Summary

- Artificial Neural Networks (ANNs) are networks of interconnected simple units that are based on a greatly simplified model of the brain. ANNs are useful learning tools by being able to compute results quickly interpolating data well.
- There are two main types of ANNs, feed forward networks and recurrent networks. Perceptrons is a special case of feed forward networks with only input and output nodes.
- Three main Perceptron learning algorithms are covered: mistake bound Perceptron algorithm, Perceptron training rule and the Delta rule. The delta rule uses gradient descent which makes it easy to compute what changes are needed to optimize the network.
- The backpropagation learning algorithm is widely used for multi-layer feed forward network. This uses gradient descent as well.
- Bayesian learning is based on statistics and knowledge of prior statistics to classify or predict. Bayes Theorem is central to Bayesian learning.
Artificial Neural Networks (ANNs) are loosely modeled after the brain. ANNs are composed of units (also called nodes) that are modeled after neurons, with weighted links interconnecting the units together. The main difference between ANNs and other learning mechanisms is that it is composed of these simple units and they work together in a highly parallel manner.

Artificial Neural Networks should be used when:

- Target function is not known
- Readability of result is not important
- Multi-dimensional input
- Continuous or discrete input values
- Possibly noisy Input data
- Output is a vector of continuous or discrete values

Applications of ANNs:

- Control (such as cars in ALVINN)
- Recognize/Classify (written/spoken words)
- Predict (trends)

Typical Unit:

In the unit shown above, there are n input units ($X_i$) connected with n links ($W_i$). It also has one output. The ANN unit is composed of two main parts: the first part sums the input and sends it to the threshold function. If the activation is greater than 0 then the unit activates and sends a “1” as the output, otherwise it sends a 0 (or −1). The $X_0$ can be set to any value so that instead of tuning the threshold function to activate at some fixed point $Y$, $X_0$ can be set to $-Y$. 
Types of Neural Networks

Feed Forward Networks:

- Activation flows in one direction only
- **Multi-layer Feed Forward Networks:**
  - Can approximate any function
  - Not guaranteed to reach the best possible approximation, could be trapped in a local minima

- **Perceptrons** (No hidden units)
  - Guaranteed to converge to any linearly separable function
  - Simpler to work with and see results, only two layers

Recurrent Networks:

- Cycles used to allow for states
Perceptron Learning

**Perceptrons**

Perceptrons can represent any linearly separated surface. That is, when the values of each input are plotted into n-dimensional space (where n is the number of inputs), if there can be a straight plane that can divide all members as true on one side and false on the other. Although difficult to visualize with many input variables, it is easy to show with two variables (where the line represents the plane in the 2D space).

Many functions such as these are possible, such as “and” and “or”. Functions that are not possible to describe with perceptrons cannot be linearly separated, such as ”XOR”.

**Learning**

Learning algorithms are used to change the weights of the links between units. The goal of the learning algorithms will always be to reduce the error of the training set to the target.

Let \( \mathbf{w} \) be the vector of input weights and \( \mathbf{x} \) be the vector of input to the ANN.

Approaches to Perceptron Learning:

- **Mistake Bound Perceptron Algorithm**
  - Initialize \( \mathbf{w} = \mathbf{0} \)
  - For each training is sufficient
    - Predict 1 ifff \( \mathbf{w}^{\top} \mathbf{x} > 0 \) else 0
    - Mistake on positive: \( \mathbf{w} \leftarrow \mathbf{w} + \mathbf{x} \)
    - Mistake on negative: \( \mathbf{w} \leftarrow \mathbf{w} - \mathbf{x} \)

- **Perceptron Training Rule**
  - Initialize \( \mathbf{w} \) = small random numbers
  - Let \( \eta \) represent the learning rate, which is used to control the change at each step. \( t \) is the target of the unit, \( o \) is the output of the unit.
o For each training example
  ▪ $\Delta w_j \leftarrow \eta (t-o)x_j$
  ▪ $w_j \leftarrow w_j + \Delta w_j$

[Delta Rule [Gradient Descent]]
  o Initialize $\vec{w} = $ small random numbers
  o Until termination condition is met (error bound, or iterations of training examples)
    ▪ Initialize all $\Delta w_i \leftarrow 0$
    ▪ For each training example $(\vec{x}, t)$
      ▪ Compute the output of each node: $O(\vec{x})$
      ▪ For each weight unit $w_i$: $\Delta w_i \leftarrow \Delta w_i + \eta (t-o)x_j$
    ▪ For Each weight unit $w_i$: $w_i \leftarrow w_i + \Delta w_i$

**Delta Rule**

- Attempts to minimize the squared error of the training examples
- Guaranteed that there is a single minimum error
- Uses Sigmoid function $\sigma(x): 1/(1+e^{-x})$
- $d\sigma(x)/dx = \sigma(x)(1-\sigma(x))$
- The sigmoid function is used because the derivative is very easy to find
- Gradient: $\nabla E[\vec{w}] = [dE/dw_0, dE/dw_1, dE/dw_2, \ldots, dE/dw_n]$
- Training Rule: $\Delta \vec{w} = -\eta \nabla E[\vec{w}]$

**Advantages of Delta training rule versus Perceptron training rule:**

- Guaranteed to always converge to a hypothesis with minimum squared error (with a small learning rate)
- Allows for noise in the data
- Allows for non-separable functions
**Backpropagation Algorithm**

The backpropagation algorithm is a multi-layer network using a weight adjustment based on the sigmoid function, like the delta rule. The backpropagation method, as well as all the methods previously mentioned are examples of supervised learning, where the target of the function is known.

The following is an example of the backpropagation algorithm working on a small Artificial Neural Network. The Network has a single hidden layer of size two and input and output nodes of size 3.

![Neural Network Diagram](image)

Initialize the weighted links. Typically the weights are initialized to a small random number.

Then, for each training example in the testing set:

![Neural Network Diagram](image)

Input the training data to the input nodes, then calculate \( O_k \), which is the output of node \( k \). This is done for each node in the hidden layer(s) and output layer.

![Neural Network Diagram](image)

Then calculate \( \delta_k \) for the each output node, where \( t_k \) is the target of the node:

\[
\delta_k \leftarrow O_k(1 - O_k)(t_k - O_k)
\]
Now calculate $\delta_k$ for the each hidden node:

$$\delta_k \leftarrow \sigma_k (1 - \sigma_k) \sum_{k\in \text{child}(h)} w_{h,k} \delta_k$$

Finally adjust the weights of all the links, where $x_i$ is the activation and $\eta$ is the learning rate:

$$W_{i,j} \leftarrow W_{i,j} + \eta \delta_j x_i$$

Most likely the neural network will need to be trained at many iterations of the training set to find an acceptable approximation of the function it is being trained on.

---

**Bayesian Learning**

Bayesian learning is based on probability distributions that are previously known. Using these statistics, a Bayesian network can learn to classify data and predict outcomes.

**Bayes Theorem:**

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

**A simple proof of Bayes Theorem:**

By definition,

$$P(A|B) = \frac{P(A,B)}{P(B)}$$

It follows that: $P(A,B) = P(A|B)P(B)$

By the same reasoning: $P(A,B) = P(B|A)P(A)$

Therefore, $P(B|A)P(A) = P(A|B)P(B)$, since both equal to $P(A,B)$

Then,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$
Definitions

**Artificial Neural Networks:** (ANNs) these networks allow for learning using highly parallel series of simple units and are suited for data that is noisy and vector based.

**Backpropagation:** a learning algorithm for multi-layered feed forward networks that uses the sigmoid function

**Hidden layer:** the set of nodes that are not input or output units

**Learning rate:** a value greater than 0 but less than 1, this is used so that the weights on the links do not change to quickly, or the ANN might never converge onto the optimal solution.

**Linearly separable function:** A function where if plotted in a n-dimensional plane, the negative and positive examples of the function can be totally separated using a straight plane across the space.

**Multi-layer Feed Forward Networks:** a network with at least one unit that is not output or input, where the direction of data flow is in only one direction.

**Perceptrons:** a network with no units that are output or input, where the direction of data flow is in only one direction.

**Supervised learning:** all learning algorithms where the known targets are used to adjust the network.

**Target:** The expected output of the input. This is used to calculate the error.

**Threshold function:** the function to decide whether a unit should fire or not. Typically 1 for exceeding the threshold and 0 or –1 otherwise.

**Units/Nodes:** simple elements of a ANN, they take in input from other nodes or training data, sum up the data and applies a threshold function to decide what output to send.

**Weighted links:** connects units together, conceptually shows the strength of the bond between two units.