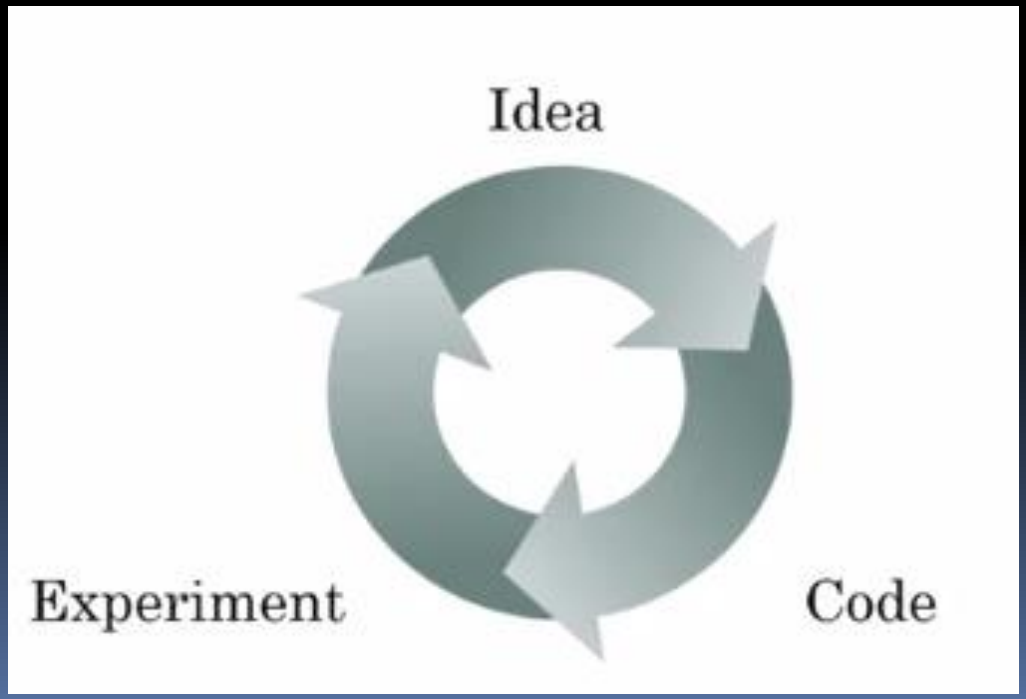
Network Considerations

CSCI 447/547 MACHINE LEARNING

Outline

- Setting up the Network
- Regularization
- Optimization



Applied Machine Learning

- Neural network
  - Number of layers
  - Number of hidden units
    - Number of input units is just the dimension of features
    - Number of output units is just the number of classes
  - Learning rates
  - Activation functions
- Highly iterative process to determine all of this
- Many different domain areas
  - Intuition from one application may not transfer to another domain

Neural Network Considerations

- Hidden layers…
  - Default is to use one hidden layer
  - If you use more than one, default is to use same number of units in each layer
  - Usually the more hidden units the better, but more computationally complex (performance suffers)
- To get number of hidden layers
  - Test with different numbers of layers and see which performs best

Training the network

1. Randomly initialize weights (to small values near 0)
2. Implement forward propagation
3. Implement code to compute cost function
4. Implement back propagation to compute partial derivatives
5. Use gradient checking to compare partial derivatives back propagation vs. using numerical estimates
6. Use gradient descent or optimization methods to minimize the cost function (which is non-convex with a neural network, so it can get stuck in local minima)

Debugging a Learning Algorithm

- Get more training data
- Try smaller set of features
- Try getting additional features
- Try adding polynomial features
- Try increasing or decreasing λ (regularization parameter)

- Most often, people use random approach (not the best choice)

Applied ML

- Train / Validation / Test Data Sets
- Traditional wisdom said split data 60/20/20
- With big data, may not need that much validation data
  - Maybe out of 1,000,000 use 10k for validation, 10k for test
    - 98/1/1

Applied ML

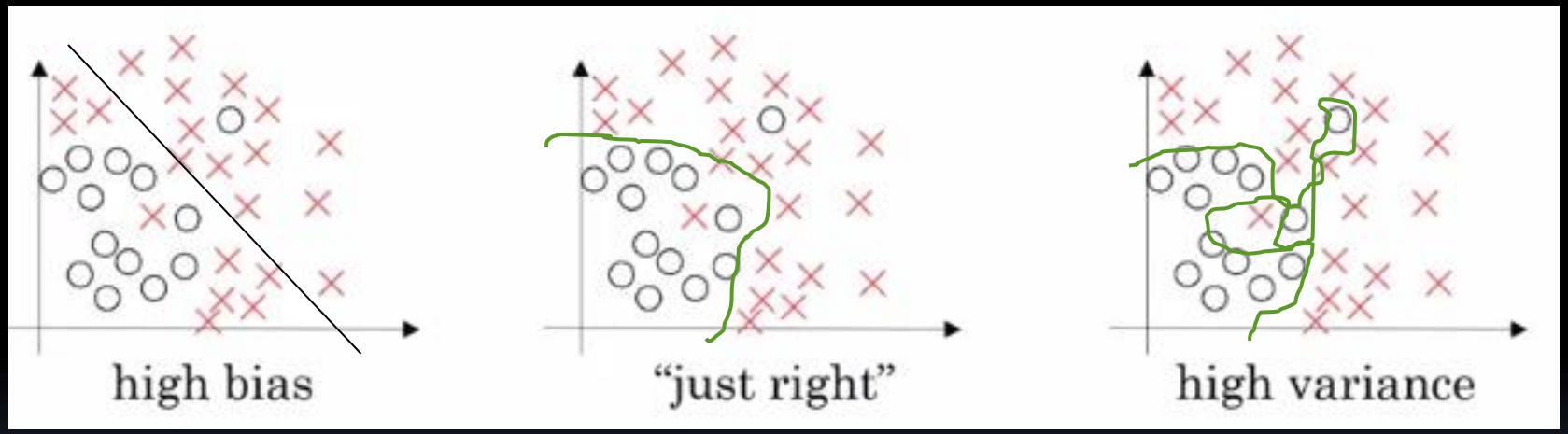
- Mismatched train/test distribution
  - Population sample must be the same for all data sets
  - Make sure development (train and validation) data comes from same distribution as test data
- Might be ok to not have a test set – ONLY if you don't need the unbiased estimate

Bias / Variance

- Bias – an error from erroneous assumptions in the learning algorithm
  - Missing relevant relations between features and outputs
  - Underfitting
- Variance – an error from sensitivity to small fluctuations in the training set
  - Model noise rather than intended relationships
  - Overfitting

Bias / Variance

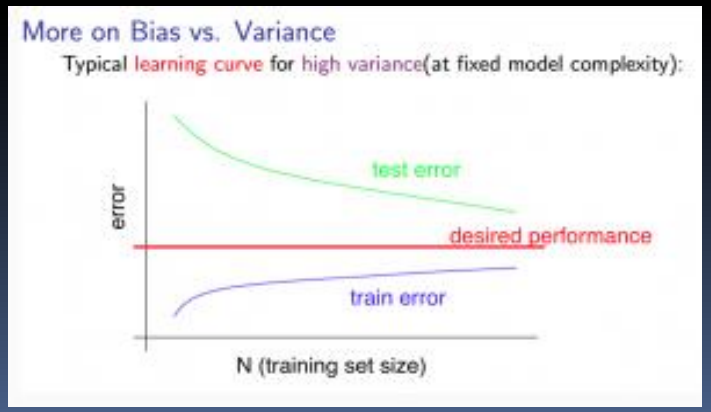
- Less discussion of bias / variance trade off in big data era

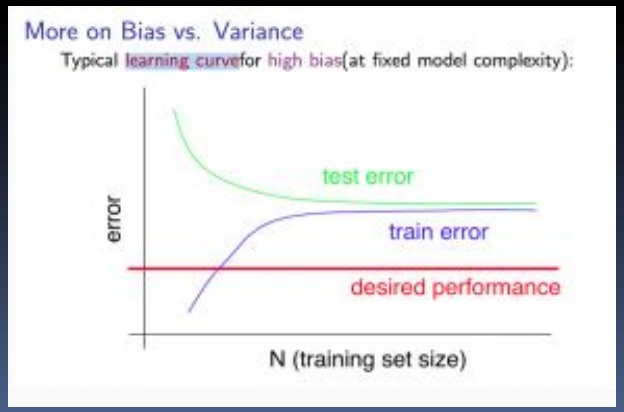


- Can't visualize decision boundary in high dimensional data
- Use training and validation set error instead

Bias / Variance

		Training Set Error	
		High	Low
Validation Set Error	High	High Bias; If Validation Error is even higher, High Variance also	High Variance
	Low	Never happens…	Low Bias, Low Variance





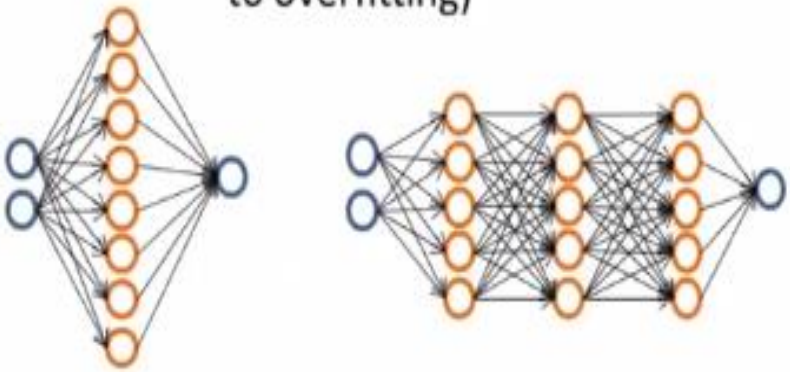
Overfitting

Neural networks and overfitting

"Small" neural network (fewer parameters; more prone to underfitting)

Computationally cheaper

"Large" neural network (more parameters; more prone to overfitting)



Computationally more expensive.

Use regularization (λ) to address overfitting.

Choosing Regularization Parameter

- First, try algorithm with no regularization
- Then train with different values for λ
  - $\lambda = 0$
  - $\lambda = 0.01$
  - $\lambda = 0.02$
  - $\lambda = 0.04$
  - …
  - $\lambda = 10$
- Test on validation set
- Choose best and test on test set

Learning Curves

- Plotting these is a good diagnostic
- Plot error on training set and on validation set as a function of the number of training examples
  - Start with few examples
  - Increase examples and plot
  - Error will be small with few examples, but error should grow as number of examples increases on training set
  - Opposite will hold on validation set

What to do next?

- Get more training examples
  - Good if you have high variance
- Try smaller set of features
  - Good if you have high variance
- Try getting additional features
  - Good if you have high bias
- Try adding polynomial features
  - Good if you have high bias
- Try decreasing lambda
  - Good for high bias
- Try increasing lambda
  - Good for high variance

Optimal Error

- Often called Bayes error
- Predicating on idea that human performance is good
  - May be the case that it is not, then model error anything less than optimal error is good

Basic Recipe for ML

- High bias? (training data performance); Try
  - Bigger network – more layers, more units
  - Train longer
  - Better architecture
  - Making assumption that task is solvable
  - Once reduced to acceptable level…
- High variance?
  - More data
  - Regularization
  - Different NN architecture
- Iterate

Bias/Variance Tradeoff

- Really pre-deep learning era
- But with big data, additional training, more data, usually helps both

Regularization

- Reduces variance
- Using logistic regression as an example
  - Add a term to punish overfitting
  - L2 regularization
  - L1 regularization
    - W will end up being sparse – a lot of 0's in it
    - But not used as much
  - Lambda is called regularization parameter
    - Hyperparameter

Regularization in Neural Network

$$J(W^{[1]}, b^{[1]}, \ldots, W^{[l]}, b^{[l]}) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^{L} \|w^{[l]}\|^2$$

$$\|w^{[l]}\|^2 = \sum_{i=1}^{l-1} \sum_{j=1}^{l} (W_{ij}^{[l]})^2$$

- "Frobenius norm"
- L2 Regularization also called "weight decay" in neural networks

$$\frac{\partial J}{\partial W^{[l]}} = dW^{[l]}$$

$$dW^{[l]} = (\text{from packprop}) + \frac{\lambda}{m} W^{[l]}$$

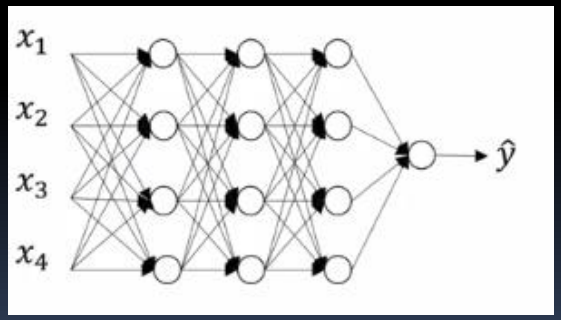
$$W^{[l]} \leftarrow W^{[l]} - \alpha dW^{[l]}$$

Why Regularization Prevents Overfitting

- Reduces impact of some of the hidden units
  - By reducing connection weights
- If weights are small, then output will be small
  - Assume we're using tanh for activation function
  - Small output range keeps us in the (almost) linear region of the function
- Implementation tip:
  - Plot cost function as a function of number of iterations – make sure you include regularization term

Dropout Regularization

- Set some probability for eliminating some of the nodes in the network
  - Remove incoming and outgoing links as well
- Change at each iteration



Inverted Dropout

- Set a "dropout vector"
  - Set a "keep" probability
  - Generate a matrix of random numbers
  - If a particular element is greater than "keep" probability, set the element to 0
  - Divide remaining values by keep probability so that you have the same overall value in the network
    - Keeps the scaling values the same

Making Predictions at Test Time

- Don’t use dropout at test time
  - Don’t want randomness at test time

Understanding Dropout

- Intuition: Can't rely on any one feature so have to spread out weights
  - Any one input of a node could go away at random, so not too much weight should be given to any one input
    - Shrinks weights
    - Possible to show that dropout has a similar effect to L2 regularization
- Have another hyperparameter – keep probability
  - This can vary for different layers

Understanding Dropout

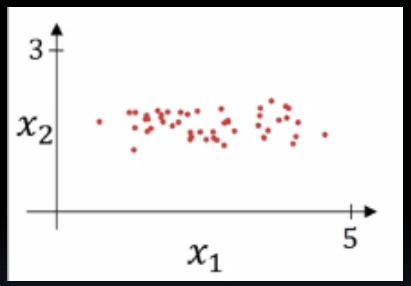
- First used in computer vision
  - Input is very large
  - But not all pixels really contribute…
  - But, really used as a regularization technique
- Disadvantage
  - Harder to check gradient descent
    - Cost function is less well defined
    - Technique is to check without dropout first to make sure you are getting decreasing gradient descent and then add dropout

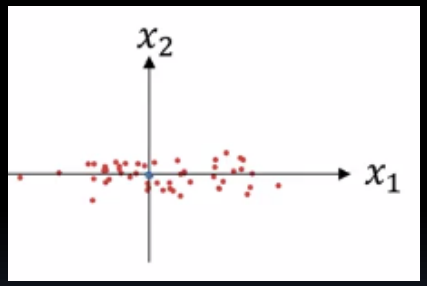
Other Regularization Methods

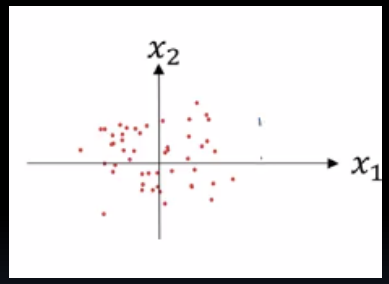
- Data augmentation
  - Instead of getting more data
  - We saw this with convolutional networks
  - Imposes random rotation and distortions to input data
- Early Stopping
  - Plot error vs. iterations
    - Both training and validation sets
  - Validation error will likely increase at some point
    - Pick that point (iterations) to stop training
  - Downside: Optimization is all about minimizing cost, but early stopping couples regularization and minimizing cost

Setting up Optimization

- Normalizing Training Sets
  - Speeds up training
  - 1. Subtract out mean (now have 0-mean)
  - 2. Normalize by the variance



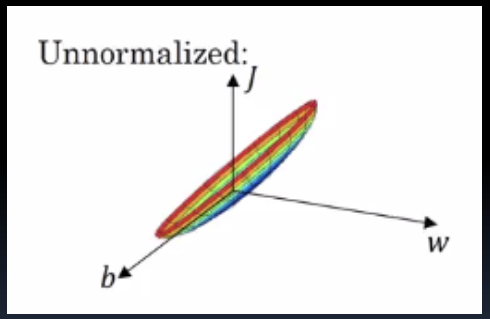


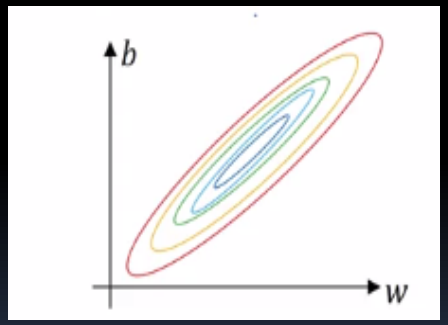


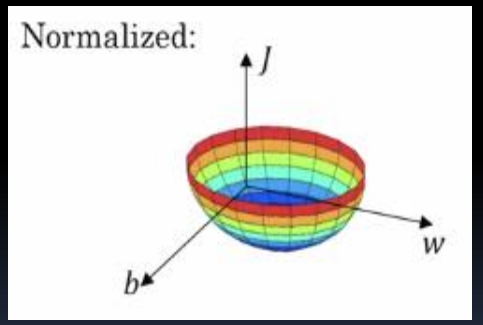
  - Important to use the same mean and variance to normalize validation and test sets as well, rather than calculating them separately

Setting up Optimization Problem

- Why normalize?
  - Get elongated error surface
    - Gradient descent will take much longer

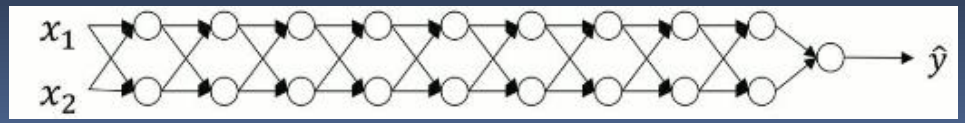






Setting up Optimization Problem

- Vanishing and Exploding Gradients
  - Derivatives / slopes can get very small or very big
  - Say you're training a very deep network
    - As you feed forward through the network, assuming a linear (or ReLU) activation function, the output can grow exponentially
    - As you back propagate the error and multiply small numbers together, weights will get smaller and smaller toward input layer, decreasing exponentially



Setting up Optimization Problem

- Vanishing and Exploding Gradients
  - Partial solution is to take care when initializing weights
    - The larger your number of features, the smaller you want your weights to be
    - One solution is to set weights proportional to 1/n
      - Get random Gaussian random numbers * sqrt(2/(n-1))
      - Works best on ReLU
        - If tanh, use sqrt(1/(n-1))
    - Reduces problem, though doesn't solve it

Setting up Optimization Problem

- Gradient Checking
  - Take all weights and biases and reshape them into a big vector, θ
  - Do the same thing with the derivatives into vector dθ
  - Question is, is dθ the same (or close enough) as the approximate derivatives of your J(θ) cost function
    - Do this by checking Euclidean distance between the two (L2 norm)

$$\frac{|d\theta_{approx} - d\theta|_2}{|d\theta_{approx}|_2 + |d\theta|_2}$$

  - This is only done for debugging – not during general training
  - Look at components that are off and see what might be contributing to the error
  - If you are using a regularization term, this must be included
  - Doesn't work with dropout
  - Run this after some iterations of training, not at initialization

Summary

- Setting up the Network
- Regularization
- Optimization