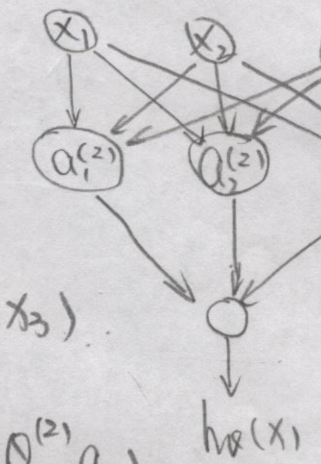
Neural Networks.

- If network has S_j units in layer j, S_{j+1} units in layer $j+1$, then $\theta^{(j)}$ will be of dimension $\boxed{S_{j+1} \times (S_j + 1)}$ where $\theta^{(j)}$ is the matrix of weights controlling function mapping from layer j to layer $j+1$.

$$\begin{cases} a_1^{(2)} = g(\theta_{10}^{(1)} \cdot x_0 + \theta_{11}^{(1)} \cdot x_1 + \theta_{12}^{(1)} \cdot x_2 + \theta_{13}^{(1)} \cdot x_3) \\ a_2^{(2)} = g(\theta_{20}^{(1)} \cdot x_0 + \theta_{21}^{(1)} \cdot x_1 + \theta_{22}^{(1)} \cdot x_2 + \theta_{23}^{(1)} \cdot x_3) \\ a_3^{(2)} = g(\theta_{30}^{(1)} \cdot x_0 + \theta_{31}^{(1)} \cdot x_1 + \theta_{32}^{(1)} \cdot x_2 + \theta_{33}^{(1)} \cdot x_3). \end{cases}$$



$$h_\theta(x) = g(\theta_{10}^{(2)} \cdot a_0^{(2)} + \theta_{11}^{(2)} \cdot a_1 + \theta_{12}^{(2)} \cdot a_2 + \theta_{13}^{(2)} \cdot a_3).$$

- $$x^{(1)} = \begin{bmatrix} x_0^{(1)} \\ x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix}, \quad \theta^{(1)} = \begin{bmatrix} \theta_{10}^{(1)} & \theta_{11}^{(1)} & \theta_{12}^{(1)} & \theta_{13}^{(1)} \\ \theta_{20}^{(1)} & \theta_{21}^{(1)} & \theta_{22}^{(1)} & \theta_{23}^{(1)} \\ \theta_{30}^{(1)} & \theta_{31}^{(1)} & \theta_{32}^{(1)} & \theta_{33}^{(1)} \end{bmatrix}$$

$$a^{(2)} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = g(\underline{\theta^{(1)} \cdot x^{(1)}}) \Rightarrow a = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}, \quad \theta^{(2)} = [\theta_{10}^{(2)} \ \theta_{11}^{(2)} \ \theta_{12}^{(2)} \ \theta_{13}^{(2)}]$$

$$\rightarrow a_m^{(j+1)} = g(\theta_{m0}^{(j)} \cdot a_0^{(j)} + \theta_{m1}^{(j)} \cdot a_1^{(j)} + \cdots).$$

$$h_\theta(x) = g(\theta^{(2)} \cdot a)$$

- Applications:

i). And. e.g: $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 \cdot x_2)$, where $\theta_0 = -1.5$, $\theta_1 = \theta_2 = 1$.

ii) Or. e.g: $h_\theta(x) = g(\theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2)$, where $\theta_0 = -0.5$, $\theta_1 = \theta_2 = 1$.

iii). Not. e.g: $h_\theta(x) = g(\theta_0 + \theta_1 \cdot x)$; where $\theta_0 = 0.5$, $\theta_1 = -1$.

iv). XOR: $(\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2)$

Multiple Output Units:

eg $y^{(i)} \in \left\{ \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right\}$.

Cost Function of Neural Network:

$h_\theta(x) \in \mathbb{R}^K$, $(h_\theta(x))_i = i^{th}$ output , K output units.

m: the # of training examples. , L: total # of layers in network.

S_l: # of units (excluding the bias unit) in layer l.

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

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

K is the # of outputs units.

penalize all θ.

$\min_\theta J(\Theta)$

Need code to compute:

i). $J(\Theta)$

ii). $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$.

Backpropagation algorithm:

1. Create a feed-forward network
2. Initialize all network weights to small random number
3. until the termination condition is met, do

   for $d \in D$ do

   i) compute output o of every unit.

   ii). for each output unit k, its error term is:

   $$\delta_k = o_k(1-o_k) \cdot (t_k - o_k)$$ % t_k is from y.

   iii) for each hidden unit h,

   $$\delta_h = o_h(1-o_h) \cdot \sum_{k \in outputs} w_{k,h} \cdot \delta_k$$

   iv). update all w : $w_{ji} = w_{ji} + \eta \cdot \delta_j \cdot x_{ji}$

reshape function in matlab/octave.

Suppose : $\Theta_1 \in R^{10\times 11}$, $\Theta_2 \in R^{10\times 11}$, $\Theta_3 \in R^{1\times 11}$

- thetaVec = [Θ_1(:) ; Θ_2(:) ; Θ_3(:)]

To get $\Theta_1, \Theta_2, \Theta_3$ back from thetaVec, we can Apply "reshape" function.

e.g: Θ_1 = reshape(thetaVec$\underbrace{(1:110)}_{\text{range}}$, $\underset{\text{row}}{10}$, $\underset{\text{column}}{11}$)

Gradient Checking:

$\Theta \in R^n$, the unrolled version of $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}, \cdots$

$\Theta = [\Theta_1, \Theta_2, \cdots, \Theta_n]$.

then we have:

- $$\frac{\partial}{\partial\Theta_1}J(\Theta) \approx \frac{J(\Theta_1+\varepsilon, \Theta_2, \cdots, \Theta_n) - J(\Theta_1-\varepsilon, \Theta_2, \cdots, \Theta_n)}{2\varepsilon}$$

$$\frac{\partial}{\partial\Theta_2}J(\Theta) \approx \frac{J(\Theta_1, \Theta_2+\varepsilon, \cdots, \Theta_n) - J(\Theta_1, \Theta_2-\varepsilon, \cdots, \Theta_n)}{2\varepsilon}$$

$$\vdots$$

$$\frac{\partial}{\partial\Theta_n}J(\Theta_n) \approx \frac{J(\Theta_1, \Theta_2, \cdots, \Theta_n+\varepsilon) - J(\Theta_1, \Theta_2, \cdots, \Theta_n-\varepsilon)}{2\varepsilon}.$$

Apply this properties to check if the gradient get from backpropagation is correct.

e.g: i). implement backpropagation to compute DVec (unrolled $D^{(1)}, D^{(2)}, D^{(3)}$).

- ii). Implement numerical gradient check to compute grapApprox.

iii). Make sure they give similar values.

iv). Turn off gradient checking, and using backpropagation for learning.

Random initialization to break symmetry.

Training a neural network.

i). # of input units: dimension of features $x^{(i)}$

ii) # of output units: number of classes.

iii). Reasonable default: one or more hidden layer, and have same # of hidden units in every layer (usually the more the better).

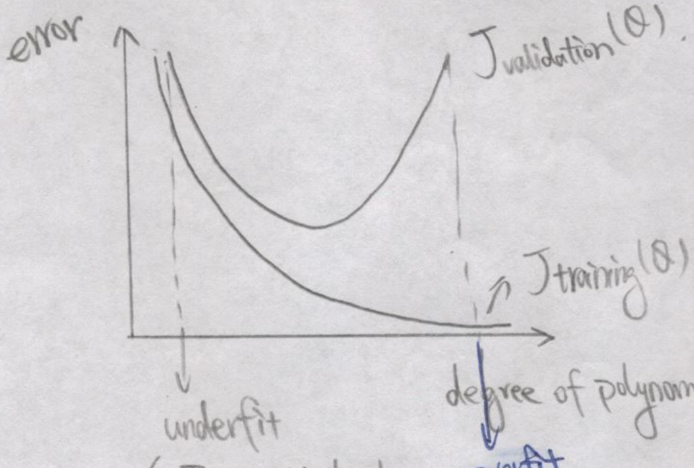
Advices for Applying Machine Learning.

- What you should try next if your hypothesis makes large errors on a new dataset?
  - i). Get more training examples.
  - ii). Try smaller sets of features

    } fix overfitting (high variance)
  - iii). Try getting additional features.
  - iv). Try adding polynomial features.

    } fix high bias (underfitting).
  - V). Try decreasing λ or increasing λ → fix ~~underfitting~~ overfitting.

Evaluating your hypothesis: 60% Dataset for training.
20% Dataset for validation.
20% Dataset for testing.



- Bias / Variance:

  High bias → underfit

  High variance → overfit

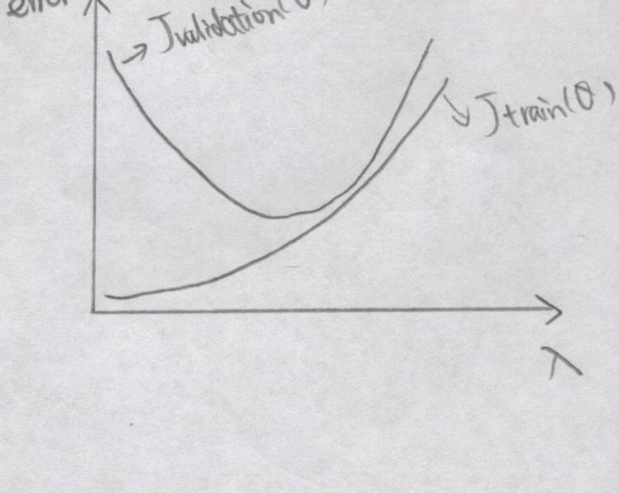
underfit
{ $J_{train}(\theta)$ high
$J_{validation}(\theta)$ high.

overfit
{ $J_{train}(\theta)$ low
$J_{validation}(\theta)$ high

Linear Regression with Regularization:

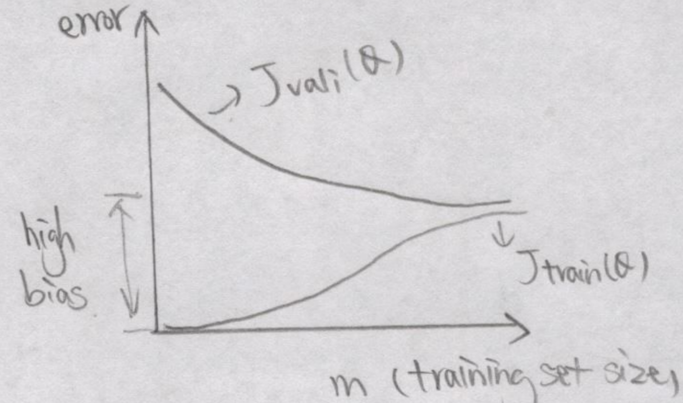
i) Large λ → high bias, underfit.

ii) Too small λ, → high variance, overfit



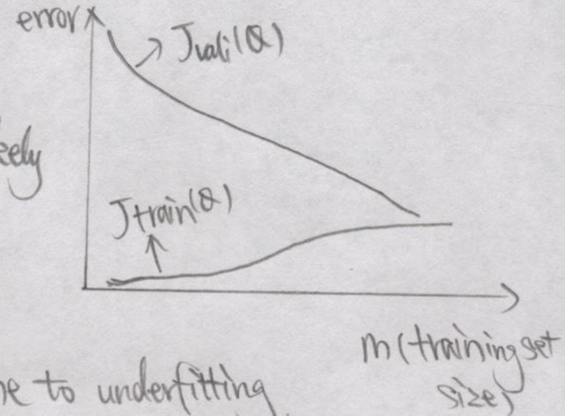
Learning Curve.

i). high bias.



If a learning algorithm is suffering from high bias, getting more training data will not help much.

ii). High variance



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

small neural network: fewer parameters, more prone to underfitting

Large neural network: more parameters, more prone to overfitting,

Recommended Approach:

1. Implement an algorithm and test it on your cross-validation data.
2. Plot learning curves to decide if underfit or overfit.
3. Error Analysis: manually check the examples your algorithm made error on to see what type of errors.

Error matrices for skewed classes.

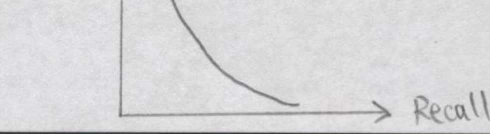
Actual class

		1	0
predicted class	1	True pos	False Pos
	0	False neg	True neg

i). Precision: $\frac{\text{True pos}}{\text{\#pred Pos}} = \frac{\text{True Pos}}{\text{True pos + False Pos}}$

ii). Recall: $\frac{\text{True pos}}{\text{Actuall Pos}} = \frac{\text{True Pos}}{\text{True Pos + False neg}}$

Trade off: predict 1 if $h_\theta(x) \geq$ threshold.



How to compare precision/recall numbers?

F_1 score : $2 \cdot \dfrac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$, maximize F_1 score

Data for machine learning:

i). many parameters. → low bias

ii). large training set. → to avoid overfitting.

Combine i) + ii), always get low $J_{test}(\theta)$