Understanding Andrew Ng's Machine Learning Course – Notes and codes

(Matlab version)

Note: All source materials and diagrams are taken from the Coursera's lectures created by Dr Andrew Ng. Everything I have written below is learnt and compiled from the course's materials and programming assignments.

Supervised Learning

A. Linear Regression + Gradient Descent + Regularization

The essence of cost function and gradient descent

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

- Cost function is a function of the squared error (difference between predicted h(x) and actual training example y value)
- Our goal is to minimize the cost function (minimize the errors between prediction and actual)
- Gradient descent is used to minimize cost function and find the optimal theta parameters and thus the optimal hypothesis function that will predict the most accurate h(x) values

- Outline of gradient descent method:
  1. Start with some theta (matrix of theta1 – theta(n))
  2. Keep changing theta to reduce J cost function
  3. Simultaneously update theta

     $$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1), \text{ for } (j = 0, j = 1)$$

  4. Until we converge to a minimum J cost

- Choice of alpha learning rate variable:
  - If too small – gradient descent convergence is very slow
  - If too large – J cost function may not even converge
  - Try choosing these numbers: 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, …
  - Convergence test: We can declare that J is converging if J decreases by less than 0.001 in one iteration

From ML Ex. 5

Hypothesis:

```
h = X * theta ;
```

- Training set consists of (X, y) data points
- Vectorized form, whereby X is a row matrix of all X1 to Xm and theta is a column matrix of all theta(1) to theta(n)

Cost function:

```
J = (1 / (2*m))* sum((h - y).^2) + (lambda / (2*m)) * sum(theta(2:size(theta)).^2);
```

- Blue part: Sum of all squared differences of the h predicted values and the actual y values
- Yellow part: Regularization term
  - theta(2:size(theta)) omits the first theta1 parameter as it is the bias term θ_0 (it is theta1 on matlab as index starts at 1 on matlab instead of 0 mathematically)
  - Sum up all the squared theta parameters

Gradient:

```
grad = (1/m)* X' * (h - y);
grad(2:size(theta)) = grad(2:size(theta)) + (lambda/m) * theta(2:size(theta));
```

- Gradient descent term that is multiplied by an alpha (learning rate) term before being subtracted from theta to converge towards a minimum theta parameter set that will give us an optimized hypothesis
- The grad term does not include the alpha learning rate

Feature Scaling

Many variables *x* may have vastly different scales, thus it is needed to normalize every feature within the same range to allow the algorithm to reach convergence more efficiently.

- Subtract each feature by its mean and dividing by its standard deviation

Normal Equation Method (instead of gradient descent optimization)

- With this, there is no need for iteration, no need alpha learning rate, and no need feature scaling. However, this method is slow if n is large
- Compute the inverse[2] of a n x n matrix: $(X^TX)^{-1} \sim O(n^3)$.

$$X^TX\theta = X^Ty$$

$$\theta = (X^TX)^{-1}X^Ty$$

B. Logistic Regression + Advanced Optimization + Regularization

The essence of cost function and advanced optimization techniques

Unlike linear regression, the hypothesis function of logistic regressions includes a sigmoid function:

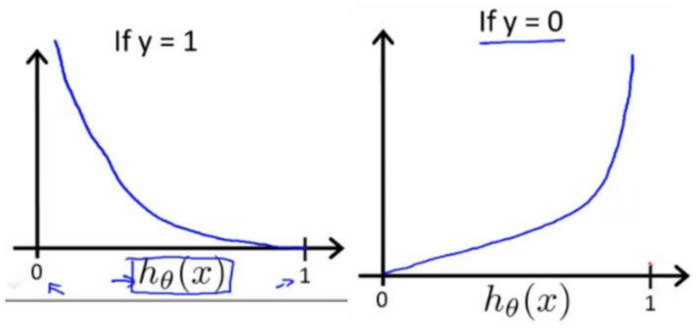
$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}.$$

$$h_\theta(x) = P(y = 1 | x; \theta),$$
$$P(y = 0 | x; \theta) + P(y = 1 | x; \theta) = 1$$

- In which the hypothesis will generate a probability that y = 1 (or y = 0) given input x, with the threshold set to a certain number between 0 and 1. For example, we can predict y = 1 if h(x) => 0.5 or 0.7, and y = 0 if h(x) < 0.5 or 0.3 respectively.

Intuition for cost functions

- Generally, we can think cost function as a penalty of the incorrect classification by our hypothesis. In linear regressions, we use the squared error cost function.
- In logistic regressions, the sigmoid function h(x) is a complex function and thus will not generate a convex shape graph. It will be challenging to use gradient descent or other optimization functions to minimize the cost with a non-convex curve.



- With logarithmic functions, we penalize the learning algorithm by a large cost (up to infinity) if h(x) predicts 0 but y = 1, and if h(x) predicts 1 but y = 0 respectively. The cost will be 0 if h(x) predicts accurately to the y value in the training examples. The below cost function has added in the logarithmic functions for logistic regression.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)}) = -\frac{1}{m} \left[\sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$$

Gradient descent for logistic regression

- The gradient descent algorithm looks identical to that of linear regression, however, the hypothesis function h(x) is no longer linear but a sigmoid function instead

Advanced optimization methods

- Many optimization algorithms include conjugate gradient, BFGS, and L-BFGS which can minimize the cost function and they are more efficient to compute, thus are preferred to gradient descent
- In matlab, there is a built-in optimization function fminunc()

fminunc()?

- It is an optimization function in matlab which finds the minimum of unconstrained multi-variable function (aka the min point of cost function J)
- fmincg() is a more efficient method for a large number of parameters and it is similar to fminunc()

Utilizing fminunc() on matlab:

1. Calculate the cost function J(θ)
2. Calculate the gradient functions (grad) ; which is just the derivative of cost function J
3. Give the initial theta value
4. Call the built-in functions optimiset() and fminunc():

```
function [jVal, gradient] = costFunction(theta)
jVal = (theta(1)-5)^2 + (theta(2)-5)^2;
gradient = zeros(2,1);
gradient(1) = 2*(theta(1)-5);
gradient(2) = 2*(theta(2)-5);

options = optimset('GradObj', 'on', 'MaxIter', '100');
initialTheta = zeros(2,1);
[optTheta, functionVal, exitFlag]  = fminunc(@costFunction, initialTheta, options);
```

From ML Ex. 2 and Ex. 3

Hypothesis:

```
h = sigmoid(X * theta);
```

- sigmoid function is defined as sigmoid = 1./ (1 + (e.^(-z)))

```
theta_new = [0; theta(2:length(theta))] ;
```

- This replaces the first row of theta with 0 while keeping the rest, thus allowing us to ignore theta1 (θ_0)

Cost function:

```
J = (-1 / m) * (y' * log(h) + (1 - y') * log(1 - h)) + (lambda / (2*m)) * sum(theta_new .^2) ;
```

- Blue part: Generates value from 0 to 1 (cost)
- Yellow part: Regularization term
    - Sum up all the squared theta parameters

Gradient:

```
grad = (1 / m) * ( (X' * (h - y)) + lambda * theta_new);
```

Multiclass classification (oneVsAll) add-on codes

From ML Ex. 3

```
initial_theta = zeros(n + 1, 1);
options = optimset('GradObj', 'on', 'MaxIter', 50);
```

- Set the initial theta to a single column of n+1 zeros (extra 0 for bias theta)
- Use fmincg as an alternative to fminunc optimization, and set the above options for Matlab

```
for c = 1 : num_labels

        [theta] = fmincg( @(t) (lrCostFunction(t, X, (y == c), lambda)), initial_theta, options);
        all_theta(c, :) = theta' ;

end;
```

- num_labels = k number of digits (0, 1, 2, …, 9)
- Loop over the 10 different classes of numbers (and using fmincg to optimize the training sets to the num_labels)
- The fmincg(@(t) …) function will return the theta and the costs
- theta(:) will return a column vector of theta
- all_theta(c, :) = theta' will save theta for each C class as a new row in all_theta

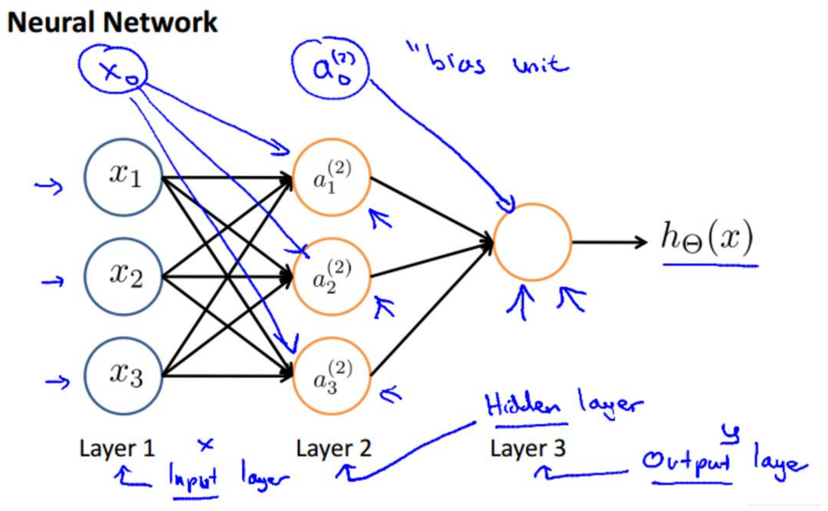
```
sigmoid(X * all_theta')
[max_value, p] = max(h, [], 2);
```

- h = These functions will return a vector of predictions (h) for each of the example in the matrix X
- The max function will return the index of the max element
  - max(h, [], 2) will return the max for each row
- All in all, these functions will predict the label for a trained one-vs-all classifier where the label is from the range of values in K (1 to 10), where 10 is substitute label for 0

C. Neural Networks, Backpropagation, and Optimizations

The essence of NNs, forward propagations, backpropagations, and all the steps to optimizations

Neural networks aim to replicate the brain's one learning algorithm to learn information, with input layers, hidden layers, and an output layer that generates the h(x) hypothesis value.



Use the following notation convention:

- $a_i^{(j)}$ to represent the "activation" of unit i in layer j
- $\Theta^{(j)}$ to represent the matrix of weights controlling function mapping from layer j to layer $j+1$

The value at each nodes can be calculated as

$$a_1^{(2)} = g(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)})$$

If the network has s_j units in layer `j`, s_{j+1} units in layer `j+1`, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$. It could be interpreted as **"the dimension of $\Theta^{(j)}$ is number of nodes in next layer's $\times$ the current layer's node + 1".**

All steps to train a Neural Network

Codes all are from ML Ex. 4

1. Randomly initialize the weights (theta)

Initializing all theta weights to zero (as usual) does not work with NNs because when we back propagate, all nodes will update to the same value repeatedly. As such, we will randomly initialize our weights with each theta to a random value between [−ϵ,ϵ].

```
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(10,11) generates a random 10x11 matrix of values between 0 to 1 for each cell

2. Forward propagation to get h(x) for any x^i

```
h = sigmoid(z) ;
g = h .* (1-h) ;
```

Refer to codes in step 4

3. Compute cost function J

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\Theta\left(x^{(i)}\right)\right)_k + \left(1 - y_k^{(i)}\right)\log\left(1 - \left(h_\Theta\left(x^{(i)}\right)\right)_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{l}\right)^2$$

- The summation from k = 1 to K: sums up all k output units
- We do not sum over theta = 0 and the bias terms (a0 and x0)
- L = total number of layers on the network
- S_l = number of units in the layer l (less the bias unit)

```
X = [ones(m, 1), X] ;
a1 = X ;

z2 = Theta1 * X' ;
a2 = sigmoid(z2) ;

a2 = [ones(m, 1), a2'] ;
z3 = Theta2 * a2' ;
a3 = sigmoid(z3) ;
h = a3 ;
```

```
y_i = zeros(num_labels, m) ;

for i = 1 : m

            y_i(y(i), i) = 1 ;
end

sum1 = sum(sum( (-y_i .* log(h)) - ((1 - y_i) .* log(1-h)) ) ) ;
J_i = sum1 / m ;


% Regularized cost function


Theta1_reg = sum( sum( Theta1(:, 2:end).^2)) ;
Theta2_reg = sum( sum( Theta2(:, 2:end).^2)) ;

J = J_i + (lambda/(2*m)) *  (Theta1_reg + Theta2_reg) ;
```

4. Backpropagation

To get partial derivatives (D), loop through and perform forward then backpropagation to obtain activations and derivatives for all layers.

```
for u = 1:m

   % Forward propagation

   a1 = X(u, :) ;  % X includes first column of bias
   z2 = Theta1 * a1' ;
   a2 = sigmoid(z2) ;
   a2 = [1; a2] ;  % adds bias

   z3 = Theta2 * a2 ;
   a3 = sigmoid(z3) ; % which is h

   % Backpropagation starts

   z2 = [1; z2] ; % bias!
   d3 = a3 - y_i(:, u) ;  % to get columns of u th element
   d2 = (Theta2' * d3) .* sigmoidGradient(z2) ;
   d2 = d2(2: end) ;

   Theta2_grad = Theta2_grad + (d3 * a2') ;
   Theta1_grad = Theta1_grad + (d2 * a1) ;  %% Why does this not require transposing of a1???

end
```

```
Theta2_grad = Theta2_grad / m ;
Theta1_grad = Theta1_grad / m ;


% Regularized backpropagation

Theta1_reg2 = Theta1(: , 2:end) * lambda/m ;
Theta2_reg2 = Theta2(: , 2:end) * lambda/m ;
Theta1_grad(:, 2:end) = Theta1_grad(:, 2:end) + Theta1_reg2 ;
Theta2_grad(:, 2:end) = Theta2_grad(:, 2:end) + Theta2_reg2 ;
```

5. **Unroll theta and D parameters to get thetaVec and DVec**

```
function [jVal, gradient] = costFunction(theta)
…
optTheta = fminunc(@costFunction, initialTheta, options)



thetaVec = [Theta1(:); Theta2(:); Theta3(:)];
DVec = [D1(:); D2(:); D3(:)];
```

- The first portion of codes compute for us gradient, theta, and initialTheta in vectors of n+1 dimensions
- In order to use optimization functions like fminunc(), it is necessary to unroll all parameters and put them into one long vector

6. **Gradient checking, then disable gradient checking codes**

```
gradApprox = (J(theta + epsilon) - J(theta - epsilon)) / (@ * epsilon);

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;
```

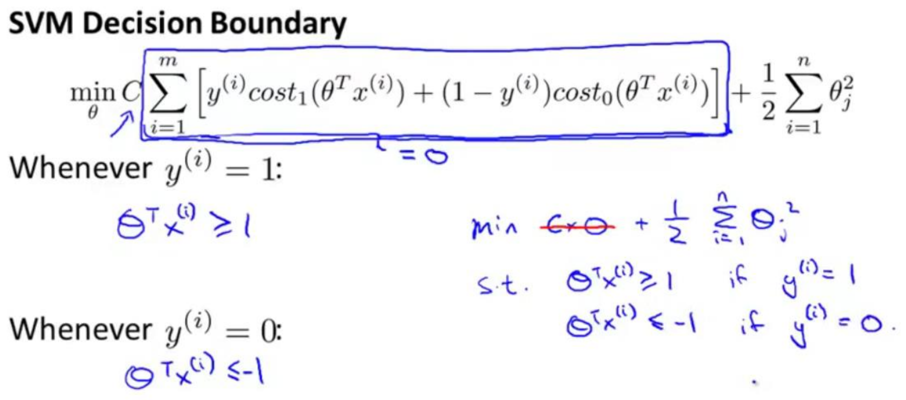
- Check that gradApprox ~ DVec

7. **Use gradient descent / advanced optimizations with backpropagation to minimize J as a function of parameters theta**

D. Support Vector Machines

The essence of SVMs and Kernels (similarity functions)

- Generally, the cost function for SVM looks similar to that for logistic regression. The shape also resembles the ReLu function (check out deeplearning.ai course)



SVM Decision Boundary

$$\min_{\theta} C \boxed{\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right]} + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

$= 0$

Whenever $y^{(i)} = 1$:

$\theta^T x^{(i)} \geq 1$

$\min$ ~~$C \times 0$~~ $+ \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$

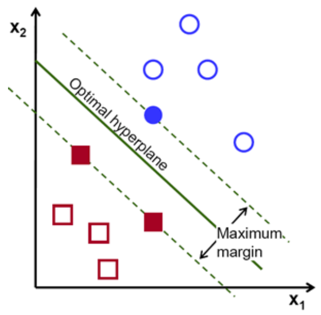
s.t. $\theta^T x^{(i)} \geq 1$ if $y^{(i)} = 1$

$\theta^T x^{(i)} \leq -1$ if $y^{(i)} = 0$.

Whenever $y^{(i)} = 0$:

$\theta^T x^{(i)} \leq -1$

- C is equivalent to 1/(lambda) in logistic regression. When C is not very large, we will have large margin properties, in which we can classify items with large margins between them (sometimes ignoring outliers)

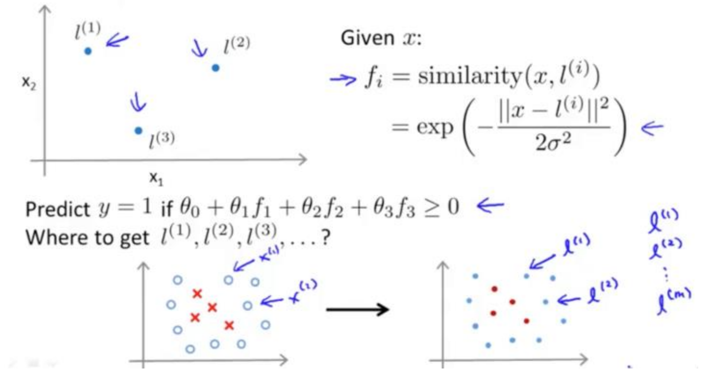


Kernels (similarity functions)

- A kernel function takes 2 inputs and returns a number representing how similar they are
- Gaussian kernel function refers to this form
- If x is near to a certain landmark reference point, k ~ 1
  Otherwise, k ~ 0 (not similar)

$$k(x, \ell^{(i)}) = \exp(-\frac{\|x - \ell^{(i)}\|^2}{2\delta^2})$$

Choosing landmarks → We choose the landmarks with exact positions



Given x:

$$f_i = \text{similarity}(x, l^{(i)}) = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$$

Predict $y = 1$ if $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

Where to get $l^{(1)}, l^{(2)}, l^{(3)}, \ldots$?

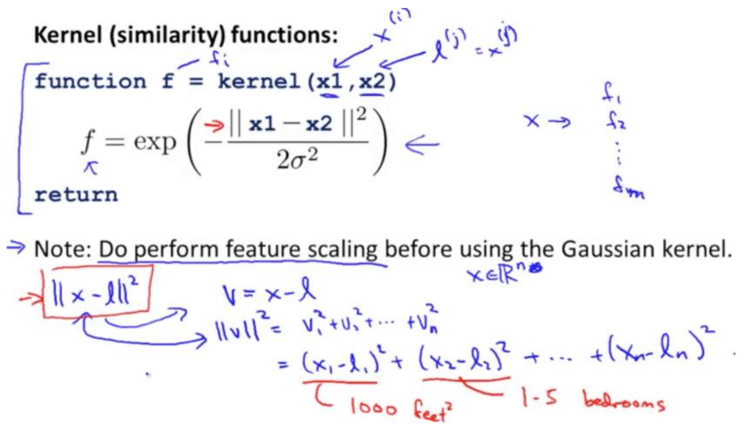
SVM parameters and bias/variance *(refer to Part E)*

- Large C = Low bias, high variance (~ overfitting)
- Large σ^2 = Features vary more smoothly (higher bias, lower variance)

SVMs in practice

- We do not usually write our own codes or software to optimize and solve for theta ourselves. We use SVM software packages like liblinear or libsvm
- When using these packages, we need to specify the following:
  - Choice of parameter C
  - Choice of kernel
    - No kernel ('linear kernel') – Predict y = 1 if theta.T (X) >= 0 ; choose this if n is large with lots of features but m is small (lacking in training examples)
    - Gaussian kernel – Need to choose σ^2 as well; We choose this if n is small and/or m is large

Matlab implementation



Kernel (similarity) functions:

```
function f = kernel(x1,x2)
```

$$f = \exp\left(-\frac{\| \mathbf{x1} - \mathbf{x2} \|^2}{2\sigma^2}\right)$$

```
return
```

Note: Do perform feature scaling before using the Gaussian kernel.

f_i
$x^{(i)}$
$l^{(j)} = x^{(j)}$
$x \rightarrow f_1, f_2, \vdots, f_m$
$x \in \mathbb{R}^n$
$\boxed{\|x - l\|^2}$
$v = x - l$
$\|v\|^2 = v_1^2 + v_2^2 + \cdots + v_n^2$
$= (x_1 - l_1)^2 + (x_2 - l_2)^2 + \cdots + (x_n - l_n)^2$
1000 feet^2
1-5 bedrooms

Note: We need to perform ***feature scaling (normalization)*** to get all features to the same scale before using Gaussian kernel

Logistic regression vs SVMs – Choose this if …

	Logistic regression	**SVMs**
If n is large relative to m (10000 > 1000)	Yes	Linear kernel
If n is small, m is intermediate (n: 1 – 1000; m: 10 – 10000)		Gaussian kernel
If n is small, m is large (n: 1 – 1000; m: 50000+)	Yes	Linear kernel

For NNs: Likely to work well for most of these above 3 settings, but might be slower to train when there are a lot of features and training examples (since a deep NN with many hidden layers is involved)

From ML Ex. 6

Gaussian kernel function

```
sim = exp(-sum((x1-x2).^2)/(2*sigma^2));
```

Finding the most optimal parameters (C and sigma for Gaussian kernel)

```
test = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30];
results = [];

for Ci = 1:8
    for Sigmai = 1:8

        Ctest = test(Ci);
        Sigmatest = test(Sigmai);

        model = svmTrain(X, y, Ctest, @(x1, x2) gaussianKernel(x1, x2, Sigmatest));
        predictions = svmPredict(model, Xval);

        testError = mean(double(predictions ~= yval));

        results = [results; Ctest, Sigmatest, testError];
    end
end

[minError, minIndex] = min(results(:, 3)) ;

C = results(minIndex, 1);
sigma = results(minIndex, 2);
```

E. Debugging Learning Algorithms

The essence of regularization, bias/variance, and precision/recall

Machine learning diagnostics

A test that we can run to gain insight what is and what is not working with a learning algorithm, and thus gain guidance as to how to improve its performance. It can be tedious but it will be a good use of time.

Model selection and training/cross validation/test sets

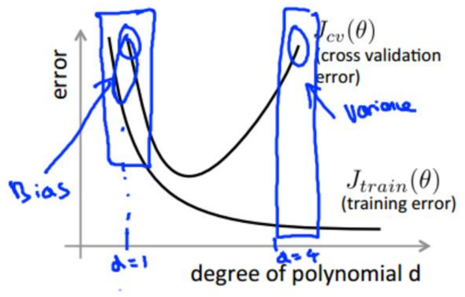
We can divide our available data into training set, cross validation set, and test set:

1. 60% for training set – To train all the models (with different degrees of polynomials). After hypothesis optimization has been done for all these models an optimized theta parameter set will be computed.
2. 20% for cross validation set – To select the model (from all models) with the least cv error
3. 20% for test set – To most accurately test the chosen model how it performs with 'new' and future data to estimate the generalization error that this best model will have.

Avoid using the test data to select the model (select the least cost model calculated on the test data set) and then report the generalization error using the test data.

Bias and Variance characteristics

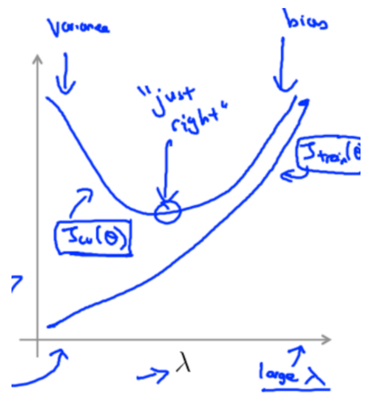
High bias	High variance
Underfitting	Overfitting
Low degree model	High degree model
J_{train} high	J_{train} low
$J_{cv} \sim J_{train}$	$J_{cv} >> J_{train}$
Large λ	Small λ



Regularization term λ

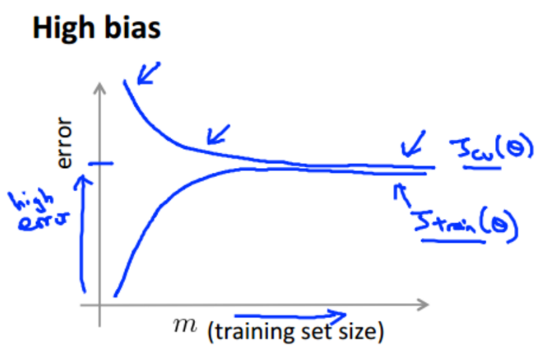
Increasing λ will cover up higher degree terms, causing the function to become simpler models.

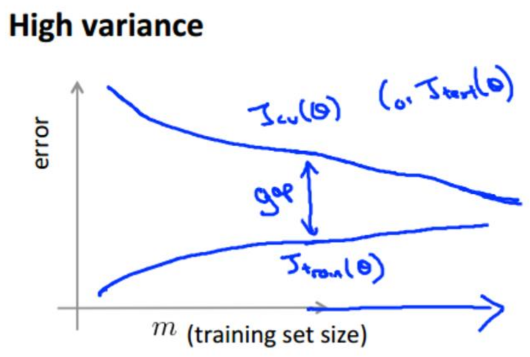
- Become higher bias / lower variance models



Learning curves

Plotting learning curves (J error against m, training data) will help us understand bias and variance of a learning algorithm, and thus we will be able to devise certain steps to improve the algorithm.





As shown, increasing m (the number of training data) will not help algorithms with high bias but will likely decrease the errors in high variance algorithms (more data to help fit higher degrees of polynomials better)

- It seems that having a high variance (more complicated model) and having more data is a better problem to have than having high bias
- Having more data is definitely good!

All in all, to fix:

High bias	High variance
Get more features	Decrease features
Increase polynomial degree (d)	
Decrease λ	Increase λ
	Increase data (m)

How about Neural Networks?

In NNs, small NNs are simpler and thus have less features (high bias, easier to underfit new data) while large NNs are more complex and thus have more hidden layers/features (high variance, easier to overfit new data)

Error metrices for skewed classes in logistic regression problems

It is important to have different error metrices for skewed classes (small % of test data have positive y) whereby we need to choose a different threshold to predict y = 1 instead of the simple 50% rule.

Precision and Recall

$$\text{Precision} = \frac{True\ positive}{True\ positive + False\ positive} \qquad \text{Recall} = \frac{True\ positive}{True\ positive + False\ negative}$$

- Predict y=1 only if very confident → Higher P, Lower R
- To avoid too many false negatives in prediction → Lower P, Higher R

We use P and R to finetune our threshold (for h(x)) to predict y=1

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

Whereby the score ranges from 0 to 1 (best).

- Based on different models for the logistic regression, we find P and R, and calculate the F1 score for every model → Choose the model with the best F1 score