

Neural Nets:

Many possible refs

e.g., Mitchell Chapter 4



Simple Model Selection Cross Validation Regularization Neural Networks

Machine Learning – 10701/15781

Carlos Guestrin

Carnegie Mellon University

February 13th, 2006

Announcements



- Recitations stay on Thursdays
 - 5-6:30pm in Wean 5409
 - This week: Cross Validation and Neural Nets
- **Homework 2**
 - Due next Monday, Feb. 20th
 - Updated version online with more hints
 - Start early

OK... now we'll learn to pick those darned parameters...

■ Selecting features (or basis functions)

- Linear regression
- Naïve Bayes
- Logistic regression

■ Selecting parameter value

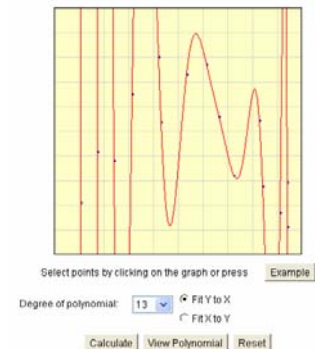
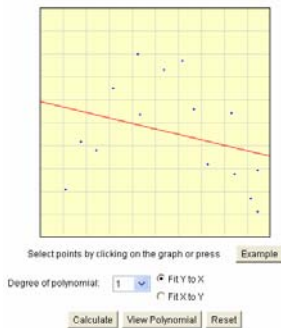
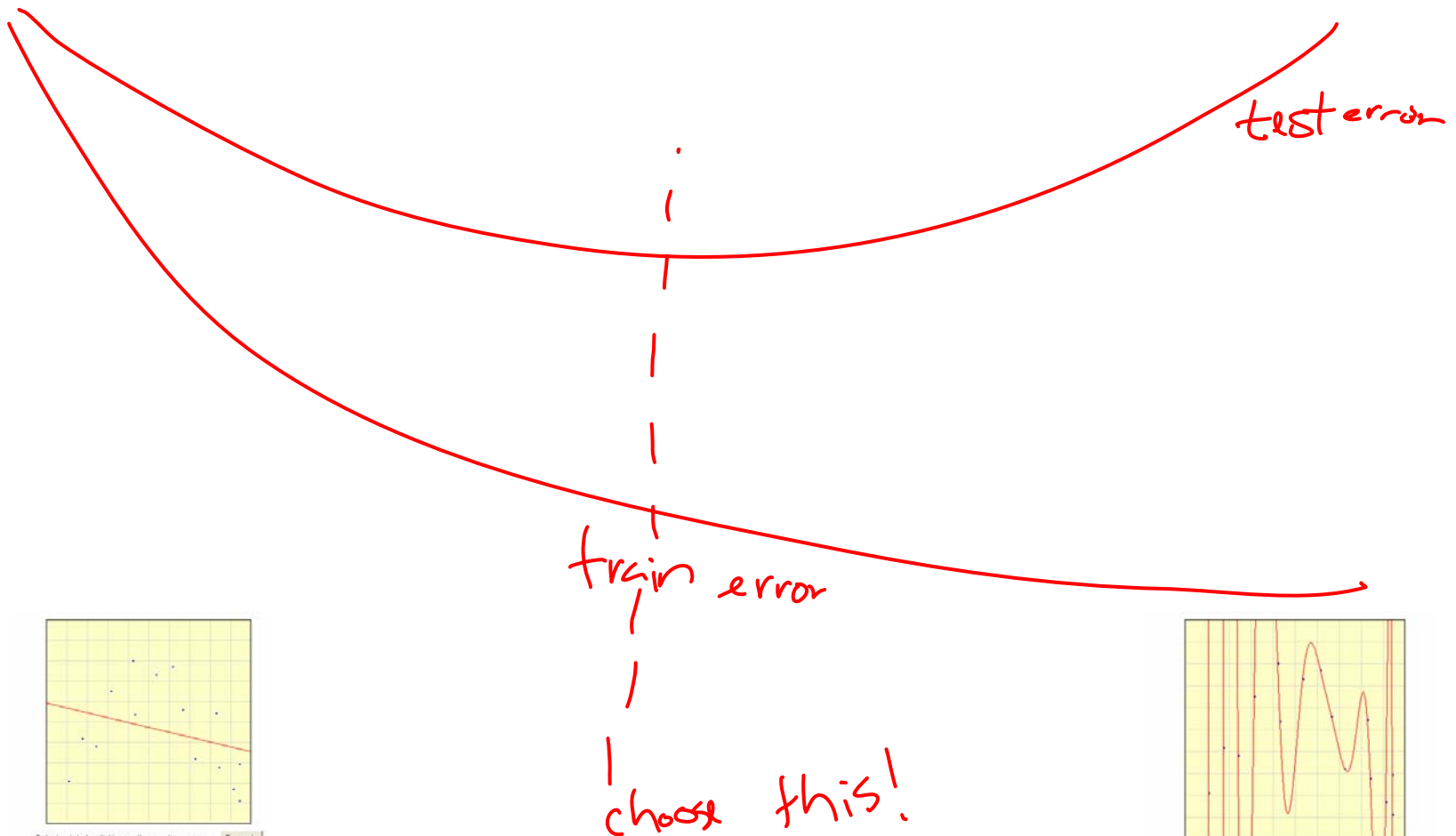
- Prior strength
 - Naïve Bayes, linear and logistic regression
- Regularization strength
 - Naïve Bayes, linear and logistic regression
- Decision trees
 - MaxpChance, depth, number of leaves
- Boosting
 - Number of rounds

■ More generally, these are called **Model Selection Problems**

■ Today:

- Describe basic idea
- Introduce very important concept for tuning learning approaches: **Cross-Validation**

Test set error as a function of model complexity



Simple greedy model selection algorithm

- Pick a dictionary of features (A hard part)
 - e.g., polynomials for linear regression $1, x, x^2, x^3, x^4, \dots$
- Greedy heuristic:
 - Start from empty (or simple) set of features $F_0 = \emptyset$ *e.g., $1, x$*
 - Run learning algorithm for current set of features F_t
 - Obtain h_t
 - Select **next best feature X_i**
 - e.g., X_i that results in lowest training error learner when learning with $F_t \cup \{X_i\}$
 - $F_{t+1} \leftarrow F_t \cup \{X_i\}$
 - Recurse

Greedy model selection

- Applicable in many settings:
 - Linear regression: Selecting basis functions
 - Naïve Bayes: Selecting (independent) features $P(X_i|Y)$
 - Logistic regression: Selecting features (basis functions)
 - Decision trees: Selecting leaves to expand
- Only a heuristic!
 - But, sometimes you can prove something cool about it
 - e.g., [Krause & Guestrin '05]: Near-optimal in some settings that include Naïve Bayes
- There are many more elaborate methods out there

Simple greedy model selection algorithm

- Greedy heuristic:

- ...

- Select **next best feature** X_i

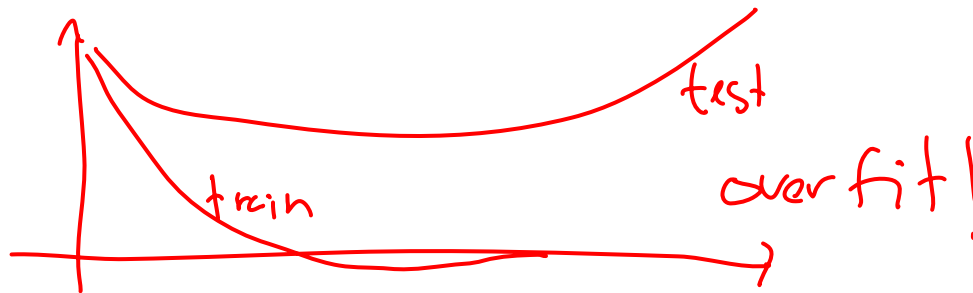
- e.g., X_j that results in lowest training error learner when learning with $F_t \cup \{X_j\}$

- $F_{t+1} \leftarrow F_t \cup \{X_i\}$

- Recurse

When do you stop???

- When training error is low enough?



Simple greedy model selection algorithm

- Greedy heuristic:

- ...

- Select **next best feature** X_i

- e.g., X_j that results in lowest training error learner when learning with $F_t \cup \{X_j\}$

- $F_{t+1} \leftarrow F_t \cup \{X_i\}$

- Recurse

When do you stop???

- ~~When training error is low enough?~~

- When test set error is low enough?

never ever ever learn on test data!!!

Validation set

- Thus far: Given a dataset, **randomly** split it into two parts:
 - Training data – $\{\mathbf{x}_1, \dots, \mathbf{x}_{N_{\text{train}}}\}$
 - Test data – $\{\mathbf{x}_1, \dots, \mathbf{x}_{N_{\text{test}}}\}$
- But **Test data must always remain independent!**
 - Never ever ever ever learn on test data, including for model selection
- Given a dataset, **randomly** split it into three parts:
 - Training data – $\{\mathbf{x}_1, \dots, \mathbf{x}_{N_{\text{train}}}\}$
 - Validation data – $\{\mathbf{x}_1, \dots, \mathbf{x}_{N_{\text{valid}}}\}$
 - Test data – $\{\mathbf{x}_1, \dots, \mathbf{x}_{N_{\text{test}}}\}$
- Use validation data for tuning learning algorithm, e.g., model selection
 - Save test data for very final evaluation



Simple greedy model selection algorithm

■ Greedy heuristic:

□ ...

□ Select **next best feature** X_i

- e.g., X_j that results in lowest training error learner when learning with $F_t \cup \{X_j\}$

□ $F_{t+1} \leftarrow F_t \cup \{X_i\}$

□ Recurse

When do you stop???

- ~~When training error is low enough?~~
- ~~When test set error is low enough?~~
- When validation set error is low enough?

overfit to validation set.

Simple greedy model selection algorithm

■ Greedy heuristic:

- ...

- Select **next best feature** X_i

- e.g., X_j that results in lowest training error learner when learning with $F_t \cup \{X_j\}$

- $F_{t+1} \leftarrow F_t \cup \{X_i\}$

- Recurse

When do you stop???

- ~~When training error is low enough?~~

- ~~When test set error is low enough?~~

- ~~When validation set error is low enough?~~

- Man!!! OK, should I just repeat until I get tired???

- I am tired now...

- **No, “There is a better way!”**

(LOO) Leave-one-out cross validation

- Consider a **validation set with 1 example**:
 - D – training data
 - $D \setminus i$ – training data with i th data point moved to validation set
- **Learn classifier $h_{D \setminus i}$ with $D \setminus i$ dataset**
- **Estimate true error as**:
 - 0 if $h_{D \setminus i}$ classifies i th data point correctly
 - 1 if $h_{D \setminus i}$ is wrong about i th data point
 - Seems really bad estimator, but wait!
- **LOO cross validation**: Average over all data points i :
 - **For each data point you leave out, learn a new classifier $h_{D \setminus i}$**
 - **Estimate error as**:

$$error_{LOO} = \frac{1}{m} \sum_{i=1}^m \mathbb{1} \left(h_{D \setminus i}(\mathbf{x}^i) \neq y^i \right)$$

LOO cross validation is (almost) unbiased estimate of true error!

- When computing **LOOCV error**, we only use $m-1$ data points

- So it's not estimate of true error of learning with m data points!
- Usually pessimistic, though – learning with less data typically gives worse answer

- **LOO is almost unbiased!**

- Let $error_{true,m-1}$ be true error of learner when you only get $m-1$ data points
- In homework, you'll prove that LOO is unbiased estimate of $error_{true,m-1}$:

$$E_{\mathcal{D}}[error_{LOO}] = error_{true,m-1}$$

- **Great news!**

- **Use LOO error for model selection!!!**

Simple greedy model selection algorithm

■ Greedy heuristic:

□ ...

□ Select **next best feature** X_i

- e.g., X_j that results in lowest training error learner when learning with $F_t \cup \{X_j\}$

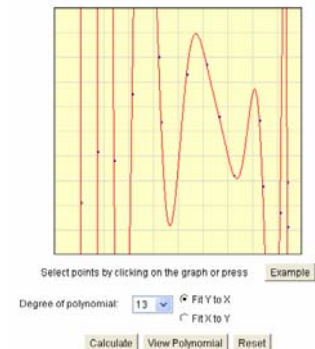
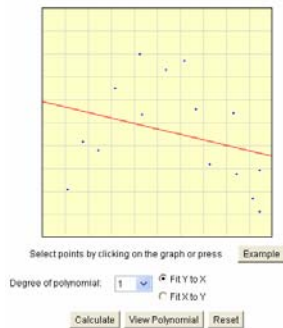
□ $F_{t+1} \leftarrow F_t \cup \{X_i\}$

□ Recurse

When do you stop???

- ~~When training error is low enough?~~
- ~~When test set error is low enough?~~
- ~~When validation set error is low enough?~~
- **STOP WHEN error_{LoO} IS LOW!!!**

Using LOO error for model selection



Computational cost of LOO

- Suppose you have 100,000 data points
- You implemented a great version of your learning algorithm
 - Learns in only 1 second
- Computing LOO will take about 1 day!!!
 - If you have to do for each choice of basis functions, it will take fooooooreeeve'!!!
- Solution 1: Preferred, but not usually possible
 - Find a cool trick to compute LOO (e.g., see homework)

Solution 2 to complexity of computing LOO:

(More typical) **Use k -fold cross validation**

- Randomly **divide** training data into k equal parts

- D_1, \dots, D_k

- For each i

- Learn classifier $h_{D \setminus D_i}$ using data point not in D_i

- Estimate error of $h_{D \setminus D_i}$ on validation set D_i :

$$error_{\mathcal{D}_i} = \frac{k}{m} \sum_{(\mathbf{x}^j, y^j) \in \mathcal{D}_i} \mathbb{1}(h_{D \setminus D_i}(\mathbf{x}^j) \neq y^j)$$

- **k -fold cross validation error is average** over data splits:

$$error_{k\text{-fold}} = \frac{1}{k} \sum_{i=1}^k error_{\mathcal{D}_i}$$

- k -fold cross validation properties:

- **Much faster to compute** than LOO

- **More (pessimistically) biased** – using much less data, only $m(k-1)/k$

- Usually, **$k = 10$** 😊

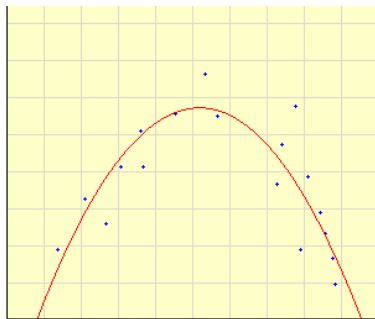
Regularization – Revisited

- Model selection 1: **Greedy**
 - Pick subset of features that have yield low LOO error
- Model selection 2: **Regularization**
 - Include **all possible features!**
 - **Penalize “complicated” hypothesis**

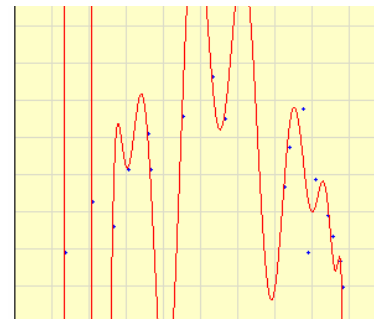
Regularization in linear regression

- Overfitting usually leads to very large parameter choices, e.g.:

$$-2.2 + 3.1 X - 0.30 X^2$$



$$-1.1 + 4,700,910.7 X - 8,585,638.4 X^2 + \dots$$



- Regularized least-squares (a.k.a. ridge regression), for $\lambda \geq 0$:

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \lambda \sum_{i=1}^k w_i^2$$

Other regularization examples

■ Logistic regression regularization

- Maximize data likelihood minus **penalty for large parameters**

$$\arg \max_{\mathbf{w}} \sum_j \ln P(y^j | \mathbf{x}^j, \mathbf{w}) - \lambda \sum_i w_i^2$$

- **Biases towards small parameter values**

■ Naïve Bayes regularization

- **Prior** over likelihood of features
- **Biases away from zero probability** outcomes

■ Decision tree regularization

- Many possibilities, e.g., **Chi-Square test** and **MaxPvalue** parameter
- **Biases towards smaller trees**

How do we pick magic parameter?

Cross Validation!!!!



λ in Linear/Logistic Regression

(analogously for # virtual examples in Naïve Bayes,
MaxPvalue in Decision Trees)

Regularization and Bayesian learning

$$p(\mathbf{w} \mid Y, \mathbf{X}) \propto P(Y \mid \mathbf{X}, \mathbf{w})p(\mathbf{w})$$

- We already saw that **regularization for logistic regression** corresponds to **MAP for zero mean, Gaussian prior** for \mathbf{w}
- Similar interpretation for other learning approaches:
 - **Linear regression**: Also zero mean, Gaussian prior for \mathbf{w}
 - **Naïve Bayes**: Directly defined as prior over parameters
 - **Decision trees**: Trickier to define... but we'll get back to this

Occam's Razor



- William of Ockham (1285-1349) *Principle of Parsimony*:
 - “One should not increase, beyond what is necessary, the number of entities required to explain anything.”
- Regularization penalizes for “*complex explanations*”
- Alternatively (but pretty much the same), use *Minimum Description Length (MDL) Principle*:
 - minimize $length(\text{misclassifications}) + length(\text{hypothesis})$
- $length(\text{misclassifications})$ – e.g., #wrong training examples
- $length(\text{hypothesis})$ – e.g., size of decision tree

Minimum Description Length Principle

- MDL prefers small hypothesis that fit data well:

$$h_{MDL} = \arg \min_h L_{C_1}(\mathcal{D} | h) + L_{C_2}(h)$$

- $L_{C_1}(D|h)$ – description length of data under code C_1 given h
 - Only need to describe points that h doesn't explain (classify correctly)
- $L_{C_2}(h)$ – description length of hypothesis h

- Decision tree example

- $L_{C_1}(D|h)$ – #bits required to describe data given h
 - If all points correctly classified, $L_{C_1}(D|h) = 0$
- $L_{C_2}(h)$ – #bits necessary to encode tree
- Trade off quality of classification with tree size

Bayesian interpretation of MDL Principle

■ MAP estimate

$$\begin{aligned} h_{MAP} &= \operatorname{argmax}_h [P(\mathcal{D} | h)P(h)] \\ &= \operatorname{argmax}_h [\log_2 P(\mathcal{D} | h) + \log_2 P(h)] \\ &= \operatorname{argmin}_h [-\log_2 P(\mathcal{D} | h) - \log_2 P(h)] \end{aligned}$$

■ **Information theory fact:**

- Smallest code for event of probability p requires $-\log_2 p$ bits

■ MDL interpretation of MAP:

- $-\log_2 P(D|h)$ – length of D under hypothesis h
- $-\log_2 P(h)$ – length of hypothesis h (there is hidden parameter here)
- MAP prefers simpler hypothesis:
 - minimize $length(\text{misclassifications}) + length(\text{hypothesis})$

■ **In general, Bayesian approach usually looks for simpler hypothesis** – Acts as a regularizer

What you need to know about Model Selection, Regularization and Cross Validation

- Cross validation
 - (Mostly) Unbiased estimate of true error
 - LOOCV is great, but hard to compute
 - k -fold much more practical
 - Use for selecting parameter values!
- Model selection
 - Search for a model with low cross validation error
- Regularization
 - Penalizes for complex models
 - Select parameter with cross validation
 - Really a Bayesian approach
- Minimum description length
 - Information theoretic interpretation of regularization
 - Relationship to MAP

Logistic regression

- $P(Y|X)$ represented by:

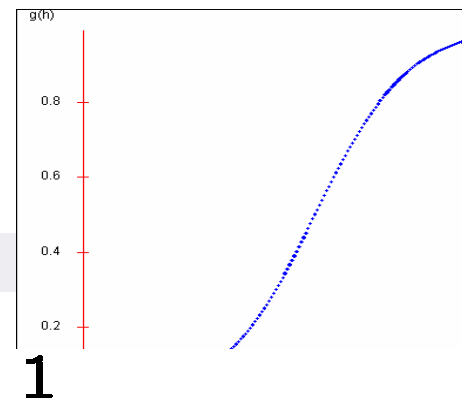
$$\begin{aligned} P(Y = 1 \mid x, W) &= \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}} \\ &= g(w_0 + \sum_i w_i x_i) \end{aligned}$$

- Learning rule – MLE:

$$\begin{aligned} \frac{\partial \ell(W)}{\partial w_i} &= \sum_j x_i^j [y^j - P(Y^j = 1 \mid x^j, W)] \\ &= \sum_j x_i^j [y^j - g(w_0 + \sum_i w_i x_i^j)] \end{aligned}$$

$$w_i \leftarrow w_i + \eta \sum_j x_i^j \delta^j$$

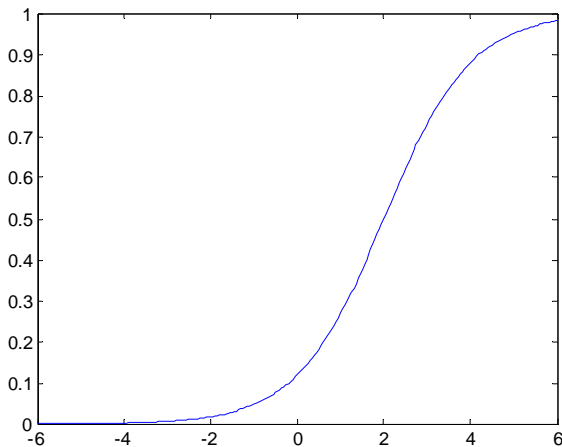
$$\delta^j = y^j - g(w_0 + \sum_i w_i x_i^j)$$



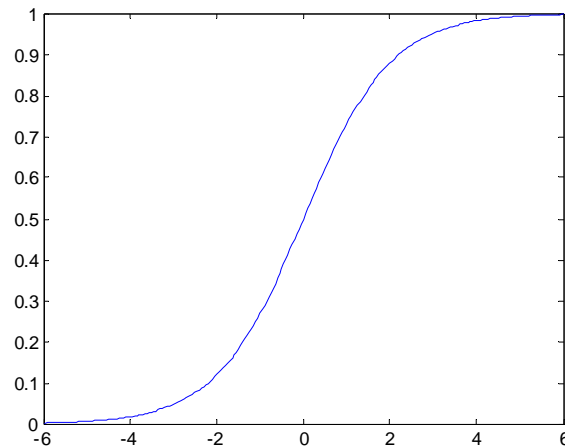
Sigmoid

$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}}$$

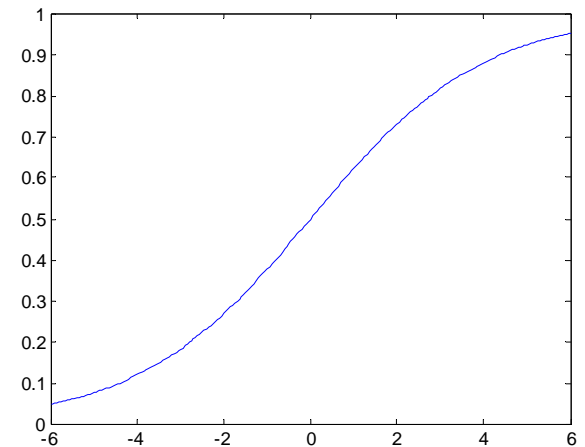
$w_0=2, w_1=1$



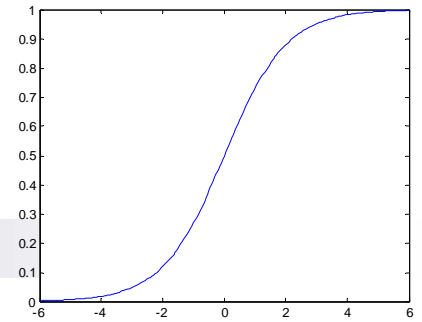
$w_0=0, w_1=1$



$w_0=0, w_1=0.5$

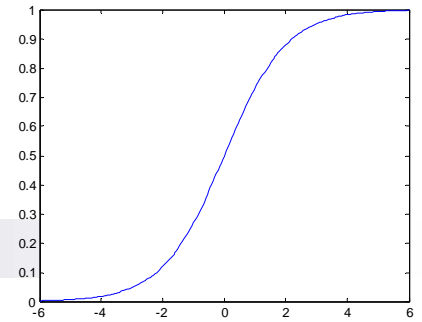


Perceptron as a graph



$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}}$$

Linear perceptron classification region



$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}}$$

Optimizing the perceptron

- Trained to minimize sum-squared error

$$\ell(W) = \frac{1}{2} \sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)]^2$$

Derivative of sigmoid

$$\frac{\partial \ell(W)}{\partial w_i} = - \sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)] x_i^j g'(w_0 + \sum_i w_i x_i^j)$$

$$g(x) = \frac{1}{1 + e^{-x}}$$

The perceptron learning rule

$$w_i \leftarrow w_i + \eta \sum_j x_i^j \delta^j$$

$$\delta^j = [y^j - g(w_0 + \sum_i w_i x_i^j)] g^j (1 - g^j)$$

$$g^j = g(w_0 + \sum_i w_i x_i^j)$$

■ Compare to MLE:

$$w_i \leftarrow w_i + \eta \sum_j x_i^j \delta^j$$

$$\delta^j = [y^j - g(w_0 + \sum_i w_i x_i^j)]$$

Perceptron, linear classification, Boolean functions

- Can learn $x_1 \vee x_2$
- Can learn $x_1 \wedge x_2$
- Can learn any conjunction or disjunction

Perceptron, linear classification, Boolean functions



- Can learn majority
- Can perceptrons do everything?

Going beyond linear classification



- Solving the XOR problem

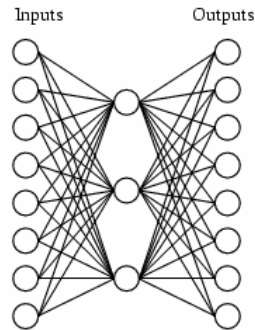
Hidden layer

- Perceptron: $out(\mathbf{x}) = g(w_0 + \sum_i w_i x_i)$

- 1-hidden layer:

$$out(\mathbf{x}) = g \left(w_0 + \sum_k w_k g(w_0^k + \sum_i w_i^k x_i) \right)$$

Example data for NN with hidden layer



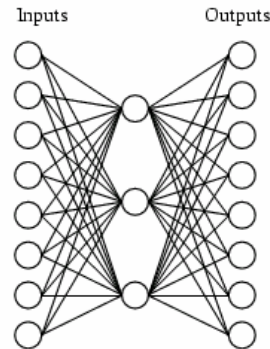
A target function:

Input	Output
10000000 →	10000000
01000000 →	01000000
00100000 →	00100000
00010000 →	00010000
00001000 →	00001000
00000100 →	00000100
00000010 →	00000010
00000001 →	00000001

Can this be learned??

Learned weights for hidden layer

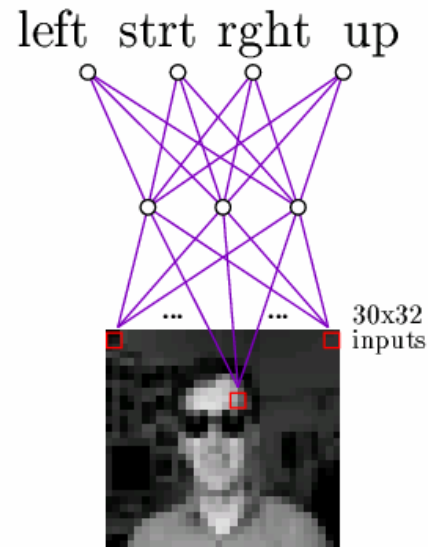
A network:



Learned hidden layer representation:

Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

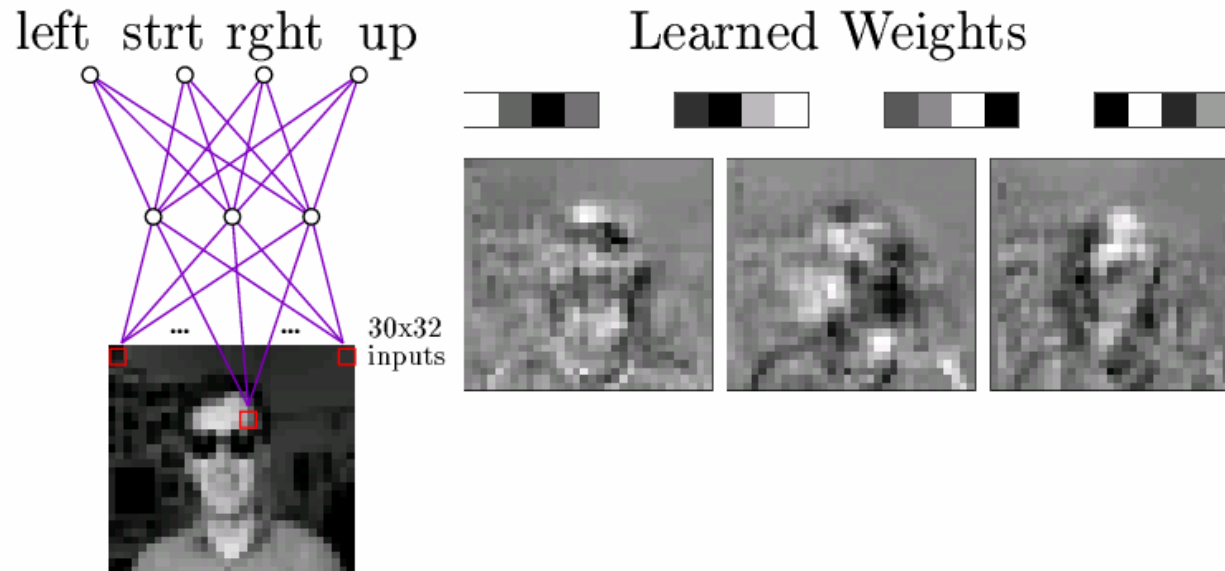
NN for images



Typical input images

90% accurate learning head pose, and recognizing 1-of-20 faces

Weights in NN for images



Typical input images

Forward propagation for 1-hidden layer - Prediction

- 1-hidden layer:

$$out(\mathbf{x}) = g \left(w_0 + \sum_k w_k g(w_0^k + \sum_i w_i^k x_i) \right)$$

Gradient descent for 1-hidden layer – Back-propagation: Computing $\frac{\partial \ell(W)}{\partial w_k}$

Dropped w_0 to make derivation simpler

$$\ell(W) = \frac{1}{2} \sum_j [y^j - out(\mathbf{x}^j)]^2$$

$$out(\mathbf{x}) = g \left(\sum_k w_k g \left(\sum_i w_i^k x_i \right) \right)$$

$$\frac{\partial \ell(W)}{\partial w_k} = -[y - out(\mathbf{x})] \frac{\partial out(\mathbf{x})}{\partial w_k}$$

Gradient descent for 1-hidden layer – Back-propagation: Computing $\frac{\partial \ell(W)}{\partial w_i^k}$

Dropped w_0 to make derivation simpler

$$\ell(W) = \frac{1}{2} \sum_j [y^j - out(\mathbf{x}^j)]^2$$

$$out(\mathbf{x}) = g \left(\sum_k w_k g \left(\sum_i w_i^k x_i \right) \right)$$

$$\frac{\partial \ell(W)}{\partial w_i^k} = -[y - out(\mathbf{x})] \frac{\partial out(\mathbf{x})}{\partial w_i^k}$$

Multilayer neural networks



Forward propagation – prediction

- Recursive algorithm
- Start from input layer
- Output of node V_k with parents U_1, U_2, \dots :

$$V_k = g \left(\sum_i w_i^k U_i \right)$$

Back-propagation – learning

- Just gradient descent!!!
- Recursive algorithm for computing gradient
- For each example
 - Perform forward propagation
 - Start from output layer
 - Compute gradient of node V_k with parents U_1, U_2, \dots
 - Update weight w_i^k

Many possible response functions



- Sigmoid
- Linear
- Exponential
- Gaussian
- ...

Convergence of backprop

- Perceptron leads to convex optimization
 - Gradient descent reaches **global minima**
- Multilayer neural nets **not convex**
 - Gradient descent gets stuck in local minima
 - Hard to set learning rate
 - Selecting number of hidden units and layers = fuzzy process
 - NNs falling in disfavor in last few years
 - We'll see later in semester, *kernel trick* is a good alternative
 - Nonetheless, neural nets are one of the most used ML approaches

Training set error



- Neural nets represent complex functions
 - Output becomes more complex with gradient steps
- Training set error

What about test set error?



Overfitting

- Output fits training data “too well”
 - Poor test set accuracy
- Overfitting the training data
 - Related to bias-variance tradeoff
 - One of central problems of ML
- Avoiding overfitting?
 - More training data
 - Regularization
 - Early stopping

What you need to know



- Perceptron:
 - ☐ Representation
 - ☐ Perceptron learning rule
 - ☐ Derivation
- Multilayer neural nets
 - ☐ Representation
 - ☐ Derivation of backprop
 - ☐ Learning rule
- Overfitting
 - ☐ Definition
 - ☐ Training set versus test set
 - ☐ Learning curve