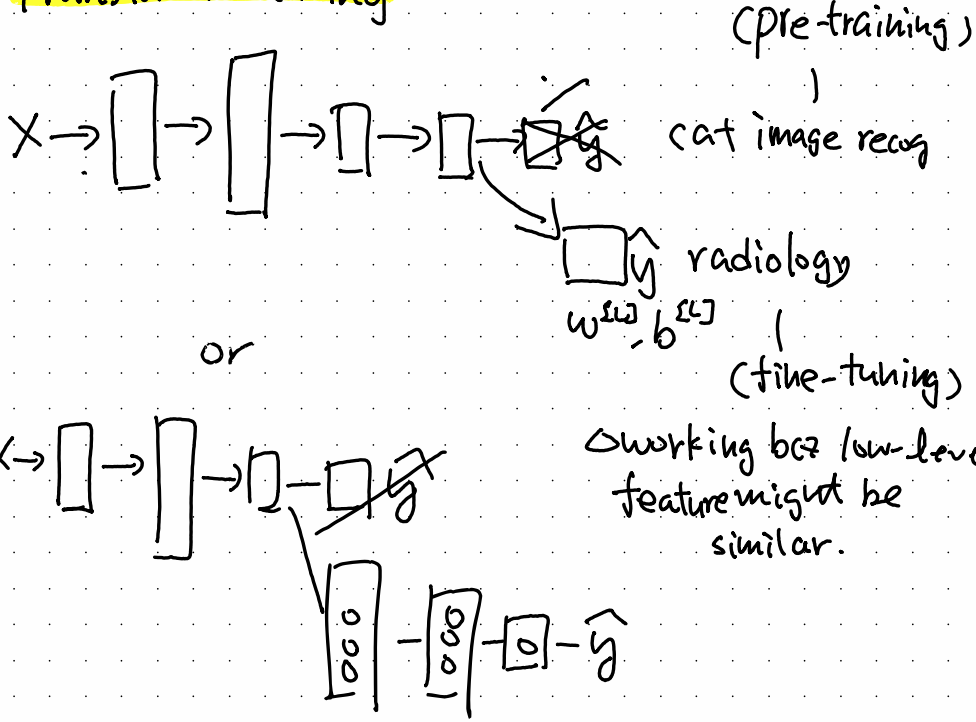
note: 10,000 hrs (speech), 1 hr (car noise), might be overfitting the 1hr car noise

e.g. car photo



" ↖ if using syn data, make sure it capture enough variety.

Transfer Learning



Transfer from A → B. useful when.

- task A & B have the same input
- much more data in A
- lower level feature of A is helpful for B

Multi-task Learning

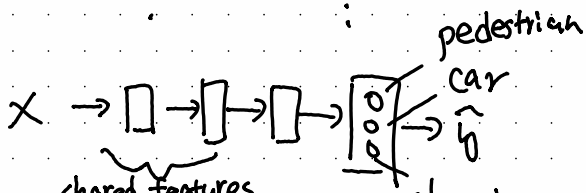
(less often used) except CV. Compared to TL

e.g. autonomous driving

△ one image with multiple labels

pedestrian
cars
stop sign
⋮

$$\begin{bmatrix}1\\0\\0\\\vdots\end{bmatrix}$$



$$\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{3} L(\hat{y}_j^{(i)}, y_j^{(i)})$$

$$-y_j^{(i)}\log\hat{y}_j^{(i)} - (1-y_j^{(i)})\log(1-\hat{y}_j^{(i)})$$

△ might still work when some images only have partial labels

MAKE SENSE WHEN:

- tasks share similar low-level features
- amount of data for each feature are similar
- can train a big enough NN to do well on all tasks

End-to-End DL

✗ audio $\xrightarrow{\text{MFCC}}$ features $\xrightarrow{\text{ML}}$ phonemes → words → transcript

audio ⟶ transcript

need a lot of data!

---

e.g.

Face recognition

image $\xrightarrow[\text{face detection}]{\text{tracking}}$ face $\xrightarrow{\text{ML}}$ recognition

& have lot data for each of 2 sub-tasks.
even more data for end-to-end data.

Machine Translation

✗ Eng → text analysis → ... → French

✓ Eng ⟶ French

Estimate child age

✓ image $\xrightarrow{①}$ bone $\xrightarrow{②}$ age

✗ image ⟶ age

wheather to use End-to-End DL

Pro:
- let the data speak
- less hand-design features

Con:
- data hunger
- exclude hand-designed features

△ hand-design feature helps when traing set is small

KEY Q:
sufficient data to learn a function maps X to Y?

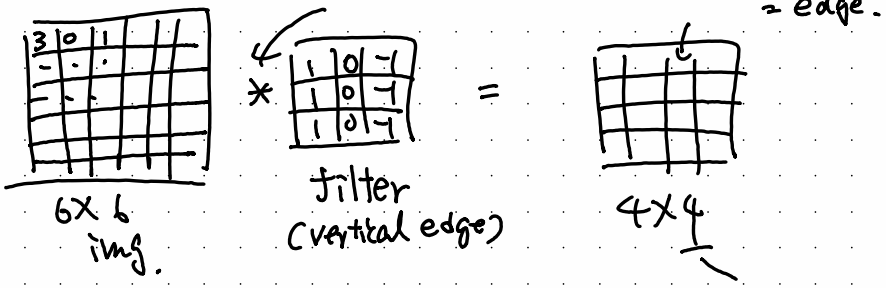
Course 4

Convolutional Neural Networks

CovNet

Computer Vision

Edge Detection



python: conv-forward

tensor: tf.nn.conv2d

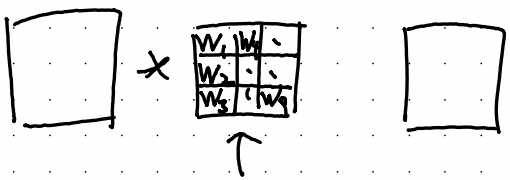
⋮

sobel filter

1	0	-1
2	0	-2
1	0	-1

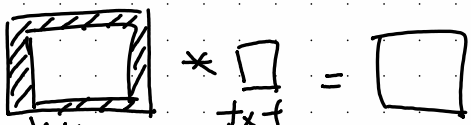
Schors filter.

3	0	-3
10	0	-10
3	0	-3



use NN to learn filter

Padding



valid (no padding) — $n \times n$ * $f \times f$ → $n-f+1 \times n-f+1$

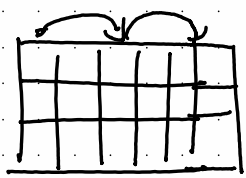
same (padding) — $(n+p) \times (n+p)$ * $f \times f$ → $n+2p-f+1 \times n+2p-f+1$

$$p = \frac{f-1}{2}$$

e.g. $f=3 \rightarrow p=1$, $f=5 \rightarrow p=2$

f is almost always odd (3×3, 5×3, ⋮)

Strided convolution

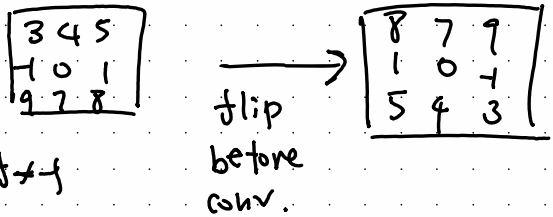


$n \times n$ * $f \times f$

pad: p, stride: s

$$\Rightarrow \left\lfloor \frac{n+2p-f}{s}+1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s}+1 \right\rfloor$$

Convolution in math

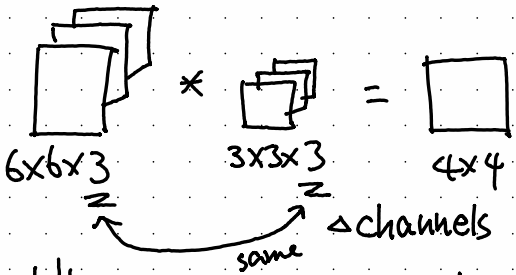


cross-corr : no flip filter

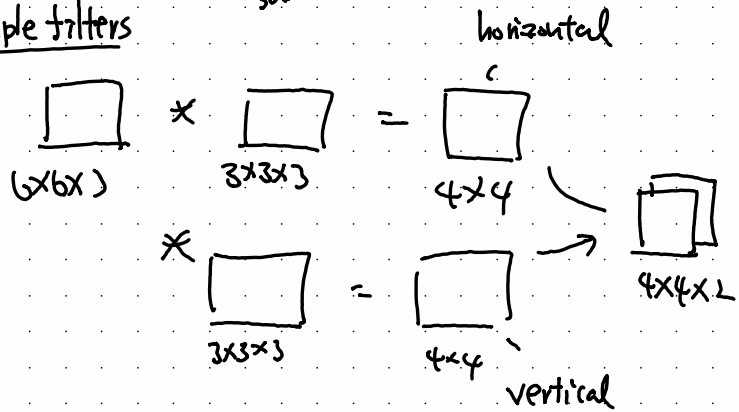
conv : flip filter

ML community just call cross-corr as convolution

Conv in volume

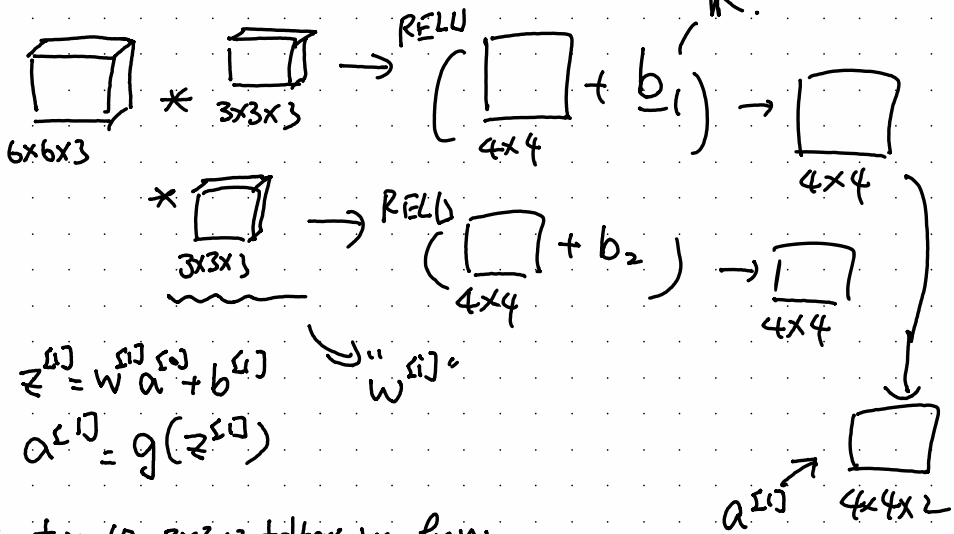


multiple filters



$n \times n \times n_c \quad * \quad f \times f \times n_c \quad = \quad n-f+1 \times n-f+1 \times n_c$

one-layer CNN



$z^{[1]} = w^{[1]} a^{[0]} + b^{[1]}$

$a^{[1]} = g(z^{[1]})$

e.g. for 10 3x3x3 filters, we have

$(3 \cdot 3 \cdot 3 + 1) \times 10 = 280$ parameters

* parameters are independent from image resolution

Notation

$f^{[l]}$ - filter
$p^{[l]}$ - padding
$s^{[l]}$ - stride
$n_c^{[l]}$ - number of filters

each filter:
$f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

Input: $n_H^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]}$

output: $n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

$$n_w \left\lfloor \frac{h_w^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

Activation

$a^{[l]} \rightarrow n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$ $A^{[l]} \rightarrow m \times n_H^{[l]} \times h_w^{[l]} \times n_c^{[l]}$

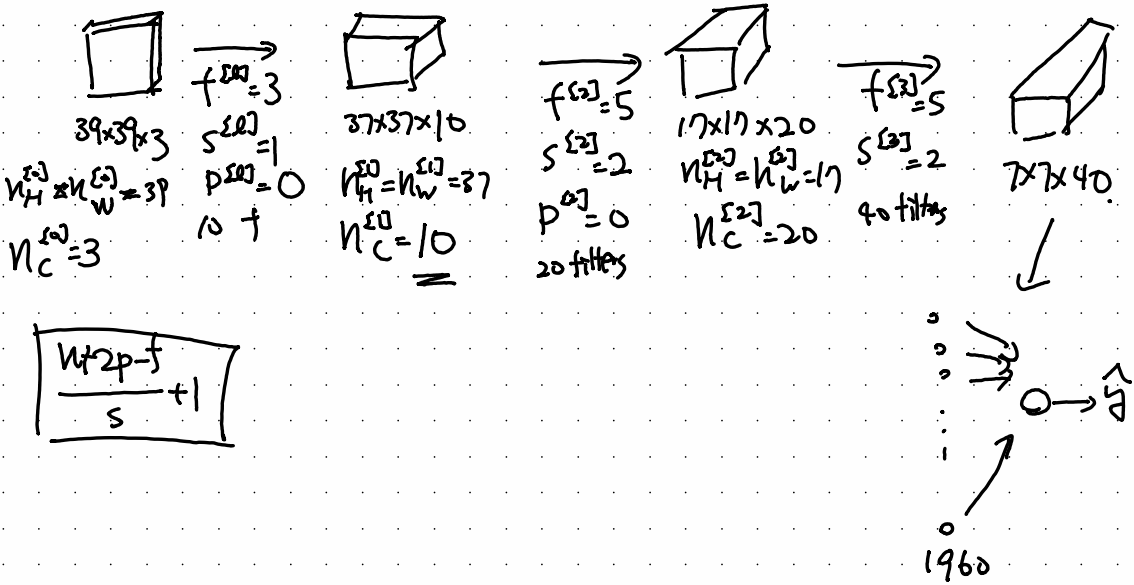
weights

$f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

bias:

$h_c^{[l]} - (1,1,1,h_c^{[l]})$

CNN



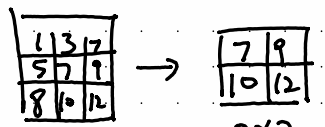
$$\frac{n+2p-f}{s}+1$$

TYPE of LAYERS in CNN

- Convolution (Conv)
- Pooling (Pool)
- fully connected (FC)

POOLING LAYER

Max pooling - preserve feature?



4×4 f-2 s-2 → 2×2

△ no parameters to learn △

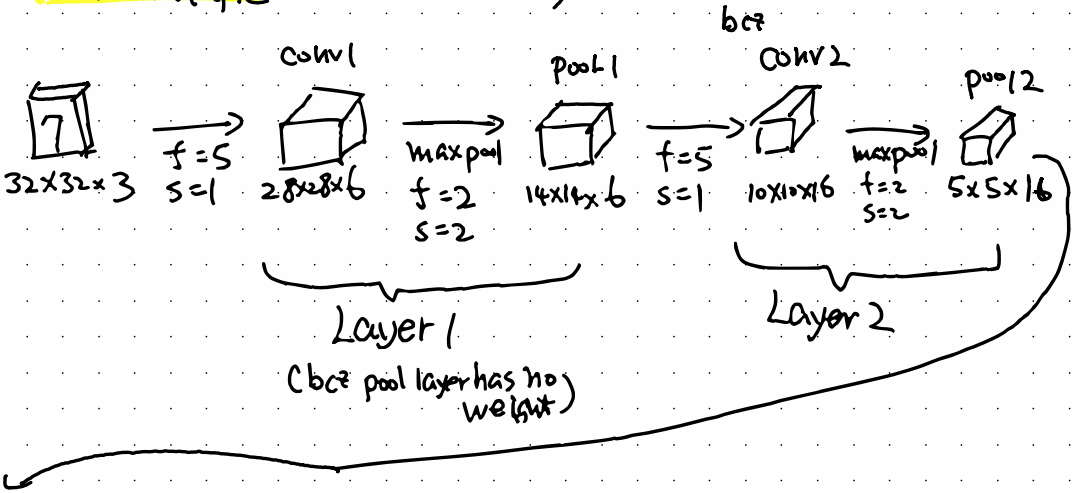
Average pooling - not often used

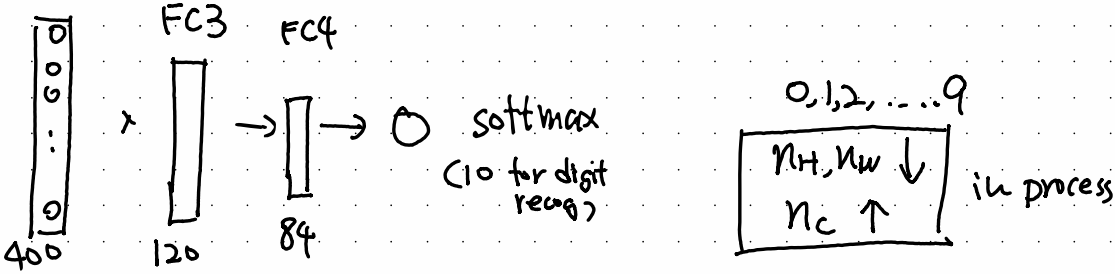
hyperparameters
- f - filter size (f=2, s=2; f=3, s=2)
- s - stride
- type of pooling layer

$\left\lfloor \frac{n-f}{s} + 1 \right\rfloor$

* no parameter to learn

NN example (LeNet-5)





$W^{[3]}$ (120, 400)

$b^{[3]}$ (120)

△ FC layers have lots parameters!

✡ Why conv? ✡

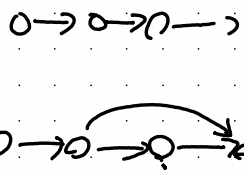
reduce # parameters compared

- parameter sharing - feature detector useful for all parts of image
- sparsity of connections - each output value depends on small part of image

△ translation invariant.
(pooling layer)

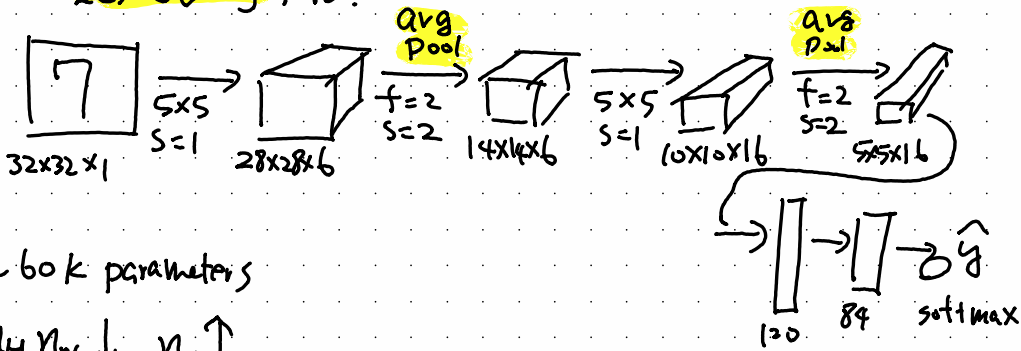
Keras.

- sequential API
- functional API



CASE STUDY

LeNet-5 . 98.



~ 60k parameters

$n_H\ n_w \downarrow$ $n_c \uparrow$

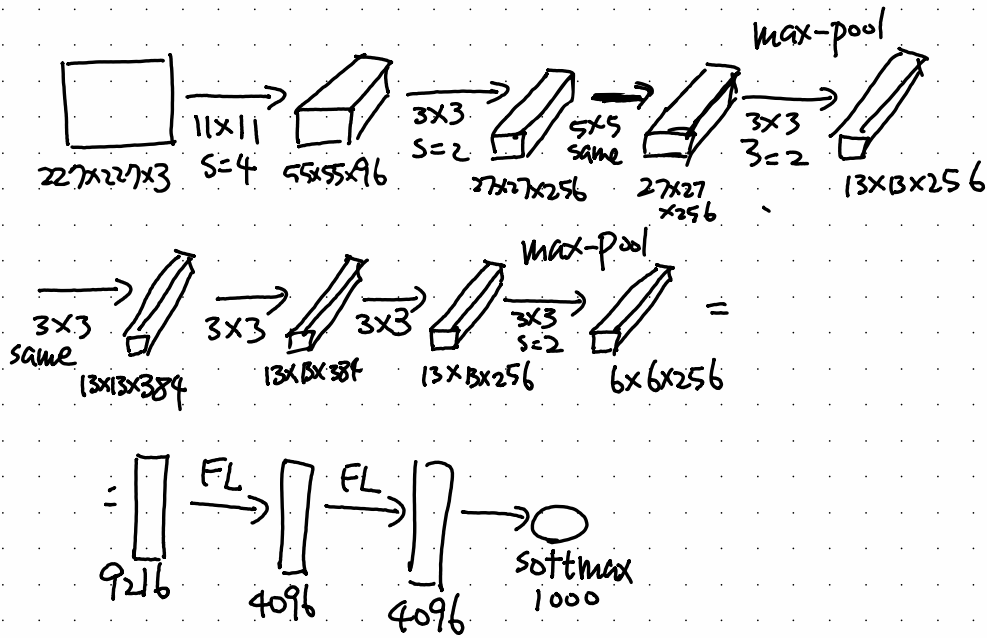
conv pool conv pool fc fc pool

original

sigmoid/tanh
different filter
nonlinearity after pooling

section II, III

AlexNet. 2012



~ 60M parameters

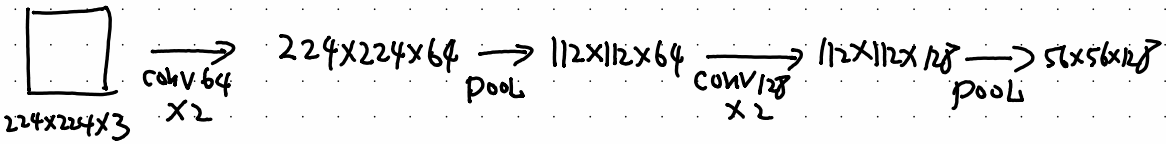
- ReLU

Orig

- multiple GPU
- local response normalization (LRN)
  - normalize along axis
  - less often used.

VGG-16, 2015

CONV = 3×3, S=1, same, MAX-POOL = 2×2, S=2



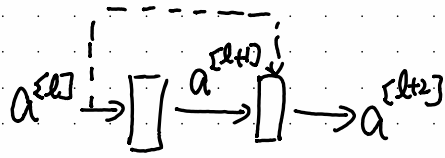
~138M parameters LARGE!

~uniform

△ nH nW ↓ nC ↑

Residual Network (ResNets) 2015

Residual Block.

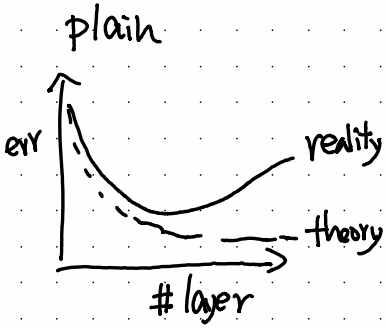


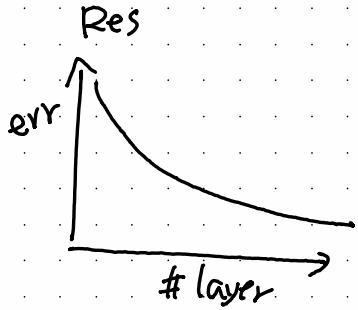
SHORT CUT / SKIP CONNECTION

$a^{[l]}$ → linear → ReLU → $a^{[l+1]}$ → linear → ReLU → $a^{[l+2]}$

$z^{[l+1]} = w^{[l+1]} a^{[l]} + b^{[l+1]}$, $a^{[l+1]} = g(z^{[l+1]})$, $z^{[l+2]} = w^{[l+2]} a^{[l+1]} + b^{[l+2]}$, $a^{[l+2]} = g(z^{[l+2]})$

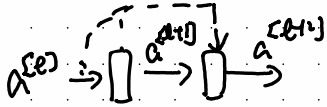
ResNet $\Rightarrow a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$





△ help training deeper network.

Why ResNet Work?



$W_s a^{[l]}$ if dimension changes.

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

$$= g(w^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]})$$

IF $w^{[l+2]} \to 0$, $b^{[l+2]} \to 0$

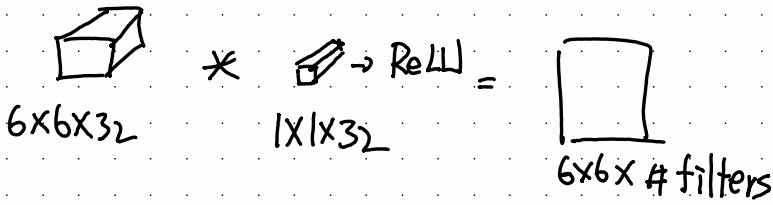
$\Rightarrow a^{[l+2]} = g(a^{[l]})$

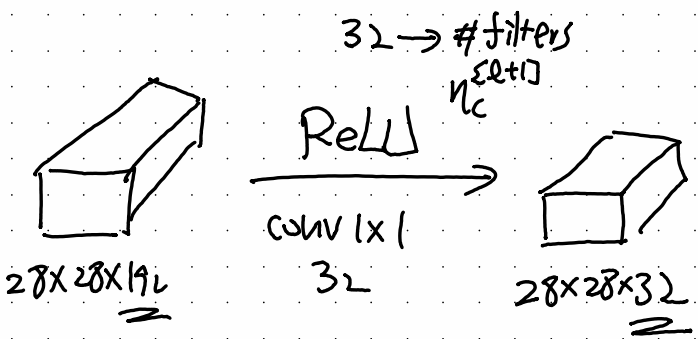
→ identity function is easy to learn

→ ① res block doesn't hurt.

→ ② prevent deeper layers making result worse.

1×1 convolution / network in network



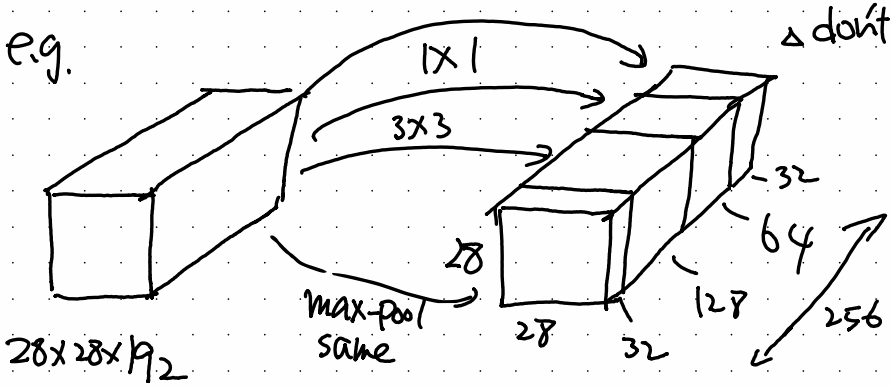


1×1 → shrink n_c
filter → shrink w, h.

shrink n_c, if we like.

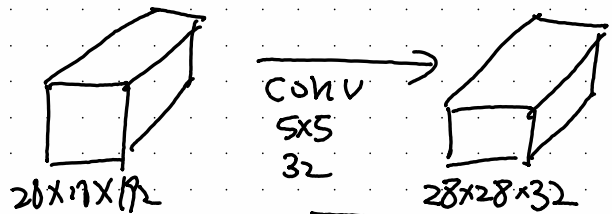
Inception Network, 2014

e.g.



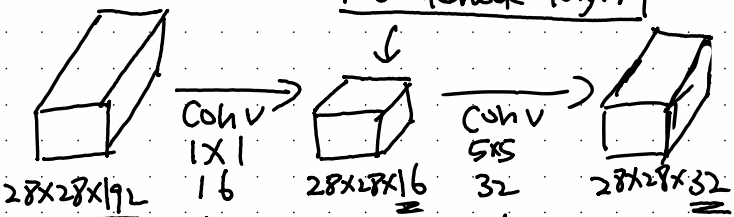
△ don't confuse w. perception.

△ try all different size at once!



#op
28×28×192×28×28×32 ≅ 120M!

bottleneck layer



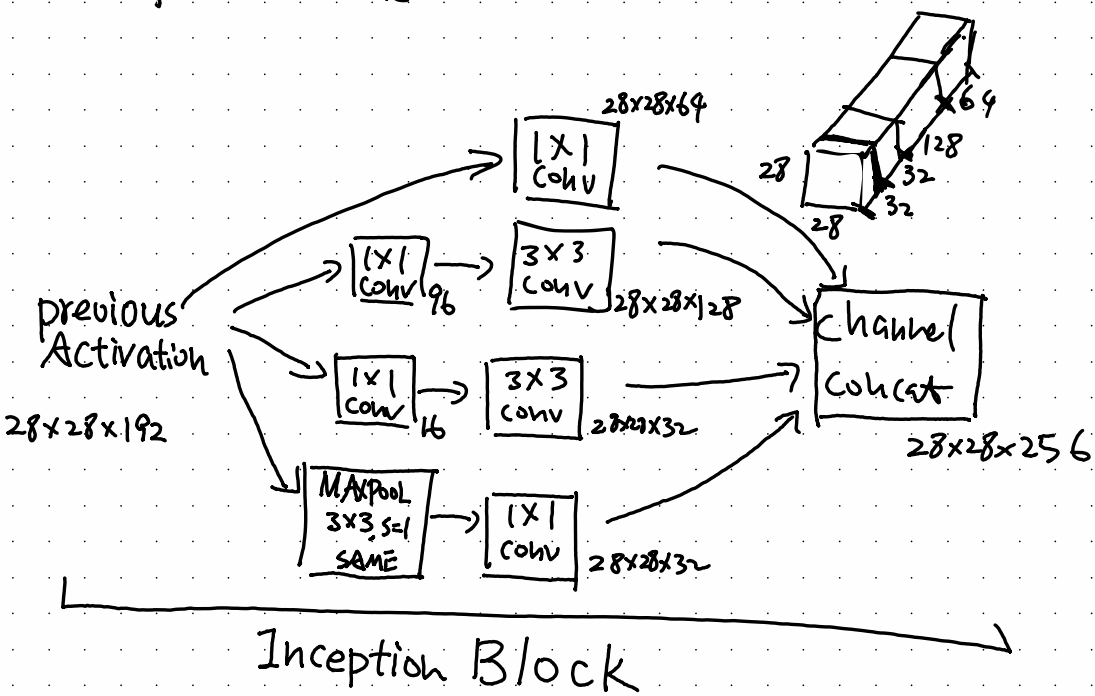
← Inception

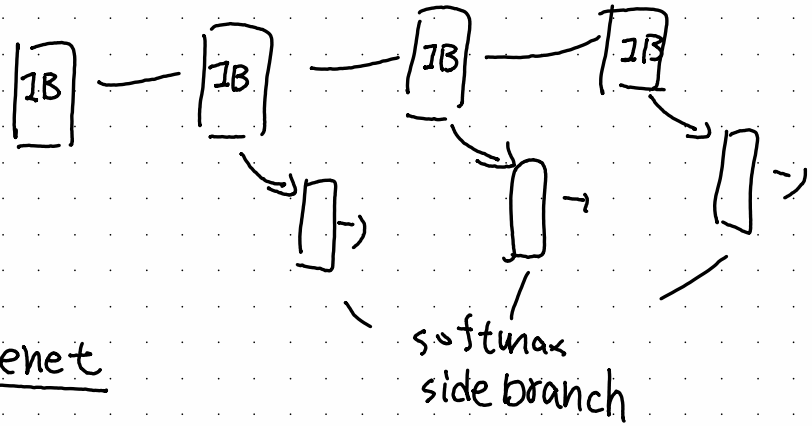
28×28×192×16
= 2.4M

28×28×16×5×5×16 ≅ 10M

⇒ ~12.4M

Inception Module





goolenet

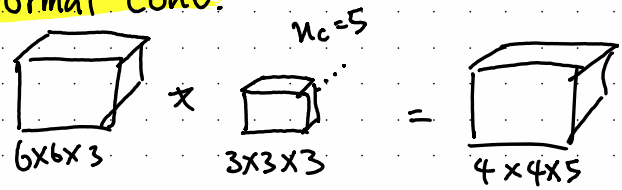
"we need to go deeper"

Inception

MobileNet. 2017

- low computational cost.
- Normal v.s depthwise-separable convolution
  ⇒ depth-wise + point-wise.

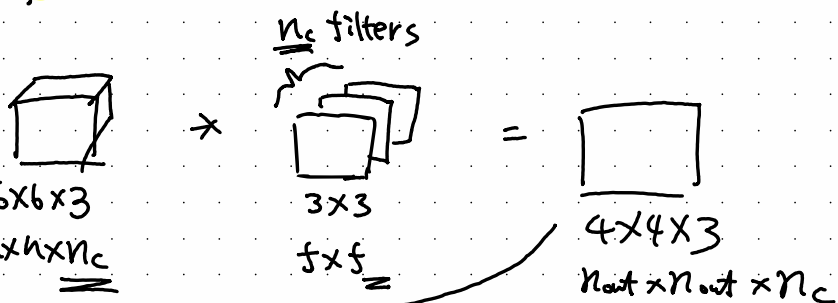
Normal Conv.



cost = # filter para x # filter pos x # filter

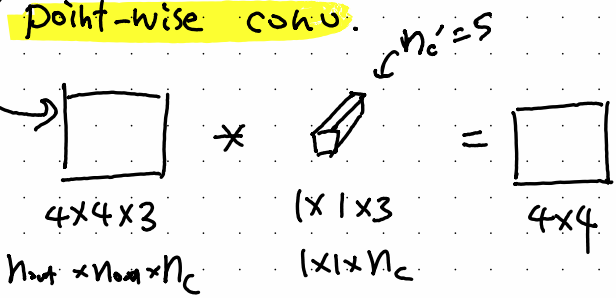
= 2160 op.

Depth-wise Conv



→ 432 op

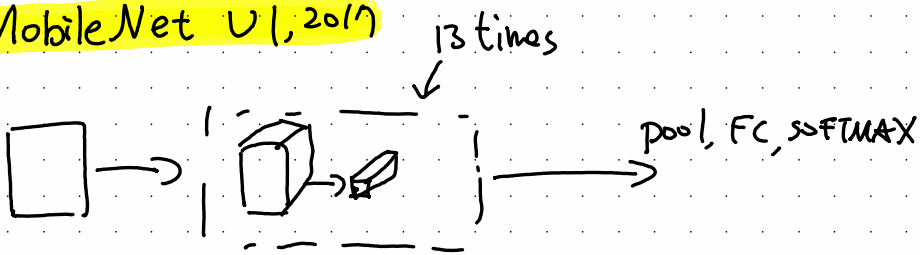
Point-wise conv.



→ 240 op

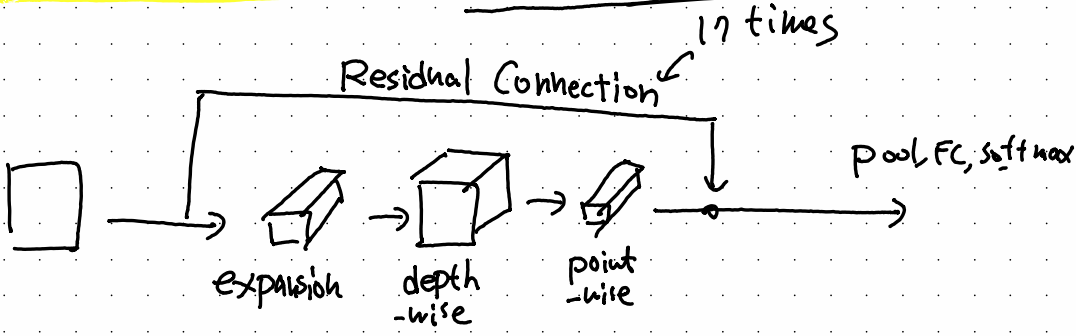
~10 times cheaper

MobileNet V1, 2017

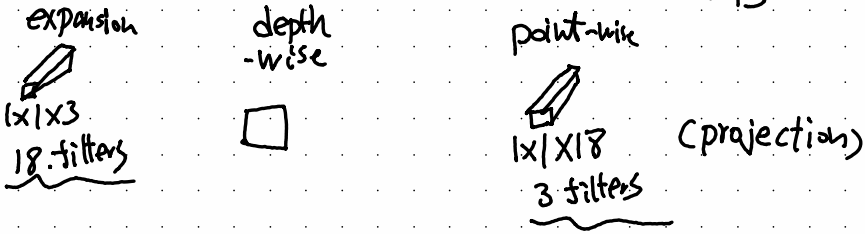


MobileNet V2, 2019

bottleneck blocks



eg. $n \times n \times 3 \rightarrow n \times n \times 18 \rightarrow n \times n \times 18 \rightarrow n \times n \times 3$



- expansio → increase representation in bottleneck block
- projection → reduce bandwidth between layers

EfficientNet 2019

- automatic scale up/down model

scale
- r resolution
- d depth
- w width

the model selects r,d,w that suits device.

Transfer Learning

precompute this, to accelerate (avoid forward pass)

① $X \to [] \to [] \to \dots \to [] \to \hat{y}$ (Softmax)

ImageNet weights

freeze all

$\to \hat{y}$ (Softmax) our new lables.

② $X \to [] \to [] \dots \to [] \to [] \to \hat{y}$

train

① ② if small dataset.

$\to y$

$\to [] - [] - [] \to \hat{y}$

★ always start from existing weights.
for CV problems

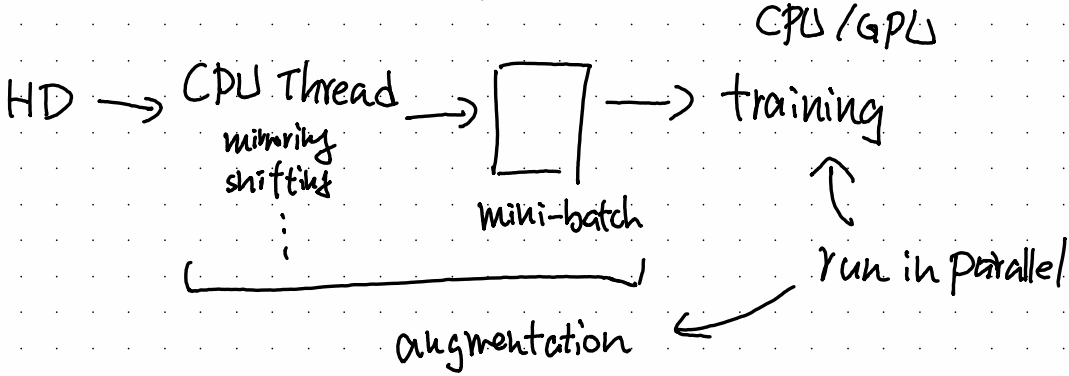
③ $X \to [] \to \dots \to [] \to \hat{y}$

use previous weights
as initialization
(if we have a large data set)

Data Augmentation

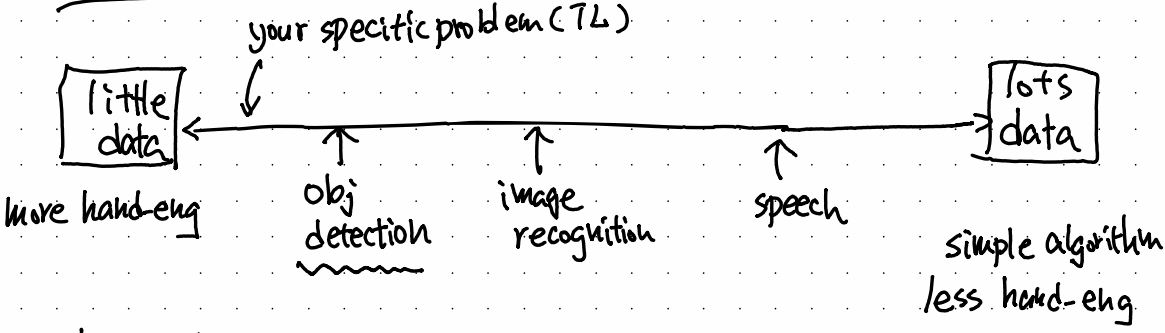
- mirroring
- random cropping
  - might crop some less informative area
- color shifting
  - PCA color augmentation
  - simple RGB shifting, e.g different sunlight.

Distortion during training



State for CV.

data vs hand-engineering (hack)



knowledge
- label
- hand engineered features / network architecture / ...

Tips for doing well on benchmarks

- Ensembling (3~15 networks)
  - train several networks and average outputs ($\hat{c}_j$)
    (less useful in practical)
- Multi-crop at test time
  - run classifier on multiple vers of test images and avg results
    e.g. (10-crops)

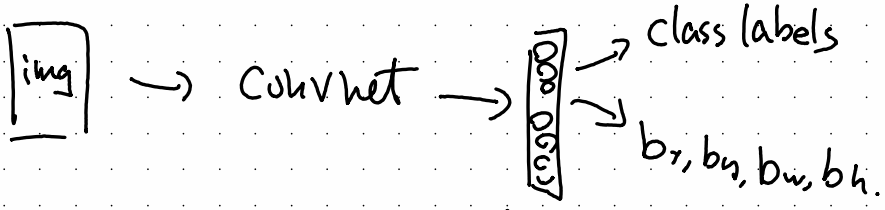
use open source code

- use network arch.
- use implementation
- use pre-trained model

object detection

- classification - car, cat etc
- localization - bounding box
- detection - multiple objs of different categories.

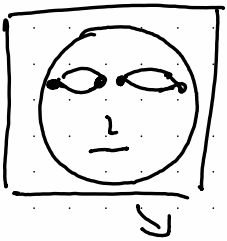
classification with localization



$$y = \begin{bmatrix} P_c & \text{prob of objects exist} \\ b_x & \\ b_y & \text{bb} \\ b_w & \\ b_h & \\ c_1 & \\ c_2 & \text{Category of the obj} \\ \vdots & \end{bmatrix}$$

$$L(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + \dots (\hat{y}_j - y_j)^2 & \text{if } y_1 = 1 \\ (\hat{y}_1 - y_1)^2 & \text{if } y_1 = 0 \end{cases}$$

Landmark detection



eyes: l_{1x}, l_{1y}
l_{2x}, l_{2y}
⋮
l_{4x}, l_{4y}

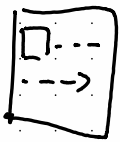
additional landmarks e.g. mouse. nose ⋮

ConvNet →  each out a landmark on face

- same for pose detection
  skeleton landmarks in a photo.

obj detection

sliding window detection

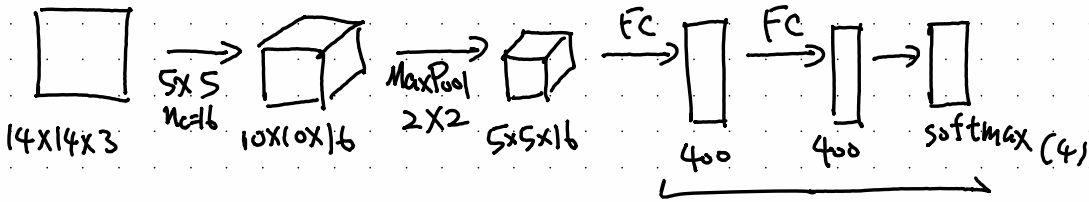


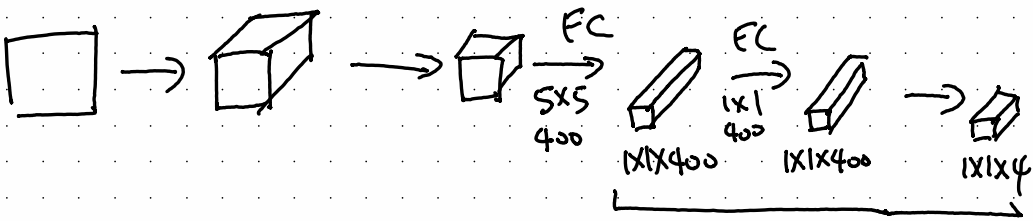
send each window through ConvNet.
from small to large window size

- high computation cost.

Convolutional Implementation

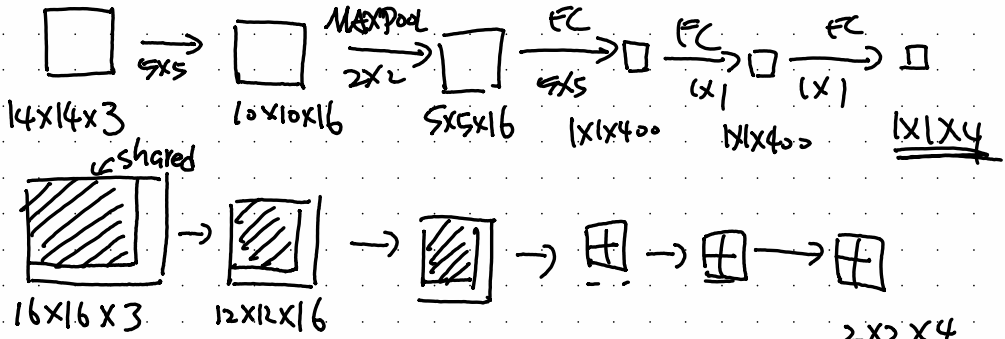
turn FC to convolutional layer





OverFeat. 2014.

Conv implementation



→ reuse overlapped computation result.

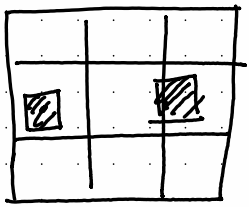
2x2x4 corresponding to 4 sliding window position

→ instead of sequentially send each sliding window through ConvNet, simultaneously do prediction for all window position

→ the window position is not precise though.

Output accurate bounding boxes

YOLO- You only look once, 2015 (a difficult paper to read)



100

100

target output

3x3x8

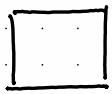
grid

P, b, c

for each grid cell

$$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_w \\ b_h \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

- assign obj to grid based on mid-point
- relative coord to the grid cell
- b_x, b_y - between 0~1
  b_w, b_h - could be > 1



100x100x3

→ conv → max pool →

3x3x8

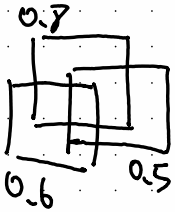
⇒ increase grid resolution if need higher precision.

~ still one convnet implementation

Intersection over Union (IoU)

correct if IoU > 0.5

Non-max suppression



find max → supress others with high IoU

each grid cell $\begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$

discard all boxes with $P_c \leq 0.6$

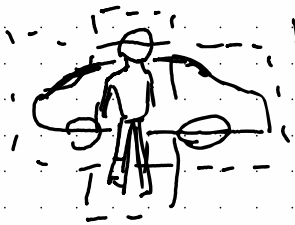
for remaining boxes
- pick largest P_c

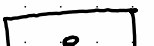
discard remaining with IoU ≥ 0.5 with other boxes.

Anchor boxes

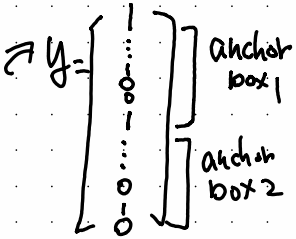
predefined shapes

box1 box2



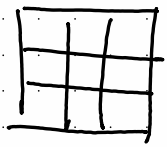


$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ c_1 \\ c_2 \\ c_3 \\ P_c \\ b_x \\ b_y \\ i \end{bmatrix}$ anchor box1, anchor box2



each obj, assigned to cell
- Contain mid-point
- anchor box for the cell with high IoU

Yolo algorithm



c
1 - pedestrian
2 - car
3. - motocycle

y $3 \times 3 \times 2 \times 8$

2 ← anchor boxes

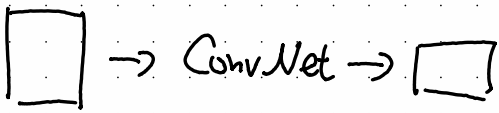
8 ← 5 + #classes

$$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_w \\ b_h \\ c_1 \\ c_2 \\ c_3 \\ P_c \\ b_x \\ b_y \\ b_w \\ b_h \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

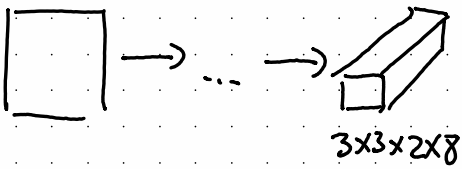
anchor box 1

anchor box 2

Training



Prediction



$$y = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \\ 0.2 \\ 0.3 \\ 0.2 \\ 0.1 \\ 1 \\ 0 \\ 0 \\ \vdots \\ ? \end{bmatrix}$$

non-max supression

for each cell,

- get 2 predicted bb (for each anchor boxes)
- get rid of low P_c prediction
- for each class, run non-max supression

Region Proposals. R-CNN. 2013

- pick interesting region to run CNN

segmentation → run CNN on interesting regions

R-CNN: propose region, CNN, output

Fast R-CNN: propose region, Convolution implementation, output (2015)

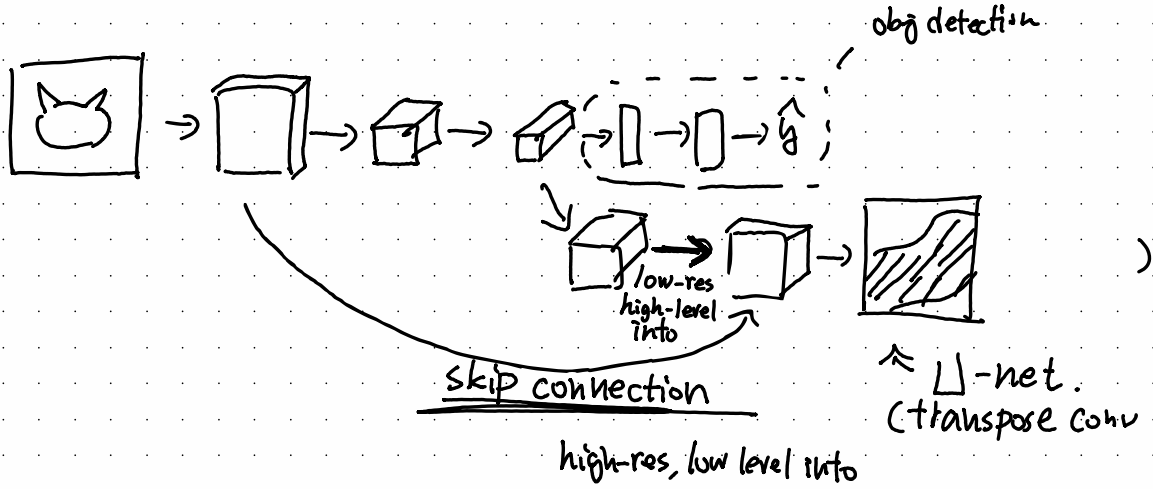
Faster R-CNN: convnet to propose regions (2016)

still slower than YOLO...

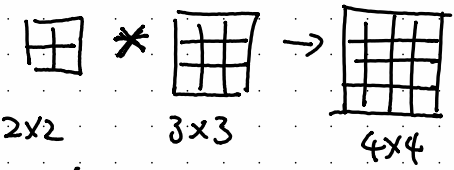
Semantic Segmentation U-net, 2017

obj detection - bb

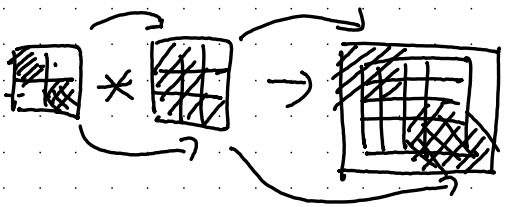
semantic seg - per-pixel labeling



Transpose Conv, deconvolution, upscale conv

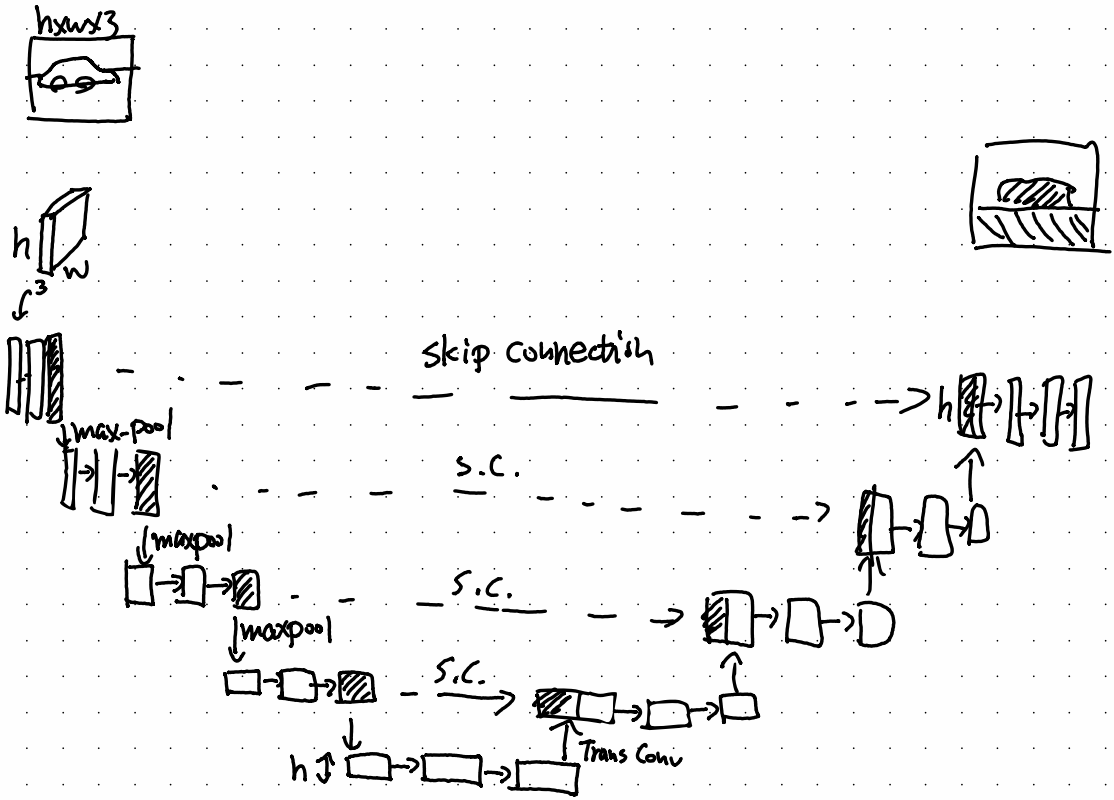


f=3, p=1, s=2



if overlap, add values together.

U-Net, 2015 (on biomedical image seg)



Face Recognition

- verification
  - input image, name
  - output whether image/name match (1 to 1)
- Recognition
  - database of k people
  - input image
  - output ID of the image among k. (1 to k)

one-shot learning

learn from one example to recognise the person again

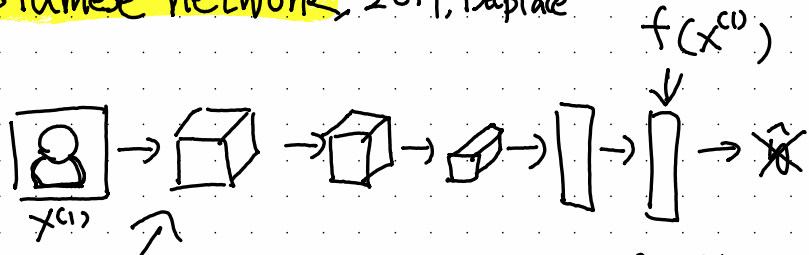
learn a "similarity" function

$d(\text{img1}, \text{img2})$ = degree of difference between imgs.

if $d(\text{img1}, \text{img2}) > \tau$, same person

$< \tau$, no match

Siamese network, 2014, DeepFace



$$d(x^{(1)}, x^{(2)}) = \| f(x^{(1)}) - f(x^{(2)}) \|_2^2$$

parameters of NN define $f(x^{(i)})$

learn parameters that

if $x^{(i)}, x^{(j)}$ are same person $d(x^{(i)}, x^{(j)})$ small/

different large

Triplet Loss FaceNet, 2015


Anchor
A


Positive
P


Anchor
A


Negative
N

WANT: $\|f(A)-f(P)\|^2 + \alpha \leq \|f(A)-f(N)\|^2$

$\|f(A)-f(P)\|^2 - \|f(A)-f(N)\|^2 + \alpha \leq 0$

α: margin
to prevent trivial solution
e.g. $f(img) = 0$
or $f(img^{(1)}) = f(img^{(2)})$,

Loss function

$L(A,P,N) = \max(\|f(A)-f(P)\|^2 - \|f(A)-f(N)\|^2 + \alpha, 0)$

$J = \sum_{i=1}^{m} L(A^{(i)}, P^{(i)}, N^{(i)})$

training set: 10k picture of 1k persons

Choosing A, P, N

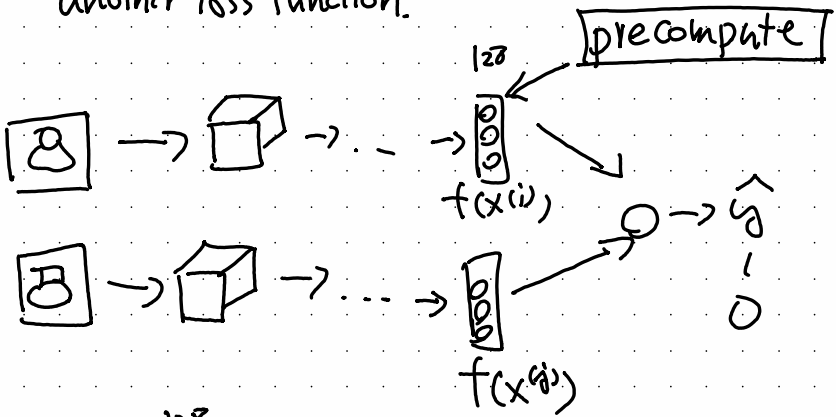
if chosen randomly, $d(A,P) + \alpha \leq d(A,N)$ is too easy.

→ choose A, P, N that are "hard" (NN learns little)
(check paper)

current K = 100m images
commercial

DeepFace. 2014

another loss function.



$$\hat{y} = \sigma\left(\sum_{k=1}^{128} w_k \left| f(x^{(i)})_k - f(x^{(j)})_k \right| + b\right)$$

- kth feature

or χ^2

Neural Style Transfer, 2015

Content (C) + Style (S) → generated (G)

Visualize ConvNet 2013

hidden unit covers larger patch →

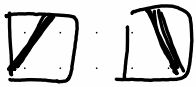
[8] → conv → conv → .... → FC → FC → $\hat{y}$

pick a unit in layer 1, find image patches that maximize unit activation

repeat for all units

(patch size ↑, for deeper layer.)

lower lv –



higher level

Neural Style Transfer. 2015

$$J(G) = \alpha J_{content}(C,G) + \beta J_{style}(S,G)$$

1. init G randomly (like noise)
2. gradient descent to minimize $J(G)$

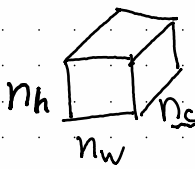
$J_{content}(C,G)$

- use hidden layer l to compute content cost (usually middle of network)
- use pre-trained ConvNet.
- Let $a^{[l](C)}$ and $a^{[l](G)}$ be activation of layer l

$$J_{cont}(C,G) = \frac{1}{2} \| a^{[l](C)} - a^{[l](G)} \|^2$$

$J_{style}(S,G)$

at layer l, define "style" as correlation between activation across channels



$a^{[l]}_{i,j,k}$ = activation at (i,j,k). $G^{[l]}: n_c^{[l]} \times n_c^{[l]}$

intuition: each channel correspond to hidden units that capture different levels of features

$$G^{[l](S)}_{kk'} = \sum_i \sum_j a^{[l]}_{ijk} a^{[l]}_{ijk'}$$ ← style

"Gram Matrix"

$G^{[l](G)}_{kk'} = \ldots$ ← generated

$$J_{style}(S,G) = \| G^{[l](S)} - G^{[l](G)} \|_F^2 = \frac{1}{2 n_H^{[l]} n_w^{[l]} n_c^{[l]}} \sum_k \sum_{k'} (G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'})$$

Course 5

Sequence Model

Sequence Model

Sequence Model.

X: "Harry Potter" and "Hermione Granger" invented a spell

$x^{<1>}$ $x^{<2>}$ ... $x^{<9>}$

$T_x = 9$

y: 1 1 0 1 1 0 0 0

$y^{<1>}$ $y^{<2>}$ ... $y^{<9>}$

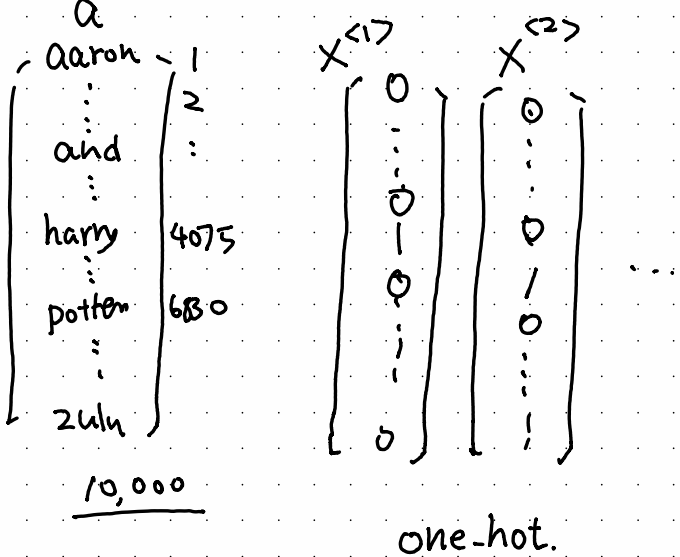
$T_y = 9$

ith training example

$X^{(i)<t>}$ $T_x^{(i)}$

$y^{(i)<t>}$ $T_y^{(i)}$

Representation

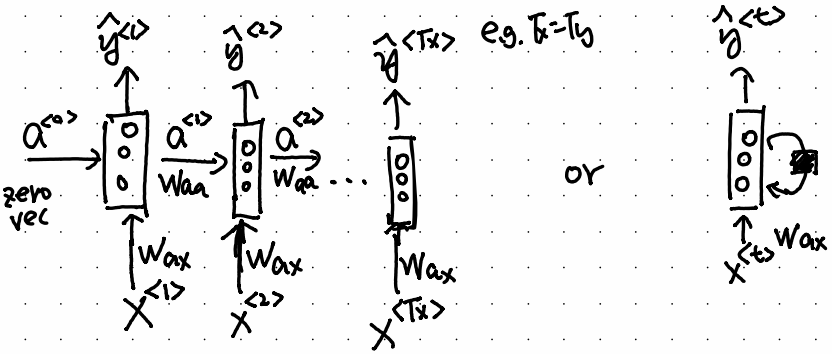


~30,000 for commercial software

Recurrent NN

why not standard network

- input / output could be of different length
- doesn't share features across different pos.



limitation: only use info "before" t △ B-RNN

$a^{<0>} = \vec{0}$

$a^{<1>} = g(W_{aa}a^{<0>} + W_{ax}x^{<1>} + b_a)$ ← tanh, ReLU

$\hat{y}^{<1>} = g(W_{ya}a^{<1>} + b_y)$ ← sigmoid

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y)$$

$$a^{\langle t \rangle} = g(W_{aa} a^{\langle t-1 \rangle} + W_{ax} x^{\langle t \rangle} + b_a)$$

$$\hat{y}^{\langle t \rangle} = g(W_{ya} a^{\langle t \rangle} + b_y)$$

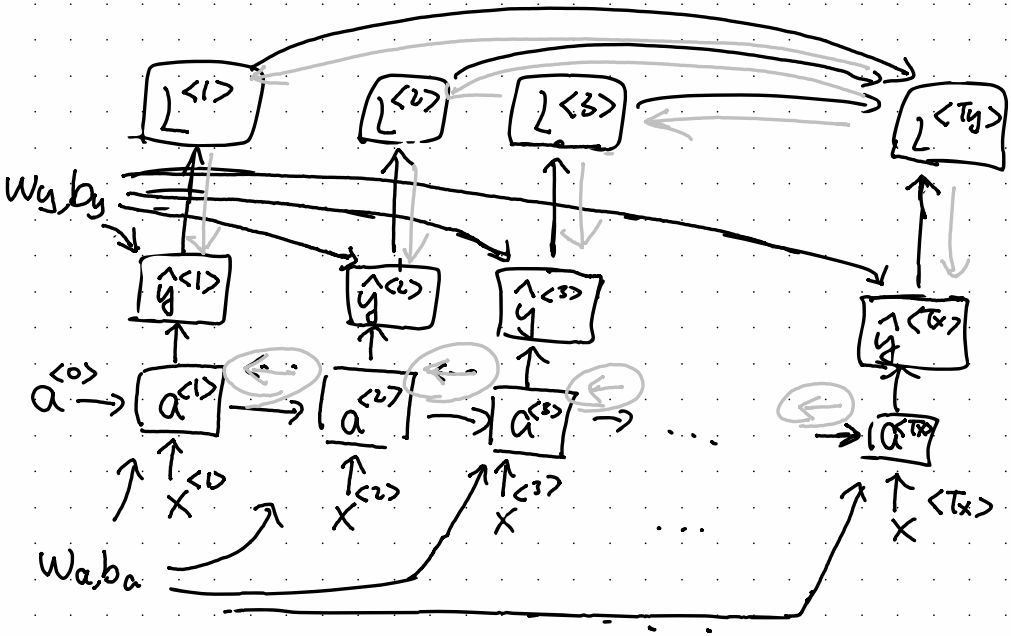
⇓ simplify

$$a^{\langle t \rangle} = g(W_a [a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_a)$$

$$\underset{100}{\left[\underset{100}{W_{aa}} \vdots \underset{10000}{W_{ax}} \right]} = \underset{(100,\,10100)}{W_a}$$

$$\begin{bmatrix} a^{\langle t-1 \rangle} \\ \hline x^{\langle t \rangle} \end{bmatrix} \begin{matrix} \updownarrow 100 \\ \updownarrow 10{,}000 \end{matrix} \quad \updownarrow 10{,}100$$

Backprop. through time

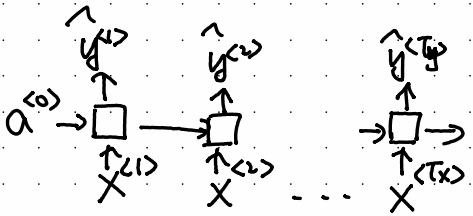


$$L^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -y^{<t>}\log\hat{y}^{<t>} - (1-y^{<t>})\log(1-\hat{y}^{<t>})$$

$$L(\hat{y}, y) = \sum_{t=1}^{T_x} L^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

Different RNN types

$T_x = T_y$

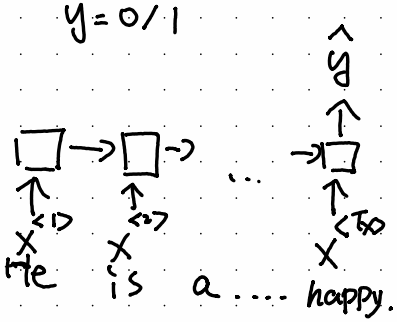


many-to-many

e.g. sentiment detection

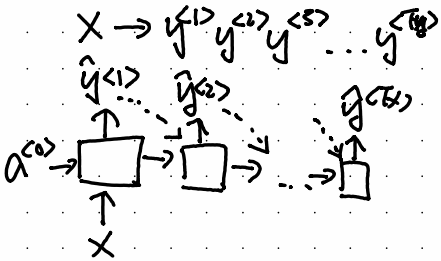
X = text

y = 0/1



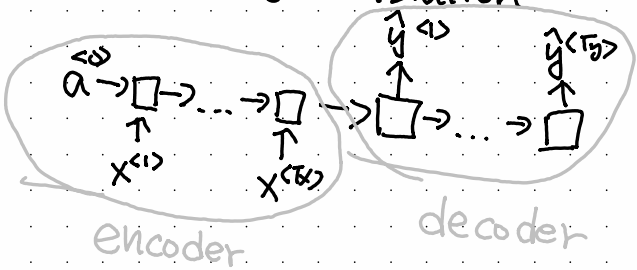
many-to-one

music generation



one-to-many

machine translation



many-to-many

Language Model

Speech Recognition

the apple and pair salad

the apple and pear salad ← higher p, although both sound the same

language model: $P(y^{<1>} \dots y^{<T_y>})$?

Training Set: large corpus of text.

- An apple a day ... doctor away. <EOS>

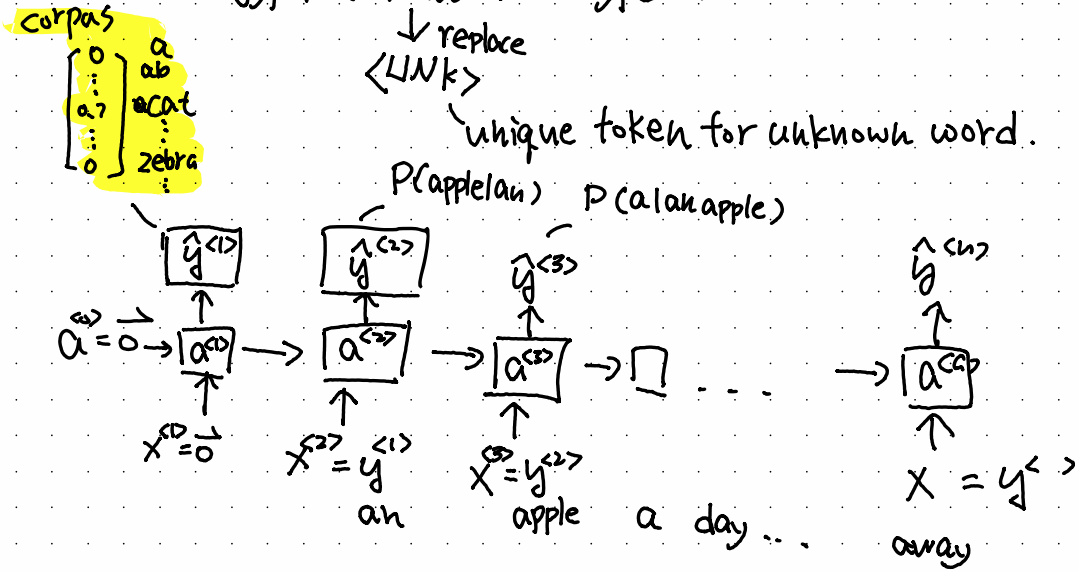
  $y^{<1>}$ $y^{<2>}$ $y^{<3>}$ ... $y^{<n>}$ $y^{<>}$ Tokenize

- The Egyptian Mau is a type of cat

  ↓ replace

  <UNK>

  unique token for unknown word.



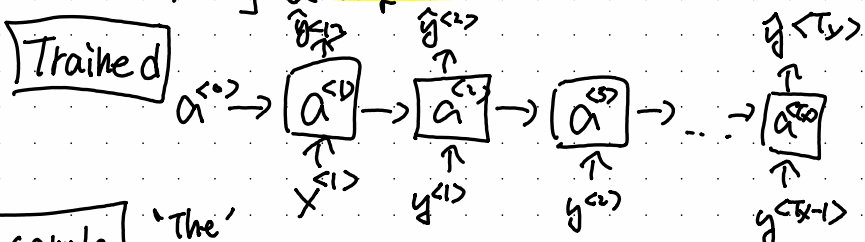
$P(y^{<1>}, y^{<2>}, y^{<3>})$

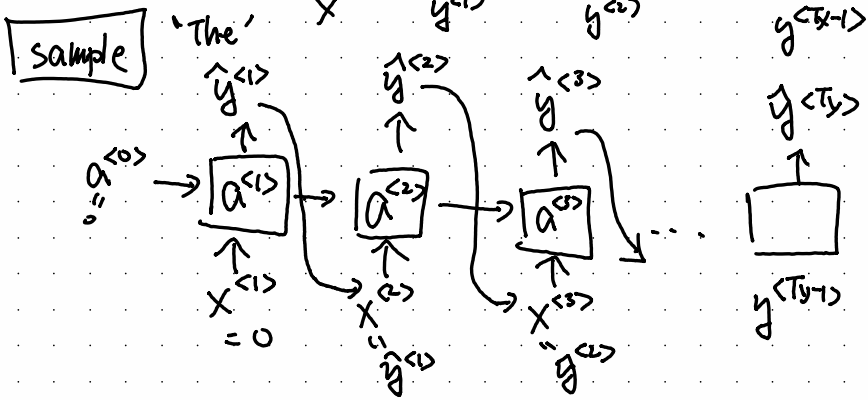
$= P(y^{<1>}) \cdot P(y^{<2>} | y^{<1>}) \cdot P(y^{<3>} | y^{<1>}, y^{<2>})$

$L(\hat{y}^{<t>}, y^{<t>}) = -\sum_i y_i^{<t>} \log \hat{y}_i^{<t>}$

Sampling a seq from trained RNN

- generate random sentence

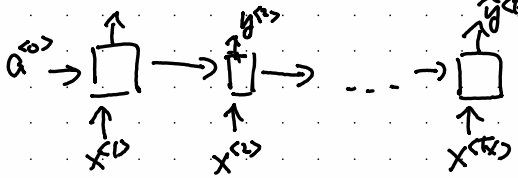




< np.random.choice >

- can also do "character-level" language model
  - very long
  - slower to train
- word level model is more popular.

Vanishing gradients with RNN



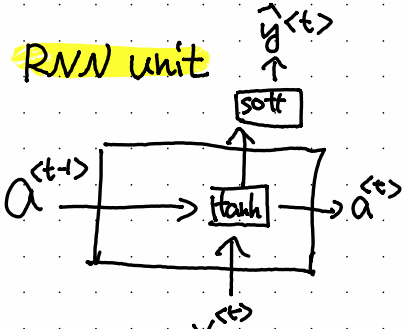
e.g. cats, which ..... , are cute
is cute

- RNN cannot memorize long-term influence

exploding gradient - solve with clipping

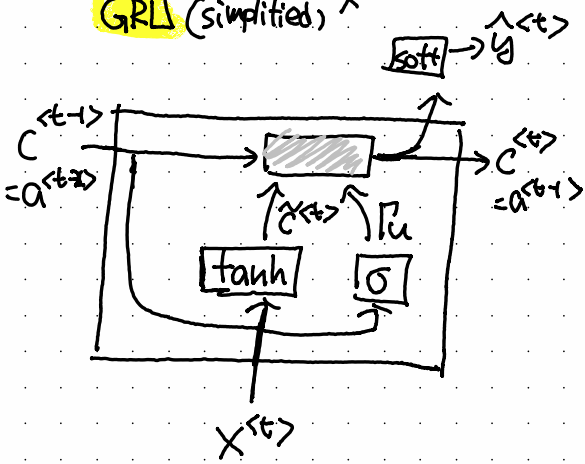
GRU - Gated Recurrent Unit 2014

RNN unit



$$a^{<t>} = g(W_a[a^{<t-1>}, x^{<t>}] + b_a)$$ (tanh)

GRU (simplified)



c = memory cell

$$c^{<t>} = a^{<t>}$$

$$\tilde{c}^{<t>} = \tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$$

Γ_u: update (gamma)

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

the cat, which ate ... is full

$c^{<t>}$ = 1 . . . = 1

Γ_u = 1 0, 0 . . . . . , 1

GRU (Full) — simpler, and can build bigger model

$\tilde{h}$ $\tilde{c}^{\langle t \rangle} = \tanh\left(W_c[\Gamma_r * c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_c\right)$

u $\Gamma_u = \sigma\left(W_u[c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u\right)$

r $\Gamma_r = \sigma\left(W_r[c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_r\right)$ — relevance

h $c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + (1 - \Gamma_u) * c^{\langle t-1 \rangle}$

$c^{\langle t \rangle} = a^{\langle t \rangle}$

LSTM, 1997 — more complex, but more powerful.

$\tilde{c}^{\langle t \rangle} = \tanh\left(W_c[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_c\right)$

$\Gamma_u = \sigma\left(W_u[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u\right)$ — update

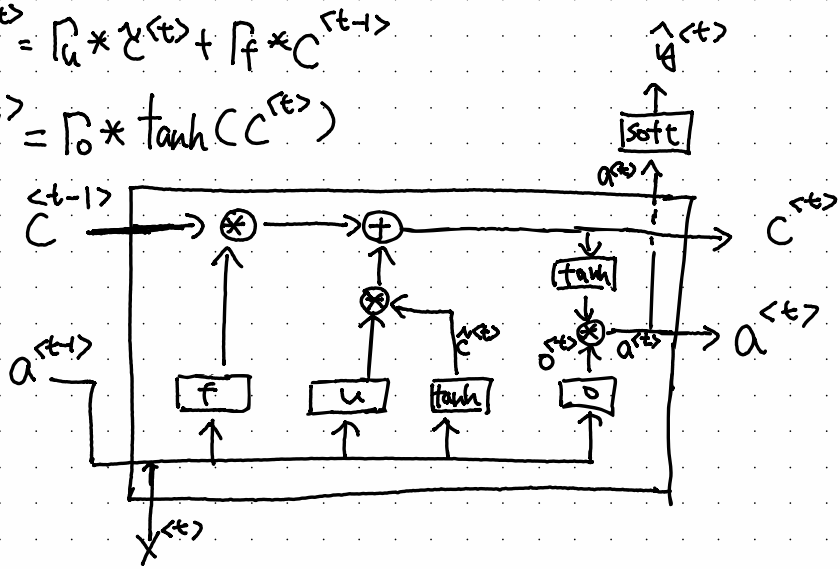
$\Gamma_f = \sigma\left(W_f[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_f\right)$ — forget

$\Gamma_o = \sigma\left(W_o[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_o\right)$ — output

- peephole connection: $c^{\langle t-1 \rangle}$ affects $\Gamma_u, \Gamma_f, \Gamma_o$

$c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + \Gamma_f * c^{\langle t-1 \rangle}$

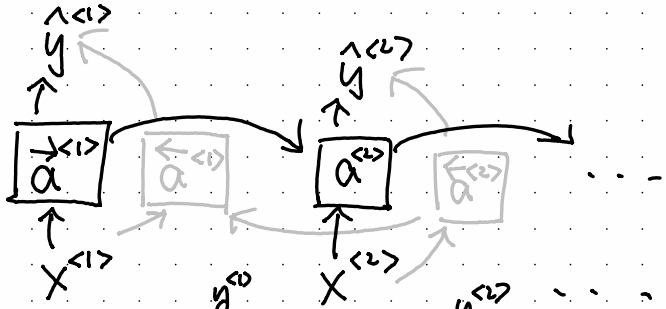
$a^{\langle t \rangle} = \Gamma_o * \tanh(c^{\langle t \rangle})$



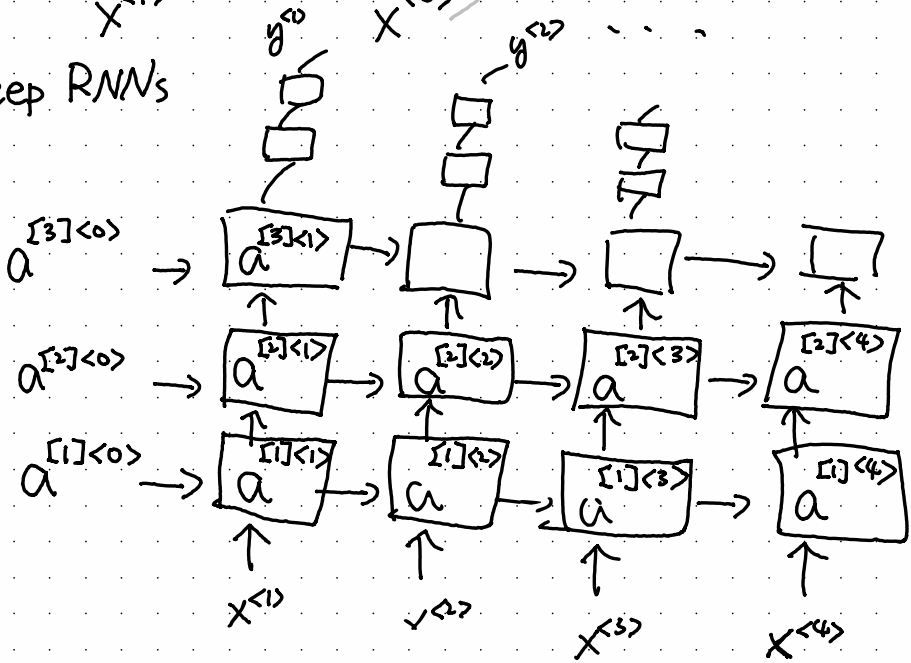
Bidirectional RNN (BRNN)

getting info from future

He said Teddy bears are cute
He said Teddy Roosevelt was a good man



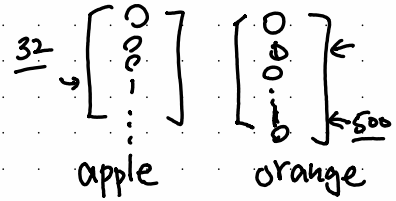
Deep RNNs



$$a^{[2]\langle 3\rangle} = g\left(W_a^{[2]}\left(a^{[2]\langle 2\rangle}, a^{[1]\langle 3\rangle}\right) + b_a^{[2]}\right)$$

Word Embeddings

one-hot representation



← dot prdct is 0, doesn't reflect semantic distance

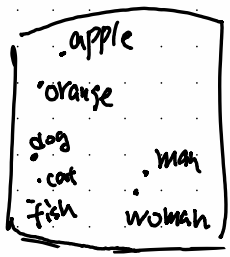
featurized representation: word embedding

	man	woman	apple	orange	...
gender	-1	1	0	0	
age	0.02	0.03	0.02	0.03	
size	0.4	0.4	0.01	0.02	
food	0.01	0.02	0.95	0.94	
⋮	⋮	⋮	⋮	⋮	

e.g. 300D

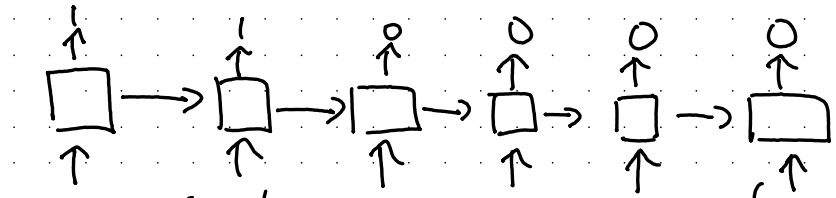
Visualize word embedding t-SNE

embed 300D to 2D for viz



"embed" words into feature space

ex: Name entity recognition



name not coporate name

what if we have fruit not trained before?

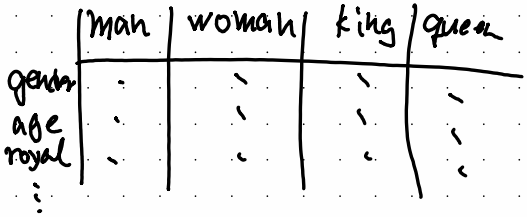
Transfer Learning

1. learn from large text corpus
   (or download pre-trained embedding online)
2. transfer embedding to new tasks w. smaller set
3. (optional) continuously finetune word embedding

properties of word embedding, 2013 Mikolov

e.g. man → woman

king → ?



$e_{man} - e_{woman} \approx e_{king} - e_?$

find word w

$$\arg\max_w \underline{\underline{sim}}(e_w, e_{king} - e_{man} + e_{woman})$$

30~75% accuracy in papers.

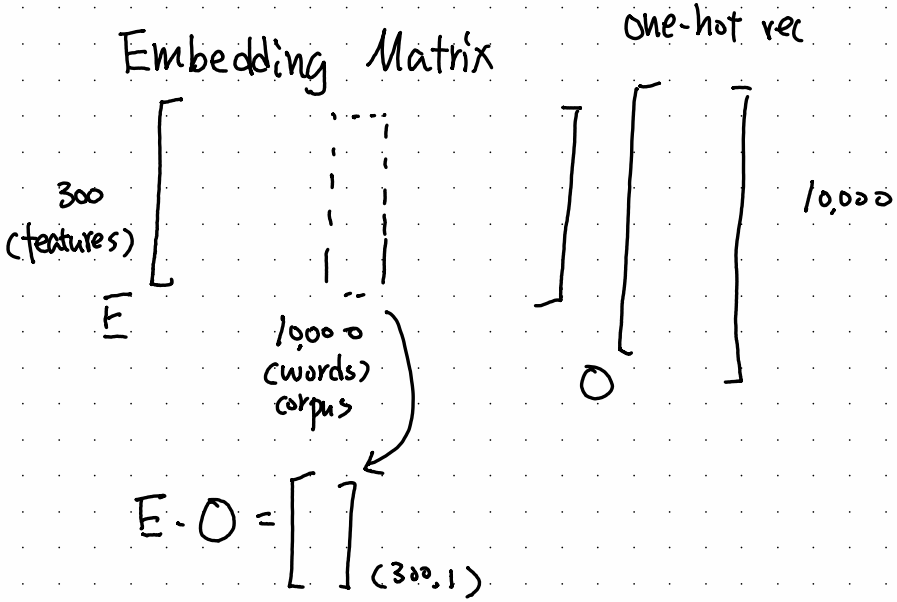
Cosine similarity

$$sim(u,v) = \frac{u^T v}{\|u\|_2 \|v\|_2}$$

man : woman = boy : girl

RMB : China = Rupee : India

⋮



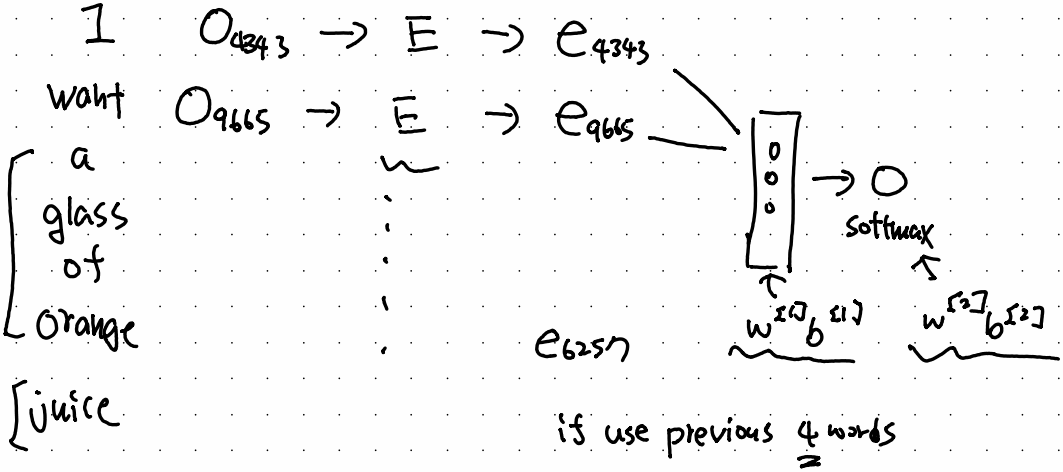
$E \cdot O_j = e_j$ - embedding for word j

(in practise, API extract the column)

Learning Word Embeddings

history: difficult → simple

Neural Language Model, 2003, Bengio



if use previous 4 words

$4 \times 300 = 1200$ dim

learn E, $w^{[1]}$, $b^{[1]}$, $w^{[2]}$, $b^{[2]}$

I want a glass of orange juice to go along with my cereal

↑ target

Context:

last 4 words

last / next 4 words

last 1 word

"nearby" 1 word (skip gram)

Word2Vec, 2013, Mikolov

skip-gram

I want a glass of orange juice to go alone with my cereal

context	target
orange	juice
orange	glass
orange	my

randomly chose

e.g. vocab size = 10000.

Model

$x \rightarrow y$

Context c ("orange") $\rightarrow$ Target t ("juice")

$O_c \rightarrow E \rightarrow e_c \rightarrow \underset{\text{softmax}}{o} \rightarrow \hat{y}$

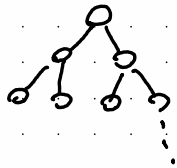
θ_t: parameter associates with output t
i.e. target t being the label

$$\text{Softmax: } p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10000} e^{\theta_j^T e_c}}$$

slow. so,

$$\mathcal{L}(\hat{y}, y) = -\sum_j y_i \log \hat{y}_i \qquad y = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

hierarchical softmax



how to sample cotext c?

too common → the, a, of, ...

→ orange, apple...

$P(c)$ based on frequency

Negative Sampling

2013, Mikolov

problem: are context/word together.

give x, predict y

context (c)	word (t)	target (y)	
orange	juice	1	← context in data
orange	king	0	← random chosen from dict
orange	book	0	k
orange	of	0	

k=5~20 small data

k=2-5 larger

Model

softmax:

$$P(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_j e^{\theta_j^T e_c}}$$ ← 10,000 softmax, slow!

$$P(y=1|c,t) = \sigma(\theta_c^T e_c)$$

orange: O_{6257}

$O_{6257} \rightarrow E \rightarrow e_{6257}$ → 10,000 binary classification

each iteration, train k+1 classifier

juice

king

sampling:

freq $P(w_i)$

uniform $\frac{1}{|V|}$

$$\Rightarrow P(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum f(w_i)^{\frac{3}{4}}}$$ heuristic

GloVe Word, 2014. Pennington.
(global vector)

C, t

X_{ij} = # times j appear in context of i $\quad X_{ij} = X_{ji}$

$$\min \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(X_{ij}) \left(\theta_i^T e_j + b_i + b_j' - \log X_{ij} \right)^2$$

(t) (c)

weighting

$f(X_{ij}) = 0$ if $X_{ij} = 0$ $\quad 0 \log 0 = 0$

this, is..
durian
zulu

θ_i, e_j are symmetric

$$e_w^{(final)} = \frac{e_w + \theta_w}{2}$$

note:

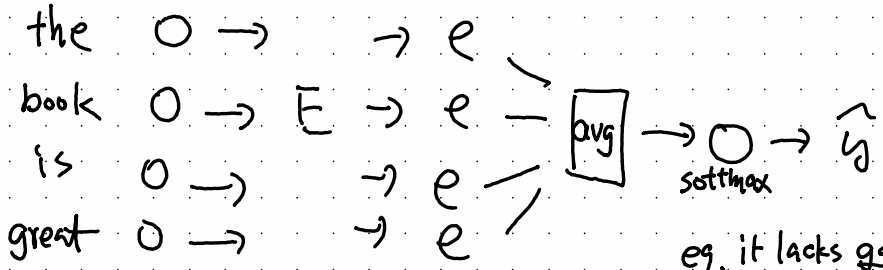
~~e_i~~ might not align with defined interpretable feature axis of the embedding matrix

sentiment classification

x	y
the book is great	★★★★
the book lacks depth	★
reads ok	★★★
⋮	

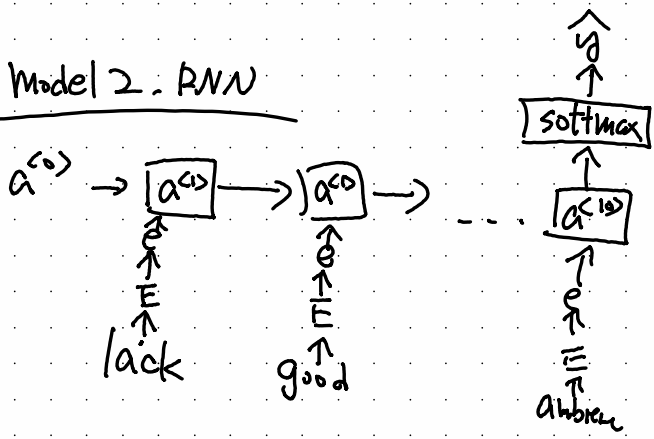
data set is smaller.

model 1.



eg. it lacks good insight, good depth, good taste ... problem!

Model 2. RNN



many-to-one

Debiasing Word Embeddings, 2016.

Man : ComputerProgrammer as Woman : Homemaker ✗

word embedding reflect gender, ethnicity used to train the model.

1. identify
   identify bias axis
2. neutralize
   project onto axis except definitive terms
3. equalize
   move definitive pairs to same distance around axis

Seq to Seq model 2014

e.g. translation

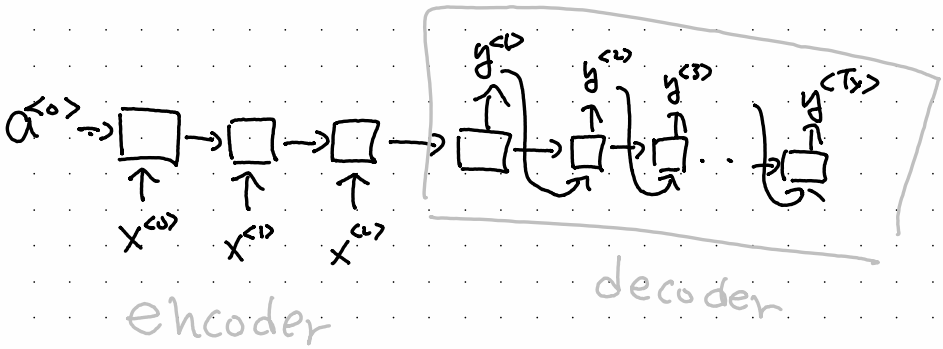
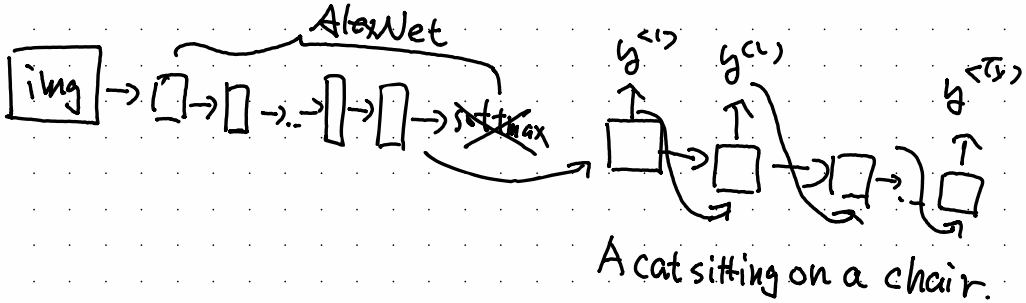


Image Captioning, 2014, 2015

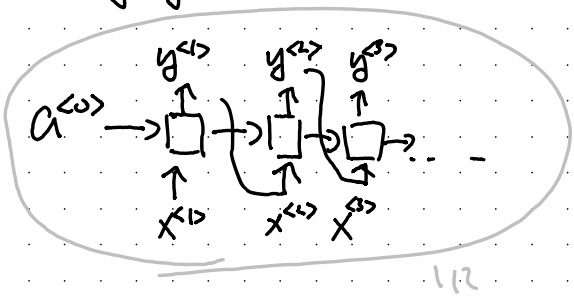


seq2seq vs language model

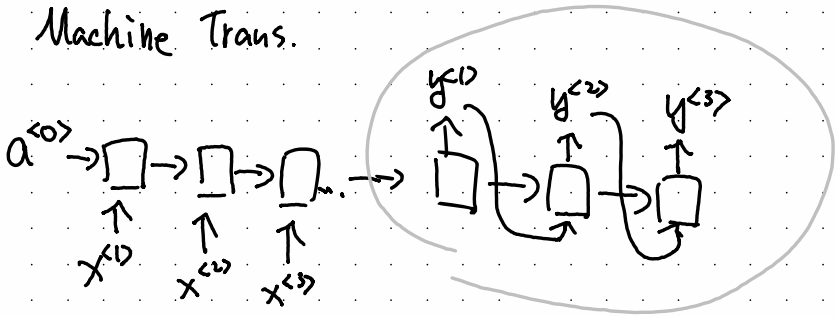
machine trans as building conditional language model.

Language Model:

$P(y^{<1>}, \ldots y^{<T_y>})$



Machine Trans.



conditional language model $P(y^{<1>} \ldots y^{<T_y>} | x^{<1>} \ldots x^{<T_x>})$

e.g. $P(\text{Eng} \mid \text{Franch})$

$\underset{y^{<1>} \ldots y^{<T_y>}}{\text{argmax}} P(y^{<1>} \ldots y^{<T_y>} | x)$

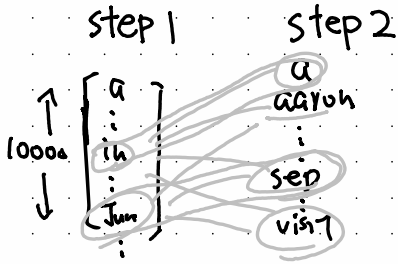
x: French sentence

(beam search

△ greedy search usually result in worse sentence

e.g. picking common word like "going" "the"... etc

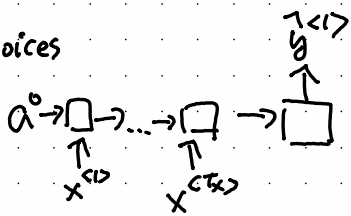
Beam Search



B=3 beam width ⇒ keep track of top B choices

step 1 $P(y^{<1>}|x)$

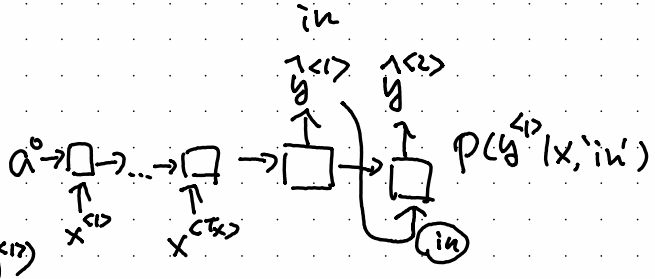
memorize 3 good choices

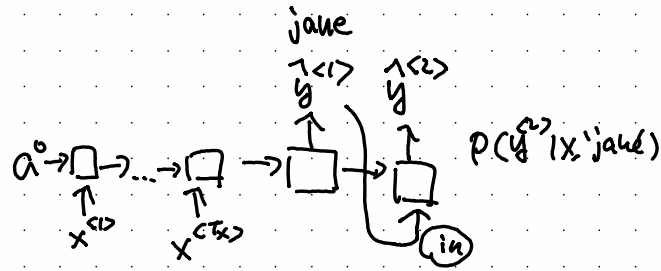


step 2

$P(y^{<1>}, y^{<2>}|x)$

$= P(y^{<1>}|x) P(y^{<2>}|x, y^{<1>})$

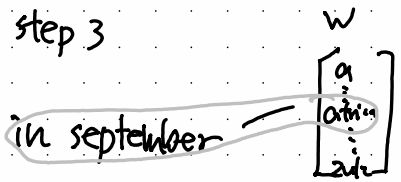




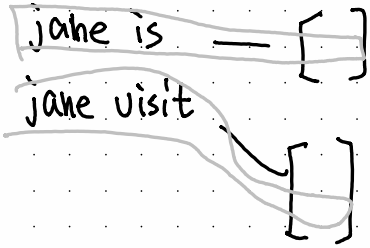
beam width

eventually pick B=3 most likely output then moves on to next step.

step 3



keep B NNs.



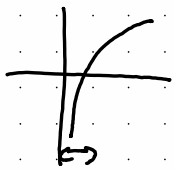
B=1 ~ beam search
"greedy" search.

Refinement: Length normalization

$$\arg\max_y \prod_{t=1}^{T_y} P(y^{<t>} | X, y^{<1>} \dots y^{<t-1>})$$

- multiplication of p minimize value
  → tend to pick short sentence

$$\Rightarrow \arg\max_y \sum_{t=1}^{T_y} \log P(y^{<t>} | X, y^{<1>} \dots y^{<t-1>})$$



$$\Rightarrow \frac{1}{T_y^{\alpha}} \sum_{t=1}^{T_y} \log P(y^{<t>} | X, y^{<1>} \dots y^{<t-1>})$$

$\alpha = 0.7$

---

large B → slow, better
small B → fast, faster.

production B ~ 10
research : B ~ 100 ~ 1000
to squeeze performance

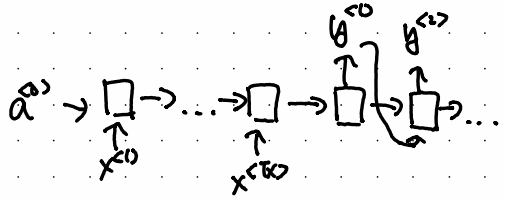
△ compared to BFS, DFS, beam search doesn't gurantee best solution
- heuristic search

Error analysis in Beam Search.

Jane visite l'Afrique en septembre

Human: Jane visit Africa in September (y^*)

Machine: Jane visit Africa last September ($\hat{y}$)



use RNN to compute

$P(y^*|x)$, $P(\hat{y}|x)$

Case 1.

$P(y^*|x) > P(\hat{y}|x)$

beam search can improve

Case 2.

$P(y^*|x) \leq P(\hat{y}|x)$

RNN model is at fault.

→ y^* is better, but RNN says $\hat{y}$ is better, thus RNN is at fault.

↑

do the above analysis for all dev set,

then decide what to do next. - increase B

or

update RNN

Bleu score : bilingal evaluation understudy, 2002
- deal with equally good translations

Bleu: Unigram
sentence: Le chat est sur le tapis

ref 1 : the cat is on the mat
ref 2 : There is a cat on the mat.
⋮

clipping # occurrance

MT output: the the the ...

precision: $\frac{7}{7}$ modified: $\frac{2}{7}$ ← Count_{clip}('the') / Count('the')

Bleu score on bigrams

MT: The cat the cat on the mat.

	Count	Count_{clip}
the cat	2	1
cat the	1	0
the cat	1	1
cat on	1	1
on the	1	1
the mat		

$$P_1 = \frac{\sum_{e \in \hat{y}} Count_{clip}(unigram)}{\sum_{e \in \hat{y}} Count(unigram)}$$

n-gram

$$P_n = \frac{\sum_{n\text{-}gram \in \hat{y}} Count_{clip}(n\text{-}gram)}{\sum_{n\text{-}gram \in \hat{y}} Count(n\text{-}gram)}$$

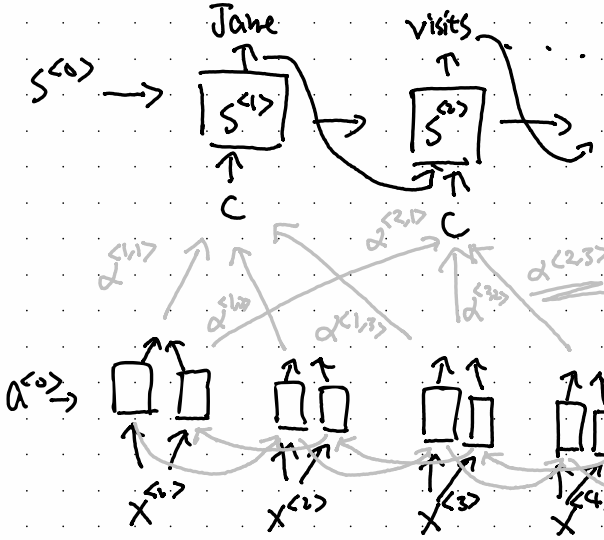
Combined: $P = \exp\left(\frac{1}{n}\sum_{i=1}^{n} P_i\right)$

BP. brief penality → penalize short trans

$$Bp = \begin{cases} 1 & \text{if MT output longer than human} \\ e^{\left(1 - \frac{human\_len}{MT\_len}\right)} & \end{cases}$$

Attention Model, 2014

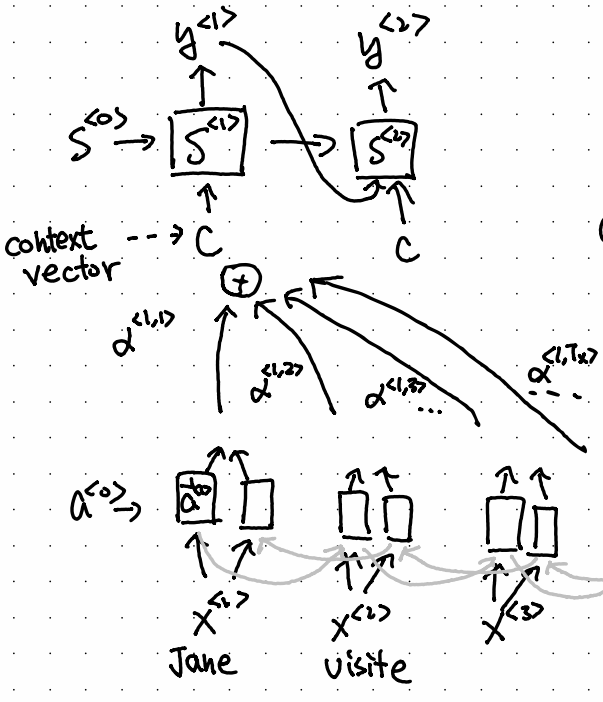
- cannot handle long sequences



$\alpha^{<t,t'>}$

attention weight.

BRNN



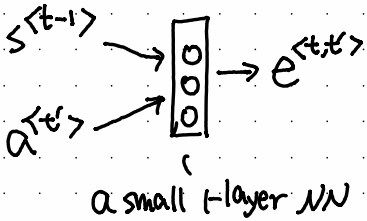
$$a^{<t'>} = (\vec{a}^{<t'>}, \overleftarrow{a}^{<t'>})$$

$$\sum_{t'} \alpha^{<1,t'>} = 1$$

$$c^{<1>} = \sum_{t'} \alpha^{<1,t'>} a^{<t'>}$$

$\alpha^{<t,t'>}$ is amount of attention $y^{<t>}$ pay to $a^{<t'>}$

$$\alpha^{<t,t'>} = \frac{e^{e^{<t,t'>}}}{\sum_{t'=1}^{Tx} e^{e^{<t,t'>}}}$$ <-- similar to softmax



runtime

$Tx \cdot Ty$ parameters $\alpha^{<t,t'>}$

usually Tx, Ty are not huge.

2015, image captioning, -> pay attention to a part of the picture

speech recognition

x (audio) → y (transcript)

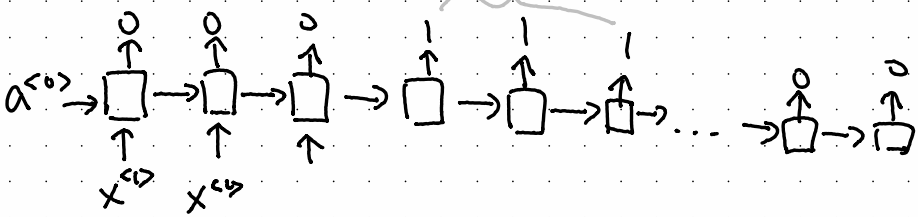
data: 300h ~ 3000h

commercial: 100,000 h

- attention model works well
- CTC cost [2006]
  connectionist temporal classification ttt_h_eee__ _ qqq__
  △ Tx >> Ty, e.g. 1000 Hz in audio → the quick brown fox ...
  → collapse repeated chars not separated by "blank"

trigger word detection

multiple 1 for balanced data set.

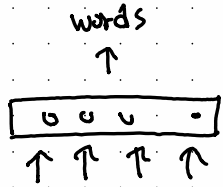


Transformer. 2017

RNN → GRU → LSTM

con:
- increase complexity
- sequential

transformer:
- attention + CNN
  - self-attention. $A^{<1>} A^{<2>} \dots$
  - multi-head attention. multiple copies



---

self-attention

transformer attention

$A(q, K, V)$ – attention-based vector representation

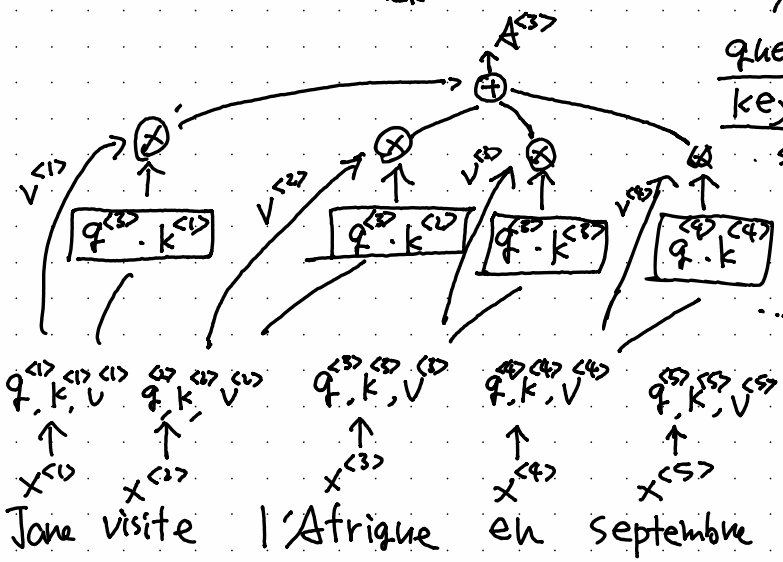
$$= \sum_i \frac{e^{q \cdot k^{<i>}}}{\sum_j e^{q \cdot k^{<j>}}} v^{<i>}$$

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$q^{<3>} = W^Q x^{<3>}$$

$$k^{<3>} = W^K x^{<3>}$$

$$v^{<3>} = W^V x^{<3>}$$



query – ask question

key – used to calculat similarity between word and query

value – for plugging into the representation.

↓↓ more flexible than fixed word embedding.

Multi-head Attention

for example

#heads = 8 (usually)

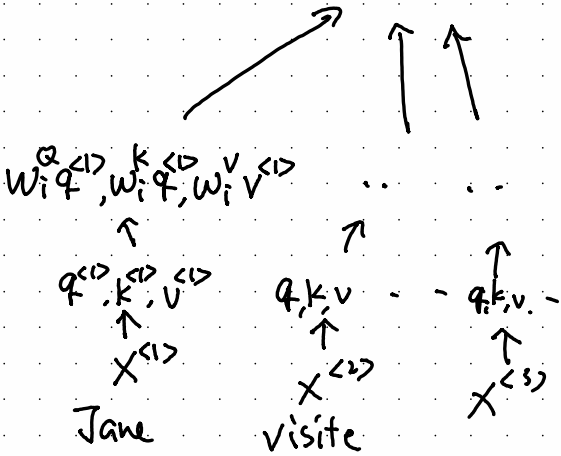
head1 W_1^Q, W_1^K, W_1^V – what's happening

head2 W_2^Q, W_2^K, W_2^V – when?

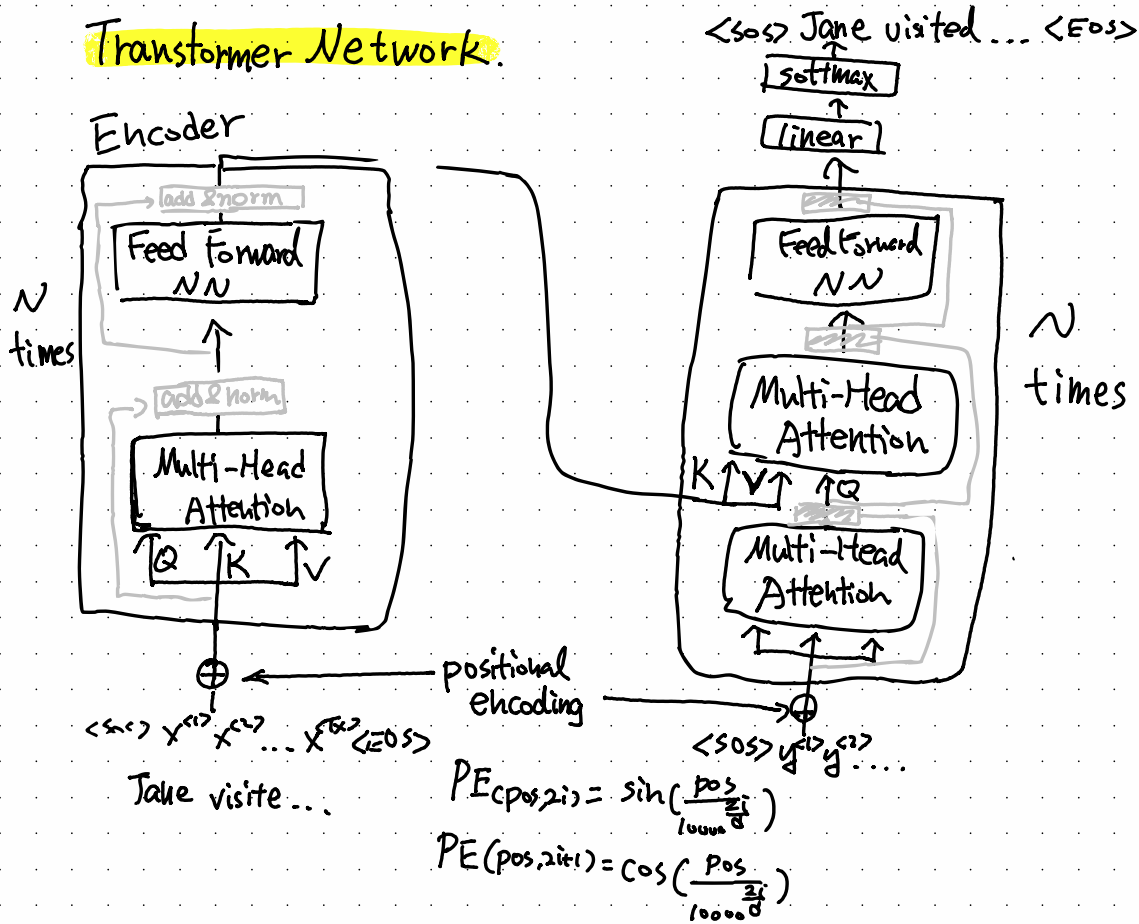
⋮ ⋮ – how?

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1, \text{head}_2, \text{head}_3)W_0$$

$$\text{Attention}(W_i^Q Q, W_i^K K, W_i^V V)$$



Transformer Network



$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

masked multi-head attention

- for training.