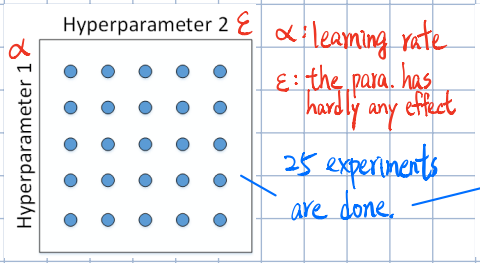
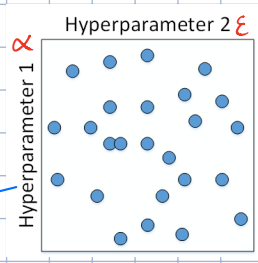
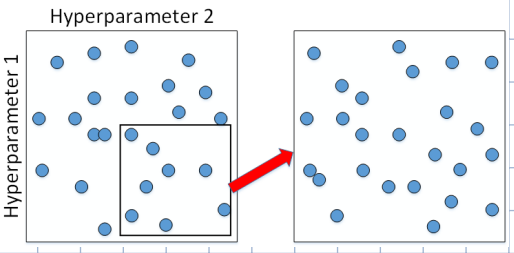
Tuning Process : Hyperparameters Searching



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.

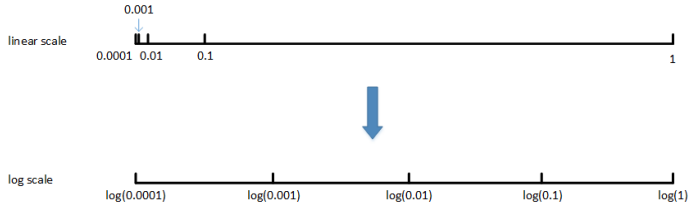


Coarse to fine.
Sample more densely on the area where shows better results.

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 $\alpha = 10^{-4} \sim 10^{0}$.



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.

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

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

$r = np.random.rand() \# 0 \sim 1$
$r = a + (b-a) \cdot r$
$\alpha = 10^r$

10^{-3} 10^{0}
$r = -3 \cdot np.random.rand()$
$\alpha = 10^r$

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

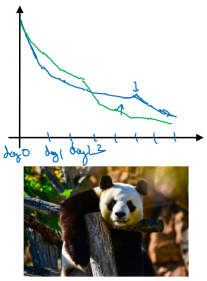
$\beta = 0.9, \dots, 0.999$
$1-\beta = 0.1, \dots, 0.001$
$\Rightarrow 10^{-3}, \dots, 10^{-1}$
$\beta = 1 - 10^r$

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LineED待了五年，真實到最後已經有點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)。
夢如逆水行舟，不進則退。
但是在有限的時間和熱情下，我必須挑食的學習。

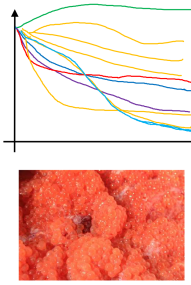
Babysitting one model



Panda

Huge dataset. Few resources.

Training many models in parallel

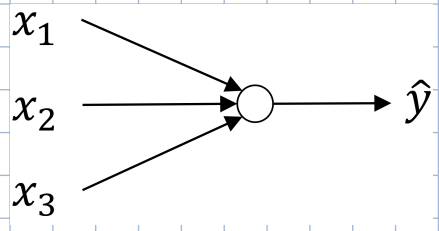


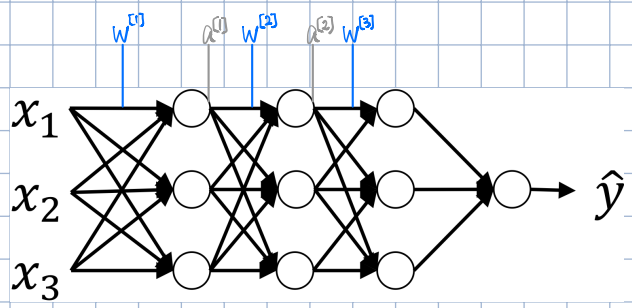
Caviar

Lots of resources.

Batch Normalization

Normalizing Activations in a Network:





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

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

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

$$X = \frac{X-\mu}{\sigma} \quad \text{(element-wise)}$$

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

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

Implement Batch Norm on the hidden layer l. Given $z^{[l](i)}$ (omitted), $i = 1, \dots, m$ (m is the number of training samples. It's not the number of hidden units.)

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

$n^{[l]}$: the number of units in the layer l, $z^{[l]}: (n^{[l]}, m)$

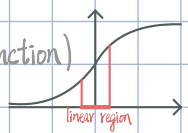
$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(z^{(i)}-\mu)^2$$

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

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

Eg: $X^{[l]} = \begin{bmatrix} z_1^{(1)} & z_1^{(2)} & \cdots & z_1^{(m)} \\ z_2^{(1)} & z_2^{(2)} & \cdots & z_2^{(m)} \end{bmatrix}$ (mean 1, variance 1)

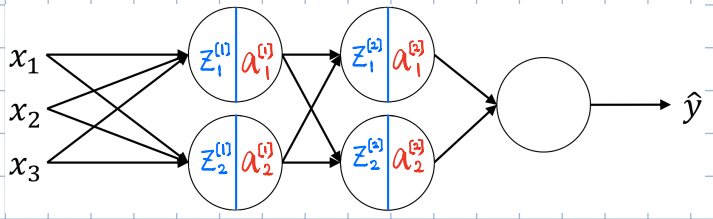
It actually makes more sense to make $z^{(i)}_{norm}$ have different scales. (Why? Recall nonlinearity of activation function)



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

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

Fit Batch Norm into a Neural Network



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

For iteration t: (from 1 to num_minibatch)

Compute ForwardProp on $X^{\{t\}}$

In each hidden layer, use BN to compute $\tilde{z}^{[l]}$ from $z^{[l]}$.

BackwardProp: Compute $dw^{[l]}$, $d\gamma^{[l]}$, $d\beta^{[l]}$, ~~$db^{[l]}$~~

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

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

Wait! Where is $b^{[l]}$? BN cancels out $b^{[l]}$ when subtracting mean.

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

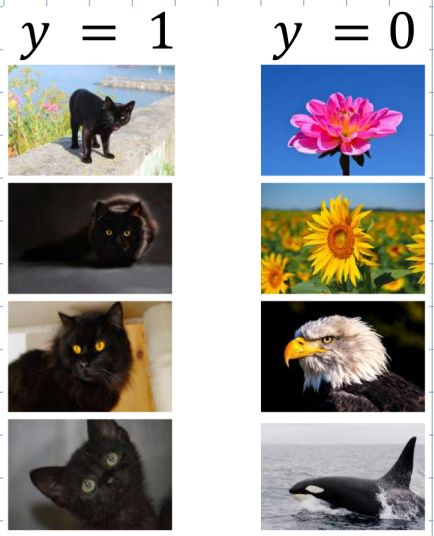
Notice for $\gamma, \beta: (n^{[l]}, 1)$

each sample $\tilde{z}^{[l]}: (n^{[l]}, 1)$

Why does Batch Norm work?

Given a neural network,

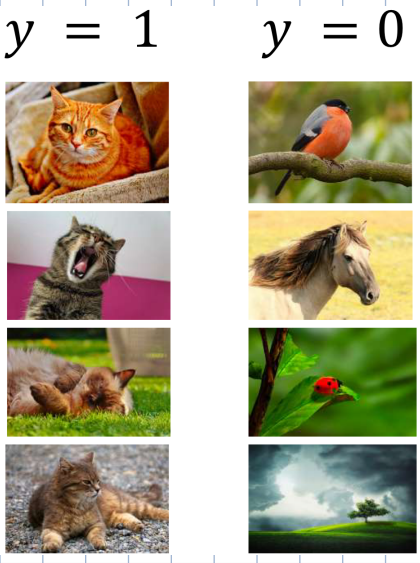
Train a network on black cats



Covariate Shift

$X \rightarrow Y$

Test the network on color cats



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), the we might 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 $z^{(i)}$.

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

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

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

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

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: $X^{\{1\}}$, $X^{\{2\}}$, $X^{\{3\}}$, ...
For the layer l: $\mu^{\{1\}[l]}$, $\mu^{\{2\}[l]}$, $\mu^{\{3\}[l]}$, ...

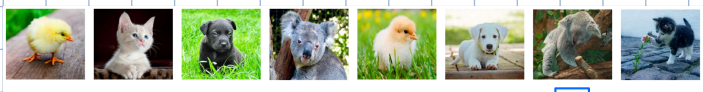
We run exponentially weighted average across the layer l in all mini-batches, and we get a final μ and σ^2.

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

Multi-Class Classification

Softmax Regression

Recognizing cats, dogs, and baby chicks



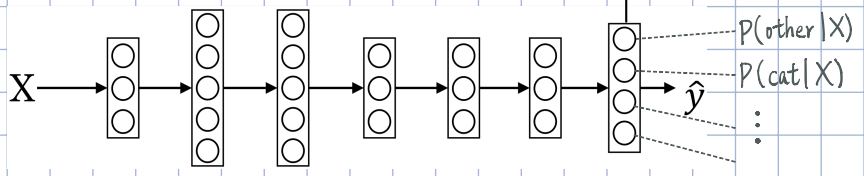
3 1 2 0 3 2 0 1

None-of-the-above labels

C: # classes $= 4$ $(0, \dots, 3)$

Softmax Layer:

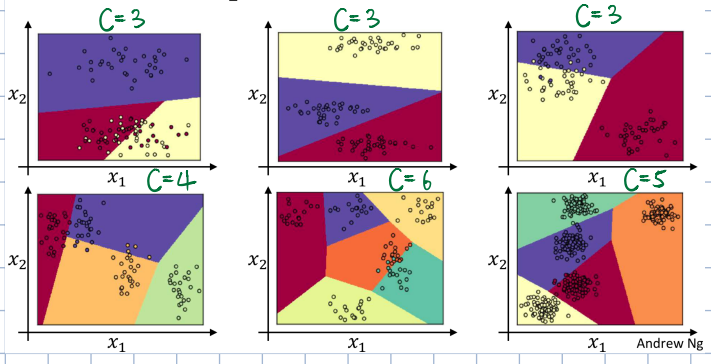
$z^{[L]} = W^{[L]} a^{[L-1]} + b^{[L]}$



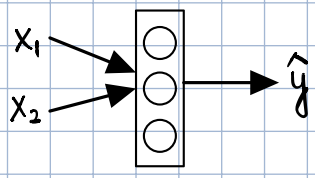
The Softmax Activation Function $g(z) = \frac{e^z}{\sum_{i=1}^{C=4} e^z_i}$

$\hat{y} = a^{[L]} = g^{[L]}(z^{[L]})$ $(4,1)$ $(4,1)$

Softmax examples



The network:



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

$a^{[1]} = \hat{y} = \underset{g}{\text{softmax}}(z^{[1]})$

Training a Softmax Classifier:

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 $y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $\hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$, $C=4$, we define $\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{C} y_j \log \hat{y}_j$ $\mathcal{L}(\hat{y}, y) = -y_2 \log \hat{y}_2 \downarrow$, $\hat{y}_2 \uparrow$

Cost Function: $J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$, $dz^{[L]} = A^{[L]} - Y$