Artificial Neural Network

Lecture # 17

Definition

Artificial Neural Network (ANN) provides an approach to approximate real valued, discrete valued and vector valued target functions. It is very effective in learning to recognize hand written characters, spoken words or faces. The ANN has been inspired by the workings of our neural system. This is a densely interconnected set of units where each unit takes a number of real-valued inputs (possibly the output of other units) and produces a single real-valued output (which may become the input to many others).

To put our neural connections in prospect, the human brain is estimated to have a densely interconnected network of 10^{11} neurons each connected to an average of 10^{4} other neurons. Neuron activity is often excited or inhibited through connections to other neurons. The neuron switching time is of the order of 10^{-3} seconds. The fast response time for the biological systems implies a highly parallel processing of the input data.

Artificial Neural Networks

A supervised learner models the relationship between input and output variables. The neural network technique approaches this problem by developing a functional relation between input and output variables by resembling the architecture of a neuron. Here follows a simple mathematical formulation. Consider the linear model

$$Y = 1 + 2X1 + 3X2 + 4X3$$

Where Y is the output and X1, X2 and X3 are input attributes. The intercept is 1 and 2, 3 and 4 are the coefficients for the input attributes X1, X2 and X3 respectively. The simple relation is shown in Figure 1. In this topology, X1 is the input value and passes through a node, shown by a circle. The value of X1 is multiplied by its weight, which is 2 as noted in the connector. Similarly all the attributes go through a node and a scaling transformation. The last node has no input – it is the intercept. The values of all the connectors go to the output node that predicts Y. The topology in Figure 1 shows an Artificial Neural Network (ANN). The neural networks could also model more complex nonlinear relations and learn through adaptive adjustments of weights of nodes.

Terminology

In neural net terminology, nodes are called units. The first layer of nodes closest to the input is called the input layer or input nodes. The last layer of nodes is called the output layer or output nodes. The transfer function scales the output to desired range. The output layer performs an aggregation function. This simple two layer topology (Figure 1) with one input and one output is called a perceptron. It is the most simplistic form of an ANN. A perceptron is a feed-forward neural net where the input moves in one direction and there are no loops in the topology.

A ANN is used to model nonlinear relations between input and output variables. This is made possible by the presence of more than one layer in the topology, apart from the input and output layers, called hidden layers. A hidden layer contains a layer of nodes that connects inputs from previous layers and applies an activation function. The output is calculated by a more complex combination of input values.

Figure 2 shows a more complex topology with hidden layers. It has four input variables as characteristics of an iris- Sepal length, Sepal width, Petal length and Petal width. An ANN based on iris structure has three-layer structure with three output nodes, one for each class variable. This predicts spices for the iris with the ANN providing output for each class type. A winning class type is selected based on the maximum value of the output class label.

Transfer functions commonly used are: sigmoid, normal bell curve, logistic, hyperbolic or linear functions. The purpose of sigmoid and bell curve is that they provide linear transformation for a particular range of values and a nonlinear transformation for the rest of the values. Because of the presence of multiple hidden layers, one can closely approximate any mathematical continuous relationship between input and output variables.

Figure 1: Shows the architecture of a neural Network

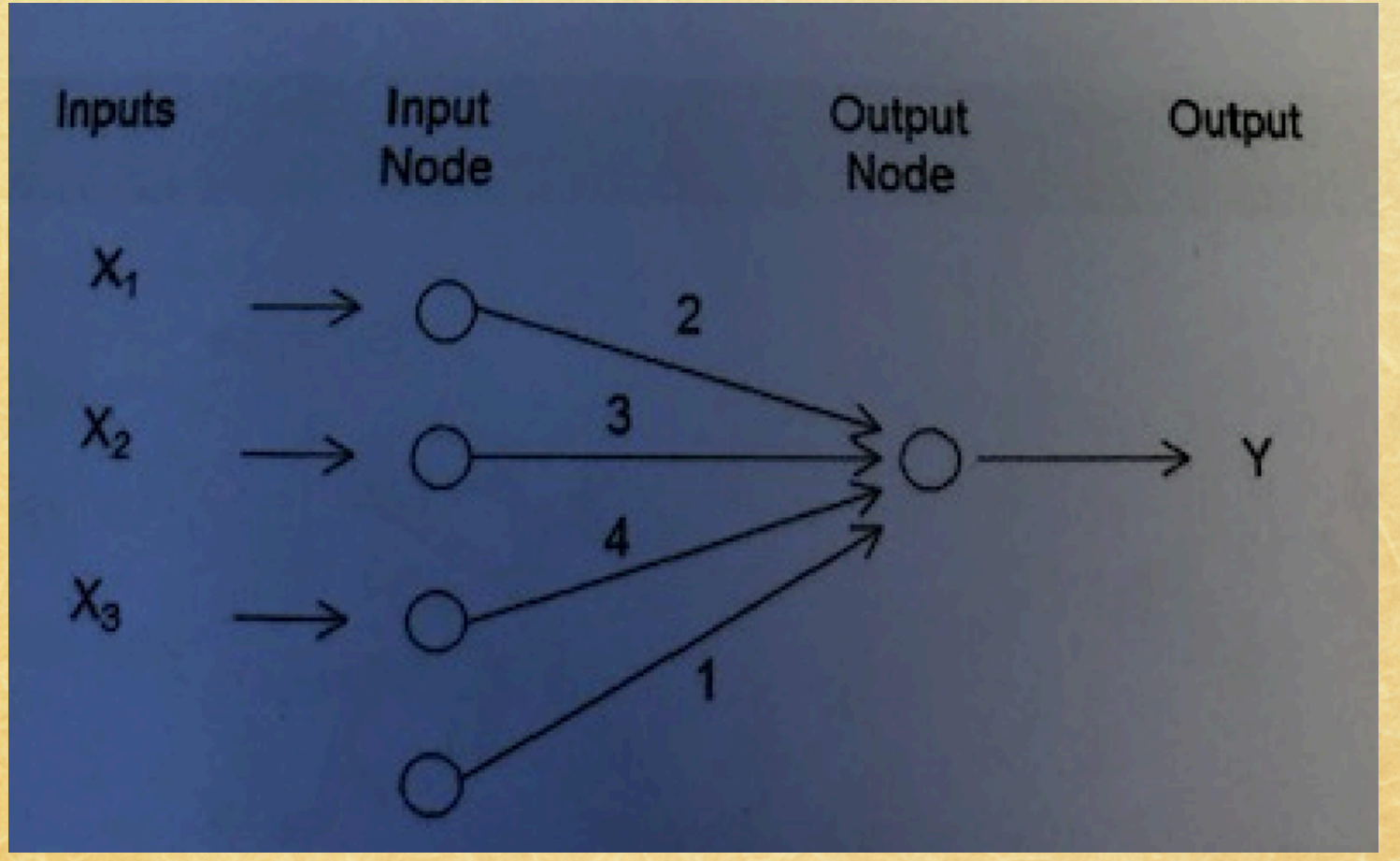
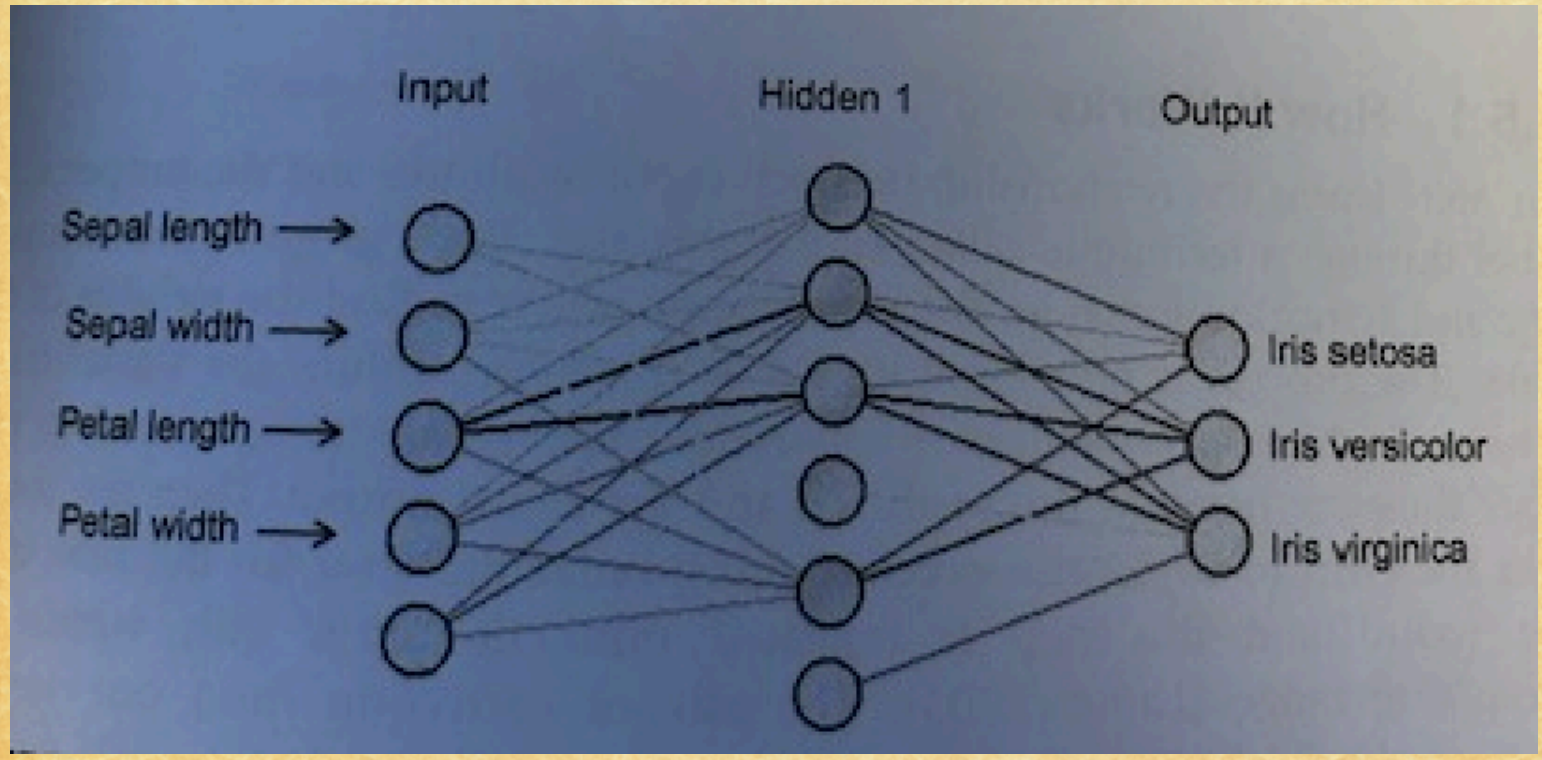


Figure 2: Shows hidden layers in an ANN



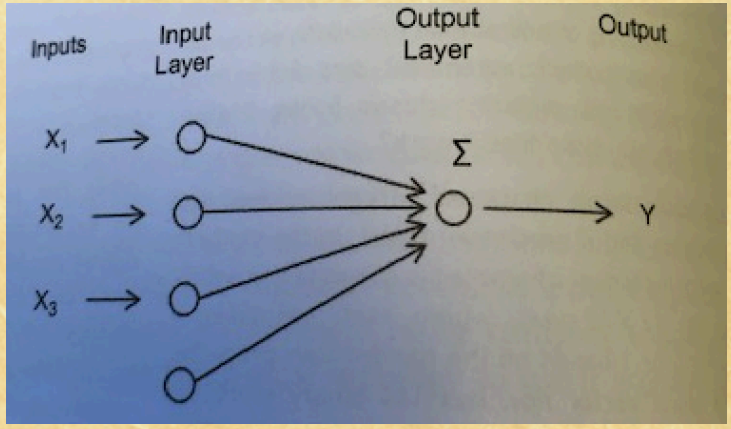
Types of ANN

There are three types of ANN models available: A simple perceptron with one input and one output layer, a flexible ANN model called neural net with all the parameters for complete model building and an advanced AutoMLP (Automatic Multi-layer Perceptron) that combines concepts from genetic and stochastic algorithms. It leverages an ensemble group of ANNs with different parameters like hidden layers and learning rates. It also optimizes by replacing the worst performing model with better ones to maintain an optimal solution.

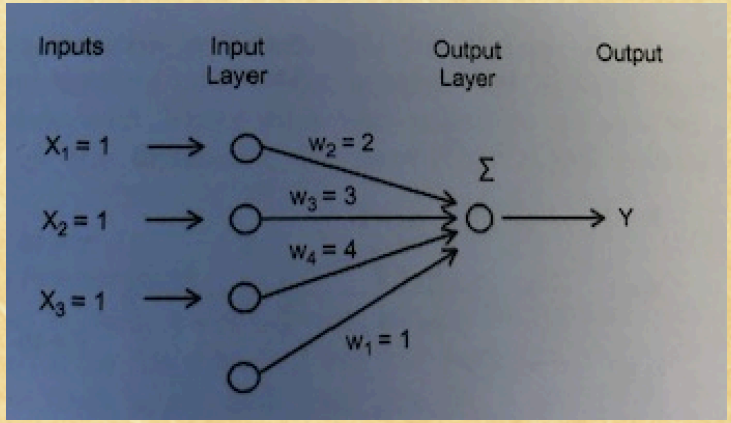
How ANN Works

An ANN learns the relationship between input attributes and output class label through a technique called back propagation. For a given network topology and activation function, the key training task is to find weights of the links. The model uses every training record to estimate the error between the predicted and actual output. Then the model uses the errors to adjust the weights to minimize the error for the next training record and this step is repeated until the error falls within the acceptable range. The rate of correction from one step to another should be managed carefully so that the model does not overcorrect. The steps in developing an ANN from a training dataset include:

Step 1: Determine the topology and Activation Function: For the above example the dataset has three numeric input attributes (X1, X2, X3) and one numeric output (Y). To model the relationship, a topology with two layers and a simple aggregation activation function is being used. There is no transfer function here.



Step 2: Initiation: Assume the initial weights for the four links are 1, 2, 3 and 4. Take an example model and a training record with all the inputs as 1 and the known output as 15. Therefore X1=X2=X3=1 and output Y=15.



Step 3: Calculating errors:

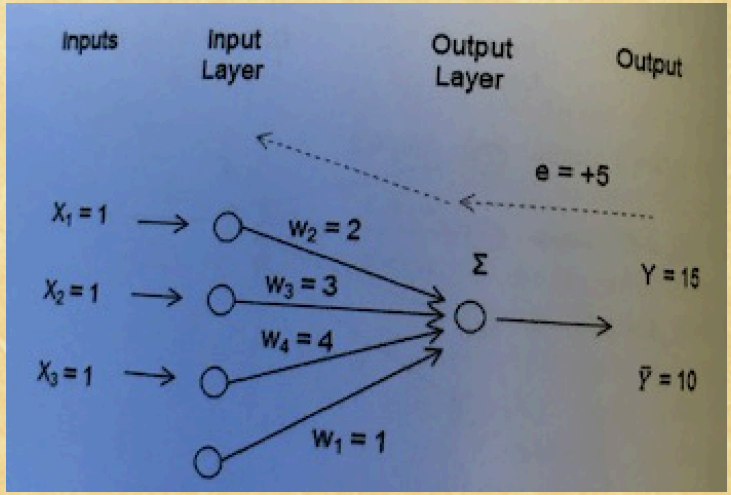
The predicted output of the record can be calculated. This is a feed-forward process when the input data passes through the nodes and the output is calculated. The predicted output Y' according to the current model is

1+ (1 x 2) + (1 x 3) + (1 x 4) =10

The difference between the actual output from the training record and the predicted output is the model error:

e = Y – Y'

The error for this example is 15 – 10 =5



Step 4: Weight Adjustment

This is an important part of the learning process within the ANN. The error estimated is passed back from the output node to all other nodes in the reverse direction. The weights of the links are adjusted from their old value by a fraction of the error. The fraction λ applied to the error is called the learning rate and takes values from 0 to 1. A value close to 1 results in a drastic change to the model for each training record and a value close to 0 results in smaller changes and less correction. The new weight of the link (w) is the sum of the old weight (w') and the product of the learning rate and proportion of the error (λ x e)

$w = w' + \lambda \times e$

The choice of the λ is important. Some models start with λ close to 1 and reduce the value of λ while training each cycle. By this approach any outlier record later in the training cycle will not degrade the relevance of the model.

The current weight of the first link is w2 = 2. Assume the learning rate is $\lambda = 0.5$. the new weight will be $w2=2+0.5 \times 5/3 = 2.83$. the error is divided by 3 because the error is back propagated to three links from the output node. Similarly, the weight will be adjusted for all the links.

In the next cycle a new error will be computed for the next training record. This cycle goes on until all the training records are processed by iterative runs. The same training example can be repeated until the error rate is less than a threshold.

In reality, there are more complex ANN structures with many hidden layers and multiple output links (one for each nominal class value). Because of the numeric calculations, an ANN works best with numeric inputs and outputs.

Perceptron

One type of ANN system is based on a unit called perceptron. A perceptron takes a vector of real-valued inputs, calculates a linear combination of all these inputs and then outputs a 1 if the result is greater than some threshold and -1 otherwise (Figure 3). Given inputs x_1 through x_n, the output $o(x_1,\ldots x_n)$ computed by perceptron is

$o(x_1,\ldots,x_n) = 1$ if $Y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0$

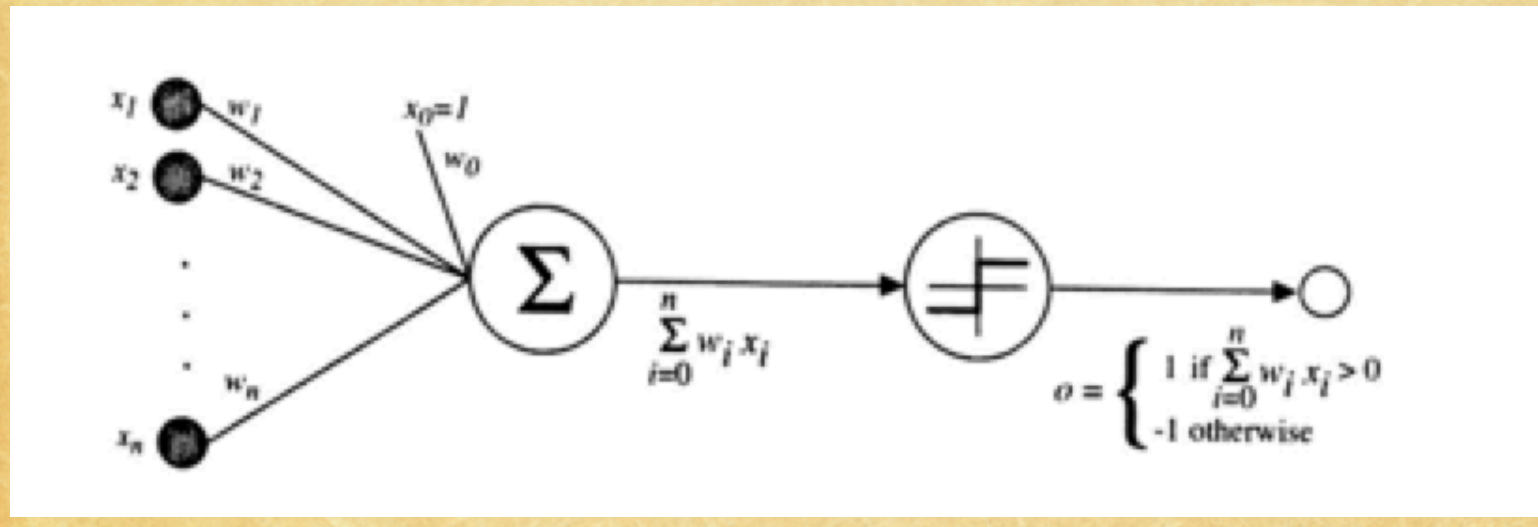
-1 otherwise

where w_1 is a real valued constant or weight that determines the contribution of input x_i to the perceptron output. The quantity $-w_0$ is the threshold that the weighted combination of the inputs should surpass in order for the perceptron to output a 1. To simplify notation, we consider an additional constant input $x_0 = 1$, allowing us to write the above relation as

$$\sum_{i=0}^{n} w_i x_i > 0$$

Or in vector form as $\vec{w}.\vec{x} > 0$ (Figure 3).

Fig 3: A Perceptron with input and output data nods



We could sometimes write the perceptron function as

$$o(\vec{x}) = sgn\,(\vec{w}.\vec{x})$$

Where *sgn =1* if *y > 0* and -1 otherwise.

We can view perceptron as representing a hyperplane decision surface in the *n-dimensional* space of points (instances). The perceptron outputs a 1 for instances lying on one side of the hyperplane. The equation for this hyperplane is

$$\vec{w}.\vec{x} = 0.$$

The Perceptron Training Rule

In ANN we often learn networks of many interconnected units. Let us understand how to learn from a single perceptron. The learning problem here is to find a weight vector that causes the perceptron to produce the correct +/- 1 output for each of the given training examples. Here we consider two learning problems – the perceptron rule and data rule.

We begin with random weights and iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example. This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly. Weights are modified at each step according to the perceptron training rule, which revises the weight w_i according to the rule

$w_i \leftarrow w_I + \Delta w_i$

Where $\Delta w_I = \eta (t - o) x_i$

Here t is the target output for the current training example, o is the output

generated by the perceptron, and η is a positive constant called the learning rate. The role of the learning rate is to moderate the degree to which weights are changed at each step. It is usually set to some small value (eg 0.1) and is sometimes made to decay as the number of weight-tuning iterations increases. When the training example is correctly classified, by the perceptron, the value of *(t-o)* is equal to zero, making $\Delta w_i = 0$ so that no weights are updated.

Suppose the perceptron outputs -1, when the target output is +1. To make the perceptron output a +1 instead of -1, the weight must be altered to increase the value of *w.x*. For example, if $x_i=0.8$, $\eta=0.1$, $t=1$ and $o=-1 \rightarrow \Delta w_i = \eta\,(t-o)x_i = 0.1\,(1-(-1))\,0.8 = 0.16$. This learning procedure converges within a limited number of applications of the perceptron training rule.

Multilayer Networks and Backpropagation

A single perceptron can only express linear decision surfaces. In contrast a kind of multilayer networks learned by the backpropagation algorithm are able of expressing a variety of nonlinear decision surfaces. Here the speech recognition distinguishing among 10 possible vowels (“hid”, “had”, “head”, “hood”) all in the context of task involves (Figure 4). As shown in Figure 4, it is possible for the multilayer network to represent highly non-linear decision surfaces that are much more expressive than the linear decision surfaces of single units shown earlier. To learn multilayer networks we use a gradient descent algorithm (Figure 6).

Figure 4: An ANN designed for speech recognition

(From Machine Learning by Tom. M. Mitchell).

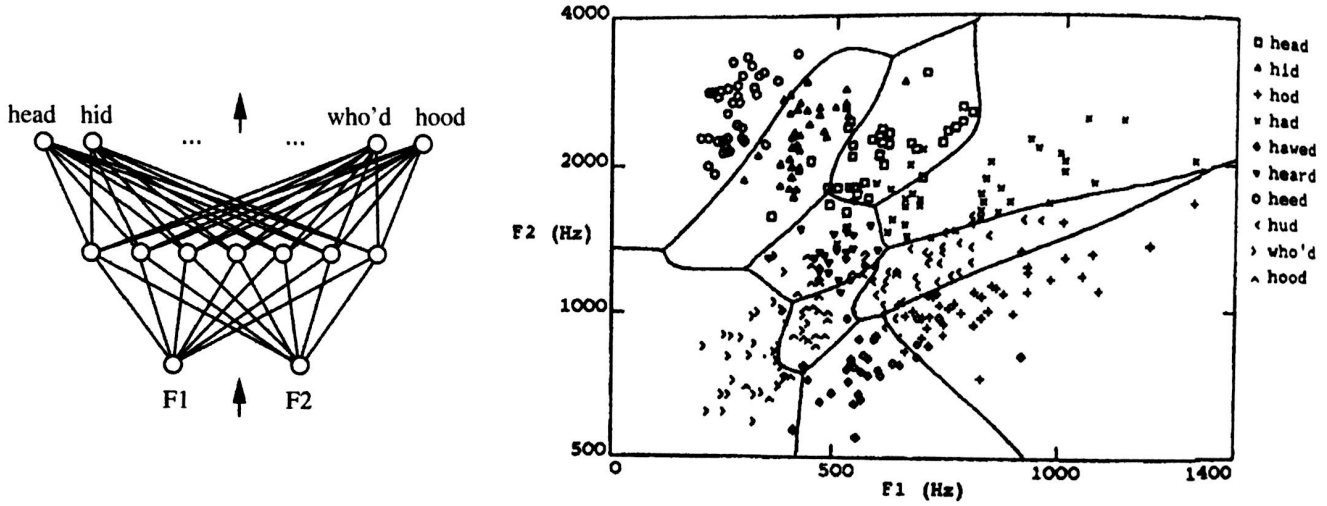


FIGURE 4.5
Decision regions of a multilayer feedforward network. The network shown here was trained to recognize 1 of 10 vowel sounds occurring in the context "h_d" (e.g., "had," "hid"). The network input consists of two parameters, F1 and F2, obtained from a spectral analysis of the sound. The 10 network outputs correspond to the 10 possible vowel sounds. The network prediction is the output whose value is highest. The plot on the right illustrates the highly nonlinear decision surface represented by the learned network. Points shown on the plot are test examples distinct from the examples used to train the network. (Reprinted by permission from Haung and Lippmann (1988).)

The unit we can use as the basis for constructing multilayer networks is a unit whose output is a nonlinear function of its inputs. One solution here is the sigmoid unit- a unit much like a perceptron, but based on a smoothed, differentiable threshold function (Figure 5). Like perceptron, the sigmoid function first calculates a linear combination of its inputs then applies a threshold to the result. In the case of the sigmoid unit, the threshold output is a continuous function of its input. More precisely, the sigmoid unit computes its output o as

$$\sigma = \sigma(\vec{w}.\vec{x})$$

where

$$\sigma(y) = \frac{1}{1+e^{-y}}$$

σ is the sigmoid function or alternatively the logistic function.

The sigmoid function has the useful property that its derivative is easily expressed in terms of its output. This will be useful in the gradient descent learning rule when we need to take the derivative of the function (Figure 5).

Figure 5: The sigmoid threshold unit for ANN (Figure from Machine Learning by Tom M. Mitchell).

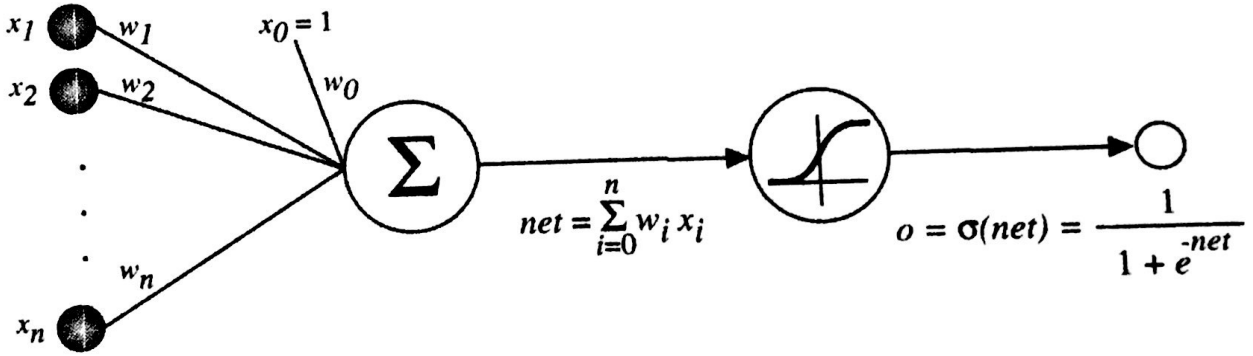


FIGURE 4.6
The sigmoid threshold unit.

The Backpropagation Algorithm

This learns the weights for a multilayer network, given a network with a fixed set of units and interconnections. It employs gradient descent to minimize the squared error between the network output values and the target values for these outputs.

Since we are considering networks with multiple output units rather than single units as before, we first define E to sum the errors over all of the network output units

$$E(\vec{w}) -= \frac{1}{2} \sum_{d} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

where outputs are the set of output units in the network and t_{kd} and o_{kd} are the target and output values associated with the k-th output unit and training example, d.

The learning problem for backpropagation is to search a large hypothesis space defined by all possible weight values for all the units in the network (Table 1). The situation can be visualized in terms of an error surface similar to linear units (Figure 6). Gradient descent is used to find a hypothesis to minimize E. The error surface in the case of multilayer networks can contain multiple minima. The global descent in this case will converge to a number of minima instead of a single minima.

Figure 6:The gradient descent technique finds the minima of the error.

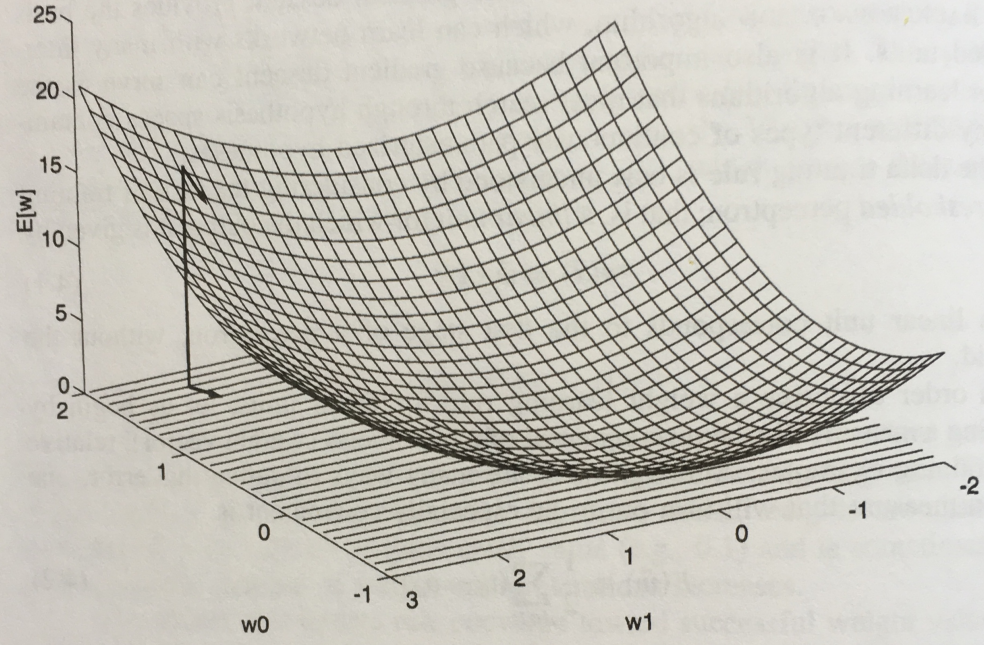


FIGURE 4.4
Error of different hypotheses. For a linear unit with two weights, the hypothesis space H is the w_0, w_1 plane. The vertical axis indicates the error of the corresponding weight vector hypothesis, relative to a fixed set of training examples. The arrow shows the negated gradient at one particular point, indicating the direction in the w_0, w_1 plane producing steepest descent along the error surface.

The structure of a code for backpropagation in ANN (From Machine Learning by Tom M. Mitchell)

BACKPROPAGATION(*training_examples*, η, n_{in}, n_{out}, n_{hidden})

Each training example is a pair of the form $(\vec{x}, \vec{t})$, *where* $\vec{x}$ *is the vector of network input values, and* $\vec{t}$ *is the vector of target network output values.*

η *is the learning rate (e.g., .05).* n_{in} *is the number of network inputs,* n_{hidden} *the number of units in the hidden layer, and* n_{out} *the number of output units.*

The input from unit i into unit j is denoted x_{ji}, *and the weight from unit i to unit j is denoted* w_{ji}.

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers (e.g., between $-.05$ and .05).
- Until the termination condition is met, Do
  - For each $(\vec{x}, \vec{t})$ in *training_examples*, Do

Propagate the input forward through the network:

1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{T4.3}$$

3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k \tag{T4.4}$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = \eta \, \delta_j \, x_{ji} \tag{T4.5}$$

Summary

Data preparation: The input data should be numeric attributes. The ANN model will not work with categorical or nominal data types. The output is the number of classes the input data will be assigned to.

Modeling operator and parameters: The training dataset is connected to the NN operator which accepts real data types and normalizes the values. These parameters are available in ANN for users to change and customize:

1. Hidden layer: Determines the number of layers, size of each hidden layer and names of each layer for easy identification in the output. The default size of the node is -1 calculated as [(# of attributes+ # of classes) / 2] + 1

2. Training cycle: this is the number of times a training cycle is repeated. In a NN, every time a training record is considered, the previous weights are quite different. Therefore, it is needed to repeat the cycle many times.

3. Learning Rate: this is the rate with which the error between the inout and output reduces.

Convolutional Neural Nets (CNN)

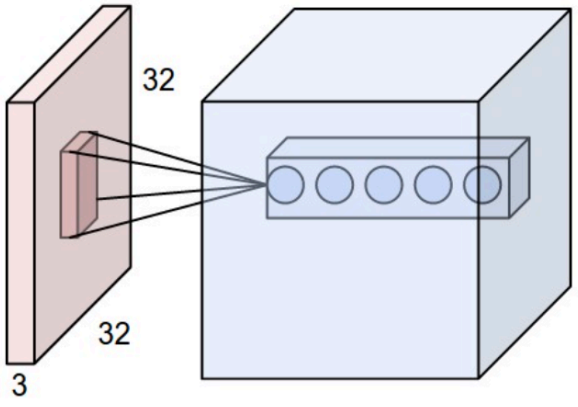
Convolutional Neural Nets (CNN) are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The network has a single differentiable score function: from the raw image pixels on one end to class scores at the other. It also has a loss function on the last (fully-connected) layer.

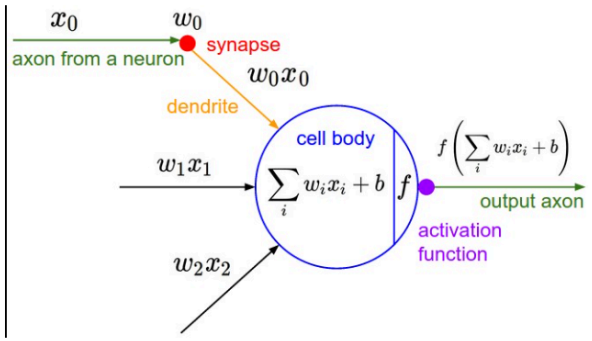
The difference between the conventional and convolutional neural nets: ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

CNN Architecture

3D volumes of neurons. Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. (the word depth here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network.)- (Figure 7a). For example, the input images have an input volume of activations, and the volume has dimensions 32x32x3 (width, height, depth respectively). The neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would have dimensions 1x1x10, because by the end of the ConvNet architecture we will reduce the full image into a single vector of class scores, arranged along the depth dimension (Figure 7b)

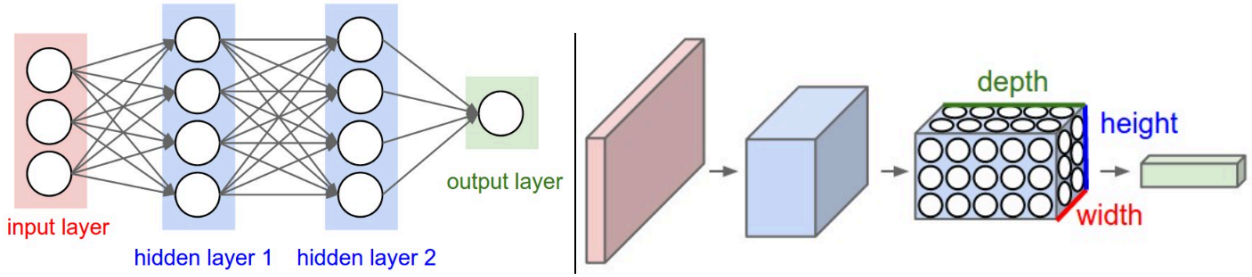
Figure 7a: Shows details of the archtecture





Left: An example input volume in red (e.g. a 32x32x3 CIFAR-10 image), and an example volume of neurons in the first Convolutional layer. Each neuron in the convolutional layer is connected only to a local region in the input volume spatially, but to the full depth (i.e. all color channels). Note, there are multiple neurons (5 in this example) along the depth, all looking at the same region in the input - see discussion of depth columns in text below. **Right:** The neurons from the Neural Network chapter remain unchanged: They still compute a dot product of their weights with the input followed by a non-linearity, but their connectivity is now restricted to be local spatially.

Fig 7b: Architecture of an CNN



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Layers for a CNN

A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer (exactly as seen in regular Neural Networks). We will stack these layers to form a full ConvNet architecture.

A simple ConvNet classification could have the architecture [INPUT - CONV - RELU - POOL – FC] as briefly described below:

INPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.

CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.

RELU layer will apply an elementwise activation function, such as the max (0,x) thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).

POOL layer will perform a down sampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].

FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

ConvNets transform the original image layer by layer from the original pixel values to the final class scores. Note that some layers contain parameters and other don’t. In particular, the CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons). On the other hand, the RELU/POOL layers will implement a fixed function. The parameters in the CONV/FC layers will be trained with gradient descent so that the class scores that the ConvNet computes are consistent with the labels in the training set for each image.

Summary

- A ConvNet architecture is in the simplest case a list of Layers that transform the image volume into an output volume (e.g. holding the class scores)
- There are a few distinct types of Layers (e.g. CONV/FC/RELU/POOL are by far the most popular)
- Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function
- Each Layer may or may not have parameters (e.g. CONV/FC do, RELU/POOL don't)
- Each Layer may or may not have additional hyperparameters (e.g. CONV/FC/POOL do, RELU doesn't)

Gradient Descent in ANN

Gradient descent for every weight wl ij and every bias bl I in the neural network is

$$w_{ij}^{(l)} = w_{ij}^{(l)} - \alpha \frac{\partial}{\partial w_{ij}^{(l)}} J(w,b)$$

$$b_{i}^{(l)} = b_{i}^{(l)} - \alpha \frac{\partial}{\partial b_{i}^{(l)}} J(w,b)$$

Where J(w,b) are cost functions. Each step in the NN depends on the slope of the cost vs. error term with respect to the weight. Therefore the gradient of this is measured and plotted, with the weight corresponding to the minimum on the cost vs. error plot selected (Figure 8).

Figure 8: Shows the change in error as a function of the weight

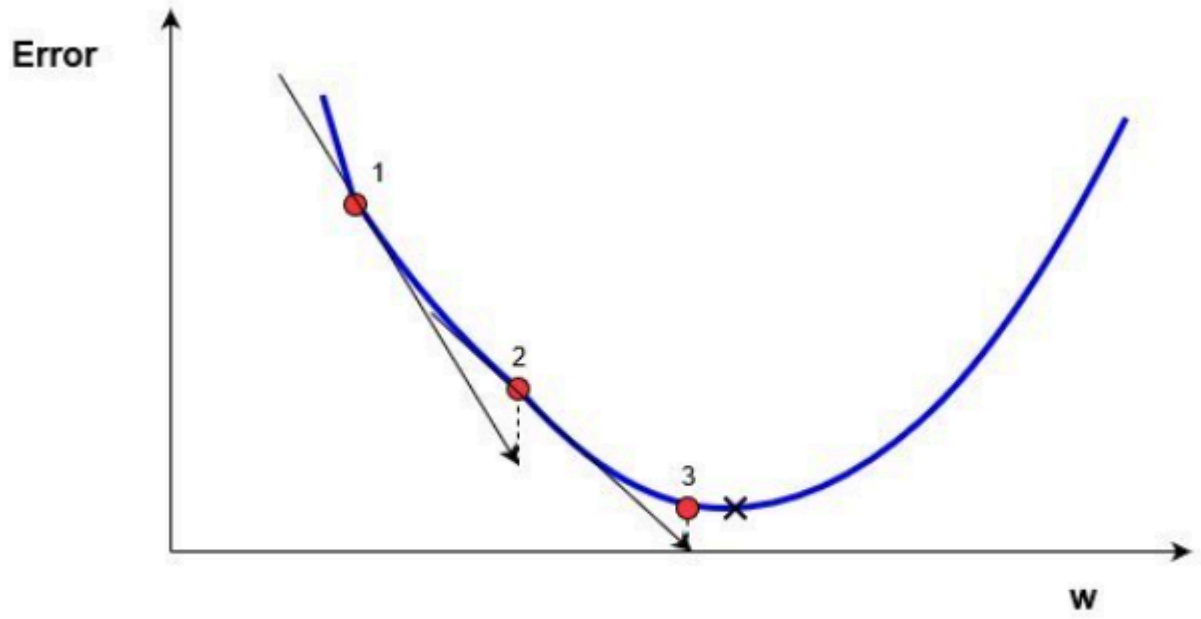


Figure 8. Simple, one-dimensional gradient descent

Figure 9: Cost function vs. error

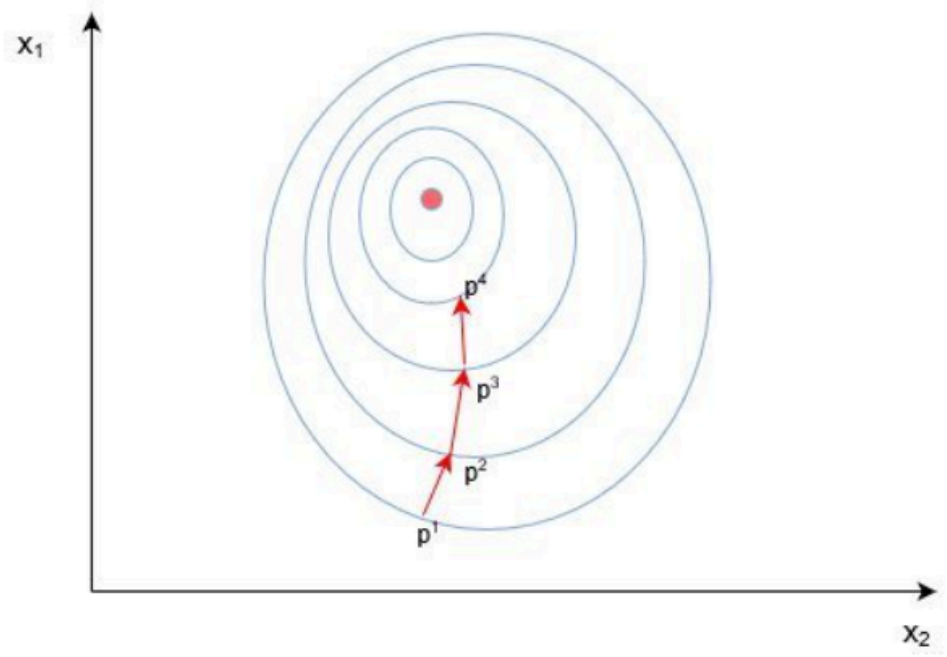


Figure 9. Two-dimensional gradient descent

Sources used for this study

1. *Machine Learning*

by Tom M. Mitchell (Chapter 4)

2. *http://cs231n.github.io/convolutional-networks/*

CS231 in Convolutional Neural Networks for Visual Recognition