Table of contents

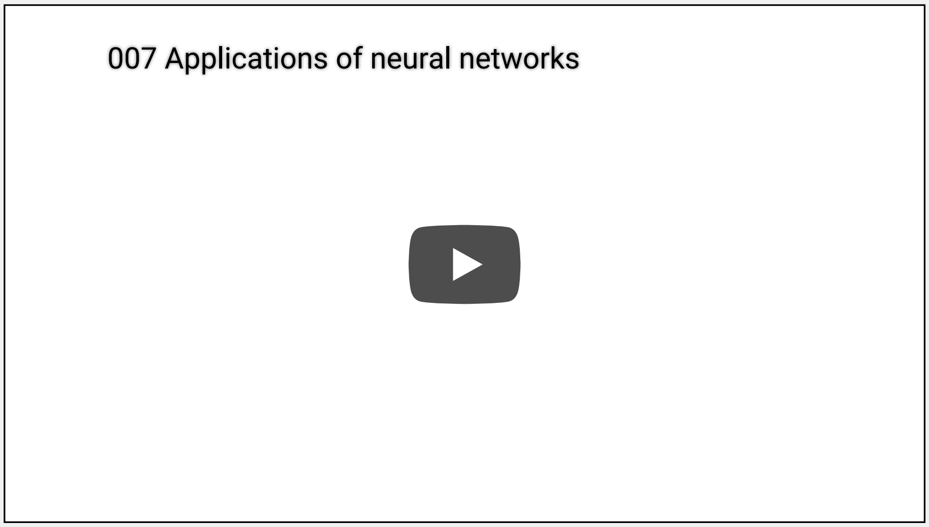
1. Introduction

Most of this material is derived from the text, Han, Kamber and Pei, Chapter 9, or the corresponding powerpoint slides made available by the publisher. Where a source other than the text or its slides was used for the material, attribution is given. Unless otherwise stated, images are copyright of the publisher, Elsevier.

Here we continue the topic of classification and prediction, firstly introducing the classic technique of regression, or fitting a line, and then the bio-inspired technique of neural networks and a brief overview of some other "soft" techniques.

For in-depth learning on neural nets and other soft techniques, the courses *Neural Networks, Deep Learning and Bio-inspired Computing* COMP8420 and COMP4660 are recommended.

ACTION: If you would like more information on the applications for neural networks, refer to this page https://www.smartsheet.com/neural-network-applications#real-world-and-industry-applications-of-neural-networks or watch this youtube video.



ACTION: You can watch this video lecture overview of the classsification and prediction topic if you find it helpful but all it covers is also in the written notes. Note that this recording covers topics in this e-book as well as topics to be addressed in the subsequent e-book.

Prerecorded lecture on Regression, NN, SVM, LL

2. Linear Regression (Not in textbook)

Linear Regression

- models relationship between one *dependent variable* (numerical target variable) and *explanatory variables* (other variables)
  - Input: independent, explanatory, predictor variables
  - Output: one dependent, target, response variable
- provides a way to **predict a numerical variable** given other variables. The other variables are typically numeric too. Ordinal variables can be mapped to numeric values if required. Other categorical variables can also be mapped to numeric values but be very careful to assess the effect of such variables on results as they may obscure more informative relationships.

Regression is different from classification

- Classification predicts categorical class labels
- Regression models continuous-valued functions and is a *numerical prediction* method because the values for explanatory variables of an unseen object can be plugged in to the model to predict the value of the dependent variable.

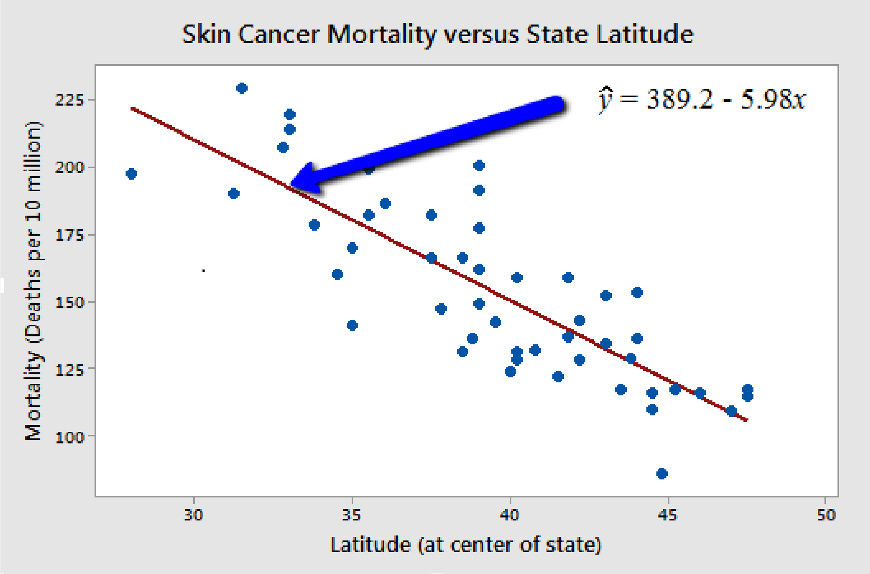
Types of relationships considered in regression

Deterministic relationship

- Relationship between explanatory and dependent variables is deterministic
- If we know the value of an explanatory variable then we can predict dependent variable without error
- For example:
  - $Circumference = \pi \times diameter$
  - $Fahrenheit = \frac{9}{5} \times Celsius + 32$

Statistical relationship

- Relationship between explanatory and dependent variable is not perfect
- For example:
  - Height and weight: as height increases, you'd expect weight to increase, but not perfectly
- Another example of a statistical relationship:



  - Dependent variable is the mortality due to skin cancer (number of deaths per 10 million people)
  - Explanatory variable is the latitude of State (measured at the centre of the State in US)
  - Living in a northern area may reduce the mortality ratio but the relationship is not perfect.

2.1. Simple Linear Regression

Simple linear regression is a statistical method that allows us to study relationships between two continuous variables:

- One variable, denoted x, is regarded as the **predictor**, explanatory, or independent variable
- The other variable, denoted y, is regarded as the **response**, outcome, target, or dependent variable
- The goal is to predict the value of response variable when the value of predictor variable is given

Simple linear regression gets its adjective "simple," because it concerns the study of **only one predictor variable**. In contrast, multiple linear regression concerns the the general case of two or more predictor variables.

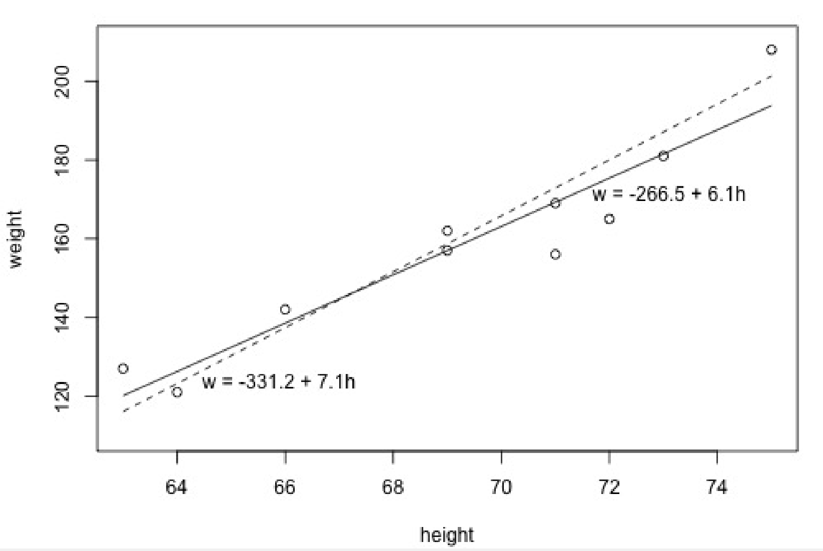
Linear Regression Model

Simple linear regression models the relation between predictor and response variables as follows:

$$\hat{y} = w_0 + w_1 \times x$$

where w_0 (y-intercept) and w_1 (slope) are coefficient parameters and $\hat{y}$ is a predicted response variable based on the predictor variable x. If the value of coefficient parameter is positive then there is a positive relation between predictor and response variable (the value of response variable increases as the value of predictor variable increases).

The relationship between the two variables can be plotted in x-y plane as follows:



This scatter plot (dots) represent an observed pair of weight and height. Here the height is the predictor variable, and the weight is the response variable. In addition, two linear relations are suggested in this graph. You may have two natural questions from above graph:

- which line is the best line? and
- how can we find it?

What is the Best Fitting Line?

Let's assume that we have a training data $D = \{(x_1, y_1), (x_2, y_2), ...(x_n, y_n)\}$ which consists of n-pairs of x and y.

The goal of parameter estimation is to estimate proper values of w_0 and w_1. To estimate parameters, we first define the sum of square error:

$$E = \sum_{i=1}^{|D|}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - (w_0 + w_1 \times x_i))^2,$$

where $\hat{y}$ is the predicted response variable. The sum of square error computes, first, the difference between the true value y and the predicted value $\hat{y}$, second, square of the difference, and then summation of the squared difference across all training data points.

Parameter estimation finds the values for w_0 and w_1 that minimise E and so this technique is called the **least square method**.

Due to the effect of squaring the differences above, linear regression is very senstitive to outliers that fall well away from the line--that is the whole fitted line will be pulled away from the non-outlying values.

How to evaluate performance?

The parameter estimation technique suggests an evaluation technique. Accuracy does not make sense here as there are no classes; accuracy requires the independent variable value to fall either inside the class or outside the class.

The usual methods for assessing performance are

- **MAE, Mean Absolute Error.** It is the average of the individual errors, each one computed as the absolute value of the difference between the actual value and the predicted value. MAE of zero is a perfect fit. As there is no squared term in MAE, it does not penalise long-distance outliers as much as the regression fitting does.
- **RMSE, RMSD, root mean square error.** It is the square root of the average of squared errors for each value of the dependent variable, where error is the simple difference of the actual value and predicted value. It is equal to the standard deviation of the errors, and so tells you how spread out from the prediction line the actual observations are. An RMSE value of zero represents a perfect fit and there is no upper bound.
- **R-squared**: the goodness of fit or coefficient of determination. It is the proportion of the dependent variable variance that is explained by a linear model. R-squared = Explained variance / Total variance. More precisely, it is 1 minus (the ratio of the sum of squared prediction error for each value of the dependent variable to the sum of squared difference from the mean for each value of the dependent variable). A value of zero says the model is only just as good as simply predicting the mean value every time, and a value of one represents a perfect fit. Minor variations include pseudo-R-squared methods and adjusted or predicted R-squared.
- **Plot** the predictions vs observations on a Cartesian plane and visually assess if the points appear to line up. It is a good idea to *do this every time* for linear regression, in combination with quantitative measures. Are the points *close enough* to the theoretical *prediction = observation* line? Do they fall somewhat symmetrically around that line? Are there too many points a long way off? Would a line fitted to the predictions vs observations fall close to the *prediction = observation* line? If it doesn't look good, it probably isn't.

Obviously these approaches should be applied to test data rather than training data for a proper assessment, and indeed **multifold cross-validation** may be applied using the quantitative performance measures above in place of accuracy.

2.2. Multiple Linear Regression

In many applications, it is more general to have more than one predictor variable.

Multiple linear regression is a direct extension of the linear regression model.

Let's assume that we have n predictor variables for each response variable, i.e., $(x_1, x_2, ..., x_n, y)$.

In the multiple linear regression model, the relation between a response variable and predictor variables are modelled as:

$$\hat{y} = w_0 + w_1 \times x_1 + w_2 \times x_2 ... + w_n \times x_n$$

Again, parameters $w_0, .., w_n$ can be estimated in least square fashion:

$$E = \sum_{i=1}^{|D|} (y_i - \hat{y}_i)^2$$

A least square method will find optimal parameters to minimise the squared error E.

Significance of the weight

Most regression libraries or programs support significance testing for each weight. Usually, significant weights are denoted by *** (or ** or * depending on the significance level).

The significant attributes indicate that the effect (or influence) of the attribute is not just random! In other words, those significant **attributes have high influence on predicting your response variable**. This alone may be valuable information, even if you discard the linear model itself.

Performance evaluation

The methods for evaluation of simple linear regression apply straightforwardly to the multiple variable case.

3. Classification or Prediction by Neural Networks (Text: 9.2)

Neural Network Learning

- Neural networks were first investigated by psychologists and neurobiologists to develop and test computational analogues of neurons
- A neural network: a set of connected nodes where each node is an *input* (corresponding to an independent variable), *output* (corresponding to a dependent variable for numeric prediction or a class for classification; there may be several output nodes) or *hidden* and where each connection has a **weight** associated with it.
- During the learning phase, the **network learns by adjusting the weights** so as to be able to output the correct class label or predicted value for the input tuples.
- Also referred to as *connectionist* learning due to the connection between units
- The classic training algorithm is called *backpropagation.*
- *Convolutional Neural Networks* (CNNs) are neural networks where there are specific roles and connection architectures for hidden nodes and these have been applied very sucessfuly to image recognition and other problems.
- The term *deep learning* often refers to training CNNs. They are deep because there are many layers of hidden nodes.

Strength/Weakness of neural network as a classifier

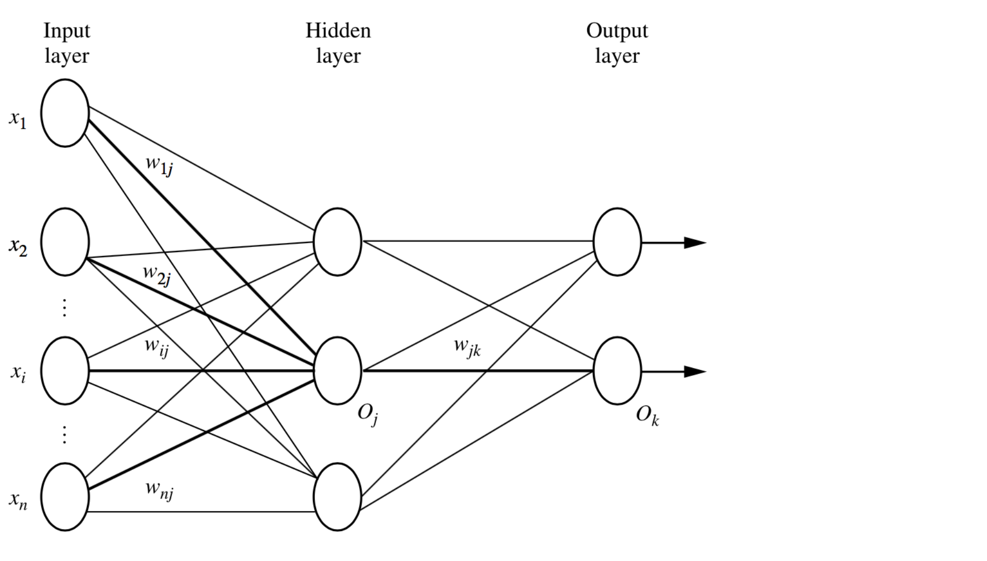
Strength

- High tolerance to noisy data
- Ability to classify untrained patterns
- Well-suited for continuous-valued inputs and outputs
- Successful on an array of real-world data, e.g., hand-written letters, images, voice
- Algorithms are inherently parallel
- Techniques have recently been developed for the extraction of rules from trained neural networks

Weakness

- *Long* training time
- Require a number of parameters typically best determined empirically, e.g., network topology or structure
- Poor interpretability: difficult to interpret the symbolic meaning behind the learned weights and of hidden units in network, that is, behaves as a "black box" model.

3.1. Multi-Layer Feed-Forward Neural Network



Multilayer feed-forward neural network.

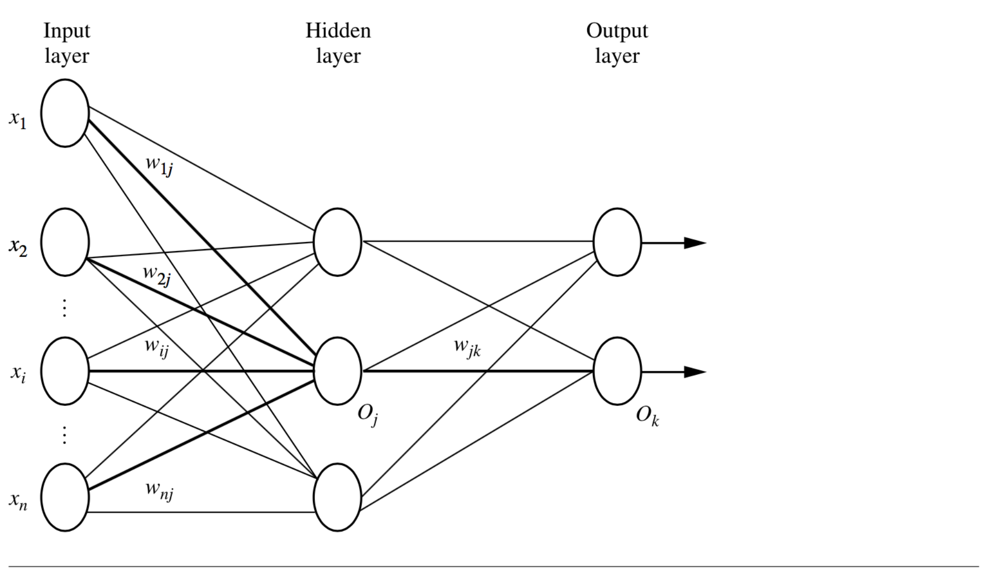
Structure of Multilayer neural network

- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
    - The number of hidden layers is arbitrary (1 hidden layer in the example above).
    - The number of hidden units is arbitrary (3 hidden units in the example above).
- The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction
- The network is **feed-forward**: none of the weights cycles back to an input unit or to an output unit of a previous layer
- From a statistical point of view, networks perform **nonlinear regression**: *given enough hidden units and enough training samples, they can closely approximate any function* (output)

Defining a network topology

- Decide the **network topology**:
    - Specify the number of units in the input layer: usually *one input unit per attribute* in the data (but nominal attributes can have one input per value).
    - the number of hidden layers (at least one, and commonly only one)
        - the number of units in each hidden layer
    - and the number of units in the output layer
- **Output**, one unit per response variable. In a typical binary classification problem only one output unit is used and a threshold is applied to the output value to select the class label. The value can be interpreted as a probability of belonging to the positive class, and in this way a neural network can be used as a *probablistic model*. For classification of more than two classes, one output unit per class is used and the highest-valued class is selected for the label.
- Choose an **activation function** for each hidden and output unit (explained later)
- Usually *randomly*, determine initial values for the weights.
- Once a network has been trained, if its accuracy is unacceptable, try a different network topology or a different set of initial weights or maybe different activation functions. Use trial-and-error or there are methods that systematically search for a high-performing topology.

3.2. Prediction with Neural Network



Multilayer feed-forward neural network.

We first discuss how to predict an output variable using a feed-forward neural network model *that has already been trained*.

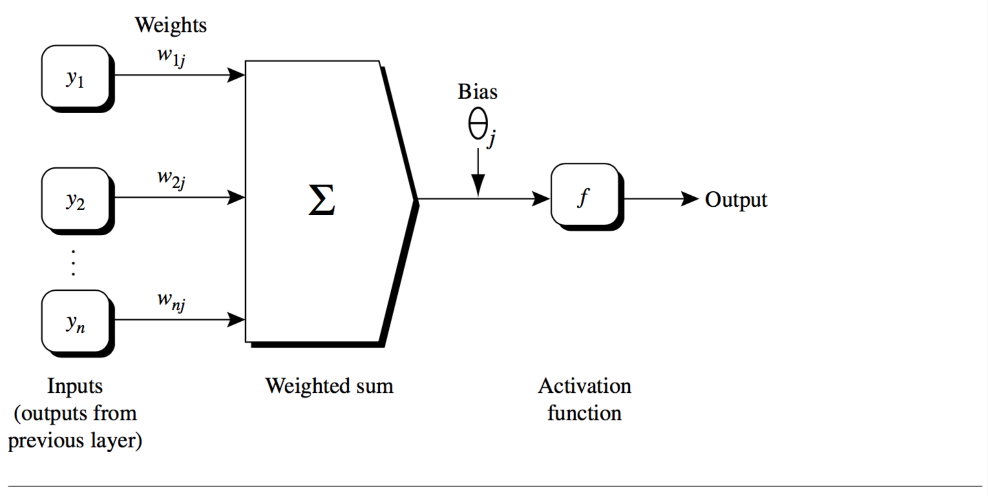
1. Initialise the weights by training

- The weights in the network are set to small random numbers (e.g., ranging from −1.0 to 1.0, or −0.5 to 0.5) by training.
- Each unit has a bias associated with it, as explained later. The biases are similarly set to small random numbers by training.

Each testing tuple, is first *normalised* to [0.0 ~ 1.0]. Consider a normalised tuple X, processed by the following steps.

2. Propagate the inputs forward

1. First, the testing tuple is *normalised* to [0.0 ~ 1.0] and the normalised tuple X is fed to the network's input layer. The inputs pass through the input units, unchanged.
2. Hidden layer
    - Input of hidden unit: all outputs of the previous layer, e.g. if the hidden unit is in the first hidden layer, then the inputs are $x_1, x_2, \ldots, x_n$.
    - Output of hidden unit j: weighted linear combination of its input followed by an activation function
        - $O_j = f(\sum_{i=1}^{n} w_{ij} \cdot x_i + \theta_j)$
        - where w_{ij} is the weight of the connection from input i in the previous layer to unit j
        - θ_j is the bias variable of unit j
        - $f(\cdot)$ is a non-linear activation function which will be described later.
    - If there is more than one hidden layer, the above procedure will be repeated until the final hidden layer will result outputs.
3. Output layer
    - The outputs of the final hidden layer will be used as inputs of the output layer.
    - The number of output units are determined by a task
        - If our goal is to predict a single numerical variable, then one output unit is enough
    - Final output O_k:
        - $O_k = \sum_{j=1}^{h} w_{jk} \cdot O_j + \theta_k$ or $O_k = f(\sum_{j=1}^{h} w_{jk} \cdot O_j + \theta_k)$
        - O_k is the final predicted value given $x_1, x_2, \ldots, x_n$.

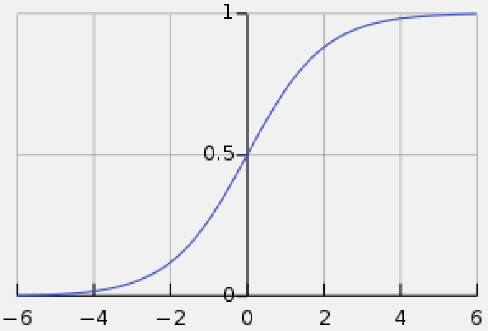


> Graphical illustration of the computational flow of hidden unit j: The inputs to unit j are outputs from the previous layer. These are multiplied by their corresponding weights to form a weighted sum, which is added to the bias θ_j associated with unit j. A nonlinear activation function f is applied to the net input. Again the output of this unit will be used as an input of the successive layer.

Non-linear activation function

An activation function imposes a certain non-linearity in a neural network. There are several possible choices of activation functions, but **sigmoid** (or logistic) function is the most widely used activation function.

- **Sigmoid function**
  - Sigmoid functions have domain of all real numbers, with return value monotonically increasing most often from 0 to 1.
  - $f(x) = \frac{1}{1+e^{-x}}$



  - Also referred to as a *squashing function*, because it maps the input domain onto the range of 0 to 1
- Other activation functions
  - Hyperbolic tangent function (tanh)
  - Softmax function (use for output nodes for classification problems to push the output value towards binary 0 or 1)
  - ReLU
  - etc.

3.3. Training a Neural Network

Backpropagation

The backpropagation algorithm, based on the **gradient descent** algorithm, is designed to *train the weights and biases of a neural network*.

How does backpropagation work?

For training, each training tuple,is first *normalised* to [0.0 ~ 1.0].

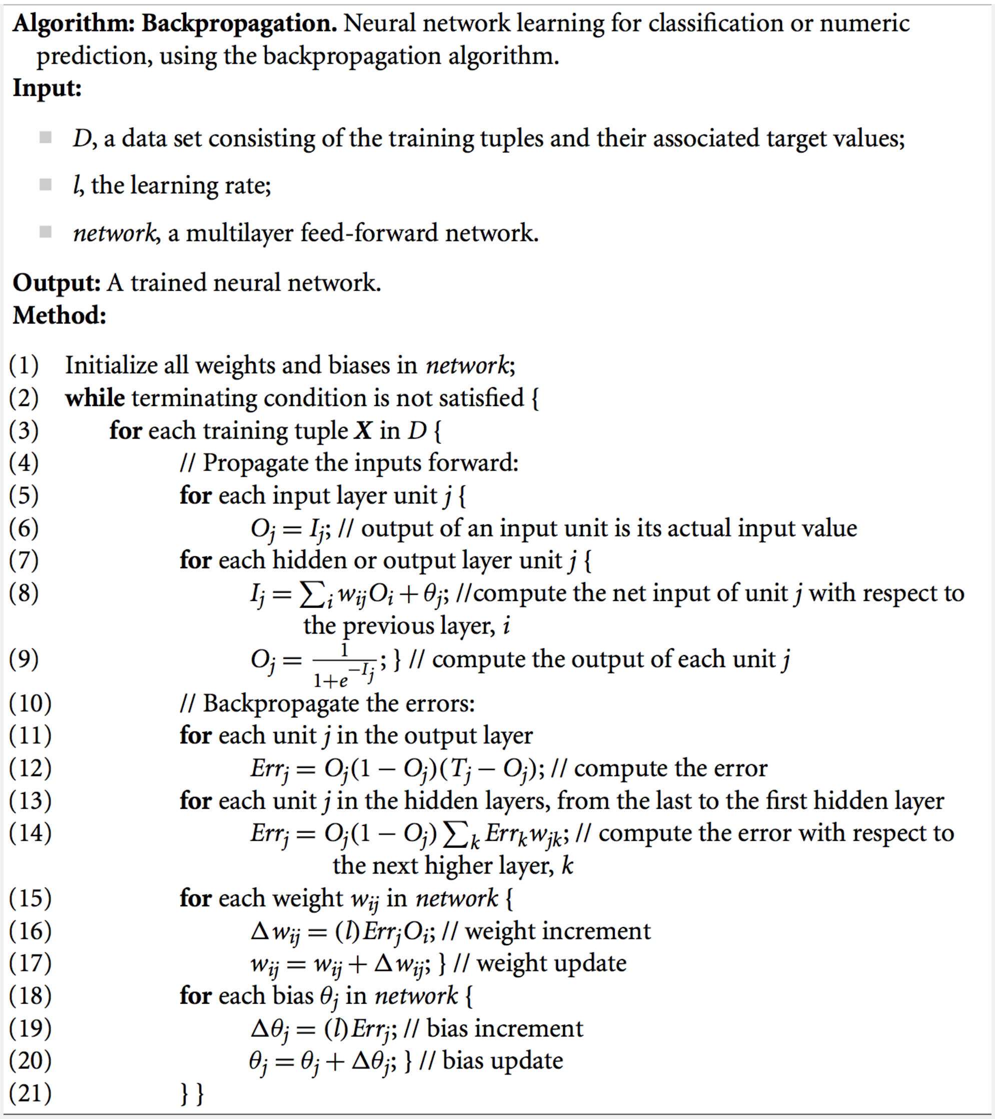
The error is **propagated backward** by **updating the weights and biases** to reflect the error of the network's prediction for the dependent variable for a training tuple.

- For unit k in the output layer, its error E_k is computed by
  - $E_k = O_k(1 - O_k)(T_k - O_k)$,
  - where O_k is the actual output of unit k, and T_k is the known target value of the given training tuple.
- To compute the error of a hidden layer unit j, the weighted sum of the errors of the units connected to unit k in the next layer are considered.
  - $E_j = O_j(1 - O_j) \sum_k E_k w_{jk}$,
  - where w_{jk} is the weight of the connection from unit j to a unit k in the next higher layer.
- The weights and biases are **updated to reflect the propagated errors**.
  - Each weight w_{jk} in the network is updated by the following equations, where Δw_{jk} is the change in weight w_{jk}:
    - $\Delta w_{jk} = \ell \times E_k O_j$
    - $w_{jk} = w_{jk} + \Delta w_{jk}$
  - Each bias θ_j in the network is updated by the following equations, where $\Delta\theta_j$ is the change in bias θ_j:
    - $\Delta\theta_j = \ell \times E_j$
    - $\theta_j = \theta_j + \Delta\theta_j$

Learning rate: The parameter ℓ is the learning rate, typically, a value between 0.0 and 1.0. If the learning rate is too small, then learning (ie reduction in the Error) will occur at a very slow pace. If the learning rate is too large, then oscillation between inadequate solutions may occur. A rule of thumb is to set the learning rate to $1/t$, where t is the number of iterations through the training set so far. In principle the network should *converge* towards a minimal Error, but if this is not happening, try reducing this parameter value and retrain.

Updating schedules

- **Case updating**: The backpropagation algorithm given here updates the weights and biases immediately after the presentation of each tuple.
- **Epoch updating**: Alternatively, the weight and bias increments could be accumulated in variables, so that the weights and biases are updated after all the tuples in the training set have been presented.
- Either way, we keep updating until we have a satisfactory error; presenting each tuple of the training set many times over
- In theory, the mathematical derivation of backpropagation employs epoch updating, yet in practice, case updating is more common because it tends to yield more accurate results.



Algorithm: Backpropagation. Neural network learning for classification or numeric prediction, using the backpropagation algorithm.

Input:

- *D*, a data set consisting of the training tuples and their associated target values;
- *l*, the learning rate;
- *network*, a multilayer feed-forward network.

Output: A trained neural network.

Method:

```
Initialize all weights and biases in network;
while terminating condition is not satisfied {
    for each training tuple X in D {
        // Propagate the inputs forward:
        for each input layer unit j {
            $O_j = I_j$; // output of an input unit is its actual input value
        for each hidden or output layer unit j {
            $I_j = \sum_i w_{ij} O_i + \theta_j$; //compute the net input of unit j with respect to
                the previous layer, i
            $O_j = \frac{1}{1+e^{-I_j}}$; } // compute the output of each unit j
        // Backpropagate the errors:
        for each unit j in the output layer
            $Err_j = O_j(1 - O_j)(T_j - O_j)$; // compute the error
        for each unit j in the hidden layers, from the last to the first hidden layer
            $Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$; // compute the error with respect to
                the next higher layer, k
        for each weight $w_{ij}$ in network {
            $\Delta w_{ij} = (l) Err_j O_i$; // weight increment
            $w_{ij} = w_{ij} + \Delta w_{ij}$; } // weight update
        for each bias $\theta_j$ in network {
            $\Delta \theta_j = (l) Err_j$; // bias increment
            $\theta_j = \theta_j + \Delta \theta_j$; } // bias update
        } }
```

> Backpropagation algorithm from the textbook

How do you decide to terminate training?

- When your weights and biases are changing very little , ie. all the Δs are small; or
- Accuracy on the training set is good enough; or
- A prespecified number of epochs have passed.

3.4. Performance Evaluation

Objective performance evaluation

For numerical prediction tasks, the methods used for evaluating linear regression may also be used for evaluating neural nets.

For classification tasks, any of the methods used for evaluating classification tasks may also be used for neural nets.

In addition, as a neural net can be used as a probablistic classifier, methods for probablistic classifiers, such as ROC charts may be used.

Interpretability of neural network

"Neural networks are like a black box. How can I 'understand' what the backpropagation network has learned?"

- A major disadvantage of neural networks lies in their knowledge representation.
- Acquired knowledge in the form of a network of units connected by weighted links is difficult for humans to interpret.
- Easier-to-interpret rules may be extracted from the network by pruning or by sensitivity analysis.

Rule extraction by network pruning for interpretability

- Simplify the network structure by removing weighted links that have the least effect on the trained network
- The set of input and activation values are studied to derive rules describing the relationship between the input and hidden unit layers

Sensitivity analysis for interpretability

- Assess the impact that a given input variable has on a network output.
- The input to the variable is varied while the remaining input variables are fixed at some value.
- The knowledge gained from this analysis can be represented in rules
  - Such as "IF X decreases 5% THEN Y increases 8%."

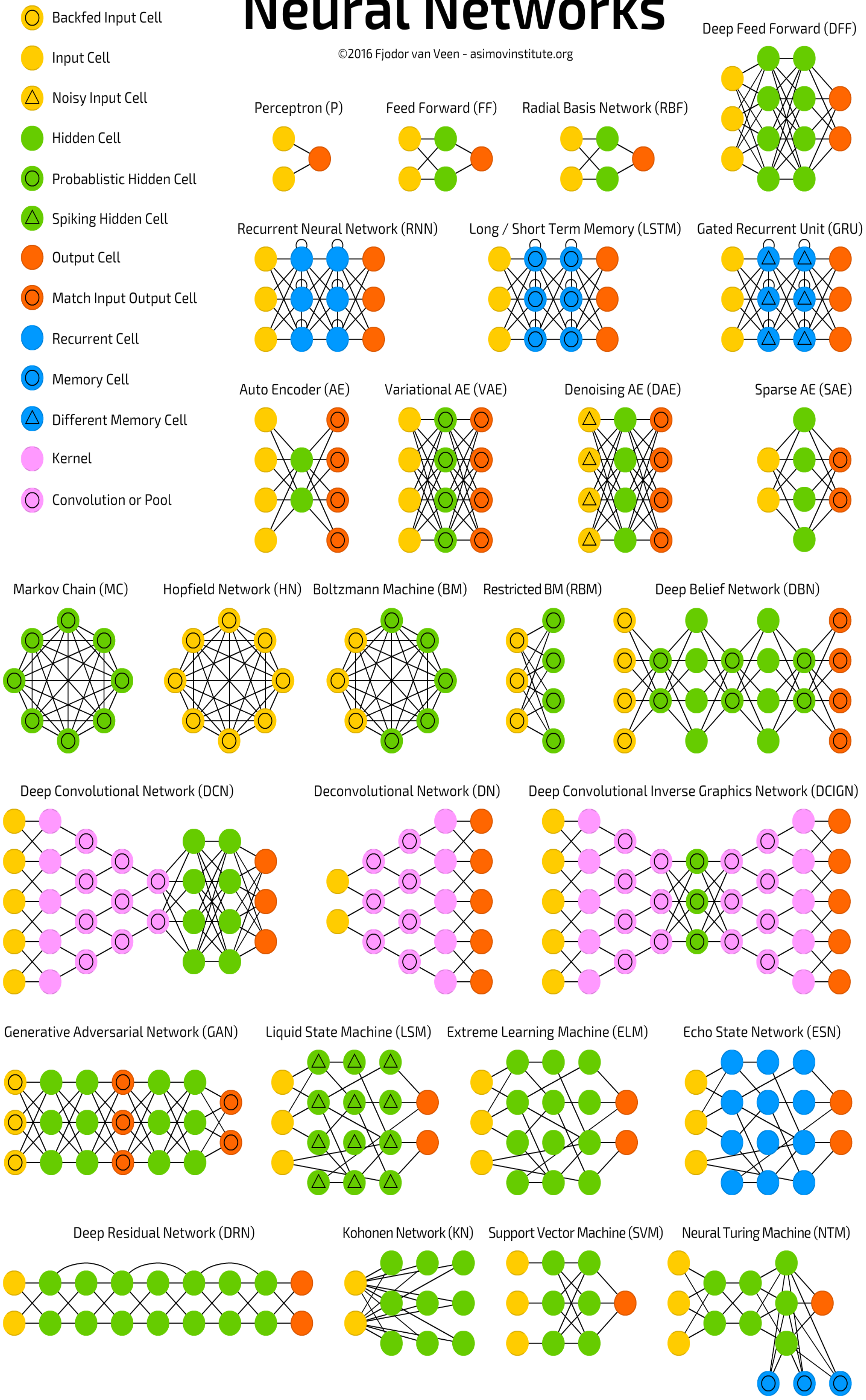
3.5. Other Neural Net Architectures (Not in text)

There are many adaptations of the basic Neural Network architecture presented here.

Here is a convenient graphical summary from http://www.asimovinstitute.org/neural-network-zoo/

A mostly complete chart of

Neural Networks



- ◎ Backfed Input Cell
- ● Input Cell
- △ Noisy Input Cell
- ● Hidden Cell
- ◎ Probablistic Hidden Cell
- △ Spiking Hidden Cell
- ● Output Cell
- ◎ Match Input Output Cell
- ● Recurrent Cell
- ◎ Memory Cell
- △ Different Memory Cell
- ● Kernel
- ◎ Convolution or Pool

Deep Feed Forward (DFF)

Perceptron (P)

Feed Forward (FF)

Radial Basis Network (RBF)

Recurrent Neural Network (RNN)

Long / Short Term Memory (LSTM)

Gated Recurrent Unit (GRU)

Auto Encoder (AE)

Variational AE (VAE)

Denoising AE (DAE)

Sparse AE (SAE)

Markov Chain (MC)

Hopfield Network (HN)

Boltzmann Machine (BM)

Restricted BM (RBM)

Deep Belief Network (DBN)

Deep Convolutional Network (DCN)

Deconvolutional Network (DN)

Deep Convolutional Inverse Graphics Network (DCIGN)

Generative Adversarial Network (GAN)

Liquid State Machine (LSM)

Extreme Learning Machine (ELM)

Echo State Network (ESN)

Deep Residual Network (DRN)

Kohonen Network (KN)

Support Vector Machine (SVM)

Neural Turing Machine (NTM)

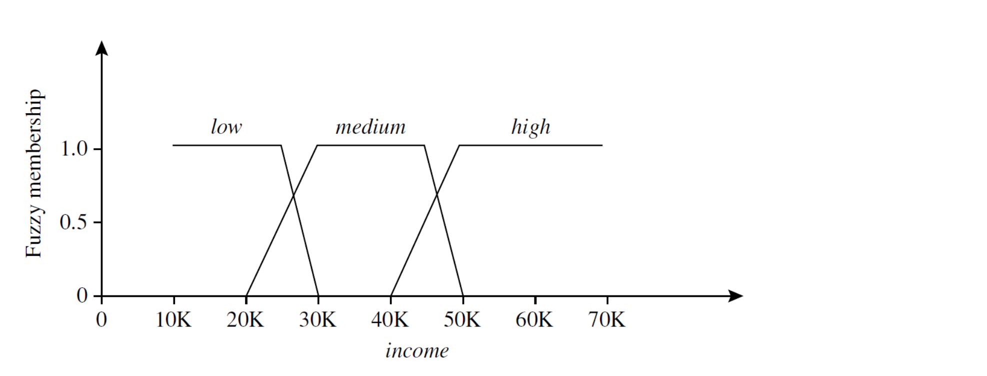
4. Other "Soft" Classification Algorithms (Text 9.6)

Genetic Algorithm

- Genetic Algorithm: based on an analogy to biological evolution
- An initial population P is created consisting of randomly generated rules
  - Each rule is represented by a string of bits
  - E.g., if A1 and ¬A2 then C2 can be encoded as 100
  - If an attribute has k > 2 values, k bits can be used
- Based on the notion of survival of the fittest, a new population is formed to consist of the fittest rules and their offspring
  - The fitness of a rule is represented by its classification accuracy on a set of training examples
- Offspring are generated by crossover and mutation
  - **Crossover**: sub-strings from pairs of rules are swapped to form new pairs of rules
  - **Mutation**: randomly selected bits in a rule's string are inverted
- The process continues until a population P evolves when each rule in P satisfies a prespecified threshold
- Slow but easily parallelisable

Fuzzy-Set Approach

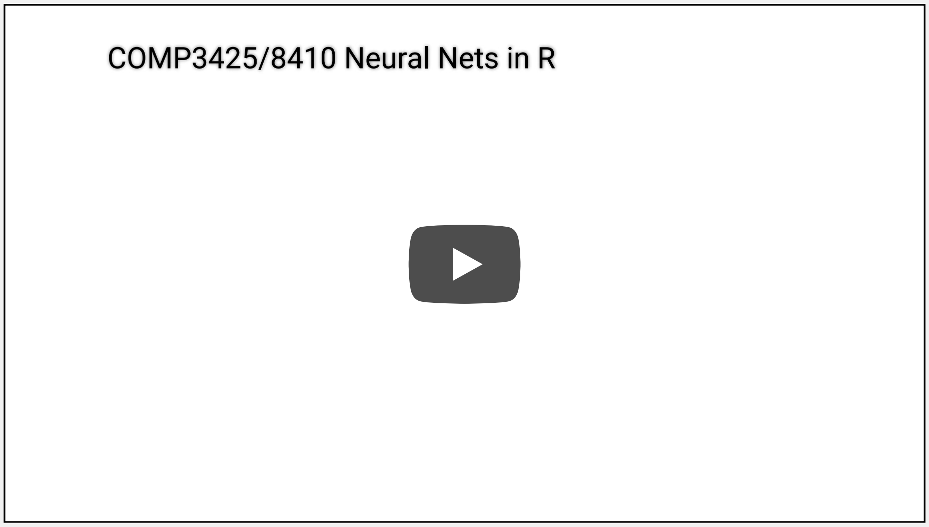
- Fuzzy logic uses truth values between 0.0 and 1.0 to represent the degree of membership (such as in a fuzzy membership graph)
- Attribute values are converted to fuzzy values:
  - Income is assigned a fuzzy membership value to each of the discrete categories {low, medium, high}, without fuzzy logic $49K income will be categorized as "medium income"
  - In fuzzy logic:
    - $49K belongs to “medium income” with fuzzy value 0.15
    - but belongs to “high income” with fuzzy value 0.96



  - Fuzzy membership values do not have to sum to 1.
- Each applicable rule contributes a vote for membership in the categories
- Typically, the truth values for each predicted category are summed, and these sums are combined

5. Practical Exercises: Neural Network and Linear Regression in R

ACTION: Attempt these practical exercises with Rattle. There is a video showing the mechanics to get you started, and written instructions for you to work through.



COMP3425/8410 Neural Nets in R

Neural Networks and Linear Regression in R