Deep learning

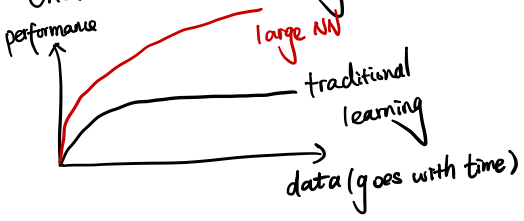
- Neural networks and deep learning

— Coursera
Andrew Ng

Neural networks and deep learning.

Structured data: like an excel list, very well defined meaning of each input

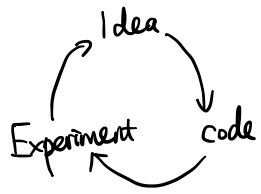
Unstructured data: Images, Audio.



scale drives deep learning progress.

- Data
- Computation
- Algorithms

→ enhance



Binary classification:

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

$$X = \begin{bmatrix} | & | & | & & | \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots & x^{(m)} \\ | & | & | & & | \end{bmatrix} \quad X.shape = (n, m)$$

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

Logistic regression

Given x, want $\hat{y} = P(y=1 \mid x; \theta)$

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

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

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

Computation graph:

$J = 3(a + uv)$

$a = 5$

$b = 3$, $c = 2$ → $u = bc$

$v = a + u$ → $J = 3v$

In code $\frac{d\,\text{Final output}}{d\,\text{var}}$ is represented by 'dvar'

e.g.

x_1, w_1, x_2, w_2, b

$z = w_1 x_1 + w_2 x_2 + b$ → $a = \sigma(z)$ → $\mathcal{L}(a, y)$

$dz = da \cdot (1 - a) a = a - y$

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

For m examples, implement average of gradients and update.

SIMD - single instruction multiple data.

In python, vectorized implementation:

```
for i in range(1000):
    Z = w.T x + b = np.dot(w.T, x) + b
    A = σ(Z)
    dZ = A - Y
    dw = 1/m x dZ^T
    db = 1/m np.sum(dZ)
    w := w - α dw
    b := b - α db
```

Broadcasting in python:

```
cal = A.sum(axis=0)   ↓ 0
percentage = 100*A/cal.
```

(m, n) $\pm$ $(1, n)$ $\sim (m, n)$

$*$ $/$ $(m, 1)$ $\sim (m, n)$

$\mathbb{R}$

use `np.random.rand(5, 1)` this 1 here will make your code easy to understand

```
assert(a.shape == (5,1))
a = a.reshape((5,1))
```

Note:

$p(y|x) = \hat{y}^y (1 - \hat{y})^{(1-y)}$ max p ← maximize likelihood estimation

$\mathcal{J} = -\log p = y \log \hat{y} + (1 - y) \log (1 - \hat{y})$.

min loss

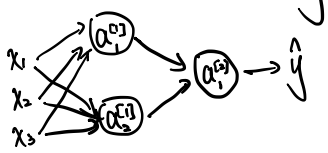
Neural networks and deep learning

Week 3 shallow neural network

$W^{[l]}$ the l layer

$x^{(i)}$ the i the x example

$a^{[0]} = x$ "2 layer NN"



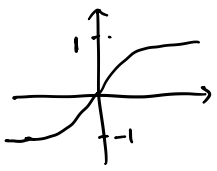
$$\begin{bmatrix} -w_1- \\ -w_2- \\ \vdots \\ -w_n- \end{bmatrix} \begin{bmatrix} x_1^{(1)} & \cdots & x_1^{(m)} \\ x_2^{(1)} & \cdots & x_2^{(m)} \\ x_3^{(1)} & \cdots & x_3^{(m)} \end{bmatrix} = \begin{bmatrix} z_1^{(1)} & \cdots & z_1^{(m)} \\ z_2^{(1)} & & \\ z_n^{(1)} & & z_n^{(m)} \end{bmatrix}$$

horizontal stands for different training example.

vertically stands for different features/nodes

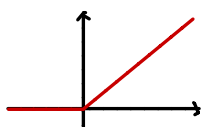
Activation function:

$\tanh = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ tanh is nearly always better than sigmoid.

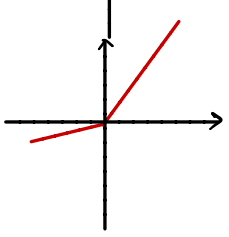


gives mean = 0 activation function

so you can normalize dataset mean to 0



ReLu. $a = \max(0, x)$



leaky ReLu

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

rule: if want to output a probability 0~1. use sigmoid.

normally use ReLu because it learn a lot faster than sigmoid tanh

because the derivative does not go to zero if x very large.

sigmoid < tanh < ReLu ≈ leaky ReLu. way: try them all.

default.

derivative of activation functions:

$g'(z) = g(z)(1 - g(z))$ Sigmoid

$\tanh'(x) = 1 - (\tanh(x))^2$ tanh

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

gradient descent algorithm in python:

$dz^{[2]} = A^{[2]} - Y \rightarrow loss'. g(z)'$

$dw^{[2]} = \frac{1}{m} dz^{[2]} A^{[1]T}$ $\quad w^{[2]} \Rightarrow A^{[1]} \rightarrow z^{[2]}$

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

$dz^{[1]} = w^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$ $\quad x \xrightarrow{w_1, b_1} a_1 \xrightarrow{w_2, b_2} a_2$

$dw^{[1]} = \frac{1}{m} dz^{[1]} x^T$

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

Random initialization:

to zero will not wok. break symetric.

$w^{[1]} = np.random.randn(2,2) * 0.01$

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

Week 4 deep neural network.

Input is not a layer.

$l = 4$ (layers)

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

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

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

$w^{[l]}$ weight for $z^{(l)}$

$n^{[0]} = n_x$ $x = a^{[0]}$

$a^{[l]} = y$

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

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

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

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

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

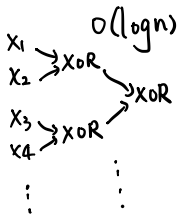
use a for loop to loop through all code.

getting matrix dimension right.

Curcuit theory and deep learning

x_1 xor x_2 xor x_3 xor ---- x_n

$O(\log n)$

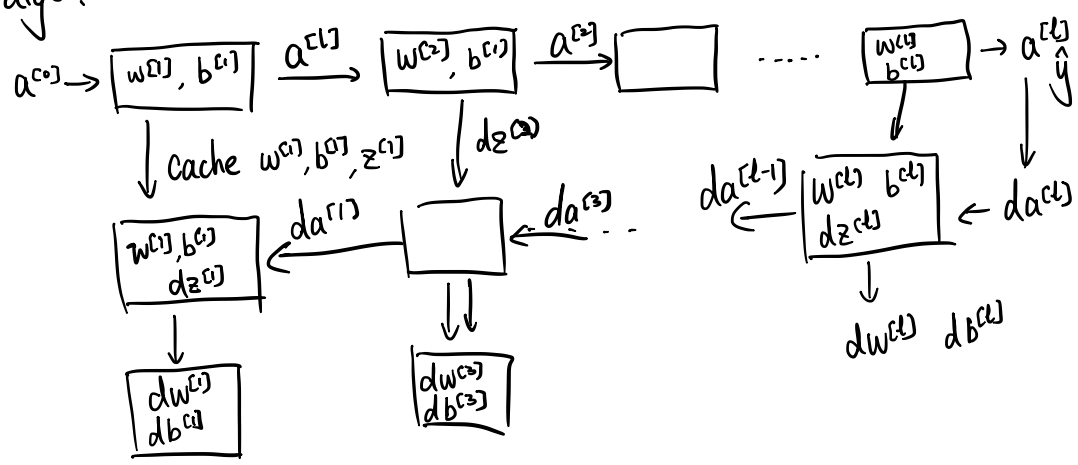


one hidden layer

$O(2^n)$ grows exponentially!!

FW BW.

algo:



Implementation:

Forward prop

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

Output $a^{[l]}$, cache $[z^{[l]}]$

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

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

Backward prop for layer l

Input $da^{[l]}$

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

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

$$dw^{[l]} = \frac{1}{m} dz^{[l]} \cdot A^{[l-1]T}$$

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

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

if sigmoid function at last layer.

$$da^{[l]} = \begin{bmatrix} -\frac{y^{[i]}}{a} + \frac{(1-y)^{[i]}}{1-a} \\ \vdots \end{bmatrix}$$

Applied deep learning is a very emperical process.

Improving Deep Neural Networks:

Hyperparameter tuning.

Regularization and

Optimization

- Andrew Ng

Setting up Machine learning application.

60% training set 20% valid set 20% test test small dataset

large dataset

1,000,000 train 10,000 valid 10,000 test

make sure dev/test set from same distribution.

trainning data can be download from internet.

Bias variance.:

training set error:	1%	15%	15%	0.5%
Dev set error:	11%	16%	30%	1%
	high variance	high bias	high bias	low bias
			high variance	low variance

Optimum (Bayes) Error: 0%

In some ways / in other aspect/regions

(training Performance)

High bias → (Y) · Bigger Network · train longer · (NN architecture search)

(N) ↓

High variance? (dev set performance) → (Y) More data / Regularization (less trade off here, just compute time) / (NN architecture tools)

(N) ↓

Done

Regularization:

L2 regularization: $\|w_2\|^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$

L1 regularization: $\frac{\lambda}{2m} \sum_{j=1}^{n_x} |w_j| = \frac{\lambda}{2m} \|w\|_1$ w will be sparse. a lot of zeros

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

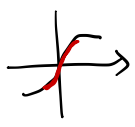
$$\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l-1]}} \sum_{j=1}^{n^{[l]}} (w_{ij}^{[l]})^2 \quad \text{"Frobenius norm"}$$

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

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

regularization: ① use all the weights, each weights

② activation will be more linear,



more linear assumption.

Dropout regularization:

Implement dropout:

```
d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep-prob
a3 = np.multiply(a3, d3)
a3 /= keep-prob.
```

also zero out at back.prob

No dropout at test time!

Intuition: Can't rely on any feature. so have to spread out weights.

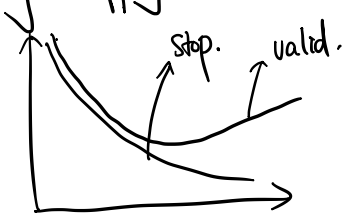
→ compute vision. apply a lot. because have a lot of inputs and each works.

When plotting J(θ) first set keepdim=1 to find J is decreasing

because dropout may affect J.

other technics:

1. Data augmentation
2. early stopping



bad side of early stopping:

Orthogonalization first:

— Optimize cost Function — Gradient descent / momentum

— overfitting solving — R_2 regularization

but early stopping make us to do these two steps together.

Or we can set a good R_2 regularization parameter λ.

So we can train as much as we want.

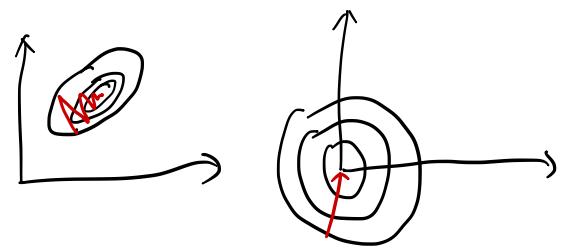
but need to try several times.

faster your tranning:

Normalizing trainning set. Store μ_{train} σ_{train}

and apply it on test set.

Why normalize:



Vanishing / exploding gradient.

when Initialize W, random initialize weights.

$[W] \overset{L}{=} \chi$ grows exponentially.

Random initialization:

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

$np.sqrt(\frac{1}{n^{[l-1]}})$ tanh (Xvaier initialization)

$np.sqrt(\sqrt{\frac{2}{n^{[l]}+n^{[l-1]}}})$ (also can)

checking derivative: numerical checking.

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

turn $[W_1\ b_1 \cdots W_n\ b_n] \rightarrow [\theta_1\ \theta_2 \cdots \theta_n]$

for each i:

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

$\varepsilon = 10^{-7}$

check $\frac{\|d\theta - d\theta_{approx}\|_2}{\|d\theta\|_2 + \|d\theta_{approx}\|_2} \approx 10^{-7}$ – correct!

10^{-5}

10^{-3} – worry.

notes:

- Don't use in training - only to debug
- If algorithm fails, debug.
- Remember regularization
- Doesn't work with dropout keep dims = 1.
- (Run at random initialization, check again after some traning) less common.

Week2: optimization algorithm:

Batch vs mini batch gradient descent.

$X^{\{t\}}, Y^{\{t\}}$ ⇒ first + mini batch.

minibatch - run faster

minibatch 1 → stochastic gradient descent: longer traning time.
update parameter each time. make progress everytime.

choose minibatch size:

If small training set ≤2000: use batch gradient descent.

Typical minibatch sizes: 64, 128, 256, 512,

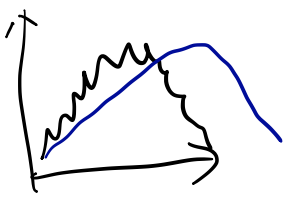
depend on GPU.

Exponentially weighted averages

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

average over steps: $\frac{1}{1-\beta}$.

β bigger, the less adaptive. got latent changes



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

$(1-\beta) + (1-\beta)\beta + (1-\beta)\beta^2 + \cdots \approx 1.$

$0.9^{10} \approx \frac{1}{e}$ $0.98^{50} \approx \frac{1}{e}$

memory saving.

$V_\theta := 0$

$V_\theta := \beta V + (1-\beta)\theta_1$

⋮

bias correction in initial phase:

$V_\theta = \frac{(1-\beta)V_{\theta-} + \theta_t}{1-\beta^t}$ (usually do not bother do it)

$\beta \geq 0.9$

Gradient descent with momentum.

can get off local min.

On iteration t:

Compute dW, db, on current mini batch

$V_{dw} = \beta V_{dw} + (1-\beta) dW$ → have this term all need to rescale α everytime you change β

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

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

RMS Prop:

On iteration t:

Compute dW, db on current minibatch:

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

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

$W := W - \alpha \frac{dW}{\sqrt{S_{dw}} + \varepsilon}$ $b := b - \alpha \frac{db}{\sqrt{S_{db}} + \varepsilon}$ → 10^{-8}.

slow learning rate when gradient very large

for numeric steability.

Adam optimization: Adaptive moment estimation

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

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

On iteration t,

Compare dw, db using correct minibatch

$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dW$, $V_b = \beta_1 V_{db} + (1-\beta_1) d_b$

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

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

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

$W := W - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}} + \varepsilon}$ $b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon}$

α: need to tune

$\beta_1 = 0.9$ (dw)

$\beta_2 = 0.999$ (dw^2)

$\varepsilon = 10^{-8}$.

learning rate decay:

1. $\alpha = \frac{1}{1 + \text{decay\_rate} * \text{epoch num}} \alpha_0$

2. $\alpha = 0.95^{\text{epochnum}} \alpha_0$

3. $\alpha = \frac{k}{\sqrt{\text{ep-num}}} \cdot \alpha_0$ or $\frac{k}{\sqrt{t}} \alpha_0$

4. 

5. Manual decay.

Local optima in neural network:

most saddle point.

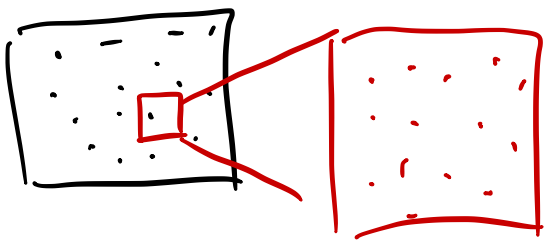
hyperparameters tuning:

- α
- $\beta \sim 0.9$
- # hidden units
- # minibatch size
- layers
- learning rate decay.

↓ decrease importance.

depend on case

random parameter, do not use grid..



Appropriate scale for hyperparameters

```
r = -4 * np.random.rand()
α = 10^r
```

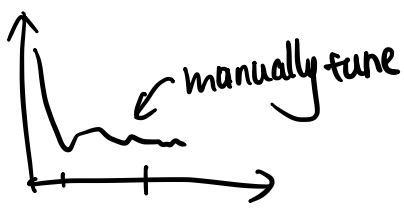
← log scale.

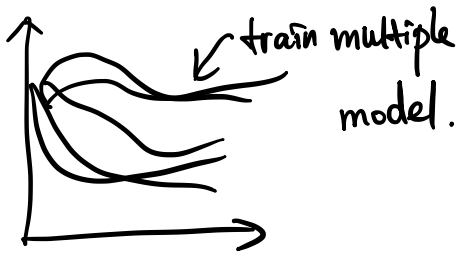
$\beta = 0.9 \cdots 0.9999$

$1-\beta = 0.1 \cdots 0.001$.

↑ focus on.

One model





Batch normalization:

usually normalize z[i]

Implement:

Given some intermediate values in NN.

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

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

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

$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$ ← learn γ & β. use nonlinearity better

Parameters: $w^{[1]}, b^{[1]} \cdots w^{[n]}, b^{[1]}$

$\beta^{[1]}, \gamma^{[1]} \cdots \beta^{[n]}, \gamma^{[n]}$

tf.nn.batch_normalization.

if use batch norm, we can ignore $b^{[l]}$.

Implementing grading descent

for t=1.... num MiniBatches.

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

In each hidden layer, use BN replace $z^{[l]}$ with $\tilde{z}^{[l]}$

Use back prop to find $dw^{[l]}, d\beta^{[l]}, d\gamma^{[l]}$

updata parameters

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

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

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

This works with adam/momentum as well.

Why batch norm work?

① combat covariance shift of each layer, each nuual network can still perform task independently even though the input from previous layer change.

② regularization effect:

each minibatch is scaled by mean/variance computed on just that minibatch.

This add noise to value $\tilde{z}^{[l]}$ within that minibatch.

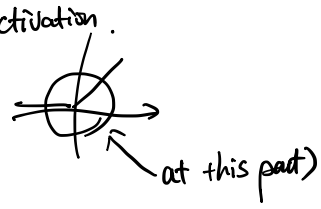
So similar to dropout. it slightly force nn not to rely on individual models.

(bigger minibatch → smaller regularization)

(③ make nn to make better use of activation.

↓

morvan taught instead of Ng.



Batch Norm at test time:

keep track of moving average μ and σ^2. in trainning time and use it at test time.

multi class classification:

Soft max layer:

Activation function.

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

$a^{[l]} = \frac{e^{z^{[l]}}}{\sum_i t_i}$ → if only one layer, softmax decision boundary will be linear.

Softmax is soft hard max.

logistic is one example of softmax.

$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{\#\text{category}} y_j \log \hat{y}_j$$

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

Back prop: $dz^{[l]} = \hat{y} - y$ ← 4 by 1 vector. # category = 4.

Deep learning frame waks:

- Caffe/caffe2
- CNTK
- DL4J
- keras
- Lasagne
- mxnet
- PaddlePaddle
- Tensorflow
- Theano
- Torch.

choosing criteria:

1. easy to code (deploy)
2. run time.
3. open source (in the future) maintenance.

Tensorflow:

see code

+-* / is overloaded in tensorflow so they are = tf.add...

makes your code cleaner.

Structuring machine learning Project

Andrew Ng.

ML strategy

Orthogonalization:

separately each factor that only will affect one factor/function of algo.

Chain of ML assumptions

Fit training set well on cost function → bigger network / Adam
↓ (≈ human level performance)
Fit dev set well on cost function → regularization / bigger training set
↓
Fit test set well on cost function → bigger dev set
↓
real world performance well. → change dev set or cost Function.

Single number evaluation metric. (at start project)

So we can pick out a better solution.

e.g. recall. precision. F1 score.

speed up iterating.

How to set evaluation metric.

1 Optimizing metric : accuracy
N-1 satisfying metric : running time.

set dev/test sets:

Come from same source. can reflect real world problem. test. you care about.

size of test set:

set your test set to be big enough to give high confidence in overall performance of your system.

- When to change dev/tests sets and metrics.

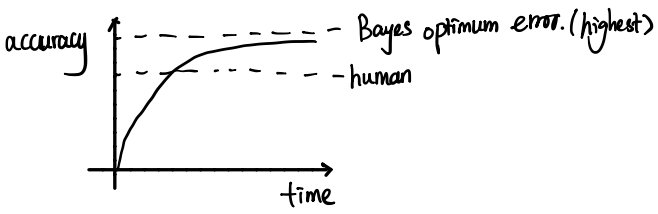
eg. pornagraphic images weight more. in loss Function.

define new metric perform your favourite.

1. set metric to evaluate classifiers (do not worry it is not very accurate, set early)
2. Worry separately about how to do well on this metric.

$$J=\frac{1}{\sum w^{(i)}}\sum_{i=1}^{m} w^{(i)}\mathcal{L}(\hat{y}^{(i)},y^{(i)}) \qquad w^{(i)}=\begin{cases}1. \text{ norm image.}\\ 10, \text{ image you do not want.}\end{cases}$$

Human-level performance.



Why compare to human level performance:

so long as ML is worse than humans:

- Get labelled data from humans
- Gain insight from manual error analysis
- Better analysis on bias/variance.

bias or not?

humans (≈ Bayes)	1%	7.5%	↕ avoidable bias
Training error	8%	8%	↕ variance
Dev error	10%	10%	
	focus on bias	focus on variance.	

Human level as a proxy for Bayes error.

after surpass human level performance. (team of experts)

- Online advertising
- Product recommendation
- logistic (predict transit time)
- Loan Approvals.

↑ Structured data.
Not natural perception
lots of data

- speech recognition
- Some image recognition
- material. ECG. skin cancer.

Bayes error (human)
↕ avoidable bias → bigger model / train longer optimization algo / NN architecture/hyperparameter search.
Training error
↕ variance → more data / regularization – L2, dropout, data augmentation, etc / NN architecture, hyperparameter search.
Dev error

machine learning idea:

Flight simulator: train with scenarios that will take time to encounter in real experience

error analysis:

Look at dev examples to evaluate ideas:

Evaluate multiple ideas in parelled.

Ideas for cat detection:

- Fix pictures dogs being recognized as cats
- Fix great cats (lions, panther) misclassification
- Blurry images affect performance.

draw a sheet of error!

Image	Dog	Great cats	Blurry	Comments
1	✓			pitfall.
2		✓	✓	
of total	8%	43%	61%	12%

Incorrect labels:

DL algorithms are quite robust to random errors in training set.

it is not robust to systematic errors in training set.

If In test test/valid set: add another column in sheet of errors to see how much.

Is it worth while to go and fix incorrect labels:

(whether incorrect labels makes up a large percent of errors)

Correct incorrect dev/test set:

- apply same process to your dev and test sets to make sure they continue to come from same distribution.
- Consider what you got right as well as you got wrong.

(Sometimes only look at what you got wrong make you over feature engineering and turn some correct cases to wrong)

- train & dev/test sets now may come from different distribution.

Build your first system quickly, and iterate.

speech recognition example:

ways to improve:

- Noisy background
  - different noises (car)
- accented
- Far from microphone.

work flow:

① Set up dev/test set and metric

② build initial system quickly

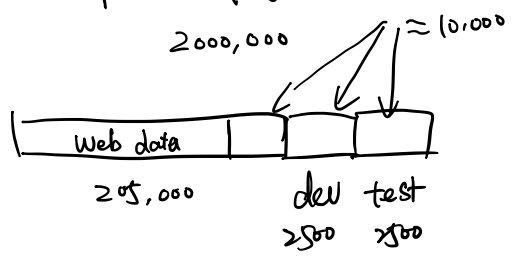
③ Use bias/variance analysis & Error analysis to prioritize next steps.

Mismatched training and dev/test set:

training and testing on different distribution:

Data from other source (big) Data matters (small)

Data from webpage: Data matters:

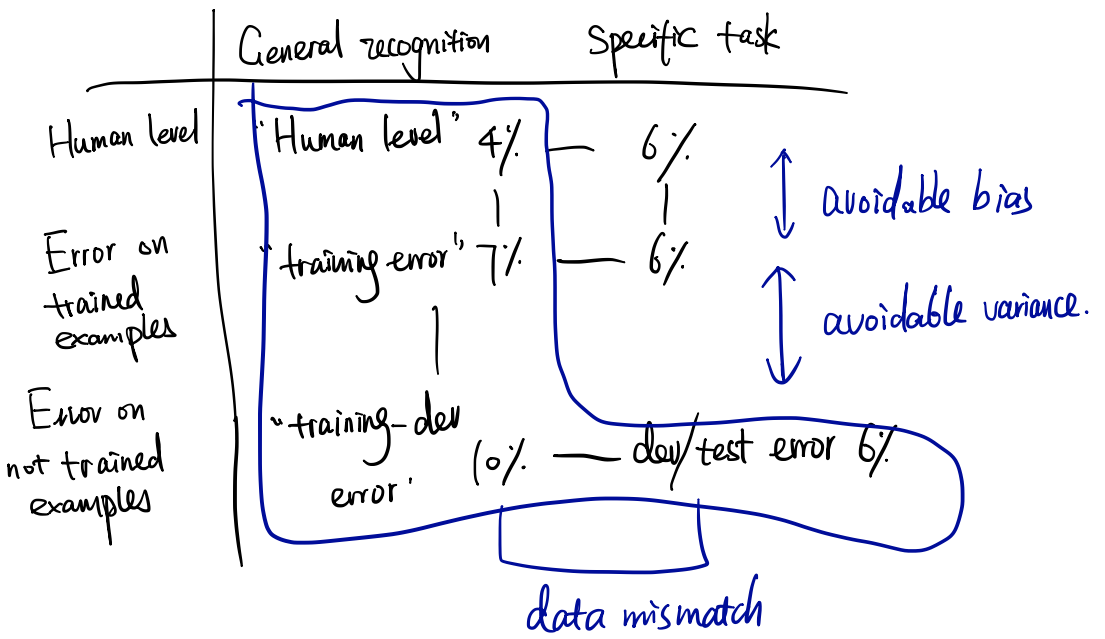


if they come from different distribution:

training-dev set: apart in training set your training data is test on. (you do not use in training)

to see whether it is high bias or data mismatch.

sheet of analysis



Addressing data mismatch:

- carry out manual error analysis to find difference between training & dev/test sets.
- Make training data more similar. or recollect data.

Artificial data analysis

(make sure the way you synthesize data, do not overfit)

It is hard to know sometimes.

Transfer training:

When transfer learning make sense:

- Task A and B have same input. type.
- A have more data than B.
- A and B shares common small features.

Multiclass learning:

assign multiple labels to one example.

sometimes this perfoms better than single class.

When multiclass make sense:

- training on a sets of task that could benefit from shared-low level features.
- Usually: amount of data you have for each task is similar.
- can train big enough network do well on all tasks.

(usually used in object detection)

End to End deeplearning:

speech recognition example:

x → (MFCC) features → (ML) phonemes → words → y: Script

audio → transcript

training size matters a lot.

Pros and cons of end-to-end deep learning

Pros:

- let data speak.
- Less hand-designing of compenent needed.

Cons:

- need a large data. (have enough data for subtask)
- cannot add hand designed features.

Should I try end2end:

key question: do you have sufficient data to learn a function of complexity needed to map x to y?

Convolutional NNs

—Andrew Ng

CNN:

tf.nn.conv2d

keras : Conv2D

Convolutional operation : $*$ (cross-correlation)

Edge detector:

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

sobel filter for vertical line:

if there is a line : extreme larger or extrem negative

Padding:

- shrink output
- throw away info from edge (there is not over lapping at edge)

padding zeros.

valid convolution (no padding):

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

same: pad to make size output same

$\frac{p-1}{2}$. p is usually odd.

stride convolution:

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

Convolution over volume:

e.g.

$n \times n \times 3 * f \times f \times 3 \rightarrow n-f+1 * n-f+1 \times n_c'$ ← # filters.

CNN layer:

$\text{ReLu}(\boxplus + b)$

notation:

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

$p^{[l]}$ = padding size

$s^{[l]}$ stride

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

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

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

order is not consistent in different networks

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

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

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

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

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

Pooling layer:

Max pooling $\lfloor \frac{n_H-f}{s}+1 \rfloor \times \lfloor \frac{n_H-f}{s}+1 \rfloor$

$5 \times 5 \times n_c \rightarrow 3 \times 3 \times n_c$

↓ does not change

No parameter to learn.

see hyperparameters works for others will also wok for you.

why CNN wok:

Parameter sharing

Sparsity of connections

Classic Networks:

LeNet -5

$(32\times32\times1) \xrightarrow[s=1]{5\times5} (28\times28\times6) \xrightarrow[s=2]{\text{avg pool } f=2} (14\times14\times6) \xrightarrow[s=1]{5\times5} (10\times10\times16) \xrightarrow[s=2]{\text{avg pool } f=2} 5\times5\times16$

$\xrightarrow{FC} 120 \xrightarrow{Fc} 84 \xrightarrow{\text{softmax}} 10\ \hat{y}$

$n_H, n_W \downarrow \quad n_C \uparrow$

conv pool conv pool fc fc output

activation (sigmoid) at that time.

AlexNet

$227\times227\times3 \xrightarrow[s=4]{11\times11} 55\times55\times96 \xrightarrow[s=2]{\text{MaxPool } 3\times3} 27\times27\times96 \xrightarrow[5\times5]{\text{same}} 27\times27\times256$

$\xrightarrow[s=2]{\text{MaxPool } 3\times3} 13\times13\times256 \xrightarrow[\text{same}]{3\times3} 13\times13\times384 \xrightarrow{3\times3} 13\times13\times384 \rightarrow 13\times13\times256 \xrightarrow[s=2]{\text{Max Pool } 3\times3} 6\times6\times256$

$= 9216 \xrightarrow{FC} 4096 \xrightarrow{FC} 4096 \xrightarrow{\text{Softmax}} 1000$

— Similar to Lenet, but much bigger ~60M params

— Relu

VGG-16

conv 3×3 filter s=1 same Max-pool =2×2, s=2

$224\times224\times3 \xrightarrow{\text{conv64 }\times2} 224\times224\times64 \xrightarrow{\text{pool}} 112\times112\times64 \xrightarrow{\text{Conv64 }\times2} 112\times112\times128$

$\xrightarrow{\text{pool}} 56\times56\times128 \xrightarrow{\text{conv256 }\times3} 56\times56\times256 \xrightarrow{\text{pool}} 28\times28\times256 \xrightarrow{\text{Conv 512 }\times3} 28\times28\times512$

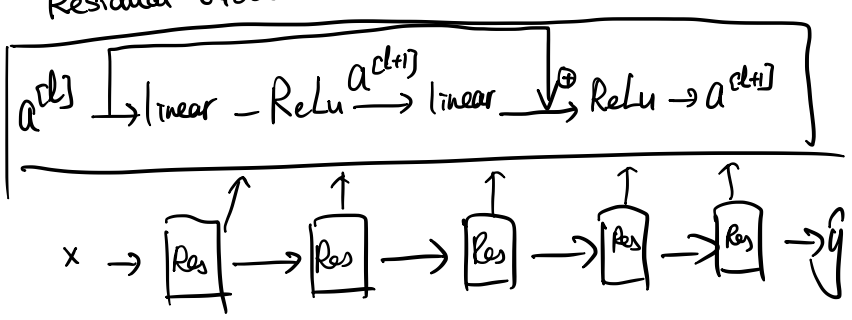
$\xrightarrow{\text{pool}} 14\times14\times512 \xrightarrow{\text{conv512}\times3} 14\times14\times512 \xrightarrow{\text{pool}} 7\times7\times512 \rightarrow FC \rightarrow FC \rightarrow \text{Softmax}$

$4096 \quad 4096 \quad 1000$

~138M.

ResNets:

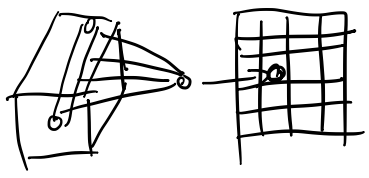
Residual blocks:



identify function is easier for Residual block to learn.

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

1×1 Conv do?

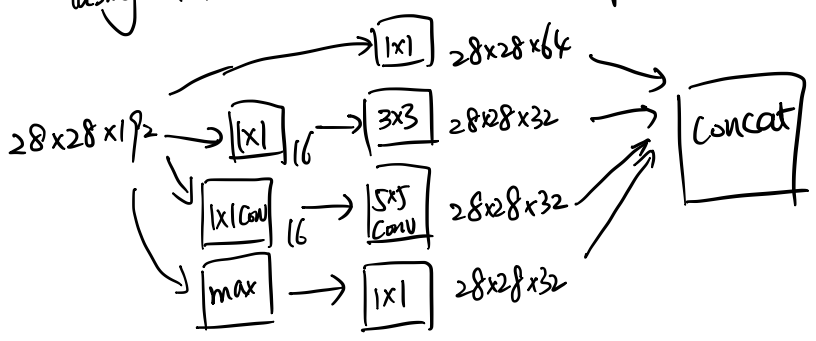


network in networks.

add linearity

Inception network:

using 1×1 convolution to reduce parameters to $\frac{1}{10}$ M.



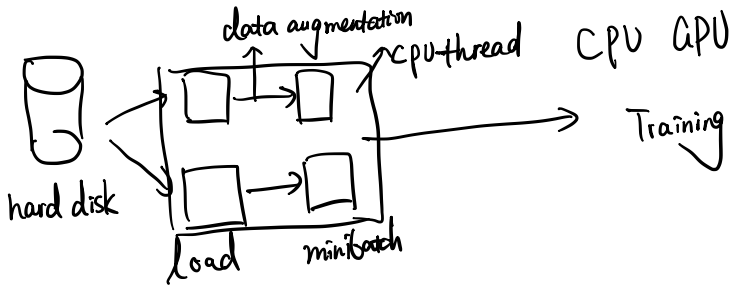
(also side output, to regularize and make sure intermediate layers do not go wrong too much)

Transfer learning:

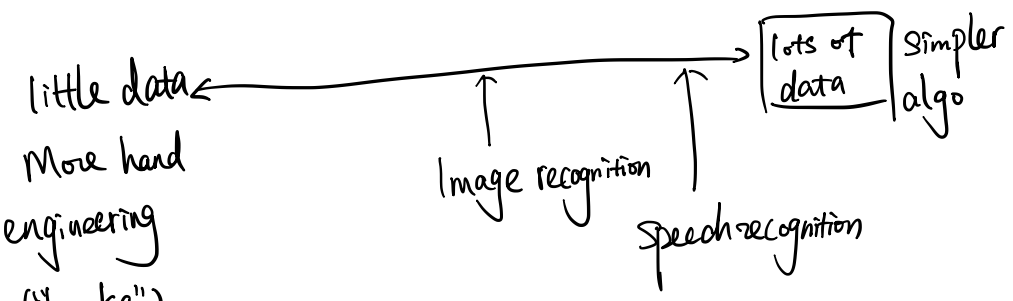
if data really large: can freeze more modules.

Data augmentation:

Color shifting, mirroring. "PCA color augmentation".



Data vs hand-engineering



Two source of knowledge:

① labelling

② complex nn and hand engineering feature.

Tips on breaking bench marks:

- Ensembling: train several network independently and average their output.
- Multi crop at test time.
  test data augmentation.

object localization:

output

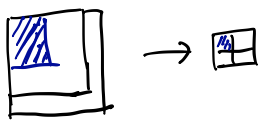
$y = \begin{bmatrix} p_1 \\ p_x \\ p_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$ $p_1, p_x, p_y \rightarrow$ logistic loss. $b_h, b_w \rightarrow$ squared error $c_1, c_2, c_3 \rightarrow$ Log likelihood loss

Same nn.

landmark detection.

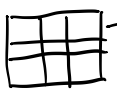
object detection:

- sliding window detection.
- Turn FC into convolutional layers:



make predictions convolutionally.

YoLO algorithm:



→ assign a classification label $\begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$ 8 to one cell.

(b_h, b_w can be bigger than 1)

Intersection over Union (IoU): "correct if $\frac{\text{detected area has object}}{\text{cell area}} \geq 0.5$.

Non max suppression:

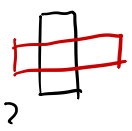
① discard < 0.5 p.

while there is cell:

① highlight the biggest probability

② discard ones have high overlap. with the first one.

anchor box: have two pair.



choose window box by kmeans to cover all objects?

Region proposal:

segmentation algorithm

R-CNN: Propose region, Classify proposed region one at a time.
Output label + bounding box

Fast-RCNN: Propose region, Use convolution implementation of sliding windows to classify all proposed regions.

Faster-RCNN: Use convolutional neural network to propose regions.

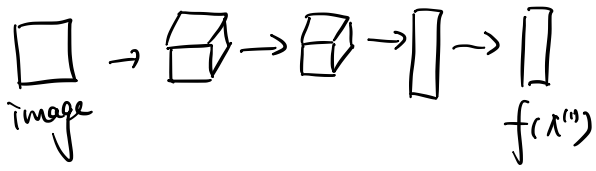
Face verification:

one shot learning: have only 1 training image.

learn $d(\text{image 1}, \text{image 2}) \leq \tau$. is some body.

↓

Siamese network



If $x^{(i)}, x^{(j)}$ is same person, $\|f(x^{(i)}) - f(x^{(j)})\|$ is small.

Triplet loss function:

Given 3 images Anchor. positive. negative.

$$\mathcal{L}(A, p, N) = \max(\|f(A) - f(p)\|^2 - \|f(A) - f(N)\|^2 + \underbrace{\alpha}_{\text{margin.}}, 0)$$

training set. 10k pictures of 1k person.

choose triplets that are "hard" to train on. (paper: FaceNet).

$$\hat{y} = \sigma\left(\sum_{k=1}^{128} w_i \left|f(x^{(i)})_k - f(x^{(j)})_k\right| + b\right) \qquad \hat{y}: 0 \text{ True}, 1 \text{ False}$$

or

$$\frac{\left(f(x^{(i)})_k - f(x^{(j)})_k\right)^2}{f(x^{(i)})_k + f(x^{(j)})_k} \quad \chi^2.$$

Neural transfer:

[Zeiler and Fergus 2013. Visualizing and understanding CNNs]

↓ ways to visualize each layers of CNN.

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

[Gatys et al. 2015. A neural algorithm of artistic style

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

Content C Style S

↓

Generate image G

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

Style matrix:

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

$G^{[l]}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l]}_{ijk}\, a^{[l]}_{ijk'}$ covariance.

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

Convolution in 1D.

ECG signal using 1D filter.

3D conv. on CT scan.

movie data using 3D filter.

Sequence model

- Andrew Ng cousera

RNNs

Notation

$x^{(i)<t>}$ The t-th word in training sample i

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

$T_x^{(i)}$ The length of ith training example.

$T_y^{(i)}$

<UNK> for word not in the model.

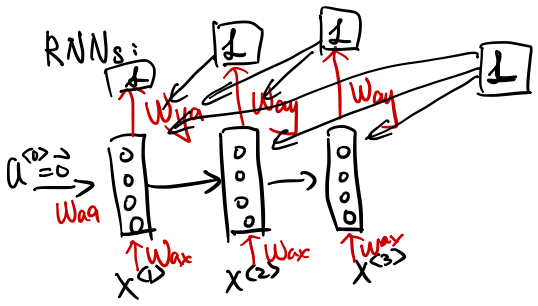
Why not standard network:

Problems:

Inputs, outputs can input use various length.

dont share weights across different position of text.

RNNs:



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

$a^{<1>} = g_1(W_{aa}a^{<0>} + W_{ax}x^{<1>} + b_a)$ ← tanh. ReLu

$\hat{y}^{<1>} = g_2(W_{ya}a^{<1>} + b_y)$ ← can be sigmoid depend on output.

$a^{<t>} = g_1(W_a[a^{<t-1>}, x^{<t>}] + b_a)$

$\hat{y}^{<t>} = g_2(W_y a^{<t>} + b_y)$ ← Sigmoid if output 0/1

Backprop through time (BPTT)

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

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

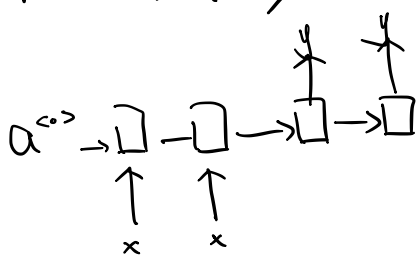
Different RNNs

Many-to-Many (NER machine-translation)

Many-to-one (movie Review)

one to one (standard neural network)

one to Many (music generation)



language model and sequence generation.

P(sentense) $P(y^{<1>}, y^{<2>}, \ldots y^{<T_y>}) <EOS>$

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

$y^{<n-1>} = x^{<n>}$

Sampling novel sequences. feed any random word to next cell to create a novel sentence.

Vanishing gradient / exploding gradient — gradient clipping, clip over a threshold.

GRU (Gated Recurrent unit)

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

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

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

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

$\Gamma \approx 0$. does not contribute very much to

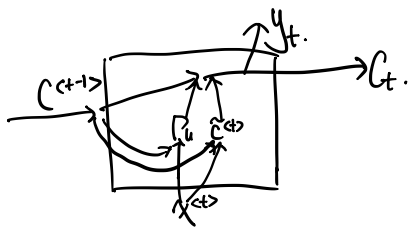
or the full version:

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

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

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

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



papers:

Cho et al. 2014 On the properties of neural machine-translation encoder decoder approaches

Chung et al. 2014. Empirical evaluation of Gated RNN on sequential Modeling.

LSTM:

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

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

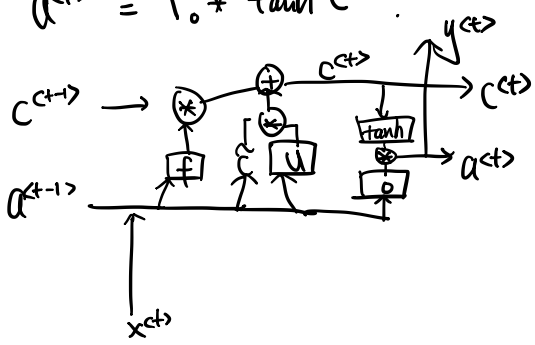
$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \quad \text{forget gate}$$

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

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

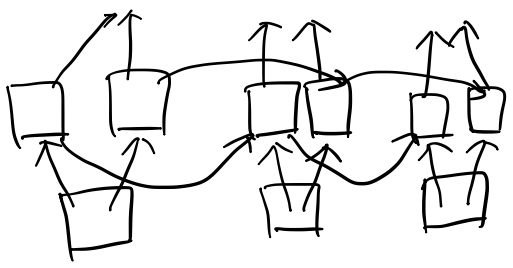
$c^{<t-1>}$ (sometimes)

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



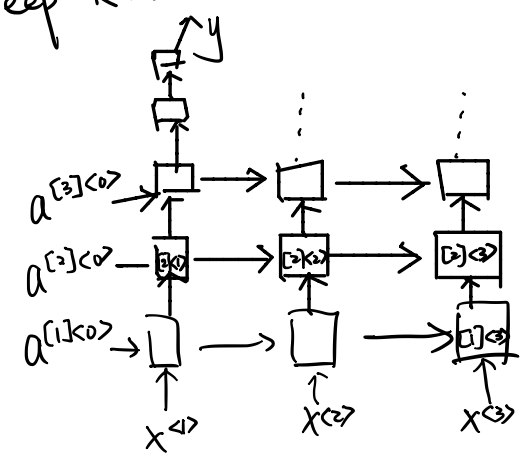
[Hochreiter & Schmidhuber 1997. Long short term memory]

Bi-RNN:



(need to wait until person finish talking)

Deep RNN:



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

Word representation.

[Van der Maaten and Hinton, 2008. visualizing data t-SNE]

Transfer learning and word embedding.

(face encoding have unlimitted faces while word embedding is limited Vocab)

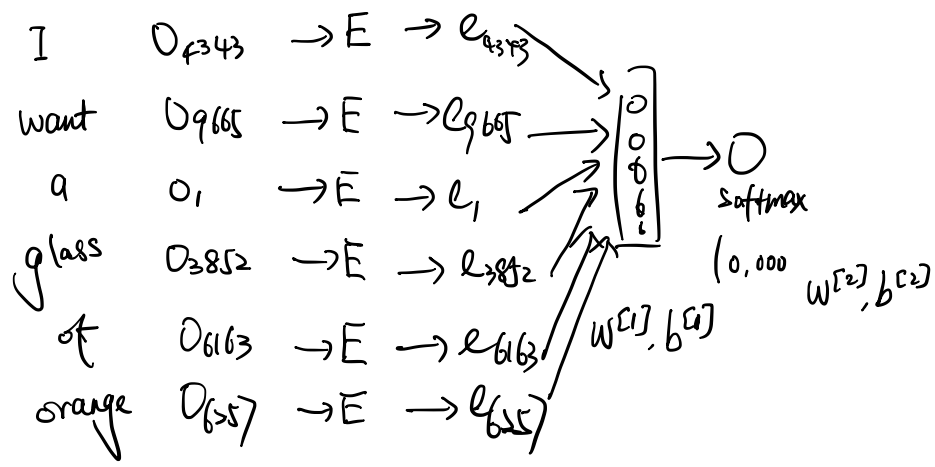
[Mikolov et al 2013 Linguistic regularities in continuous space word representations)

$E \cdot O_j = e_j$

E: embedding for word j

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

I want a glass of orange —



4 words on left & right

Context: last 4 words:

embedding: { nearby 4 words: previous 1 words: nearby words.

·word vector model [Mikolov et al 2013 Efficient estimation of word embeddings in vector space.

skip grams:

Context Target. ±10 window.

I want a orange juice

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

$$\mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{100000} y_i \log \hat{y}_i$$

Hierchical softmax can reduce computational cost to:

$O(|Vocab|)$ to $O(\log|Vocab|)$

Negative Sampling:

[Mikolov et al 2013 Distributed representation of words and phrases and their compositionality]

orange juice 1
orange king 0
orange book 0

choose nearby k words and replace with random words. to predict True or False.
common word is placed with false.
randomly chosen (of and)

K = 5-20 small dataset
k = 2-5 large dataset

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

each iteration.

$p(y=1|c,t) = \sigma(\theta_t^T e_c)$ k+1. binary classification. problem

selecting negative examples:

$$p(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{j=1}^{10000} f(w_j)^{\frac{3}{4}}}$$ in middle of from emperical frequency to standard distribution

Glove algo:

[Pennington et al. 2014 GloVe. Global vector for word representation]

X_{ij} = # times i appears in context of j.

$$\min \left(\sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(x_{ij}) (\theta_i^T e_j + b_i^t + b_j^c - \log x_{ij}^0)^2 \right)$$

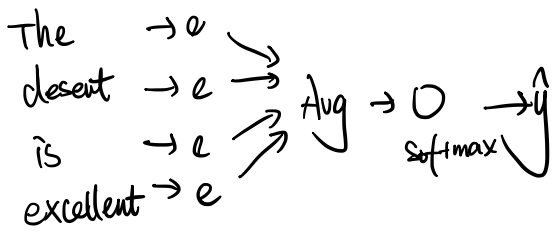
$f(x_{ij}) = 0$ if $x_{ij} = 0$

Stop words low weights
Common words big weights

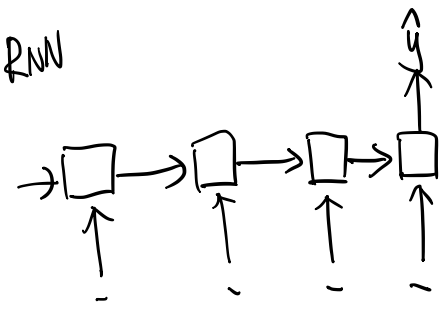
θ_i. e_j are symmetry

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

Sentiment analysis algorithm.



RNN



The problem of bias in word embeddings

Word embedding can reflect gender, age.

[Bolukbasi et al 2016. Man is programmer as women is homemaker Debiasing word embedding]

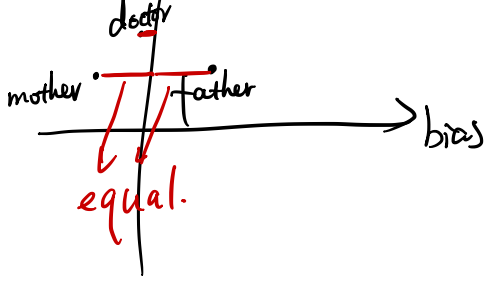
1. identify bias direction → (gender)

singular value decomposition to find gender

$e_{male} - e_{male}$
$e_{he} - e_{she}$

2. for word that are not definitional, project to get rid of bias.

3. equalize pairs.



sequence to sequence model.

[Sutskever et al. 2014 Sequence to Sequence learning with neural networks]

[cho et al 2014 Learning phase representations using RNN encoder decoder for statistical machine translation

machine translation

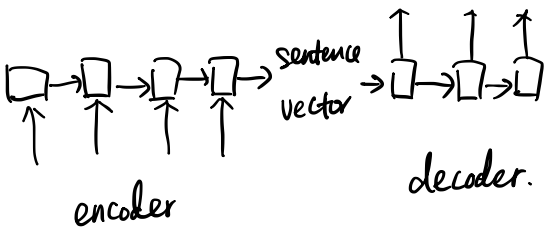
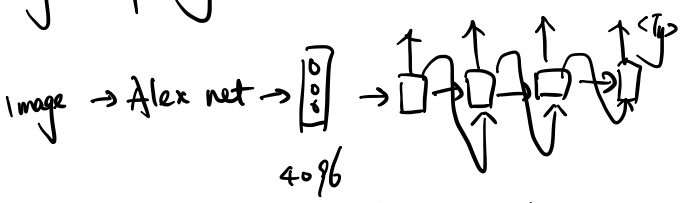


Image captioning:

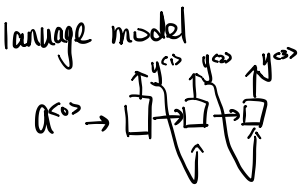


[Mao et al 2014 Deep captioning with multimodal recurrent neural networks]

[Vinyals et al 2014. Show and tell: Neural image caption generator]

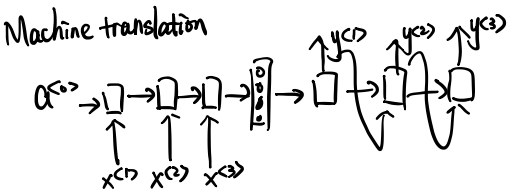
[karpathy and Li 2015 Deep visual semantic alignments for generating image descriptions]

language model



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

Machine translation



"conditional model" $P(y^{<1>}, y^{<2>}, y^{<3>} \cdots | x^{<1>} x^{<2>} \cdots)$

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

Beam search (not greedy search)

B = 3 (beam width)

try first B most possible candidate

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

every step evaluate 3 candidate. in total.

B=1, beam search = greedy search.

$$\underset{y}{\text{arg max}} \frac{\sum_{y=1}^{T_y} \log P(y^{<t>} | x, y^{<1>} \cdots y^{<t-1>})}{T_y^{\alpha}}$$

α: this makes do not prefer very short length.

Unlike exact search algo BFS (Breadth first Search) or DFS (Depth first search). Beam search runs faster but not guaranteed to find exact maximum for $\underset{y}{\text{arg max}} P(y^{<1>} \cdots y^{<T_y>} | x)$

Error analysis in Beam search.

RNN compute $P(y^* | x) \cdot P(\hat{y} | x)$. if human > RNN. beam is bad increase B

y^* human annotate

$\hat{y}$ RNN output.

see which one taking up more error

Bleu Score (Bilingual evaluation understudy)

Ref 1. A go to B to do C

Ref 2 A visit B to do C

each word appear in both sentences (bigram)

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

P_n compute on only gram.

combined Bleu score: $BP \exp\left(\frac{1}{4}\sum_{n=1}^{4} P_n\right)$

BP: brevity penalty

$$BP = \begin{cases} 1 & \text{if MT\_length} > \text{human\_len} \\ \exp\left(1 - \frac{\text{MT\_len}}{\text{reference\_len}}\right) & \text{otherwise} \end{cases}$$

[Papineni et al. 2002. Bleu: A method for automatic evaluation of machine translation]

Attention Model intuition.

[Bahdanau et al 2014 Neural machine translation by jointly learning to align and translate]

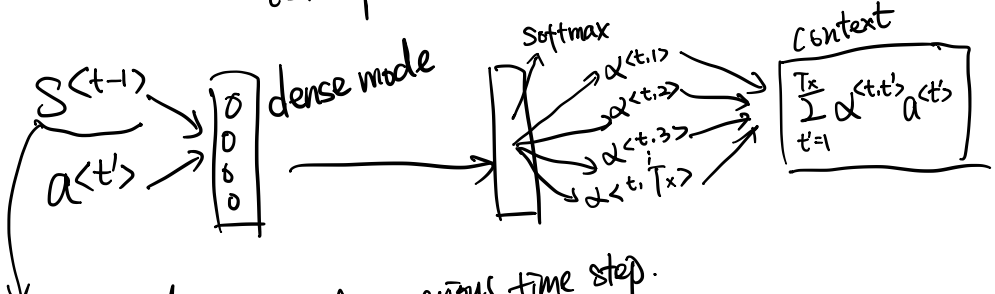
attention weights:

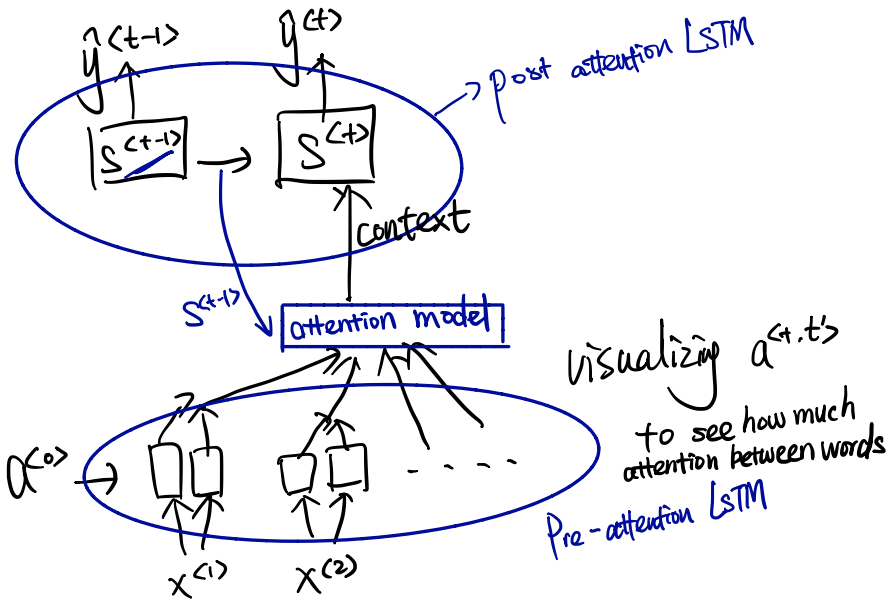
$$\sum_{t'} \alpha^{<1,t'>} = 1$$

$$C^{<1>} = \sum_{t'} \alpha^{<1,t'>} a^{<t'>} \quad \left(a^{<t'>} = (\vec{a}^{<t'>}, \overleftarrow{a}^{<t'>})\right)$$

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

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





The cost is quadratic cost, relatively high.

image captioning use attention model:

[Xu et al 2015 show, attend and tell: Neural image caption generation with visual attention]

speech recognition:

audio clip → transcript

previously: phonemes.

now: end2end system with 1000,000 h. audio

CTC loss [Graves et al 2006 Connectionist Temporal Classification: Labelling unsegmented sequence data with Recurrent neural networks.]

Trigger word detection System.

—programming exercise.