Hyperparameters

Priority 1

Priority 2

Priority 3

$\boxed{\alpha}$

$\boxed{\beta}$ ~0.9

Almost never tune with ADAM

β_1, β_2, ε
0.9, 0.999, 10^{-8}

$\boxed{\text{\# layers}}$

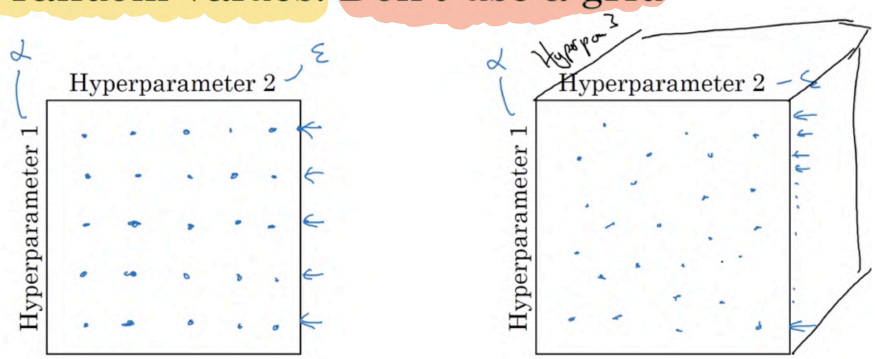
$\boxed{\text{\# hidden units}}$

$\boxed{\text{learning rate decay}}$

$\boxed{\text{mini-batch size}}$

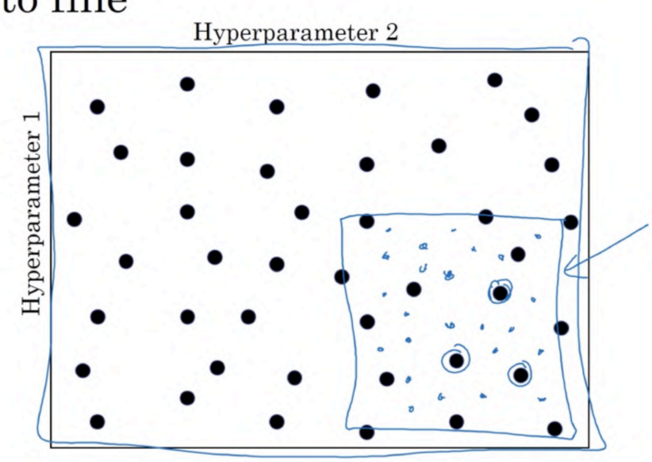
Andrew Ng

Try random values: Don't use a grid



Andrew Ng

Coarse to fine



Andrew Ng

Picking hyperparameters at random

$\rightarrow n^{[l]} = 50, \ldots, 100$

50 — 100

$\rightarrow$ #layers $L: 2 - 4$

$2, 3, 4$

OR

Appropriate scale for hyperparameters

$\alpha = 0.0001, \ldots, 1$

Uniform

log scale

0.0001 — 1

0.1 — 1

0.0001 0.001 0.01 0.1 1

10^{-4} 10^{0}

10^{a} $a = \log_{10} 0.0001 = -4$

10^{b} $b = \log_{10} 1 = 0$

$r = -4 *$ np.random.rand() $\leftarrow r \in [-4, 0]$

$\alpha = 10^{r}$ $\leftarrow 10^{-4} \ldots 10^{0}$

$10^{a} \ldots 10^{b}$

$r \in [a, b]$

$[-4, 0]$

$\alpha = 10^{r}$

so $\alpha = 10^{r}$ where $r \in [-4, 0]$

Hyperparameters for exponentially weighted averages

$\beta = 0.9 \ldots 0.999$

$\downarrow$ 10 $\downarrow$ 1000

$1 - \beta = 0.1 \ldots 0.001$

$\beta: 0.9000 \rightarrow 0.9005 \} \sim 10$

$\beta: 0.999 \rightarrow 0.9995$

~ 1000 ~ 2000

$\frac{1}{1-\beta}$

0.9 — 0.999

0.9 0.99 0.999

0.1 0.01 0.001

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

$r \in [-3, -1]$

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

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

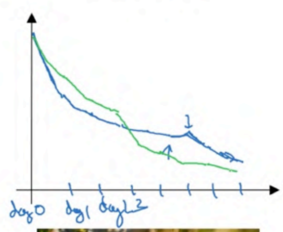
Re-test hyperparameters occasionally

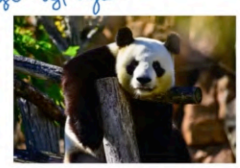
Idea

Experiment Code

- NLP, Vision, Speech, Ads, logistics, ...
- Intuitions do get stale. Re-evaluate occasionally.

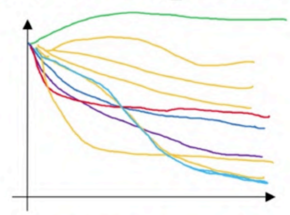
Babysitting one model





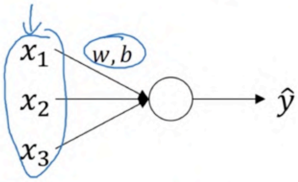
Panda ←

Training many models in parallel



Caviar ←

Normalizing inputs to speed up learning

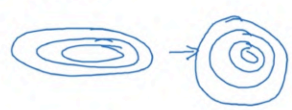


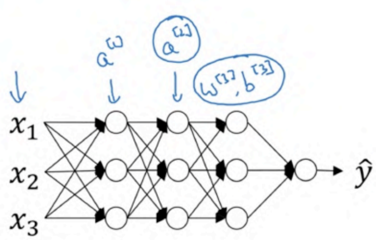
$\mu = \frac{1}{m} \sum_i x^{(i)}$

$X = X - \mu$ ← element-wise

$\sigma^2 = \frac{1}{m} \sum_i x^{(i)2}$

$X = X / \sigma^2$





Can we normalize $a^{[2]}$ so as to train $W^{[3]}, b^{[3]}$ faster

Normalize $z^{[2]}$

Implementing Batch Norm

Given some intermediate values in NN $\underbrace{z^{(1)}, \dots, z^{(m)}}_{z^{[l](i)}}$

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

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

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

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

learnable parameters of model.

learnable

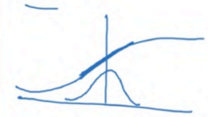
If $\gamma = \boxed{\sqrt{\sigma^2 + \varepsilon}}$

$\beta = \boxed{\mu}$

then $\tilde{z}^{(i)} = z^{(i)}$

X ←

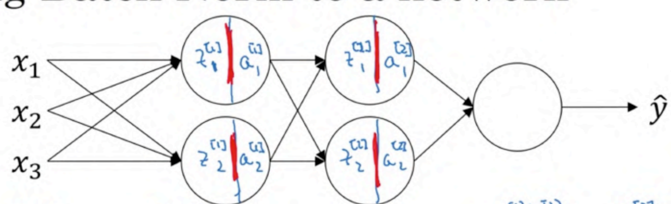
$z^{(i)}$ ←



Use $\tilde{z}^{[l](i)}$ instead of $z^{[l](i)}$.

Andrew Ng

Adding Batch Norm to a network



$$X \xrightarrow{W^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{Batch Norm (BN)}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \to a^{[1]} = g^{[1]}(\tilde{z}^{[1]}) \xrightarrow{W^{[2]}, b^{[2]}} z^{[2]} \xrightarrow[BN]{\beta^{[2]}, \gamma^{[2]}} \tilde{z}^{[2]} \to a^{[2]} \to \dots$$

Parameters: $\left.\begin{matrix} W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, \dots, W^{[l]}, b^{[l]} \\ \to \beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]}, \dots, \beta^{[l]}, \gamma^{[l]} \end{matrix}\right\}$

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

tf.nn.batch-normalization ←

$\to \beta$

Andrew Ng

Working with mini-batches

$$X^{\{1\}} \xrightarrow{W^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[BN]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \to g^{[1]}(\tilde{z}^{[1]}) = a^{[1]} \xrightarrow{W^{[2]}, b^{[2]}} z^{[2]} \to \dots$$

$$\boxed{X^{\{2\}}} \to z^{[1]} \xrightarrow[BN]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \to \dots$$

$$X^{\{3\}} \to \dots$$

Per minibatch

Parameters: $W^{[l]}, \cancel{b^{[l]}}, \beta^{[l]}, \gamma^{[l]}$.

$(n^{[l]}, 1)$ $\quad (n^{[l]}, 1)$ $\quad (n^{[l]}, 1)$

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

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

$z^{[l]}_{norm}$

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

$z^{[l]}$

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

Andrew Ng

Implementing gradient descent

for $t = 1 \dots$ numMiniBatches

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

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

Use backprop to compute $dW^{[l]}$, $\cancel{db^{[l]}}$, $d\beta^{[l]}$, $d\gamma^{[l]}$

Update params $W^{[l]} := W^{[l]} - \alpha\, dW^{[l]}$

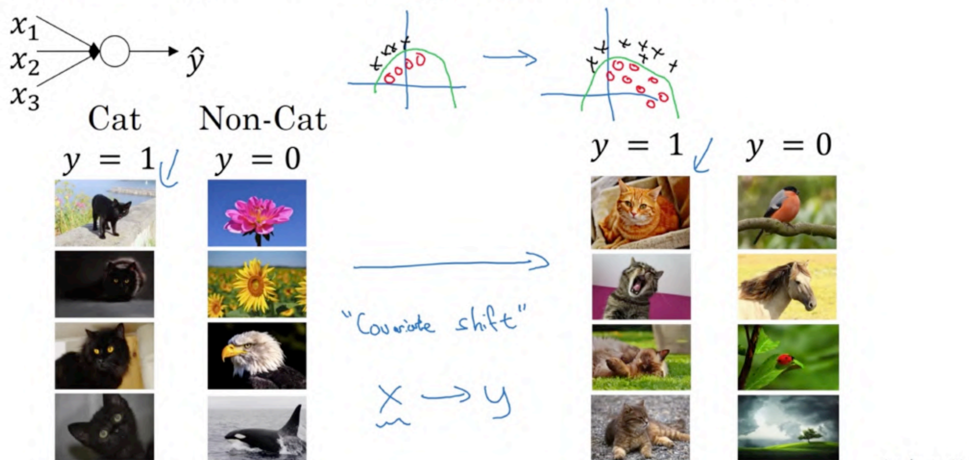
$\beta^{[l]} := \beta^{[l]} - \alpha\, d\beta^{[l]}$

$\gamma^{[l]} := \dots$

Works w/ momentum, RMSprop, Adam.

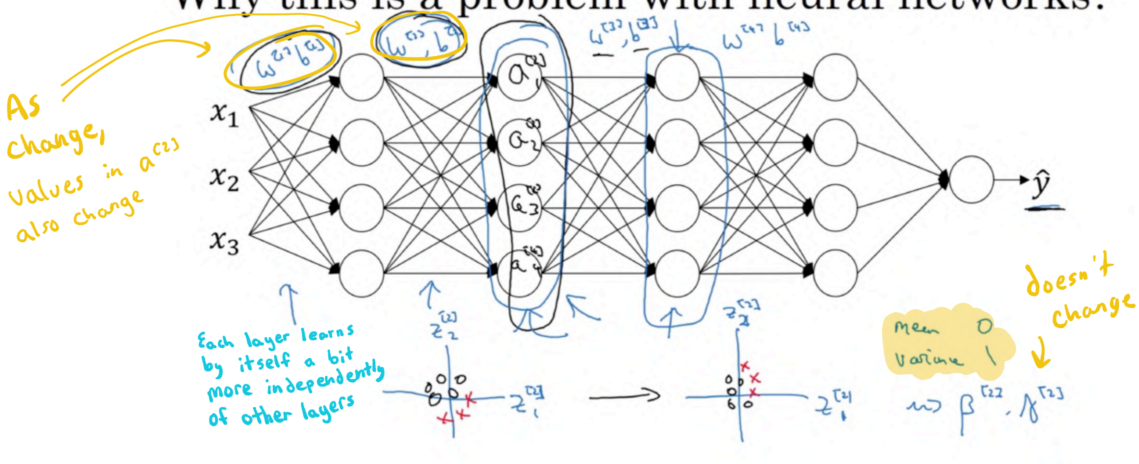
Andrew Ng

Learning on shifting input distribution



Andrew Ng

Why this is a problem with neural networks?



Andrew Ng

Batch Norm as regularization

X $X^{\{t\}}$ $z^{[l]}$

- Each mini-batch is scaled by the mean/variance computed on just that mini-batch.
- This adds some noise to the values $z^{[l]}$ within that minibatch. So similar to dropout, it adds some noise to each hidden layer's activations.
- This has a slight regularization effect.

$\tilde{z}^{[l]}$ 64, 128 μ, σ^2

mini-batch : 64 ⟶ 512

batch norm with dropout for greater regularization effect of batch norm

I wouldn't really use batch norm as a regularizer, that's really not the intent of batch norm, but sometimes it has this extra intended or unintended effect on your learning algorithm. But, really, don't turn to batch norm as a regularization. Use it as a way to normalize your hidden units activations and therefore speed up learning. And I think the regularization is an almost unintended side effect

Andrew Ng

Batch Norm at test time

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

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

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

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

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

$X^{\{1\}}, X^{\{2\}}, X^{\{3\}}, \ldots$

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

θ_1, θ_2, θ_3 σ^2

$\sigma^{2\{1\}[l]}$, $\sigma^{2\{2\}[l]}$, ...

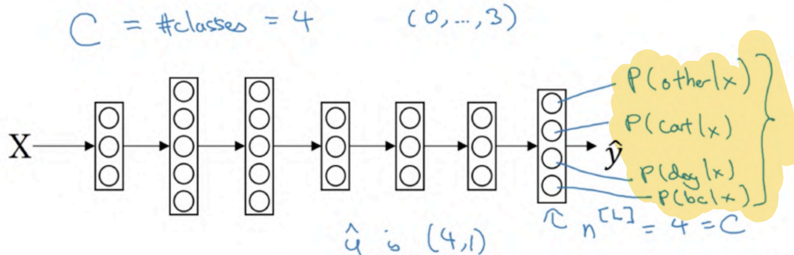
$z_{\text{norm}} = \frac{z-\mu}{\sqrt{\sigma^2+\varepsilon}}$ $\tilde{z} = \gamma z_{\text{norm}} + \beta$

Andrew Ng

Recognizing cats, dogs, and baby chicks , other

3 1 2 0 3 2 0 1

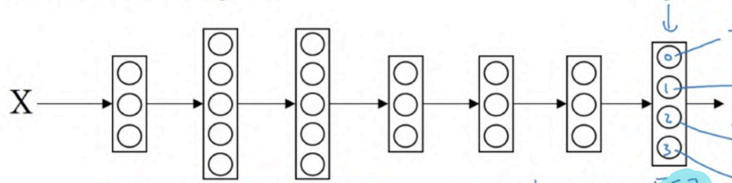
C = #classes = 4 (0, ..., 3)



Andrew Ng

Softmax layer

layer L



$\frac{e^5}{176.3} = 0.842$, $\frac{e^2}{176.3} = 0.042$, $\frac{e^{-1}}{176.3} = 0.002$, $\frac{e^3}{176.3} = 0.114$

$\rightarrow z^{[L]} = W^{[L]} a^{[L-1]} + b^{[L]}$ $\quad (4,1)$

Activation function:

$\rightarrow t = e^{(z^{[L]})}$ $\leftarrow$

$(4,1)$

$\rightarrow \boxed{a^{[L]}} = \frac{e^{z^{[L]}}}{\sum_{j=1}^{4} t_i}, \quad a_i^{[L]} = \frac{t_i}{\sum_{j=1}^{4} t_i}$

$\boxed{(4,1)}$ $\hat{y}$

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

$(4,1)$ $\quad (4,1)$

$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \leftarrow$

$t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix} \rightarrow \sum_{j=1}^{4} t_j = 176.3$

$a^{[L]} = \frac{t}{176.3}$

Andrew Ng

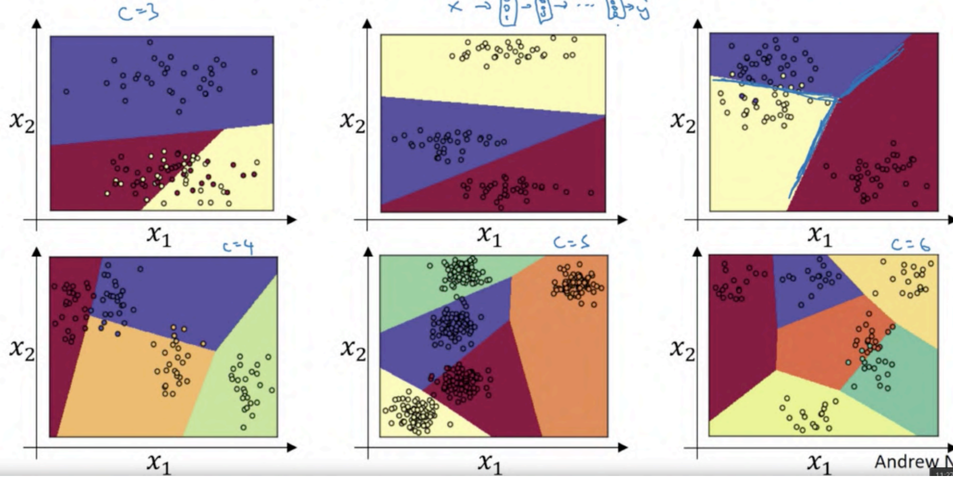
Softmax examples

$x_1, x_2 \rightarrow [\,] \rightarrow \hat{y}$ $\quad z^{[L]} = W^{[L]}x + b^{[L]}$

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

$x \rightarrow [\,] \rightarrow [\,] \rightarrow \cdots [\,] \rightarrow \hat{y}$

C=3 $\quad$ C=4 $\quad$ C=5 $\quad$ C=6



Andrew Ng

Understanding softmax

$(4,1)$

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \qquad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$$

C=4 $\qquad g^{[L]}(\cdot)$

"soft max" $\qquad$ "hard max"

$$a^{[L]} = g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5/(e^5+e^2+e^{-1}+e^3) \\ e^2/(e^5+e^2+e^{-1}+e^3) \\ e^{-1}/(e^5+e^2+e^{-1}+e^3) \\ e^3/(e^5+e^2+e^{-1}+e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix} \qquad \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Softmax regression generalizes logistic regression to C classes.

If C=2, softmax reduces to logistic regression. $a^{[L]} = \begin{bmatrix} 0.842 \\ \cancel{0.158} \end{bmatrix}$

Andrew Ng

Loss function

$$\underset{(4,1)}{y^{(i)}} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \quad \text{— cat}, \quad y_2 = 1, \quad y_1 = y_3 = y_4 = 0$$

$$a^{[L](i)} = \underset{(4,1)}{\hat{y}^{(i)}} = \begin{bmatrix}0.3\\0.2\\0.1\\0.4\end{bmatrix} \qquad C = 4$$

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

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

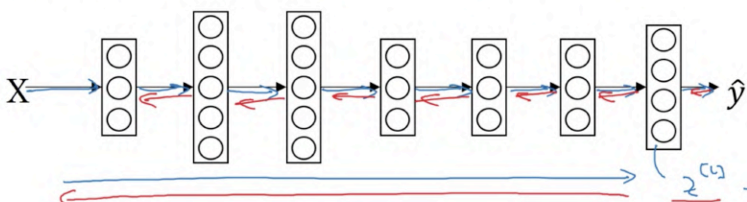
$$-y_2 \log \hat{y}_2 = -\log \hat{y}_2 \rightarrow \text{Make } \hat{y}_2 \text{ big.}$$

$$Y = [y^{(1)} \; y^{(2)} \dots, y^{(m)}] = \begin{bmatrix}0 & 0 & 1 & \dots\\1 & 0 & 0 & \\0 & 1 & 0 & \\0 & 0 & 0 & \end{bmatrix} \quad (4, m)$$

$$\hat{Y} = [\hat{y}^{(1)}, \dots, \hat{y}^{(m)}] = \begin{bmatrix}0.3 & \dots\\0.2 & \\0.1 & \\0.4 & \end{bmatrix} \quad (4, m)$$

Andrew Ng

Gradient descent with softmax



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

Backprop: $\underset{(4,1)}{dz^{[L]}} = \hat{y} - y$

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

Andrew Ng