CSE/STAT 416 - Review Notes

Instructor: Hunter Schafer *Date :* 05/24/2023

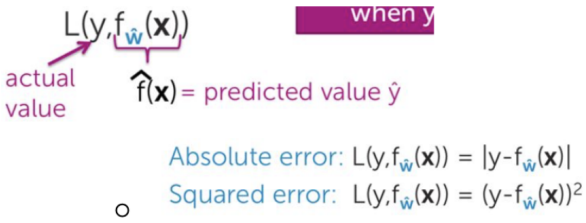
1 Linear Regression Model

1.1 Fitting a regression model

1. Find the best line : $RSS\left(w_0, w_1\right) = \sum_{i=1}^{N}\left(y_i - \left[w_0 + w_1 x_i\right]\right)^2$ (quality metric)
2. Gradient Descent
3. Regression model:
   $y_i = w_0 + w_1 x_i + \epsilon_i$ (w_0 : intercept; w_1 : change in y per unit change in x)
4. Polynomial regression
   (a) Parameter/Coefficient: w_j (scalar)
   (b) $x[j] = j^{\text{th}}$ input (scalar)
   (c) $h_j(x) = j^{\text{th}}$ feature (scalar)
   (d) $x_i =$ input of i^{th} data point (vector)
   (e) $x_i[j] = j^{\text{th}}$ input of i^{th} data point (scalar)
5. Model:
$$y_i = w_o h_0\left(x_i\right) + w_1 h_1\left(x_i\right) + \ldots + w_D h_D\left(x_i\right) + \epsilon_i = \sum_{j=0}^{D} w_j h_j\left(x_i\right) + \epsilon_i$$

1.2 Assessing the model

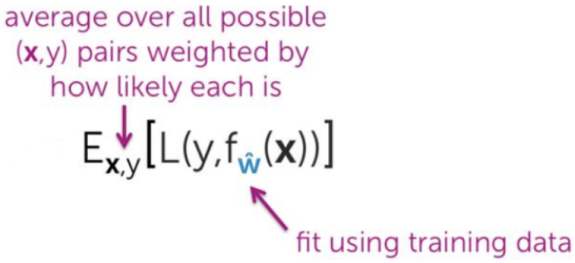
1. Loss:



2. Training error:

$$\frac{1}{N}\sum_{i=1}^{N} \mathsf{L}(\mathsf{y}_i, \mathsf{f}_{\hat{\mathbf{w}}}(\mathbf{x}_i)) \qquad \mathbf{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\mathsf{y}_i - \mathsf{f}_{\hat{\mathbf{w}}}(\mathbf{x}_i))^2}$$

   (a) Training error vs. model complexity: decrease
3. Generalization (true) error



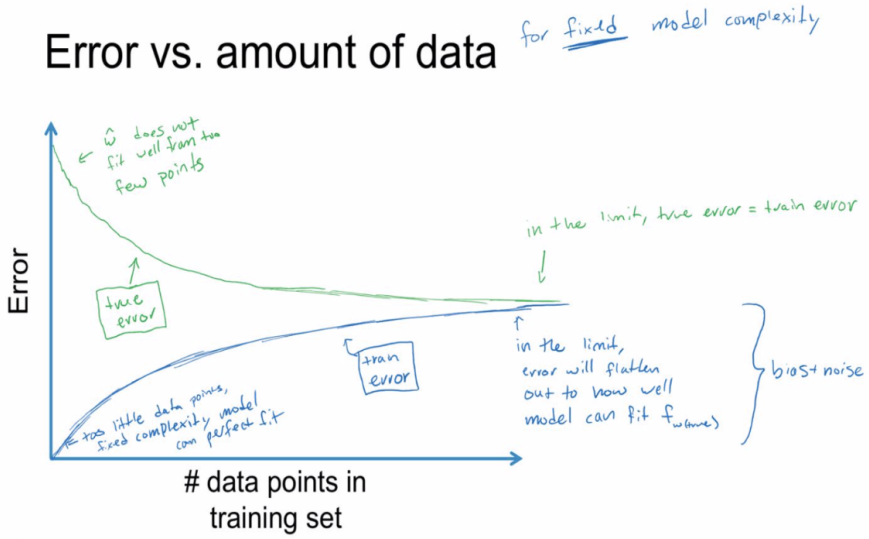
(a) Generalization error vs. model complexity: decrease and then increase

4. Test error

$$\frac{1}{N_{test}} \sum_{i \text{ in test set}} L(y_i, f_{\hat{w}}(x_i))$$

(fit using training data)

5. Sources of error
   (a) Noise
      i. Irreducible Error
      ii. ϵ_i
   (b) Bias
      i. Low complexity, high bias
      ii. High complexity, low bias
   (c) Variance
      i. Low complexity, low variance
      ii. High complexity, high variance



1.3 Regularization more specifically

1.3.1 Ridge regression

1. Overfitting of polynomial regression
   (a) Very large estimated parameters w
   (b) Observation influence

   i. Few observation (N small)
      A. Rapidly overfit as model complexity increase
   ii. Many observations (N very large)
      A. Harder to overfit
(c) Number of inputs influence
   i. 1 input
      A. Data must include representative examples of all possible pairs to avoid overfitting (HARD)
   ii. D input
      A. Data must include examples of all possible combos to avoid overfitting (MUCH HARDER!!)

2. TOTAL COST = measure of fit + measure of magnitude of coefficient
3. Large λ
   (a) If w $\neq 0$, If $\lambda = \infty$, Total cost = ∞. If w = 0, then total cost = RSS(0)
   (b) High bias, low variance
4. Small λ
   (a) Low bias, high variance
   (b) Standard least squares (RSS) fit of high-order polynomial for $\lambda = 0$.
5. Coefficient will converge to 0 as increasing λ but will never = 0

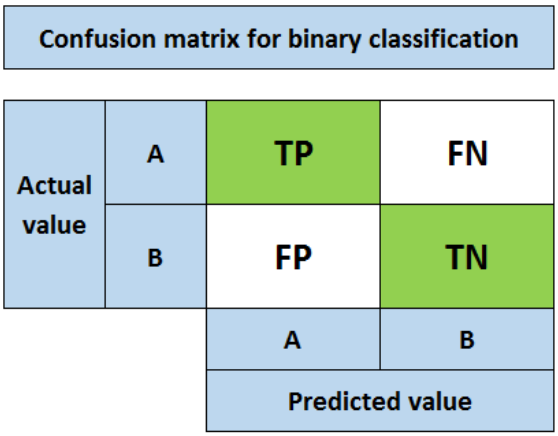
1.3.2 Lasso Regression

1. Option 1: All subsets or greedy variants
   (a) Feature selection
      i. Access RSS for each selection of feature in different size
   (b) Choosing model complexity
      i. Assess on validation set
      ii. Cross Validation
   (c) Greedy algorithms
      i. Forward stepwise:
         A. Starting from simple model and iteratively add features most useful to fit
      ii. Backward stepwise:
         A. Start with full model and iteratively remove features least useful to fit
      iii. Combining forward and backward steps:
         A. In forward algorithm, insert steps to remove features no longer as important
2. Option 2: Regularize
   (a) TOTAL COST = measure of fit + measure of magnitude of coefficient
   (b) Coefficient path: features will become 0 one by one as λ increase
   (c) Cross validation
      i. Choose various validation set in data and average performance over all choices K-fold cross validation
      ii. Randomly assign data to K groups
      iii. For k = 1 . . . k
         A. For specific λ, estimate total cost on the training blocks
         B. Compute error on validation block: error (λ).
      iv. Compute average error:
      v. Choose $\lambda*$ to minimize CV(λ)
      vi. Leave-one-out cross validation = N-fold (n = number of examples in training set)
      vii. 5-fold CV, 10-fold CV

2 Classification

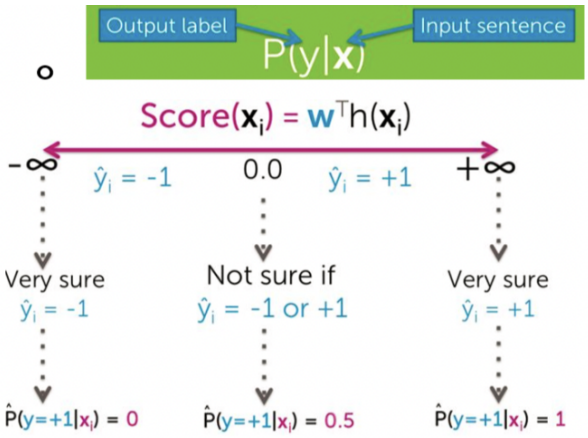
2.1 Evaluation metric - Accuracy

1. Error = No of mistakes / Total No of sentences
2. Accuracy = No of correct / Total No of sentences But a classifier with 90% accuracy good may not be a good thing
3. False positives, false negatives and confusion matrices
4. Even with infinite data, test error will not go to 0
5. More complex models tend to have less bias
   (a) Models with less bias tend to need more data to learn will, but do better with sufficient data



3 Logistic Regression

1. $$Score(x_i) = w_0 h_0(x) + w_1 h_1(x) + ... + w_D h_D(x_i)$$
2. How confident is your prediction?



4 Logistic function (sigmoid, logit)

1. $$P(y = +1|x_i, w) = sigmoid(Score(x_i)) = \frac{1}{1 + e^{-w^T h(x)}}$$

   Quality metric for logistic regression: Maximum likelihood estimation
2. Likelihood $l(w)$: measures quality of fit for model with coefficients w .
3. Find 'best' classifier = maximize quality metric over all possible w .
4. Overfitting for logistic regression
   (a) Large coefficient values
   (b) Sigmoid goes to 1 or 0
   (c) Model becomes extremely overconfident of predictions
5. TOTAL QUALITY = measure of fit – measure of magnitude of coefficients
6. Could use ridge or lasso
   $$l(w) - \lambda||w||_2^2$$
   (a) Pick λ using validation set (for large data set)
   (b) Cross-validation (for small data set)

5 Decision Tree

1. Error = No of incorrect predictions / No of examples
2. Measure effectiveness of split
   (a) Error = No of mistakes / No of data points
   (b) Compute, choose to split on the feature with lowest error
   (c) Recursive stump learning
3. Stop condition
   (a) All data agrees on y
   (b) Already split on all features
   (c) Don't stop if error doesn't decrease
4. Find the best threshold split

6 Overfitting

1. Early stopping: stop the learning algorithm before tree becomes too complex
   (a) Limit tree depth (max_depth =)
   (b) Do not consider splits that do not cause a sufficient decrease in classification error
   (c) Do not split an intermediate node which contains too few data points
   (d) Pros
      i. A reasonable heuristic for early stopping to avoid useless splits

   (e) Cons
      i. Too short sighted: we may miss out on 'good' splits may occur right after 'useless' splits
      ii. Saw this with 'xor' example

2. Pruning: simplify the tree after the learning algorithm terminates - Complements early stopping

3. Scoring trees (BALANCE!)
   (a) TOTAL COST = measure of fit + measure of complexity
   (b) Total $CostC(T) = Error(T) + L(T)$

4. Pruning
   (a) Consider a split
   (b) Compute total cost $C(T)$ of split
   (c) 'Undo' the split on $Tsmaller$
   (d) Prune if total cost is lower: $C(Tsmaller) \leq C(T)$
   (e) Repeat steps before for every split

7 Boosting

7.1 Ensemble Classifier

1. Ensemble methods: Each classifier 'votes' on prediction

2. $$F(x_i) = sign(w_1 f_1(x_i) + w_2 f_2(x_i) + w_3 f_3(x_i) + w_4 f_4(x_i))$$

3. Prediction:
$$\hat{y} = sign(\sum_{t=1}^{T} \hat{w}_t f_t(x)) \qquad T = \text{total number of classifiers}$$

4. Boosting = Focus learning on 'hard' points
   (a) Focus next classifier on places where $f(x)$ does less well
   (b) More weight on 'hard' or more important points
   (c) Prediction: (T = total no of classifier)

7.1.1 AdaBoost: Learning Ensemble

1. Normalizing weights
   (a) Can cause numerical instability after many iterations
   (b) Normalize weights to add up to 1 after every iteration

2. AdaBoost Theorem:
   (a) Training error will decrease and oscillating in the middle, become 0 eventually
   (b) But not always possible
   (c) Often yields great training error

3. True error will also go down – robust to overfitting
   (a) But will eventually overfit, so must choose max number of components
   (b) Choosing T using validation set / Cross-validation

7.1.2 Gradient Boosting

7.1.3 Random Forests

1. Bagging: pick random subsets of the data
2. Learn a tree in each subset
3. Average predictions
   (a) Simpler than boosting & easier to parallelize
   (b) Typically higher error than boosting for same no of trees (no iterations T)

8 Precision & Recall (evaluation metric)

1. Precision: fraction of positive predictions that are actually positive
   (a) Precision = No true positives / (No true positives + No false positives)
   (b) High precision means positive predictions actually likely to be positives
      i. Pessimistic model: predict positive only when very true
2. Recall: fraction of positive data predicted to be positive
   (a) Recall = No true positives / (No true positives + No false negatives)
   (b) High recall means positive data points are very likely to be discovered
      i. Optimistic model: predict almost everything as positive
   (c) Which classifier is better? Using precision-recall curve?
      i. Large area under curve
      ii. Depends on pessimistic/optimistic situation

9 Clustering and Similarity: Retrieving Documents

9.1 Document representation

1. Bag of words model
   (a) Count no of instances of each word in vocabulary
   (b) Issues with word counts – rare words
2. TF-IDF
   (a) Term frequency = word counts (common locally)
   (b) Inverse doc freq. = #docs1+#docs2 using words (‘the’ = 0)
   (c) tf * idf

9.2 Distance Metrics: defining notion of 'closest'

1. Euclidean distance in 1D: $distance(x_j, x_q) = |x_j - x_q|$
2. In multiple dimensions
   (a) Can define many interesting distance functions
   (b) Might want to weight different dimensions differently
      i. Some features are more relevant
      ii. Some features vary more than others

$$\text{distance}(\mathbf{x}_i, \mathbf{x}_q) = \sqrt{a_1(\mathbf{x}_i[1]-\mathbf{x}_q[1])^2 + \ldots + a_d(\mathbf{x}_i[d]-\mathbf{x}_q[d])^2}$$

3. Similarity = (matrix multiplication)
4. COSINE similarity – normalize
   (a) not a proper distance metric
   (b) efficient to compute for sparse vecs
   (c) but not always desired
      i. normalizing can make dissimilar objects appear more similar (with short tweet)
      ii. common compromise: just cap maximum word counts
5. combining distance metrics

- text of document
  (a) distance metric: cosine similarity
- No of reads of doc
  (a) Distance metric: Euclidean distance
- Add together with user-specified weights

9.3 Locality Sensitive Hashing

1. Simple 'binning' of data into 2 bins
   (a) Challenging to find good line
   (b) Poor quality solution
      i. Points close together get split into separate bins
   (c) Large computational cost
      i. Bins might contain many points, so still searching over large set for each NN query
2. Improving: reduction no points examined per query
   (a) more bins
   (b) improving search quality by searching neighboring bins
3. Recap
   (a) draw h random lines
   (b) compute 'score' for each point under each line and translate to binary index
   (c) use h-bit binary vector per data point as bin index

(d) create hash table

(e) for each query point x, search bin(x) then neighboring bins until time limit

4. Cost of binning points in d-dim

(a) per data point, need d multiplies to determine bin index per plane

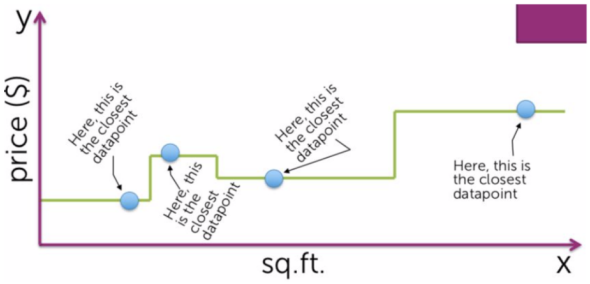
(b) one-time cost offset if many queries of fixed dataset

9.4 Nearest Neighbor Regression

1. 1-NN

(a) Sensitive to regions with little data

(b) Sensitive to noise in data



2. K-NN

(a) Weighted k-NN : Weigh more similar houses more than those less similar in list of k-NN

$$\hat{y}_q = \frac{c_{qNN1}y_{NN1} + c_{qNN2}y_{NN2} + c_{qNN3}y_{NN3} + \ldots + c_{qNNk}y_{NNk}}{\sum_{j=1}^{k} c_{qNNj}}$$

- Want weight C_{qNNj} to be small when distance(X_{nnj}, X_q) large
- Want weight C_{qNNj} to be large when distance(X_{nnj}, X_q) small

3. Kernel regression

(a) Instead of just weighting NN, weight all points

(b) Choice of bandwidth λ using cross validation

4. K-NN and kernel regression are examples of nonparametric regression

(a) Flexibility

(b) Make few assumptions about $f(x)$

(c) Complexity can grow with the number of observations N.

5. NN and kernel methods work well when the data cover the space, but

(a) The more dimensions d you have, the more points N you need to cover the space

(b) Need $N = O(exp(d))$ data points for good performance

(c) This is where parametric models become useful

9.5 Clustering: An unsupervised learning K-means: A clustering algorithm

9.5.1 K-Means: A Clustering Algorithm

1. Assume: Score = distance to cluster center (smaller better)
2. Initialize cluster centers
3. Assign observations to closest cluster center
4. Revise cluster centers as mean of assigned observations
5. Repeat last two steps until convergence
6. Converges to: LOCAL OPTIMUM

9.5.2 K-means++ (smart initialization)

1. choose first cluster center uniformly at random from data points
2. for each observation x, compute distance$d(x)$ to nearest cluster center
3. choose new cluster center from amongst data points, with probability of x being chosen proportional to $d(x)^2$
4. repeat the last to steps until k centers have been chosen
5. pros/cons
   (a) computationally costly relative to random initialization, but the subsequent k-means often converges more rapidly
   (b) tends to improve quality of local optimum and lower runtime

9.5.3 K-means objective

Trying to minimize the sum of squared distances

9.5.4 Cluster heterogeneity

1. measure of quality of given clustering
   (a)
   $$\sum_{j=1}^{k} \sum_{i:z_i=j} ||\mu_j - x_i||_2^2$$
   (b) lower is better (tighter clusters)
2. overfitting as k increases, if k = N, heterogeneity = 0.
3. How to choose k? - Roughly same heterogeneity as for much larger k

9.5.5 Limitation failure modes of k-means

1. Learn user preference
2. Uncertainty in cluster assignments
3. Assign observations to closest cluster center -Only center matters

9.5.6 Hierarchical clustering

1. Dendrograms helps visualize
2. Allow user to choose any distance metric - K-means restricted to Euclidean distance
3. Can find more complex shapes than k-means or Gaussian mixture models
4. Two main types of algorithms
   (a) Divisive: top-down: start with all data in one big cluster and recursively split
      i. Recursive k-means
   (b) Agglomerative: bottom-up: Start with each data point as its own cluster. Merge clusters until all points are in one big cluster
      i. Single linkage
         A. Initialize each point to be its own cluster
         B. Define distance between clusters
         C. Merge the two closest clusters
         D. Repeat step 3 until all point are in one cluster

10 Dimension reduction

1. Easier learning – fewer parameters
2. Visualization – hard to visualize more than 3D/4D
3. Discover 'intrinsic dimensionality' of data - High dimensional data that is truly lower dimensional
4. Linear projection - Reconstruction
5. **Principal component analysis (PCA)**
   (a) Choose projection with minimum reconstruction error
   (b) Eigenfaces

11 Recommender Systems

1. Popularity
   (a) No personalization
   (b) No capture context
2. Classification model
   (a) Personalized
   (b) Can capture context
   (c) Even handles limited user history
   (d) But features may not be available
   (e) Often doesn't perform as well as collaborative filtering methods
3. People who bought this also bought

   (a) Co-occurrence matrix
   (b) Normalize co-occurrences: similarity matrix
      i. Jaccard similarity: normalizes by popularity
      ii. No purchased i and j No purchased i or j
   (c) Limitation
      i. Only current page matters, no history
      ii. No personalization
      iii. No capture context
   (d) (weighted) Average of purchased items
      i. Limitation
         A. No context
         B. no user features
         C. no product features
         D. cold start problem

4. Discovering hidden structure by matrix factorization
   (a) Rating(u,v)
   (b) But still – cold start problem

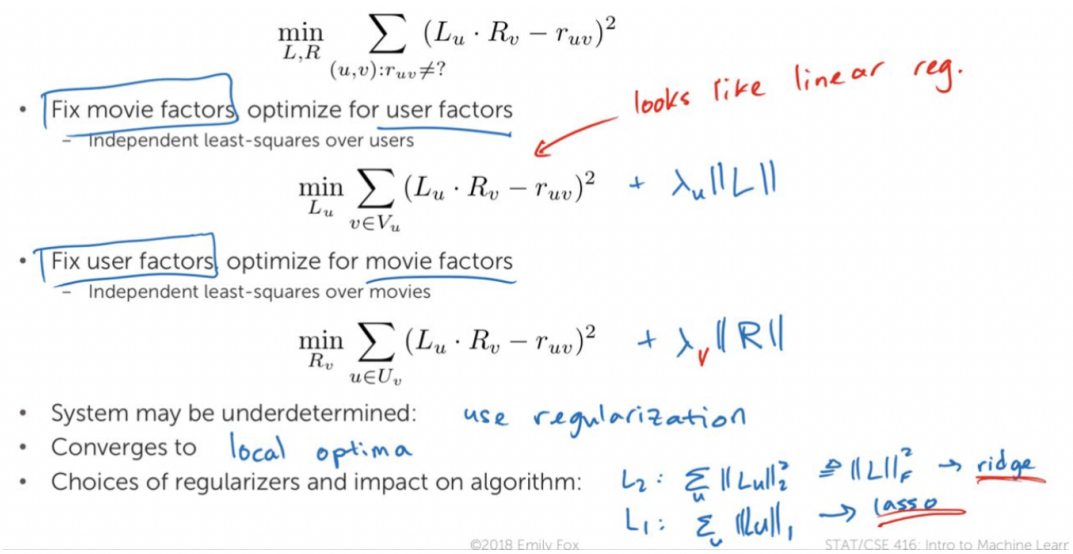
   Therefore,

5. Featurized matrix factorization
   (a) Features capture context
   (b) Discovered topics from matrix factorization capture groups of users who behave similarly
   (c) Combine to mitigate cold-start problem
      i. Ratings for a new user from features only
      ii. As more information about user is discovered, matrix factorization

11.1 Performance Metrics

1. Classification accuracy = fraction of items correctly classified
   (a) Not interested in what a person does not like
2. Precision and recall
3. Area under the curve
4. Set desired recall and maximize precision (precision at k)

12 Coordinate Descent

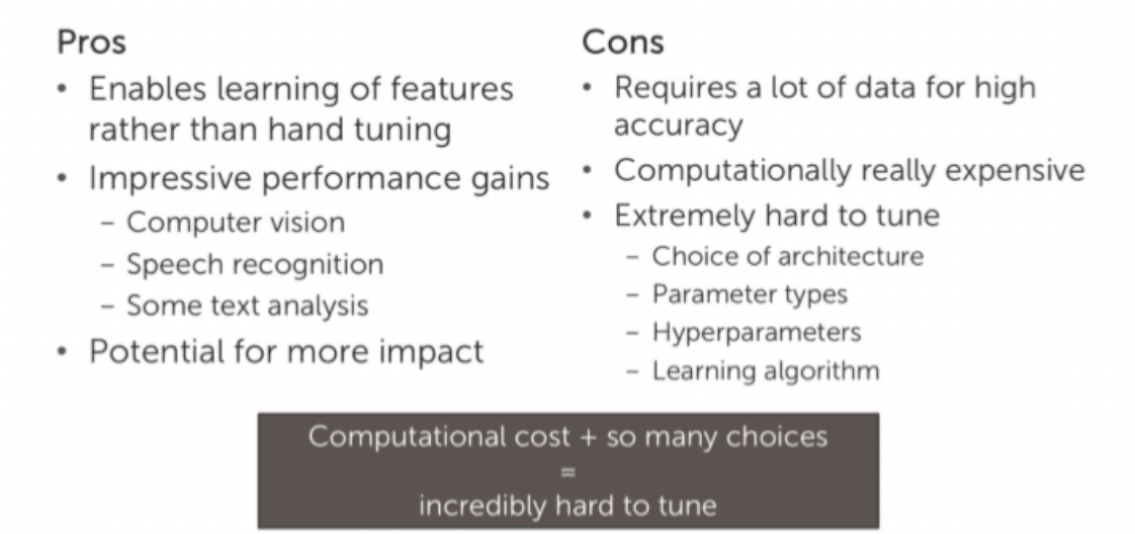


13 Deep learning: Neural networks (very non-linear features)

1. Input edges $x[i]$, along with intercept $x[0]$
2. Sum passed through an activation function g
3. Simple linear classifier can't represent XOR problem
4. Hidden layer
   (a) Going beyond linear classification by adding a layer
   (b) no longer convex function
5. Sigmoid neuron (like logistic regression)
   (a) just change g
6. overfitting
   (a) likely to overfit
   (b) avoid by
      i. more training data
      ii. fewer hidden nodes / better topology (For eg: 3-layer NNs outperform 2-layer NNs, but going deeper rarely helps)
      iii. Regularization
      iv. Early Stopping

13.1 Images

1. Convolution networks
   (a) Stride and zero-padding
   (b) Max pooling (e.g. size 2 by 2 with stride 2) -Tends to work better than average pooling



They aren't sensitive to settings of tuning parameters (i.e. you don't have to spend too much time trying many different tuning parameter values)

2. False: This comes from the complexity of neural networks. If you change the number of layers, the number of nodes in each layer, even the step size for gradient descent, you can get wildly different results.

Deep features: Deep learning + Transfer features

Transfer learning: use data from one task to help learn on another