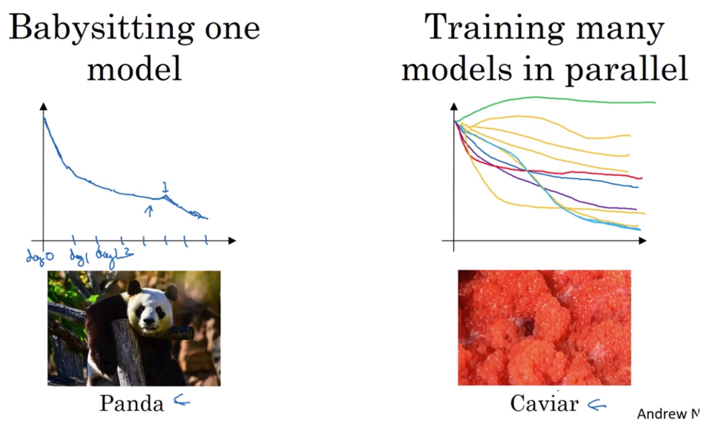
Tuning Process.

- Try Random values, don't use grid
- Coarse to fine

Pick Scale for hyper parameters
- Log scale
- Uniform scale

2 Approach



Batch Normalization
- More robust
- Easier for hyperparam search
- Much bigger range for hyperparam
- Train deep network

Basically normalize each layer for a multilayer neural network so that each layer train faster. Apply normalization to hidden layers

Implementing Batch Norm

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

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

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

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

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

γ, β: learnable parameters of model.

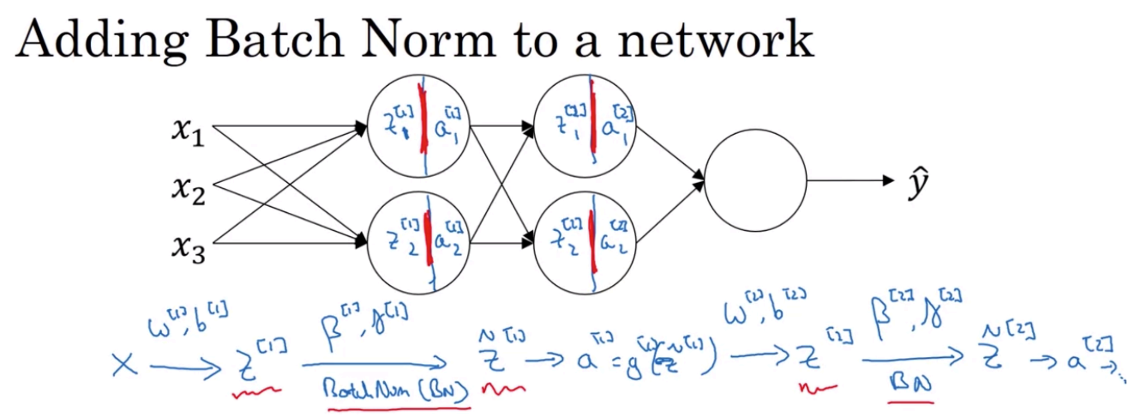
If

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

$$\beta = \boxed{\mu}$$

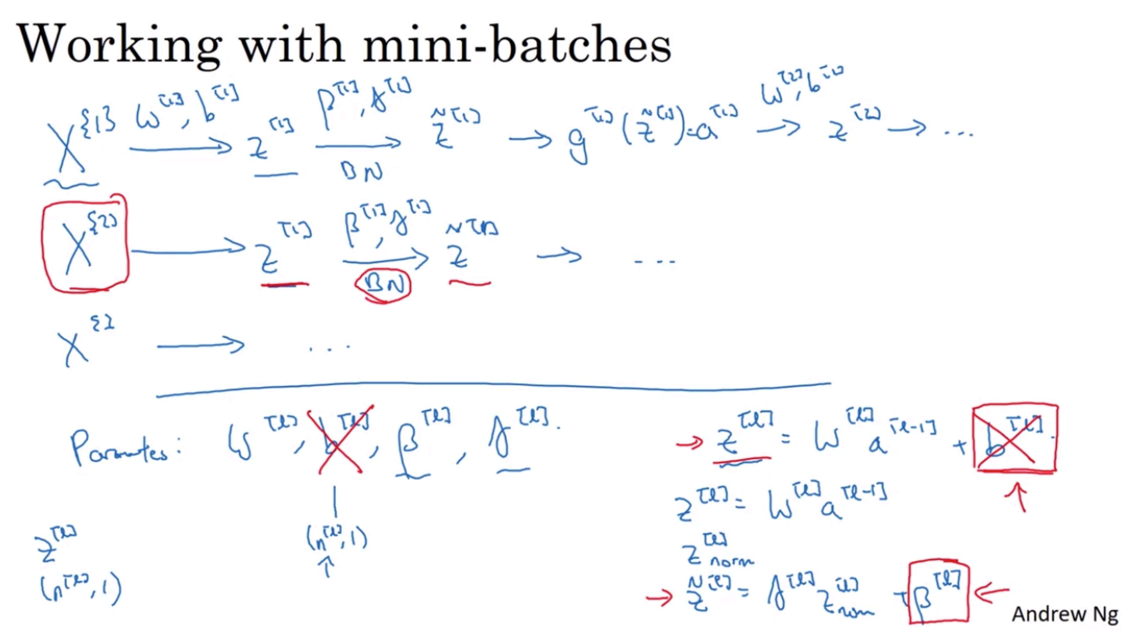
then $\tilde{z}^{(i)} = z^{(i)}$

Essentially computing the identity function. Use Z Tilda _i instead of Z_i



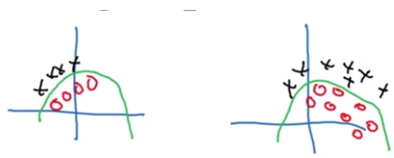
Adding batch norm before activation.

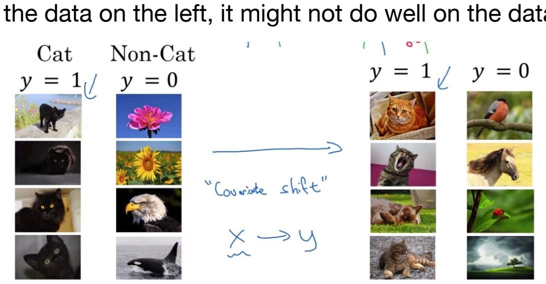
tf.nn.batch_normalization



This is no point keeping bias term, beta for BatchNorm controls the shift.

Covariance shift -> if you learn some mapping (like with the data on the left, it might not do well on the data on the right even though the ground truth remain the same)





Batch norm reduce the intermediate hidden value changes, reduce covariance shift's effect.
Limit the amount the input value changes (earlier layer changes) affect the later layer changes. (More stable)
Make the later layer learn easier.

Batch Norm as regularization

X $X^{\{t\}}$ $\tilde{z}^{[l]}$ $\{64, 128\}$ $z^{[l]}$ μ, σ^2

- 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.

Using a bigger mini batch size, reduce regularization effect. Don't rely on batch norm as a regularization as the effect is not as good as dropout.

Batch Norm at test time

$$\boxed{\begin{aligned}\mu &= \frac{1}{m}\sum_i z^{(i)} \\ \sigma^2 &= \frac{1}{m}\sum_i (z^{(i)}-\mu)^2\end{aligned}}$$

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

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

Mean and variance computed on the entire mini batch. But sometimes at test time, maybe only one instance is available.

Solution. Use Exponentially weighted moving average.

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

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

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

θ_1 θ_2 θ_3 σ^2

$\sigma^{2\{1\}[l]}$ $\sigma^{2\{2\}[l]}$ $\ldots$

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

Epsilon added to variance for numerical stability

Andrew Ng

Softmax vs Hardmax

(4,1)

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

"hard max"

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

- If C = 2, softmax reduces to logistic regression
- Softmax is a generalization of logistic regression to more classes

Loss function

(4,1) $\qquad$ (4,1) $\qquad C = 4$

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

$$a^{[L](i)} = \hat{y}^{(i)} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$$

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

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

$$-y_2 \log \hat{y}_2 = \underline{-\log \hat{y}_2}.$$

Make $\hat{y}_2$ big.

Tensorflow

remember to initialize your variables, create a session and run the operations inside the session.

```
a = tf.constant(2)
b = tf.constant(10)
c = tf.multiply(a,b)
print(c)
```

```
> Tensor("Mul:0", shape=(), dtype=int32)
```

```
sess = tf.Session()
print(sess.run(c))
```

```
> 20
```

Feed data like this, used for feeding in training data

```
x = tf.placeholder(tf.int64, name = 'x')
print(sess.run(2 * x, feed_dict = {x: 3}))
sess.close()
```

```
> 6
```