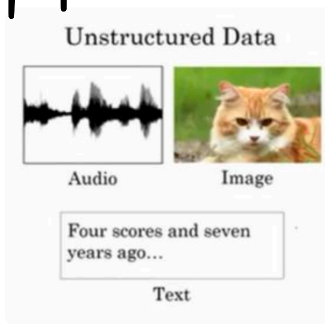
Deep Learning 学习笔记

Lecture One : Neural Network and Deep learning

Introduction to Deep Learning :

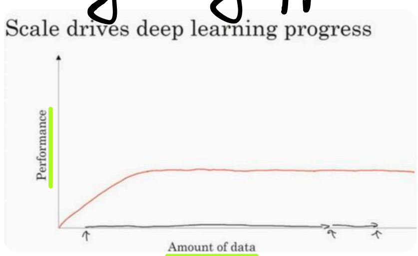
① Neural Network has high performance on :

Unstructured Data :



② Why is deep learning taking off?

1' Big Data



2' From Sigmoid to ReLu : FASTER ! and avoid Vanishing Gradient.

Basic of Neural Network programming :

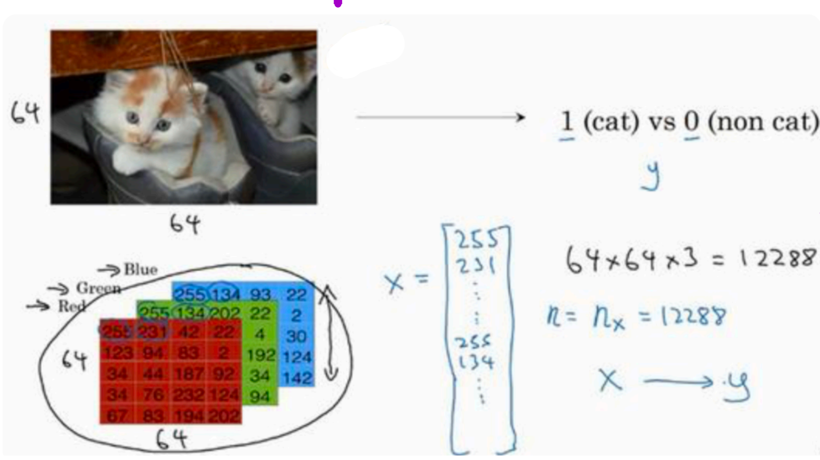
① Binary Classification :

1' Logistic Regression :

A image of cat is defined as : (64 * 64 * 3) RGB

The feature vector will be : n_x .

The dimension of feature vector is : 12,288



⇒ Input : X.shape

(n_x, m)

Dimension quantity

Label : Y.shape

$(1, m)$

0 or 1

Via sigmoid function ($\delta(z) = 1/1+e^{-z}$)

$\hat{y} = \delta(b + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n)$. let $x_0 = 1$.

$\hat{y} = \delta(\theta^T X)$, where, $\theta = [\underset{b}{\theta_0}, \underbrace{\theta_1, \cdots, \theta_{n_x}}_{W}]^T$

② Logistic Regression Cost Function:

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

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

③ Gradient Descent in Logistic Regression:

Given two features: x_1, x_2, we have $z = w_1x_1 + w_2x_2 + b$, $\hat{y} = a = \delta(z)$

Cost of EACH sample is: $L(a, y) = -y\log(a) - (1-y) \cdot \log(1-a)$

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

$\partial L/\partial w_1 = x_1 \cdot dz = x_1(a-y)$, $\partial L/\partial w_2 = x_2 \cdot dz = x_2(a-y)$, $\partial L/\partial b = dz = (a-y)$

$\Rightarrow w_1' = w_1 - \alpha \cdot \partial L/\partial w_1$, $w_2' = w_2 - \alpha \cdot \partial L/\partial w_2$, $b = b - \alpha \cdot \partial L/\partial b$

The disaster of "For loop": to traverse m samples is only one step, you also need to traverse each feature!

④: Vectorization: for accelerating computation.

Without vectorization: $z = w^T x + b$

```
z=0;
for i in range(n_x):
    z+=W[i]*X[i]
z+=b
```

With vectorization:

```
z = np.dot(w,x)+b
```

Conclusion: Try not to use "FOR loop"

Homework:

Potential Risk

① In numpy : { 1 means RAW, 0 means COLUMN } and { a= np.ran~~dom~~.randn(5) ; a= np.random.randn(5, 1) }

② ASSERT is always useful : assert(a.shape == (5,1)) for reducing risk.

③ L2 normalization : np.sum(~~(yhat-y)~~) VS np.dot((y-yhat),(y-yhat).T)

④ np.squeeze(): delete shape where dimension is one. [[17]] ⟹ 17

⑤ Remember Reshape your training set!

★⑥ Trick : X_flatten (X.shape[0], -1).T ⇒ (a,b,c,d) ⇒ (b*c*d, a)

Shallow neural networks:

Neural Network Overview:

$a^{[0]} = X$: $\boxed{x_1, x_2, x_3}$ $\quad$ $a^{[1]}$: $\boxed{a_1^{[1]}, a_2^{[1]}, a_3^{[1]}, a_4^{[1]}}$ $\quad$ $\boxed{a^{[2]}}$

where $a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$, $W_{4\times3}$, $b_{4\times1}$

Activation Functions:

sigmoid : Only used in output layer of two-classification

tanh : widely used.

ReLu . Leak ReLu : widely used.

Random Initialization:

Without Random Initialization, Gradient Descent will not work!

Deep Neural Network:

Forward and backward propagation:

① Forward

1' $\underline{z^{[l]}} = W^{[l]} \cdot \underline{a^{[l-1]}} + b^{[l]}$

Save as cache! input

2' $\underline{a^{[l]}} = g^{[l]}(z^{[l]})$

output

② Backward:

1' $dz^{[l]} = \underline{da^{[l]}} \cdot g^{[l]\prime}(z^{[l]})$

input

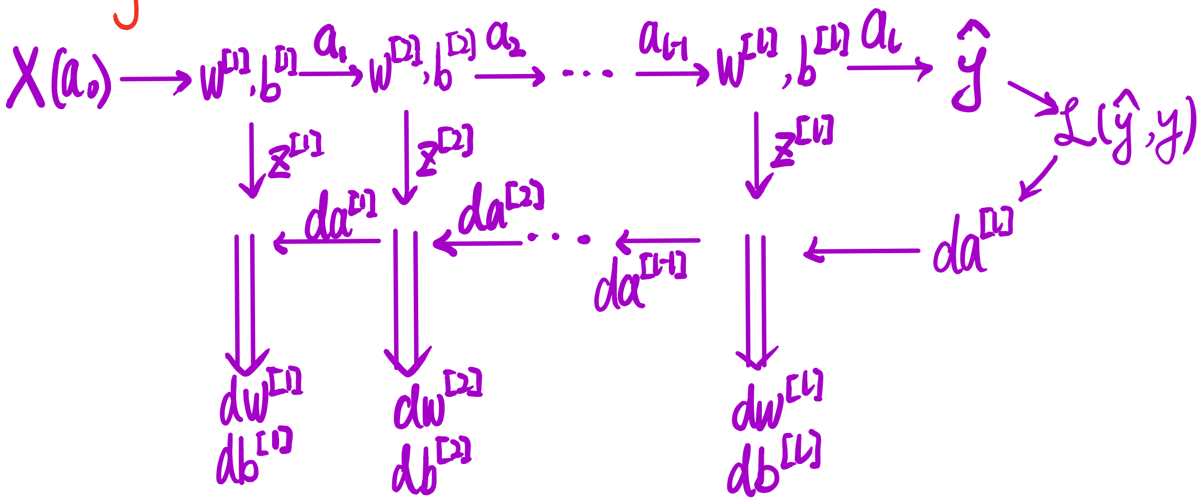
2' $dw^{[l]} = dz^{[l]} \cdot a^{[l-1]}$

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

4' $da^{[l-1]} = W^{[l]T} \cdot dz^{[l]}$

output

③ Summary



Lecture Two: Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization.

Practical aspects of Deep Learning:

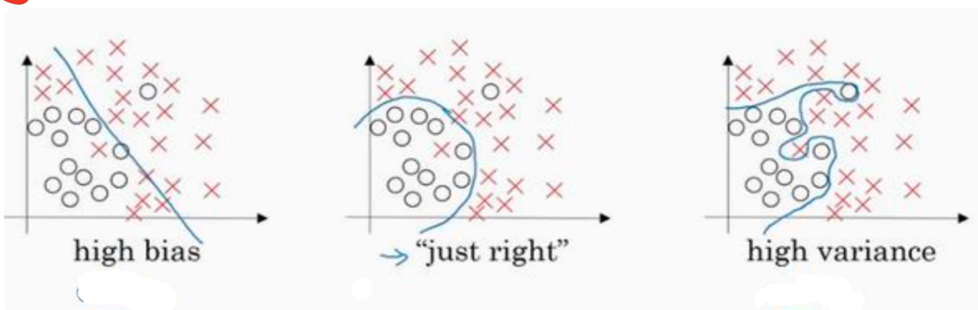
① Divide data set into: Train / Dev / Test

small data: 0.6 0.2 0.2

1m big data: 0.98 0.01 0.01

over 1m big data: 0.995 0.0025 0.0025

② Bias / Variance:



high bias: underfitting ⇒ change parameter or model

hight variance: overfitting ⇒ increase data size or Regularization

③ Regularization:

☆ L2 Regularization: $J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|^2$ sum the square of each element

L1 Regularization: $J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|$

where $\|w\|^2 = \sum_i\sum_j (w_{ij})^2$

④ Dropout Regularization:

⑤ Normalizing inputs:

⑥ Vanishing / Exploding gradients:

Weight Initialization: For certain layer:

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

2 is better for ReLu

number of $l-1$ layer's neuron.

Optimization algorithms:

① mini-batch:

$\underbrace{\{1\}\quad\{2\}\quad\{3\}\quad\{4\}\quad\{5\}\quad\{6\}}_{m}$ → $m/6$

one epoch only means one traverse of data

mini-batch makes you do more gradient-descent

② Gradient descent with Momentum: usually set 0.9

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

we obtain: $W = W - \alpha \cdot V_{dw}$, $b = b - \alpha V_{db}$ learning rate

③ RMS prop:

$W := W - \alpha \cdot dw / \sqrt{S_{dw}}$ $b := b - \alpha \, db / \sqrt{S_{db}}$

④ Adam: Widely Used!!!

⑤ Learning rate decay:

set $\alpha = 1 / (1 + decayrate * epoch\text{-}num) * a_0$

⑥ The problem of local optimal): This may not that important!

Hyperparameter tuning:

① Batch normalization: ⇒ normalization "$z^{[l]}$" Before it came to "$a^{[l]}$"

For certain layer: $\mu = \frac{1}{m} \sum_i z^{(i)}$, $\delta^2 = \frac{1}{m} \sum_i (z_i - \mu)^2$

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

② Softmax regression: (multi-classify)

1' activation: $a_i^{[L]} = e^{z^{[i]}} / \sum_j e^{z^{[j]}}$

2' loss: $\mathcal{L}(\hat{y}, y) = -\sum_i y_i \log \hat{y}_i$

Lecture 3: Structuring Machine Learning Projects.

Orthogonalization: resolve complicated thing into single function

Error analysis:

Transfer Learning: (Serial) VS Multi-Task Learning (parallel)

End-to-end deep learning: DIRECT! better with lots of data.

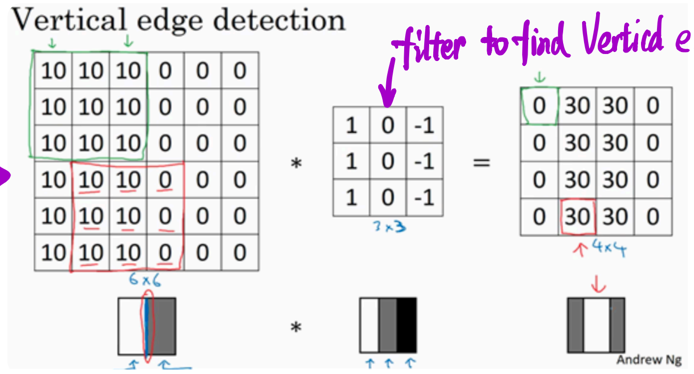
Lecture 4: Convolutional Neural Network

Foundations of Convolutional Neural Networks:

Big Picture ⇒ High Dimention ~~⇒~~ Low memory

Edge detection

filter to find Vertical edge

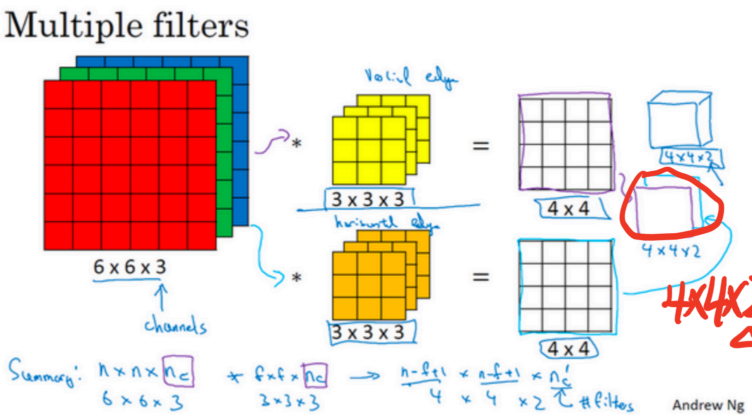


Padding:

Prevent image from becoming "1x1" size, padding image with "0".

Stride convolution:

Convolutions over volumes:



4x4x2!

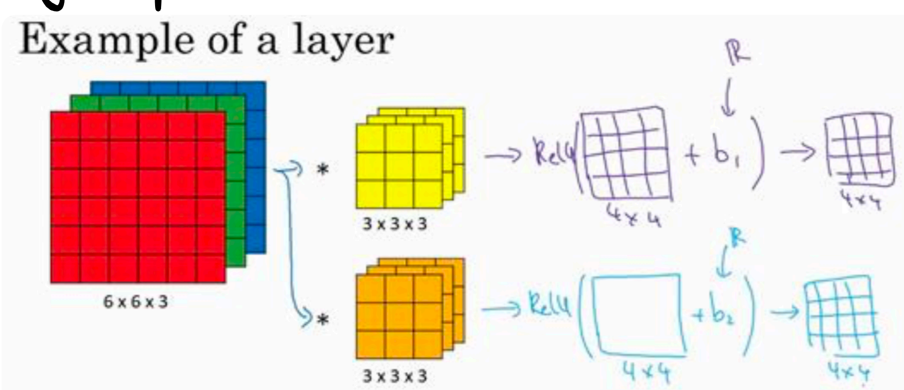
Summary:

$n \times n \times n_c * f \times f \times n_c \rightarrow n-f+1 \times n-f+1 \times n_c'$

n_c: chanel

n_c': the number of filters!!

One layer of a convolutional network:



→ 4x4x2

∴ the number of parameter : $(\underbrace{3\times3\times3}_{\text{each filter}}+\underbrace{1}_{\text{bias}}) \times \underbrace{2}_{\text{filter\_num}} = 56$

Parameter description: if layer l is a convolution layer

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

$p^{[l]}$ = padding

$s^{[l]}$ = stride

$n_c^{[l]}$ = number of filters

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

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

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

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

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

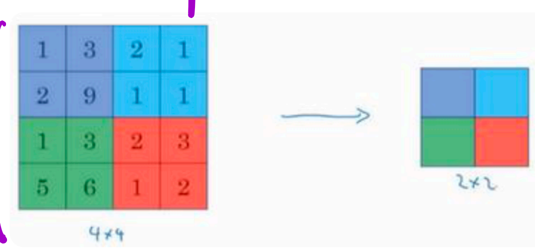
Bias: $n_c^{[l]} \rightarrow (1, 1, 1, n_c^{[l]})$

Pooling layers:

Further reduce the size of the model.

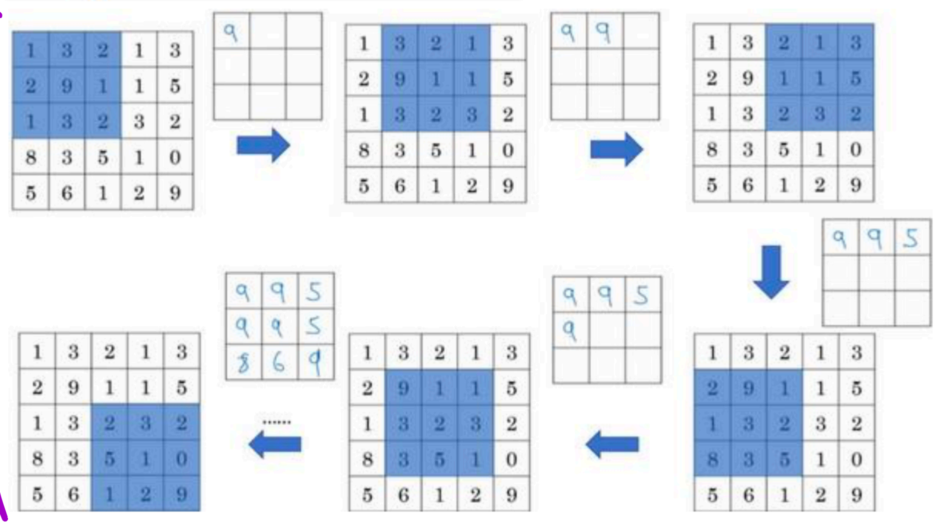
① max pooling:

One example:



$\begin{cases} f=2 \\ s=2 \end{cases}$ widely used

another example $\begin{cases} f=3 \\ s=1 \end{cases}$

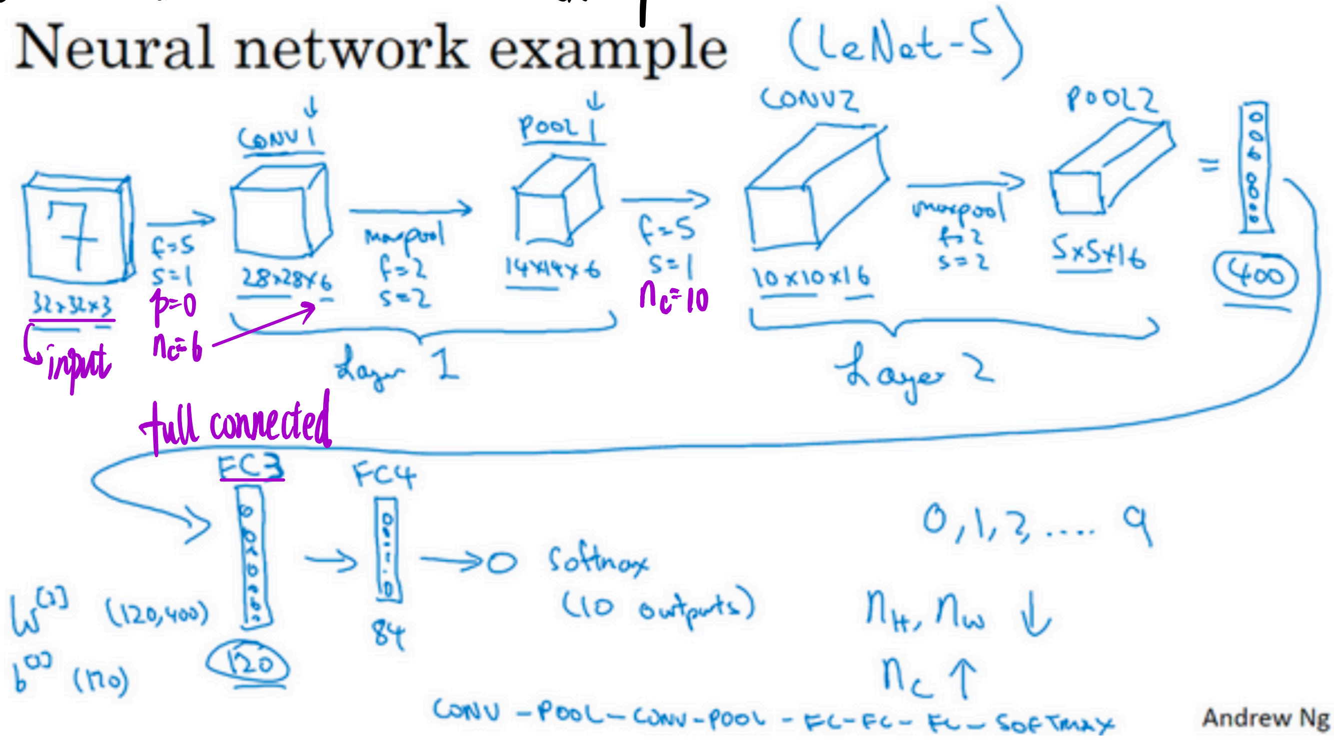


② formula:

max pooling output size: $\left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_W - f}{s} + 1 \right\rfloor \times n_c$

convolution output size: $\frac{n+2p-f}{s} + 1$

Convolutional neural network example



Lower height and weight & higher channels.

Why convolutions?

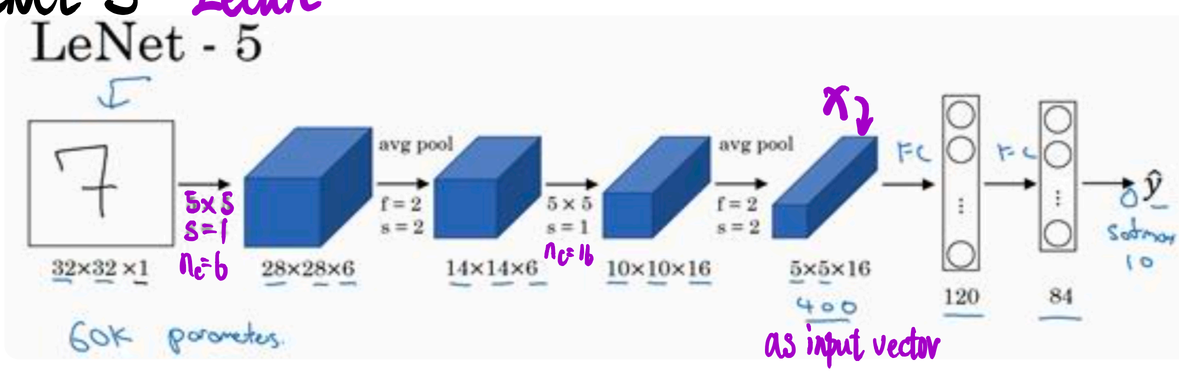
① Shared Parameters.

② Sparse Connection.

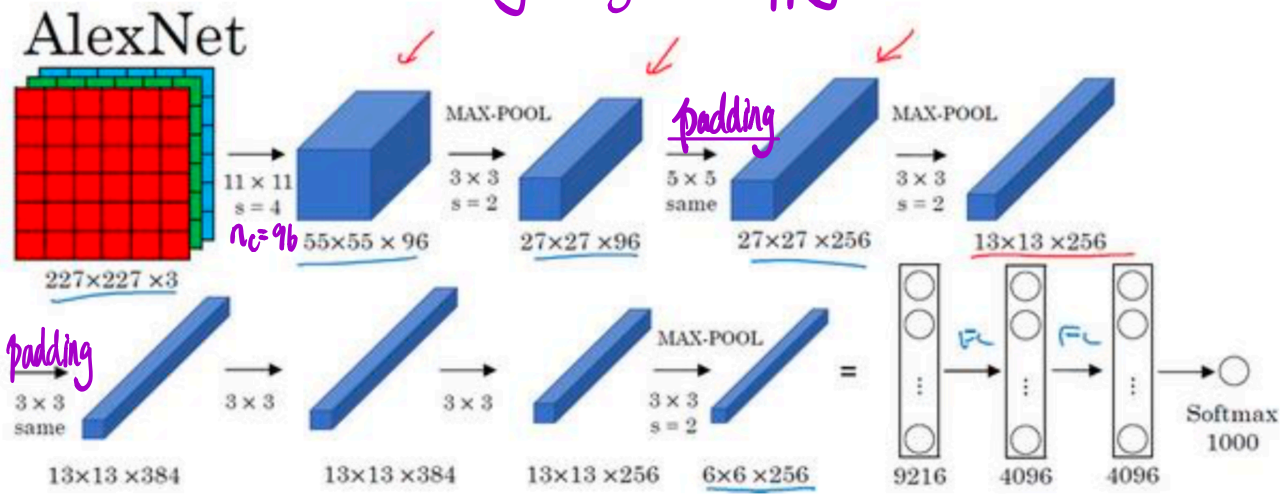
Deep Convolutional models: case studies.

Classic networks:

① LeNet-5: Lecun



② AlexNet: Alex Krizhevsky . Ilya . Geoffery Hinton.

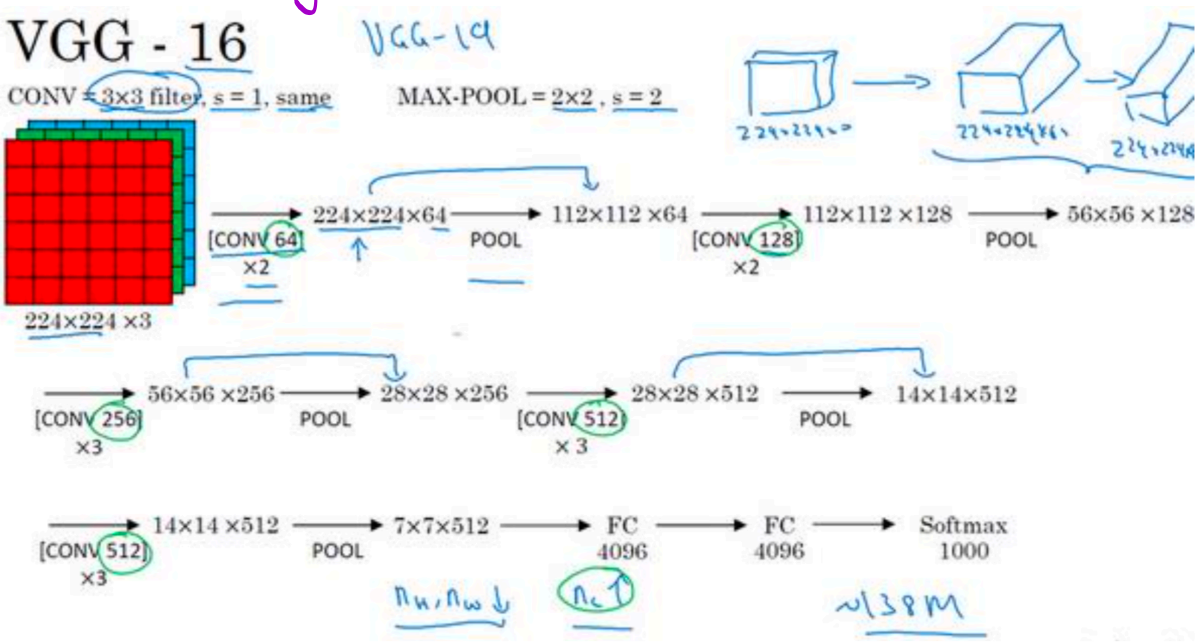


Activation is: ReLu compare to Sigmoid in LeNet-5

③ VGGNet:

filter_num: 64 → 128 → 256 → 512 more and more

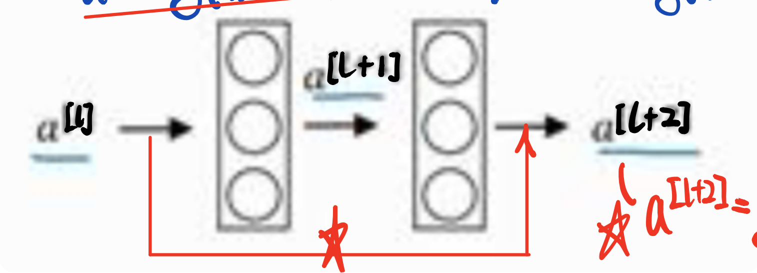
Convolutional_layer_num: 16



Residual Network [ResNets]

① Residual Block: message send to deeper layer directly!

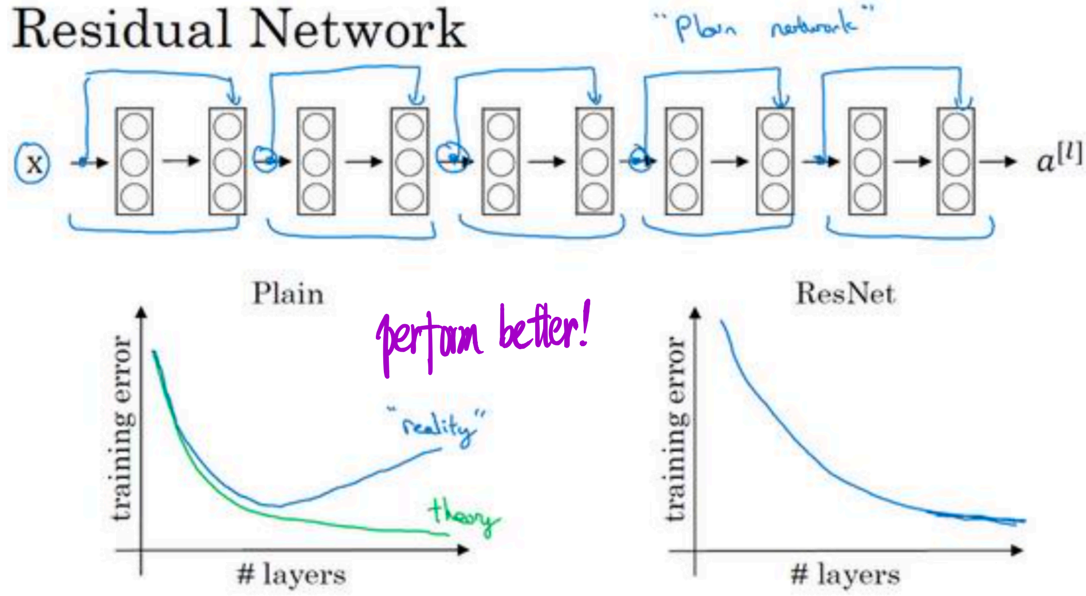
$\cancel{a^{[l+1]} = g(W^{[l+1]} \cdot a^{[l]} + b^{[l+1]}) \quad a^{[l+2]} = g(W^{[l+2]} \cdot a^{[l+1]} + b^{[l+2]})}$



before Relu, after Linear

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

② Residual Network with 5 blocks:



③ Why ResNets work?

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

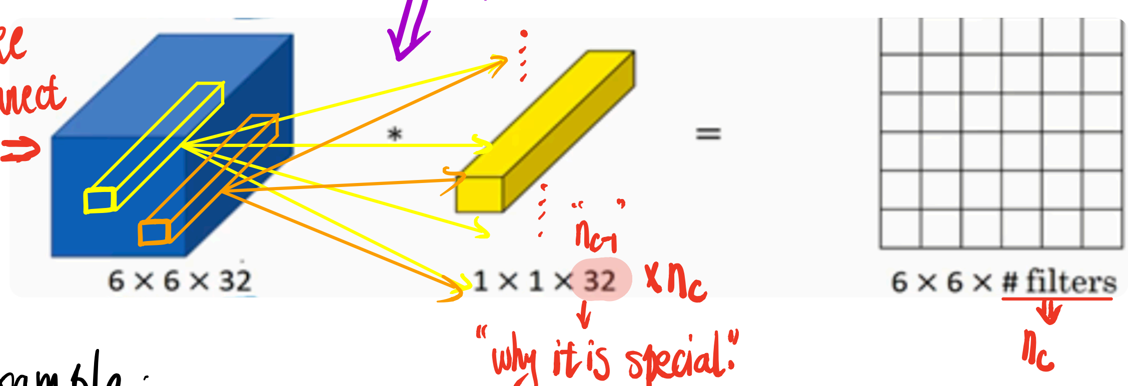
Relu   L2 regularization / Weight Decay make them to 0

$\therefore a^{[l+2]} = a^{[l]}$

1×1 convolutions / (Network in network)

① Real size: $1 * 1 *$ "n_{c-1}", this part like FNN, so it also called

Just like full-connect NN, ⇒ so



② An example:

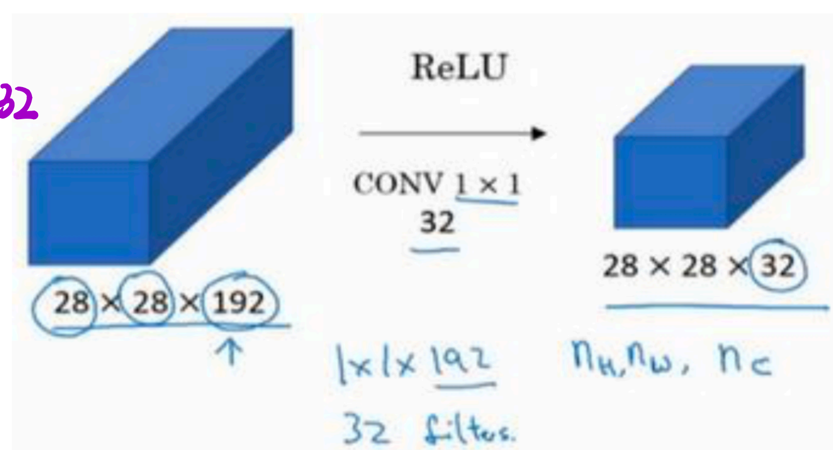
How could we convert 28×28×192 to 28×28×32

with 1×1 convolutions

$28 \times 28 \times \underline{192}$ ↗ n_{c-1}

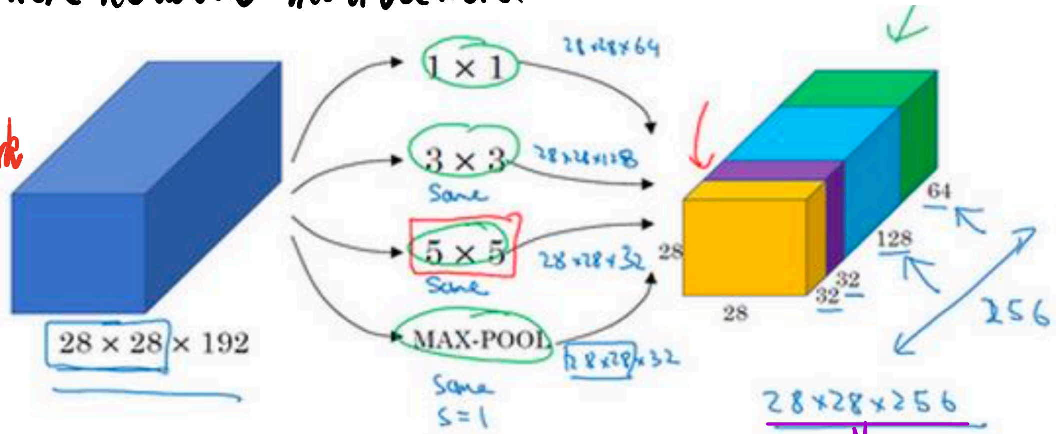
↓ $1 \times 1 \times \underbrace{192}_{n_{c-1}} \times \underbrace{32}_{n_c}$

$28 \times 28 \times \underbrace{32}_{n_c}$



Google Inception network motivation.

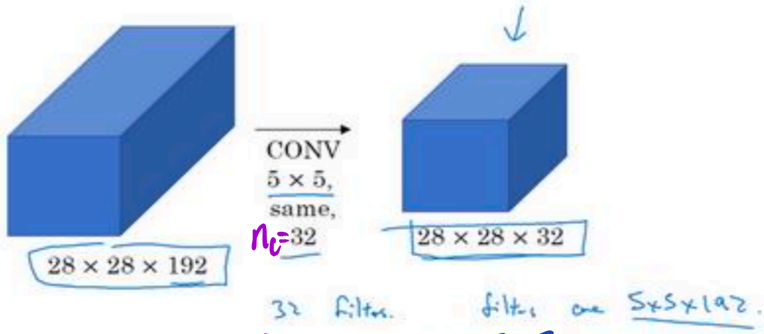
Let neural network learn by itself



it is called Inception Module

Problem: COST!

The problem of computational cost

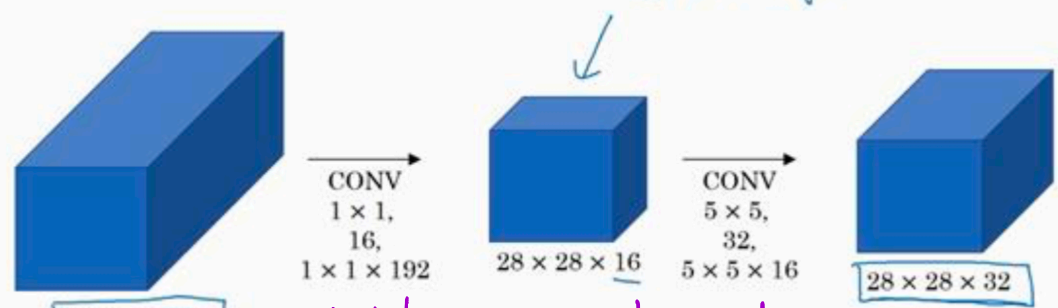


Calculation: $(28\times28\times192\times5\times5\times32=120M)$

Solution:

Using 1×1 convolution

"bottleneck layer"

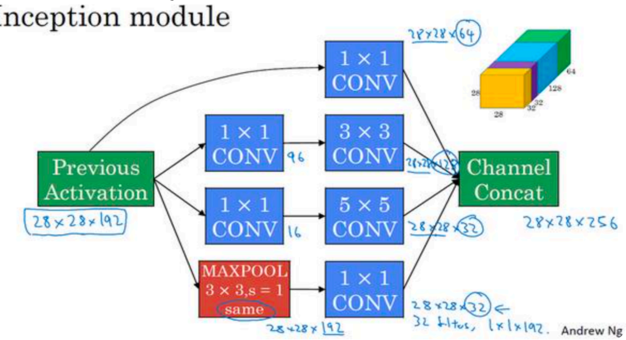


Shrink it then enlarge it

Calculation: $28\times28\times192\times16+28\times28\times16\times32\times5\times5=12.4M$

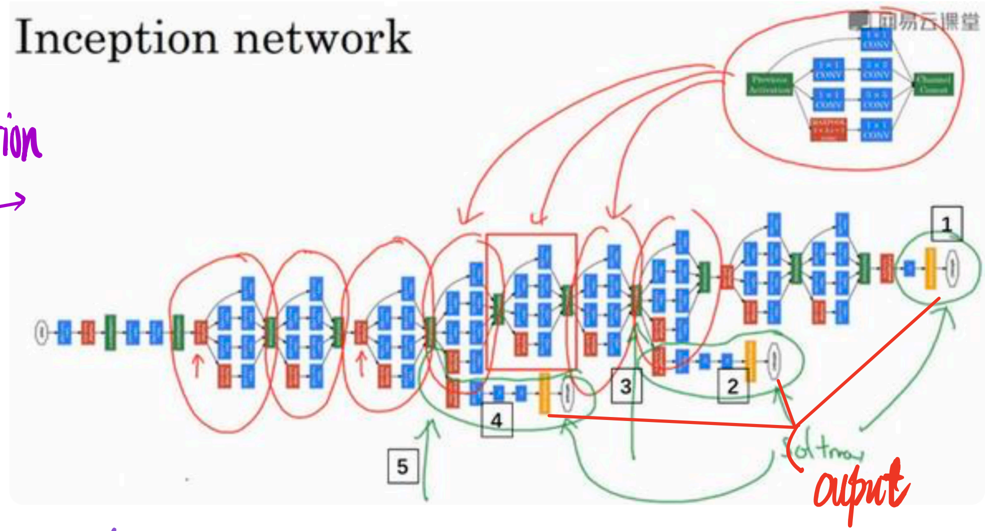
Inception network:

Inception module



with 1x1 conv to reduce calculation.

combine inception module into inception network



If you want to go deeper: you can jump (ResNet) or you can quit everywhere (Inception)

Transfer Learning: Use other weights as Prior knowledge.

Tricks for doing well on benchmarks / winning competitions.

① integration: mean the output of LOTS OF neural networks.

② enlarge dataset: Multi-crop

Object detection:

Object localization:

①Symbolic representation:

1' Top Left corner: (0,0); Bottom Right corner (1,1)

2' b_x, b_y, b_h, b_w → central point (b_x, b_y); $b_h . b_w \in [0,1]$

② How to define label "Y":

$$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}$$

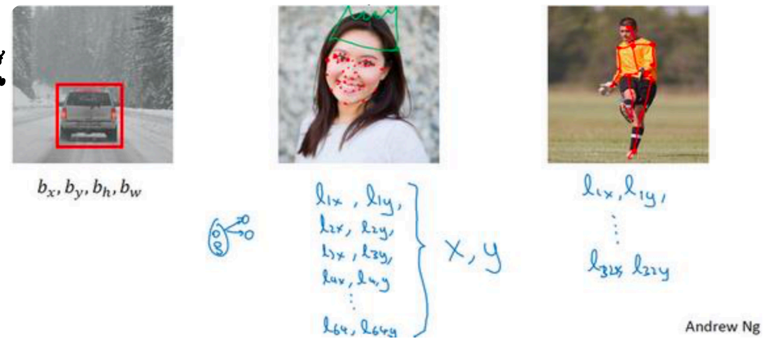
P_c → have object or not {0,1}

b_x, b_y, b_h, b_w → coordinate $\in [0,1]$

C_1, C_2, C_3 → what object is {0,1}

③ Loss function: $\mathcal{L}(\hat{y}, y) = \sum_i (\hat{y}_i - y_i)^2$

Landmark detection:



Object detection:

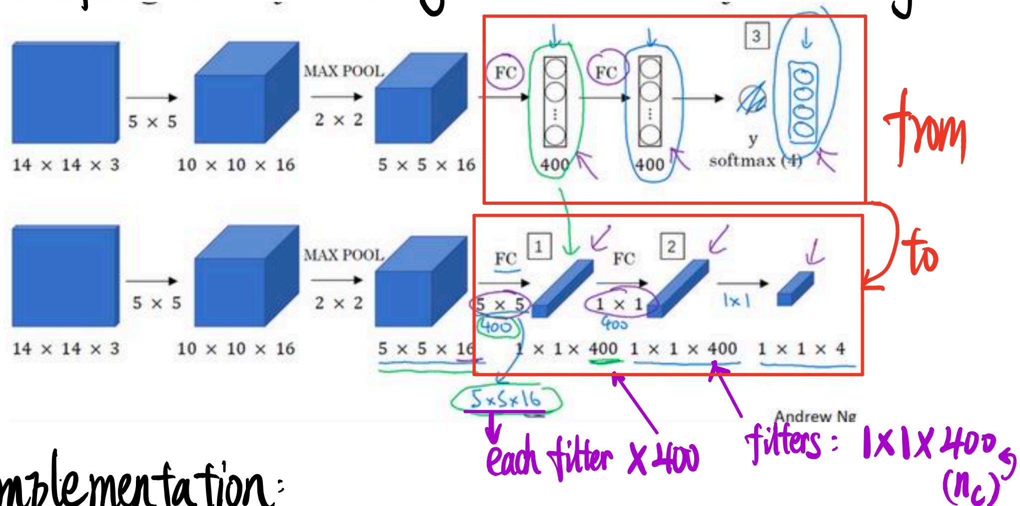
Sliding Window: → low efficiency

1' Train a CNN to detect car

2' Sliding windows in different size

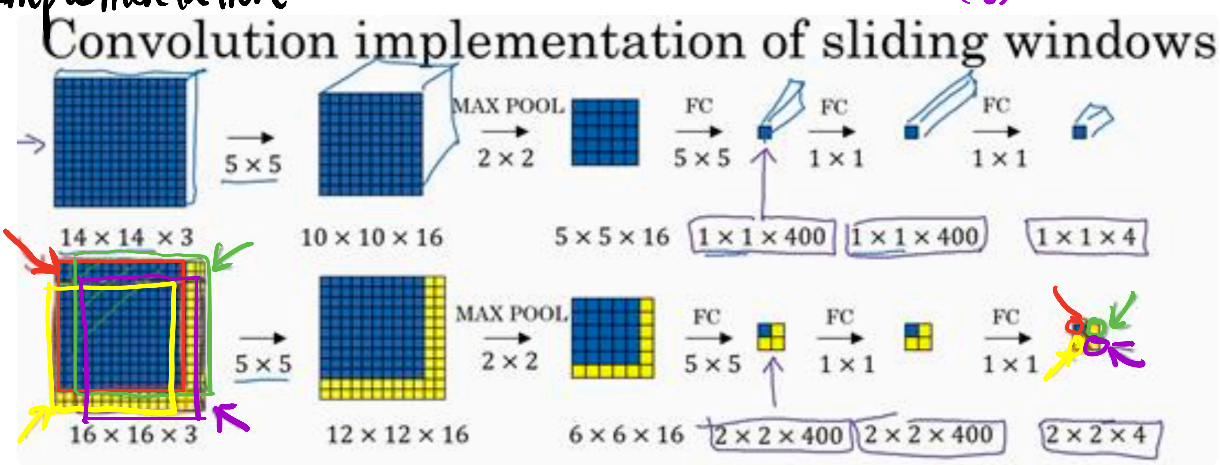
Convolutional implement of Sliding windows:

① Turn full-connected layer into Convolutional layer:



② Implementation:

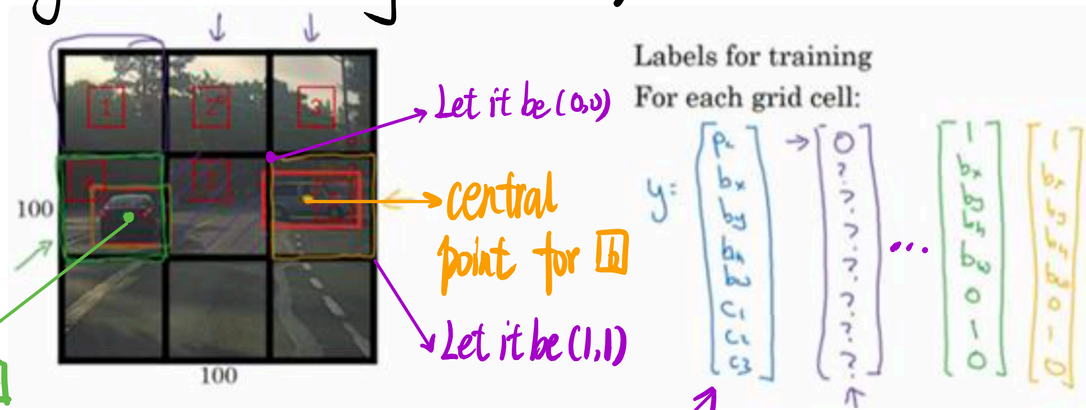
Slide 4 time



③ Summary: sub_image → CNN → Output

Bounding Box predictions: (How to be more accurate)

① Yolo Algorithm: (You Only Look Once)



central point for 4

output is: 3x3x8 — 8 vectors.

⇒ Let 6 central point be (0.4, 0.3)

then determine b_h & b_w

Intersection over union: evaluate detection algorithm.

IoU = Size of intersection / Size of Union. { ≥ threshold value ✓ ; else ✗ }

Non-max suppression:

make sure your algorithm only detective object once a time.

→ Only the classification results with the highest probability are output.

Anchor Boxes:

Let one box to detect multi-objects.

→ pre-define different kind of anchor boxes

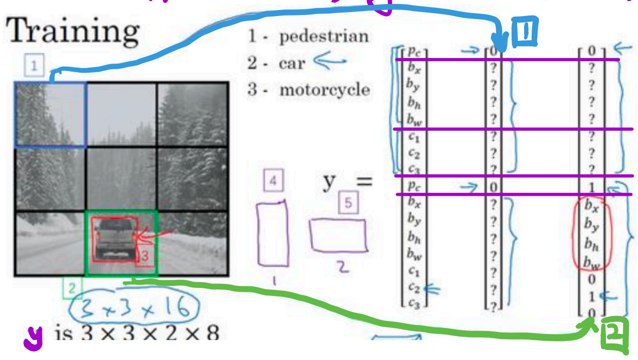
∴ $y_{old} = [p_c\ b_x\ b_y\ b_h\ b_w\ c_1\ c_2\ c_3]^T$

$y_{new} = [p_c\ b_x\ b_y\ b_h\ b_w\ c_1\ c_2\ c_3\ p_c\ b_x\ b_y\ b_h\ b_w\ c_1\ c_2\ c_3]^T$

Yolo Algorithm: Put all component above into algorithm.

① Build Training Set:

Let anchor box_num = 2, $y = 3\times3\times(2\times8)$



② Predicting

③ Outputting the non-max supressed outputs

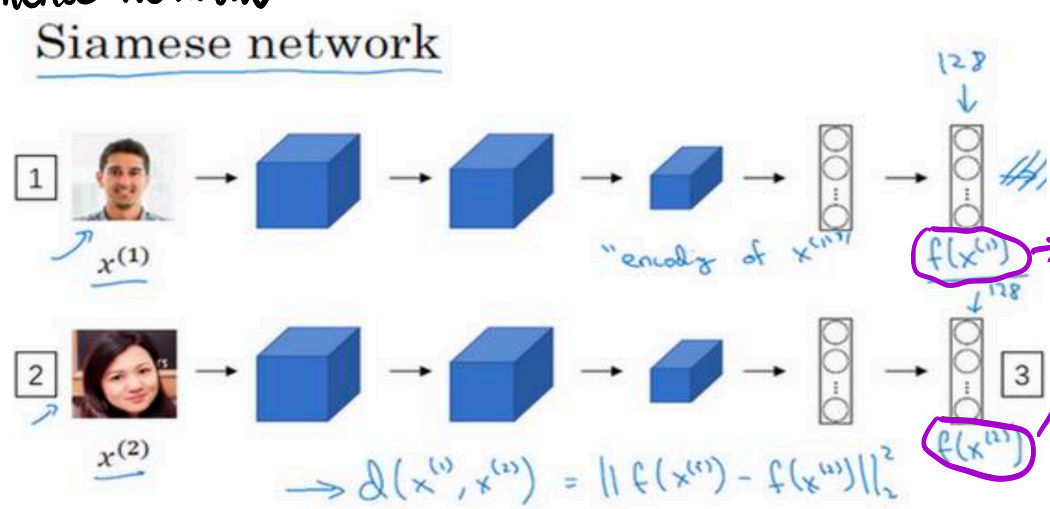
Special applications: Face recognition & Neural style transfer

One-shot learning:

One sample to recognize a person.

→ to learn similarity function is a good idea.

Siamese network:



→ encoder for $x^{(1)}, x^{(2)}$

same person: low $\|f(x^{(i)}) - f(x^{(j)})\|^2$

different peron: large $\|f(x^{(i)}) - f(x^{(j)})\|^2$

Triplet loss: (A, P, N) ═ (Anchor, Positive, Negative)

① Loss function

Want: $\underbrace{\|f(A)-f(P)\|^2}_{d(A,P)} + \alpha \leq \underbrace{\|f(A)-f(N)\|^2}_{d(A,N)}$

$\alpha \downarrow$ margin to prevent "$0 \leq 0$"

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

② Choosing the triplets A, P, N

1' A, P must be same person.

2' A, N must be different person.

③ Choosing triplets that're "hard" to train on.

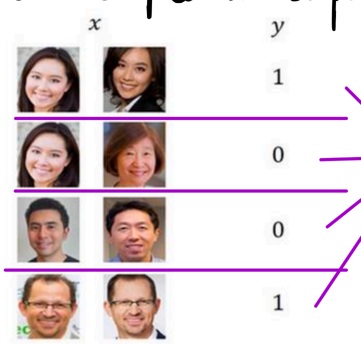
$d(A,P) \approx d(A,N)$

Face verification and binary classification:

① other parameter learning method:

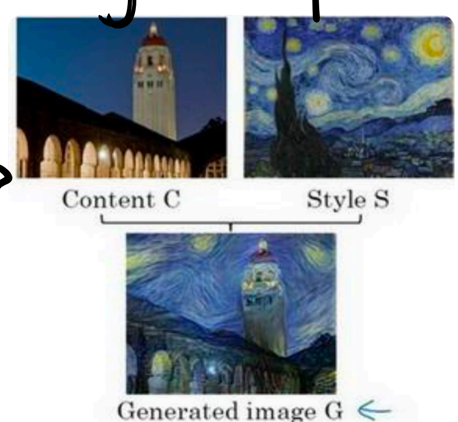
1' logistic regression

② face verification supervised learning:



$\Longrightarrow$ data & label for face verification.

Neural Style Transfer:



We define Loss function as:

① $J_{content}(C, G)$

② $J_{style}(S, G)$

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

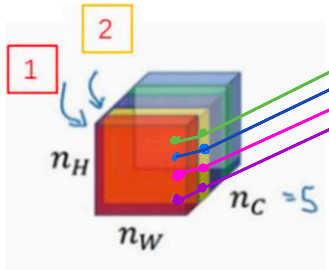
Content cost function:

If you use hidden layer l to compute content cost:

$J_{content}(C,G) = \frac{1}{2}\| a^{[l][C]} - a^{[l][G]} \|^2 \longrightarrow$ are they simliar enough?

Style cost function:

The style of a picture is correlation index accross different channels.



correlation means the uniformity probability of their own content is very high.

$\Longrightarrow$ if you measure style cost on layer l:

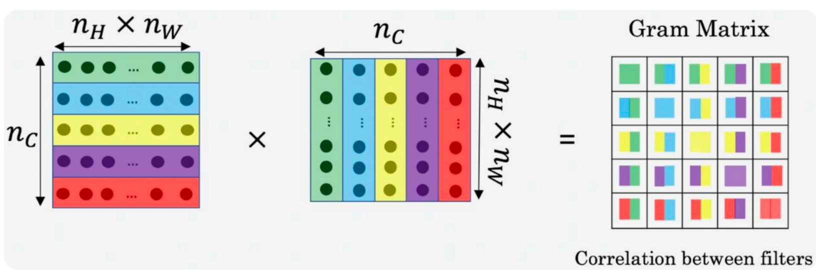
layer L style image

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

H W channel

We obtain: $G^{[l](S)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](S)}_{i,j,k} \cdot a^{[l](S)}_{i,j,k'}$

And $G^{[l](G)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](G)}_{i,j,k} \cdot a^{[l](G)}_{i,j,k}$



$\Longrightarrow$ We finally get: $J^{[l]}_{style}(S,G) = \frac{1}{\beta} \| G^{[l](S)} - G^{[l](G)} \|^2$

Lecture 5: Sequence Model

Recurrent Neural Network:

Notations:

Input: $x^{\langle 1\rangle}, x^{\langle 2\rangle}, \cdots, x^{\langle i\rangle}$

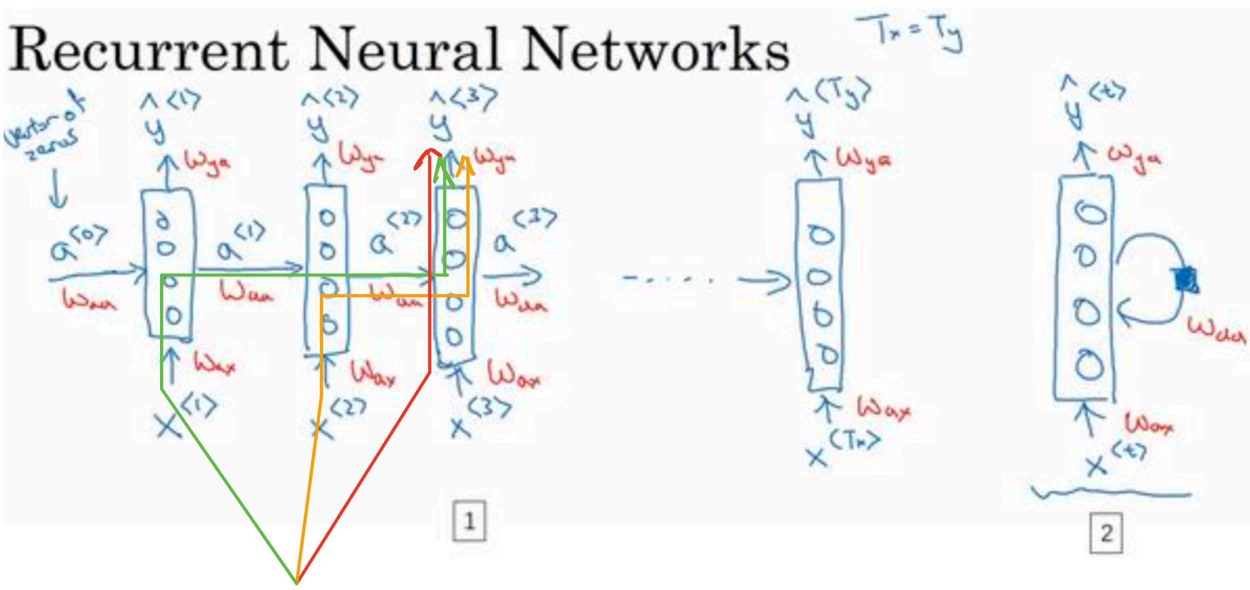
Output: $y^{\langle 1\rangle}, y^{\langle 2\rangle}, \cdots, y^{\langle j\rangle}$

Vocabulary: [a, Aaron, ⋯, and, ⋯, harry, ⋯, potter, ⋯, Zulu]

Index: 1 2 ⋯ 367 ⋯ 4075 ⋯ 6830 ⋯ 10,000

One-hot vector to represent words: lots of zero and an one.

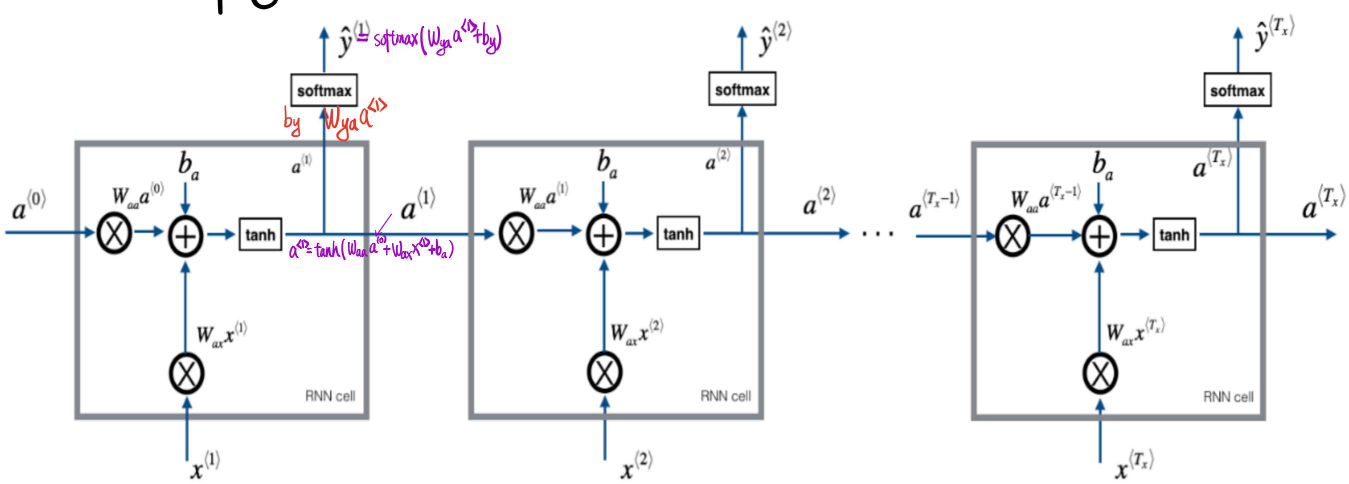
Recurrent Neural Network Model:



$X^{\langle 1\rangle}, X^{\langle 2\rangle}, X^{\langle 3\rangle}$ both effect on $Y^{\langle 3\rangle}$, BUT without other nodes X

⟹ Solution: BRNN (Bidirection)

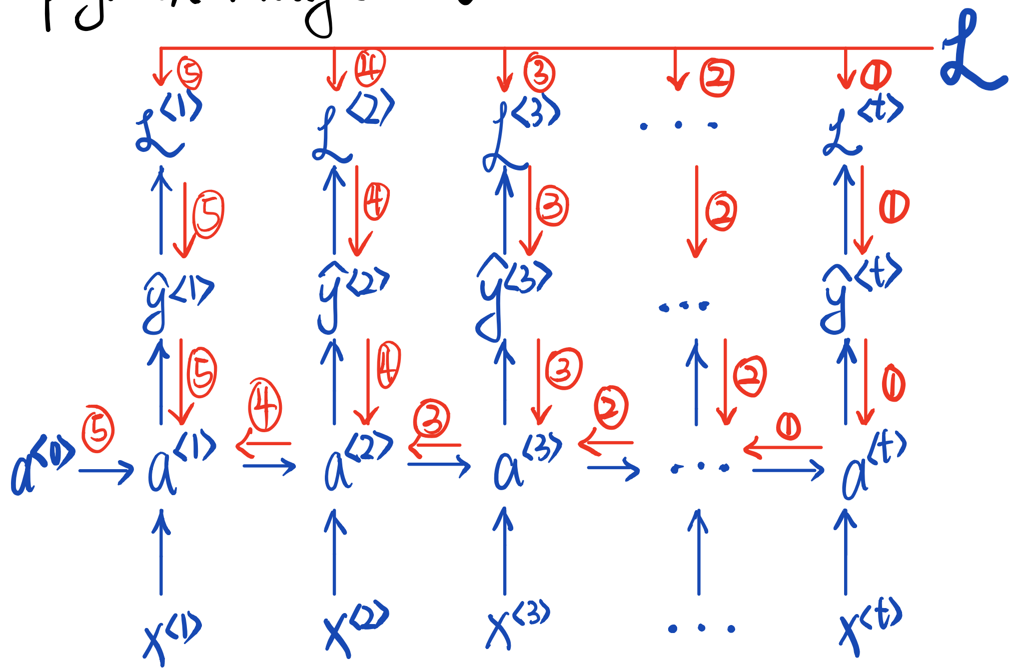
Forward Propagation:



$$a^{\langle i\rangle} = g_1(W_{aa}a^{\langle i-1\rangle} + W_{ax}x^{\langle i\rangle} + b_a)$$
$$y^{\langle i\rangle} = g_2(W_{ya}a^{\langle i\rangle} + b_y)$$

only one!

Backpropagation through time:



complement: $\mathcal{L}^{\langle t\rangle}(\hat{y}^{\langle t\rangle}, y^{\langle t\rangle}) = -y^{\langle t\rangle}\log\hat{y}^{\langle t\rangle} - (1-y^{\langle t\rangle})\log(1-\hat{y}^{\langle t\rangle})$

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T}\mathcal{L}^{\langle t\rangle}(\hat{y}^{\langle t\rangle}, y^{\langle t\rangle})$$

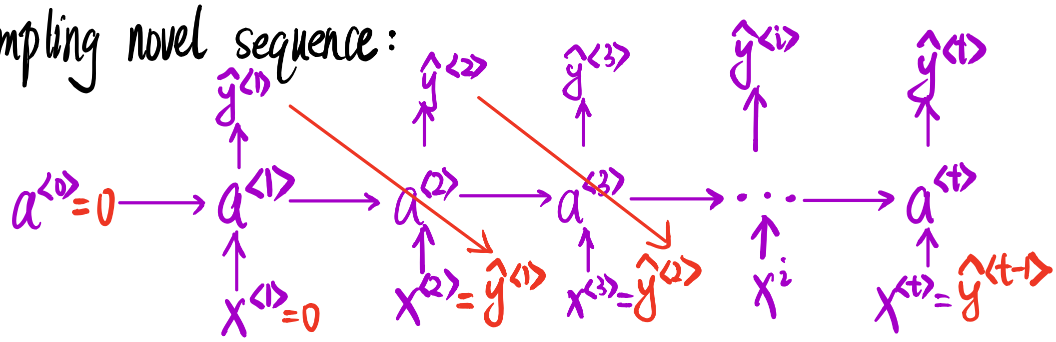
Language model

Language model: decide better sentence among context. according to probability.

P(The apple and pair salad) $= 3.2\times10^{-13}$ ✗

P(The apple and pear salad) $= 5.7\times10^{-10}$ ✓

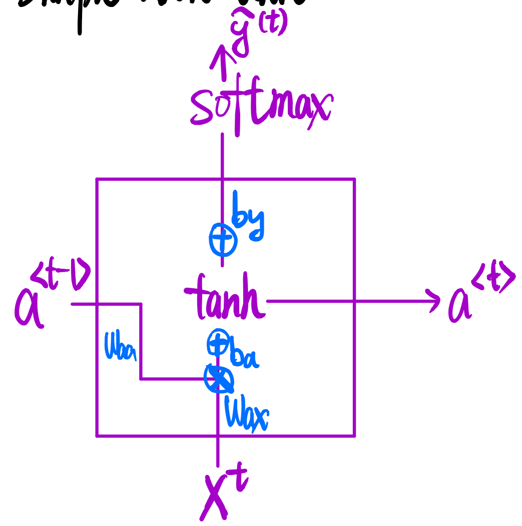
Sampling novel sequence:



Vanishing gradients with RNNs:

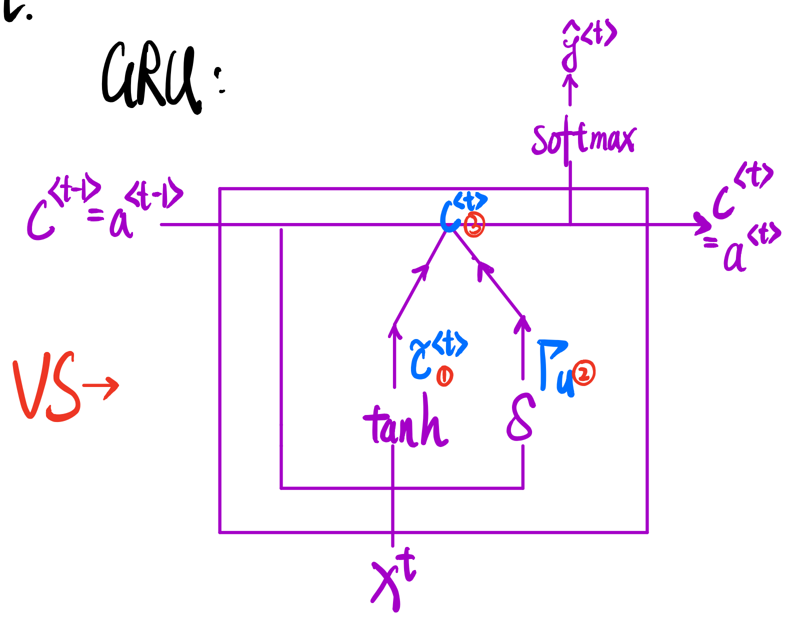
GRU: Gated Recurrent Unit.

Simple RNN Unit



VS→

GRU:



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

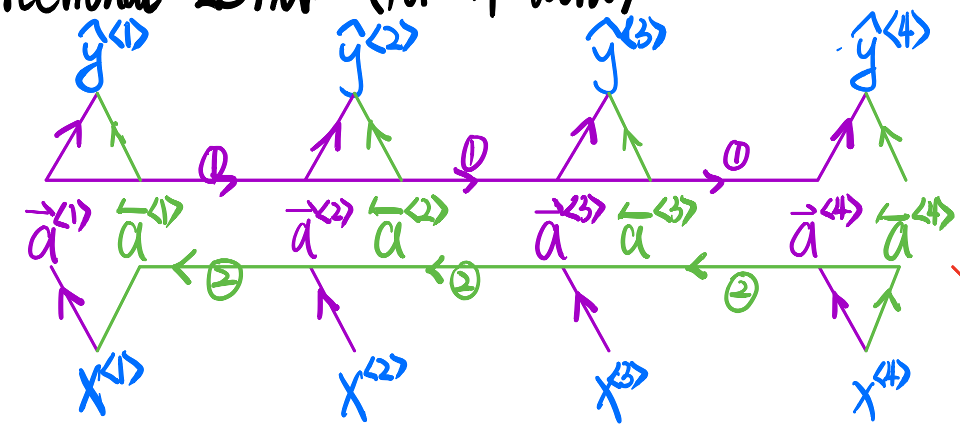
Let ① $\tilde{c}^{\langle t\rangle} = \tanh(W_c[c^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_c]$

② $\Gamma_u = \sigma(W_u[c^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_u]$

⇒ ③ $c^{\langle t\rangle} = \Gamma_u * \tilde{c}^{\langle t\rangle} + (1-\Gamma_u) * c^{\langle t-1\rangle}$

LSTM: Previous Blog Contains this part.

Bidirectional LSTM: (For 4 word)

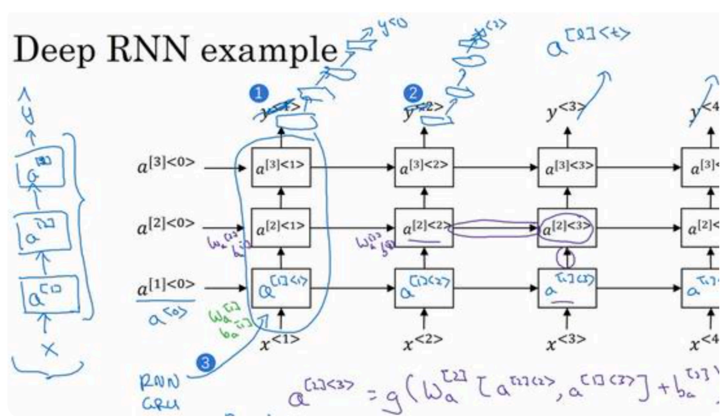


start from $\overleftarrow{a}^{\langle 4\rangle}$

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

Deep RNNs:

Something like this:



Natural Language Processing and Word Embedding:

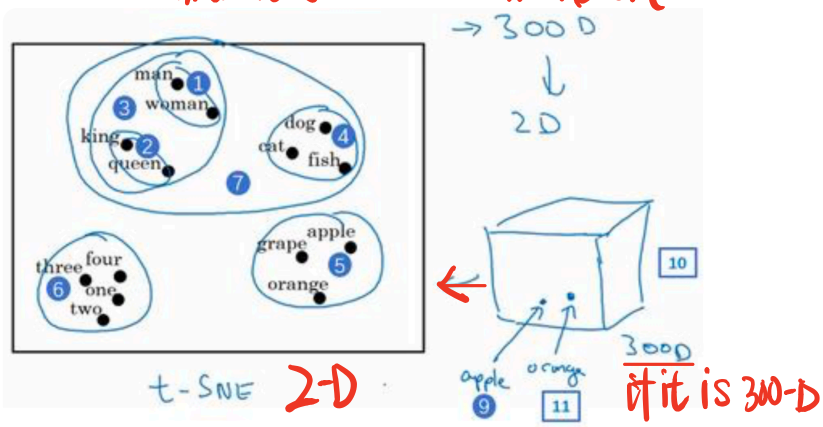
Word Representation:

① Featurized representation: word embedding (to replace one-hot)

	Man	Women	King	Queen	Apple	Orange
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97
⋮	⋮	⋮	⋮	⋮	⋮	⋮

② Visualizing word embedding: t-SNE Algorithm.

From 300-Dimension to 2-Dimension



Properties of Word Embeddings:

① Let $e_{man} = [-1, 0.01, 0.03, 0.09]^T$, $e_{woman} = [1, 0.02, 0.02, 0.01]^T$

$e_{king} = [-0.95, 0.93, 0.70, 0.02]$, $e_{queen} = [0.97, 0.95, 0.69, 0.01]^T$

then $e_{man} - e_{women} \approx [-2, 0, 0, 0]^T$

& $e_{king} - e_{queen} \approx [-2, 0, 0, 0]$

Analogical reasoning: $e_{man} - e_{women} \approx e_{king} - e_?$

So let us define Cosine similarity:

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

Embedding Matrix:

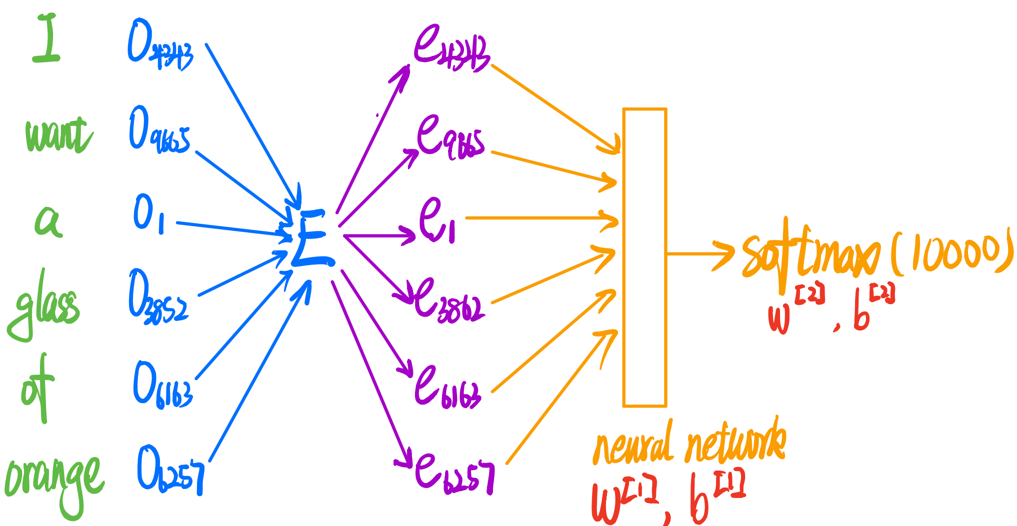
E(Dimmension, Word_num) like E(300, 10000)

Learning Word Embedding:

Neural language Model:

I want a glass of orange ___?___.

index: 4343 9665 1 3852 6163 6257



Other context/target pairs:

I want $\underbrace{\text{a glass of orange}}_{\text{context}}$ $\underbrace{\text{juice}}_{\text{target}}$ $\underbrace{\text{to go along with}}_{\text{context}}$ my cereal.

Context: { Last 4 words.
Last 1 word.
Nearby 1 word.

Word2Vec:

① Skip-Grams Model

abstract context: randomly around target word.

match target word.

if Vocabulary Size = 10,000 k

Context c("orange") ⟶ Target t("juice")

One_hot ⟵ O_c ⟶ E ⟶ e_c ⟶ O_{softmax} ⟶ $\hat{y}$

we need to minimize the softmax loss function:

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

$$L(\hat{y}, y) = -\sum_{i=1}^{10000} \underbrace{y_i}_{\text{(one-hot)}} \log \hat{y}_i$$

② Summary: { CBOW: origin sentence ⟶ target word
Skip-Gram: target word ⟶ origin sentence

Negative Sampling:

New Problem: Give a pair of words (like orange and juice), let us to predict whether they are context-target or not.

{ orange - juice - 1 → positive sample
orange - king - 0 → negative sample.

Input: x		Label: y
context	word	target
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0

K

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

K=5-20 if small datasets

K=2-5 if large datasets

GloVe Word Vectors:

Given X_{ij} is the number of i is in context of j.

After traverse whole corpus. you will found $X_{ij} = X_{ji}$

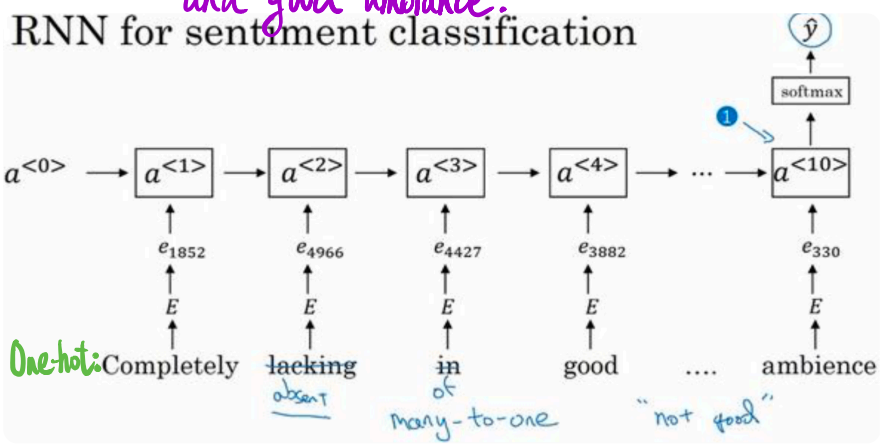
$$\therefore \text{minimize} \sum_{i=1}^{10000}\sum_{j=1}^{10000} f(X_{ij})\left(\theta_i^T e_j + \underset{\uparrow \atop t}{b_i} + \underset{\uparrow \atop c}{b_j'} - \log X_{ij}\right)^2$$

The weight function to prevent huge weight of "a, the, …"

just like t&c, a measurment of how many relationship between them.

Sentiment Classification:

Given a sentence: Completely lacking in good taste, good service and good ambiance.



Sequence model & Attention mechanism.

Various sequence to sequence architectures:

① Sequence to sequence.

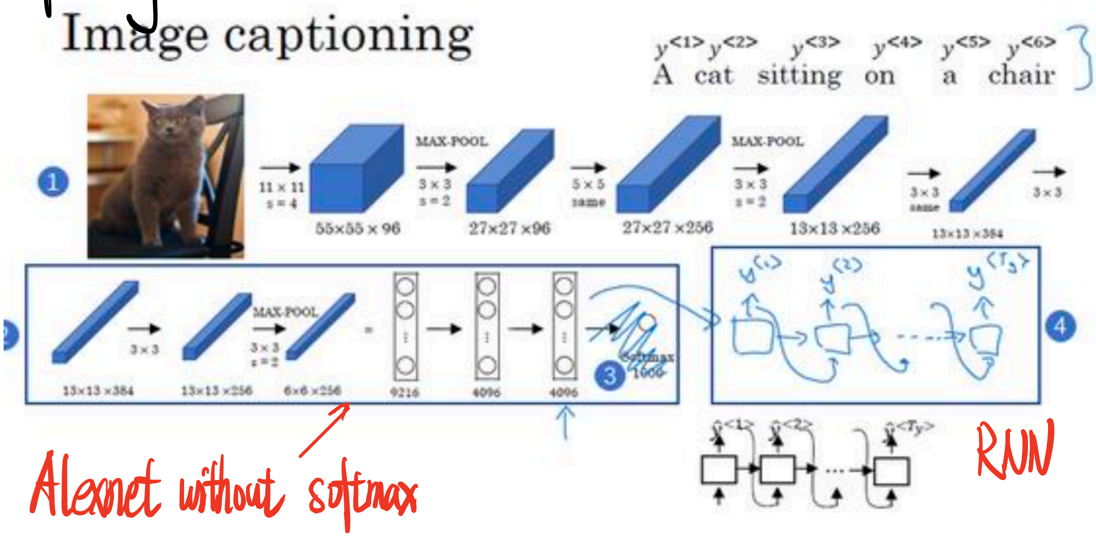
Sequence to sequence model

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ $x^{<4>}$ $x^{<5>}$
Jane visite l'Afrique en septembre

→ Jane is visiting Africa in September.
$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<4>}$ $y^{<5>}$ $y^{<6>}$

RNN (encoder network) → RNN (decoder network)

② Image captioning



Picking the most likely sentences:

Jane visite l'Afrique en septembre.
(French)

→ Jane is visiting Africa in September.

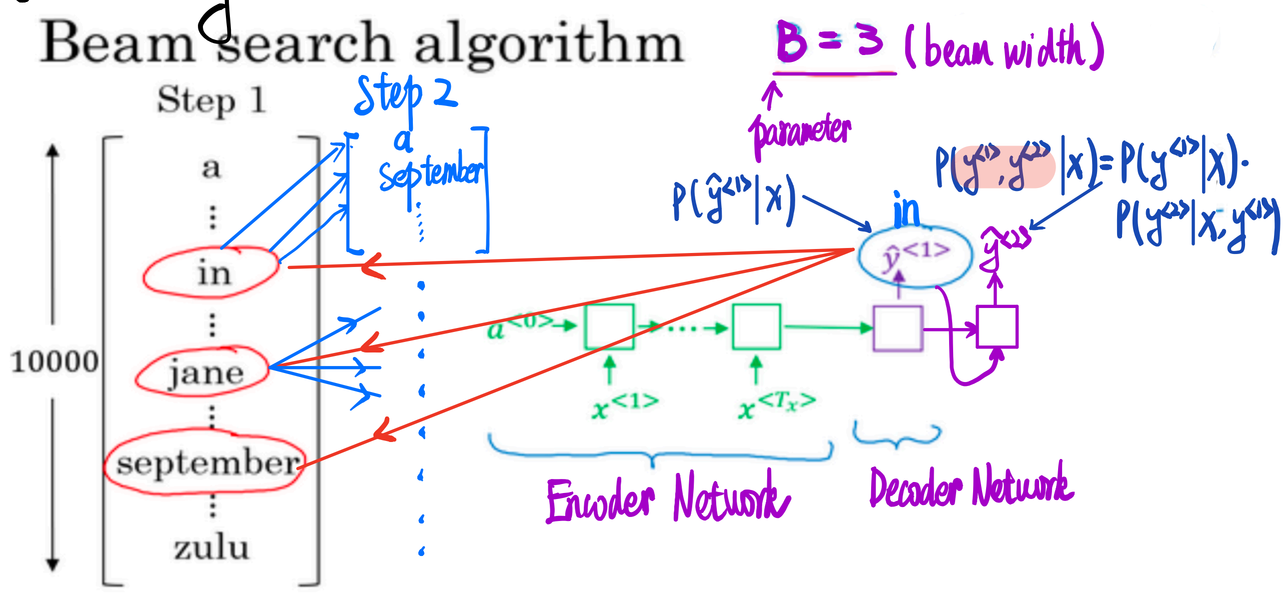
→ Jane is going to be visiting Africa in September.

→ In September, Jane will visit Africa.

→ Her African friend welcomed Jane in September.

$$\arg\max_{y^{<1>},\dots,y^{<t>}} P(\underbrace{y^{<1>},\dots,y^{<t>}}_{\text{English}} \mid \underset{\text{French}}{x})$$

Beam searching:



Output after two step: "in september", "jane is", "jane visits"

Refinements to Beam Search:

① Length normalization:

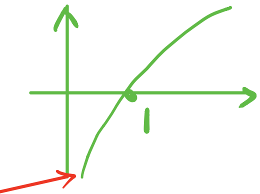
target function can write as:

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

log formation:

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

to prevent numerical underflow:



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

Error analysis in beam search:

① beam search can seem as an approximate search algorithm or a heuristic search algorithm.

② Process:

1' traverse dataset to find errors

2' find why: algorithm or model?

Bleu score: (bilingual evaluation understudy).

Given a translation from machine and autumatically compute a score to measure this translation.

① Bleu score on word:

Evaluating machine translation

French: Le chat est sur le tapis.

Reference 1: The cat is on the mat. — 2 appearance

Reference 2: There is a cat on the mat.

MT output: the the the the the the the.

Precision: $\frac{7}{7}$ Modified precision: $\frac{2}{7}$ ← $Count_{clip}$("the") / ← Count("the")

② Bleu score on bigrams:

like a sliding window

Example: Reference 1: The cat is on the mat.

Reference 2: There is a cat on the mat.

MT output: The cat the cat on the mat.

$$P = \frac{\sum_{unigram} Count_{clip}(Unigram)}{\sum_{Unigram} Count(Unigram)}$$

	Count	Count$_{clip}$
the cat	2	1
cat the	1	0
cat on	1	1
on the	1	1
the mat	1	1

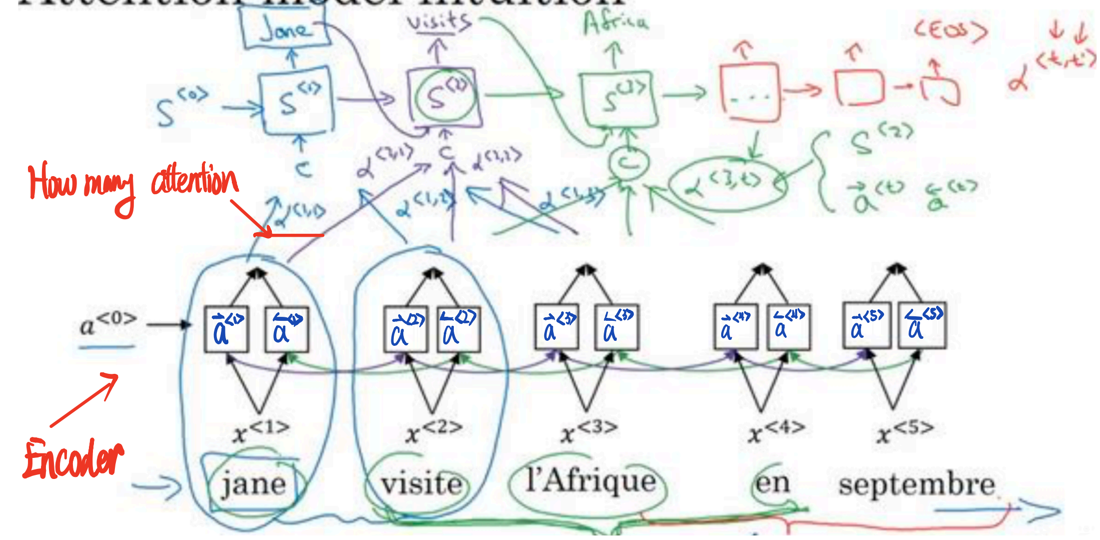
$\frac{4}{6}$

Attention Model Intuition:

① How human-being work?

Only read a part of sentence once a time.

② Attention model intuition



Key point: a set of attention weights: $a^{<i,j>}$ output input

③ Compute attention $a^{<t,t'>}$

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

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