1. Tunning process.

① α - learning rate

β momentum size.

β_1 0.9

β_2 0.999 $\quad \varepsilon \to 10^{-8}$.

Layers

hidden units.

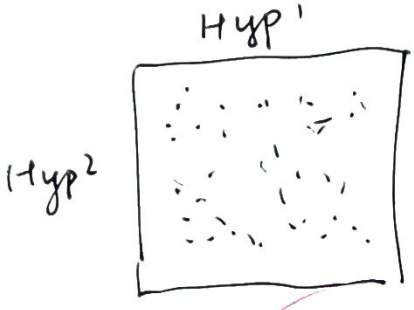
③ {

learning-rate decay

mini-batch size

② second important

Try random values.



Choose randomly ✓

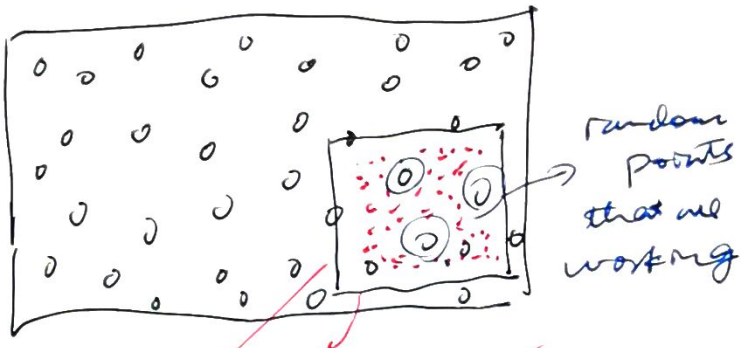
Say 25 points

try out 25 outcomes of a single possible Hyperparameter

reason

if use ✗ grid ∵ only 5 points of a single Hyp are tested out.

Coarse to fine.



making dense searching.

choose a finer area.

Using an appropriate scale to pick hyperparameters

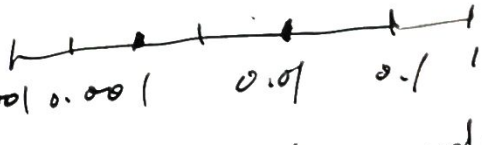
$n^{[l]} = 50, \dots, 100$ $\quad$ x x x (50 … 100)

Layer $\quad 2 \sim 4$

uniformly at random over range of contemplating.

$\alpha = 0.0001 \quad \dots \quad 1$

logrithmic distribution



$r = -4 * np.random.rand()$

$\alpha = 10^r \quad 10^{-4} \to 10^0$.

$\log_{10} 0.001 = -4$

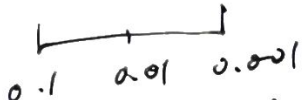
$\log_{10} 1 = 0$.

2.

momentum variable

$\beta = 0.9 \sim 0.999$

$1-\beta = 0.1 \sim 0.001$



10^{-1} 10^{-2} 10^{-3}

$\gamma \in [-1, -3]$.

$1-\beta = 10^{\gamma}$

$\beta = 1-10^{\gamma}$

Closer β is getting to 1.

it is more sensitive.

$\beta: 0.9 \rightarrow 0.905$ ~10 examples.

no big deal

$\beta: 0.999 \rightarrow 0.9995$

~1000 examples ~2000 examples (averaging over)

Re-test hyperparameters occasionally.

- NLP. Vision.

Intuitions do get stale.

Re-evaluate occasionally.

↗ new data. new prob....

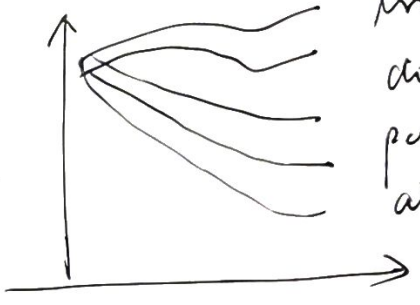
↘ re-test at least once several weeks.

Babysitting model

babysitting one model to tune every day.

Huge data but no Computation resources.

Train models in parallel



try out different parameters at a time.

{ panda model
{ caviar model

? knowledge in
↳ choosing computational resources.
Typically GPU.?
what capacity?
to handle what model?
How does it scale?
Does my laptop work?

3. Batch Normalisation

Sergey Ioffe & Christian Szegedy

Robustness.

Instead of only do norm on inputs X. → Try to normalised middle layer units a & (z).

Given some intermediate values in NN $z^{(1)}, z^{(2)}, \dots, z^{(m)}$ in L layer.

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

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

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

$$z^{(i)}_{norm} \sim N(0, 1)$$

But sometimes do not want $z^{(i)}_{norm}$ to always be.

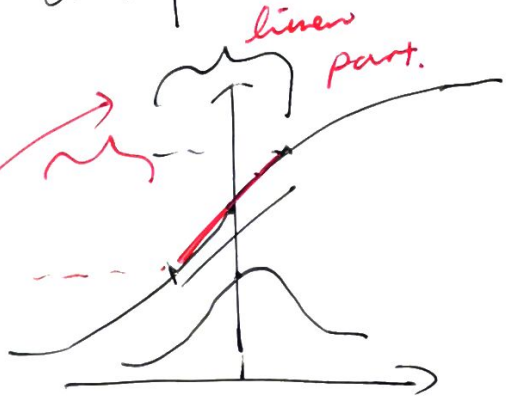
$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$

learnable parameters

Set updated just equal w.b. ←

if $\gamma = \sqrt{\sigma^2 + \varepsilon}$, $\beta = \mu$ then $\tilde{z}^{(i)} = z^{(i)}$

Use $\tilde{z}^{(i)}$ for later computation of consequent layer.



might want function has a larger variance & different means to maybe use more non-linear feature of logistic function.

can be able to control some means & variances

Adding Batch Norm to a network.

$x_1 \to \bigcirc \to \bigcirc \to \bigcirc \to \hat{y}$.

x_2

$x_3 \to \bigcirc \to \bigcirc$

$z^1 \to \tilde{z}^1$.

4.

$X \xrightarrow{w^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{Batch Norm (BN)}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]}$

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

happens between Computing z & a.

parameters $\{ w^{[1]}, b^{[1]}, w^{[2]} \cdots w^{[l]} b^{[l]}$
$\beta^{[1]}, \gamma^{[1]} \cdots \beta^{[l]}, \gamma^{[l]}$.

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

Tensorflow:

tf.nn.batch_normalization

Often applied with mini-batch

$X^{\{1\}} \xrightarrow{w^{[1]}} z^{\{1\}} \xrightarrow[BN]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow \cdots$

$X^{\{2\}} \xrightarrow{b^{[1]}} z^{\{2\}} \longrightarrow \tilde{z}^{[1]}$

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

normalized.

$z^{[l]}_{norm}$ rescale by $\beta^{[l]}, \gamma^{[l]}$

$\tilde{z}^{[l]}$

$w^{[l]}, \beta^{[l]}, \gamma^{[l]}$

cancel out the $b^{[l]}$. adding any constant do not change anything.

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

$z^{[l]}_{norm}$ decide variance. decide the mean

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

$(h^{[l]}, 1)$ $(h^{[l]}, 1)$

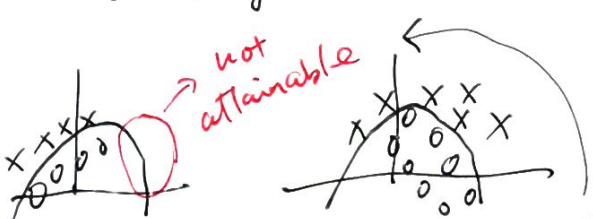
just as $b^{[l]} \rightarrow$

Reason for BN to work.

① Doing norm at hidden layers, instead only X.

② makes weights in deep layer more robust to changing weight in earlier layers.

eg: train black cats pic & apply to colored cats.



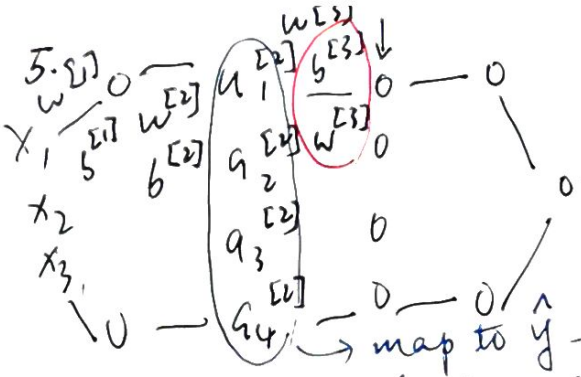
train → could not get right part.

covariate shift

TRUE FUNC DO NOT CHANGE.

$X \rightarrow Y$

mapping truly changes when X distri change

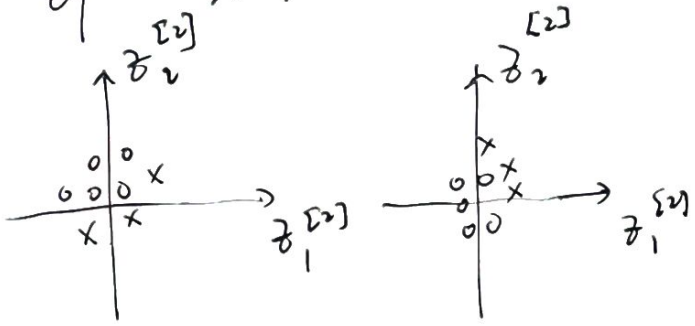


→ From perspective of third hidden layer. changing all the time. (like the change of input) Suffer from covariate shift.

reason →

Network is adapting paras $w^{[2]}, b^{[2]}, w^{[3]}, b^{[3]}$ → $a^{[2]}$ change.

BN → Reduces the amount of shifts around.



Kind of control the mean & variance of the changing paras.

→ $\beta^{[2]}, \gamma^{[2]}$ governed by.

makes the current layers more stable. The consequent layers could stand more reliably

Each mini-batch scaled by mean/var computed on just the same mini-batch

→ adds noise → $z^{[2]}$.

similar to dropout adds noise to each hidden layer. small

(operating on a small scale of data)

→ Regularization effect (side effect).

Larger mini-batch size → reduce noise ↓ → reduce Regularization effect ↓

At test time. μ, σ^2 as some sort of mean may not be calculated on with many examples

Calculate example once a time → EWA.

6. estimate μ, σ^2 using weighted average (across mini-batches)

$X^{\{1\}}, X^{\{2\}}, \ldots$

$\mu^{[l]\{1\}}, \mu^{[l]\{2\}}, \mu^{[l]\{3\}} \rightsquigarrow \mu$

Test time: process single example at a time.

In practice, use EWA to keep track of μ & σ^2.

(running average)

$\rightarrow \tilde{\mu}_t = \beta \tilde{\mu}_{t-1} + (1-\beta)\mu_t$

Activation Func. softmax.

$$t = e^{(z^{[l]})} \quad (4.1)$$

$$a^{[l]} = \frac{e^{z^{[l]}}}{\sum_{j=1}^{4} t_i} \quad (4.1).$$

normalised.

$$a_i = t_i / \sum_{i=1}^{4} t_i$$

eg: $z^{[l]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$. $t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$

$\sum_{j=1}^{4} t_j = 176.3$. $a^{[l]} = \frac{t}{176.3}$

$\Sigma p = 1$ $\left\{ \begin{array}{l} \rightarrow a = \frac{e^5}{176.3} = 0.842 \\ \rightarrow a = \frac{e^2}{176.3} = 0.042 \ldots \end{array} \right.$

For multi-class classification

Softmax Regression

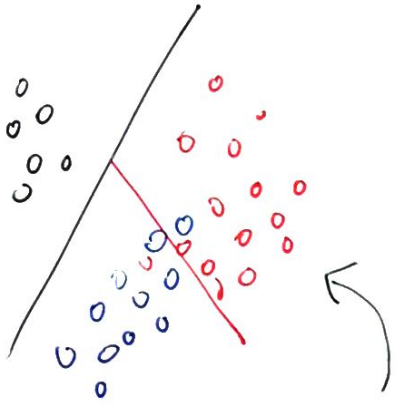
C = # classes = 4.

$X \rightarrow [] \rightarrow [] \rightarrow \hat{y}$ (P(other|x), p(bc|x).)

on layer L:

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

(4.1).



softmax single layer result (linear boundary)

could achieve non-linear through deep training

Hardmax

$$a^{[l]} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

generalize logistic regression into C classes instead of just two classes.

Loss function (entropy loss)

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

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

$$\mathcal{L}(\hat{y}, y) = -\log \hat{y}_2$$

→ make $\mathcal{L}$ ↓. $\hat{y}_2$ ↑ corresponding prob as high as possible ↑

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

$$Y = [y^{[1]}, y^{[2]}, \dots, y^{[m]}]$$

$$= \begin{bmatrix} 0 & 0 & \\ 1 & 0 & \cdots \\ 0 & 1 & \\ 0 & 0 & \end{bmatrix} \rightarrow (4, m)$$

$$\hat{Y} = \begin{pmatrix} 0.2 & 0.5 & \cdots \\ 0.3 & \cdots & \cdots \\ 0.6 & \vdots & \vdots \\ 0.4 & & \end{pmatrix} \rightarrow (4, m)$$

$z^{[l]} \rightarrow a^{[l]} = \hat{y} \rightarrow \mathcal{L}(\hat{y}, y)$.

back prop:

$$dz^{[l]} = \hat{y} - y.$$

(4.1)

$$\frac{\partial J}{\partial z^{[l]}}$$

start at layer L to do backprop

Tensorflow helps you in backprop. only do forward right is enough.

7

7

Carry out error analysis.

90% accurate

Something look like a cat but it is a dog.

start cat classifier do better on dog? randomly.

Error analysis procedure:

1. Get ~100 mislabeled dev set examples.
2. count up how many are dogs.

5% → 5/100 → 90% → 95%

ceiling of improvement of performance.

50% → 90% → 95%.

work several hypothesis in parallel.

dogs mislabeled

big cats wrong.

blurry images.

Cleaning up incorrected labeled data.

(mislabeled by human etc.)

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

But systematic errors *.

ADD extra column for in dev/test set.

stubbing.

10%.

6%

Errors due to mislabeled errors → 0.6%.

other causes → 9.4%.

2% { 0.6% significant. 1.4%.

Instagram

Image. Dog DC Blurry coming Pitbull too blurry.

	% of total
1	8%
2	43%
3	61%
⋮	

a sense of overall improvement ceiling for each features.

1.

2 A & B.

2.1% 1.9%. Hard to judge the not Trust them right one.

→ mislabeled data

→ fix mislabeled.

Correcting incorrect dev/test set examples.

& apply same process.

① to ensure same distribution

② Consider examples wrongly classified → not actually right.

→ otherwise → bias.

③ Train & dev/test data slightly may not vary differently in term of distribution

Build new system suggestion:

build first system quickly & start iterating

set up dev/test set & metric

→ build first system quickly. & dirty.

use. variance / bias analysis & error analysis. ?

↓

figure out which direction to go is most worthwhile.

tons of directions

Train & dev/test on different distribution

Data from webpage	Data from mobile app.
clear.	blurrier.
high resol.	huge.
≈ 200,000.	≈ 10,000

care about performance of pics from mobile app.

op1 ✗

not good for this

Combine them randomly shuffle.

op2

Train: 200,000 web. 5,000 mobile

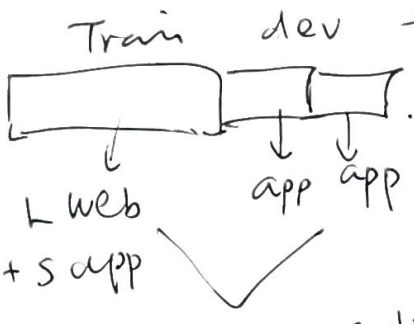
dev: app

test: app

better goal.

better for long term.

Data mismatch problem.



not same distribution.

Train error	10%	1%
Train-dev error	11%	11%
Dev error	12%	20%
Human error	0%	

avoidable bias issue

bias & Data mismatch issues

Train error 1%
dev error 10%
te ...
} no longer concluded as variance directly.

Human level error } avoidable bias
Train set error }
T-dev set error } variance.
Dev error } data mismatch
Test error } Dev overfitting

since. train data could be easy to train & does not apply well on another distribution.

↓ solution.

train/dev set.

Shuffle Train & Set data and leave one block for T/Dev.

Same distri

same distri

not training.

Train error	1%	1%
Train/Dev error	9%	1.5%
Dev error	10%	10%

} data mismatch problem.

Variance prob

avoidable bias { 4%
variance { 7%
10%
} Train data is harder } harder.

mismatch { 6%
0%
} Dev/test is easier.

Human may very perform on this Dev/test.

eg: 4% → { 6%
2%

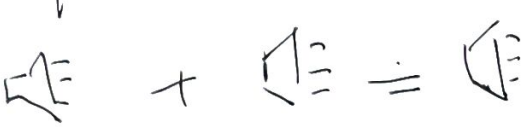
Addressing Data mismatch problem.

① Carry out manual error analysis to understand difference between training & dev/test sets.

Eg: noisy → car noise.
for dev set.
Generate numbers.
insights.

② Making training data more similar to Dev/test.
or collecting more data.
to dev/test sets.

Artificial Data Synthesis.

eg: 

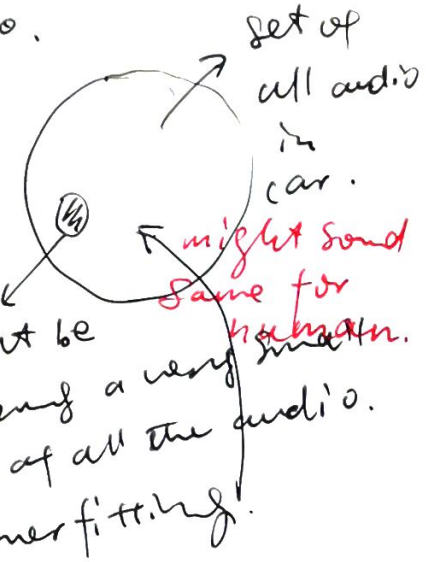
"The quick brown fox jumps over the lazy dog" → easy, sentence with all alphabet.

car noise

Synthesized in-car audio.

replicates small subset of car audio 1000 times & add it with the normal train audio.

Risk:



might overfit to the small amount of train subset.
while dev/test set very general cases.

Using knowledge trained by a task to apply to a second task
— Transfer Learning.

Transfer learning.

pre-training

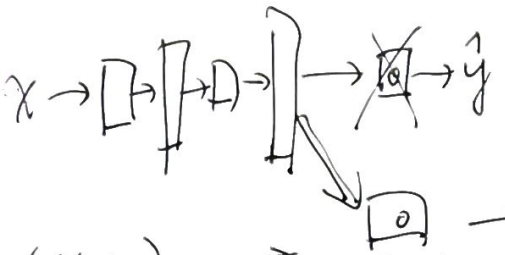


image recog

fine tuning
radiology / diagnosis.

(x, y) → $w^{[L]}, b^{[L]}$ → $\hat{y}$

↑ new inputs.

small set of data / viable

last few layers.

to only train last layer.

Large set of data. train all the layer of data.

or the layers targeted to do the recog.

pixels → edge → shape → facial → diagnosis

may create several new layers to tackle your new problem.

Makes sense when:

a lot of data (transfered from.

Limited data on Targeted (transfer to)

not viable vice-versa. not meaningful

should be inter-connected somehow.

less valuable & less helpful.

Multi-task learning.

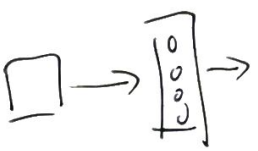
Self-driving cars.

detecting pedestrians. other cars, stop-signs. traffic lights ...

$x^{(i)}$ → $y^{(i)} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$ (4,1)

$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix}$

4 × m.



Loss: $\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{4} \mathcal{L}(\hat{y}_j^{(i)}, y_j^{(i)})$

→ usual logistic loss.

Could softmax regression:

One image can have multiple ~~Labels~~ Labels.

a car can have 4 feats.

a thing only be a car or dog or cat..

Train one neural network to do four things might be better than train 4 network to do 4 things separately.

→ earlier feature in nn can be shared.

→ multi-task learning advantage.

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

viable for some labels missing for certain examples.

↙

Sum over 1-4
vs sum over 1-2
over where labels present

makes sense:

① Amount of data for each task is quite similar

A_1 1000
A_2 1000
⋮
A_{100} 1000.

} data augment.

② train on a set of tasks that could benefit from having shared lower-level features.

③ Train a big enough nn to do well on all tasks

↓

hurt performance only when nn not big enough.

End to end deep learning.

eg: speech recog example.

X audio → MFCC → features → ML → phonemes "cat" → words → transcript.

audio → y. → transcript. (whole nn.)

bypassing all intermediate steps. with one neural n.

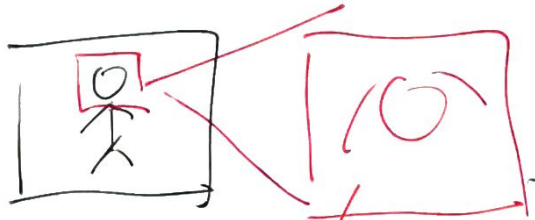
Traditional pipeline

End to End pipeline.

Large amount of data
→ End to End approach better.

Small, medium size
→ traditional or Small (partially) end to end approach

eg:



Face Recog. identity. compare match? database.

Rec. ① → ② simpler questions combined → better performance

End to End does better in machine trans.

Big data on (X, y)
English French.

Data feed in the most important, productive element.

* Pros: ① let the data speak. not enforce human intuition
② less hand-designing of components needed.

Cons: ① Large data needed.
② Excludes potentially useful hand-designed components.

small important data

A lot of data for doing both ① & ②.
Not enough for purely end to end.

But for Machine Translation
Eng → text ... → ... → French.
Eng → ——————→ French.

Combined with other subjects: motion planning
control
more promising

Core question: how to evaluate amount of data needed to map from X → y.
unsolved.

7.