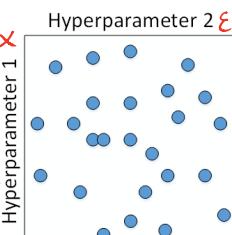
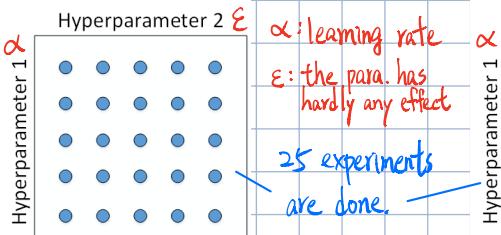
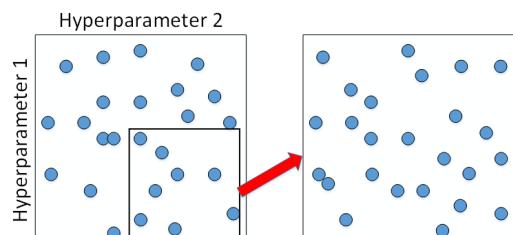


Tuning Process : Hyperparameters Searching



Don't sample averagely like a grid. If ϵ has no effect on training but only α does, then we only have five experiment results with different α .

Sample randomly. If ϵ is shown has no effect on training, then 25 experiments with different α are done, which is more efficient.

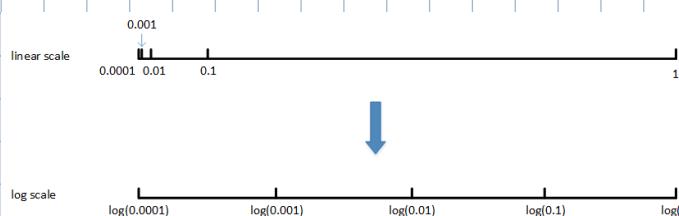


Sample more densely on the area where shows better results.

It's just a 2D example. In practice, we are searching over more hyperparameters than three. And sometimes it's hard to know in advance which hyperparameters are most important.

Using an appropriate Scale to Pick Hyperparameters

Assume we want to search $\alpha = 10^{-4} \sim 10^0$.



Draw samples smartly from 10^a to 10^b .

$$a = \log_{10}^{10^a}, b = \log_{10}^{10^b}$$

$$r = -3 \cdot np.random.rand()$$

$$\alpha = 10^r$$

$$r = np.random.rand() \# 0 \sim 1$$

$$r = a + (b - a) \cdot r$$

$$\alpha = 10^r$$

If drawing 100 samples by using linear scale, 90% samples will locate in $0.1 \sim 1$, which is just wrong.

So, we should draw samples by using log scale.

Eg. If we want to search over β from 0.9 to 0.999,

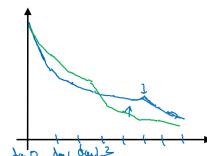
$$\beta = 0.9, \dots, 0.999$$

$$1 - \beta = 0.1, \dots, 0.001$$

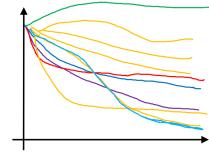
$$\Rightarrow 10^{-3}, \dots, 10^{-1}$$

$$\beta = 1 - 10^r$$

Babysitting one model



Training many models in parallel



Hyperparameters Tuning in Practice : Panda v.s. Caviar

- Study different domains, such as NLP, Vision, Speech, Logistics to get inspirations and intuition.

- Intuition do get STALE.
Re-evaluate.

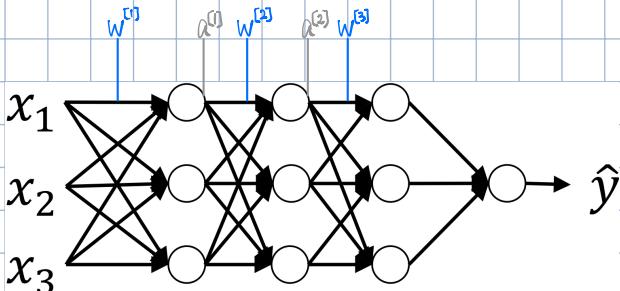
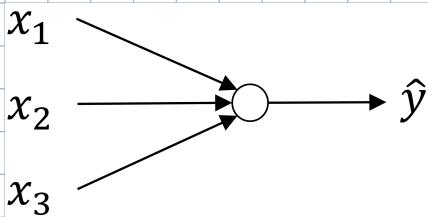
就像我在LiveED做了五年，其實到最後已經有點STALE了。不想去思考如何改進3D模型，不想去嘗試新的方法（比如用Deep Learning解決 Dense Stereo Matching），或是改善現有的方法（比如減少Rectification的Distortion或是用Geometry去filter不跟Geometry intersect的tiles）。
總知道冰消將，不進則退。
但是在有限的時間和熱情下，我必須挑戰的學習。

Huge dataset. Few resources.

Lots of resources.

Batch Normalization

Normalizing Activations in a Network:



Recall normalizing the input data can increase the speed of learning and accuracy.

$$\text{mean } \mu = \frac{1}{m} \sum_{i=1}^m X^{(i)}$$

$$\text{Variance: } \sigma^2 = \frac{1}{m} \sum_{i=1}^m (X^{(i)} - \mu)^2$$

$$Z = \frac{X - \mu}{\sigma}$$

Goal: Can we normalize $a^{(i)}/z^{(i)}$ so that $w^{(i+1)}$ and $b^{(i+1)}$ can be trained faster?

Notice $a^{(i)} = g^{(i)}(z^{(i)})$. We choose to normalize $z^{(i)}$.

Omitted

It's not the number of hidden units.

Implement Batch Norm on the hidden layer l . Given $Z^{(i)}$, $i = 1, \dots, m$ m is the number of training samples.

$$\mu = \frac{1}{m} \sum_{i=1}^m Z^{(i)}$$

$n^{(i)}$: the number of units in the layer l , $Z^{(i)} : (n^{(i)}, m)$

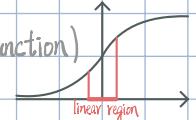
$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (Z^{(i)} - \mu)^2$$

$$(E=10^{-8}, \text{numerical stability}) \quad Z_{\text{norm}}^{(i)} = \frac{Z^{(i)} - \mu}{\sqrt{\sigma^2 + E}}$$

At this point, the sequence $Z^{(1)}, \dots, Z^{(m)}$ has zero mean and unit variance in each dimension.

$$\text{Eq: } \bar{Z}^{(i)} = \begin{bmatrix} z_1^{(i)} & z_2^{(i)} & \dots & z_m^{(i)} \\ z_1^{(i)} & z_2^{(i)} & \dots & z_m^{(i)} \end{bmatrix}$$

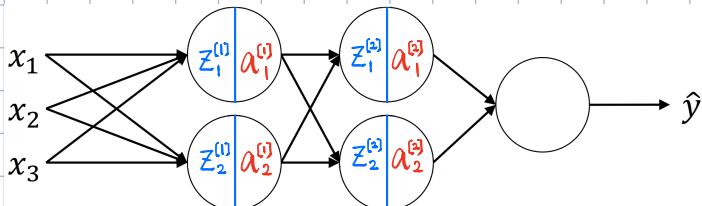
It actually makes more sense to make $\tilde{Z}^{(i)}$ have different scales. (Why? Recall nonlinearity of activation function)



$$\tilde{Z}^{(i)} = \gamma \cdot Z_{\text{norm}}^{(i)} + \beta. \gamma \text{ and } \beta \text{ are learnable parameters of the model and can be used to set the mean and variance of } \tilde{Z}^{(i)} \text{ to any value.}$$

$$\text{Eq: If } \gamma = \sqrt{\sigma^2 + E} \text{ and } \beta = \mu, \text{ then } \tilde{Z}^{(i)} \text{ has zero mean and unit variance.}$$

Fit Batch Norm into a Neural Network



X $\xrightarrow{W^{[0]}, b^{[0]}}$ Z^[1] $\xrightarrow{\beta^{[1]}, \gamma^{[1]}}$ $\tilde{Z}^{[1]}$ $\xrightarrow{\text{BN}}$ A^[1] $\rightarrow \dots \rightarrow A^{[L-1]} \xrightarrow{W^{[L]}, b^{[L]}}$ Z^[L] $\xrightarrow{\beta^{[L]}, \gamma^{[L]}}$ $\tilde{Z}^{[L]}$ $\xrightarrow{\text{BN}}$ A^[L]

For iteration t: (from 1 to num-minibatch)

Compute ForwardProp on $X^{(t)}$

In each hidden layer, use BN to compute $\tilde{Z}^{(i)}$ from $Z^{(i)}$.

BackwardProp: Compute $dW^{(i)}$, $d\gamma^{(i)}$, $d\beta^{(i)}$, $db^{(i)}$

$$\begin{aligned} w^{(i)} &= w^{(i)} - \alpha dW^{(i)} \\ r^{(i)} &= r^{(i)} - \alpha dr^{(i)} \\ \beta^{(i)} &= \beta^{(i)} - \alpha d\beta^{(i)} \end{aligned}$$

Wait! Where is $b^{(i)}$? BN cancels out $b^{(i)}$ when subtracting mean

$$\tilde{Z}^{(i)} = W^{(i)} \alpha^{(i-1)} + \beta^{(i)}$$

Notice for $\gamma, \beta: (n^{(i)}, 1)$

each sample $\tilde{Z}^{(i)}: (n^{(i)}, 1)$

Gradient Descent. But it can also be Momentum, RMSprop, or Adam.

Why does Batch Norm work?

Given a neural network,

Train a network on black cats

$$y = 1$$



$$y = 0$$



Test the network on color cats

$$y = 1$$



$$y = 0$$



Covariate shift

$$X \rightarrow Y$$

The idea is that when learning X (lots of cats and non-cats images) to Y (is it a cat or not) mapping, if the distribution of X changes (use color cats but not black cats), then we might have to retrain the learning algorithm. And this is true if the function (mapping from X to Y) remains unchanged (is it a cat or not).

Batch Norm makes the earlier layers not shift around as much, because they are constrained to have the same mean and variance, which makes the job of learning on the later layers easier.

Batch Norm at Test Time

During training, μ and σ^2 are derived with m samples in a mini-batch. And then use μ and σ^2 to normalize $Z^{(i)}$.

$$\mu = \frac{1}{m} \sum_{i=1}^m Z^{(i)}$$

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

$$Z_{\text{norm}}^{(i)} = \frac{Z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$\tilde{Z}^{(i)} = \gamma \cdot Z_{\text{norm}}^{(i)} + \beta$$

In test time, there is no mini-batch but only one sample. It doesn't make sense to use one sample to calculate μ and σ^2 .

Thus, we can estimate μ and σ^2 using exponentially weighted average during training across mini-batches.

Mini-Batch: $X^{(1)}, X^{(2)}, X^{(3)}, \dots$

For the layer l : $\mu^{(1(l))}, \mu^{(2(l))}, \mu^{(3(l))}, \dots$

We run exponentially weighted average across the layer l in all mini-batches, and we get a final μ and σ^2 .

Last, μ and σ^2 are used at test time.

Multi-Class Classification

Softmax Regression

Recognizing cats, dogs, and baby chicks



3

1

2

0

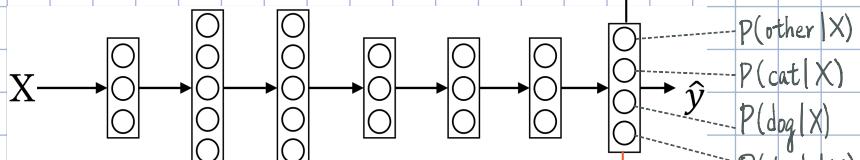
3

2

0

Softmax Layer:

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



The Softmax Activation Function $g(z) = \frac{e^z}{\sum_{i=1}^{C-1} e^z}$

$$a_j = \frac{e^{z_j}}{\sum_{i=1}^{C-1} e^{z_i}}$$

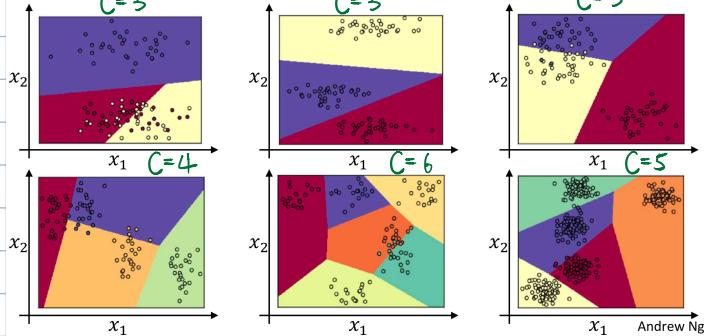
$$\hat{y} = a^{[L]} = g^{[L]}(z^{[L]})$$

(4, 1)

(4, 1)

The network: $x_1 \rightarrow \boxed{\text{softmax layer}} \rightarrow \hat{y}$

Softmax examples



Training a Softmax Classifier:

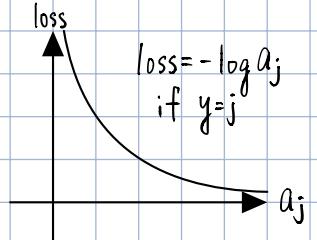
$$\text{Given } z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}, \sum_{i=1}^4 e^z_i = e^5 + e^2 + e^{-1} + e^3, a^{[L]} = g^{[L]}(z^{[L]}) = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \times \frac{1}{\sum_{i=1}^4 e^z_i} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

Softmax regression generalizes Logistic regression to C classes. (I.e., logistic regression $\rightarrow C=2$)

Loss Function: Eg: Given $y = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$, $\hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$, $C=4$, we define $L(\hat{y}, y) = -\sum_{j=1}^C y_j \log \hat{y}_j$

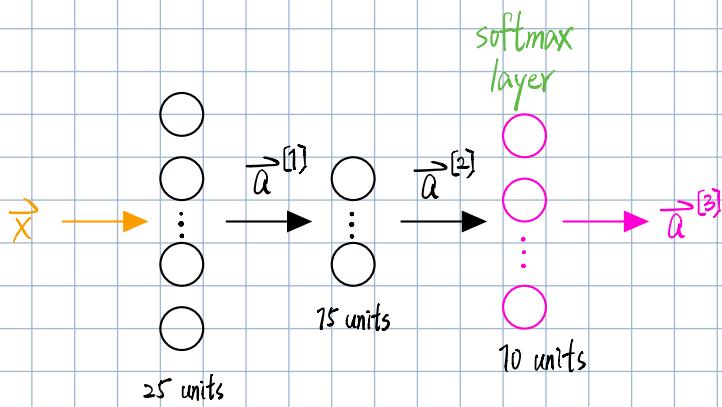
$$\text{Cost Function: } J = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)})$$

Note: $y_j = 1$



Neural Network with softmax output.

Let's use the example of recognizing handwritten digits $0, 1, \dots, 9$. $C=10$



$$z_1^{[3]} = w_1^{[3]} \cdot a^{[2]} + b_1^{[3]}$$

$$\vdots$$

$$z_{10}^{[3]} = w_{10}^{[3]} \cdot a^{[2]} + b_{10}^{[3]}$$

Then,

$$a_1^{[3]} = \frac{e^{z_1^{[3]}}}{\sum_{i=1}^{10} e^{z_i^{[3]}}} = P(y=1 | \vec{x})$$

$$a_{10}^{[3]} = \frac{e^{z_{10}^{[3]}}}{\sum_{i=1}^{10} e^{z_i^{[3]}}} = P(y=10 | \vec{x})$$

cross entropy loss $(a_1, a_2, \dots, a_N, y) = \begin{cases} -\log a_1 & \text{if } y=1 \\ \vdots \\ -\log a_N & \text{if } y=N \end{cases}$

This actually makes sense.

Recall loss function of logistic regression:

$$\text{loss} = -y \log \hat{y} - (1-y) \log(1-\hat{y})$$

if $y=1$ if $y=0$

What's multi-label classification?

Given an image, we want to know if there's car, bus, or pedestrian in the image.

Is there a car ?	Yes
Is there a bus ?	Yes
Is there a pedestrian?	No

There are multiple targets existing in an image at the same time. Typically, there are two ways to approach it.

Option 1: we can use three different neural networks to predict each individual object.

Option 2: we can train one neural network with three outputs.

$$\vec{a}^{[3]} = \begin{pmatrix} a_1^{[3]} \\ a_2^{[3]} \\ a_3^{[3]} \end{pmatrix} \begin{matrix} \text{car} \\ \text{bus} \\ \text{pedestrian} \end{matrix}$$

Notice that in this case, the softmax can not be used as the activation function.

We have to use Sigmoid for each output unit $a_1^{[3]}, a_2^{[3]}, a_3^{[3]}$. Each output unit is an independent entity whose probability is not related to each other.