deeplearning.ai

1. Neural Networks and Deep Learning
2. Improving deep Neural Networks: Hyperparameter tuning, Regularization, optimization
3. Structuring your machine learning Project
4. Convolutional Neural Networks
5. Natural Language Processing: Building sequence models

C1. Neural Networks & Deep Learning

W1: Introduction to DL

1.3 Supervised learning.

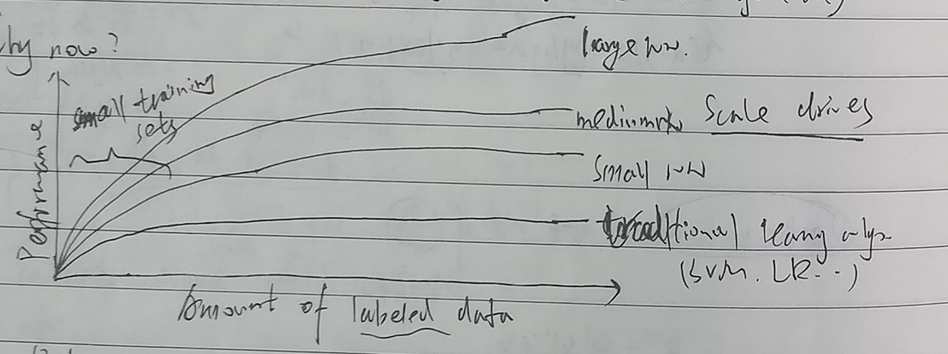
real estate, Online advertising, Photo, Speech, translation, route-driving

standard NN. CNN RNN Custom/hybrid NN

structured data: list

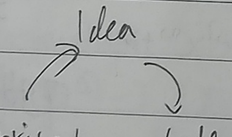
unstructured data: Audio, Image, Text (human good at)

1.4 Why now?



Data, Computation, Algorithms.

sigmoid → relu (faster gradient descent)



W2: Basics of Neural Network Programming

(x, y). $x \in R^{n_x}$, $y \in \{0,1\}$

m training examples.

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

$X \in R^{n_x \times m}$ X.shape = (n_x, m)

$Y = [y^{(1)}, y^{(2)} \cdots y^{(m)}]$ $Y \in R^{1 \times m}$ Y.shape = $(1, m)$

logistic regression

$\hat{y} = \sigma(w^T x + b)$, $w \in R^{n_x}$, $b \in R$

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

$x_0 = 1$, $x \in R^{n_x+1}$

$\hat{y} = \sigma(\theta^T x)$

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{n_x} \end{bmatrix}$ b, w

Skeleton of deeplearning.ai

C1. Neural Networks and Deep learning.

W1: Introduction to DL
supervised learning, why now?

W2: Basic of Neural network Programming.
LR. loss function. vectorization.

W3: NN overview. activation, gradient descent for NN.

W4: DNN. Forward and Backward Propagation. parameter & hyperparam

C2: Improving DNN.

W1: setting up your ML application.
Data, bias & variance, regularization, dropout, Augmentation, early stopping, normalizing training set, gradient vanish/explode, weight initialize. Grad check

W2, optimization Algos. mini-batch, moving average, momentum, RMSprop, Adam, learning rate decay, local optimal

W3: Hyperparameter tuning. tuning process, BN, Softmax, DL frameworks.

C3: Structuring ML projects

W1: intro to ML strategy. orthogonalization, evaluation metric, optimizing and satisfying metric, human performance, avoidable bias, improving model performance.

W2. Error Analysis. carry out error analysis, incorrectly labeled data, different distribution, Bias & variance, addressing data mismatch, transfer learning, Multi-task learning, end-to-end DL

C4. Convolutional Neural networks

W1: CNN. edge detect, padding, stride, CNN, Pooling, why convolutional?

W2. case studies. classic networks (LeNet, AlexNet, VGG16, ResNet, NIN, Inception), transfer learning, Data augmentation, state of

W3. object detection. landmark detection, object detection, sliding window, Bounding box, IOU, Non-max suppress, Anchor boxes, Yolo, region proposal

W4: special applications of Convnet. face recognition, one-shot learning, Siamese network, triplet loss, neural style transfer, cost func (content, style)

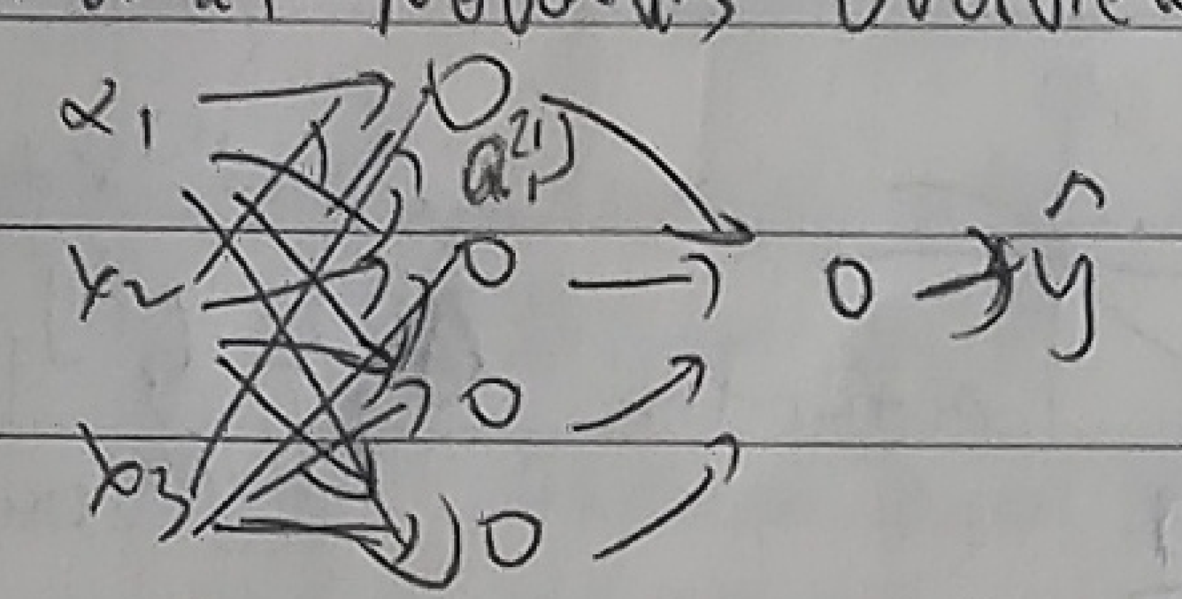
C5. Sequence model

W1: Recurrent Neural network. RNN, Backpropogation, types of RNN, Language model, sequence generation, vanishing grads, GRU, LSTM, bidirectional network, deep RNN

W2. NLP and word Embeddings. word representation/embeddings, embedding matrix, Word2Vec, negative sampling, GloVe word vectors, Sentiment classification, Debias

W3. Sequence to sequence models. Basic model, Pick the most likely sentence, Beam search, Attention model, Speech recognition, Trigger word detection.

W3: Neural Networks Overview



$a^{[0]} = X$ $a^{[1]}$ $a^{[2]}$

layer: input hidden output

$W^{[1]}$

$$z^{[1]} = \begin{bmatrix} -w_1^{[1]T}- \\ -w_2^{[1]T}- \\ -w_3^{[1]T}- \\ -w_4^{[1]T}- \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T}x+b_1^{[1]} \\ w_2^{[1]T}x+b_2^{[1]} \\ \vdots \end{bmatrix}$$

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

$z^{[1]} = W^{[1]}X + b^{[1]}$

$A^{[1]} = \sigma(Z^{[1]})$

$Z^{[2]} = W^{[2]}X + b^{[2]}$

$A^{[2]} = \sigma(Z^{[2]})$

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

activation function

- sigmoid function: $\frac{1}{1+e^{-z}}$ (0, 1) (output layer)
- tanh(z): $= \frac{e^z - e^{-z}}{e^z + e^{-z}}$; rescale. (better) (hidden layer)
- ReLU: $= \max(0, z)$ leaky ReLU $= \max(0.01z, z)$

Gradient descent for NN

cost func: $J(w^{[1]}, w^{[2]}, b^{[1]}, b^{[2]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}, y)$

Gradient descent:

repeat: { compute predictions ($\hat{y}^{(i)}$, i=1…m)

$dw^{[1]} = \frac{\partial J}{\partial w^{[1]}}$, $dw^{[2]}$, $db^{[1]}$, $db^{[2]}$

$w^{[1]} = w^{[1]} - \alpha dw^{[1]}$; $w^{[2]}$, $b^{[1]}$, $b^{[2]}$

}

Formulas for computing derivatives

Forward Propagations: (eg. 两层网络)

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

$A^{[1]} = g^{[1]}(Z^{[1]})$,

$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$

$A^{[2]} = g^{[2]}(Z^{[2]})$,

Back propagations:

$dZ^{[2]} = A^{[2]} - Y$ ($Y = [y^{(1)}, y^{(2)}, \ldots y^{(m)}]$)

$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$

$db^{[2]} = \frac{1}{m}$np.sum($dZ^{[2]}$, axis=1, keepdims=True)

$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$ — $(n^{[1]}, m)$ — $*$: element product

$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$

$db^{[1]} = \frac{1}{m}$np.sum($dZ^{[1]}$, axis=1, keepdims=True) — $(n^{[1]}, 1)$ — keepdims: reshape

if $y=1$: $P(y|x) = \hat{y}$

if $y=0$: $P(y|x) = 1-\hat{y}$ } $P(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$; 取log, 加负号为loss.

; cost on m samples

; $\log P(\text{labels on training set}) = \log \prod_{i=1}^{m} P(y^{(i)}|x^{(i)})$

; $\log P(\ldots) = \sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)}) = -\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$

loss function

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

$L(\hat{y}, y) = -(y\log\hat{y} + (1-y)\log(1-\hat{y}))$; $y=1$: $L = -\log\hat{y}$ ← $\hat{y}\uparrow$; $y=0$: $L = -\log(1-\hat{y})$ ← $\hat{y}\downarrow$

cost function

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

Gradient descent

want w, b minimize J(w, b)

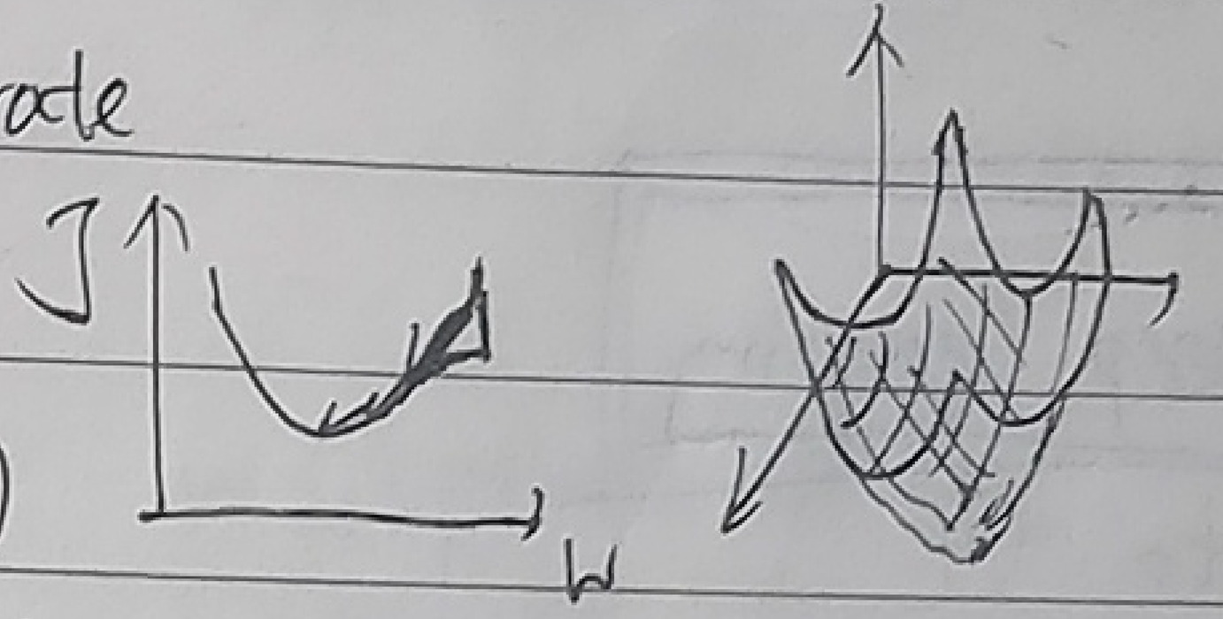
Repeat {

$w := w - \alpha \frac{dJ(w,b)}{dw}$ (→ learning rate; dw slope)

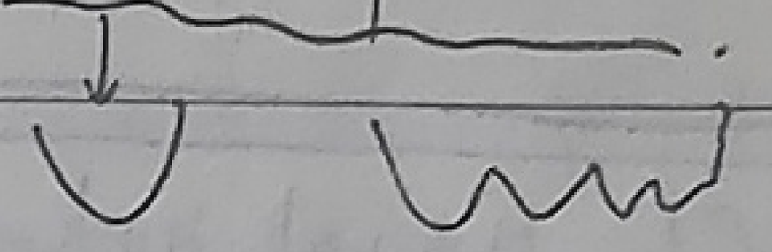
($w := w - \alpha dw$)

($b := b - \alpha db$)

}



convex function



∂: partial derivation (→ one variable) ; d: J only one variable

Derivatives of fun(a): slope $\frac{\text{height}}{\text{width}}$

Computing derivatives: chain rule, eg. $J = 3(a + bc)$ ($u = bc$, $v = a + u$) $\frac{dJ}{db} = \frac{dJ}{dv}\cdot\frac{dv}{du}\cdot\frac{du}{db}$

(forward: cost-func ; backward: derivatives)

eg. $z = w^T x + b$

$\hat{y} = a = \sigma(z) = \frac{1}{1+e^{-z}}$

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

x_1, w_1, x_2, w_2, b → $z = w_1x_1 + w_2x_2 + b$ → $a = \sigma(z)$ → $L(a, y)$

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

$dz = \frac{dL}{dz} = \frac{dL}{da}\frac{da}{dz} = da \times [a(1-a)] = a - y$

$\left(\frac{-e^{-z}}{(1+e^{-z})^2}\right)' = a(1-a)$

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

$dw_2 = x_2 dz$, $db = dz$

{ $w_1 = w_1 - \alpha dw_1$; $w_2 = w_2 - \alpha dw_2$; $b = b - \alpha db$ }

Vectorization: get rid of for, np.dot(a, b), np.exp…

(cpu/gpu): SIMD: single instruction multiple data

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

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

⋮

$Z = [z^{(1)} z^{(2)} \ldots z^{(m)}] = w^T X + [b \ldots b] = [w^T x^{(1)} + b \ldots w^T x^{(m)} + b]$ (1×m)

python: Z = np.dot(w.T, X) + b

python: broadcasting. eg. $\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + 100 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 100 \\ 100 \\ 100 \\ 100 \end{bmatrix}$; matrix $(m, n) \pm (1, n) \Rightarrow (m, n)$; $(m, 1) \Rightarrow (m, n)$

advice: ① a = np.random.randn(5, 1) not a = np.random.randn(5)

[[ ], [ ], … [ ]] matrix vector ; [, , , ,] shape: (5,) rank 1 array ; $a^T = a$ shape (5,)

$a^T \neq a$ (shape) ; np.dot(a^T, a) = 5×5 matrix ; np.dot(a^T, a) = number

② assert(a.shape == (5, 1))

a.reshape((5, 1))

Parameters: $W^{[l]}$, $b^{[l]}$

Hyperparameters: learning rate α
#iterations
#hidden layers, L
#hidden units: $n^{[1]}, n^{[2]}, \dots$
choice of activation function

} determine parameters

Idea → code → Experiment

Empirical process

later: momentum, batch size, regulations.

3.10 Computing gradients.

(logistic regression)

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

$x_1, x_2, x_3 \to \sigma \to \hat{y} = a$

x, w, b → $z = w^T x + b$ → $a = \sigma(z)$ → $L(a, y)$

$dz = a - y$

$dw = dz \cdot x$

$db = dz$

$da = \frac{d}{da} L(a,y) = -y\log a - (1-y)\log(1-a)$

$= -\frac{y}{a} + \frac{1-y}{1-a}$

$dz = da \cdot g'(z)$

$g(z) = \sigma(z)$

$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z}$

dz, da, $\frac{d}{dz}(g(z)) = g'(z)$

$dz = da \cdot \sigma'(z) = \left(-\frac{y}{a} + \frac{1-y}{1-a}\right) \cdot \sigma(z)(1-\sigma(z)), \ (a = \sigma(z))$

$= a - y$

(neural network gradients)

$x_1, x_2, x_3 \to \circ \to \circ \to \hat{y}$

x, $W^{[1]}$, $b^{[1]}$ → $z^{[1]} = W^{[1]} x + b^{[1]}$ → $a^{[1]} = \sigma(z^{[1]})$ → ($W^{[2]}$, $b^{[2]}$) $z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$ → $a^{[2]} = \sigma(z^{[2]})$ → $L(a^{[2]}, y)$

$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$

$(n^{[1]}, 1) = (n^{[1]}, n^{[2]}) \cdot (n^{[2]}, 1) * (n^{[1]}, 1)$

$dz^{[1]} = dz^{[2]} \frac{dz^{[2]}}{da^{[1]}} \cdot \frac{da^{[1]}}{dz^{[1]}}$

$= dz^{[2]} \cdot W^{[2]T} g^{[1]\prime}(z^{[1]})$

$dW^{[1]} = dz^{[1]} \cdot x^T$ $(a^{[0]})^T$

$db^{[1]} = dz^{[1]}$

$dz^{[2]} = a^{[2]} - y = \frac{\partial L}{\partial a^{[2]}} \cdot \frac{da^{[2]}}{dz^{[2]}} = \left[-\frac{y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}}\right] \cdot a^{[2]}(1-a^{[2]})$

$dW^{[2]} = dz^{[2]} a^{[1]T}$ → "$dw = dz \cdot x$" $a^{[1]}$

$db^{[2]} = dz^{[2]}$

W: [—] 行

5

(random initialize)

$W^{[1]} = np.random.randn((m,n)) * 0.01$

$b^{[1]} = np.zeros((m,1))$

⋮

if $W^{[l]} = 0$ all. all hidden layer symmetric, all iteration

$a^{[l]}_1 = a^{[l]}_2 \dots$

b →

for sigmoid, tanh

activation

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

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

slope: small

W4 Deep L-layer neural network

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

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

↕ loop

(check dims)

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

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

(shape)

$A^{[l-1]} = n^{[l-1]}$

$Z^{[l]} = n^{[l]}$

(vectorized)

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

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

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

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

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

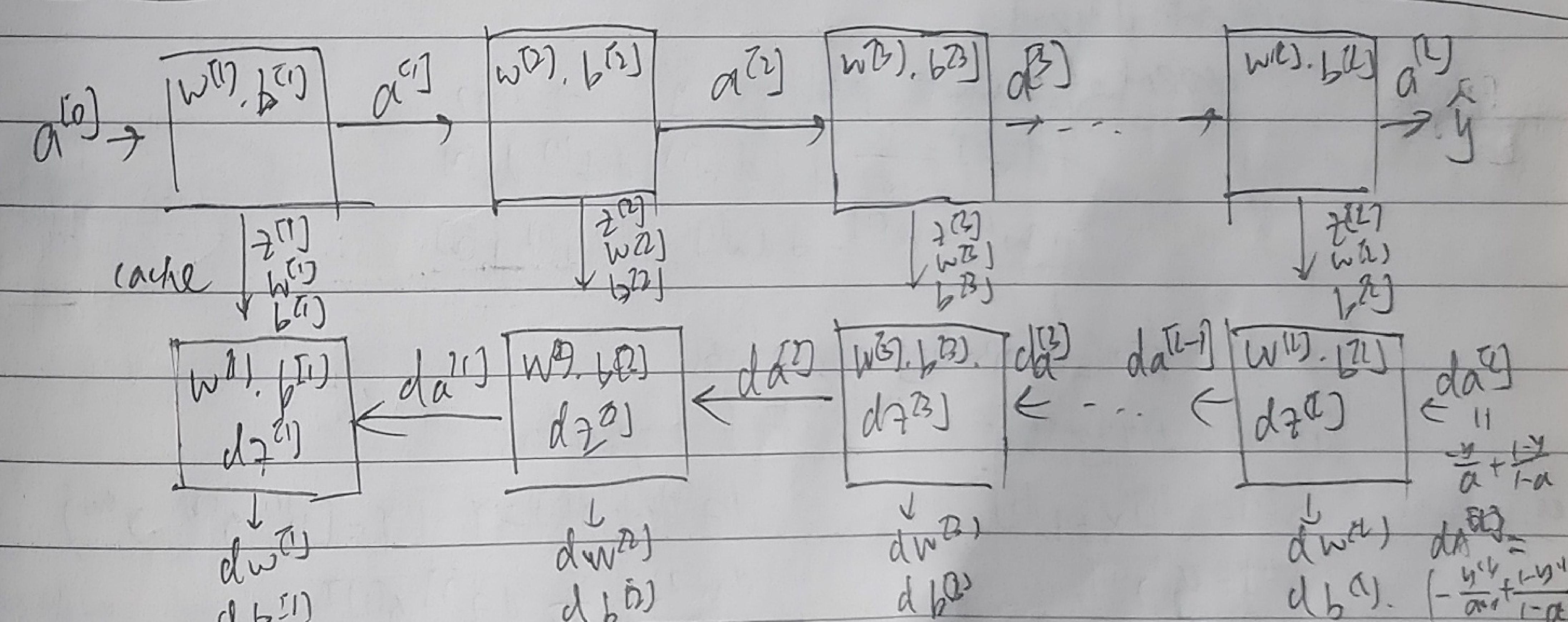
[z z]

(DL & human brain)

Circuit theory

$y = x_1 \text{ XOR } x_2 \text{ XOR } x_3 \text{ XOR} \dots x_n$

O(2^n)



$da^{[l]} = -\frac{y}{a} + \frac{1-y}{1-a}$

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

$W^{[l]} = W^{[l]} - \alpha dW^{[l]}$;

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

Forward:

Input: $a^{[l-1]}$

Output: $a^{[l]}$, cache $z^{[l]}$ ($W^{[l]}, b^{[l]}$)

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

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

$a^{[0]}$: single

$A^{[0]}$: whole

vectorized:

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

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

$X = A^{0} \to \square \to \square \to \square \to$

Backward:

Input: $da^{[l]}$

Output: $da^{[l-1]}$, $dW^{[l]}$, $db^{[l]}$

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

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

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

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

$dz^{[l]} = W^{[l+1]T} dz^{[l+1]} * g^{[l]\prime}(z^{[l]})$

vectorized.

$dZ^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]})$

$dW^{[l]} = \frac{1}{m} dZ^{[l]} A^{[l-1]T}$

$db^{[l]} = \frac{1}{m} np.sum(dZ^{[l]}, axis=1, keepdims=True)$

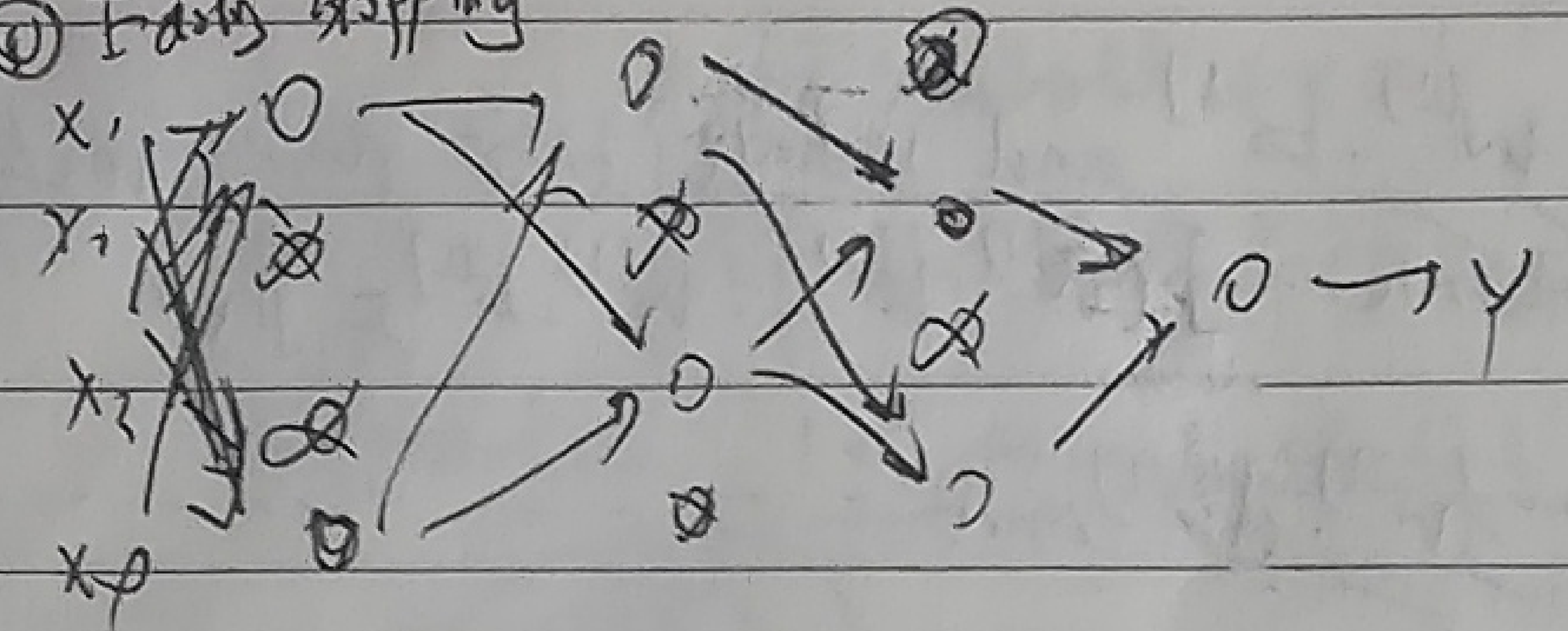
$dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$

6

Regularization { ① L2, ② Dropout, ③ Data Augmentation, ④ Early stopping

speed up train { ① Normalizing training set, ② weight initialization

Dropout ②



eg. Inverted dropout:

$l=3$: keep_prob = 0.8

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

a3 = np.multiply(a3, d3)

a3 = a3 / keep_prob

No dropout at predict set. (no random)

Intuition: can't rely on any one feature, so have to spread out weights. shrink weights. (L2)

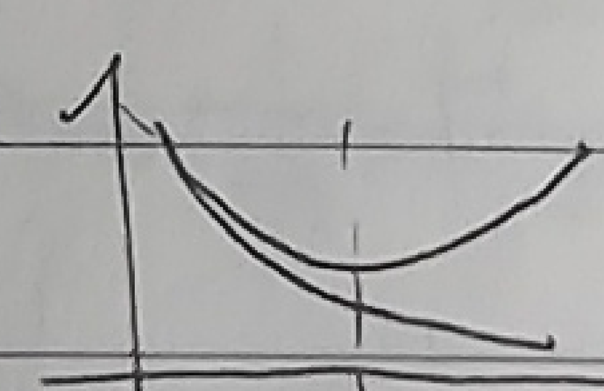
main area: computer vision.

downside: J is no longer well-defined. (trick: off dropout, J plot ↘, than dropout)

Data Augmentation ③

flip, rotate, distort (CV)

Early stopping ④



dev error, J / train error, mid w, larger, J(w, b)

orthogonalization:
- optimize J: Gradient ...
- not overfit: Regularization ... → variance ↓

can't separate

Early stopping: find middle w once. L2: search proper λ.

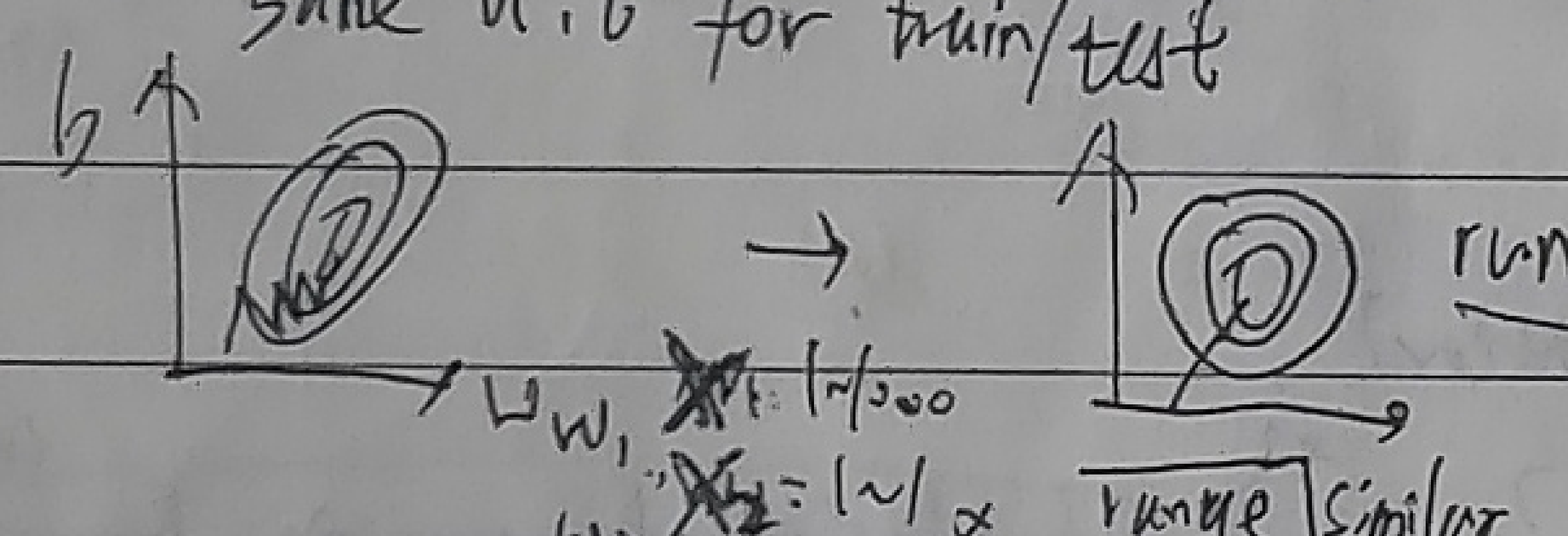
Normalizing training sets

(Setting up your optimization problem) 1

1) substract mean: $\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$, $x = x - \mu$

2) Normalize variance: $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} ** 2$, $x = x / \sigma^2$

Same μ, σ^2 for train/test



w_1: 1 / 1000, w_2: 1 / 1 → range similar, run faster

Vanishing / Exploding gradients

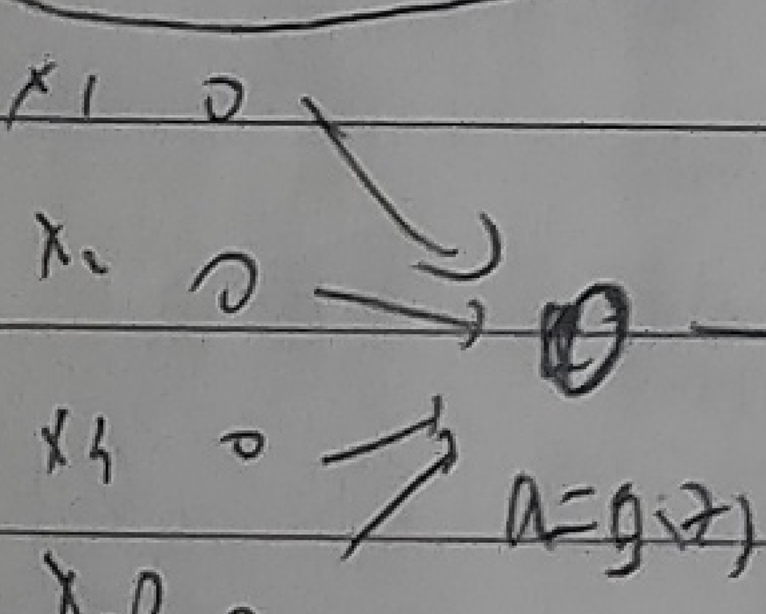
eg. NN: $g(z) = z$, $b^{[l]} = 0$

$\Rightarrow \hat{y} = W^{[L]} W^{[L-1]} \cdots W^{[2]} W^{[1]} x$

$W^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$ / 0.5, $\hat{y} = W^{[L]} \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L-1} x$, $1.5^{L-1} x$, $0.5^{L-1} x$

Weight initialization

2



$a = g(z)$

$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$

larger n → smaller w_i

$Var(w_i) = \frac{1}{n}$ (sigmoid), $\frac{2}{n}$ relu

$W^{[l]}$ = np.random.randn(shape...) * np.sqrt($\frac{2}{n^{[l-1]}}$) ReLU $\sqrt{\frac{2}{n^{[l-1]}}}$

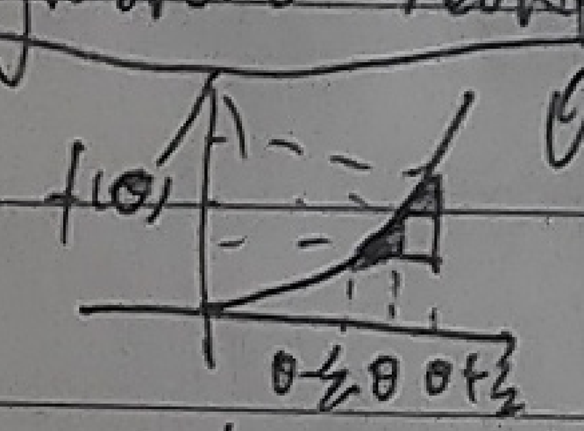
another parameter (not important)

X normalize, W normalize, Z normalize. reduce gradients vanish / explod

other variants: tanh: $\sqrt{\frac{1}{n^{[l-1]}}}$ (Xavier initializer)

$\sqrt{\frac{2}{n^{[l]} + n^{[l-1]}}}$ (Yoshua Bengio)

Gradient checking



θ^3, $\theta = 1$; $\varepsilon = 0.01$

$\frac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon} \approx g(\theta)$

$\frac{1.01^3 - 0.99^3}{0.02} = 3.0001 \approx 3 = 3\theta^2$

$\frac{f(\theta+\varepsilon) - f(\theta)}{\varepsilon} = 3.0301$

$f'(\theta) = \lim_{\varepsilon \to 0} \frac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon}$, error: $O(\varepsilon^2)$ | $f'(\theta) = \lim_{\varepsilon \to 0} \frac{f(\theta+\varepsilon) - f(\theta)}{\varepsilon}$ error: $O(\varepsilon)$

7

C2. Improving Deep Neural Networks.

W1: Setting up your ML applications.

Setting up your ML application

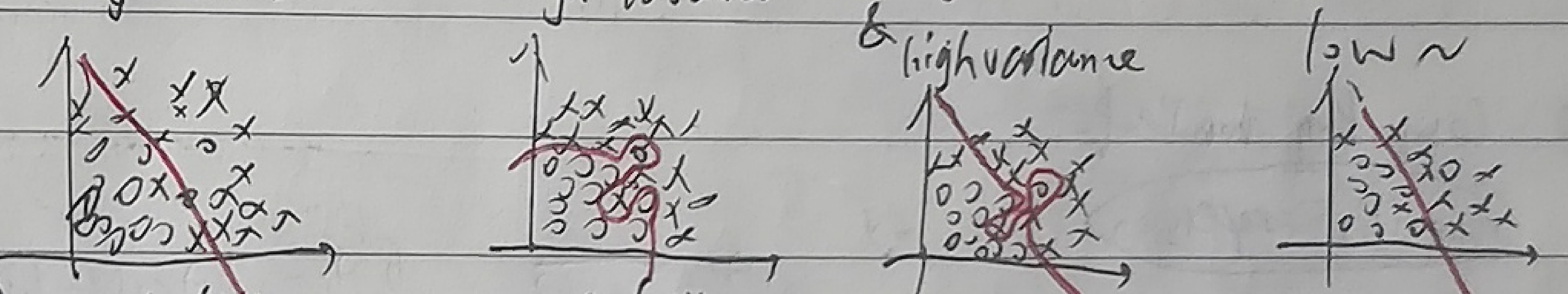
Data

training set | hold out CV development set "dev" | test — same distribution

70/30, 60/20/20, Big data: 98/1/1 (大数据)

Bias / Variance

① optimal error (Bayes) ≈ 0%
② Train & dev same distribution



	high bias	high variance	high bias & high variance	low ~, low ~
	underfitting	overfitting		
train error	15%	1%	15%	0.5%
dev error	16%	11%	30%	1%

Basic Recipe for ML

High bias? —Y→ Bigger Network, Train longer, (NN architecture search)

↓ N

High variance? —Y→ More Data, Regularization, (NN architecture search)

↓ N

Done!

DL could ↓bias ↓variance same time.

Regularization

Regularizing your neural network

logistic regression:

$\min_{w,b} J(w, b)$, $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

$J(w, b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_2^2 \left(+ \frac{\lambda}{2m} b^2\right)$ (low dim. less parameter)

λ: regularization parameter (dev / cv)

① L2 regularization: $\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$

L_1: $\frac{\lambda}{m}\sum_{j=1}^{n_x} |w_j| = \frac{\lambda}{m}\|w\|_1$ w will be sparse (more 0)

Neural Network:

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

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

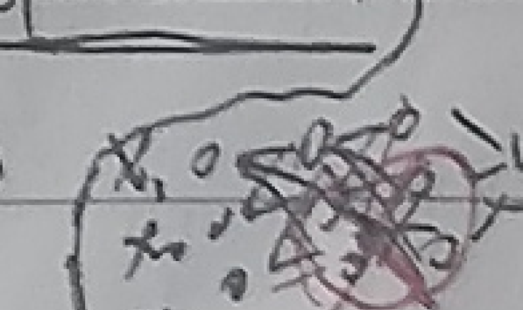
$dw^{[l]} = (\text{from backprop}) + \frac{\lambda}{m} w^{[l]} = \frac{dJ}{dw^{[l]}}$

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

L2: "weight decay" $w^{[l]} = w^{[l]} - \alpha\left((\text{from backprop}) + \frac{\lambda}{m} w^{[l]}\right)$

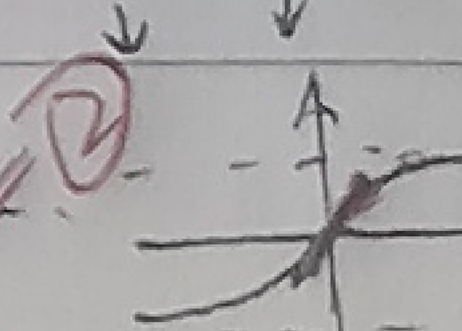
$= (1 - \frac{\alpha\lambda}{m}) w^{[l]} - \alpha(\text{from backprop})$ (<1)

why work? complex net → simple net



(1) λ ↑, $w^{[l]}$ → 0 → Network simple

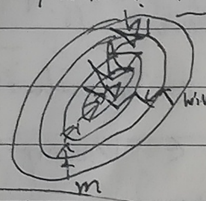
(2) simple tanh: $z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$ ↓ → linear



6

Choosing mini-batch size { if mini-batch size = 1 : Stochastic gradient descent — lose speed-up from vectorization — $X^{\{t\}}, Y^{\{t\}} = X^{(t)}, Y^{(t)}$
if mini-batch size = m : Batch gradient descent — Too long per iteration — $X^{\{t\}}, Y^{\{t\}} = X, Y$

in practice: 1 < mini-batch size < m



fastest learning:
- vectorization
- make progress without processing the entire set

m = 2000 : Batch gradient descent
m = 1 : mini-batch ...

Exponentially weighted averages (moving average)

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

V_t: an average over $\approx \frac{1}{1-\beta}$ days (approximately)

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

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

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

$\Rightarrow v_{100} = 0.1\theta_{100} + 0.1\times0.9\theta_{99} + 0.1\times0.9^2\theta_{98} + 0.1\times0.9^3\theta_{97} + \cdots$

$(1-\varepsilon)^{1/\varepsilon} \approx 1/e$. $0.9^{10} \approx \frac{1}{e}$. $0.98^{50} \approx 1/e$

efficient both: computation, memory

Statistics

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

$V_1 = \theta_1$

$V_{100} = \alpha\theta_{99} + \alpha(1-\alpha)\theta_{98} + \alpha(1-\alpha)^2\theta_{97} + \cdots$

Bias correction.

$V_0 = 0$ Repeat{ get next θ_t; $V_\theta := \beta V_\theta + (1-\beta)\theta_t$ }

$V_0 = 0$ ($\neq 0$)

$V_1 = 0.98 V_0 + 0.02\theta_1$

$V_2 = 0.98 V_1 + 0.02\theta_2$

$= 0.0196\theta_1 + 0.02\theta_2$

$\frac{V_t}{1-\beta^t}$ $t=2$: $1-\beta^t = 0.0396$

$\frac{V_2}{0.0396} = \frac{0.0196\theta_1 + 0.02\theta_2}{0.0396}$

(early stage). ($t\uparrow$, $1-\beta^t \to 1$).

Gradient descent with momentum

$V_{dW} = 0$, $V_{db} = 0$ (initialize)

On iteration t:

compute dW, db on mini-batch.

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

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

$W = W - \alpha V_{dW}$, $b = b - \alpha V_{db}$

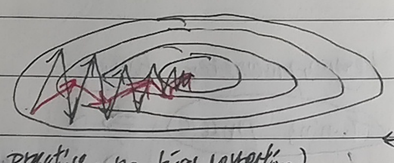
$V_{dW} = \beta V_{dW} + dW$

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

$\beta' = \beta \cdot \frac{1}{1-\beta}$

$\alpha' = \alpha \cdot (1-\beta)$

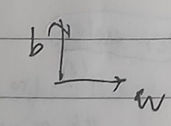
Hyperparameters: α, β ($\beta = 0.9$)



(in practice, no bias correction)

average on last 1/β gradient

↓ slow learning ↔ fast learning



RMSprop: root mean square prop

On iteration t:

compute dW, db on current mini-batch.

$S_{dW} = \beta_2 S_{dW} + (1-\beta_2)(dW)^2$ ← element-wise ← small

$S_{db} = \beta_2 S_{db} + (1-\beta_2)(db)^2$ ← large

$W := W - \alpha \frac{dW}{\sqrt{S_{dW}} + \varepsilon}$; $b := b - \alpha \frac{db}{\sqrt{S_{db}} + \varepsilon}$

(faster) (slower)

ε 10^-8. stability

could large α

Grad check

Take $W^{[1]}, b^{[1]}, \ldots W^{[L]}, b^{[L]}$ and reshape into a big vector θ.

(concatenate) $J(W^{[1]}, b^{[1]} \ldots W^{[L]}, b^{[L]}) = J(\theta)$

Take $dW^{[1]}, db^{[1]}, \ldots dW^{[L]}, db^{[L]}$ and reshape into a big vector $d\theta$

(concatenate)

check: Is $d\theta$ is the gradient of $J\theta$?

for each i:

$d\theta_{approx}[i] = \frac{J(\theta_1, \theta_2 \ldots \theta_i + \varepsilon, \ldots \theta_L) - J(\theta_1, \theta_2 \ldots \theta_i - \varepsilon, \ldots \theta_L)}{2\varepsilon}$

$\approx d\theta[i] = \frac{\partial J}{\partial \theta_i}$ $d\theta_{approx} \overset{?}{\approx} d\theta$

check: $\frac{\|d\theta_{approx} - d\theta\|_2}{\|d\theta_{approx}\|_2 + \|d\theta\|_2}$ ($\varepsilon = 10^{-7}$) $\approx$ 10^{-7} great! 10^{-5} 10^{-3} worry

Notes: ① Don't use in training. only to debug.

② If fails, check component.

③ Remember regularization. $J = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum\|W^{[l]}\|_F^2$

④ doesn't work with dropout.

⑤ Run at random initialization; perhaps again after some training. $W, b \neq 0$

W2: Optimization Algorithm. (train faster)

Mini-batch gradient descent

$X^{\{t\}}, Y^{\{t\}}$ batch size: for example, 1000. ($m = 5{,}000{,}000$)

$X = (n^x, m)$ $Y = (1, m)$

repeat{

for i in m/batch size:{

Forward prop on $X^{\{t\}}$:

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

$A^{[1]} = g^{[1]}(Z^{[1]})$

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

(batch size examples) — vectorized implement

compute cost $J^{\{t\}} = \frac{1}{1000}\sum_{i=1}^{l} L(\hat{y}^{(i)}, y^{(i)}) + \frac{1}{2\cdot 1000}\sum\|W^{[L]}\|_F^2$ for $X^{\{t\}}, Y^{\{t\}}$

Back to compute gradients $J^{\{t\}}$ (using $X^{\{t\}}, Y^{\{t\}}$)

$W^{[L]} = W^{[L]} - \alpha dW^{[L]}$, $b^{[L]} = b^{[L]} - \alpha db^{[L]}$

}

"1epoch"

}

Batch Normalization (BN)

Given some intermediate values in NN: $z^{(1)} \dots z^{(m)}$ ($z^{[l](i)}$)

$\mu = \frac{1}{m}\sum_{i=1}^{m} z^{(i)}$; $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(z_i - \mu)^2$

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

$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$ ← learnable parameters (control range)

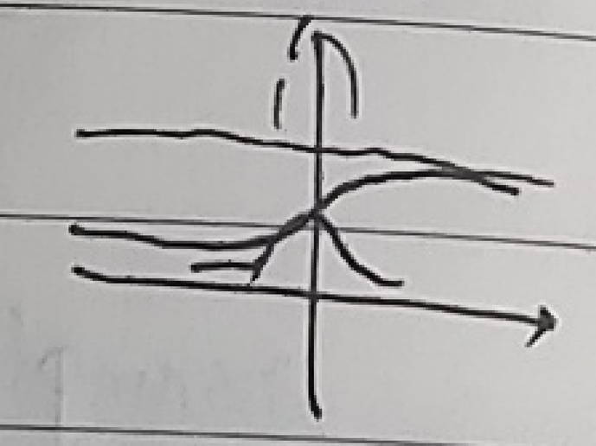
Use $\tilde{z}^{[l](i)}$ instead of $z^{[l](i)}$.

If: $\gamma = \sqrt{\sigma^2 + \varepsilon}$, $\beta = \mu$

then: $\tilde{z}^{[l](i)} = z^{[l](i)}$

X ←

$z^{(i)}$ ← not always



$X \xrightarrow{w^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[BN]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \longrightarrow a^{[1]} = g^{[1]}(\tilde{z}^{[1]}) \xrightarrow{w^{[2]}, b^{[2]}} z^{[2]} \xrightarrow[BN]{\beta^{[2]}, \gamma^{[2]}} \tilde{z}^{[2]} \longrightarrow a^{[2]} = g^{[2]}(\tilde{z}^{[2]}) \dots$

Parameters: $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]} \dots w^{[L]}, b^{[L]}$
$\beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]} \dots \beta^{[L]}, \gamma^{[L]}$ } $d\beta^{[l]}$ $\beta^{[l]} = \beta^{[l]} - \alpha d\beta^{[l]} \dots$

mini-batch

$X^{\{1\}} \xrightarrow{w^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[BN]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow g^{[1]}(\tilde{z}^{[1]}) = a^{[1]} \xrightarrow{w^{[2]}, b^{[2]}} z^{[2]} \rightarrow \dots$

$X^{\{2\}} \xrightarrow{w^{[1]}, b^{[1]}} z^{[1]} \longrightarrow \dots$

$X^{\{3\}} \dots$

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

Parameters: $w^{[l]}$, ~~$b^{[l]}$~~, $\beta^{[l]}$, $\gamma^{[l]}$

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

$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$ (crossed out)
$z^{[l]} = w^{[l]} a^{[l-1]}$
$z^{[l]}_{norm}$
$\tilde{z}^{[l]} = \gamma^{[l]} z^{[l]}_{norm} + \beta^{[l]}$

Why BN work?

learning on shifting input distribution — "covariate shift". black cat → colorful cat

dispart adjacent layers

BN as regularization (slight). (scaled by mean/variance μ, σ^2) just on mini-batch. (batch↑, regularization↓) noise similar to dropout.

BN at test time

no mini-batch.

μ, σ^2 (moving average).

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

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

$X^{\{1\}}, X^{\{2\}}, \dots X^{\{t\}}$
$\mu^{\{1\}[l]}, \mu^{\{2\}[l]} \dots \rightarrow \mu$ same for σ^2

Adam

Adaptive moment estimation

$V_{dw} = 0, V_{db} = 0, S_{dw} = 0, S_{db} = 0$

On iteration t:

compute dw, db using current mini-batch

$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dw$, $V_{db} = \beta_1 V_{db} + (1-\beta_1) db$ ← "momentum" β_1

$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw^2$, $S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$ ← "rmsprop" β_2

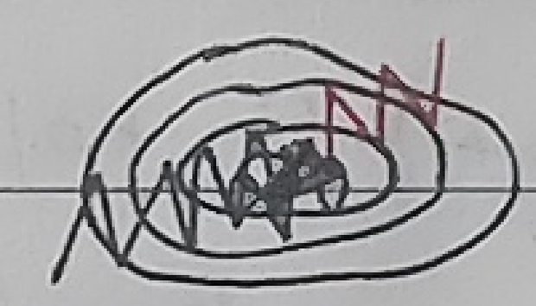
$V^{corrected}_{dw} = V_{dw}/(1-\beta_1^t)$, $V^{corrected}_{db} = V_{db}/(1-\beta_1^t)$

$S^{corrected}_{dw} = S_{dw}/(1-\beta_2^t)$, $S^{corrected}_{db} = S_{db}/(1-\beta_2^t)$

$W := W - \alpha \frac{V^{corrected}_{dw}}{\sqrt{S^{corrected}_{dw}} + \varepsilon}$, $b := b - \alpha \frac{V^{corrected}_{db}}{\sqrt{S^{corrected}_{db}} + \varepsilon}$

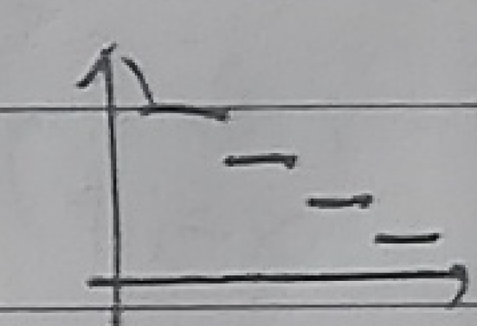
min α: needs to be tune
β_1: 0.9 (dw)
β_2: 0.999 (dw^2)
default ε: 10^{-8}

Learning rate decay



① $\alpha = \frac{1}{1 + decay\_rate * epoch\_num} \alpha_0$; ② $\alpha = 0.95^{epoch\_num} \cdot \alpha_0$, exponentially decay

③ $\alpha = \frac{k}{\sqrt{epoch\_num}} \cdot \alpha_0$, or $\alpha = \frac{k}{\sqrt{t}} \cdot \alpha_0$ ④ α discrete.



⑤ by hand

Local optimal

low-dimention ✗→ high dimention ≈ more saddle, not local optimal

all features are: ∪ 2^{-20000}

1). Unlikely to get stuck in a bad local optima;

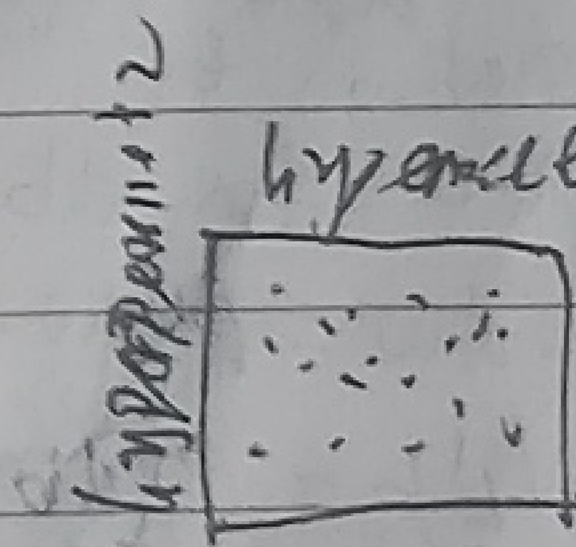
2). Plateaus can make learning slow. (rmsprop, adam)

W3: Hyperparameter tuning

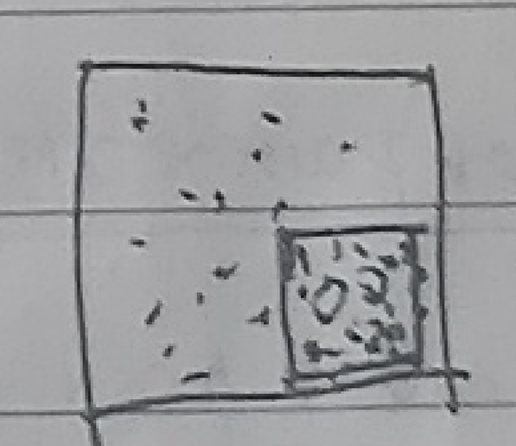
Hyperparameter Tuning

Tuning process

⇒ α
→ β (0.9), #hidden units, mini-batch size
→ #layers, learning rate decay
β_1 (0.9), β_2 (0.999), ε (10^{-8})



random search
not grid!
(not equal importance)
more search



coarse to fine
adequate search

Proper scale

$n^{[l]}$: (50 ~ 100). random search

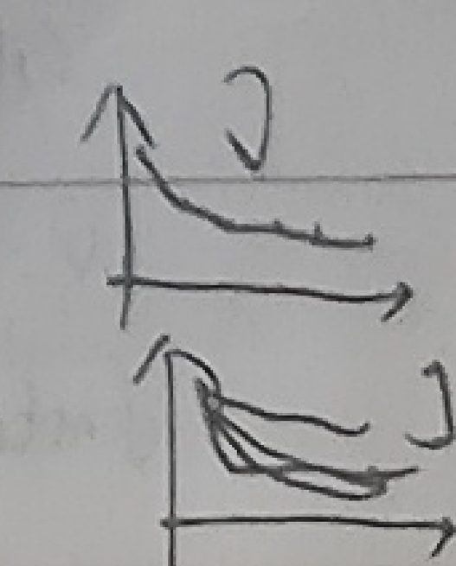
#layers: (2 ~ 4): grid search

α: (0.0001 ~ 1): log scale
(r = np.random.rand() * -4
$\alpha = 10^r$) ← - - →

β = (0.9 ~ 0.999)
$1-\beta$: (0.1 ~ 0.001)
10^{-1} 10^{-3}

r = np.random.rand()
β = 1 - 10**(-1-r)

{ Babysitting one model — Pandas
Training many models in parallel — Caviar } resources



C3. Structuring your ML strategy

W1: Introduction to ML strategy

Why ML strategy?

Intro to ML strategy

Ideas: Collect more data
Collect more diverse training set
Train algo longer with gradient descent
Try adam instead of gradient descent
Try bigger/smaller network
Try dropout
Add L2 regularization
Network structure.
- activation functions
- #hidden units...

Orthogonalization

: each knob kind of does only one thing

Chain of assumptions in ML

Fit training set well on cost function (≈human level performance)
↓
Fit dev set well on cost func.
↓
Fit test set well on cost func.
↓
Performs well in real world

- bigger network
- Adam
- ...
- Regularization
- Bigger train set
- bigger dev set
- change dev set
- change cost func.

early stopping — not easy to tune

Using a single number evaluation metric

Setting up your goal

F1 instead of Precision or Recall. $F_1 = \frac{2}{\frac{1}{P}+\frac{1}{R}}$

sometimes average.

Dev set + single real number evaluation metric: speed up iterating.

Optimizing and satisficing metrics

eg. cat classification example. → optimizing metric; → satisficing metric

classifier	accuracy	running time	
A	90%	80ms	
B	92%	95ms	✓
C	95%	1500ms	

maximize accuracy
subject to running time ≤ 100ms

N metrics { 1 optimizing; N-1 satisficing

Train/dev/test distribution

Guideline: Choose a dev and test set to reflect data you expect to get in the future and consider important to do well on.

same distribution ← randomly shuffle to dev/test

set size: old way: 70%/30% or 60%/20%/20%

Big data era: 98%/1%/1%

guidelines:
1. set test set big enough to give high confidence in the overall performance in your system.
2. Test set using < 30%
3. The dev set has to be big enough to evaluate different ideas.

Softmax

Multi-class classification

multi-class classification

class_num $C = 6$

$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$ $(6,1)$

Activation function:

$t = e^{(z^{[l]})}$ $(6,1)$

$a^{[l]} = \frac{e^{z^{[l]}}}{\sum_{j=1}^{6} t_i}$, $a_i^{[l]} = \frac{t_i}{\sum_{j=1}^{6} t_i}$ $(6,1)$

eg: $z = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$ $t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$ $\sum t_i = 176.3$

$a^{[l]} = \frac{t}{176.3} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$

hard max: $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

examples: decision boundary.

Softmax regression: $C=3$ (x_1, x_2)

$C=5$ (x_1, x_2)

$x_1, x_2 \to \circ \to \hat{y}$ (no hidden).

$C=6$ (x_1, x_2)

more layer, with hidden, even learn complex nonlinear relation

loss function

eg: $y^{(i)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ $\hat{y}^{(i)} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.3 \\ 0.1 \end{bmatrix}$ $C=4$

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

$\mathcal{L}(y^{(i)}, \hat{y}^{(i)}) = -\sum_{j=1}^{6} y_j^{(i)} \log \hat{y}_j^{(i)} = -y_2^{(i)} \log \hat{y}_2^{(i)} = -\log \hat{y}_2^{(i)}$

small / large

$Y = [y^{(1)}, y^{(2)}, \ldots y^{(m)}]$ $(4, m)$

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

$\hat{Y} = [\hat{y}^{(1)}, \hat{y}^{(2)}, \ldots \hat{y}^{(m)}]$ $(4, m)$

$= \begin{bmatrix} 0.3 & \ldots \\ 0.2 & \\ 0.3 & \\ 0.2 & \end{bmatrix}$

Back Prop. $dz^{[2]} = \hat{y} - y = \frac{dJ}{dz^{[2]}}$ $(4,1)$

Deep learning frameworks

Intro to programming frameworks

- Caffe/Caffe2
- CNTK
- DL4J
- Keras
- Lasagne
- mxnet
- PaddlePaddle
- Tensorflow
- Theano
- PyTorch

criteria:
- Ease of programming (development and deployment)
- Running speed
- Truly open (good governance)

W2: Error Analysis

Error Analysis

Carryout error analysis

Ideas for cat detection:
1: Fix pictures of dogs being recognized as cats
2: Fix great cats (lions, panthers, etc...) being misrecognized.
3: Improve performance on blurry images.

list:

Image	dog	great cat	blurry	...	comments
1	✓				
2			✓		
3		✓	✓		
⋮					
% total	8%	12%	61%		

Error analysis:
① Get ~100 images misrecognized on dev set
② Count up how many are others.
→ choose direction to improve.

Clean up incorrectly labeled data

DL is quite robust to random errors in the training set. not systematic error.

Fixed Incorrectly labeled rate, if overall dev set error: 10% | 2%
Errors due to incorrect labels: 0.6% | 0.6%
Errors due to other causes: 9.4% | 1.4%
→ not important right now; → more worthwhile to fix

Goal of dev set: help you select between classifier A & B

Guides:
- Apply same process to dev/test sets to make sure they continue to come from same distribution
- Consider examining examples your algo got right as well as one it got wrong.
- Train and dev/test set may now come from slightly different distributions.

Build your first system quickly, then iterate

Speech recognition example:
- Background noise:
  - Cafe noise
  - Car noise
- Accented speech
- Far from microphone
- Young children's speech
- ...

- Setup dev/test set and metric
- Build initial system quickly.
- Use bias/variance analysis & Error analysis to prioritize next steps.

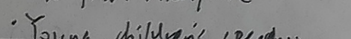
unless familiar with this domain or more experience (paper...), the more

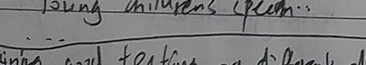
Training and testing on different distributions

Mismatched datasets

Cat app example

Data from web: 200k; Data from app: 10k (shuffle)

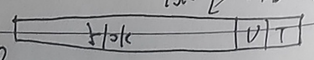
✗ option1:


✓ option2:


Speech recognition example

Training:
- Purchased
- Smart speaker control
- Voice keyboard
- ...
- 500k

Dev/test: Speech activated rearview mirror



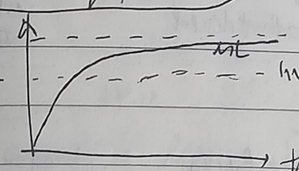
When to change dev/test set and metrics ← doing well on your metrics & dev/test set, but not doing well on application

Orthogonalization
→ 1: How to define a metric : place target
→ 2: How to do well on this metric : How to shoot

$J = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(y^{(i)}, \hat{y}^{(i)}) \Rightarrow J = \frac{1}{\sum w^{(i)}}\sum_{i=1}^{m} w^{(i)}\mathcal{L}(y^{(i)}, \hat{y}^{(i)})$, $w^{(i)} = 1$ if $x^{(i)}$ is non-porn, $w^{(i)} = 10$ if porn (punish more)

Why human performance?

Comparing to human-level performance



Human are quite good at some tasks. As long as ML is worse than humans, you can:
- Get labeled data from humans
- Gain insight from manual error analysis: (why human get this right?)
- Better analysis of bias/variance

Avoidable bias

Human (≈ Bayes)	1%	7.5%	↕ Avoidable bias
Training error	8%	8%	
Dev error	10%	10%	↕ Variance
	Focus on bias	Focus on variance	

When approach human-level, not know how far away you are from Bayes error, therefore how much you should be trying to reduce avoidable bias
harder to tease out the bias and variance effects.

Surpassing human-level performance

Team of human	0.5%	↑ avoidable bias	0.5%	? 0.1% ? 0.3% ?
Human	1%		1%	
Train error	0.6%		0.3%	
dev error	0.8%		0.4%	

ML significantly surpassing human:
- Online advertising
- Product recommendations
- Logistics (Predict transit time)
- Loan approval → structured data. not natural perception

Speech recognition
Image rec
Medical:
- ECG
- Skin cancer

Improving your model performance

two fundamental assumptions of supervised learning.
1: fit train set well ~ avoidable bias
2: train set performance generalizes well on dev/test set. ~ variance

Human-level
↓ Avoidable error →
Training error
↓ Variance →
Dev error

Train bigger model
Train longer/better optimization algorithms
- Momentum, RMSprop, Adam
NN architecture
Hyperparameter search, Cafe

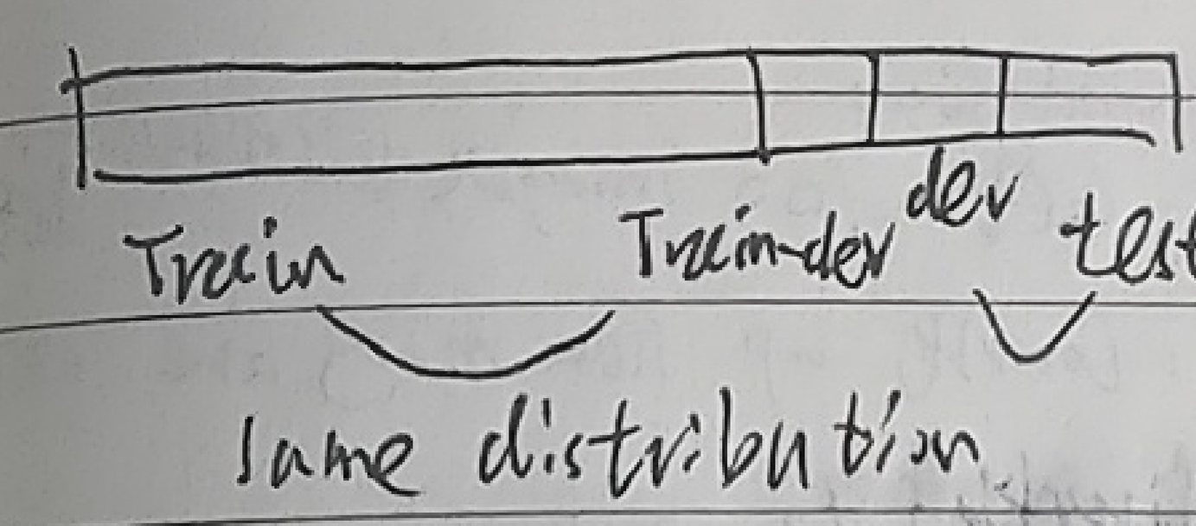
More data
Regularization
- L2, dropout, data augmentation
NN architecture
hyperparameter search

Bias and variance with mismatched data distributions

Cat classifier example

- "Human level" ≈ 0%
- Train set error. 1%
- Dev error. 10%

Train error	1% ↕ variance	1%	10% ↕ avoidable bias	10%	human level ↕ avoidable bias
Train-dev error	9%	1.5	11%	11%	
dev error	10%	10% ↕ data mismatch	12%	20% ↕ data mismatch	



Human level	↕ avoidable bias	0%	4%
Train set error	↕ variance	7%	7%
Train-dev set error	↕ data mismatch	10%	10%
dev set error	↕ degree of overfitting to dev set	12%	6%
test error		12%	6% } easy task

More general formulation (rearview mirror example)

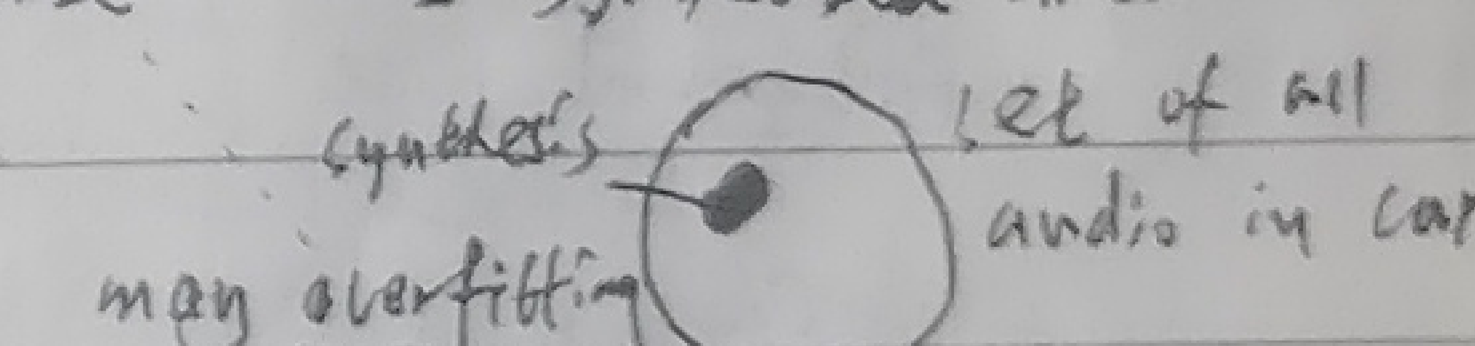
	General speech recognition	Rearview speech recognition
Human level	"Human level" 4%	6%
Error on examples NN train on	"Train error" 7%	6%
Error on examples NN not train on	"Train-dev error" 10%	Dev/Test error 6%

↕ avoidable bias
↕ variance
←→ data mismatch

Addressing data mismatch

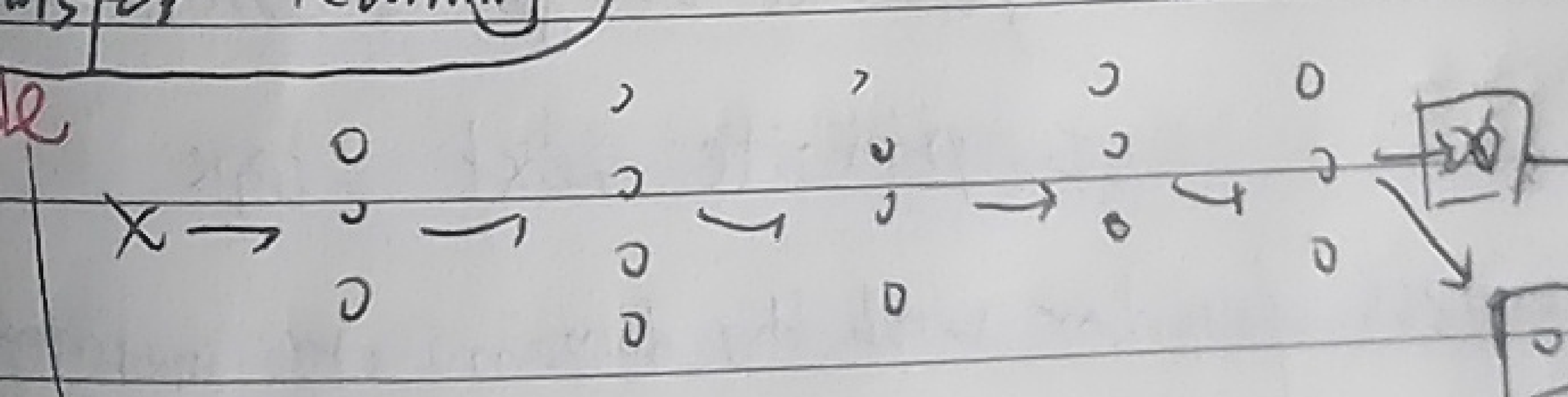
- Carry out manual error analysis to try to understand difference between training and dev/test sets.
  E.g. noisy-car noise, street numbers.
- Make training data more similar; or collect more data similar to dev/test sets.
  E.g. simulate noisy in-car data

Artificial data synthesis: clean speech (10000h) + car noise (1h) = synthesized in-car audio



Transfer learning

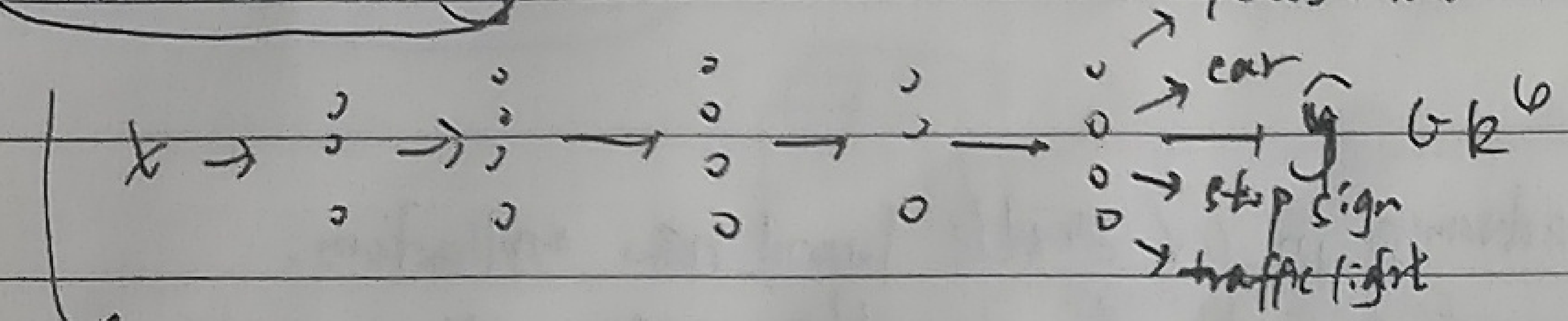
Learning for multiple tasks



When? A → B

- Task A, B have same input X
- A lot more data for A than B.
- Low level features from A could be helpful for B

Multi-task learning



loss: $\hat{y}^{(i)}_{(4,1)} \rightarrow \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{4} \ell(\hat{y}_j^{(i)}, y_j^{(i)})$ — usual logistic loss

sum only over valid of j with 0/1

Unlike softmax: one image can have multiple labels

$Y = \begin{bmatrix} 1 & ? & 0 & \cdots \\ 0 & 1 & ? & \\ ? & 0 & ? & \\ 1 & ? & ? & \end{bmatrix}$

When?

- Train on a set of tasks that share low-level features.
- Usually: Amount of data you have for each task is quite similar.
- Can train a big enough neural network to do well on all the tasks.

A1 1000, A2 1000] 99,000 ... A100 1,000

End to end DL

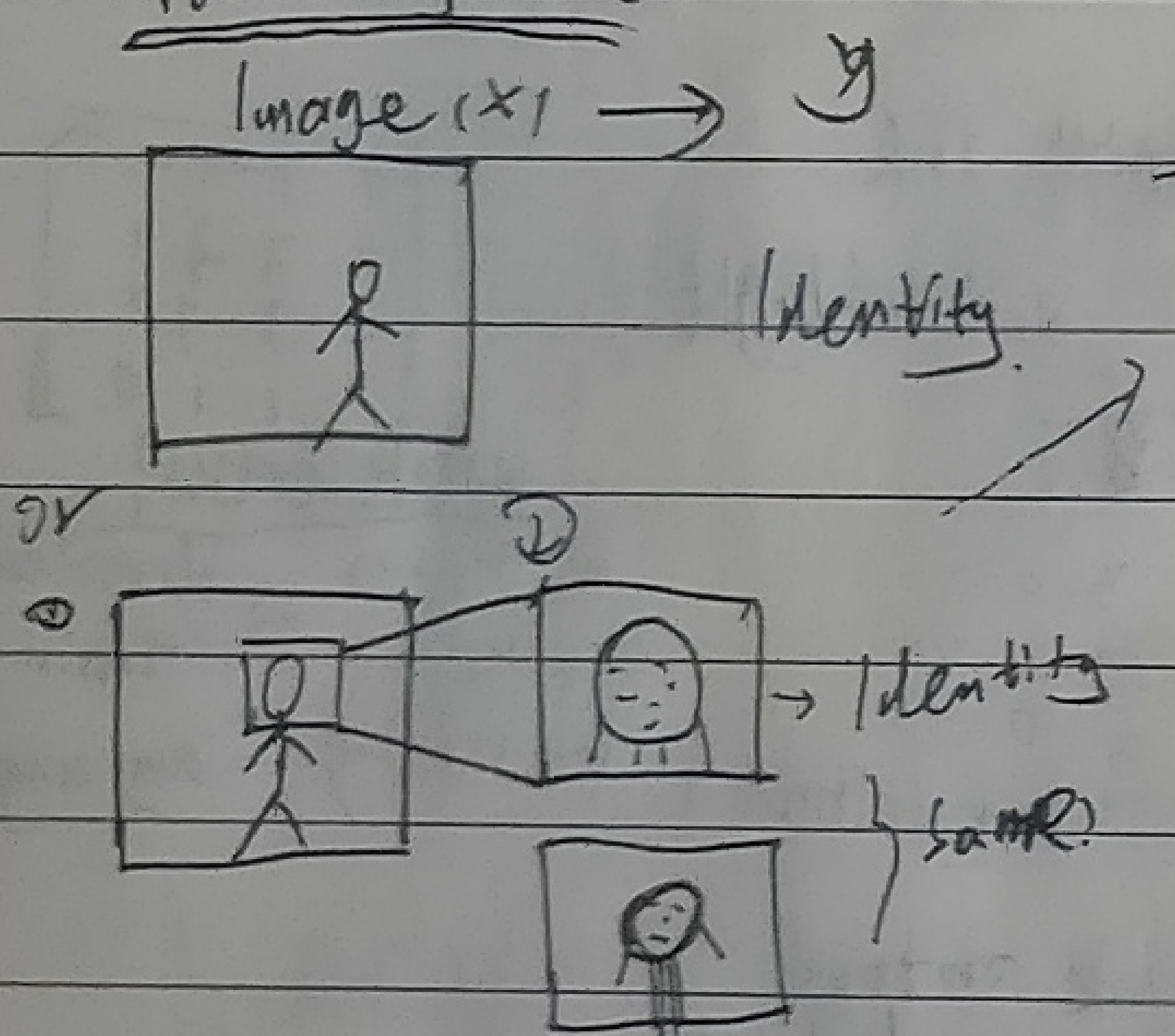
E2E learning

Speech recognition example.

→ audio →(MFCC) features →(ML) Phonemes → Words → transcript : traditional pipeline — 3,000h

→ audio → transcript : DL — 10,000 ~ 100,000h

Face recognition



Machine translation

x. English → text analysis → ... → French y

x. English → French y

Estimating child's age

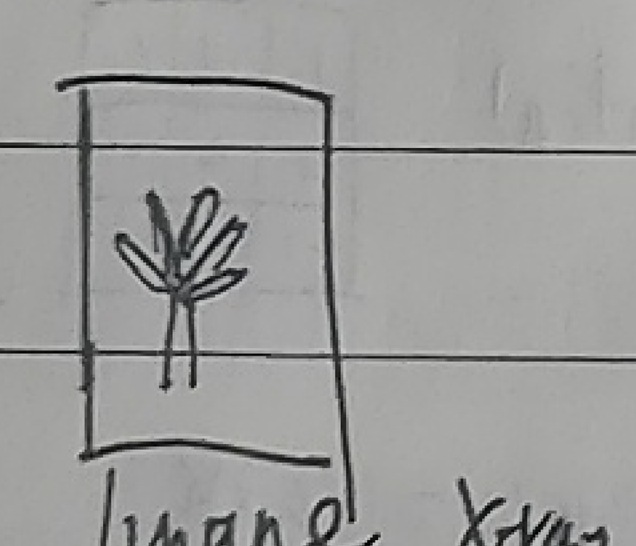


Image →(1) bones →(2) age

Image → age

End to end DL.

Pros: ① Let the data speak
② Less hand-designing of components needed.

Cons: ① may need large amount of data
② Excludes potentially useful hand-designed components.

Source of knowledge: Data > hand-design — double edged sword

When to apply end to end DL?

Key question: Do you have sufficient data for a function of the complexity needed to map x to y?

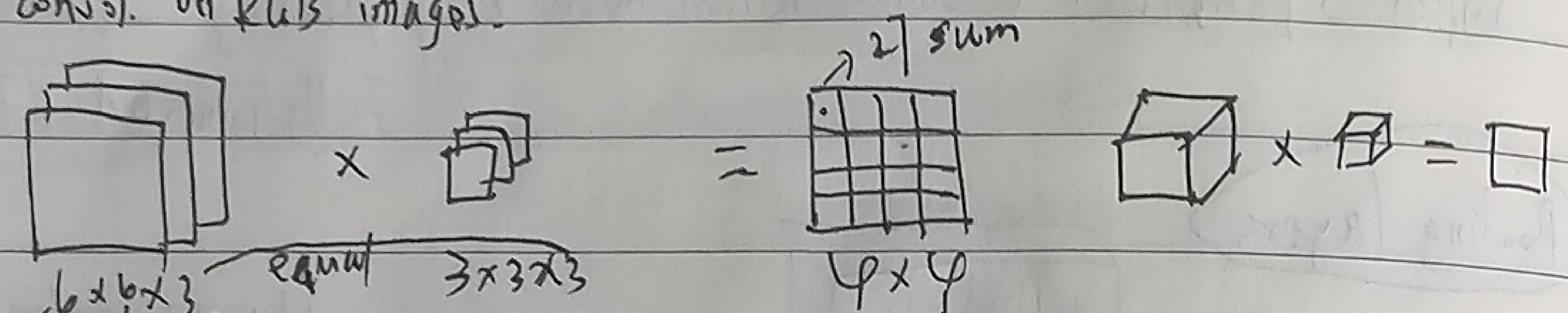
autonomous driving: {image, radar, lidar} →(DL) Cars / → Pedestrians → (motion planning) route →(control) steering

- Use DL to learn individual components.
- Carefully choose X → Y depending on what task you get data for

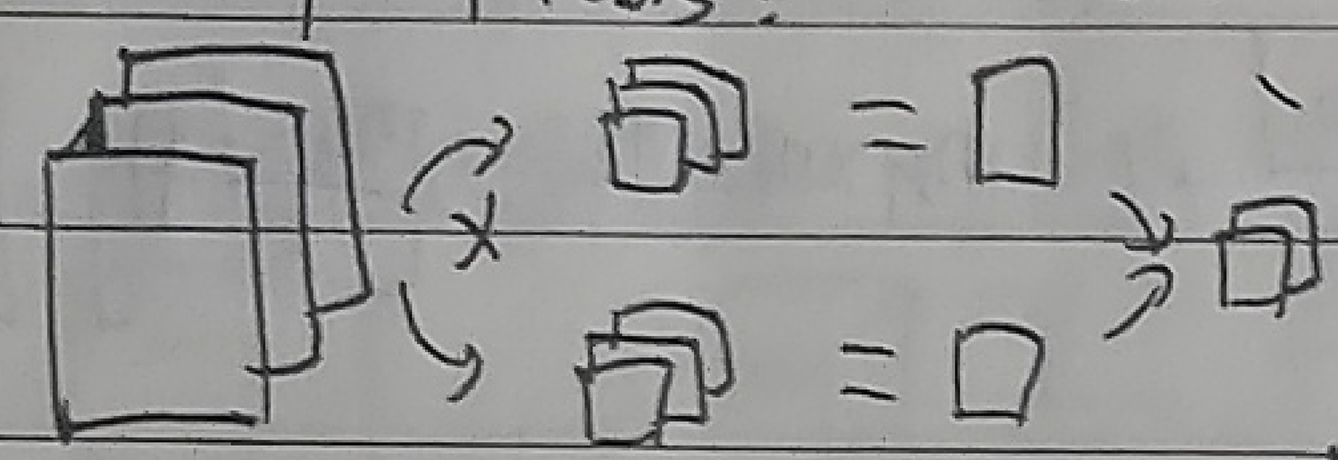
End to End is not mature now.

Convolution over volumes

conv. on RGB images.



multiple filters:



R: 1 0 -1 / 1 0 -1 / 1 0 -1	G: 0 0 0 / 0 0 0 / 0 0 0	B: 0 0 0 / 0 0 0 / 0 0 0

detect red vertical

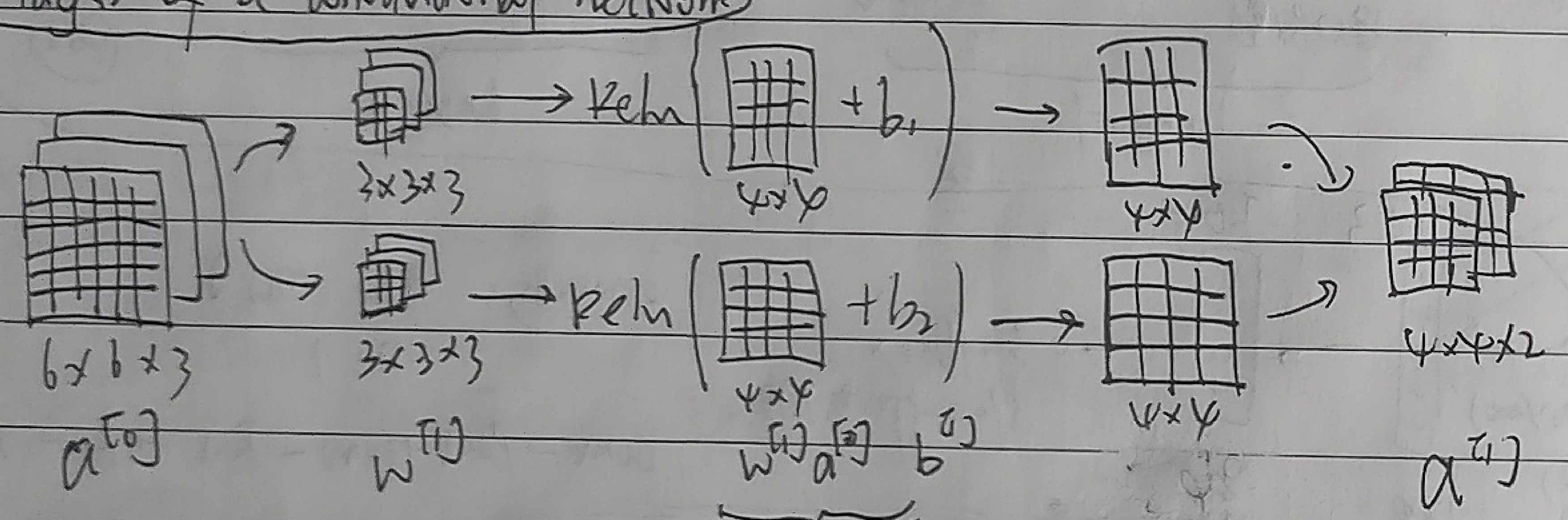
R: 1 0 -1 / 1 0 -1 / 1 0 -1	G: 1 0 -1 / 1 0 -1 / 1 0 -1	B: 1 0 -1 / 1 0 -1 / 1 0 -1

detect all vertical

$n \times n \times n_c * f \times f \times n_c \rightarrow (n-f+1) \times (n-f+1) \times n_c'$ (no padding, stride = 1) (n_c': #filters)

6×6×3   3×3×3   4×4×2

One layer of a convolutional network



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

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

$a^{[0]}$: 6×6×3

$a^{[1]}$: 4×4×2

if 10 filters. parameter nums: $(3\times3\times3+1)\times10 = 280$ (regardless of the image size)

notation: layer l.

$f^{[l]}$: filter size

$p^{[l]}$: padding

$s^{[l]}$: stride

$n_c^{[l]}$: #filters

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

Activation: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

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

bias: $n_c^{[l]} \rightarrow (1,1,1,n_c^{[l]})$

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

Output: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

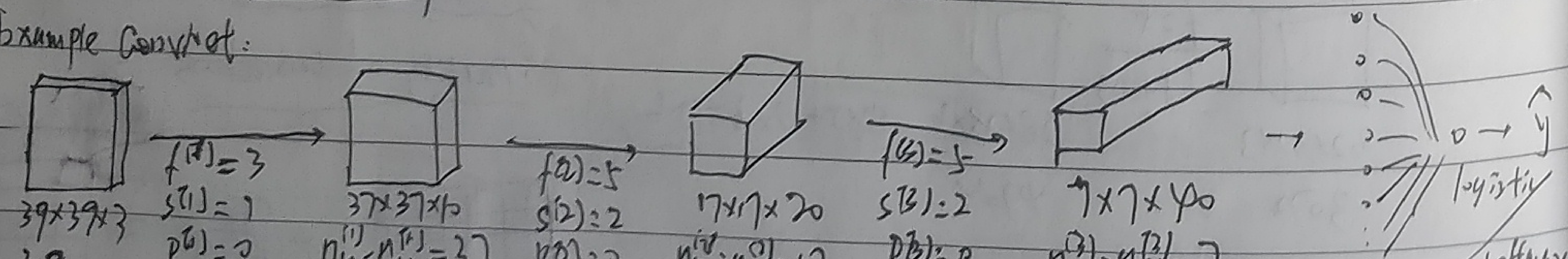
$n_{H/W}^{[l]} = \left\lfloor \frac{n_{H/W}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$

$A^{[l]} = m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

$n_c^{[l]} \times n_H^{[l]} \times n_W^{[l]}$

A simple convolution network example

Example ConvNet:



$n_H^{[0]} = n_W^{[0]} = 39$, $n_c^{[0]} = 3$

$n_H^{[1]} = n_W^{[1]} = 37$, $n_c^{[1]} = 10$

$n_H^{[2]} = n_W^{[2]} = 17$, $n_c^{[2]} = 20$

$n_H^{[3]} = n_W^{[3]} = 7$, $n_c^{[3]} = 40$

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

$n_H, n_W \downarrow$ 39, 37, 17, 7

$n_c \uparrow$ 3, 10, 20, 40

C4. Convolutional Neural Networks

W1: CNN.

Computer vision

① rapid advances in CV enable brand new applications.

② CV is creative. transfer to other area. sr.

→ problems: Image classification. Neural style transfer. object detection.

→ large image. eg. 1000×1000×3. 3M. W_1: [1000, 3M]. 3bill.

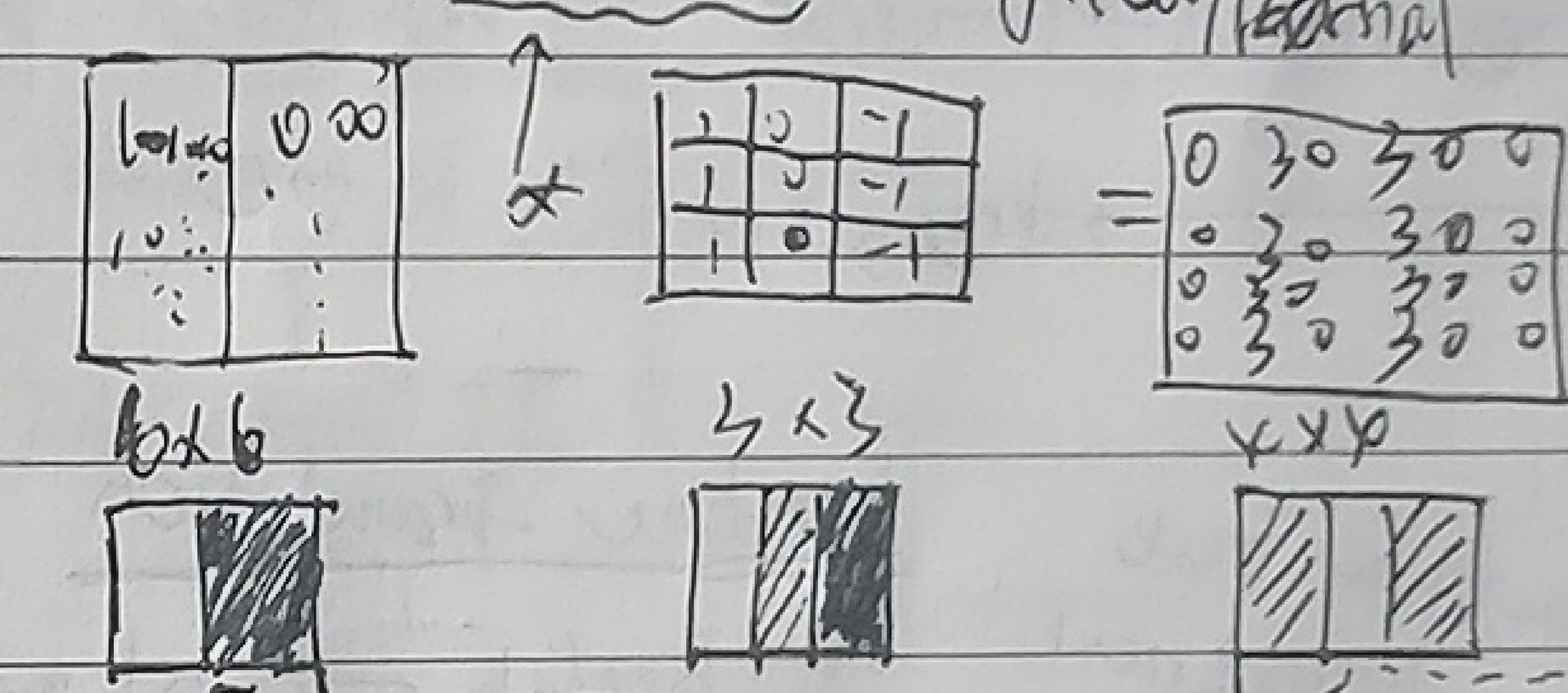
Edge detection example

Vertical edge detection. $n \times n$ * filter/kernel (convolution)

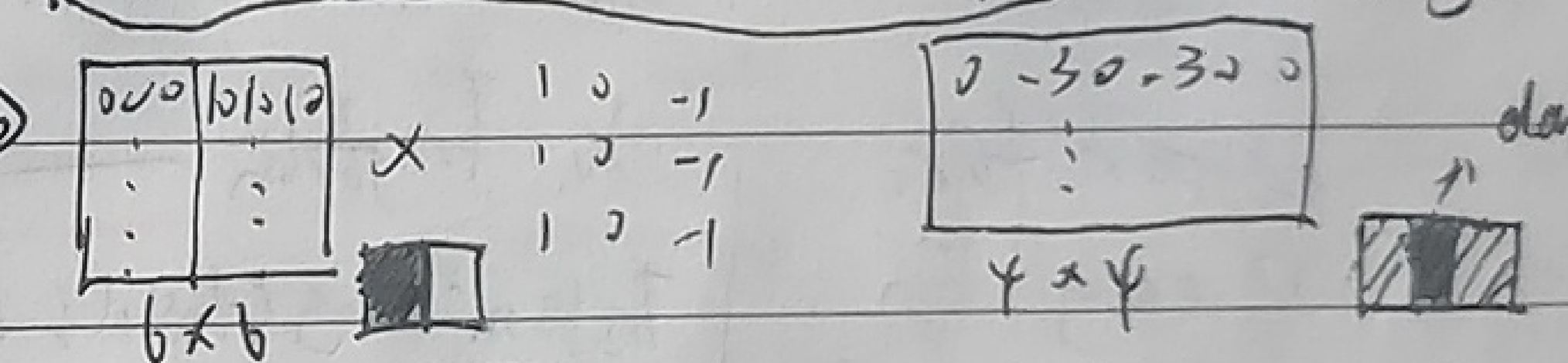
python: conv_forward

tf: tf.nn.conv2d

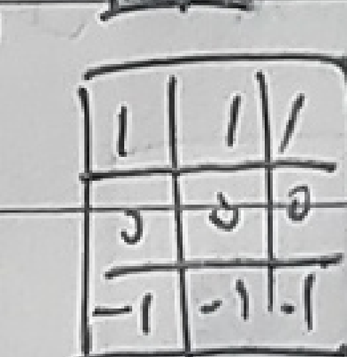
keras: Conv2D



More edge detection



horizontal



Sobel filter: 1 0 -1 / 2 0 -2 / 1 0 -1

Scharr filter: 3 0 -3 / 10 0 -10 / 3 0 -3

0 1 -1 0 / 1 3 -3 -1 / 1 3 -3 -1 / 0 1 1 0 — detect vertical

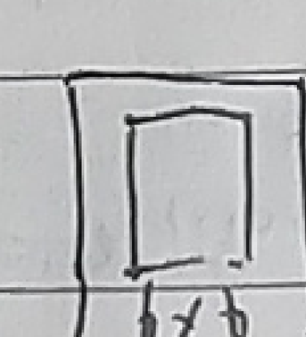
$w_1\ w_2\ w_3$ / $w_4\ w_5\ w_6$ / $w_7\ w_8\ w_9$ → deep learn, train, any angel.

Padding

6×6 * 3×3 = 4×4

$n \times n$ $f \times f$ $(n-f+1) \times (n-f+1)$.

cons. ① shrink output ② throw away info. from edge.



6×6, 8×8, p=1. $(n+2p-f+1)\times(n+2p-f+1)$

valid convolution: $n\times n * f\times f \rightarrow (n-f+1)\times(n-f+1)$ p=0

same convolution: $\rightarrow (n+2p-f+1)\times(n+2p-f+1)$

usually odd. p: int, center.

$n+2p-f+1 = n \Rightarrow p = \frac{f-1}{2}$. same

Strided convolutions

7×7 * 3×3 (stride=2) = 3×3

$n\times n$ $f \times f$ stride = s, padding = p

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

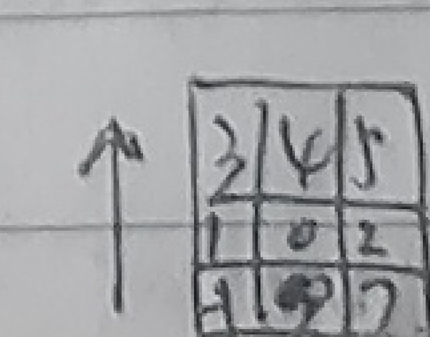
$\lfloor z \rfloor = floor(z)$ ⇓ filters no out

$n\times n$ image $f\times f$ filter

padding p   stride s

⇒ output size: $\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$

cross-correlations v.s. convolution (in math)



no flips

enable $\Rightarrow (A*B)*C = A*(B*C)$

W2 : case studies

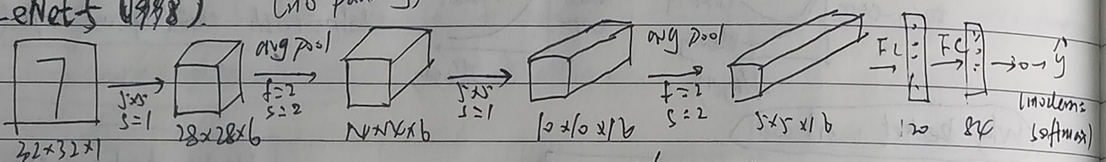
Why look at case studies?

case studies

classic networks : LeNet-5 / AlexNet / VGG / ResNet / Inception

classic networks

LeNet-5 (1998) (no padding)



60k parameters, $n_H, n_W \downarrow$, $n_C \uparrow$

Conv-Pool-Conv-Pool-Fc-Fc-output

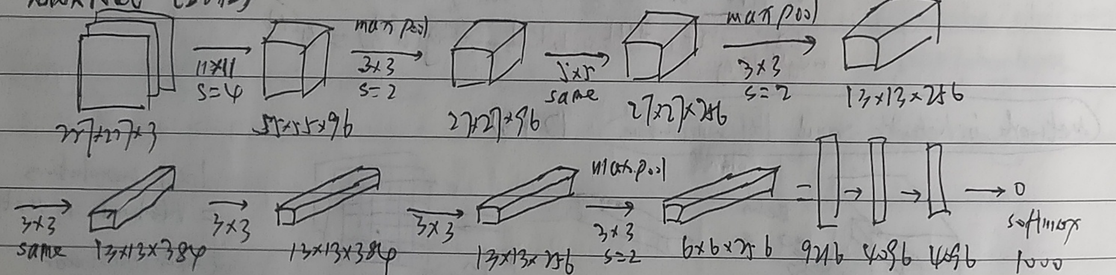
① Sigmoid/Tanh, not Relu.

② save computation. (different filter. different n_C)

③ non-linear after pooling

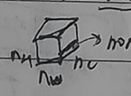
paper. Section II / Section III

AlexNet (2012)



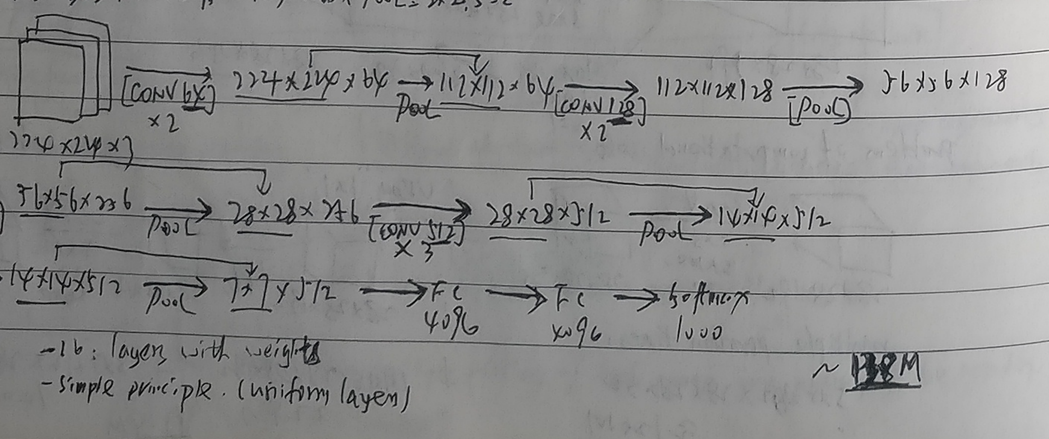
- Similarity to LeNet. much bigger ~60M parameters.
- ReLu
- Multiple GPU
- local response normalization (LRN) (doesn't help so much)

(start coming in CV)



VGG16 (2014)

CONV = 3×3 filter. s=1, same; MAX-POOL = 2×2, s=2

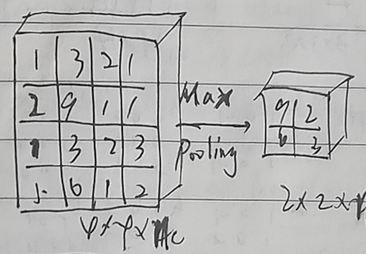


~138M

- 16: layers with weights
- simple principle. (uniform layers)

Types of layers in a convolutional network { Convolution (CONV) / Pooling (POOL) / Fully-connected (FC)

Pooling layers

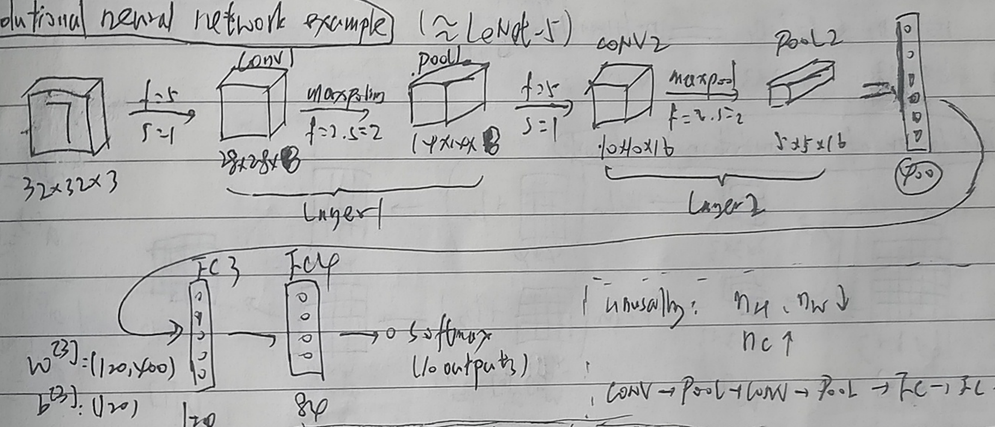


Hyperparameters: f=2, s=2; max/average pooling

(no need to learn)

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

Convolutional neural network example ($\approx$ LeNet-5)



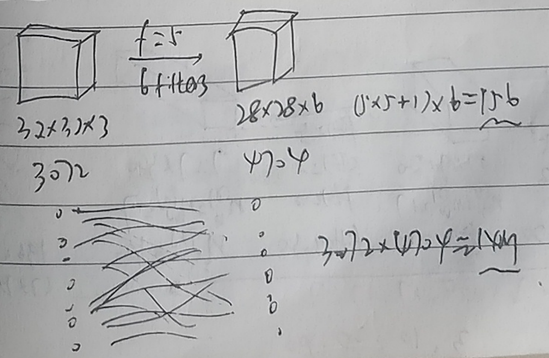
Layer1 Layer2

usually: $n_H, n_W \downarrow$, $n_C \uparrow$

CONV → Pool → CONV → Pool → FC → FC → softmax

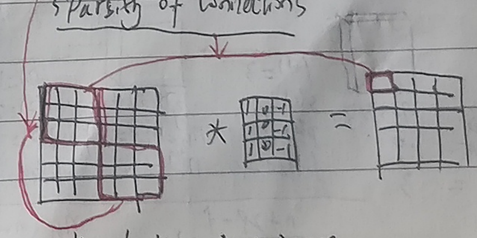
	Activation shape	Activation size	# Parameters
Input	(32, 32, 3)	3072	0
CONV (f=5, s=1)	(28, 28, 8)	6272	$208 = (5\times5+1)\times8$
Pool1	(14, 14, 8)	1568	0
Conv (f=5, s=1)	(10, 10, 16)	1600	$416 = (5\times5+1)\times16$
Pool2	(5, 5, 16)	400	0
FC3	(120, 1)	120	$48001 = (400+1)\times120$
FC4	(84, 1)	84	$10081 = (120+1)\times84$
Softmax	(10, 1)	10	$850 = (84+1)\times10$

Why convolutions?



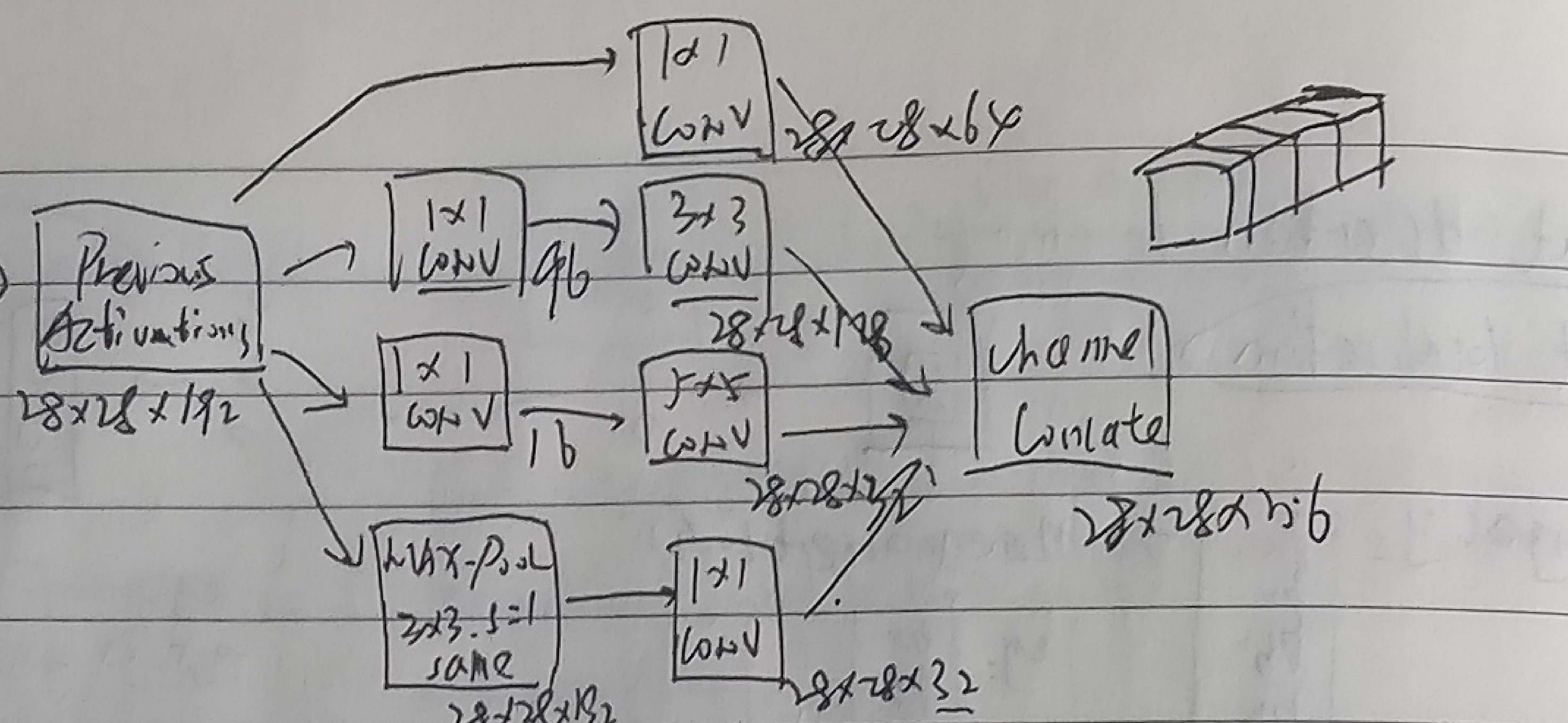
Parameter sharing

Sparsity of connections



translation invariance

Inception Network



GooleNet. Inception module + max-pool + Inception module .... → softmax

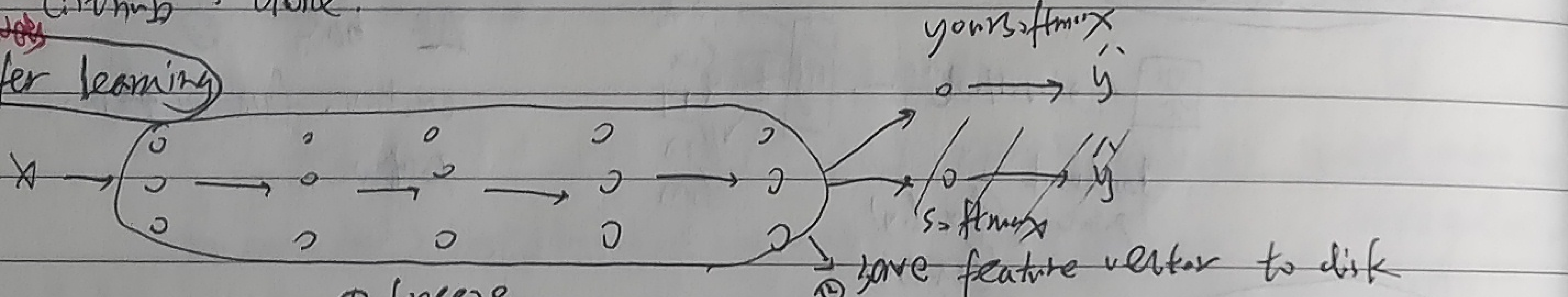
↳ softmax

Using Open-source implementation

I Practical advices for using ConvNets

Github . clone.

Transfer learning



more data. freeze less or more data. { initialize later weights; clip later layers. build own layers

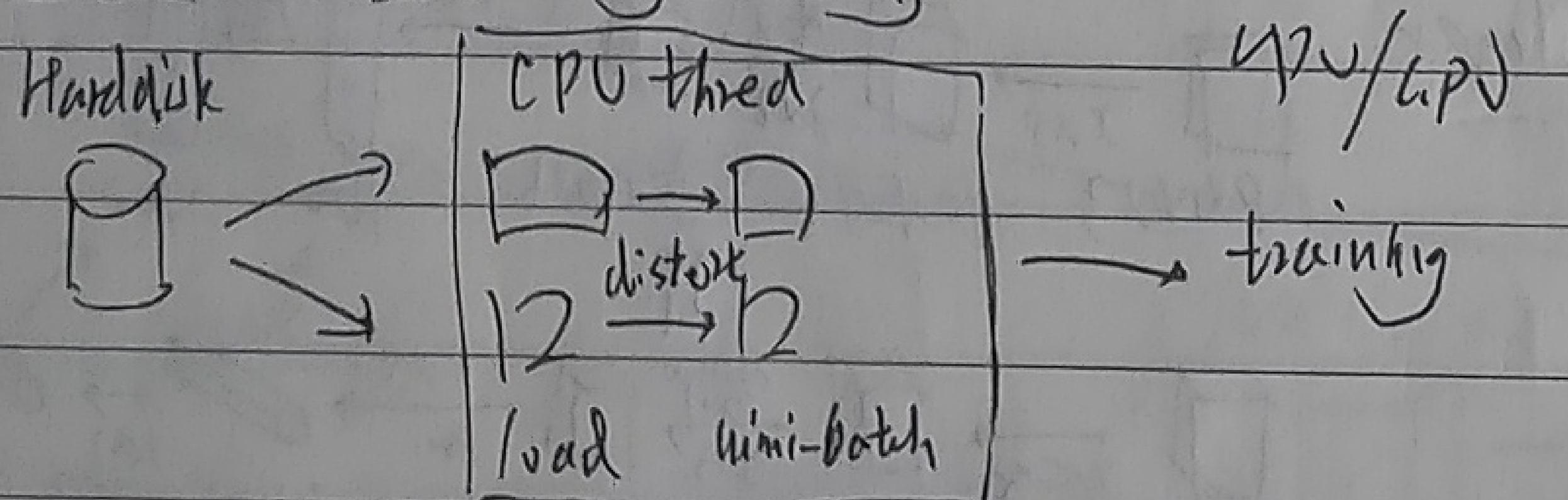
Data Augmentation

CV usually lack data.

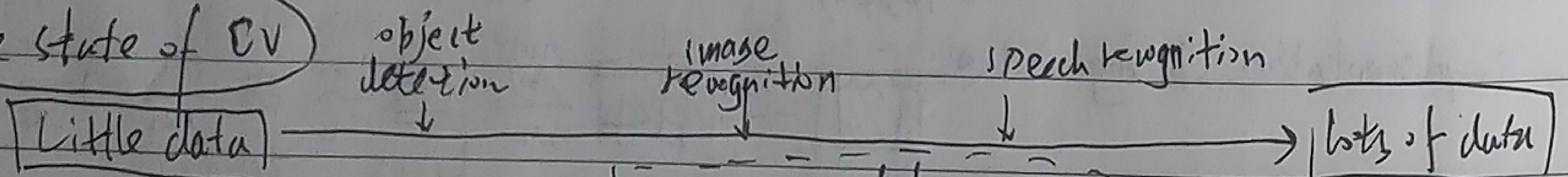
Common augmentation method. ① mirroring, random cropping, Rotation, Shearing, local wraping,

② color shifting. R G B +20, -20, +20; -20, +20, +20; +5, 0, +50. (sunlight, illumination), PCA (AlexNet paper)

Implementing distortions during training:



The state of CV



More hand-Engineering ("hack")
↑ transfer learning

Two source of knowledge.
① Labeled data
② Hand engineered features/network architecture/other components

Simpler algo.
Less hand-Engineering

Tips for doing well on benchmarks/winning competitions

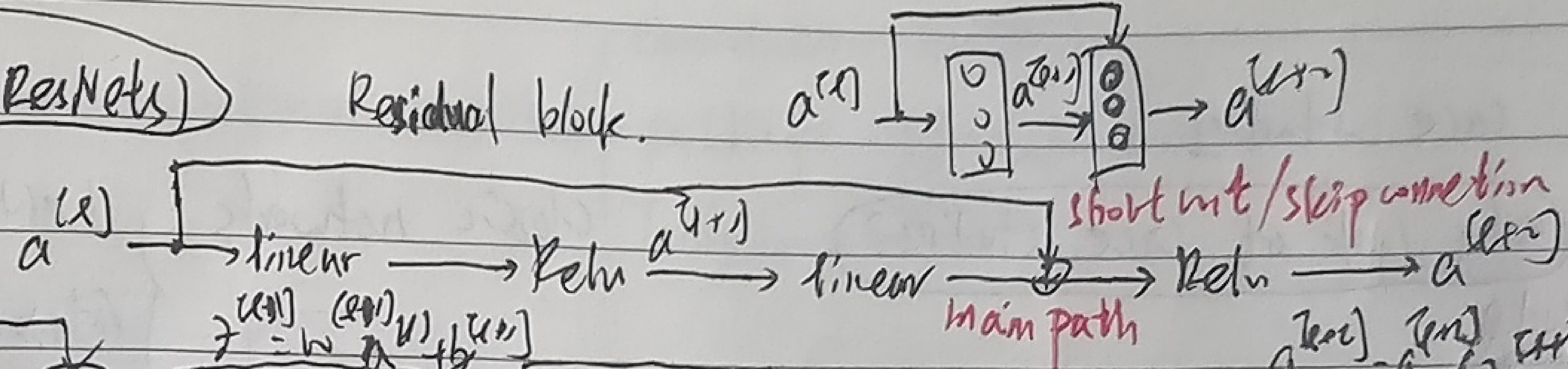
Ensembling . Train several networks independently and average the results (3~15 networks)

Multi-crop at test time
. Run classifier on multiple versions of test image and average the results

10 crop : [drawing] × 2 (mirror)

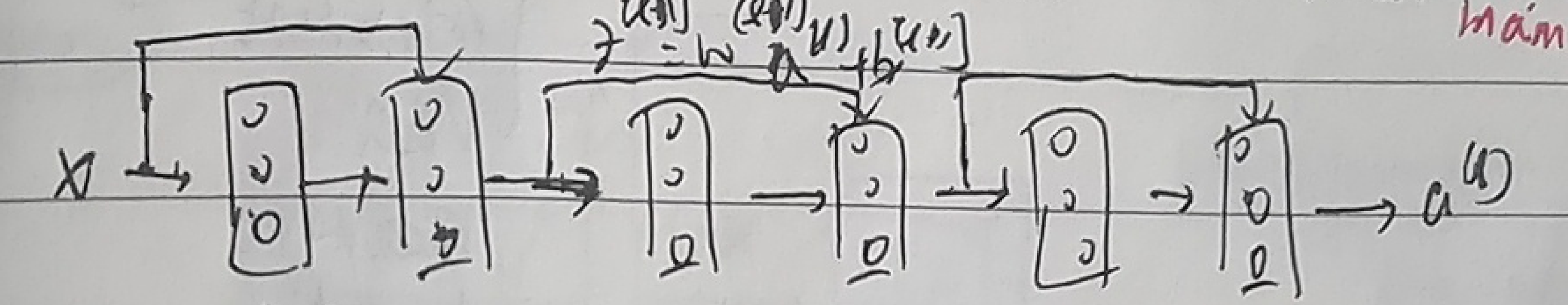
Residual Networks (ResNets)

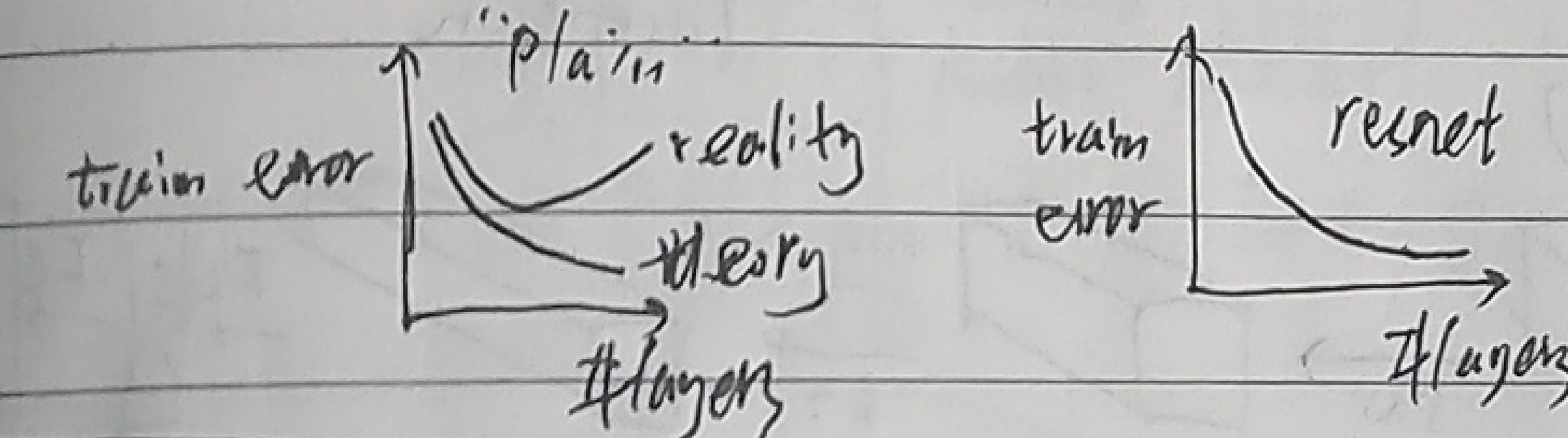
Residual block.



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

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

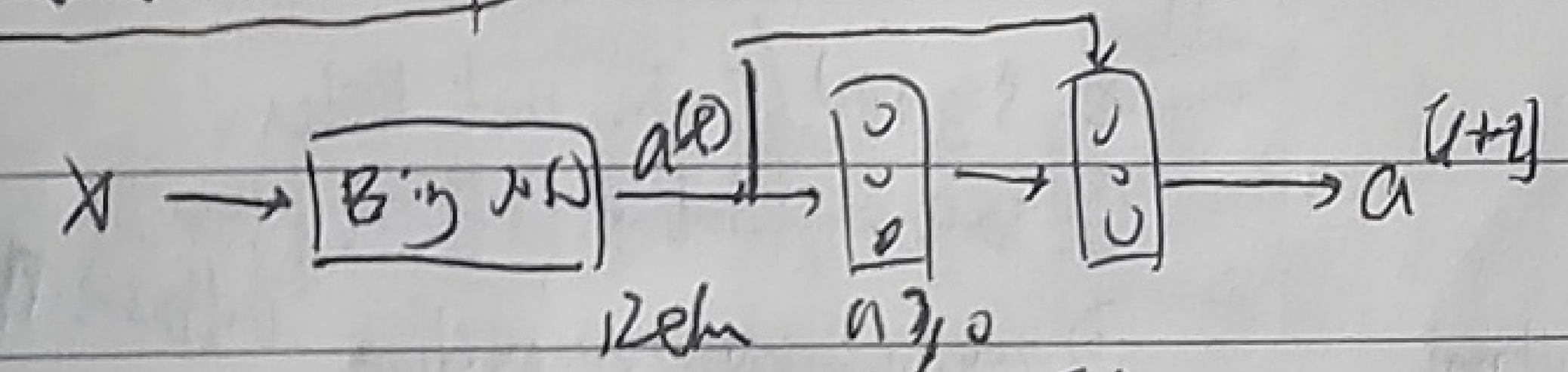




>100 layers. 1000 layers

helps with vanishing and exploding gradients

Why ResNet work well



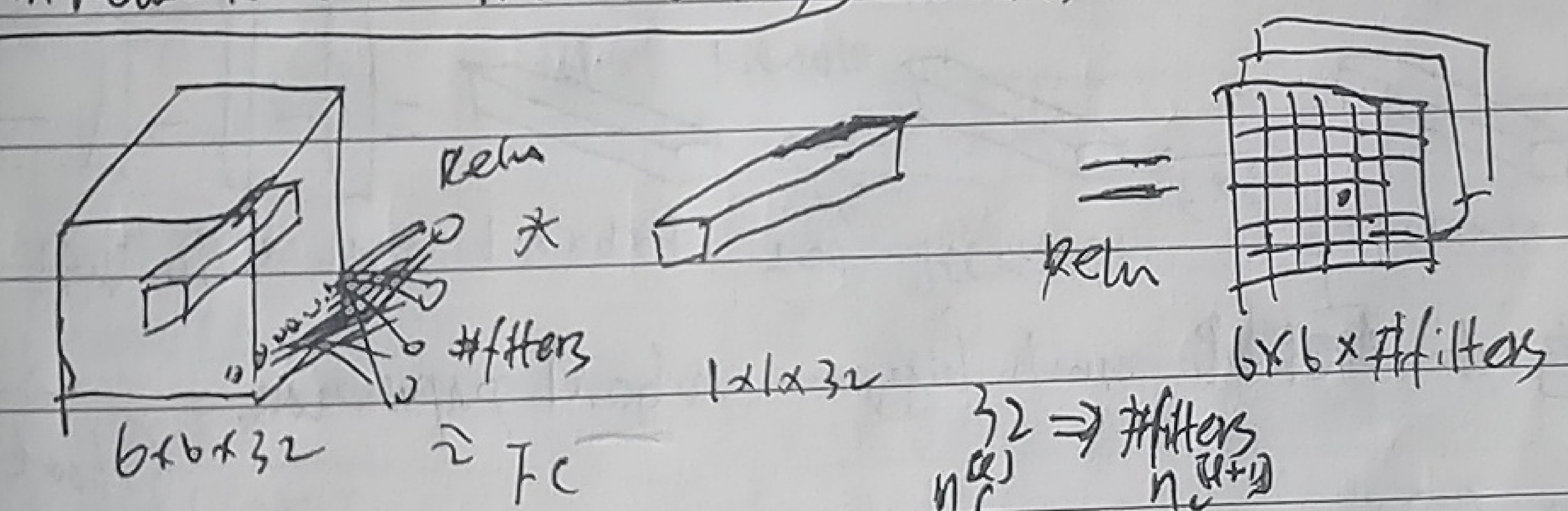
Identity function is easy for residual block to learn!
→ doesn't hurt performance, even helps

$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$ → if $\dim(z^{[l+2]}) \neq \dim(a^{[l]})$. fill 0. Padding 0

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

L2 weight decay: If $W^{[l+2]} = 0$, $b^{[l+2]} = 0$ → $g(a^{[l]}) = a^{[l]}$

Network in Network and 1×1 convolutions (2013)

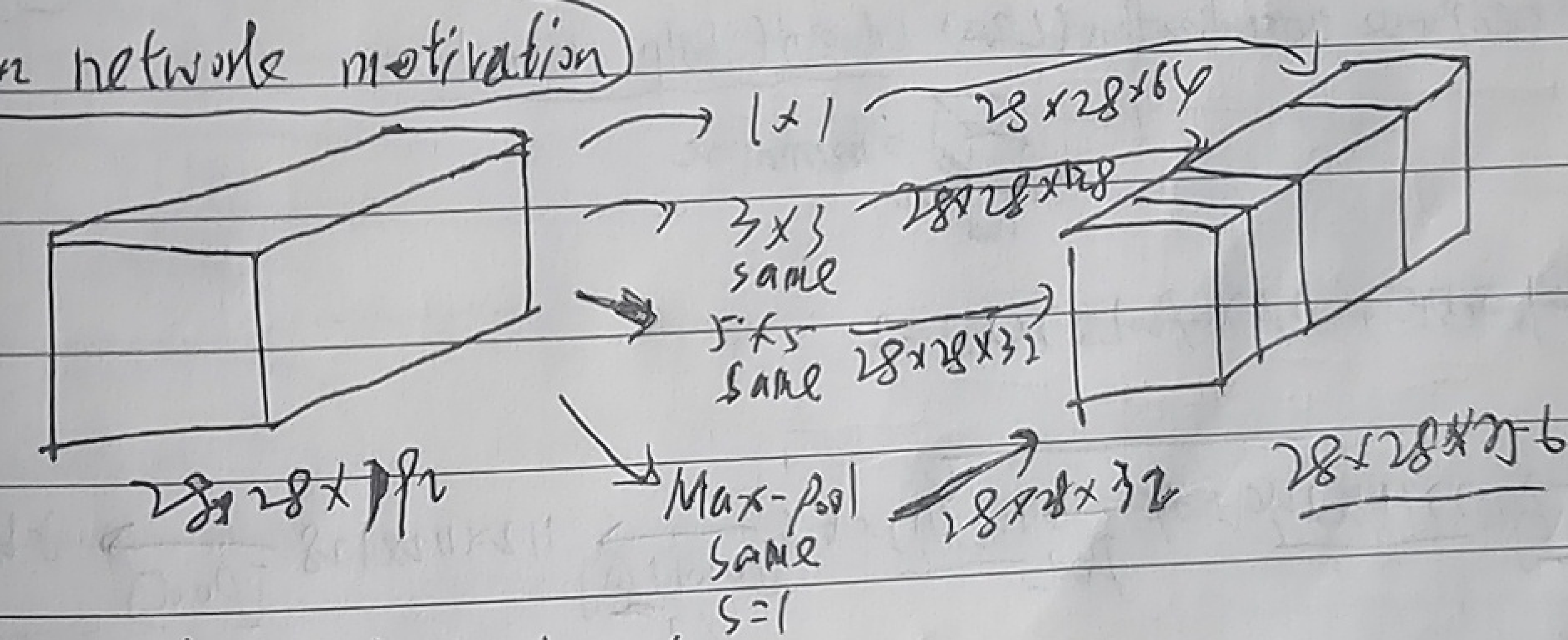


$n_c^{[l]}$ ⇒ $n_c^{[l+1]}$

1×1 convolutions = Network in network.

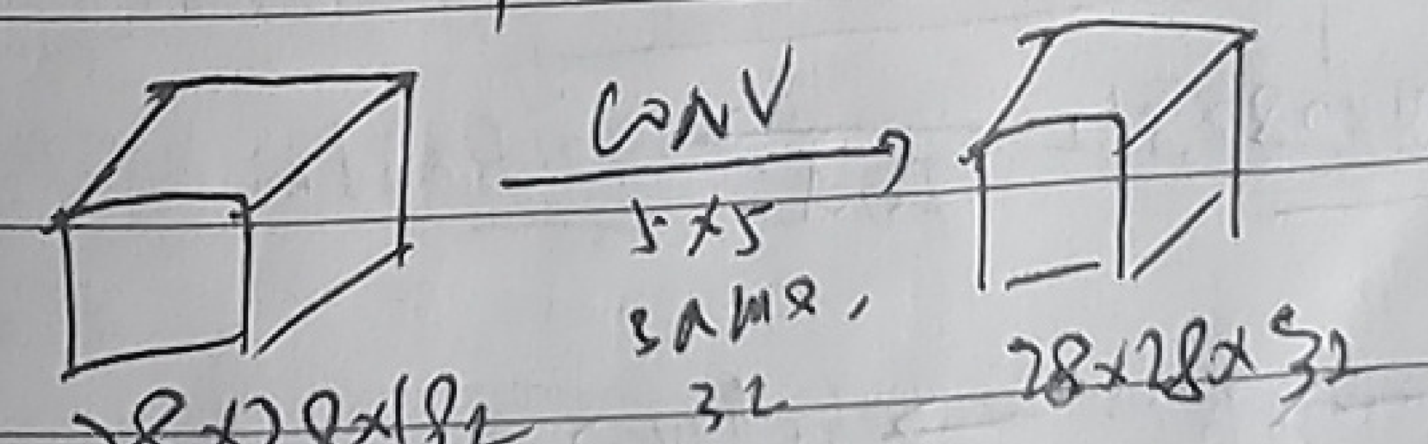
{ add non-linear; shrinkage filters remain/increased

Inception network motivation



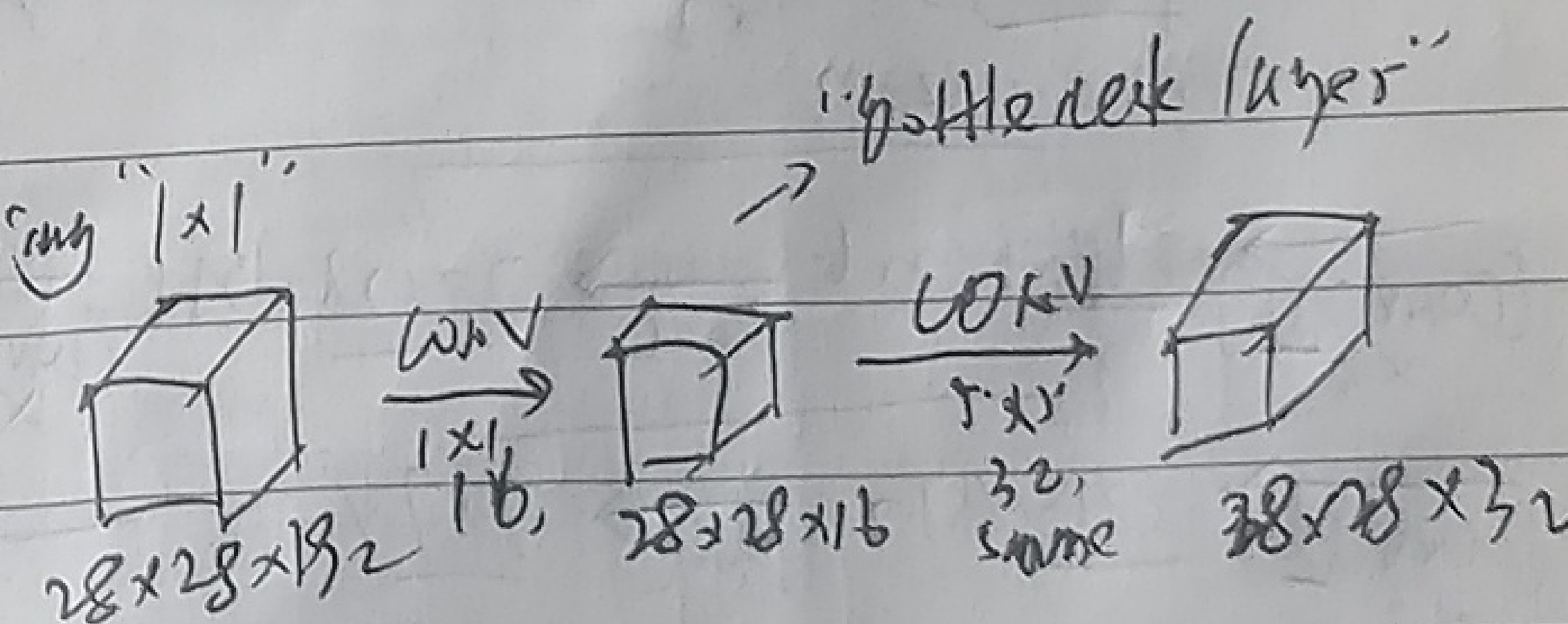
devide by itself

Problem of computational cost.



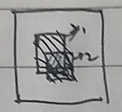
multiple operation times:
5×5×192 × 28×28×32
≈ 120M

Using "1×1"



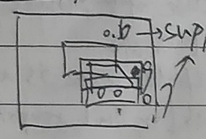
1×1×16 × 28×28×192 + 5×5×16 × 28×28×32
2.4M 10.0M
12.4M

Intersection over union (IoU)



$IoU = \frac{\text{size of } \square}{\text{size of } \square}$, "Correct" if IoU ≥ 0.5

Non-max suppress



Each output prediction is $\begin{bmatrix} P_c \\ b_x \\ \vdots \end{bmatrix}$

output maximal probabilities classifications, and suppress the close by ones that are non maximal

Discard all boxes with $P_c \le 0.6$

While remaining boxes:

- Pick the box with the largest P_c output that has a prediction.
- Discard any remaining boxes with IoU ≥ 0.5 with previous box.

(multiple object: independently implement non-max.)

Anchor Boxes

overlapping objects

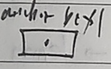
Previously:

output: 3×3×8

two anchor but 3 object; two object same grid ⇒ 19×19 grid fake

anchor box: by hand, by algo (k-means)

with anchor box.



output: 3×3×16 (3×3×2×8) (grid cell, anchor box)

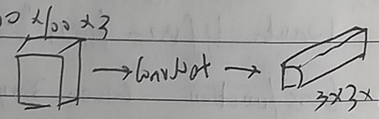
for the grid cell with highest IoU.

YOLO Algorithms

Train

1: pedestrian
2: car
3: motorcycle

Make prediction



outputting the non-max suppressed outputs

- For each grid cell, get 2 predicted bounding boxes.
- Get rid of probability predictions.
- For each class, use non-max suppression to generate final predictions

Region proposal

→ R-CNN: Propose regions. classify proposed regions one at a time. output: label + bounding box

(slow)

→ Fast R-CNN: Propose regions. Use convolution implementation of sliding window to classify all regions.

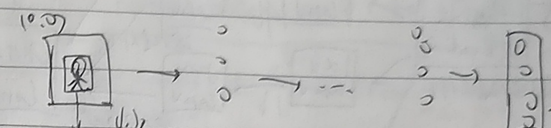
→ Faster R-CNN: Use convolutional network to propose regions

slower than YOLO

instead of traditional segmentation algos.

W3. object detection

Object localization



target $y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$ → is there an object (0/1)

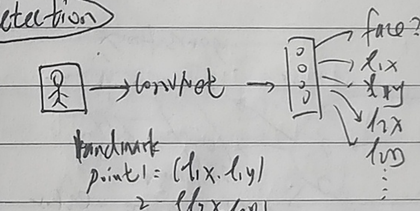
$y = \begin{bmatrix} b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \end{bmatrix}$ or $\begin{bmatrix} ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$ don't care

eg: 0.5, 0.5, 0.3, 0.4 — center, width, height

b_x, b_y, b_w, b_h

$L(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \dots + (\hat{y}_i - y_i)^2 & \text{if } y_1 = 1 \\ (\hat{y}_1 - y_1)^2 & \text{if } y_1 = 0 \ (P_c) \end{cases}$

Landmark detection



landmark point 1: (l_{1x}, l_{1y})

point 2: (l_{2x}, l_{2y})

AR hat

emotion, face recognition, posture ...

Object detection

Sliding windows detection

first: property cut image (find objects)

(x, y) → convnet. then → sliding windows detection

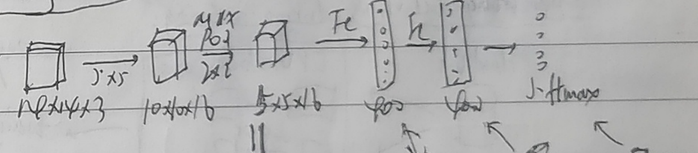
cons:

① computation.

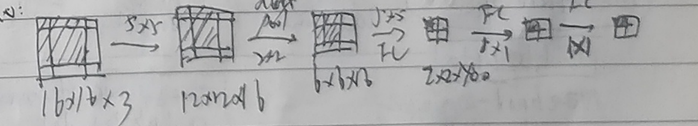
② stride

Convolutional implementation of sliding windows

Turning FC to conv layer.



implementation sliding windows:



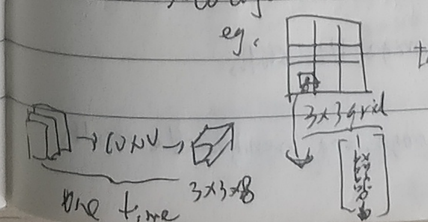
eg: 16×16×3 — 12×12×8 — 8×8×16 — 4×4×16 — 2×2×400 — 1×1×400

28×28×3 → 16×16 → 12×12 → 8×8×16 → 4×4×16 → 2×2×400 → 1×1×4

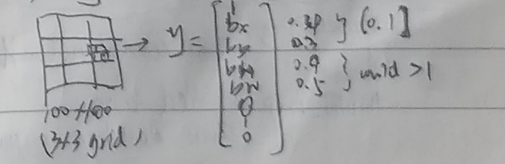
Bounding box prediction

YOLO algorithm.

eg.



Specify the bounding box



Neural Style Transfer

What is neural style transfer

Content img (C.)
Style img (S.) } → Generated img (G.)

What are deep ConvNets learning

[Zeiler et al. 2013. Visualizing and understanding convolutional networks]

- Pick a unit in layer 1,
- Find the nine image patches that maximize the unit's activation.

Repeat for other unit.

Cost function

$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$ → actually, one parameter is enough

[Gatys et al. 2015. A neural algo of artistic style]

- Initiate G randomly.
- Use gradient descent to minimize J(G). $G := G - \frac{\partial}{\partial G} J(G)$

Content cost function

- Say you use hidden layer l (middle depth) to compute content cost.
- Use pre-trained ConvNet (eg. VGG)
- Let $a^{[l](C)}$ and $a^{[l](G)}$ be the activation of layer l on the images.
- If $a^{[l](C)}$ and $a^{[l](G)}$ are similar, both image have similar content.

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

Style cost function

Meaning of the "style" of an image.
↓
: correlation between activations across channels

n_C, n_H, n_W

Style matrix

Let $a^{[l]}_{ijk}$ = activation at (i, j, k). $G^{[l]}$ is $n_C^{[l]} \times n_C^{[l]}$ (i: H, j: W, k: C)

$G^{[l]}_{kk'}$, $k = 1 \dots n_C^{[l]}$

$$\to G^{[l](S)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](S)}_{ijk} a^{[l](S)}_{ijk'}$$

$$\to G^{[l](G)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](G)}_{ijk} a^{[l](G)}_{ijk'}$$

"gram matrix"
⇓ unnormalized cross-covariance (not substract the mean)

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

$$J_{style}(S, G) = \sum_l \lambda^{[l]} J^{[l]}_{style}(S, G)$$

1D and 3D generalizations of models

filter

$14 \times 1 * 5 \times 1 \Rightarrow 10 \times 16$

1D | 3D
EKG | CT scan, movie

filter.

$14 \times 14 \times 14 \times 1 * 5 \times 5 \times 5 \times 1 \Rightarrow 10 \times 10 \times 10 \times 16$

2D

filter.

$14 \times 14 \times 3 * 5 \times 5 \times 3 \Rightarrow 10 \times 10 \times 16$

W4: special application of convnet

Face recognition

What is face recognition?

Face verification:
- Input image, name/ID
- Output whether the input image is that of the claimed person

Face recognition:
- Has a database of K persons.
- Get an input image
- Output ID if the image is any of the K persons (or "not recognized")

1:1 → 99.9% → apply → 1:K

One-shot learning

learn from one example to recognize the person again.

[img] → CNN → softmax. ✗. little train data and new people in need change CNN

→ learn a "similarity" function.

d(img1, img2) = degree of difference between images.

If $d \le \tau$: "same"
$> \tau$: "different" } verification

Siamese network

[DeepFace]

img $x^{(1)}$ → ConvNet → $f(x^{(1)})$.

Goal of learning:
Parameters of NN define an encoding $f(x^{(i)})$
if $x^{(i)}, x^{(j)}$ same person: $\|f(x^{(i)}) - f(x^{(j)})\|^2$ small
else: large

Triplet loss

Anchor, Positive, Negative

Want: $\|f(A) - f(P)\|^2 + \alpha \le \|f(A) - f(N)\|^2$

[FaceNet]

Loss function: Given 3 imgs (A, P, N)

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

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

Choosing the triplet A, P, N.
If randomly, $d(A, P) + \alpha \le d(A, N)$ is easily satisfied.
choose triplets that's are "hard" to train on. : $d(A, P) \approx d(A, N)$

Face verification and binary classification

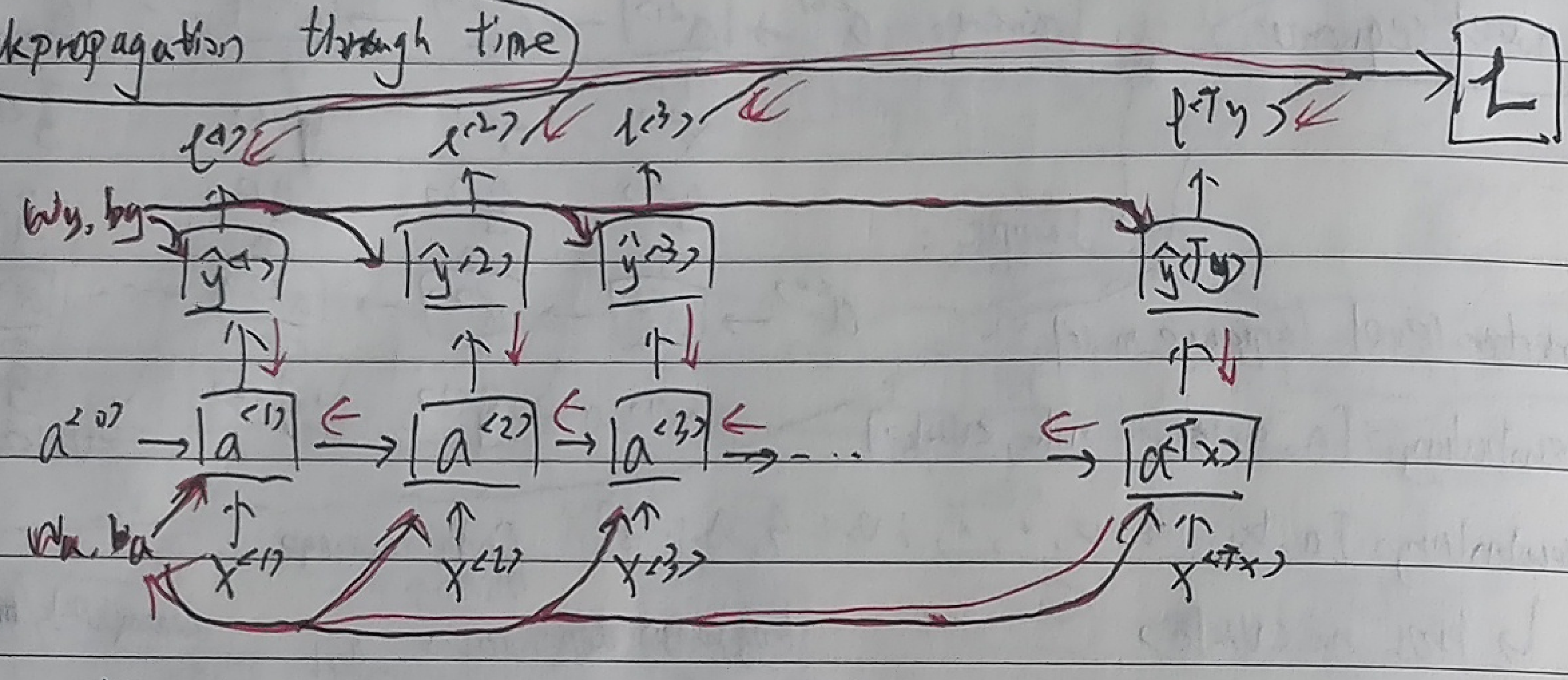
Learning the similarity function

[DeepFace]

$x^{(i)}$ → ConvNet → (e.g. 128) ↘ ○ → $\hat{y}$ (0/1)
new $x^{(j)}$ → ConvNet → could precompute
database (precompute)

$$\hat{y} = \sigma\left(\sum_{k=1}^{128} w_i |f(x^{(i)})_k - f(x^{(j)})_k| + b\right) \to \text{or } \frac{(f(x^{(i)})_k - f(x^{(j)})_k)^2}{f(x^{(i)})_k + f(x^{(j)})_k} \ (\chi^2)$$

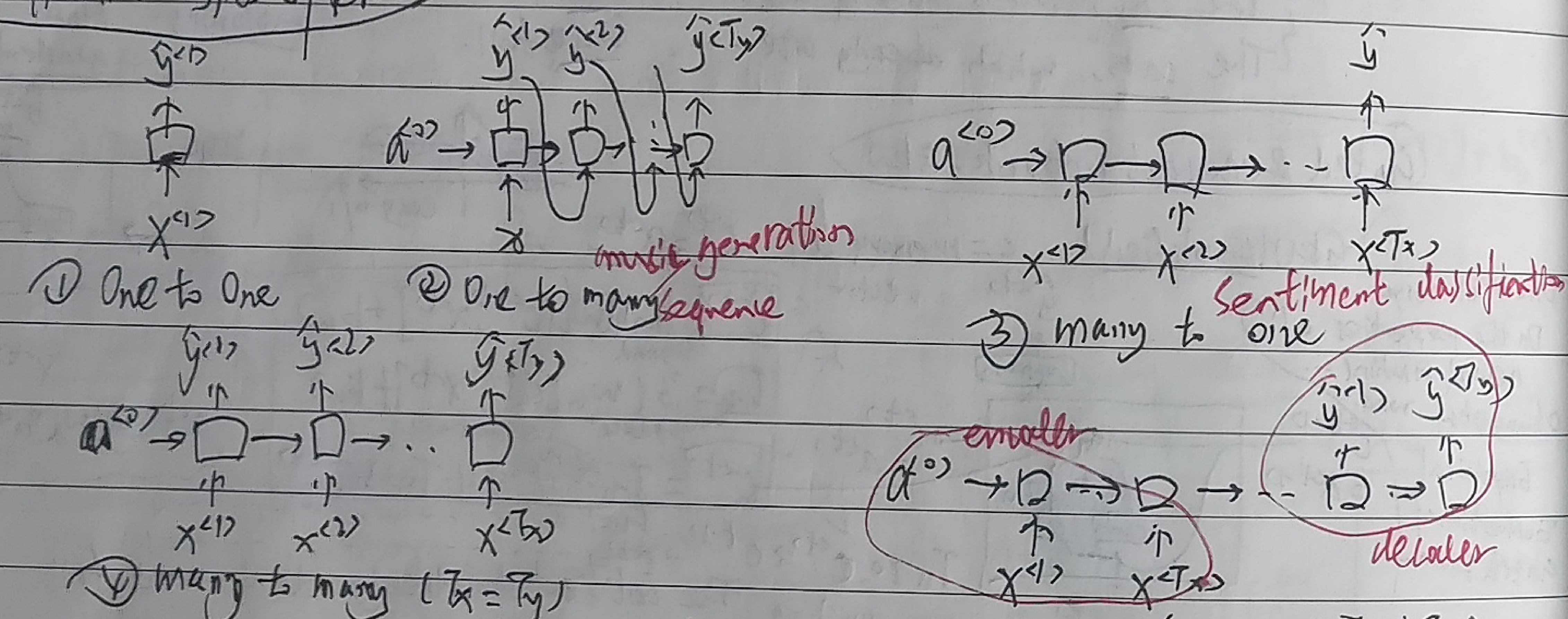
Backpropagation through time



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

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

Different types of RNNs



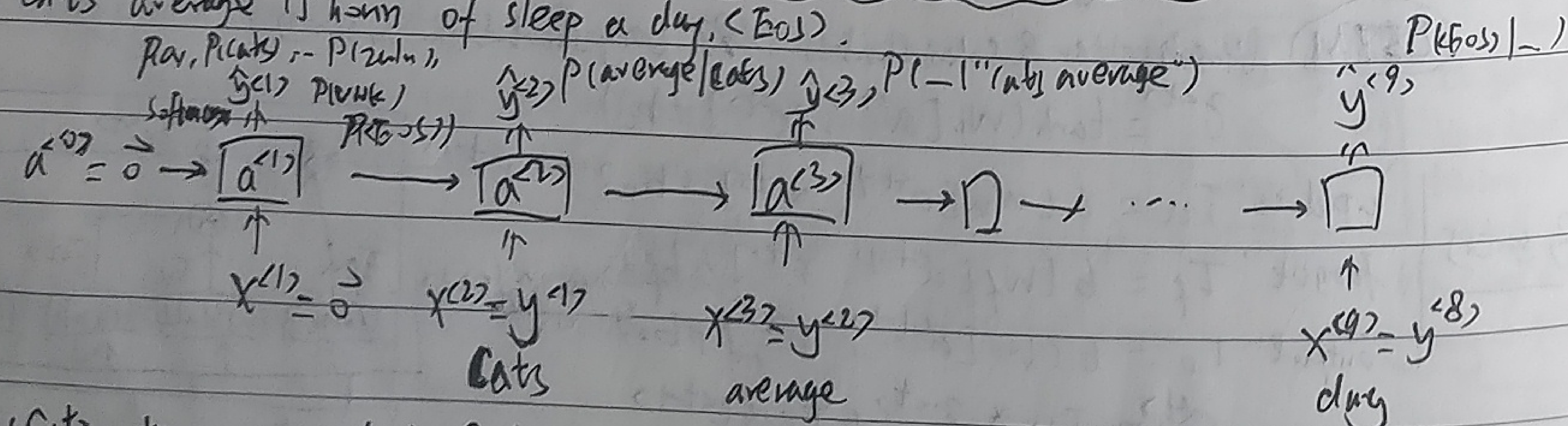
Language model and sequence generation

basic of NLP   P(sentence) = ?

P(The apple and pair salad.) = 3.2×10^{-13}
P(The apple and pear salad) = 5.7×10^{-10} ✓

Language modelling with RNN: training set: a large corpus of english text.

eg. Cats average 15 hours of sleep a day. <EOS>



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

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

2P

C5 **Sequence Model**

W1. Recurrent Neural Networks

Why sequence models?

examples: Speech recognition, music generation, sentiment classification, DNA, Machine translation, video activity recog. Name entity recog.

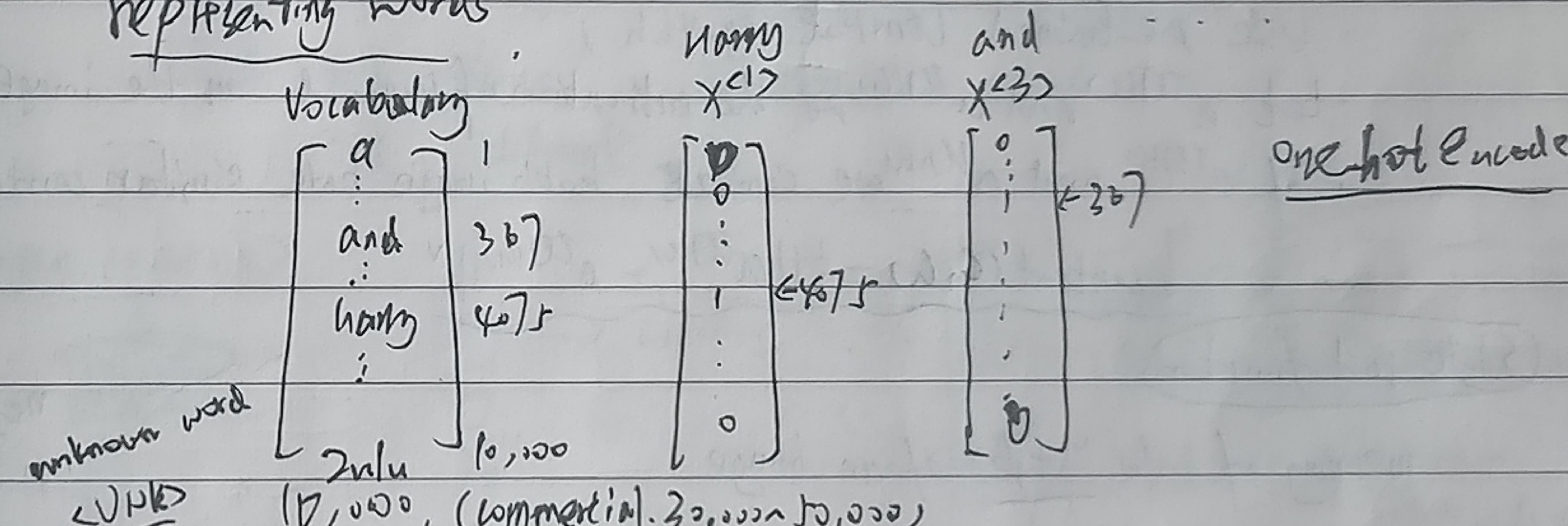
Notation

X: Harry Potter and Hermione Granger invented a new spell.
$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ … $x^{<t>}$ …… $x^{<9>}$. temporal

Y: 1 1 0 1 1 0 0 0 0
$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ … $y^{<9>}$

$x^{(i)<t>}$, $y^{(i)<t>}$. $T_x^{(i)} = T_y^{(i)} = 9$

representing words.



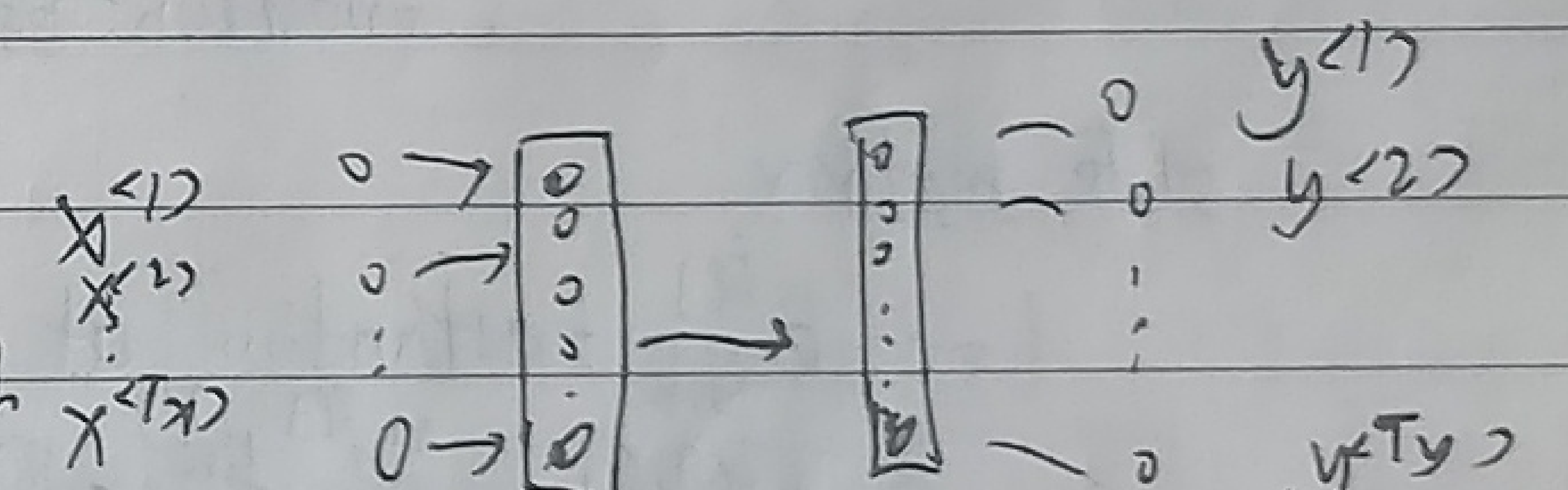
10,000. (commercial. 30,000 ~ 50,000)

Recurrent Neural Network Model

Why not a standard network?

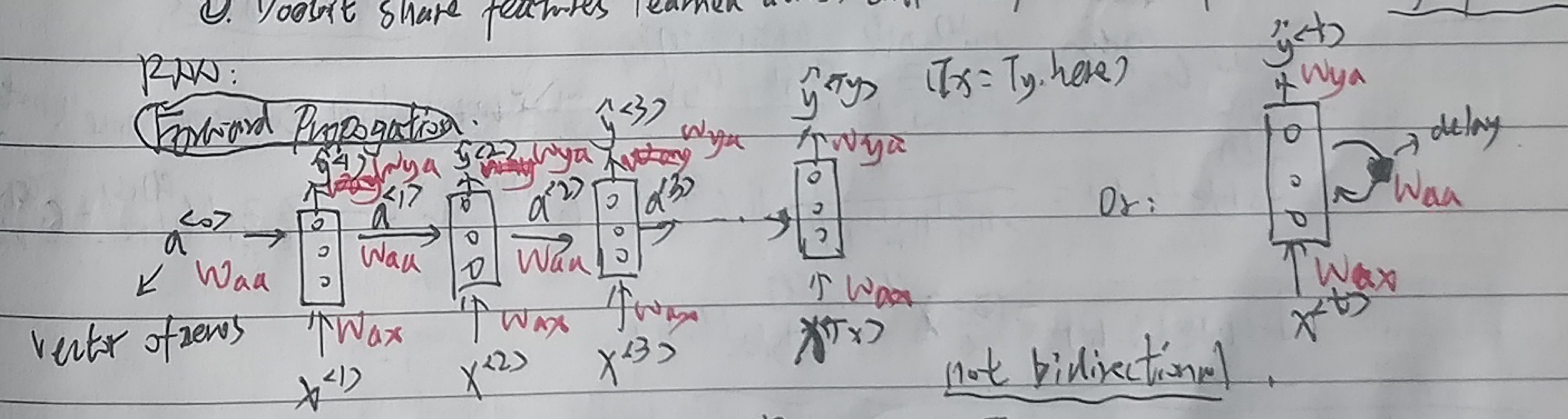
① Inputs/outputs can be different length in different examples.

② Doesn't share features learned across different positions of text, huge parameters.



RNN:

Forward Propagation



Not bidirectional.

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

$a^{<1>} = g_1(W_{aa}a^{<0>} + W_{ax}x^{<1>} + b_a)$ ← Tanh/Relu
(100,100) 100 (100,10000) 10000

$\hat{y}^{<1>} = g_2(W_{ya}a^{<1>} + b_y)$ ← Sigmoid / softmax (depending on output)

$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$; → $a^{<t>} = g(W_a[a^{<t-1>}, x^{<t>}] + b_a)$

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

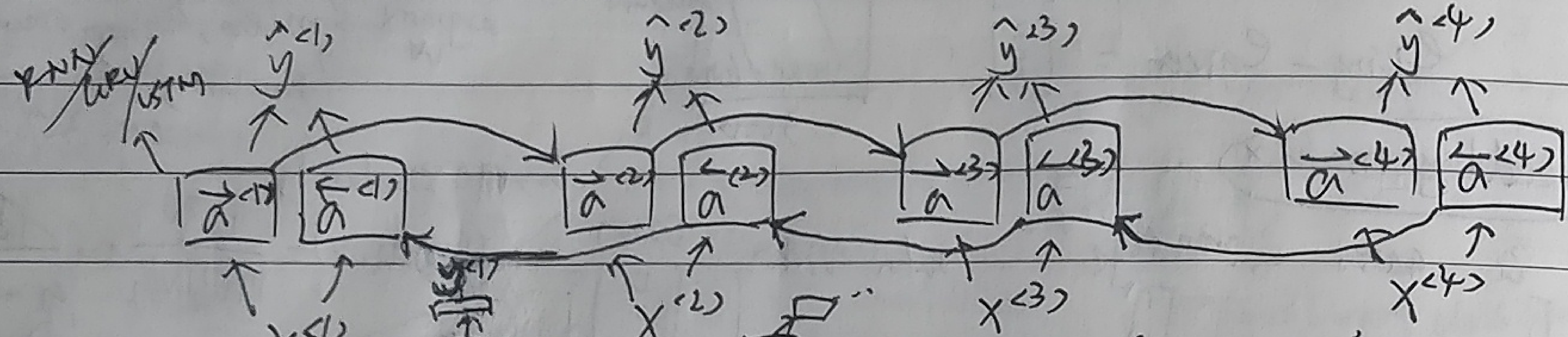
$W_a = [W_{aa} \vdots W_{ax}]$; $[a^{<t-1>}, x^{<t>}] = \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix}$

28

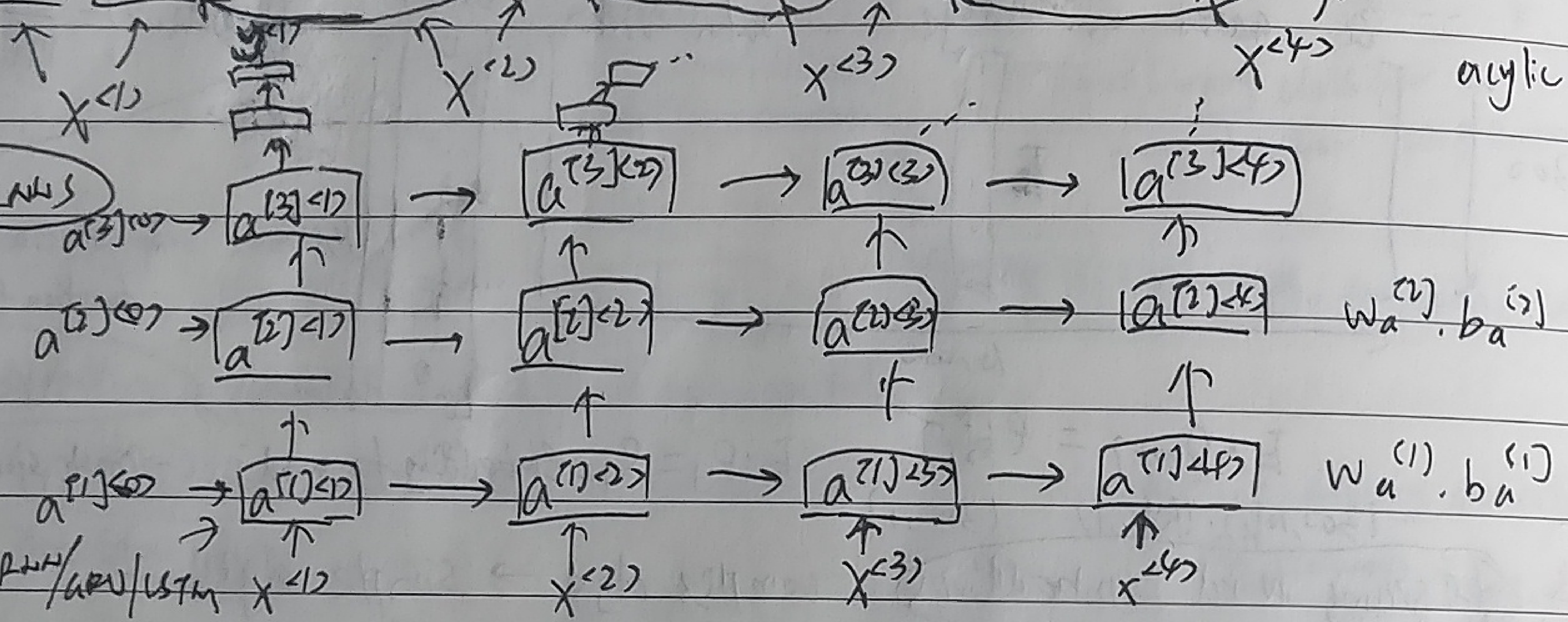
NLP: BRNN with LSTM — try first

Bidirectional network — BRNN

$$\hat{y}^{<t>} = g(W_y[\overrightarrow{a}^{<t>}, \overleftarrow{a}^{<t>}] + b_y)$$



Deep RNNs



RNN: 3 layer is deep enough. (with time)

eg: $a^{[2]<3>} = g(W_a^{[2]}[a^{[2]<2>}, a^{[1]<3>}] + b_a^{[2]})$

W2: NLP and Word Embeddings

Word representation

Intro to word Representation

$V = [a, aaron, \dots, zulu, UNK]$ $|V| = 10{,}000$

one-hot representation

eg: Apple (456) Orange (6257)

O_{456} O_{6257}

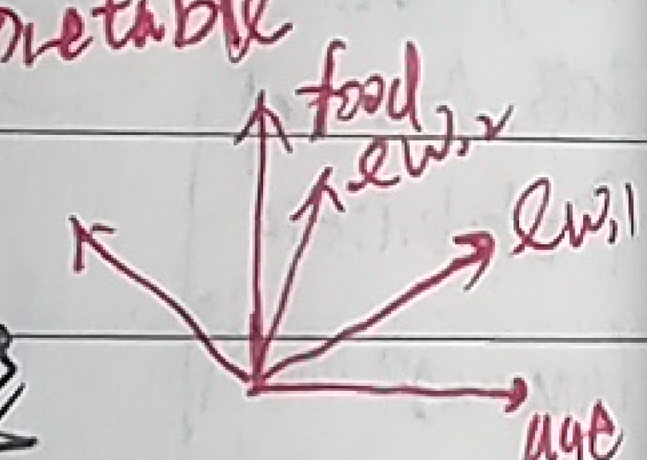
I want a glass of orange ____

.... apple ____

generalize better →

Apple, orange are similar.

individuals may not interpretable



Embedding vectors

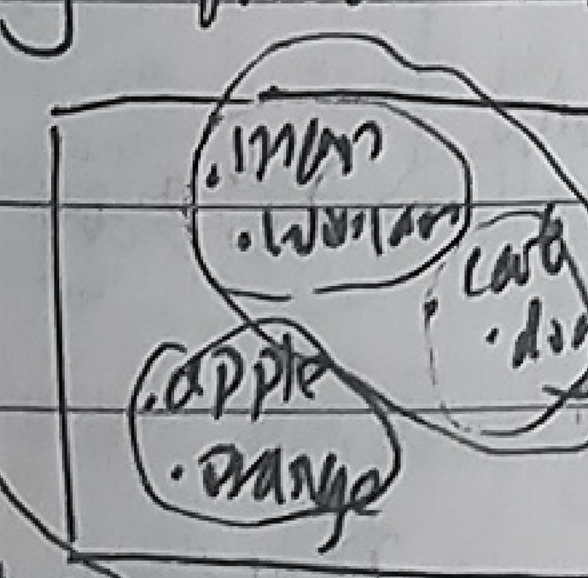
$|D| = 300$

	Apple	Orange	Man	Woman
Gender	0.00	0.01	-1	1
Age	0.03	0.00	0.03	0.01
Food	0.96	0.97	0.01	0.00
:				
size				
cost				

e_{456} e_{6257} ...

Using word embeddings

Visualizing word embedding



t-SNE (non-linear), Van der Maaten, Hinton, 2008. Visualizing data using t-SNE

Transfer learning and word embeddings

A ① Learn word embeddings from large text corpus. (1~100B words) (large data)

(Or download pre-trained embedding online.)

B ② Transfer embedding to new task with smaller training set. (100k words) (less data)

(10000 → 300D)

③ (optional) Continue to finetune the word embeddings with new data

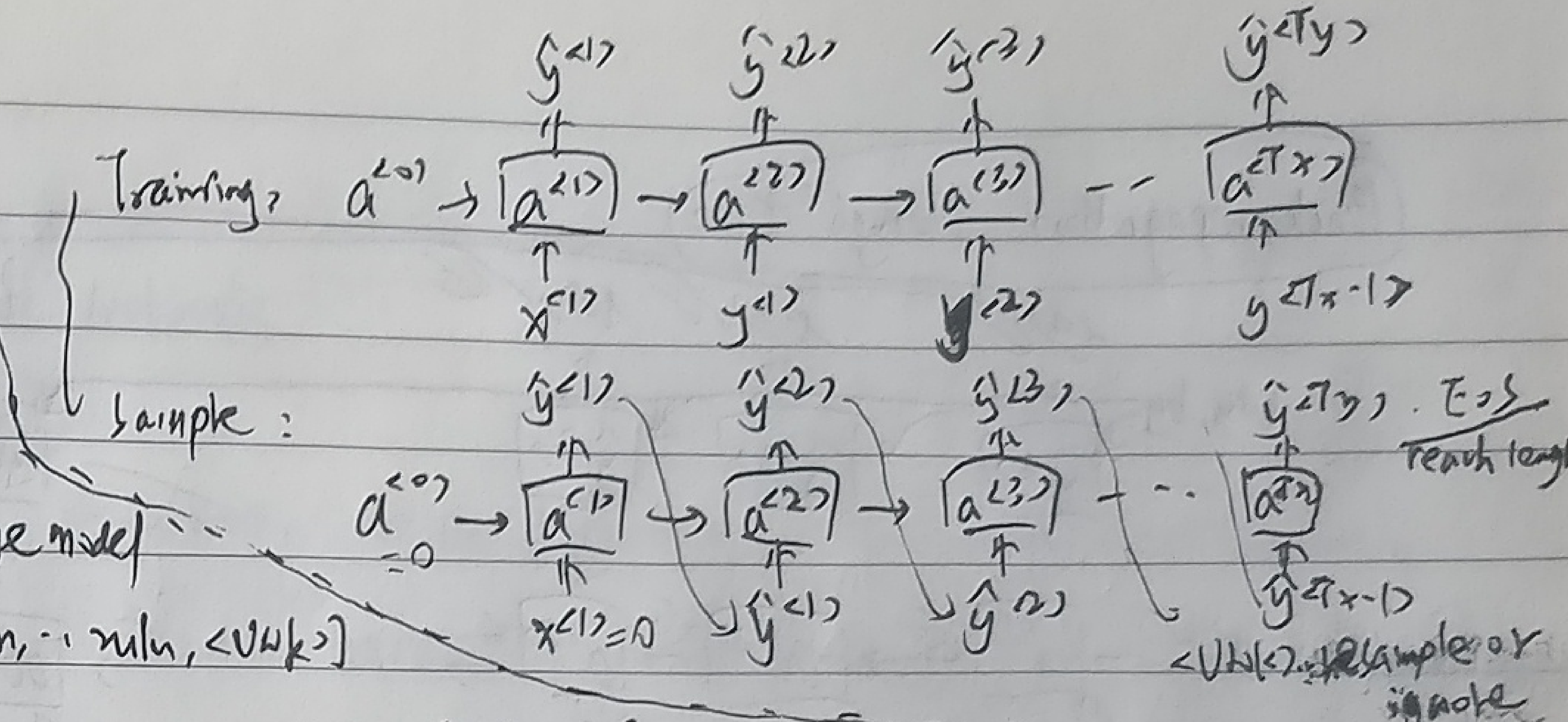
^ if new data is big enough.

applications: name entity recognition, text summarization, co-reference resolution, parsing.

less useful for language model, machine translation for its lots of data.

& face encoding = embedding

Sampling novel sequence



Character-level language model

→ Vocabulary = [a, aaron, ..., zulu, <UNK>]

→ Vocabulary = [a, b, .., z, ␣, ., ,, ;, 0, ..., 9, A, ..., Z]. Cats average...

↳ pros: no <UNK>

cons: too long, more computation, less accuracy (long range dependencies affect). but some special case use (many UNK / more specific vocabulary).

Vanishing gradients with RNNs

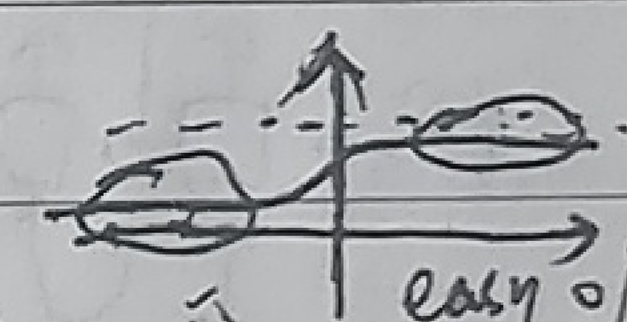
{ The cat, which already ate, ..., was full. (hard to capture)

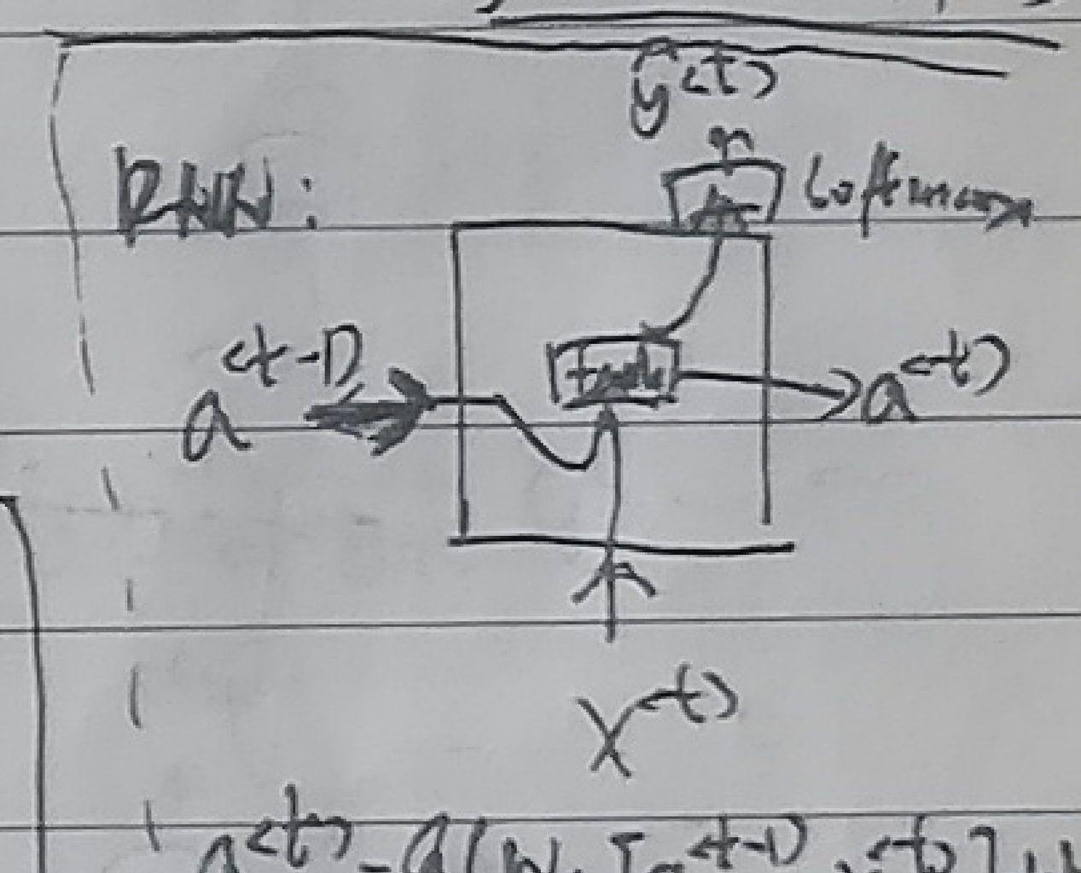
{ The cats, which already ate, ..., were full.

Vanishing → more often in RNN

exploding. NaN. gradient clipping

Gated Recurrent Network, GRU

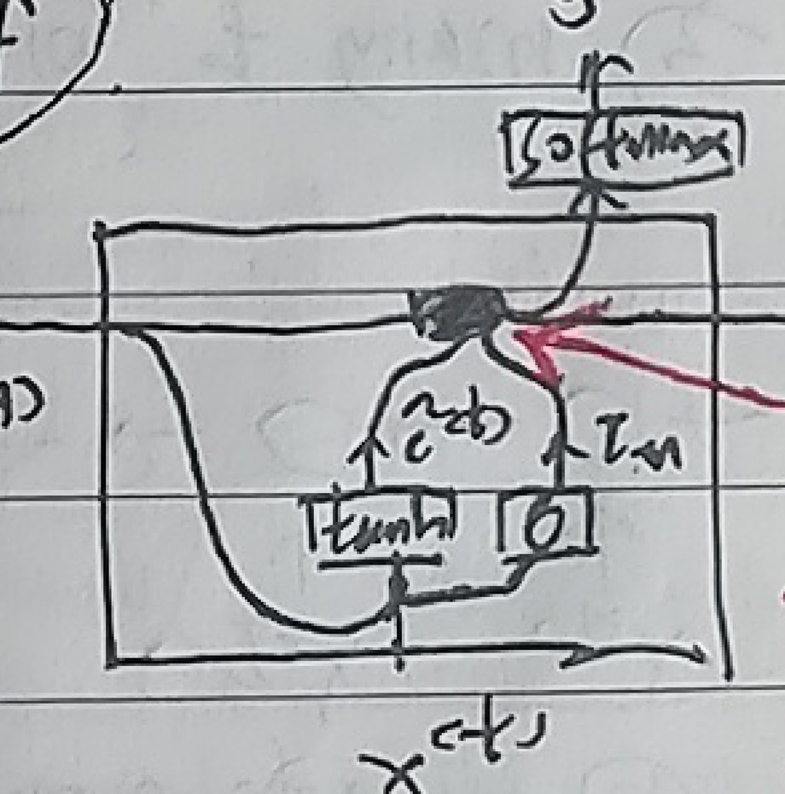




$a^{<t>} = g(W_a[a^{<t-1>}, x^{<t>}] + b_a)$

GRU (simplified), c = memory cell $c^{<t>} = a^{<t>}$

Cho et al. 2014 On the properties of neural machine... Chung et al. 2014 Empirical Evaluation of Gated RNN...



vector

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

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

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

$\Gamma_u = 1$ ... $\Gamma_u = 0$ ...

The cat, which already ate ..., was full.

$\Gamma_u \approx 0$, $c^{<t>} \approx c^{<t-1>}$ no vanishing

Full GRU: (relevance)

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

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

r $\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$

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

{ GRU: simpler, easy to bigger network, run faster (2 gates)

{ LSTM: more powerful and flexible (3 gates)

LSTM

(Hochreiter & Schmidhuber 1997. Long-short-term memory)

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

3 gates:

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

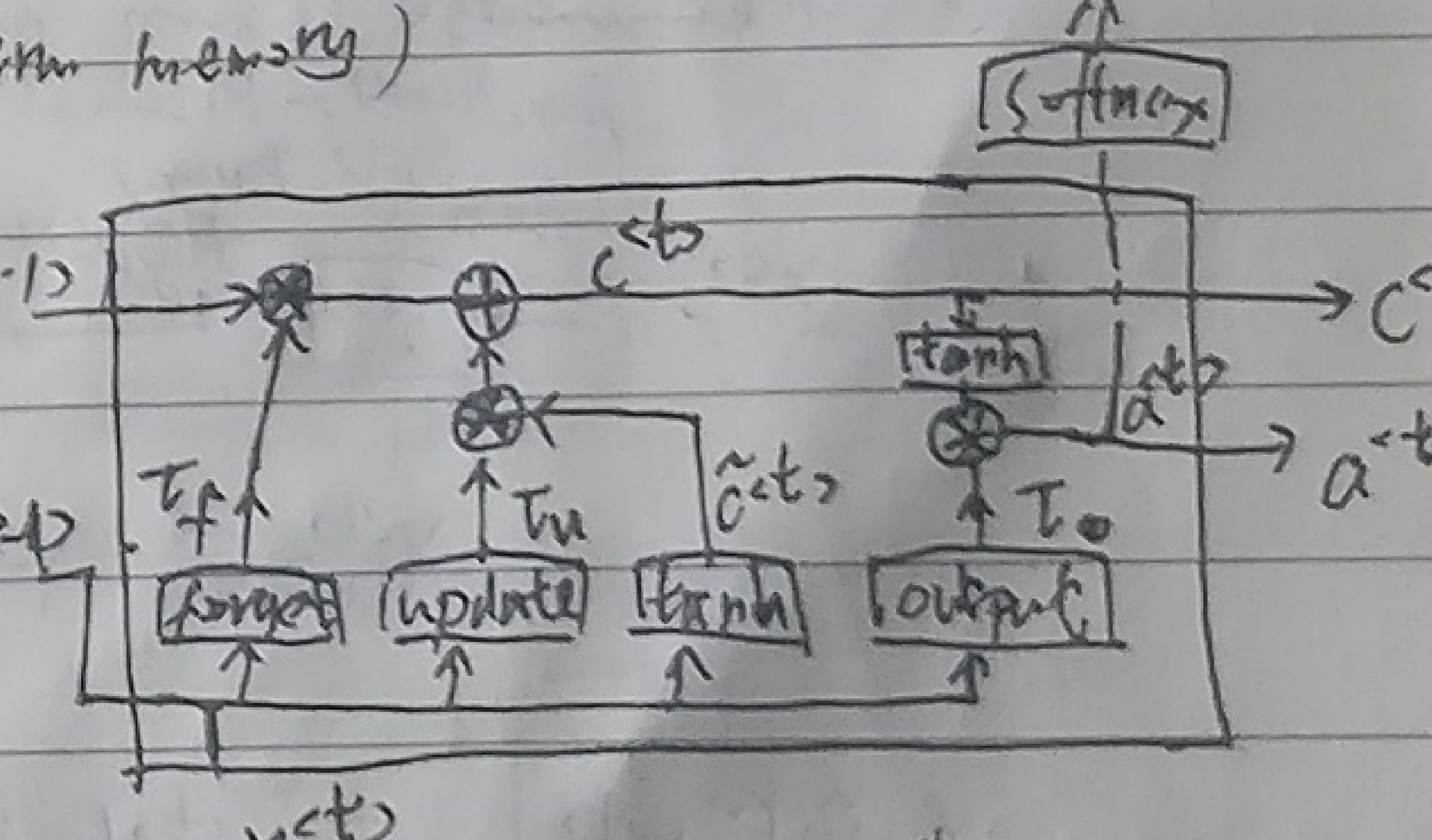
"forget" $\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$

"output" $\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$

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

$a^{<t>} = \Gamma_o * \tanh(c^{<t>})$

$c^{<t-1>}$ peephole connection



Chris Ola, Understanding LSTM Network

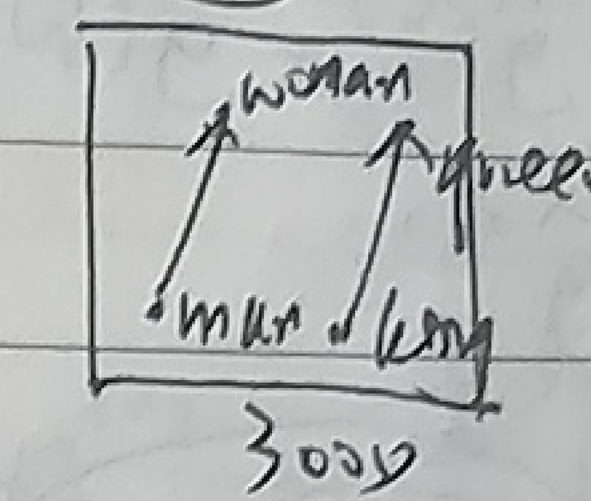
Properties of word embeddings

analogy reasoning

Mikolov et. al 2013. Linguistic regularities in ...

$e_{man} - e_{woman} = [\ ]$

$e_{king} - e_{queen} = [\ ]$



Find word w:

argmax sim$(e_w, e_{king} - e_{man} + e_{woman})$ — similarity

acc: 30%–75%

Cosine similarity

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

Embedding matrix

a aaron ... orange (6257) ... zulu <UNK>

300 × 10,000 matrix E; one-hot vector O_{6257} (10,000)

In practice, use specialized function to look up an embedding

keras: Embedding

$E \cdot O_{6257} = e_{6257}$

(300, 10k) · (10k, 1) (300, 1)

$E \cdot O_j = e_j$ = embedding for word j → much slower, large multiply

Learning word embeddings

complex algo → simple algos.

Bengio et al. 2003. A neural probabilistic language model

Neural language model. window

I want a glass of orange __ (to go along with my cereal)

4343 9665 1 3852 6163 6257

I	O_{4343}	→ E	→ e_{4343}
want	O_{9665}	→ E	→ e_{9665}
a	O_1	→ E	→ e_1
glass	O_{3852}	→ E	→ e_{3852}
of	O_{6163}	→ E	→ e_{6163}
orange	O_{6257}	→ E	→ e_{6257}

→ softmax 10,000; $W^{[1]}, b^{[1]}$; $W^{[2]}, b^{[2]}$

300×6 ← 1200 (fixed).

Other context/target pairs.

context: Last 4 words.

4 words left & right.

Last 1 word

nearby 1 word.

Word2Vec

Mikolov et al. 2013. Efficient estimation of word representations

→ skip-grams model.

eg: context c "orange" (6257) target t "juice" 4834

context orange — target juice/glass/my...

$O_c \to E \to e_c \to O$ softmax → $\hat{y}$

θ_t: parameter associate with output t

softmax: $p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$

$L(\hat{y}, y) = -\sum_{i=1}^{10,000} y_i \log \hat{y}_i$ $y = [0 \ldots 1 \ldots 0]$ ← 4834

→ CBOW model. c t c

Problems with softmax classification. computation

$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$

hierarchical softmax classifier

tree. log|V|

in practice. not balanced tree

common words. upper

How to sample the context c?

common → a, the, of, to ...

not common → apple, orange ...

P(c) not uniformly.

Negative sampling

more effient than skip-gram. (10,000 dim softmax)

Mikolov et al. 2013. Distributed representations of words and ...

context	word	target?
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0
c	t	y

} k negative samples

k: 5–20 (small dataset)

: 2–5 (large dataset)

logistic binary

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

$O_{6257} \to E \to e_{6257}$ → juice? ... king ...

10,000 binary classifier

every iteration: k+1 examples

selecting negative examples. $P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{10,000} f(w_j)^{3/4}}$ between $\frac{1}{|V|}$ (uniform), empirical distribution

GloVe word vectors

(Global Vector for word representation)

Simple

Pennington et al. 2014. GloVe: Global vectors for word representation.

X_{ij} = # times word i appears in context of j. $X_{ij} = X_{ji}$ if defining context/target pairs

X_{ct} (how related) eg. ±10 words of each other.

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

θ_i, e_j are symmetric

weighting term: $f(X_{ij}) = 0$ if $X_{ij} = 0$. $0 \log 0 = 0$

θ_i, e_j randomly initialized at beginning; $e_w^{(final)} = \frac{e_w + \theta_w}{2}$

frequent word: this, is, a, of (not too much weight)

unfrequent word: durian ... (not too little weight)

$\theta_i^T e_j = (A\theta_i)^T (A^{-T} e_j)$

can't guarantee the axis used to represent the features will be well-aligned with easily humanly interpretable axis

Sentiment classification

applications using word embeddings

challenge: not huge dataset

simple model

The $O_{8928} \to E \to e_{8928}$

dessert $O_{2468} \to E \to e_{2468}$ → Avg → O → $\hat{y}$ softmax (1-5 Star)

is $O_{4694} \to E \to e_{4694}$

excellent $O_{3180} \to E \to e_{3180}$

regardless of length; no order!

RNN for sentiment classification

$a^{<0>} \to a^{<1>} \to a^{<2>} \to a^{<3>} \to \cdots \to a^{<10>} \to$ softmax $\hat{y}$

completely lacking in good ... ambience

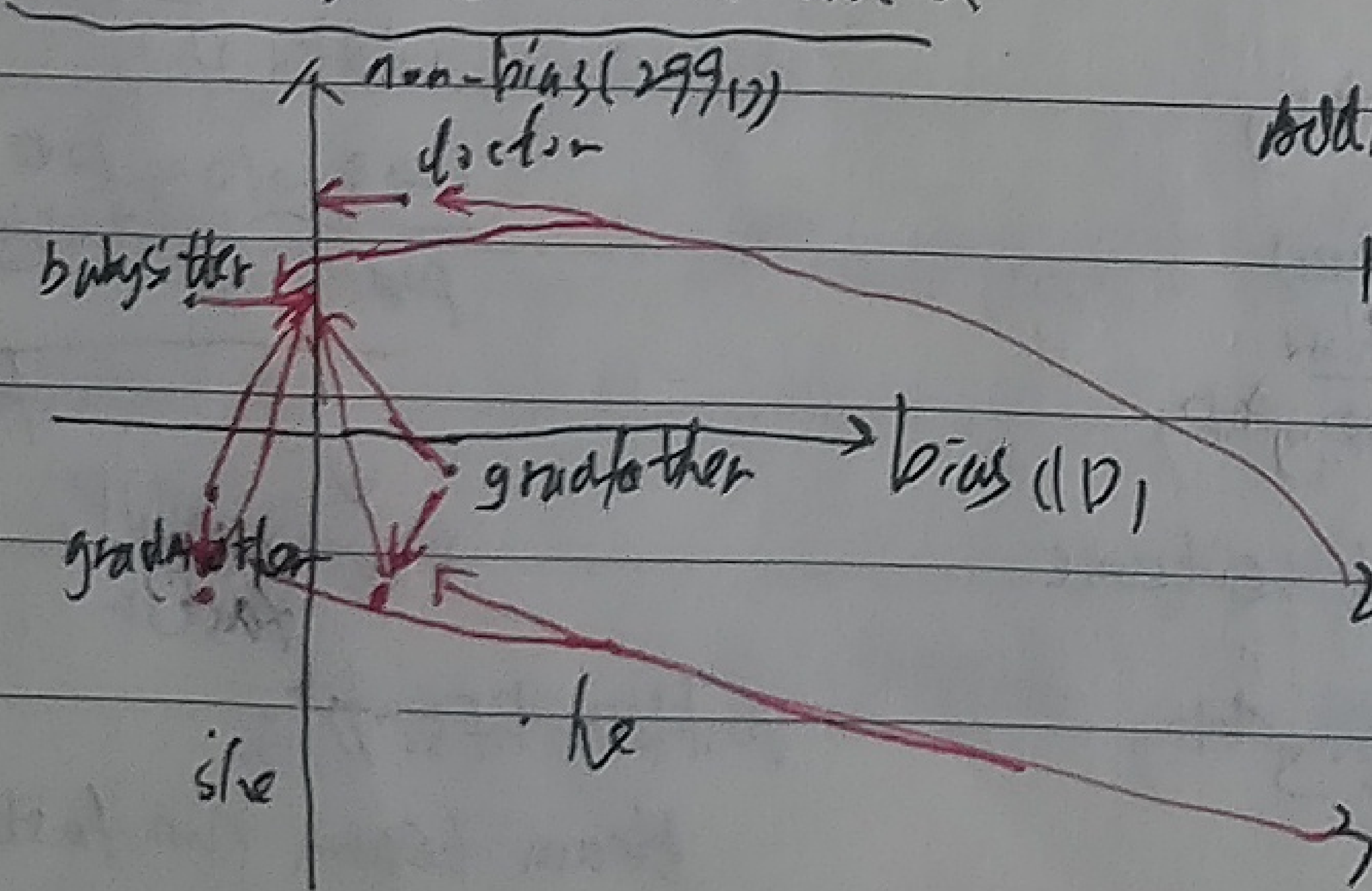
absent of → fine generalize

Debiasing word embeddings

bias: gender, ethnicity, sexual

Bolukbasi et al. 2016 Man is to computer programmer as woman ...

Father: Doctor as Mother: Nurse ×



gender for example

Addressing bias in word embeddings.

1. Identity bias direction.

$e_{she} - e_{he}$, $e_{female} - e_{male}$ } average

2. Neutralize: For every word that is not definitional, project to get rid of bias. (classifier)

3. Equalize pairs.

grandmother – grandfather

she – he

Bleu score: how well of Machine translation
[Papineni et al. 2002. Bleu: A method for automatic ...]

French ...
Human: Jane visits Africa in september (y^*)
Algo: Jane visited Africa last september. ($\hat{y}$)

Error analysis on Beam search.

Case 1. $P(y^*|x) > P(\hat{y}|x)$
Beam search is at fault.

Case 2. $P(y^*|x) \le P(\hat{y}|x)$
RNN is at fault. (y^* is better than $\hat{y}$ actually)

do List:

Human	Algo	$P(y^*\|x)$	$P(\hat{y}\|x)$	at fault?
$x^{(1)}$				B
$x^{(2)}$				R
⋮				R
$x^{(m)}$				⋮

B: ↑
R: regularization, more data, network...

Attention model intuition

Bahdanau et al. 2014. Neural machine translation by jointly ...

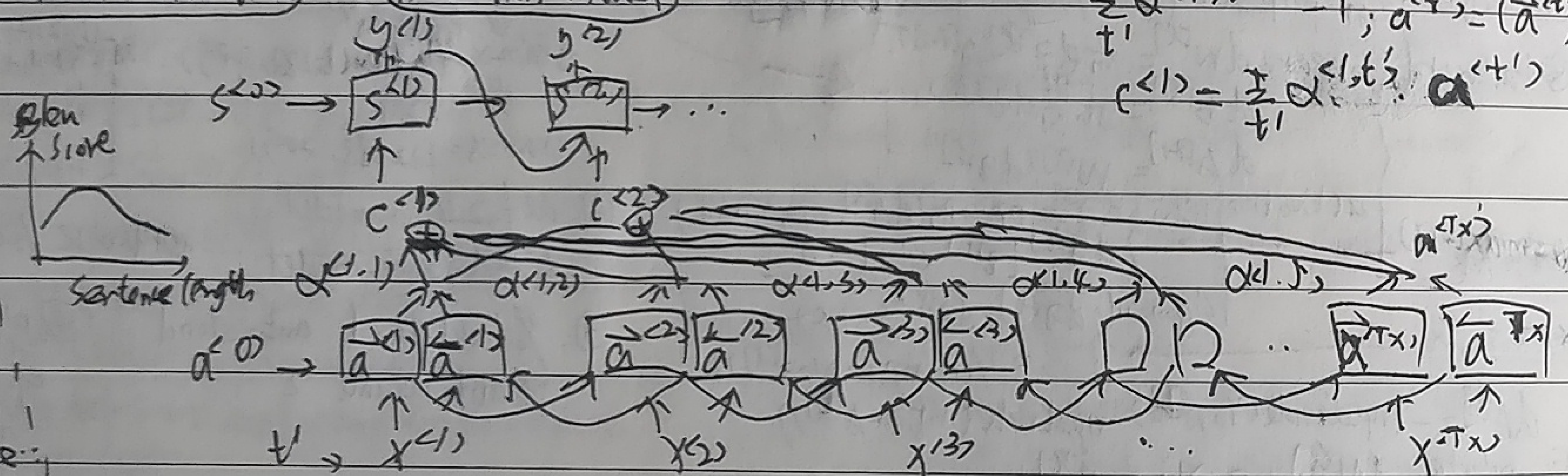
Xu et al. 2015. Show, attend and tell: Neural image ...

Attention model

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

$\sum_{t'} \alpha^{<1,t'>} = 1$, $a^{<t'>} = (\overrightarrow{a}^{<t'>}, \overleftarrow{a}^{<t'>})$

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



$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x} \exp(e^{<t,t'>})}$$

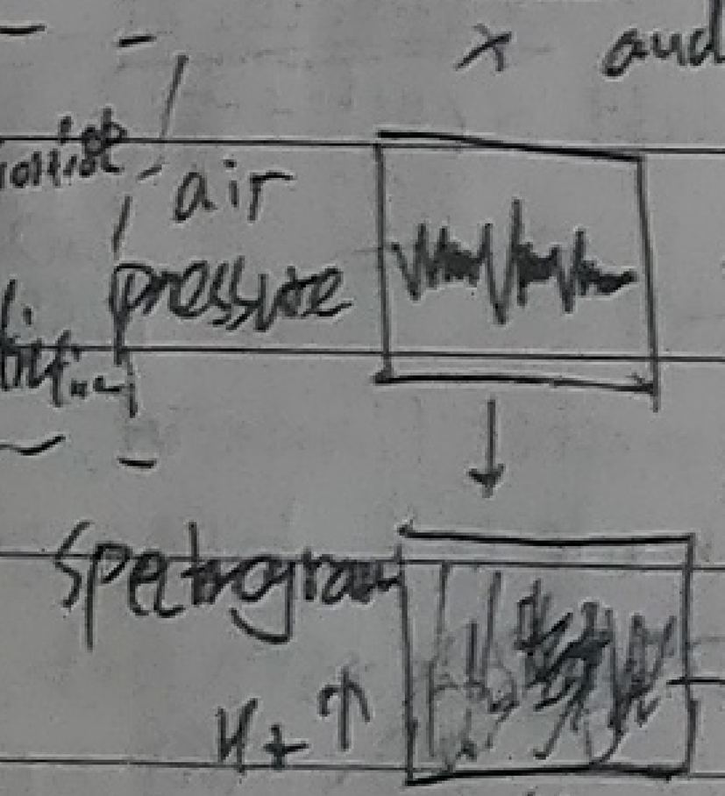
$s^{<t-1>}$, $a^{<t'>}$ → NN → $e^{<t,t'>}$ — automatic learn

drawback: computation, $O(n^3)$, $n_{\alpha^{<t,t'>}} = T_x \cdot T_y$ but sentence is not long

Speech recognition

Graves et al. 2006. Connectionist Temporal classification ...

x audio clip → y transcript



air pressure → "the quick brown fox"
pre-processing
Spectrogram — color: energy — time

/phonemes: no longer used

→ 3000h data, 100,000h (commercial)

① attention model
② CTC cost for speech recognition (Connectionist temporal classification)

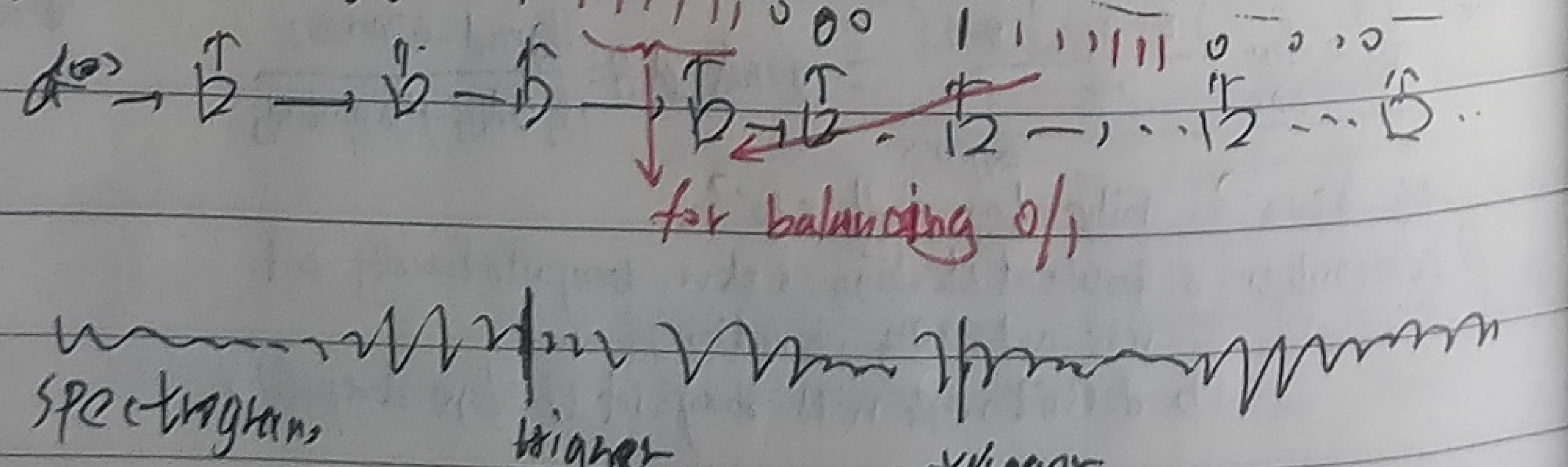
$a^{<0>}$ → $\hat{y}^{<1>}$ → $\hat{y}^{<2>}$ ... $\hat{y}^{<3000>}$; $x^{<1>}$ $x^{<2>}$ ... $x^{<3000>}$, 3000 = time × frequency

predict: ttt-h-eee---u---qqq...
"blank" "space"

Basic rule: collapse repeated characters not separated by "blank"

Trigger word detection

Amazon Echo: Alexa
Baidu DuerOS: 小度小度
Apple Siri: Hey Siri
Google Home: Okay Google



output: 0 0 0 0 111111 0 0 0 ... 1 1 1 1 1 1 1 1 0 0 0
for balancing 0/1
spectrogram, trigger, trigger

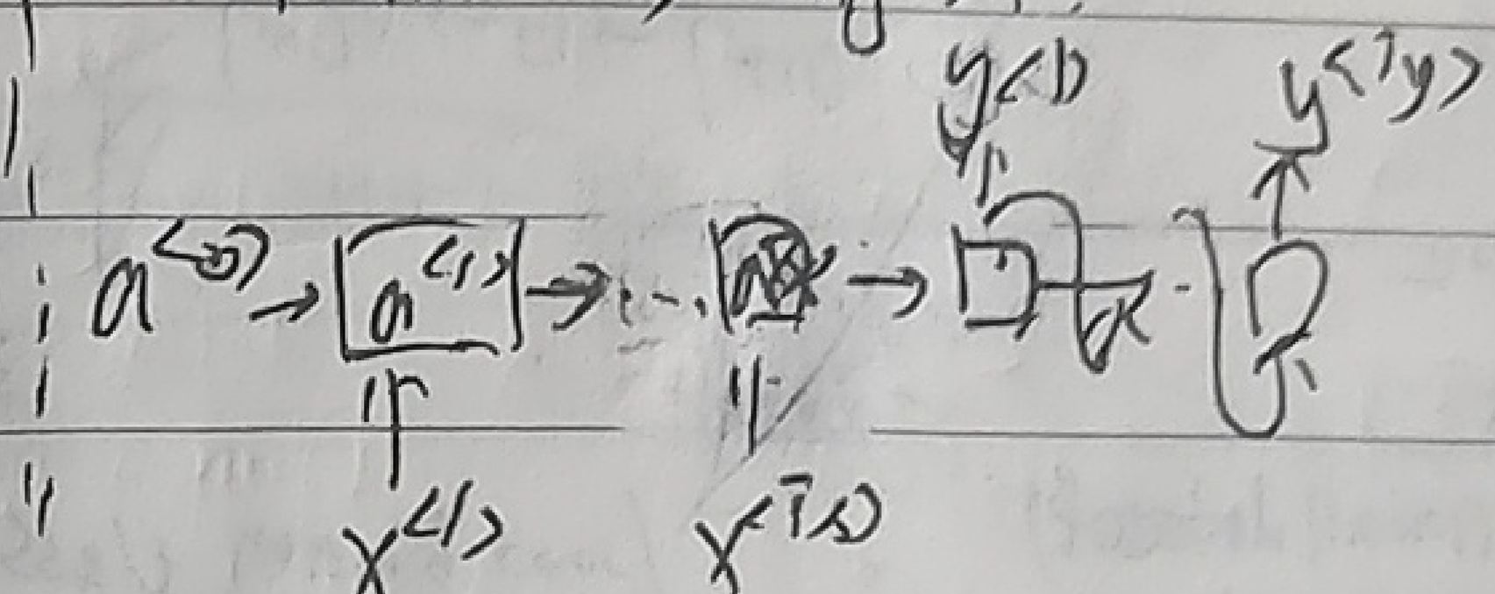
W3: Sequence to sequence models

various sequence to sequence architectures

basic models

Sequence to sequence model: French → English

Sutskever et al. 2014. Sequence to sequence learning ...
Cho et al. 2014. Learning phrase representations ...



encoder, decoder

Image caption: "a cat sitting on a chair"

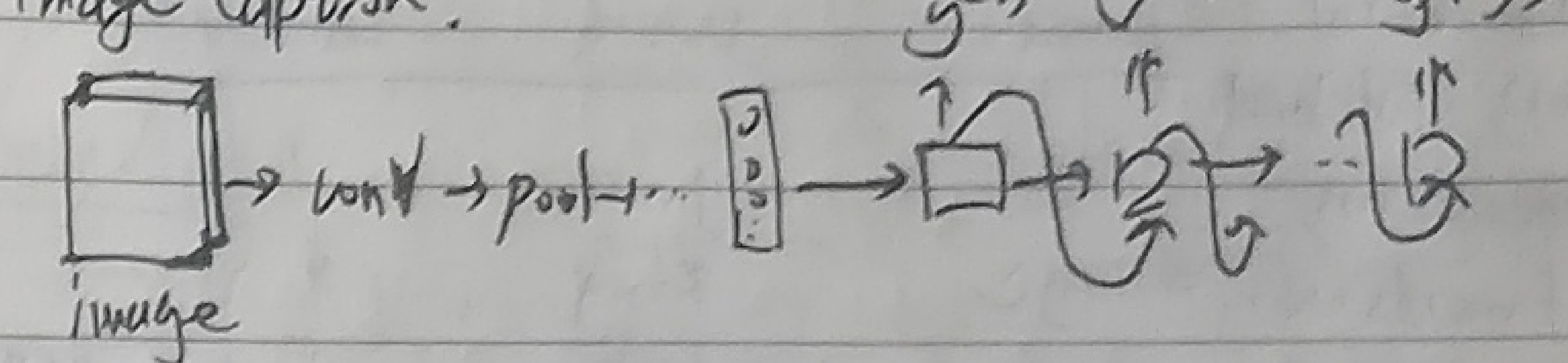


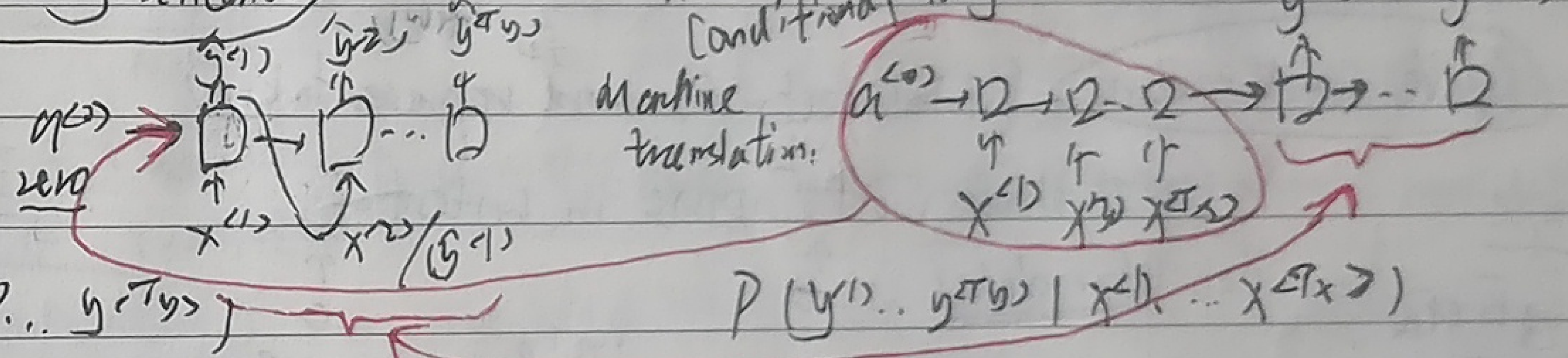
image → conv → pool ...

Mao et al. 2014. Deep captioning with multimodal ...
Vinyals et al. 2014. Show and tell: Neural ...
Karpathy et al. 2015. Deep visual-semantic alignments for ...

Picking the most likely sentence

Language model: $P(y^{<1>}, y^{<2>}, \dots, y^{<T_y>})$

Machine translation (conditional language model): $P(y^{<1>}, \dots, y^{<T_y>} | x^{<1>}, \dots, x^{<T_x>})$



Beam search

Beam width = 3 (B=1 → why not greedy search?)

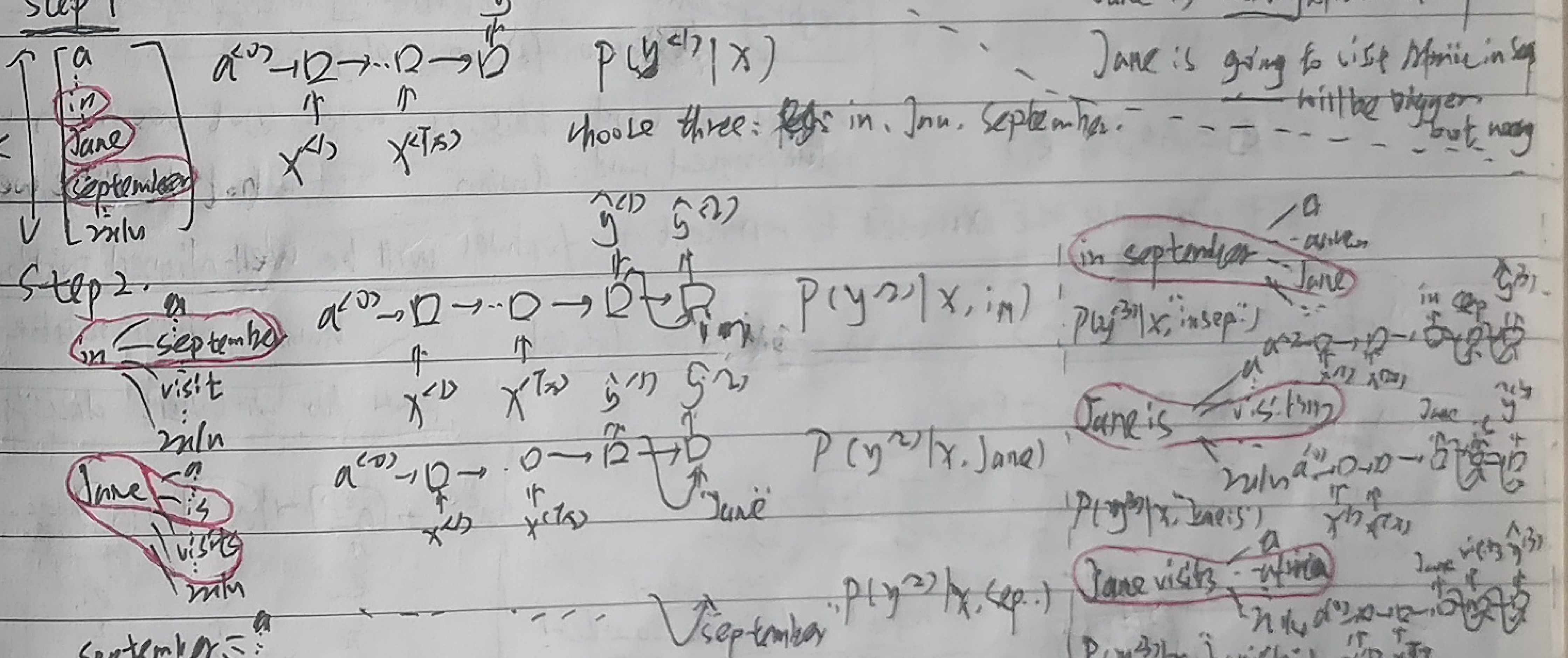
$T_y = 10$, 10000^{10}

Jane is visiting Africa in Sep.
Jane is going to visit Africa in Sep. — will be bigger but wrong

Step 1
a, in, Jane, september, zulu — 10k
$P(y^{<1>}|x)$
choose three: in, Jane, September.

Step 2
$P(y^{<2>}|x, \text{in})$
$P(y^{<2>}|x, \text{Jane})$
$P(y^{<2>}|x, \text{Sep.})$

in september; $P(y^{<3>}|x, \text{in sep.})$
Jane is visiting; $P(y^{<3>}|x, \text{Jane is})$
Jane visits Africa; $P(y^{<3>}|x, \text{Jane visits})$



Refinement to Beam search

Length Normalization

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

↓ log (multiply little numbers) numerical underflow

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

tend to short sentence
↓

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

$\alpha = 0.7$

Beam search discussion
Beam width B?
1 → 3 → 10 practice, 100 too big, 1000, 3000 do research.
performance lift less
worse result, faster — better result, slower
Unlike BFS, DFS.
Beam search run faster but not guarantee to find the exact maximum

tf.Variable 和 tf.get_variable → 面向创建新对象 / 先判断是否已有（共享变量，其余二者同）

提示命名冲突，自行处理 / 命名冲突报错

fig, axarr = plt.subplots(1, 2)
axarr[0].set_title(...)
axarr[0].imshow(x[0, :, :, 0])
axarr[1] ...

a2: optimization method 

① gradient descent: $W^{[l]} -= \alpha \cdot dW^{[l]}$, $b^{[l]} -= \alpha \cdot db^{[l]}$.

SGD, mini-batch GD. 79.7%

② a → shuffle(X, Y); b → Partition (last batch!)

③ momentum: 79.7%
$V_{dW^{[l]}} = \beta V_{dW^{[l]}} + (1-\beta) dW^{[l]}$
$W^{[l]} = W^{[l]} - \alpha \cdot V_{dW^{[l]}}$
$V_{db^{[l]}} = \beta V_{db^{[l]}} + (1-\beta) db^{[l]}$
$b^{[l]} = b^{[l]} - \alpha \cdot V_{db^{[l]}}$

initialize: $V_{dW^{[l]}}$ = np.zeros, shape = $W^{[l]}$
$V_{db^{[l]}}$ = np.zeros, shape = $b^{[l]}$

④ velocity initialize with 0, several iterations "build up". β↑, smoother update. 0.8 ~ 0.999, 0.9

⑤ Adam: combines RMSprop & momentum. 94%
t: steps of adam
$V_{dW^{[l]}} = \beta_1 V_{dW^{[l]}} + (1-\beta_1) dW^{[l]}$
$V'_{dW^{[l]}} = V_{dW^{[l]}} / (1-\beta_1^t)$
$S_{dW^{[l]}} = \beta_2 S_{dW^{[l]}} + (1-\beta_2)(dW^{[l]})^2$
$S'_{dW^{[l]}} = S_{dW^{[l]}} / (1-\beta_2^t)$
$W^{[l]} = W^{[l]} - \alpha \cdot V'_{dW^{[l]}} / \sqrt{S'_{dW^{[l]}}} + \varepsilon$

initialize: $V_{dW^{[l]}}$, $S_{dW^{[l]}}$: np.zeros, shape = $W^{[l]}$
$V_{db^{[l]}}$, $S_{db^{[l]}}$: np.zeros, shape = $b^{[l]}$

⑥ momentum usually help but small α and simple dataset. adam converge faster.
adam advantage: low memory requirements (> GD, momentum); usually works well with little tuning of hyperparameters (except α)

a3. Tensorflow Tutorial

① steps: a. create Tensors (variables). b. write operations between those Tensors. c. initialize Tensors. d. create a session. e. run the session

② tf.constant, tf.Variable, tf.placeholder, sess.run(, feed_dict=)
tf.nn.sigmoid_cross_entropy_with_logits(logits= , labels=) (softmax)
tf.one_hot(labels, depth, axis=0) [vector ↓, one hot dim ↓], tf.ones().

③ Build NN in Tensorflow.
a. create placeholder for X, Y. (n_x/n_y, none)
b. initialize parameters for $W^{[l]}$, $b^{[l]}$.
$W^{[l]}$ = tf.get_variable([,], initializer = tf.contrib.layers.xavier_initializer())
$b^{[l]}$ = tf.get_variable([, 1], initializer = tf.zeros_initializer())
c. forward. tf.add, tf.matmul, $A^{[l]}$, $Z^{[l]}$, tf.nn.relu(), tf.transpose(Z3), tf.transpose(Y)
d. cost. = tf.reduce_mean(tf.nn.softmax_cross...(logits= , labels=))
e. backward and parameters update. (automatic)
optimizer = tf.train.AdamOptimizer(α =).minimize(cost)
_ , c = sess.run([optimizer, cost], feed_dict={X: , Y: }) (c ↓ cost)

④ remember: a. Two main (object) classes in Tf: Tensors, Operators
b. can execute the graph multiple times
c. Backpropagation, update automatic.

C4. CNN

a1-1: CNN step by step

① zero-pad: np.pad(X, ((0,0), (pad, pad), (pad, pad), (0,0)), 'constant' / edge ...)

② conv-forward: a. Z = np.zeros((m, n_H, n_W, n_C)),
b. A_prev_pad = zero_pad(A_prev).
c. for i in range(m):
for h in range(n_H):
for w in range(n_W):
for c in range(n_C):
vert_start = stride * h
vert_end = vert_start + f
hori_start = stride * w
hori_end = hori_start + f
a_slice_prev = A_prev_pad[i, vs:ve, hs:he, :]
Z[i, h, w, c] = a_slice_prev * W[:, :, :, c] + b[:, :, :, c]

③ forward pool: A = ... prev, for ×4, A[i, h, w, c] = np.max (mean) (a_slice_prev)

④ Backpropagation - CNN: for ×4: dW[:, :, :, c] += a_slice × dZ[i, h, w, c]
da_prev_pad[vs:ve, hs:he, :] += W[:, :, :, c] × dZ[i, h, w, c]
db[:, :, :, c] += dZ[i, h, w, c]

⑤ backpropagation - Pool: mask = (X == np.max(X))

a1-2: CNN: application — SIGNS dataset.

① create placeholder for X, Y: (None, n_H, n_W, n_C), (None, n_y)

② initialize parameters: W = tf.get_variable('', [, , ,], init = ...) (tf. automatic for bias and W in fully connected layers)

③ forward: tf.nn.conv2d(X, W1, ...), tf.nn.relu(Z1), tf.nn.max_pool(A1, ...), tf.contrib.layers.flatten, fully_connected

④ cost: tf.reduce_mean(tf.nn.softmax_cross... (logits, labels))

⑤ model: (①, ②, ③, ④, optimizer, sess(↓)
init, for epoch: minibatch_cost = 0. num_minibatches = int(m / minibatch_size), seed += 1
for mini-batch: function for mini( )
_ , temp_cost = sess.run([optimizer, cost], feed...
minibatch_cost += temp_cost / num_minibatches
correct_pred = tf.equal(tf.argmax(Z3, 1), tf.argmax(Y, 1))
acc = tf.reduce_mean(tf.cast(, "float"))
train_acc = acc.eval({X: X_train, Y: Y_train})
test_acc = acc.eval({X: X_test, Y: Y_test})

C1: Neural Network and DL

a2-1: Python Basic with numpy.

① math.exp → np.exp; real, vector, matrix.
② normalize: np.linalg.norm(x, ord=2, axis=1
③ Broadcasting in numpy. keepdims=True)
④ Vectorize: np.dot(,): matrix multiply.
(efficiency, clarity) np.multiply(,): element-wise.
⑤ time.time() & time.process_time()

a4-1: Build your DNN: step by step. (2 & L-layer)

① initialize: W: np.random.randn(dim[l], dim[l-1]) × 0.01
b: np.zeros((dim[l], 1))

② forward: $Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$
$A^{[l]} = g(Z^{[l]})$; cache((A, W, b), Z)

③ cost: $-1/m$ * np.sum(Y * np.log($A^{[L]}$) + (1-Y) * np.log(1-$A^{[L]}$))

④ backward: linear: $dW^{[l]} = \frac{1}{m} dZ^{[l]} A^{[l-1]T}$
$db^{[l]} = \frac{1}{m} \sum_{i=1}^{m} dZ^{[l](i)}$
$dA^{[l-1]} = W^{[l]T} dZ^{[l]}$
activation: relu: $dZ^{[l]}$ = np.copy($dA^{[l]}$, copy=True)
$dZ^{[l]}[Z^{[l]} <= 0] = 0$
relu = max(0, x)
sigmoid: $dZ^{[l]} = dA^{[l]} \cdot s \cdot (1-s)$
$s = 1/(1+\text{np.exp}(-Z^{[l]}))$
$\sigma(x) = \frac{1}{1+e^{-x}}$
$dA^{[L]}$ = -(np.divide(Y, $A^{[L]}$) - np.divide(1-Y, 1-$A^{[L]}$))

⑤ update: $W^{[l]} -= \alpha \cdot dW^{[l]}$
$b^{[l]} -= \alpha \cdot db^{[l]}$

a2-2: logistic regression with neural network mindset.

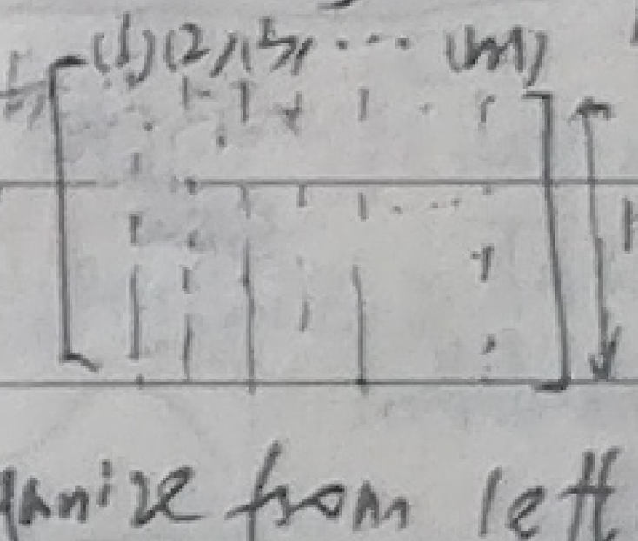
① np.squeeze, scipy.ndimage.imread, scipy.misc.imresize, plt.imshow()
② architecture of learning: a. initialize parameters; b. calculate cost and gradient; c. optimization.
③ remember: a. Preprocess is important; b. ②; c. learning rate

a3. Planar data classification with one hidden layer 

① baseline: lr. (0.47, not linearly separable)
② methodology to build NN: a. define structure; b. initialize parameters; c. loop: 1> forward; 2> loss. 3> backward; 4> update para.
③ tune # units of hidden. large model, fit better, until overfit.

a4-2. DNN for Image classification: Application.

① train_x_orig: (209, 64, 64, 3), train_y_orig: (1, 209)
↓ flatten
train_x_flatten (12288, 209).
↓ /255.
train_x: (12288, 209)



② def 2_layer_model. def L_layer_model. organize from left a4-1

③ %load_ext autoreload
%autoreload 2

④ plt.rcParams[' '] = ... eg. figure.figsize, image.{interpolation, cmap

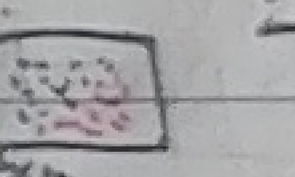
C2. Improving DNN

a1-1: initialization 

① benefits: a. speed up convergence; b. lower error.
② zero initialize, acc: 0.5 (all 0), "break symmetry".
random initia., acc (train): 0.83, cost starts high (big weight, sigmoid → 0/1, wrong example high) (x10)
every unit learn same (= n^[l] = 1)
train longer, better result, but slow optimization.
He initialization: acc: 0.99.
$\times \sqrt{2/\text{dim}^{[l-1]}}$ (Xavier initializ...: $\times \sqrt{1/\text{dim}^{[l-1]}}$)
③ remember: don't initialize too large; He initia... work well with Relu

a1-3. gradient checking

① proof that backward propagation is wright.
② $\frac{\partial J}{\partial \theta} = \lim_{\varepsilon \to 0} \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}$ (approx) $\stackrel{?}{=}$ grad, diff = $\frac{\|grad - approx\|_2}{\|grad\|_2 + \|approx\|_2}$
eg $\varepsilon = 10^{-7}$, whether diff $< 10^{-7}$?
③ remember: a. gradient checking is slow. computationally cost. use not every iteration.
b. doesn't work with dropout (before dropout)

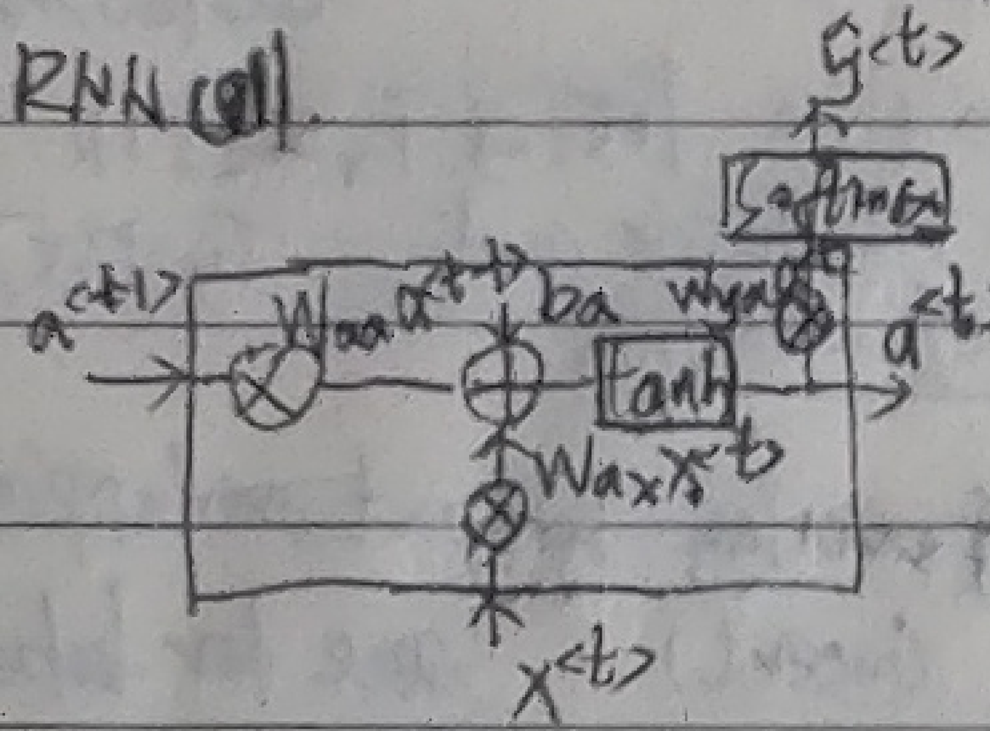
a1-2: Regularization — football field. overfit 

① non-regularized, overfit. test_acc: 0.915
L2-regularization: cost = cost + $\frac{1}{m}\frac{\lambda}{2}\sum_l\sum_k\sum_j W_{kj}^{[l]2}$ (0.93) np.sum(np.square($W^{[l]}$))
$dW^{[l]} = dW^{[l]} + \frac{\lambda}{m} W^{[l]}$
λ↑, boundary smoother (lower W, slow change)
Dropout: steps. a. create $D^{[l]}$ shape = $A^{[l]}$ (random)
0.95 forward. b. $D^{[l]}$ = ($D^{[l]}$ < keep_prob). True/False
c. $A^{[l]} = A^{[l]} * D^{[l]}$ (shut down)
d. $A^{[l]} = A^{[l]}$ / keep_prob. (scale)
backward: a: $dA^{[l]} = dA^{[l]} * D^{[l]}$
b. $dA^{[l]} = dA^{[l]}$ / keep_prob
② remember: dropout not in test. only in train.

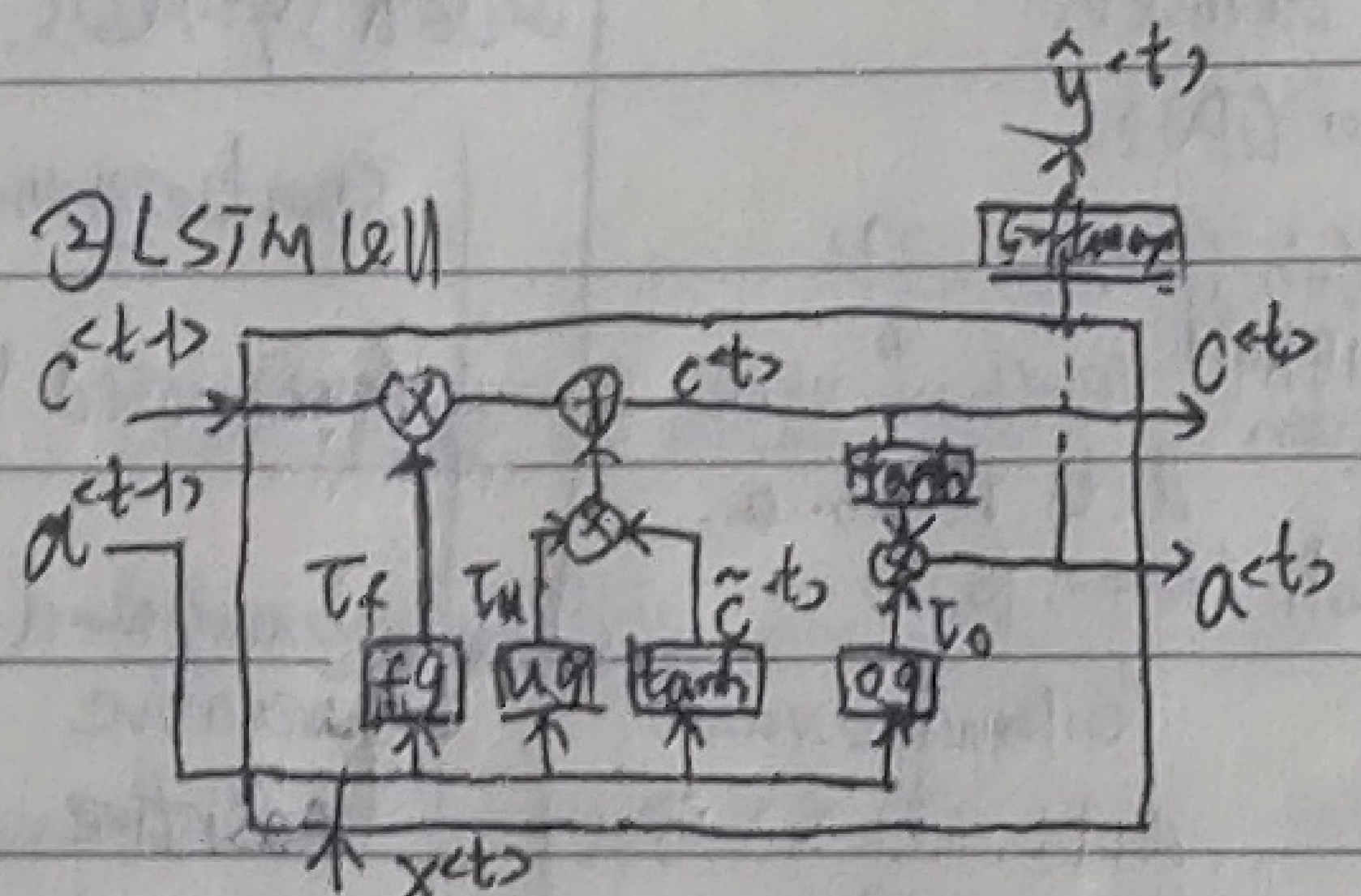
C5. Sequence model

a1-1: Build RNN: step by step

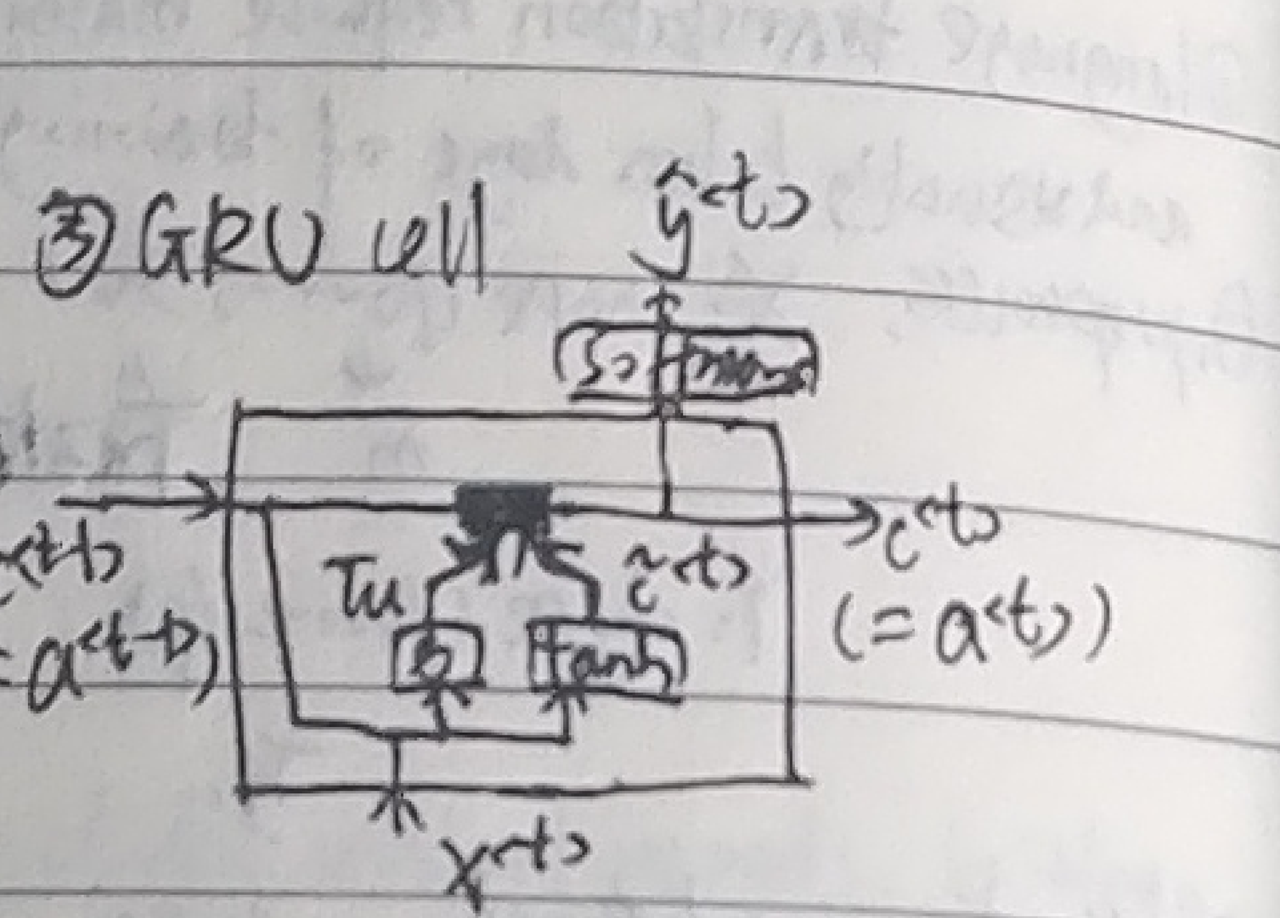
① RNN cell



② LSTM cell



③ GRU cell



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

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

~~rnn forward~~ (instruction)
1. creat a(vector of 0) store hidden state.
2. initialize "next" hidden state as a_0.
3. loop T_x: a. update "next" hidden state by rnn_cell_forward; b. store "next" in a; c. store prediction y; d. Add cache.

$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$

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

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

$c^{<t>} = \Gamma_f \cdot c^{<t-1>} + \Gamma_u \cdot \tilde{c}^{<t>}$

$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$

$a^{<t>} = \Gamma_o \cdot \tanh(c^{<t>})$

$\hat{y}^{<t>} = softmax(W_y \cdot a^{<t>} + b_y)$

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

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

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

$\hat{y}^{<t>} = softmax(W_y \cdot c^{<t>} + b_y)$

a1-2: Character-level language model: Dinosaurus land

① Clipping gradients ("exploding")

② Sampling: $x^{<1>} = \vec{0}$, $a^{<0>} = \vec{0}$; forward propagation; (on trained model) np.random.choice(.., p=), replace x.

③ language model. optimizer(rnn_forward, rnn_back, clip_gradients, update_para.)
model(initialize_para, shuffle_samples), loop optimizer

④ shakespeare writing. (keras)

a1-3: Improvise a Jazz Solo with LSTM

music21 module

① Preprocess. X: (60, 30, 78) → m, T_x, unique values (one-hot). LSTM: $n_a = 64$

② djmodel:
1. create empty outputs.
2. loop T_x: a. select x = X[:, t, :] (78,); b. reshape to (1, 78); c. run x through LSTM_cell; d. propagate a to dense, e. save out

③ post-process to musical chords. most algorithms.

④ deepjazz. github

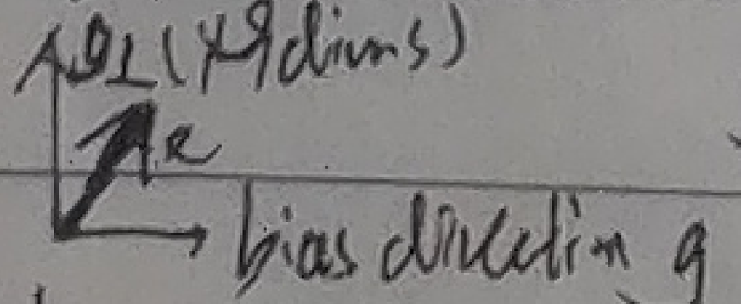
a2-1: Operations on word vectors

① Cosine similarity(u, v) $= \frac{u \cdot v}{\|u\|_2 \cdot \|v\|_2} = \cos\theta$

② Glove vectors: (40000, 50)

③ Word analogy task: $e_b - e_a \approx e_d - e_c$.

④ debias word vectors.



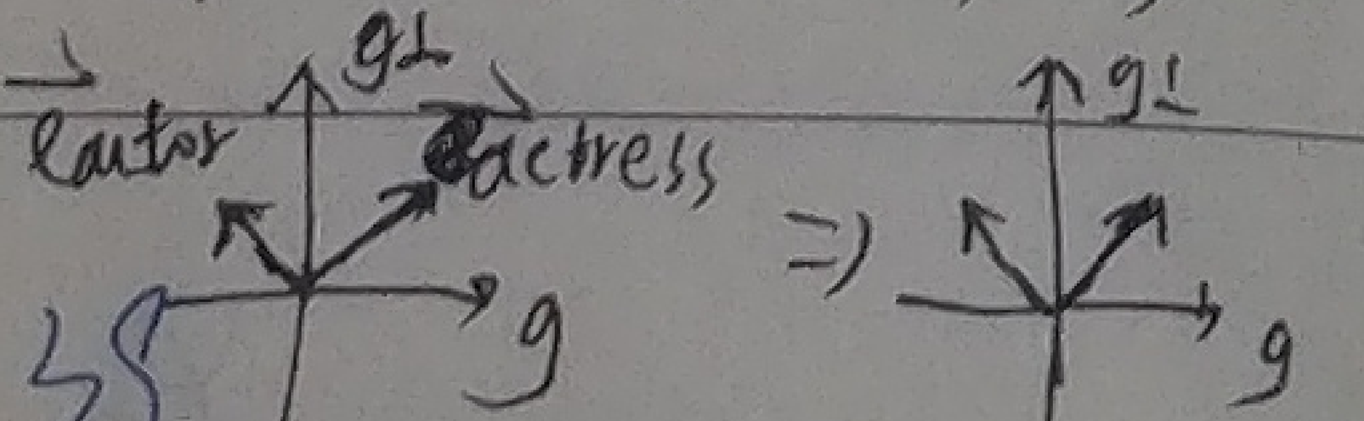
g = $e_{woman} - e_{man}$

1) Neutralize bias for non-gender specific words.

$e^{bias\_component} = \frac{e * g}{\|g\|_2^2} * g$ (projection of e onto g direction)

$e^{debiased} = e - e^{bias\_component}$

2) Equalization algorithm for gender-specific words.



a2-2: Emojify!

emoji module

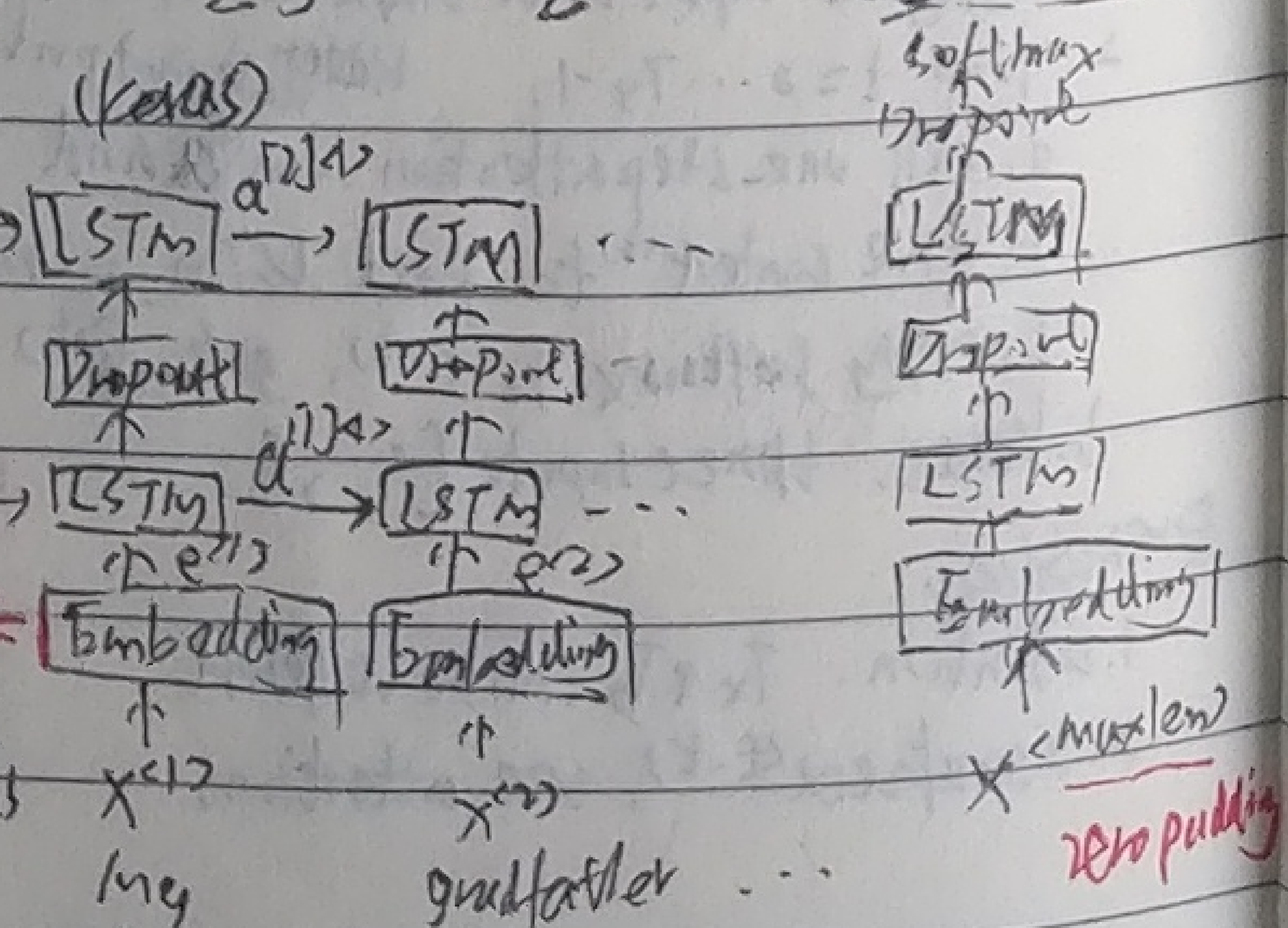
a = pd.crosstab(,) plt.matshow(a, cmap=..)

① Emojifier-V1. word_to_vec_map
sentence_to_average: vector → softmax → (one-hot)

$\mathcal{L}^{(i)} = -\sum_{k=0}^{n_y-1} Y_{oh,k}^{(i)} * \log(a_k^{(i)})$

→ 97.7% train_acc. and generalize well (eg: like/adore)
85.7% test_acc. but not for "not good, not well" (ignore order) (m=127)

② Emojifier-V2. (keras)
train_acc: 97.3%
test_acc: 83.9%



[152, 130, 5, 0, 0] ⇒ $[\vec{e}_{152}, \vec{e}_{130}, \vec{e}_5, \vec{0}, \vec{0}]$
keras layer Embedding() ← trainable = True/False; indexes of words

zero padding

③ remember:
a). NLP task, small train set. word embeddings helps significantly
keras | b). Embedding() layer can be initialized with pretrained values. (fix...)
c). LSTM(return_sequence = True/False), Drop to regularization..

a2-1. Keras Tutorial

keras to tf is like tf to py (high-level framework)

happy-house.

① define model: X_input = Input(shape),
ZeroPadding2D( ), Conv2D, BatchNormalization(axis=3),
Activation('relu'), MaxPooling2D( ), Flatten( ), Dense( ),
model = Model(inputs = X_input, outputs = X, name = '')

② model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

③ model.fit(x, y, epoches = , batch_size =)

④ preds = model.evaluates(X_test, Y_test)

⑤ model.summary( ),
plot_model(model, to_file = ' ')
SVG(model_to_dot(model).create(prog='dot', format='svg'))

IPython.display.SVG, keras.utils.vis_utils.model_to_dot, plot_model

a3: Autonomous driving. car detection

Yolo

① Yolo_architecture: (m, 608, 608, 3) → DNN → Encoding (m, 19, 19, 5, 85)
load yolo model output → reshape (19, 19, 425)
anchor ($p_c, b_x, b_y, b_h, b_w, c$) 80 classes

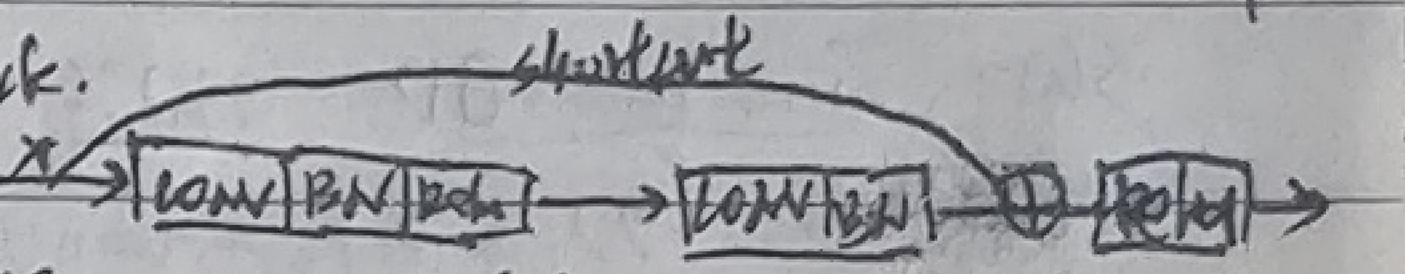
② filtering with a threshold on class scores
box_confidence: (19×19, 5, 1): p_c
boxes: (19×19, 5, 4): bx, by, bh, bw
box_class_probs: (19×19, 5, 80): ($c_1 \dots c_{80}$)
box_scores = box_confidence * box_class_probs (19×19, 5, 80)
box_classes = K.argmax(box_score, axis=-1) (19, 19, 5) — import keras.backend as K
box_class_scores = K.max(box_score, axis=-1), (19, 19, 5)
filtering_mask = box_class_scores >= threshold
scores = tf.boolean_mask(box_class_score, filtering_mask) (?,)
boxes = tf.boolean_mask(boxes, filtering_mask) (?, 4)
classes = tf.boolean_mask(box_classes, filtering_mask) (?,)

③ non-max suppression. IoU $= \frac{A \cap B}{A \cup B}$
a. select box that has the highest score
b. compute overlap with other box, remove > threshold.
c. Go back a until no lower score box
K.get_session().run( ), tf.image.non_max_suppression( )
K.gather(, n)

a2-2: Residual Networks

① deep: represent complex features. but gradient vanish / explode

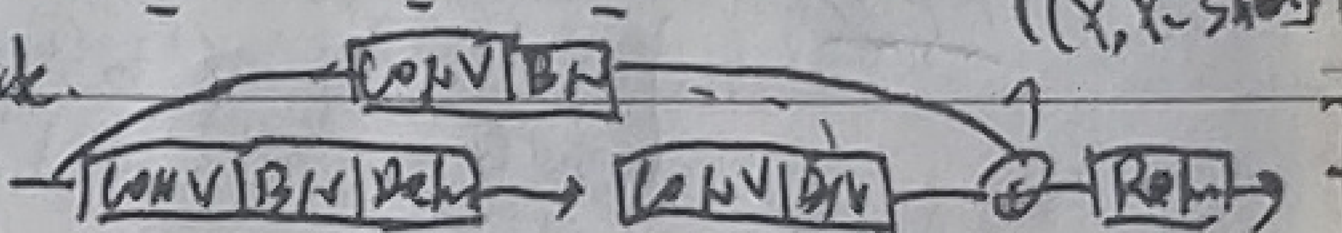
② identity block. (skip 3)



kSize: (1,1), (f, f), (1,1)
stride: (1,1), (same) (1,1), (1,1)
fs: –, –, –

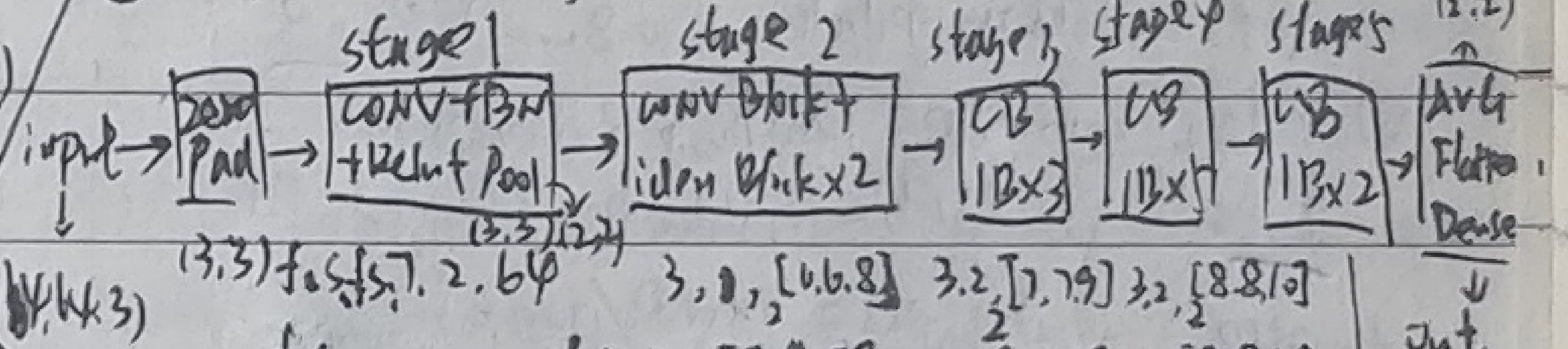
layers.add([X, X_shortcut])

③ convolutional block.



ksize: (1,1), (f, f), (1,1), (1,1)
stride: (s,s), (same) (1,1), (1,1), (s,s)
fs: –, –, –, –

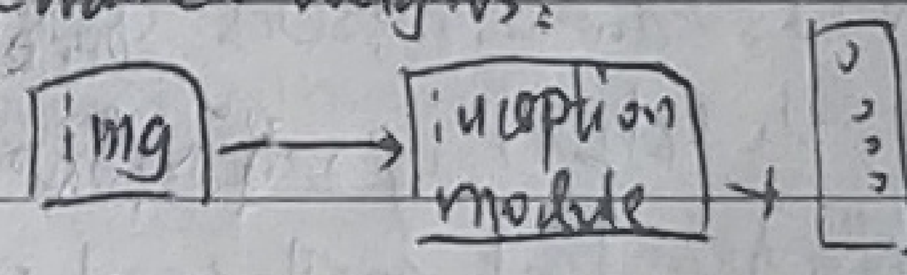
④ ResNet (50 layer)



a4-1: Face recognition

happy house

① pretrained weights.



② triplet loss:
$J = \sum_{i=1}^{m} [\|f(A^{(i)}) - f(P^{(i)})\|_2^2 - \|f(A^{(i)}) - f(N^{(i)})\|_2^2 + \alpha]_+$ → max(, 0)

③ Face verification. database { : } img_to_encoding(img)
dist = np.linalg.norm(encoding - database[name])

④ Face recognition. for (name, db_enc) in database.items( ):

a4-2: DL & Art: Neural style transfer

NST

① transfer learning. VGG19. load model. dict{ } → Tensor

② $J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$

$J_{content}(C, G) = \frac{1}{4 \times n_H \times n_W \times n_C} \sum_{\text{all entries}} (a^{(C)} - a^{(G)})^2$

a_C_unrolled = tf.reshape(tf.transpose(a_C, [3, 2, 1, 0]), [n_C, n_H * n_W])
a_G_unrolled = ...
→ single layer activation (middle)

$J_{style}^{[l]}(S, G) = \frac{1}{4 \times n_C^2 \times (n_H \times n_W)^2} \sum_{i=1}^{n_C} \sum_{j=1}^{n_C} (G_{ij}^{(S)} - G_{ij}^{(G)})^2$

style / Gram matrix

a_S_unrolled = tf.reshape(tf.transpose(a_C, [3, 2, 1, 0]), [n_C, n_H * n_W]).
$G^{(S)}$ = tf.matmul(a_S, tf.transpose( ))
a_G_unrolled = ..., $G^{(G)}$ = tf.matmul(..).

$J_{style}(S, G) = \sum_l \lambda^{[l]} J_{style}^{[l]}(S, G)$, $\lambda^{[l]}$: weight layer
→ : multiple different layer, (better result)

③ G_{ij}: how similar the activations of filter i to j → : how much different features occur together
style G_{ii}: how active filter i is. → : prevalence of different features

a3-1: Neural Machine Translation (NMT)

date translation YYYY-MM-DD

① language translation require massive datasets and usually takes days of training on GPUs.

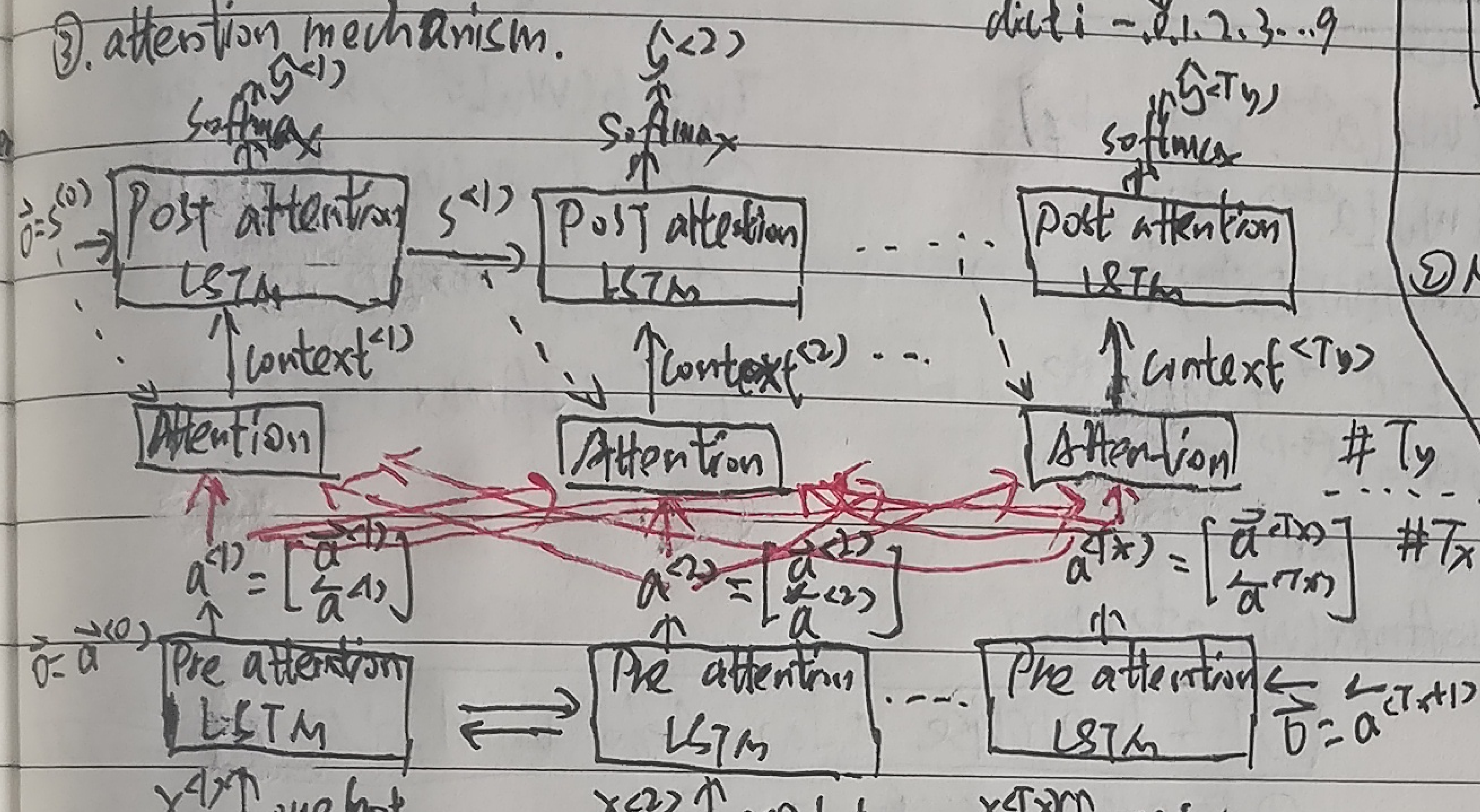
② preprocess. X: shape (10000, 30) ; Xoh: (10000, 30, 37)

m; T_x / maxlen; one hot; len(human_vocab); dict; 1.2.3..a..

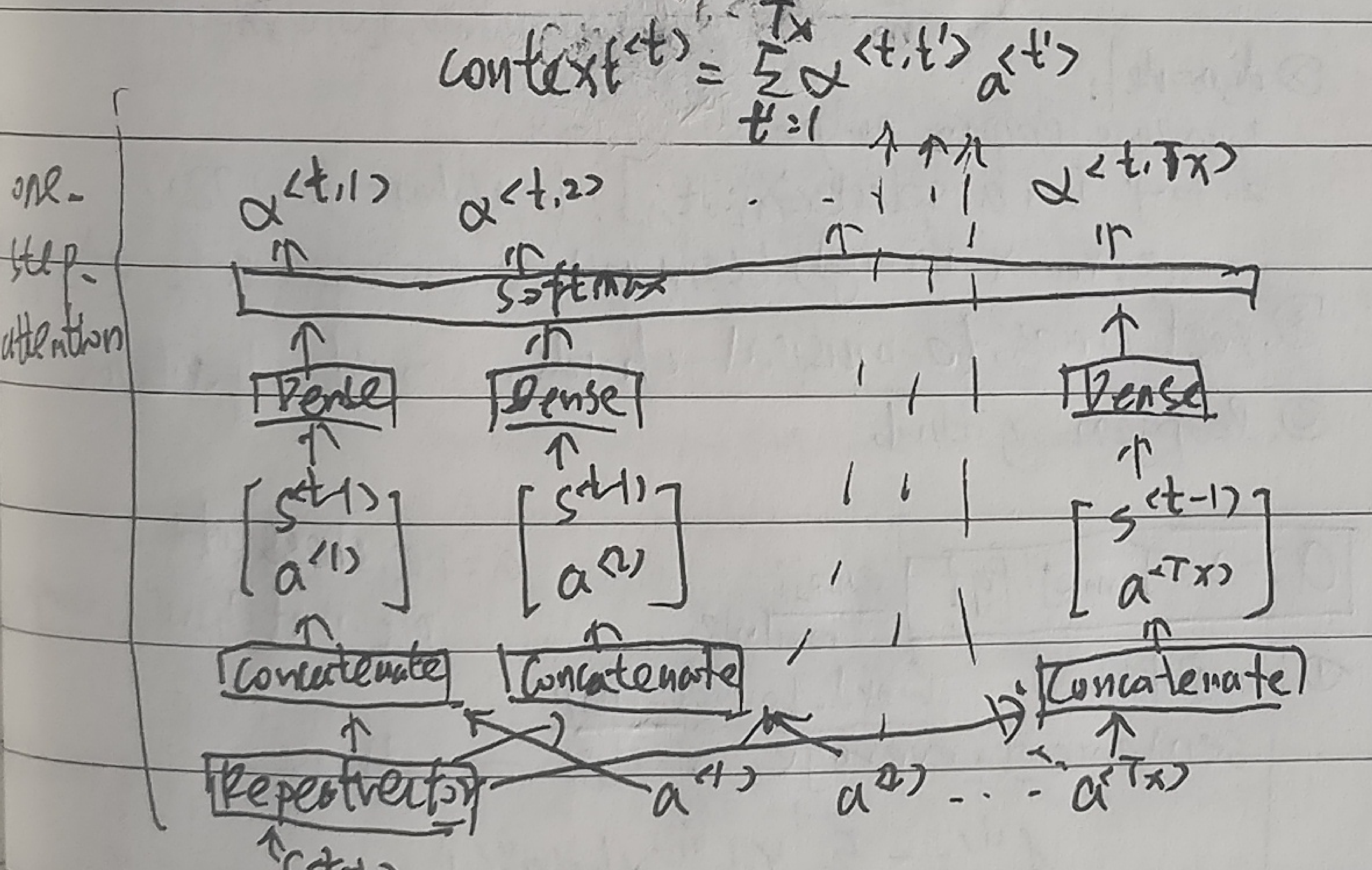
Y: shape (10000, 10) ; Yoh: (10000, 10, 11)

T_y; len(machine_vocab); dict: -.0.1.2.3...9

③ attention mechanism.



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



④ model:

1. Propagate input to a Bidirectional LSTM.
2. loop t = 0 ... T_y-1:
   a. call one_stepattention on a (hidden state output) and $s^{<t-1>}$, get $context^{<t>}$
   b. give $context^{<t>}$ to post LSTM, with s, c.
   c. apply softmax to $s^{<t>}$, get $\hat{y}^{<t>}$ — no $\hat{y}^{<t-1>}$ for this task. date strong dependency.
3. keras. three inputs [, ,].

⑤ remember:

1. attention. T_x & T_y can be different.
2. visualize $\alpha^{<t,t'>}$, see attention.

a3-2: Trigger Word detection "activate"

pydub module

computed by sliding a window over the raw audio signal, calculate the most active frequencies in each window using Fourier transform.

① data synthesis. audio recording: long list of number measuring air pressure changes. 44100Hz

spectrogram. f ↑ → time. 10s: → T_x = 5511.

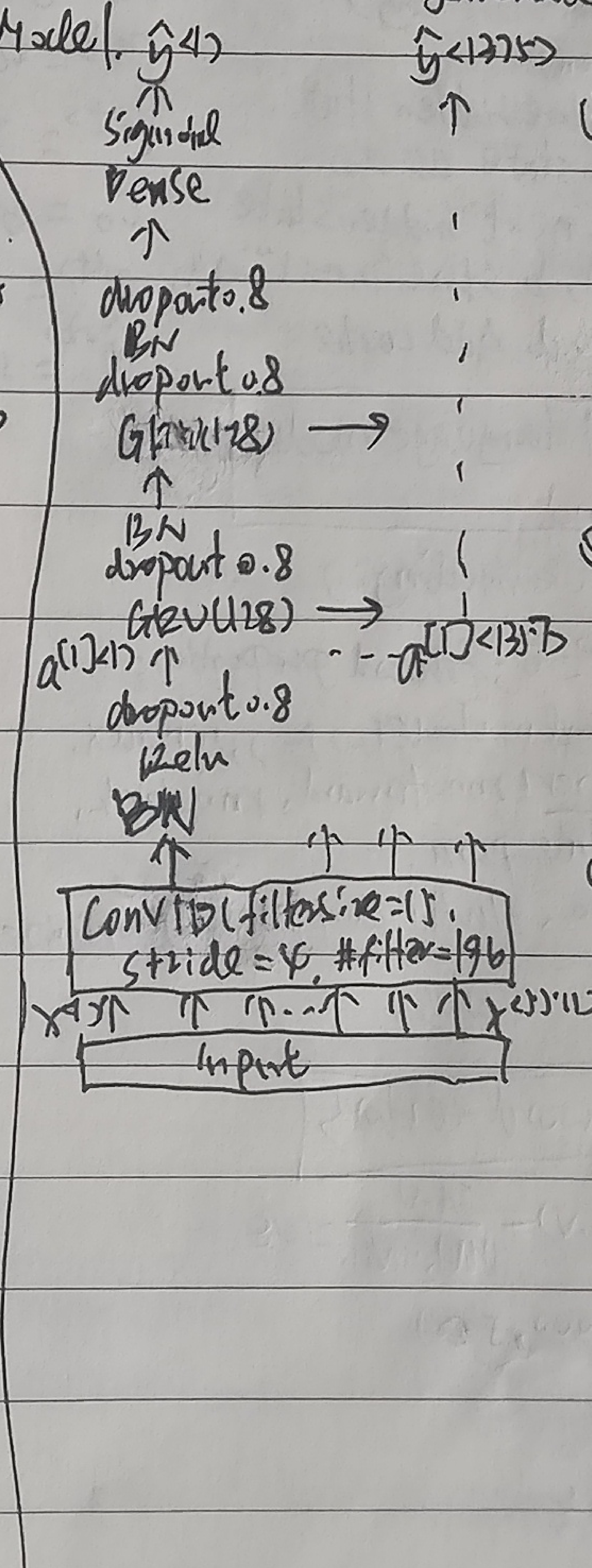
10000: used by pydub to synthesize audio. T_y = 1375. 10/10000 = 1ms

background (10s), negative, positive. } overlay, but not overlapping (insert). same for labels

→ advantages: ①. easy to record lots of negative / positive / background

② convenient to generate labels.

② Model.



③ load weight. (m=4000, 3h GPUs)

acc 93.5% dev. not good. 0/1 not balance, easy acc 90%

F1 score

④ trigger a "chiming" sound when prob > threshold; chim sound at most once every 75 output steps (non-max suppression)

⑤ remember.

a. Data synthesis effective for speech problem, specially trigger problem

b. use spectrogram. 1D conv is common pre-processing before RNN/GRU/LSTM