Hyperparameter Tuning

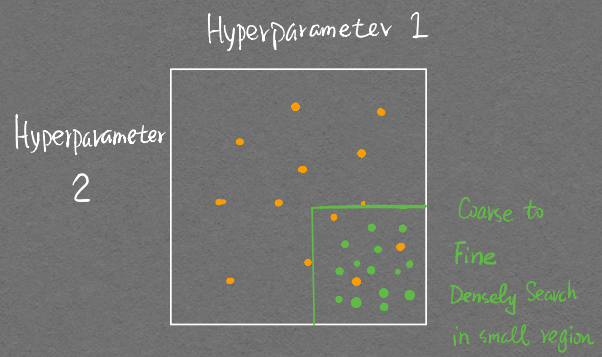
Tuning Process

Learning Rate α

β, # hidden units, mini-batch size

layers, learning rate decay

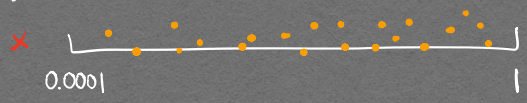
β_1, β_2, ε usually just default value



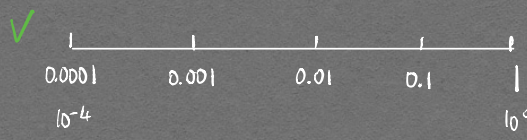
Try randomly sampling instead of grid searching

Using an appropriate scale to pick hyperparameters

learning rate



linear scale (too less smaller values)



log scale

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

$\alpha = 10^r$

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

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

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

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

$1-\beta = 10^r$

$\beta = 1 - 10^r$

10^{-1}	10^{-2}
0.9	0.99
0.1	0.01

$r \in [0, 1)$

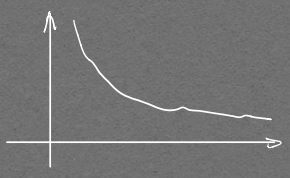
10^{-r}

$10^{-.2-1}$

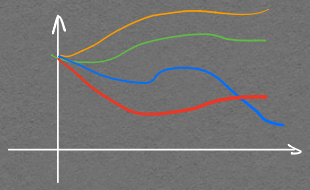
$10^{-1.2}$

In Practice: Pandas vs Caviar

Babysitting one model (not enough computational resource)



Train Many Models in Parallel



Batch Normalization

Normalize Inputs to speed up learning

↳ Can we normalize $a^{[l-1]}$ / $z^{[l-1]}$ to train $w^{[l]}$, $a^{[l]}$ faster?

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

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

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

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

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

$\beta = \mu$

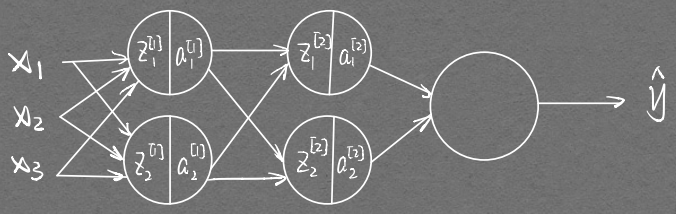
then

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

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

Use $\tilde{z}^{(i)}$ instead of $z^{(i)}_{norm}$ (mean 0, variance 1 is not always desired)

Fitting Batch Norm to a Neural Network



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

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

Parameters: $W^{[1]}, \cancel{b^{[1]}}, W^{[2]}, \cancel{b^{[2]}}, \dots W^{[L]}, \cancel{b^{[L]}}$

$\beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]}, \dots$

$b^{[l]}$ is actually eliminated due to batch normalization

Implementing gradient descent

for $t = 1, 2, \dots,$ # mini batches {

Compute Forward Prop on $X^{\{t\}}$

In each hidden layer, $z^{[l]} \leftarrow \tilde{z}^{[l]}$

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

Update Parameters

}

Why does Batch Norm work?

② Weights in deeper layers are more robust to changes in earlier layers.

"Covariance Shift" If input distribution changes,
even if the mapping does not change,
we might still need to retrain our parameters.

Batch Norm keeps hidden units' mean / variance stable.
Earlier hidden units are limited to shift less.

③ Batch Norm as Regularization

Each mini-batch is scaled by the mean/variance computed on just that mini-batch.
This add some noise to the values of $z^{[l]}$
Batch Norm could go with Dropout.
Larger mini batch size mitigates the regularization effect.

Batch Norm At Test Time

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

$X^{\{1\}} \rightarrow \mu^{\{1\}[l]}, \sigma^{\{1\}[l]}$

$X^{\{2\}} \rightarrow \mu^{\{2\}[l]}, \sigma^{\{2\}[l]}$

μ_{AVG} σ_{AVG}

Multi-class Classification — Softmax Classification

P_1
P_2
P_3
P_4

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

Activation Function

$$t = e^{z^{[l]}} \qquad a^{[l]} = \frac{e^{z^{[l]}}}{\sum_{i=1}^{4} t_i} = \frac{t}{\sum_{i=1}^{4} t_i}$$

Train A Softmax Layer

$$y = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \qquad \hat{y} = \begin{bmatrix} .3 \\ .2 \\ .1 \\ .4 \end{bmatrix}$$

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

$$= -\log \hat{y}_2$$

make $\hat{y}_2$ big

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

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

$$= \begin{bmatrix} 0 & 0 & & 1 \\ 1 & 0 & & 0 \\ 0 & 1 & \cdots & 0 \\ 0 & 0 & & 0 \end{bmatrix}$$

$$\hat{Y} = \begin{bmatrix} .2 & .3 & & .9 \\ .4 & .3 & & .1 \\ .2 & .4 & \cdots & 0 \\ .2 & 0 & & 0 \end{bmatrix}$$

$dz^{[2]} = \hat{y} - y$ for Backprop

Programming Frameworks

Deep Learning Frameworks

Pytorch

Tensorflow

keras

Ease of Programming

Truly Open Source with Good Governance

Motivating Problem

$J(w) = w^2 - 10w + 25$

w tf.variable for parameters

coefficients = np.array( )

x = tf.placeholder( )

```
cost = ...
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
init = tf.global_variables_initializer()
sess = tf.session
sess.run(init)
for i in rang(1000):
    sess.run(train, feed_dict = {x: coefficients})
    print(sess.run(w))
```