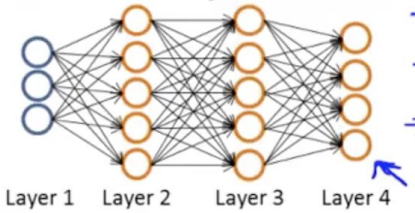
# Week 5 Neural Networks: Learning

## Cost Function and Backpropagation

### Cost Function

**Neural Network (Classification)**




$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

$L =$ total no. of layers in network $L = 4$

$s_l =$ no. of units (not counting bias unit) in layer $l$ $s_1 = 3,\ s_2 = 5,\ s_4 = s_L = 4$

Binary classification

$y = 0$ or $1$

1 output unit

$h_\Theta(x) \in \mathbb{R}$

$s_L = 1, \quad K = 1$

Multi-class classification (K classes)

$y \in \mathbb{R}^K$ E.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

K output units

$h_\Theta(x) \in \mathbb{R}^K$

$S_L = K \quad (K \geq 3)$

**Cost function**

Logistic regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

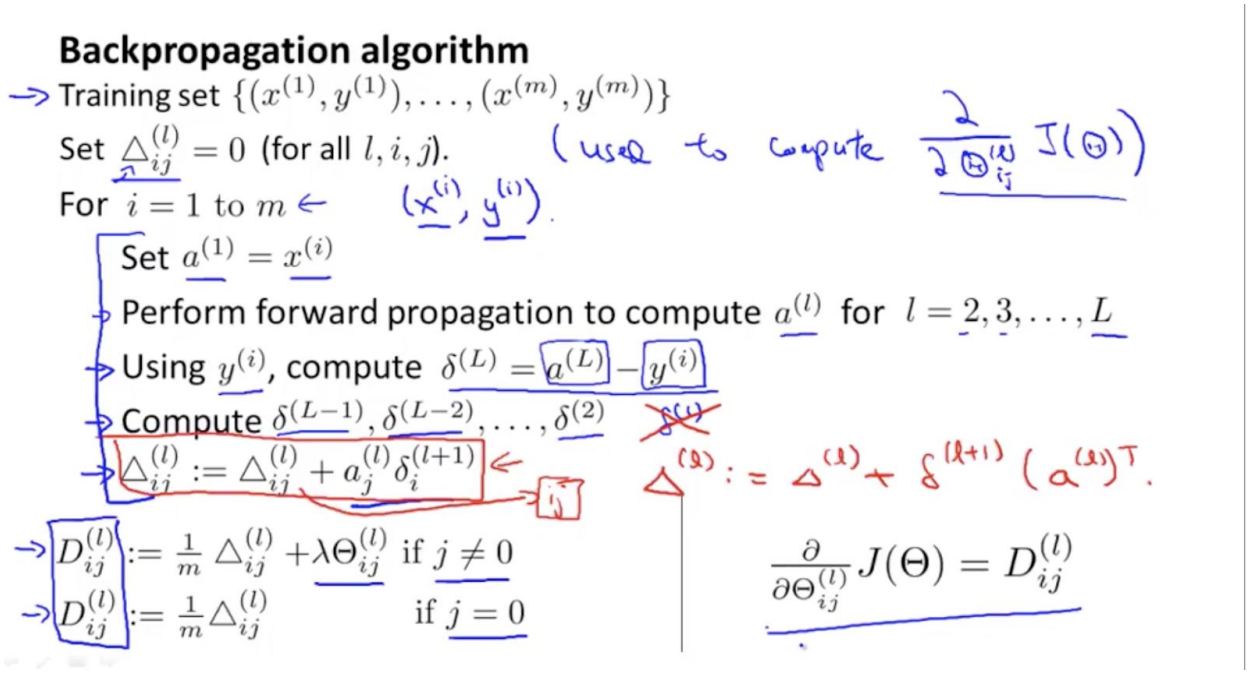
Neural network:

$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th}$ output

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

$$+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

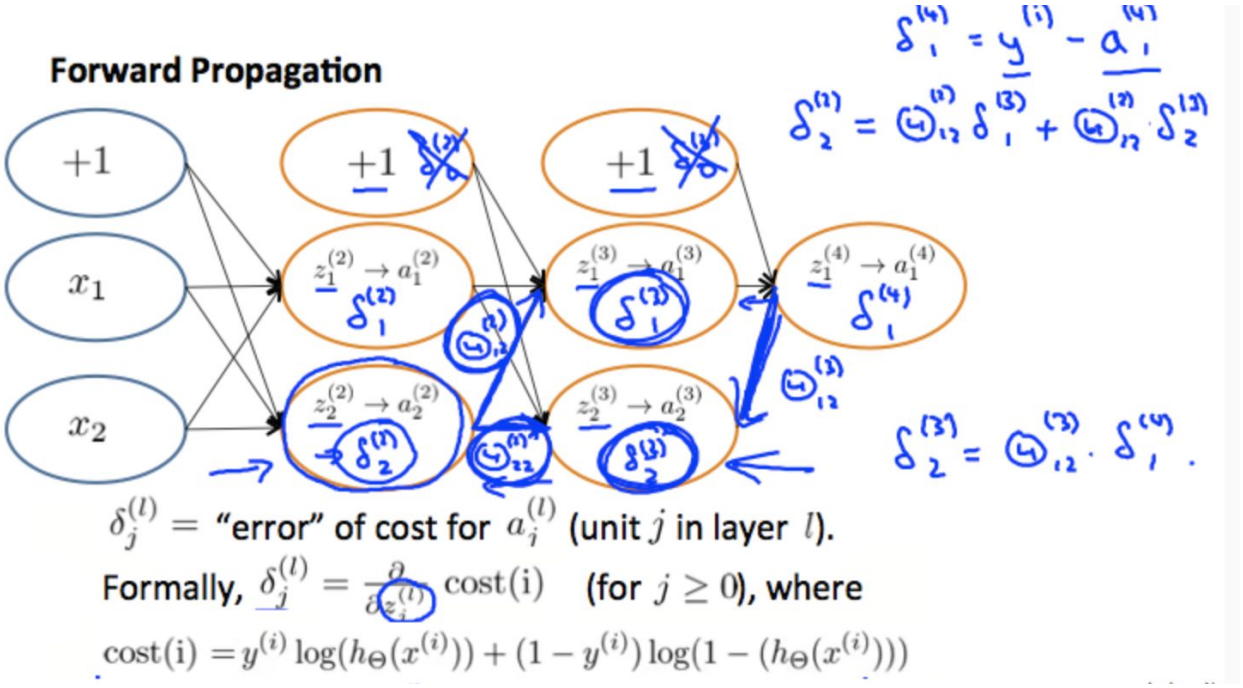
# Backpropagation Algorithm




Suppose you have two training examples $(x^{(1)}, y^{(1)})$ and $(x^{(2)}, y^{(2)})$. Which of the following is a correct sequence of operations for computing the gradient? (Below, FP = forward propagation, BP = back propagation).

- FP using $x^{(1)}$ followed by FP using $x^{(2)}$. Then BP using $y^{(1)}$ followed by BP using $y^{(2)}$.
- FP using $x^{(1)}$ followed by BP using $y^{(2)}$. Then FP using $x^{(2)}$ followed by BP using $y^{(1)}$.
- BP using $y^{(1)}$ followed by FP using $x^{(1)}$. Then BP using $y^{(2)}$ followed by FP using $x^{(2)}$.
- FP using $x^{(1)}$ followed by BP using $y^{(1)}$. Then FP using $x^{(2)}$ followed by BP using $y^{(2)}$.

## Backpropagation Intuition




# Backpropagation in Practice

## Implementation Note: Unrolling Parameters

**Learning Algorithm**

- Have initial parameters $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$.
- Unroll to get `initialTheta` to pass to
- `fminunc(@costFunction, initialTheta, options)`

```
function [jval, gradientVec] = costFunction(thetaVec)
```

From `thetaVec`, get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$.

Use forward prop/back prop to compute $D^{(1)}, D^{(2)}, D^{(3)}$ and $J(\Theta)$.

Unroll $D^{(1)}, D^{(2)}, D^{(3)}$ to get `gradientVec`.

## Gradient Checking

```
epsilon = 1e-4;

for i = 1:n,

  thetaPlus = theta;

  thetaPlus(i) += epsilon;

  thetaMinus = theta;

  thetaMinus(i) -= epsilon;

  gradApprox(i) = (J(thetaPlus) - J(thetaMinus))/(2*epsilon)

end;
```

Once you have verified **once** that your backpropagation algorithm is correct, you don't need to compute gradApprox again. The code to compute gradApprox can be very slow.

## Random Initialization

- All zero initialization: symmetry breaking
-

  ```
  %If the dimensions of Theta1 is 10x11, Theta2 is 10x11 and Theta3 is 1x11.


  Theta1 = rand(10,11) * (2 * INIT_EPSILON) - INIT_EPSILON;

  Theta2 = rand(10,11) * (2 * INIT_EPSILON) - INIT_EPSILON;

  Theta3 = rand(1,11) * (2 * INIT_EPSILON) - INIT_EPSILON;
  ```

rand(x,y) is just a function in octave that will initialize a matrix of random real numbers between 0 and 1.

## Putting It Together

1. **Randomly initialize the weights**
2. **Implement forward propagation to get** $h_\Theta(x^{(i)})$ **for any** $x^{(i)}$
3. **Implement the cost function**
4. **Implement backpropagation to compute partial derivatives**
5. **Use gradient checking to confirm that your backpropagation works. Then disable gradient checking.**
6. **Use gradient descent or a built-in optimization function to minimize the cost function with the weights in theta.**

**When we perform forward and back propagation, we loop on every training example:**

**for i = 1:m,**

**Perform forward propagation and backpropagation using example (x(i),y(i))**

**(Get activations a(l) and delta terms d(l) for l = 2,...,L**

# Application of Neural Networks

Auto driving

# Week 6:Advice for Applying Machine Learning

## Evaluating a Learning Algorithm

### Deciding What to Try Next

### Evaluating a Hypothesis

### Model Selection and Train/Validation/Test Sets

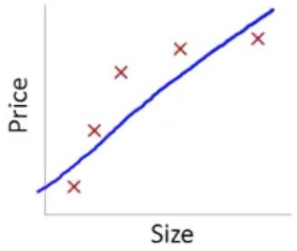
1. Optimize the parameters in Θ using the training set for each polynomial degree.
2. Find the polynomial degree d with the least error using the cross validation set.
3. Estimate the generalization error using the test set with $J_{test}(\Theta^{(d)})$, (d = theta from polynomial with lower error);

- The training set is used to train your model, that is, determine the best parameters (theta values) that fit your data.
- The Validation set is used to validate your model and choose the values of the regularization term (lambda).
- The Test set acts as a proxy to your real data and should be used to ensure that your model will generalize well (not to much error in prediction)
- If you use the Test set to validate your model and find the regularization term then you will have an overoptimistic model because you choose the regularization term that best fit your test data. If your Test set contains all types of values your model will ever encounter that is perfectly fine, however most of the time this is not the case. So you use the validation set to find the best regularized model and test it against the test set.

# Bias vs. Variance

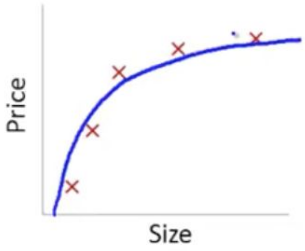
## Diagnosing Bias vs. Variance

**Bias/variance**



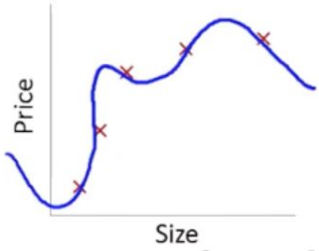

$\theta_0 + \theta_1 x$

High bias
(underfit)



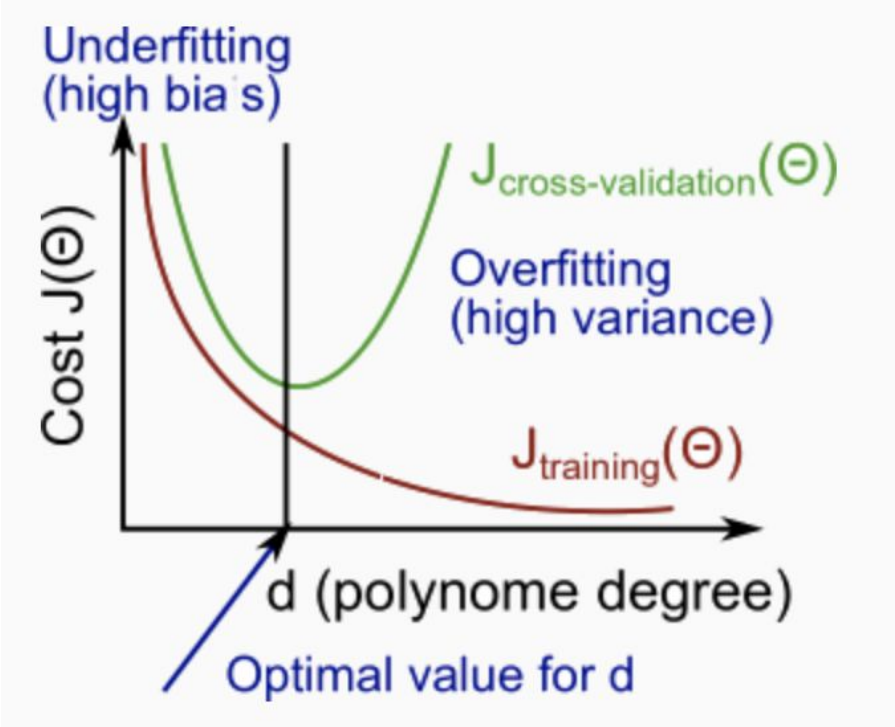

$\theta_0 + \theta_1 x + \theta_2 x^2$

"Just right"

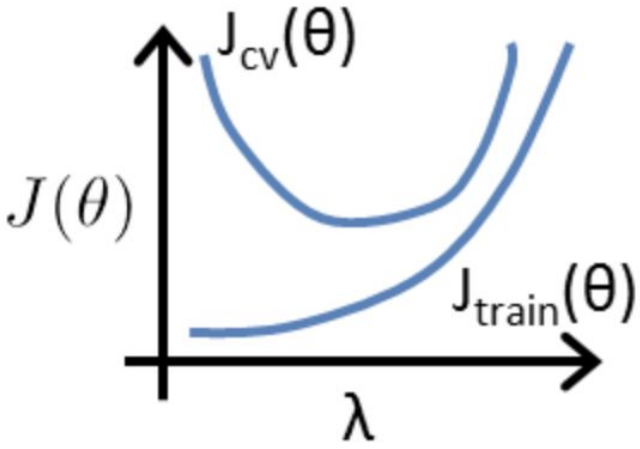



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

High variance
(overfit)

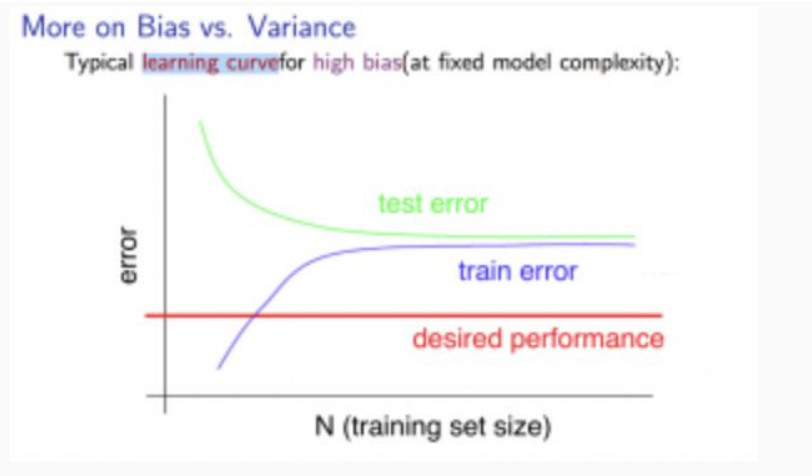


## Regularization and Bias/Variance

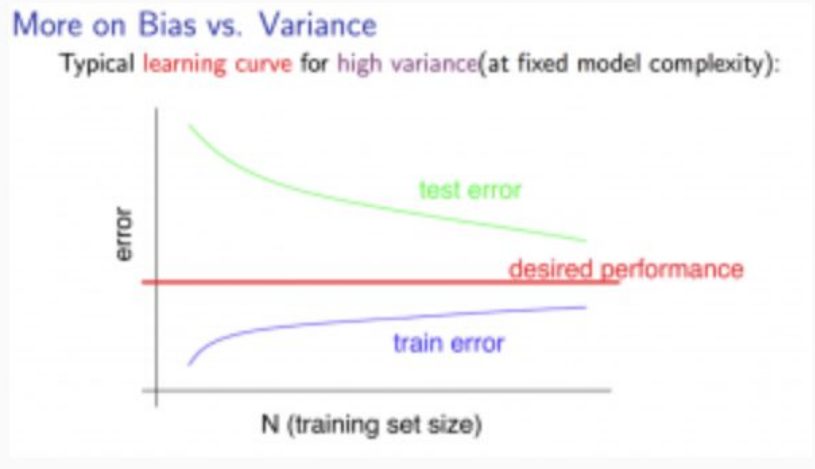



## Learning Curves

If a learning algorithm is suffering from **high bias**, getting more training data will not **(by itself)** help much.



If a learning algorithm is suffering from **high variance**, getting more training data is likely to help.




## Deciding What to Do Next Revisited

- **Getting more training examples:** Fixes high variance
- **Trying smaller sets of features:** Fixes high variance
- **Adding features:** Fixes high bias
- **Adding polynomial features:** Fixes high bias
- **Decreasing λ:** Fixes high bias
- **Increasing λ:** Fixes high variance.

# Building a Spam Classifier

## Prioritizing What to Work On

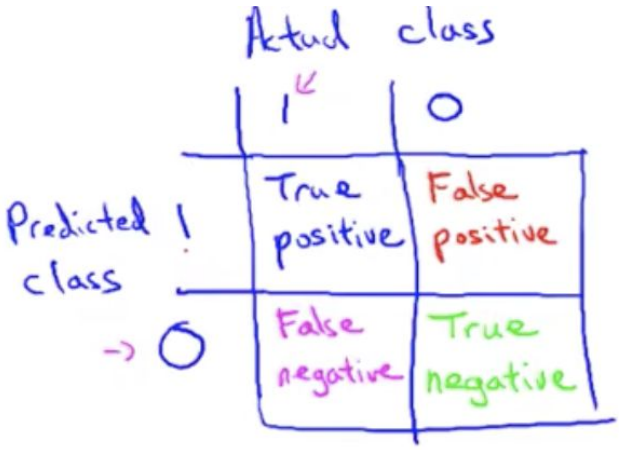
- Collect lots of data (for example "honeypot" project but doesn't always work)
- Develop sophisticated features (for example: using email header data in spam emails)
- Develop algorithms to process your input in different ways (recognizing misspellings in spam).

## Error Analysis

# Handling Skewed Data

## Error Metrics for Skewed Classes

Skewed class: the number in one category is much smaller than another

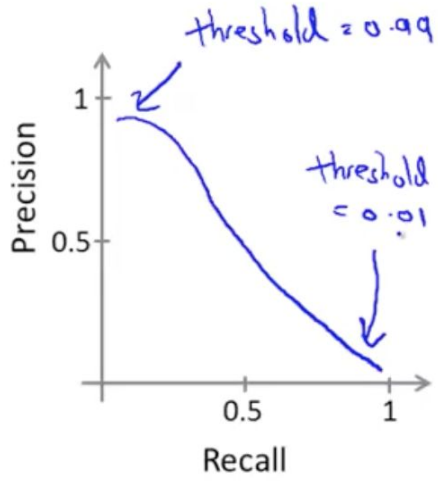



$$\text{Precision} = \frac{\text{True positives}}{\text{\# predicted as positive}} = \frac{\text{True positives}}{\text{True positives + False positives}}$$

$$\text{Recall} = \frac{\text{True positives}}{\text{\# actual positives}} = \frac{\text{True positives}}{\text{True positives + False negatives}}$$

- Accuracy = (true positives + true negatives) / (total examples)

## Trading Off Precision and Recall




threshold

Be confident: Predict 1 if >0.9 -> higher precision and lower recall

Avoid missing: predict 1 if >0.3 -> lower precision and higher recall

$F_1$ Score: $2\frac{PR}{P+R}$

# Using Large Data Sets

## Data For Machine Learning

Fix high bias: enough features

Fix high variance: enough sample