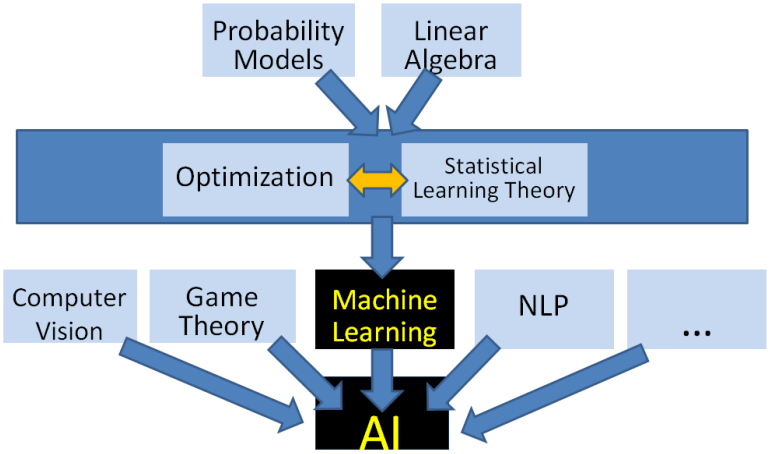
MSDS 422: Practical Machine Learning

Perceptron and Artificial Neural Networks



NORTHWESTERN UNIVERSITY

Dr. Nathan Bastian

References

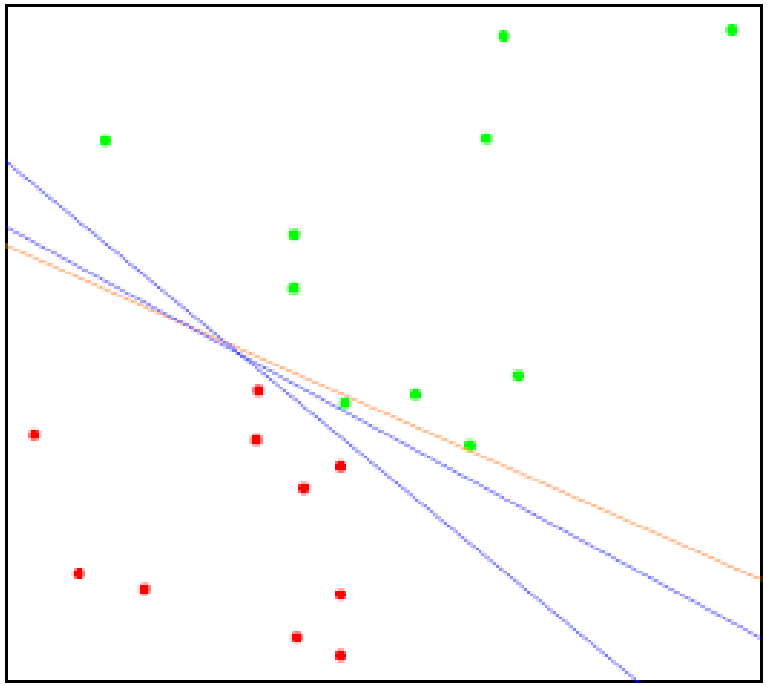
- *The Elements of Statistical Learning* (2009), by T. Hastie, R. Tibshirani, and J. Friedman.

- *Machine Learning: A Probabilistic Perspective* (2012), by K. Murphy

Separating Hyperplanes

- We seek to construct linear decision boundaries that explicitly try to separate the data into different classes as well as possible.
- LDA and logistic regression both estimate linear decision boundaries in similar but slightly different ways.
- Even when the training data can be perfectly separated by hyperplanes, LDA or other linear methods developed under a statistical framework may not achieve perfect separation.
- Instead of using a probabilistic argument to estimate parameters, here we consider a geometric argument.

Separating Hyperplanes (cont.)



- The blue lines are two of the infinitely many possible *separating hyperplanes*.
- The orange line is the OLS solution, obtained by regressing the -1/1 response Y on X:

$$\{x : \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = 0\}$$

- The OLS does not do a perfect job separating the points.

Separating Hyperplanes (cont.)

- *Perceptrons*: classifiers that compute a linear combination of the input features and return the sign (-1, +1).

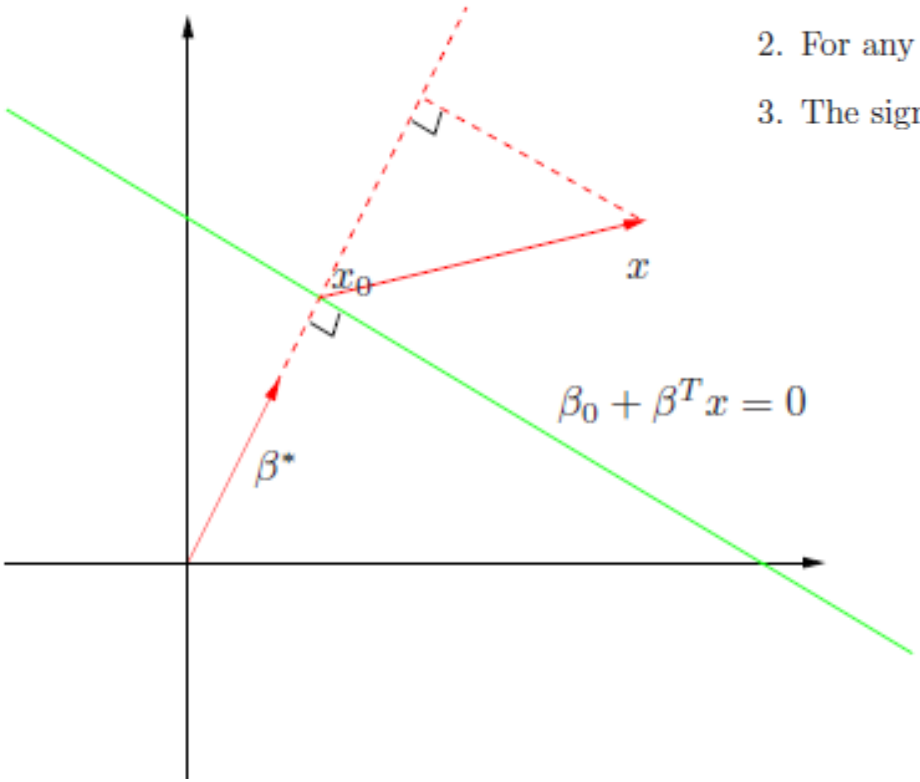
- Perceptrons set the foundations for neural network models, and they provide the basis for support vector classifiers.

- Before we continue, let's first review some vector algebra.

- A hyperplane or *affine set L* is defined by the equation:

$$L = \{x : f(x) = \beta_0 + \beta^T x = 0\}$$

Separating Hyperplanes (cont.)



Some properties:

1. For any two points x_1 and x_2 lying in L, $\beta^T(x_1 - x_2) = 0$, and hence $\beta^* = \beta/||\beta||$ is the vector normal to the surface of L.
2. For any point x_0 in L, $\beta^T x_0 = -\beta_0$.
3. The signed distance of any point x to L is given by

$$\begin{aligned} \beta^{*T}(x - x_0) &= \frac{1}{||\beta||}(\beta^T x + \beta_0) \\ &= \frac{1}{||f'(x)||} f(x). \end{aligned} \tag{4.40}$$

Hence $f(x)$ is proportional to the *signed distance* from x to the hyperplane defined by $f(x) = 0$.

Perceptron Learning Algorithm

- One of the older approaches to this problem of finding a separating hyperplane in the machine learning literature is called the *perceptron learning algorithm*, developed by Frank Rosenblatt in 1956.

- **Goal:** The algorithm tries to find a separating hyperplane by minimizing the distance of misclassified points to the decision boundary.

- The algorithm starts with an initial guess as to the separating plane's parameters and then updates that guess when it makes mistakes.

Perceptron Learning Algorithm (cont.)

- Code the two classes by $y_i = +1, -1$.
- If a response $y_i = +1$ is misclassified, then $\beta^T x_i + \beta_0 < 0$.
- If a response $y_i = -1$ is misclassified, then $\beta^T x_i + \beta_0 > 0$.
- Since the signed distance from x to the decision boundary is $\frac{\beta^T x_i + \beta_0}{\|\beta\|}$, then the distance from a misclassified x_i to the decision boundary is

$$\frac{-y_i(\beta^T x_i + \beta_0)}{\|\beta\|}$$

Perceptron Learning Algorithm (cont.)

- Denote the set of misclassified points by M.
- The goal is to minimize: $D(\beta, \beta_0) = -\sum_{i \in M} y_i(\beta^T x_i + \beta_0)$
- The quantity is non-negative and proportional to the distance of the misclassified points to the decision boundary.
- To minimize $D(\beta, \beta_0)$, compute the gradient (assuming M is fixed):

$$\partial \frac{D(\beta, \beta_0)}{\partial \beta} = -\sum_{i \in \mathcal{M}} y_i x_i,$$

$$\partial \frac{D(\beta, \beta_0)}{\partial \beta_0} = -\sum_{i \in \mathcal{M}} y_i.$$

Perceptron Learning Algorithm (cont.)

- The algorithm uses *stochastic gradient descent* to minimize the piecewise linear criterion.

- This means that rather than computing the sum of the gradient contributions of each observation followed by a step in the negative gradient direction, a step is taken after each observation is visited.

- Hence, the misclassified observations are visited in some sequence, and the parameters β are updated via:

$$\begin{pmatrix} \beta \\ \beta_0 \end{pmatrix} \leftarrow \begin{pmatrix} \beta \\ \beta_0 \end{pmatrix} + \rho \begin{pmatrix} y_i x_i \\ y_i \end{pmatrix}$$

Perceptron Learning Algorithm (cont.)

- Note that ρ is the learning rate, which in this case can be taken to be 1 without loss in generality.
- If the classes are linearly separable, the algorithm converges to a separating hyperplane in a finite number of steps.
- There are a number of problems with this algorithm:
  1. When the data are separable, there are many solutions, and which one is found depends on the starting values.
  2. The number of steps can be very large. The smaller the gap, the longer the time to find it.
  3. When the data are not separable, the algorithm will not converge, and cycles develop. The cycles can be long and therefore hard to detect.

Perceptron Learning Algorithm (cont.)

Pseudo Code

- Input: select random sample from the linearly separable training data set $x_i \in \mathbb{R}^D$ and $y_i \in \{-1, +1\} \; \forall \; i = 1, \ldots, N$;
- Initialize the vector of parameters β_0;
- $k \leftarrow 0$;
- **repeat**
  - $k \leftarrow k + 1$
  - $i \leftarrow k \bmod N$
  - **if** $\hat{y}_i \neq y_i$ **then**
    - $\beta_{k+1} = \beta_k + y_i x_i$
  - **else**
    - no-op
- **until** *converged*;

- Note that if $\hat{y}_i y_i = -1$ (i.e. < 0), then we misclassified!
- At each step, we update the weight vector by adding on the gradient (assuming the learning rate is one).

Artificial Neural Networks

- The central idea of *artificial neural networks* (ANN) is to extract linear combinations of the inputs as derived features, and then model the target as a nonlinear function of these features.

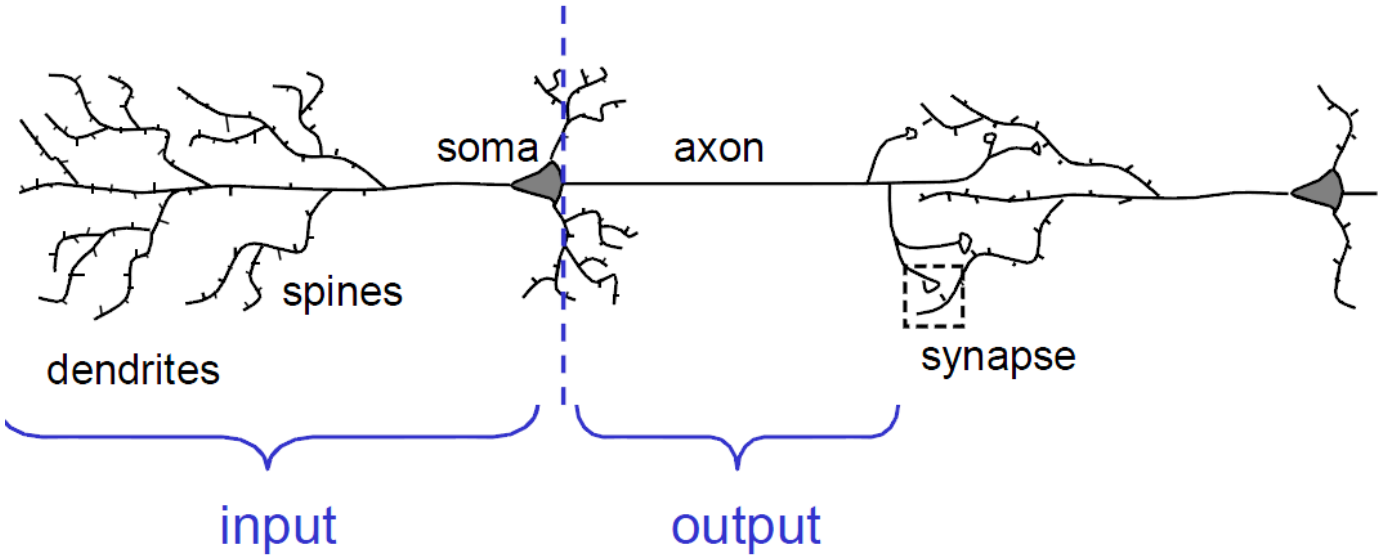
- This machine learning method originated as an algorithm trying to mimic neurons and the brain.

- The brain has extraordinary computational power to represent and interpret complex environments.

- ANN attempts to capture this mode of computation.

Artificial Neural Networks (cont.)

- Specifically, ANN is a formalism for representing functions, inspired from biological systems and composed of parallel computing units, each computing a simple function.

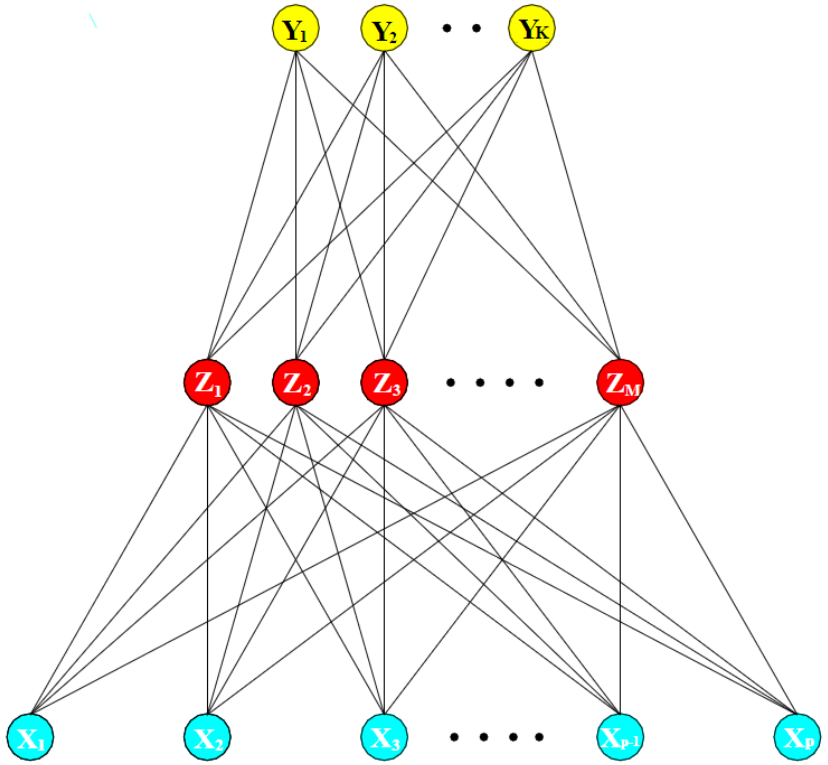


- Parts of the neuron: dendrites, soma (body), axon, synapses

Artificial Neural Networks (cont.)

- We will describe the most widely used ANN, the single layer perceptron (single hidden layer back-propagation network).

- A neural network is a two-stage regression or classification model, typically represented by a network diagram.



Artificial Neural Networks (cont.)

- For regression, typically $K = 1$ and there is only one output unit Y_1 at the top of the network diagram.
- These networks, however, can handle multiple quantitative responses in a seamless fashion.
- For K-class classification, there are K units at the top of the network diagram, with the kth unit modeling the probability of class k.
- There are K target measurements Y_k, $k = 1,\ldots,K$, each being coded as a 0 – 1 variable for the kth class.

Artificial Neural Networks (cont.)

- Derived features Z_m are created from linear combinations of the inputs, and then the target Y_k is modeled as a function of the linear combinations of the Z_m:

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X),\ m = 1, \ldots, M,$$
$$T_k = \beta_{0k} + \beta_k^T Z,\ k = 1, \ldots, K,$$
$$f_k(X) = g_k(T),\ k = 1, \ldots, K,$$

where $Z = (Z_1, Z_2, \ldots, Z_M)$ and $T = (T_1, T_2, \ldots, T_K)$

- The *activation function* $\sigma(v)$ is typically the sigmoid (logistic) function: $\frac{1}{1+e^{-v}}$

Artificial Neural Networks (cont.)

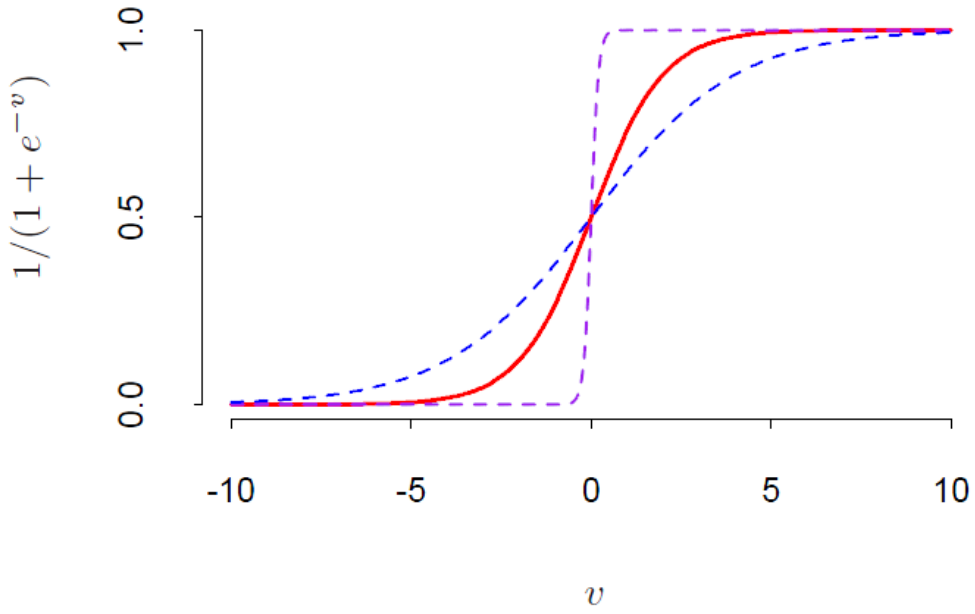


FIGURE 11.3. *Plot of the sigmoid function* $\sigma(v) = 1/(1+\exp(-v))$ *(red curve), commonly used in the hidden layer of a neural network. Included are* $\sigma(sv)$ *for* $s = \frac{1}{2}$ *(blue curve) and* $s = 10$ *(purple curve). The scale parameter* s *controls the activation rate, and we can see that large* s *amounts to a hard activation at* $v = 0$*. Note that* $\sigma(s(v - v_0))$ *shifts the activation threshold from* 0 *to* v_0*.*

Artificial Neural Networks (cont.)

- Neural network diagrams are sometimes drawn with an additional *bias* unit feeding into every unit in the hidden and output layers.

- Thinking of the constant "1" as an additional input feature, this bias unit captures the intercepts α_{0m} and β_{0k} in the model.

- The output function $g_k(T)$ allows a final transformation of the vector of outputs T.

- For regression, we typically choose the identity function $g_k(T) = T_K$

Artificial Neural Networks (cont.)

- Early work in K-class classification also used the identity function, but this was later abandoned in favor of the *softmax* function:

$$g_k(T) = \frac{e^{T_k}}{\sum_{\ell=1}^{K} e^{T_\ell}}$$

- Note that this is exactly the transformation used in the multilogit model, which produces positive estimates that sum to one.

- The units in the middle of the network, computing the derived features Z_m, are called *hidden units* because these values are not directly observed.

Artificial Neural Networks (cont.)

- We can think of the Z_m as a basis expansion of the original inputs X; the neural network is then a standard linear model, or linear multilogit model, using these transformations as inputs.

- However, the parameters of the basis functions are learned from the data, which is an important enhancement over the basis expansion techniques.

- Notice that if σ is the identity function, then the entire model collapses to a linear model in the inputs.

Artificial Neural Networks (cont.)

- Thus, a neural network can be though of as a nonlinear generalization of the linear model, both for regression and classification.

- By introducing the nonlinear transformation σ, it greatly enlarges the class of linear models.

- In general, there can be more than one hidden layer.

- The neural network model has unknown parameters, often called *weights*.

Artificial Neural Networks (cont.)

- We seek values for the weights that make the model fit the training data well.

- We denote the complete set of weights by θ, which consists of:

$$\{\alpha_{0m}, \alpha_m;\ m = 1, 2, \ldots, M\}\ \ M(p+1) \text{ weights},$$
$$\{\beta_{0k}, \beta_k;\ k = 1, 2, \ldots, K\}\ \ K(M+1) \text{ weights}.$$

- For regression, we use RSS as our measure of fit (error function):

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2.$$

Artificial Neural Networks (cont.)

- For classification, we use squared error or cross-entropy (deviance):

$$R(\theta) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log f_k(x_i),$$

and the corresponding classifier $G(x) = \text{argmax}_k f_k(x)$.

- With the softmax activation function and the cross-entropy error function, the neural network model is exactly a linear logistic regression model in the hidden units, and all the parameters are estimated by maximum likelihood.

Artificial Neural Networks (cont.)

- Typically, we do not want the global minimizer $R(\theta)$, as this is likely to be an overfit solution.
- Instead, some regularization is need; this is achieved directly through a penalty term, or indirectly by early stopping.
- The generic approach to minimizing $R(\theta)$ is by gradient descent, called *back-propagation* in this setting.
- The gradient can be easily derived using the chain rule for differentiation.

Artificial Neural Networks (cont.)

- This can be computed by a forward and backward sweep over the network, keeping track only of quantities local to each unit.

- Here is back-propagation in detail for squared error loss.

- Let $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$ and $z_i = (z_{1i}, z_{2i}, \ldots, z_{Mi})$. Then:

$$
\begin{aligned}
R(\theta) &\equiv \sum_{i=1}^{N} R_i \\
&= \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2,
\end{aligned}
$$

Artificial Neural Networks (cont.)

- With the following derivatives:

$$\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)z_{mi},$$

$$\frac{\partial R_i}{\partial \alpha_{m\ell}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)\beta_{km}\sigma'(\alpha_m^T x_i)x_{i\ell}.$$

- Given these derivatives, a gradient descent update at the $(r+1)$st iteration has the form (where γ_r is the *learning rate*):

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}},$$

$$\alpha_{m\ell}^{(r+1)} = \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}},$$

Artificial Neural Networks (cont.)

- We can simplify the derivatives to:

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi},$$
$$\frac{\partial R_i}{\partial \alpha_{m\ell}} = s_{mi} x_{i\ell}.$$

- The quantities δ_{ki} and s_{mi} are "errors" from the current model at the output and hidden layer units, respectively. Note that the output layer errors $\delta_{ki} = (\hat{f}_k(x_i) - f_k(x_i))$

- From their definitions, these errors satisfy the *back-propagation equations*:

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki},$$

Artificial Neural Networks (cont.)

- Using this, the gradient descent updates can be implemented with a two-pass algorithm.
- In the *forward pass*, the current weights are fixed and the predicted values $\hat{f}_k(x_i)$ are computed.
- In the *backward pass*, the errors δ_{ki} are computed, and then back-propagated via the back-propagation equations to give the errors s_{mi}
- Both sets of errors are then used to compute the gradients for the updates.

Artificial Neural Networks (cont.)

- This two-pass procedure is what is known as back-propagation.
- It has also been called the *delta rule*.
- The computational components for cross-entropy have the same form as those for the sum of squares function.
- The advantage of back-propagation are its simple, local nature. In the back-propagation algorithm, each hidden unit passes and receives information only to and from units that share a connection.

Artificial Neural Networks (cont.)

- Usually starting values for weights are chosen to be random values near zero.
- Hence, the model starts out nearly linear and becomes nonlinear as the weights increase.
- Use of exact zero weights leads to zero derivatives and perfect symmetry, and the algorithm never moves.
- Starting instead with large weights often leads to poor solutions.

Artificial Neural Networks (cont.)

- Often neural networks have too many weights and will overfit the data at the global minimum of R.

- An explicit method for regularization is *weight decay*, which is analogous to ridge regression used for linear models.

- We add a penalty to the error function $R(\theta) + \lambda J(\theta)$, where:

$$J(\theta) = \sum_{km} \beta_{km}^2 + \sum_{m\ell} \alpha_{m\ell}^2$$

and λ is a tuning parameter.

Artificial Neural Networks (cont.)

- Larger values of λ will tend to shrink the weights toward zero; typically cross-validation is used to estimate λ.

- Other forms for the penalty (such as *weight elimination*) are:

$$J(\theta) = \sum_{km} \frac{\beta_{km}^2}{1+\beta_{km}^2} + \sum_{m\ell} \frac{\alpha_{m\ell}^2}{1+\alpha_{m\ell}^2},$$

- Also, since the scaling of the inputs determines the effective scaling of the weights in the bottom layer, it can have a large effect on the quality of the final solution; thus, it is best to standardize all inputs.

Artificial Neural Networks (cont.)

- Generally, it is better to have too many hidden units than too few.
- With too few hidden units, the model might not have enough flexibility to capture the nonlinearities in the data.
- With too many hidden units, the extra weights can be shrunk toward zero if appropriate regularization is used.
- Typically the number of hidden units is somewhere in the range of 5 to 100, with the number increasing with the number of inputs and number of training cases.

Artificial Neural Networks (cont.)

- The error function is nonconvex, possessing many local minima, so one must try a number of random starting configurations and the choose the solution giving the lowest error.

- Probably a better approach is to use the average predictions over the collection of networks as the final prediction.

- Another approach is via *bagging*, which averages the predictions of networks training from randomly perturbed version of the training data.

Summary

- Perceptron learning algorithm for separating hyperplanes.
- Artificial neural networks and the back-propagation algorithm for regression and classification problems.