LONDON METROPOLITAN UNIVERSITY

Machine Learning
CS7052

Lecture 8, Neural Networks & Deep learning

Dr. Elaheh Homayounvala

Week 8

Outline of today's lecture

- Summary of last two weeks
  - Ethical, social and legal issues
  - Decision trees
- Neural networks, Deep learning
- Student's survey, Closing the loop

LONDON METROPOLITAN UNIVERSITY

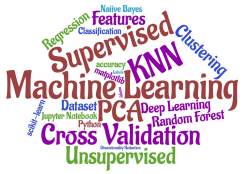
Review last weeks

Legal, Social and Ethical issues

Decision Trees

Legal, Social and Ethical issues

- Let’s look at the discussion group in week 6 folder



Decision Trees

What is a decision tree?

How to build a decision tree? Attribute selection measures

Information Gain, Gain Ratio and Gini index

Parameters, Strengths and Weaknesses

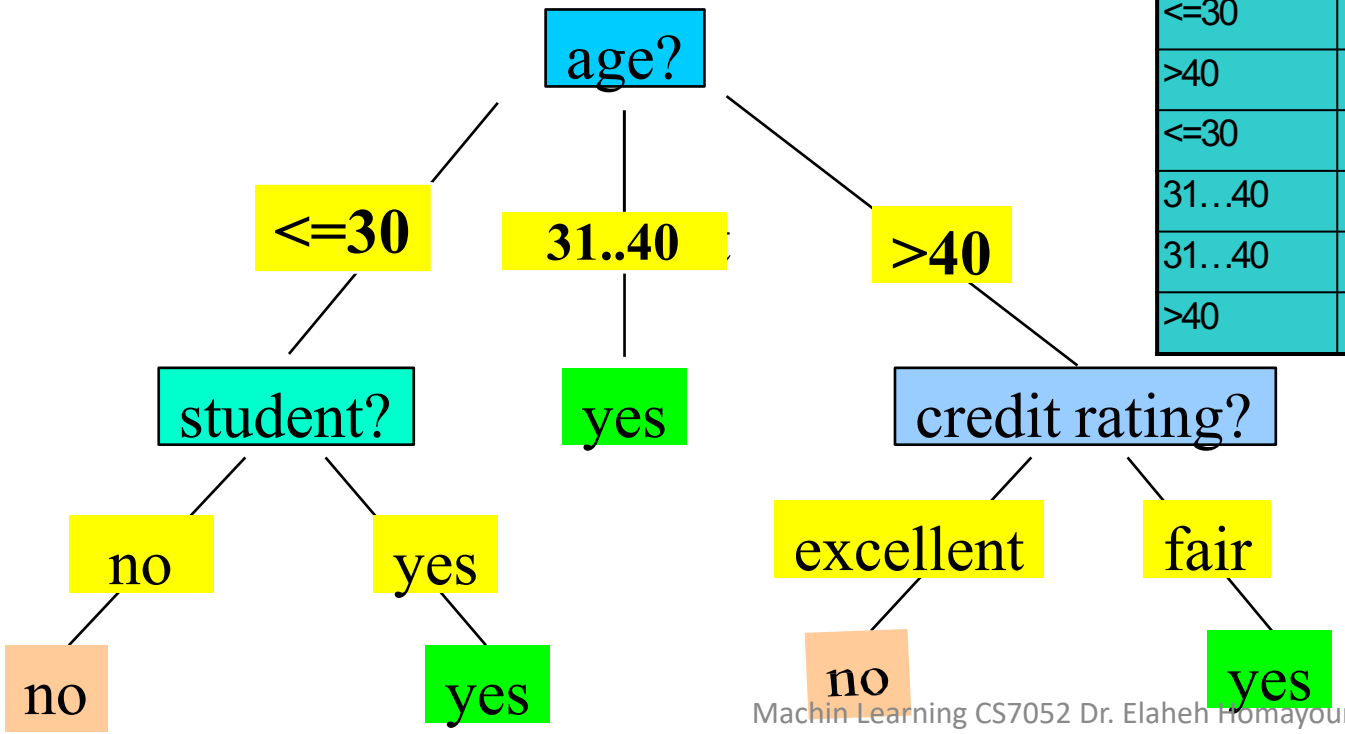
Decision Trees

- Widely used models for classification and regression tasks
- They learn a hierarchy of if/else questions, leading to a decision

Decision Tree, An Example

- Training dataset: Buys_computer
- Decision Tree:

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no



Learning a Decision Tree

- To build a tree, the algorithm searches over all possible tests and finds the one that is most informative about the target variable.
- Continue and choose the next test recursively.
- The recursive partitioning of the data is repeated until
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning majority voting is employed for classifying the leaf
  - There are no samples left

Attribute selection methods

- **Information gain**: Information gained by branching on attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

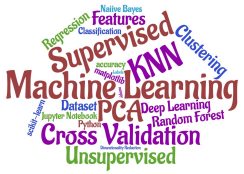
$$Info(D) = -\sum p_i \log_2(p_i) \qquad Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

- **Gain ratio**: GainRatio(A) = Gain(A)/SplitInfo(A)

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

- **Gini index**: $\Delta gini(A) = gini(D) - gini_A(D)$

$$gini(D) = 1 - \sum_{j=1} p_j^2 \qquad gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$



Neural Networks & Deep learning

What is neural network?

What is Black-box approach?

Parameters, Strengths and Weaknesses

Input-output models

$$X \rightarrow \text{B} \rightarrow Y$$

- *Explanatory variables* → B → *response*

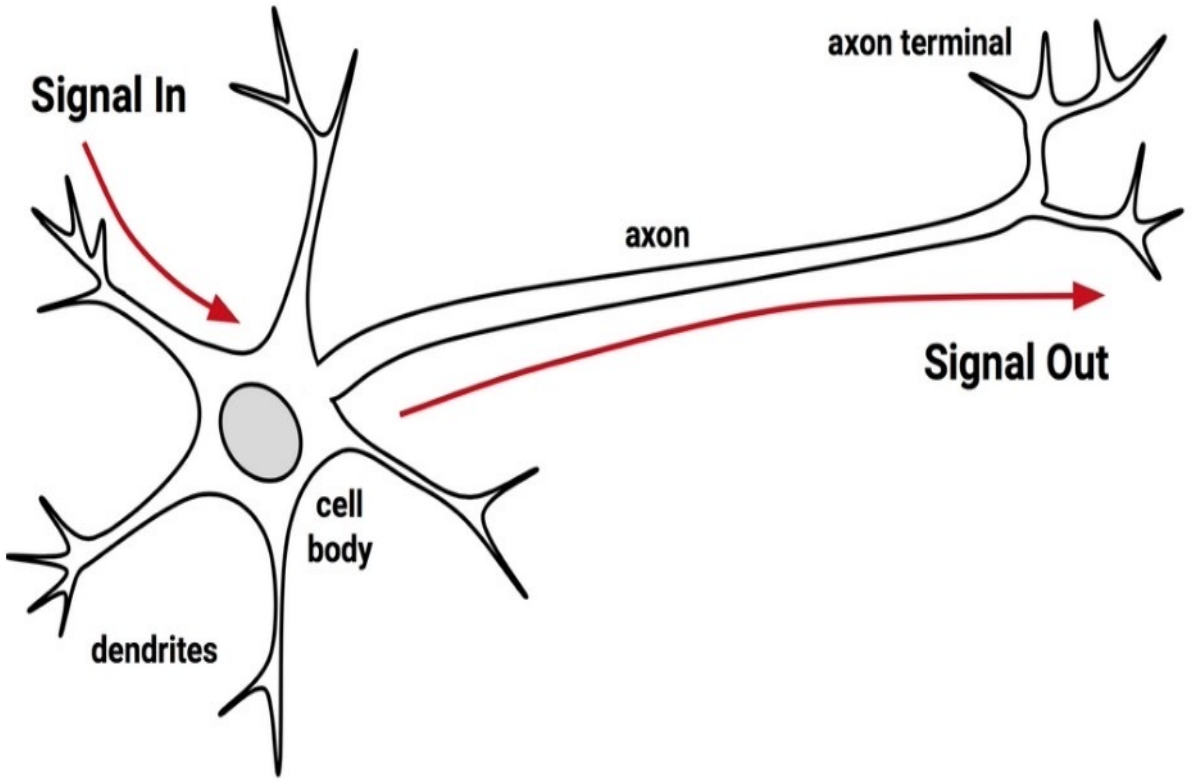
- Difficult to interpret the process going from *X* to *Y* hence called **black-box methods**
- Regression and decision trees are easy to explain to people
- Neural networks and support vector machines are not, but can be good in predicting

LONDON METROPOLITAN UNIVERSITY

Artificial Neural Networks

- Trying to mimic the brain
- Inter-connected cells: neurons
- The human brain has 85 billions
- Artificial neurons or nodes
- Turing tests: A machine is intelligent if a human being cannot distinguish its behaviour from a living creatures

Natural Neural Network



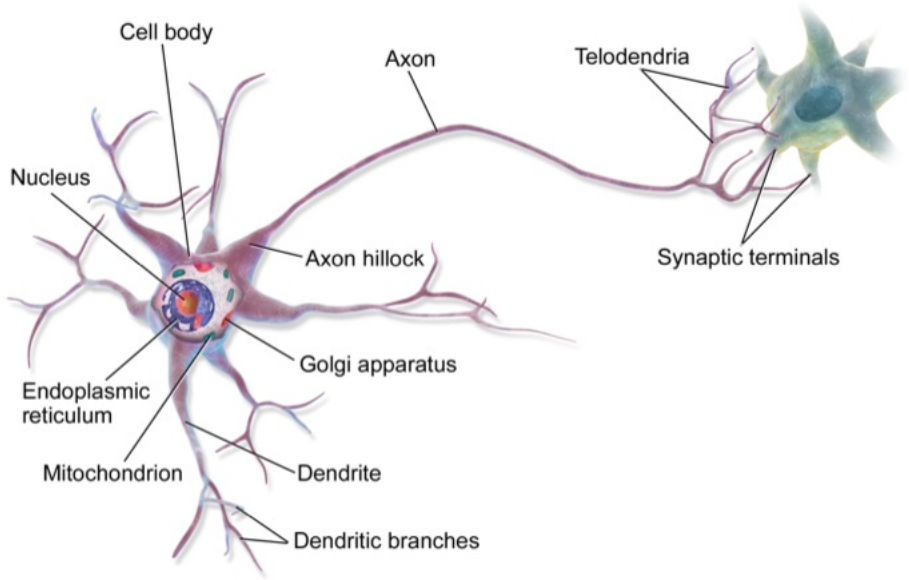
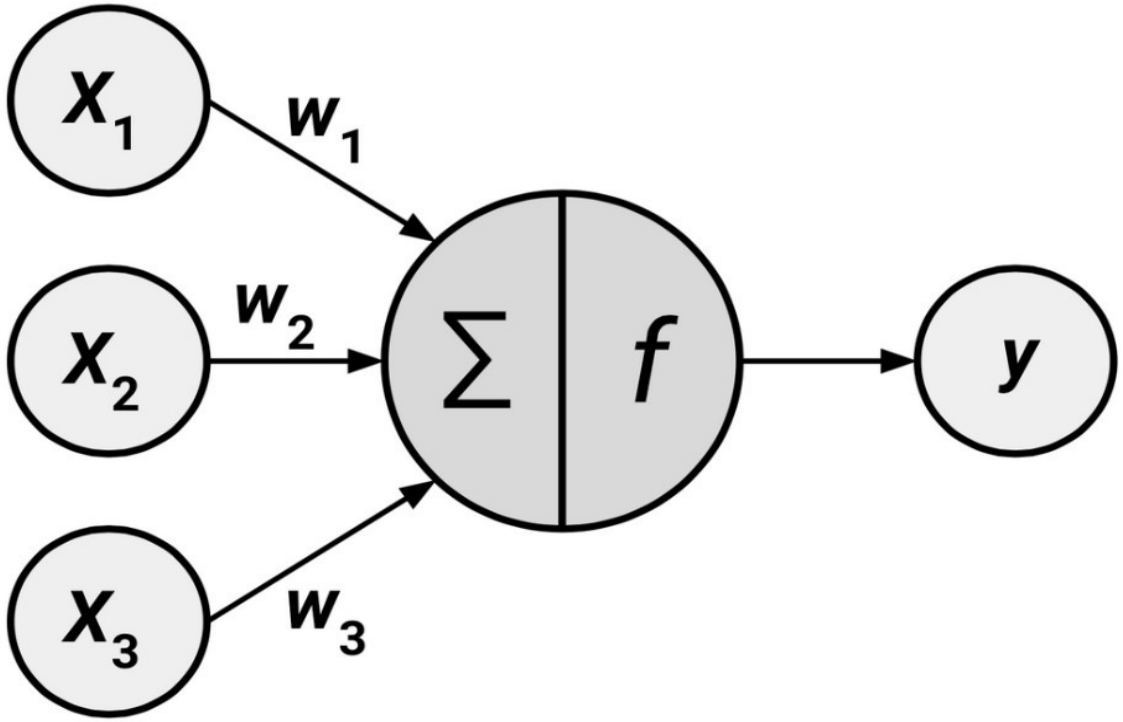


Figure 8.2: An illustration of a biological neuron. (Image credit: BruceBlaus/ CC-BY-3.0.)

Source: chapter 8, Jiang's book

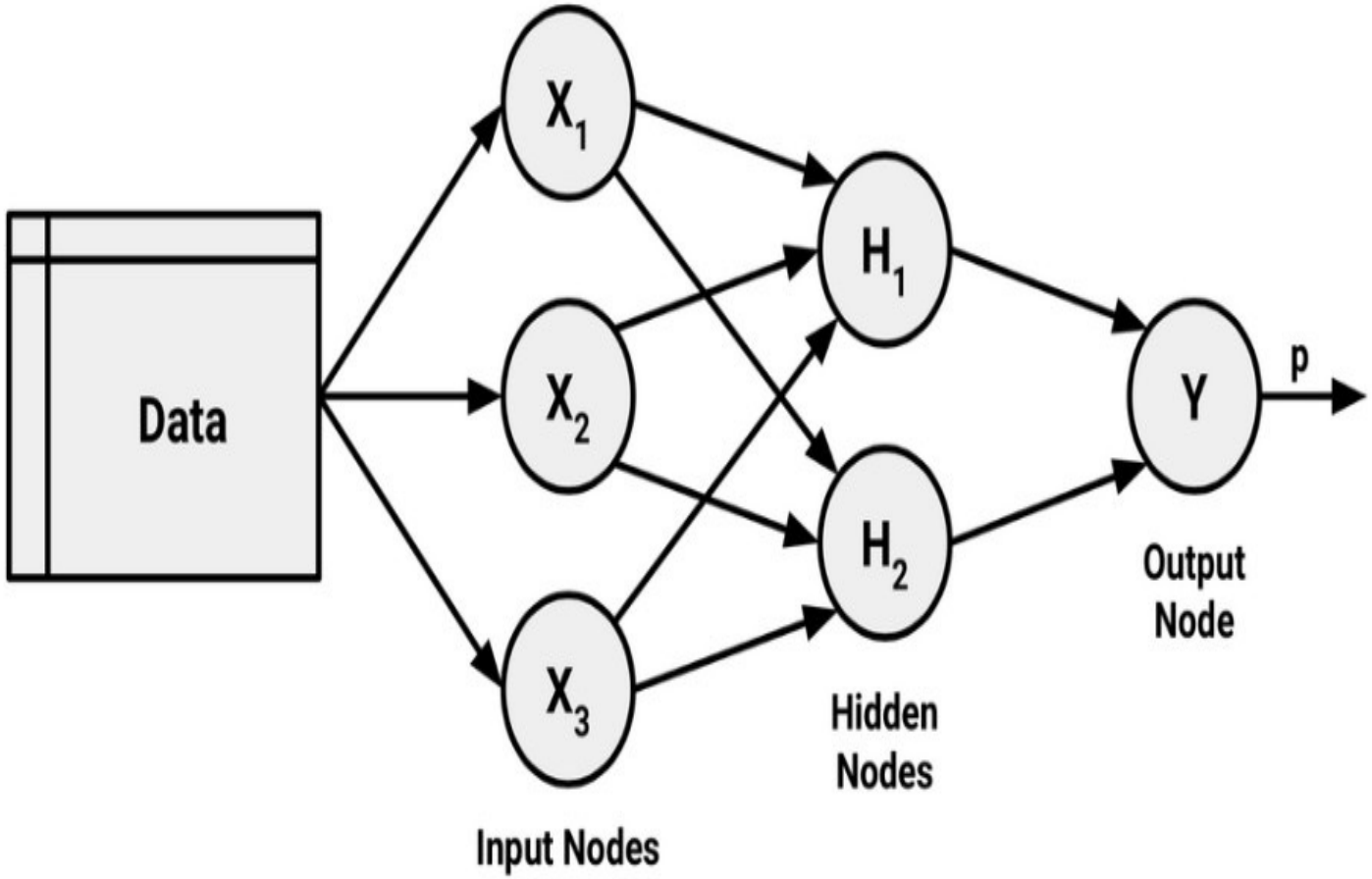
Artificial Neural Network



Artificial Neural Network

- *r* input variables
- w_i for $i = 1, \ldots, r$ called the weights but really are the coefficients
- $y(x) = f\{\sum_{i=1}^{r} w_i x_i\}$
- Activation function
- Network topology
- Training algorithm: fitting the model that is, estimating its parameters

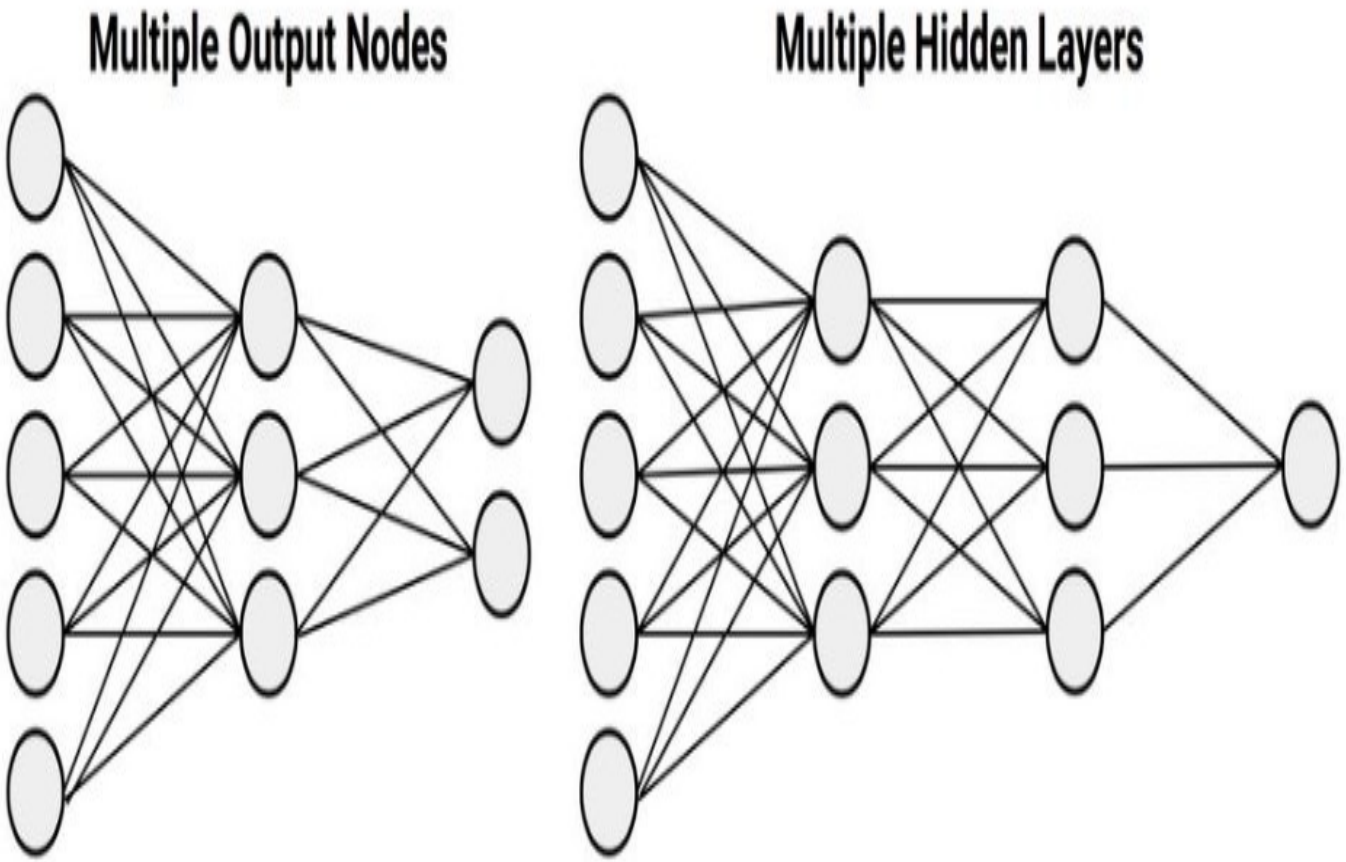
Artificial Neural Network



Artificial Neural Network (ANN / NN)

- The target more than one (multivariate response)
- Can be categorical variable (classification)
- We can have multiple hidden layers (deep learning)

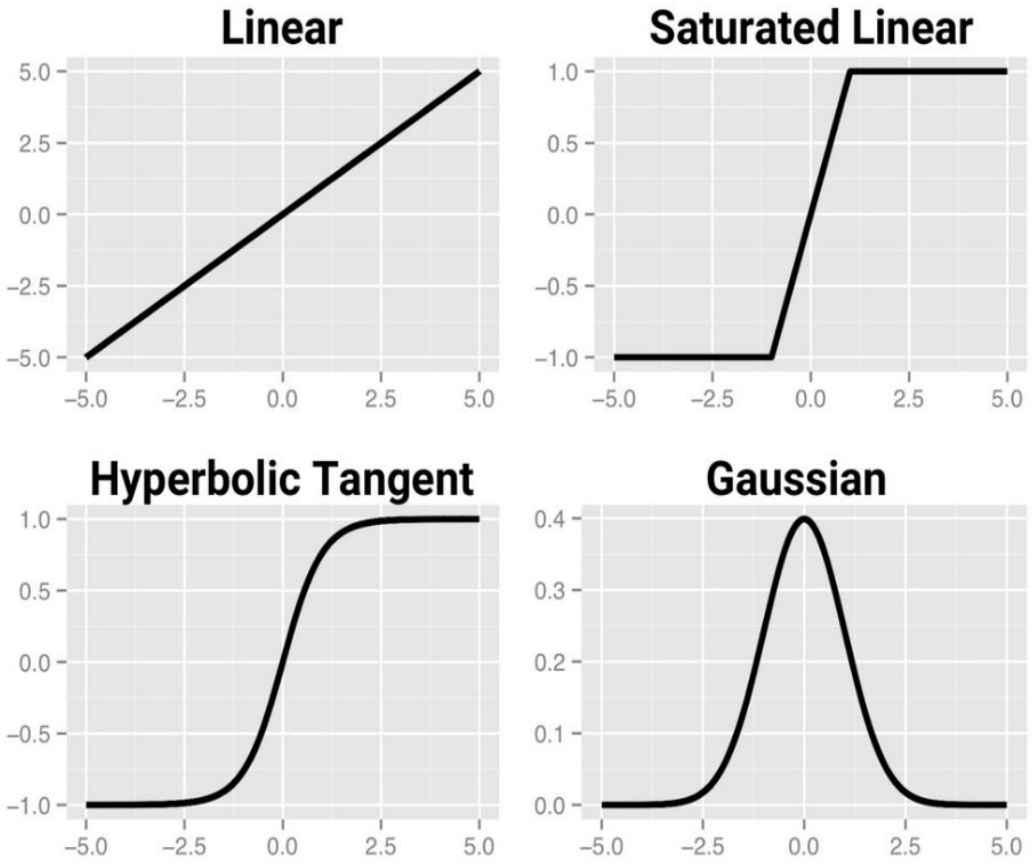
Multiple NN



Activation function

- Activation function is the mechanism of how NN process incoming information and pass it to to the network
- Threshold function (step function and therefore not differentiable)
- Sigmoid activation function (differentiable)
- Other activation functions:
  - Linear
  - Saturated linear
  - Hyperbolic Tangent
  - Gaussian

Activation function



Multi Layer Perceptron (MLP)

- Are also known as (vanilla) feed-forward neural networks
- or sometimes just neural networks.

MLP as generalisation of Linear Models

- Linear models:

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \ldots + w[p] * x[p] + b$$

- MLPs can be viewed as generalizations of linear models that perform multiple stages of processing to come to a decision

Visualisation of a linear model

- Node on the left: an input feature
- Connecting lines: the learned coefficients
- Node on the right: the output, which is a weighted sum of the inputs.

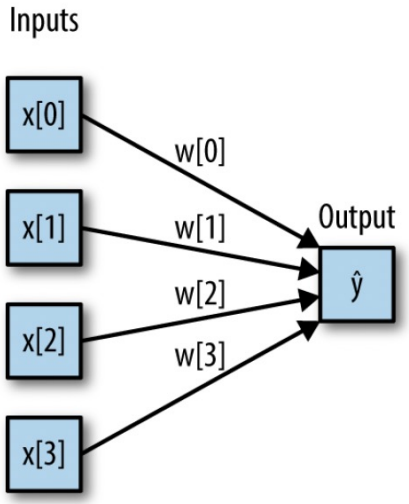


Figure 2-44. Visualization of logistic regression, where input features and predictions are shown as nodes, and the coefficients are connections between the nodes

MLP

- The process of computing weighted sums is repeated multiple times
- First computing hidden units that represent an intermediate processing step,
- Then again combined using weighted sums to yield the final result

- Muller and Guido's book, page 108

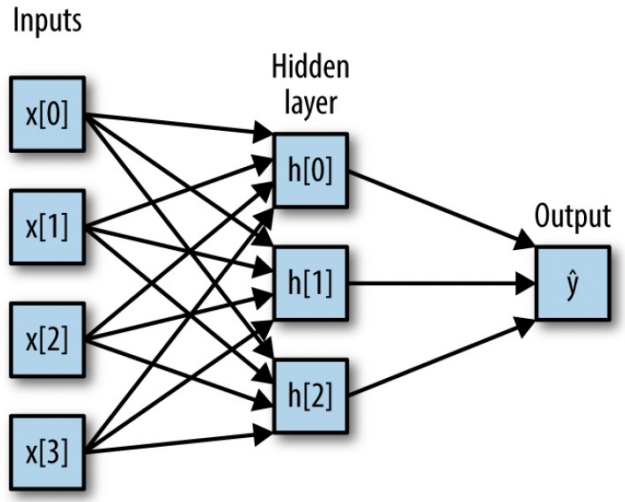


Figure 2-45. Illustration of a multilayer perceptron with a single hidden layer

$$h[0] = w[0,0] * x[0] + w[1,0] * x[1] + w[2,0] * x[2] + w[3,0] * x[3] + b[0]$$
$$h[1] = w[0,1] * x[0] + w[1,1] * x[1] + w[2,1] * x[2] + w[3,1] * x[3] + b[1]$$
$$h[2] = w[0,2] * x[0] + w[1,2] * x[1] + w[2,2] * x[2] + w[3,2] * x[3] + b[2]$$

$$\hat{y} = v[0] * h[0] + v[1] * h[1] + v[2] * h[2] + b$$

MLP, computing weights

In the previous slide

- w are the weights between the input x and the hidden layer h
- v are the weights between the hidden layer h and the output ŷ.
- The weights v and w are learned
- from data x are the input features
- ŷ is the computed output,
- and h are intermediate computations.
- An important parameter that needs to be set by the user is the number of nodes in the hidden layer.
- This can be as small as 10 for very small or simple datasets and as big as 10,000 for very complex data.

Multi-layer Perceptron

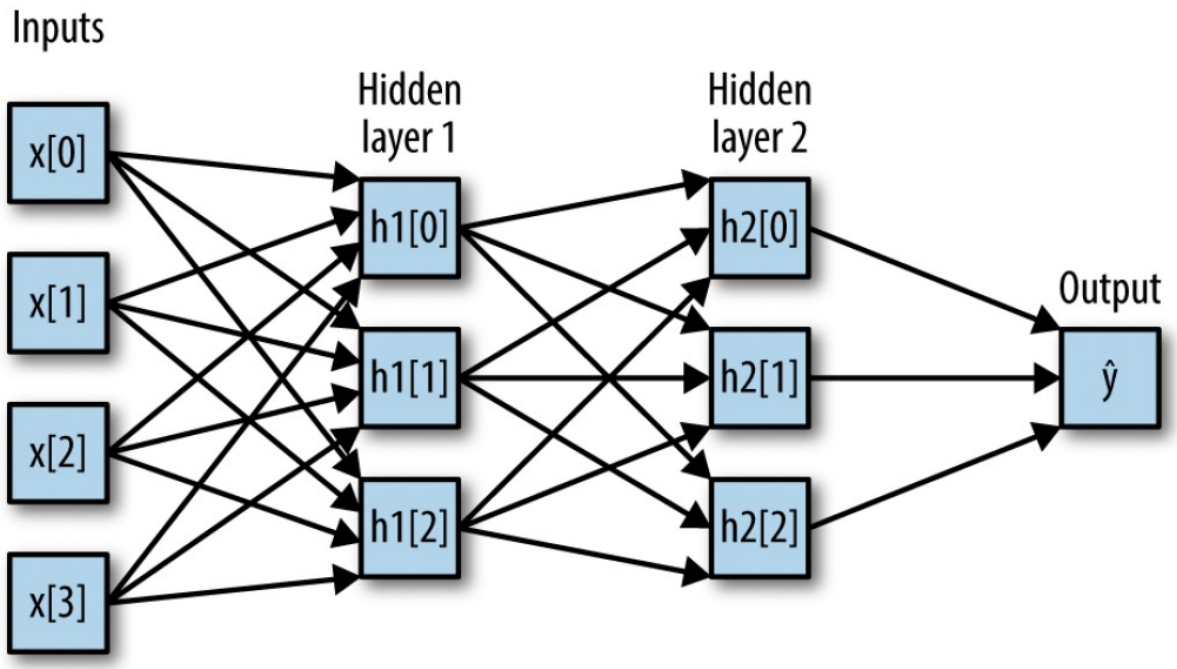


Figure 2-47. A multilayer perceptron with two hidden layers

Muller and Guido's book, page 110

Activation function

- Computing a series of weighted sums is mathematically the same as computing just one weighted sum,
- to make this model more powerful than a linear model, we need one extra trick.
- After computing a weighted sum for each hidden unit, a nonlinear function is applied to the result
- Usually the rectifying nonlinearity (also known as **rectified linear unit** or **relu**) or the **tangens hyperbolicus** (**tanh**).
- The result of this function is then used in the weighted sum that computes the output, ŷ.

relu vs tanh

- The relu cuts off values below zero
- Tanh saturates to –1 for low input values and +1 for high input values.
- Either nonlinear function allows the neural network to learn much more complicated functions than a linear model could

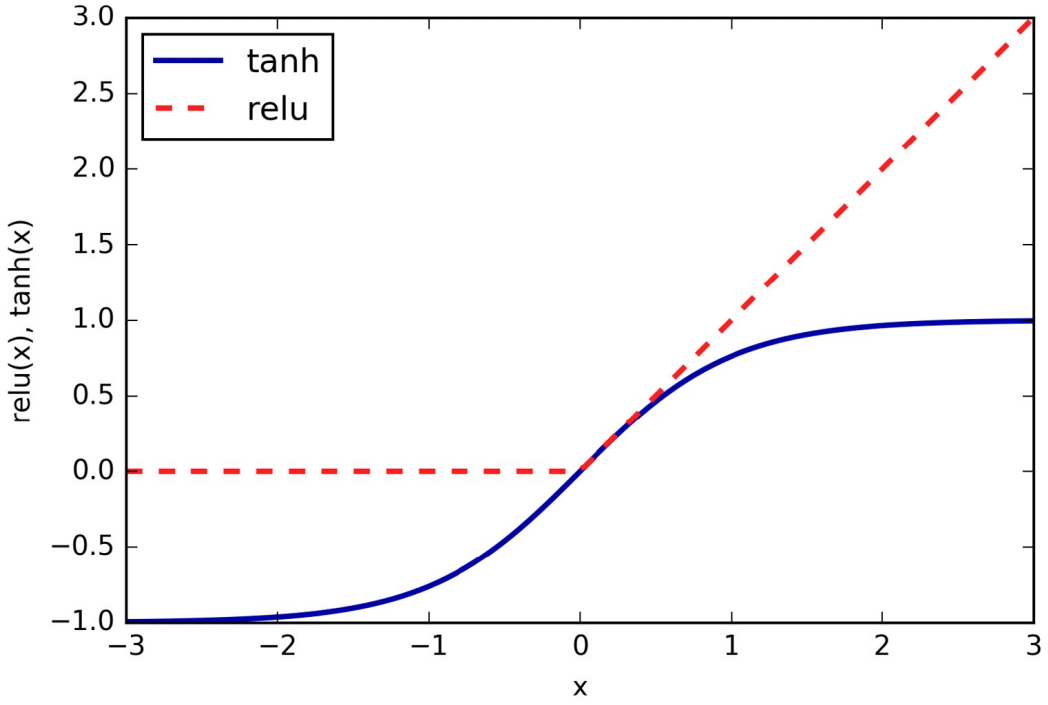


Figure 2-46. The hyperbolic tangent activation function and the rectified linear activation function

Muller and Guido's book, page 109

Computing output

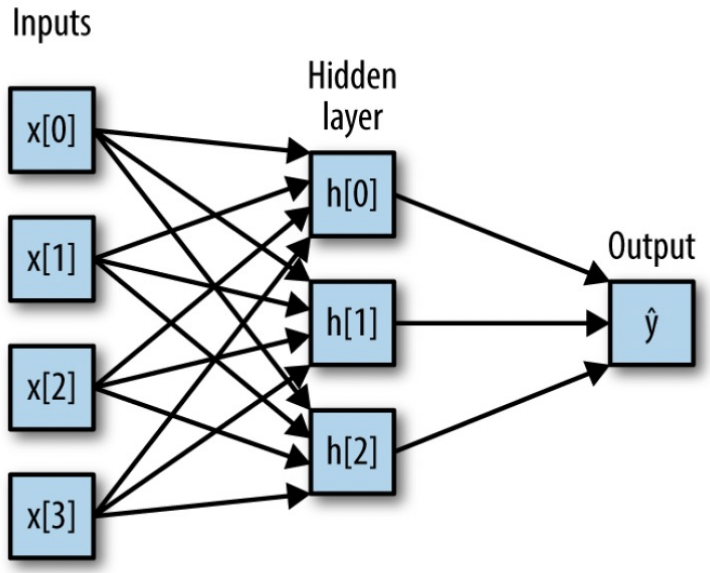
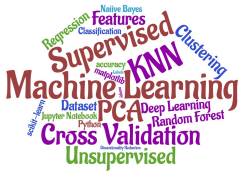


Figure 2-45. Illustration of a multilayer perceptron with a single hidden layer

$$h[0] = \tanh(w[0, 0] * x[0] + w[1, 0] * x[1] + w[2, 0] * x[2] + w[3, 0] * x[3] + b[0])$$
$$h[1] = \tanh(w[0, 1] * x[0] + w[1, 1] * x[1] + w[2, 1] * x[2] + w[3, 1] * x[3] + b[1])$$
$$h[2] = \tanh(w[0, 2] * x[0] + w[1, 2] * x[1] + w[2, 2] * x[2] + w[3, 2] * x[3] + b[2])$$
$$\hat{y} = v[0] * h[0] + v[1] * h[1] + v[2] * h[2] + b$$

LONDON METROPOLITAN UNIVERSITY



Students' Survey

Closing the Loop

Tuning Neural Networks

- Number of hidden layers
- Number of units/neurons in each layer
- Activation function (relu, tanh, …)
- Regularisation (L1, L2)

Tuning a neural network, example

- Muller and Guido's book, page 115

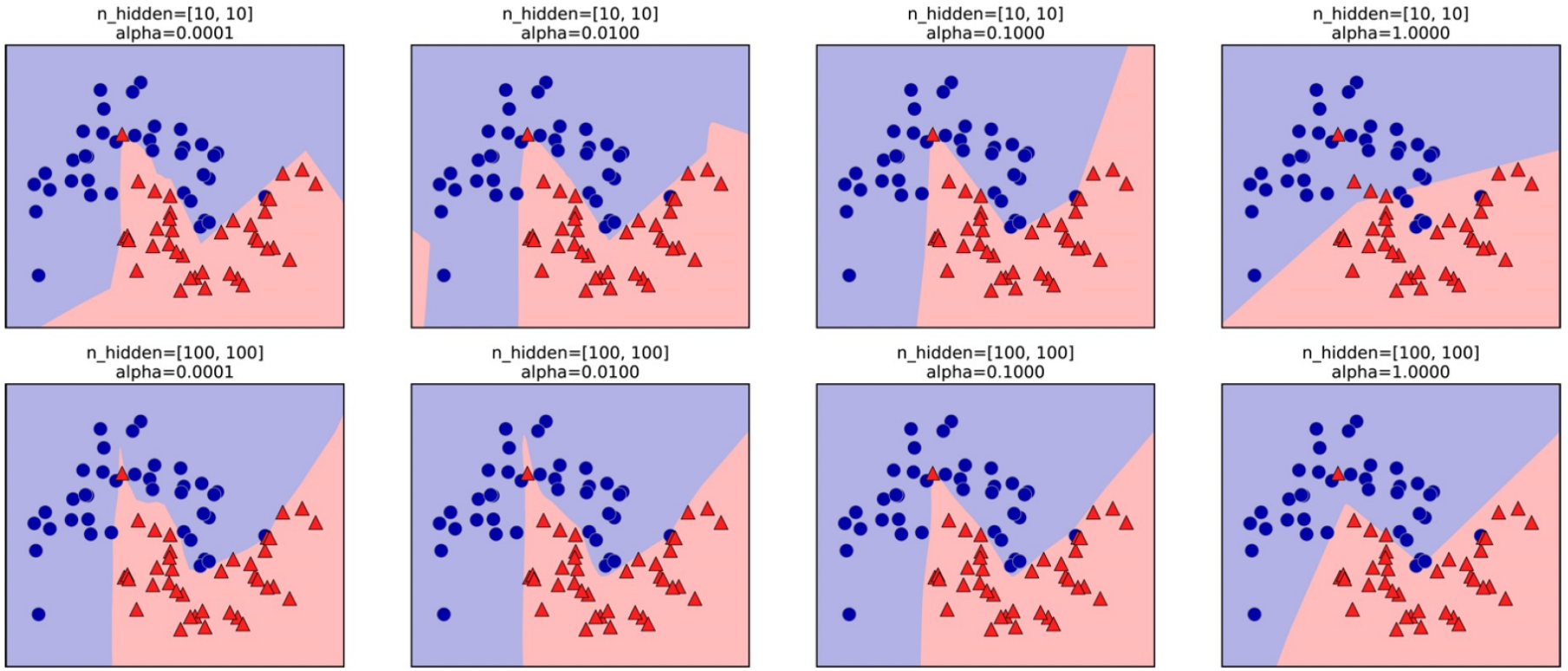


Figure 2-52. Decision functions for different numbers of hidden units and different settings of the alpha parameter

Tuning a neural network, example

- Muller and Guido's book, page 116

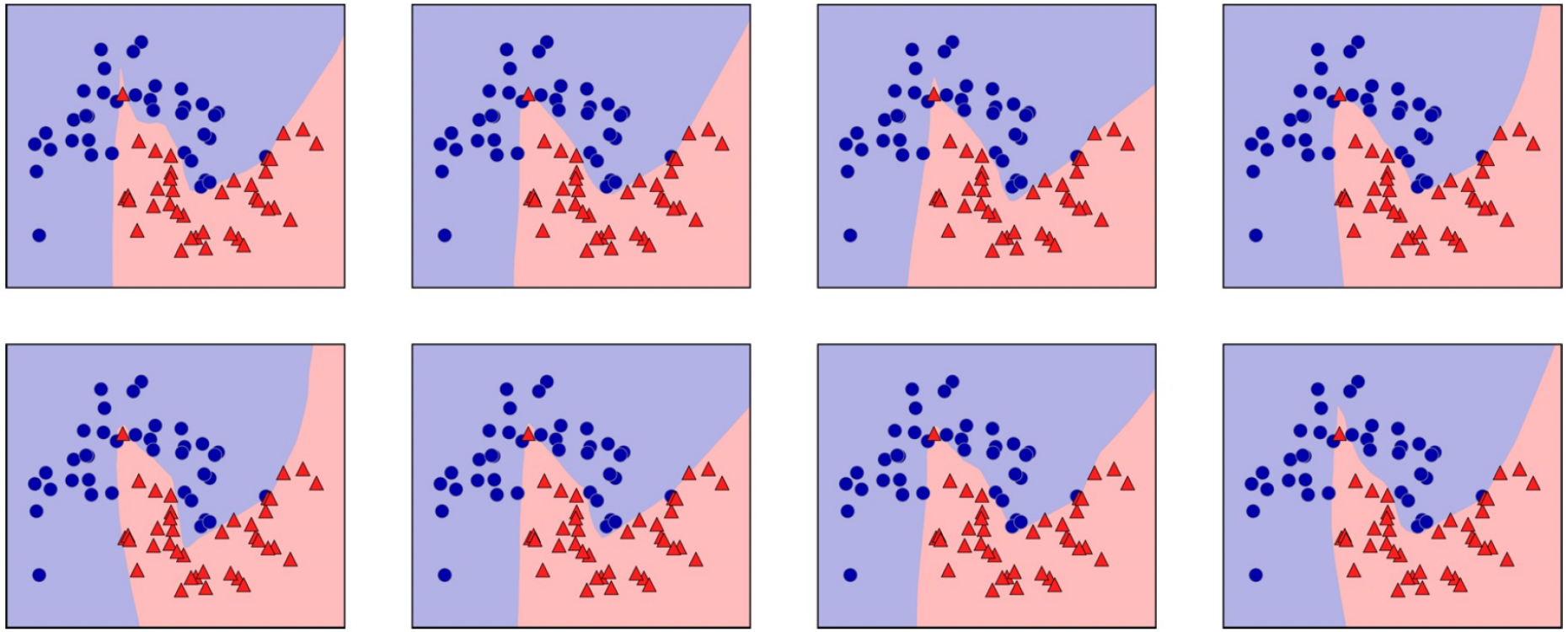
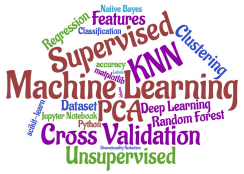


Figure 2-53. Decision functions learned with the same parameters but different random initializations



Strengths, weaknesses and parameters

Neural networks

Parameters

Tuning neural network parameters is also an art unto itself.

- the number of layers
- the number of hidden units per layer.
  - The number of nodes per hidden layer is often similar to the number of input features, but rarely higher than in the low to mid-thousands.
- Activation function
- Regularisation
- And many more

Model complexity of a NN

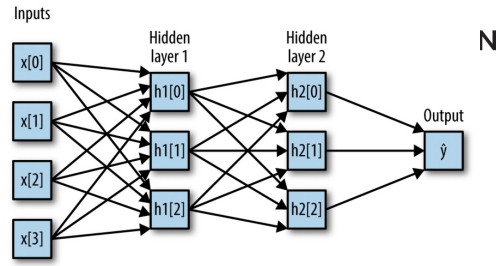


Figure 2-47. A multilayer perceptron with two hidden layers

- A binary classification dataset , 100 features, 100 hidden units
- Between the input and the first hidden layer: 100 *100 = 10,000 weights
- Between the hidden layer and the output layer: 100 * 1 = 100 weights
- Total: around 10,100 weights
- Add a second hidden layer with 100 hidden units,

  another 100 * 100 = 10,000 weights from the first hidden layer to the second hidden layer
- Total 20,100 weights

Model complexity of a NN

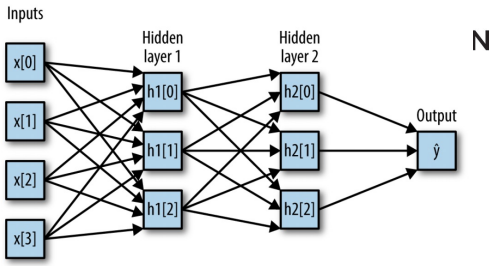


Figure 2-47. A multilayer perceptron with two hidden layers

- A binary classification dataset , 100 features, 1000 hidden units
- Between the input and the first hidden layer:
  100 *1000 = 100,000 weights
- Between the hidden layer and the output layer
  1000 * 1 = 1000 weights
- Total: around 101,000 weights.
- Add a second hidden layer with 100 hidden units:
  another 1000 * 1000 = 1,000,000 weights from the first hidden layer to the second hidden layer
- Whopping total: 1,101,000 weights
- 50 times larger than the model with two hidden layers of size 100.

How to adjust parameters?

- First create a network that is large enough to overfit
  - making sure that the task can actually be learned by the network.

- Then, once you know the training data can be learned, either:
  - shrink the network or
  - increase alpha to add regularization,

which will improve generalization performance.

Strengths

- They capture information contained **in large amounts of data** and build incredibly complex models.
- Given enough computation time, data, and careful tuning of the parameters, neural networks **often beat other machine learning algorithms** (for classification and regression tasks)

Weaknesses

- Neural networks—particularly the large and powerful ones—often take **a long time** to train.
- They also require careful **pre-processing of the data**, as we saw here
- Like SVMs, they **work best with "homogeneous" data**, where all the features have similar meanings.
- For data that has very different kinds of features, tree-based models might work better.

Conclusion

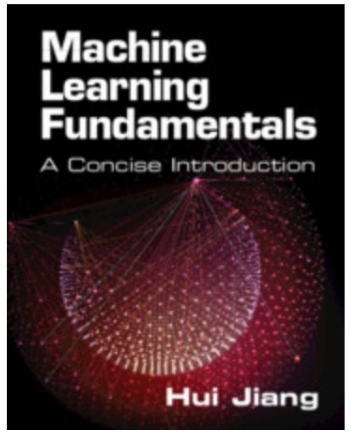
- Artificial neural networks are very flexible
- They are models with too many parameters (over-parametrised)
- Because of the over-parametrisation different local solution can occur in the fitting
- The interpretation of neural networks is difficult

Libraries in Python for NN and deep learning

- Scikitlearn:
  - MLPClassifier
  - MLPRegressor
- They only capture a small subset of what is possible with neural networks.
- More flexible or larger models for Python users:
  - keras,
  - lasagna
  - tensor-flow.
- Using **GPUs** allows us to accelerate computations by factors of 10x to 100x

More resources added to your reading list

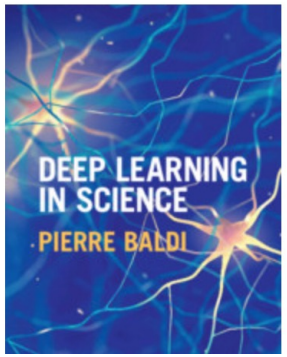
- Machine Learning, a concise introduction, 2021
- Deep Learning in Science, 2021



Access Cited by 14

Hui Jiang, *York University, Toronto*

Publisher:
Online publication date:
Print publication year:
Online ISBN:
DOI:

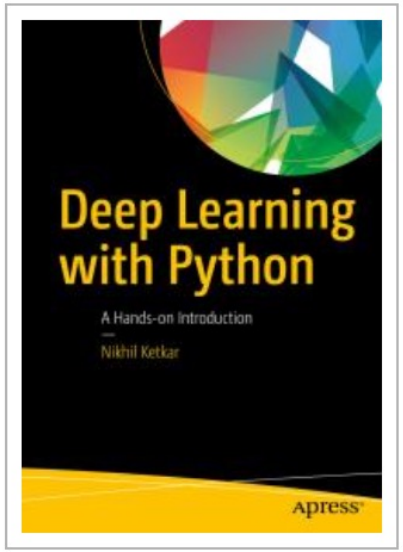


Access Cited by 21

Pierre Baldi, *University of California, Irvine*

Publisher:	Cambridge University Press
Online publication date:	April 2021
Print publication year:	2021
Online ISBN:	9781108955652
DOI:	https://0-doi-org.emu.londonmet.ac.uk/10.1017/9781108955652

- Deep Learning with Python: A Hands-On Introduction



Back-propagation

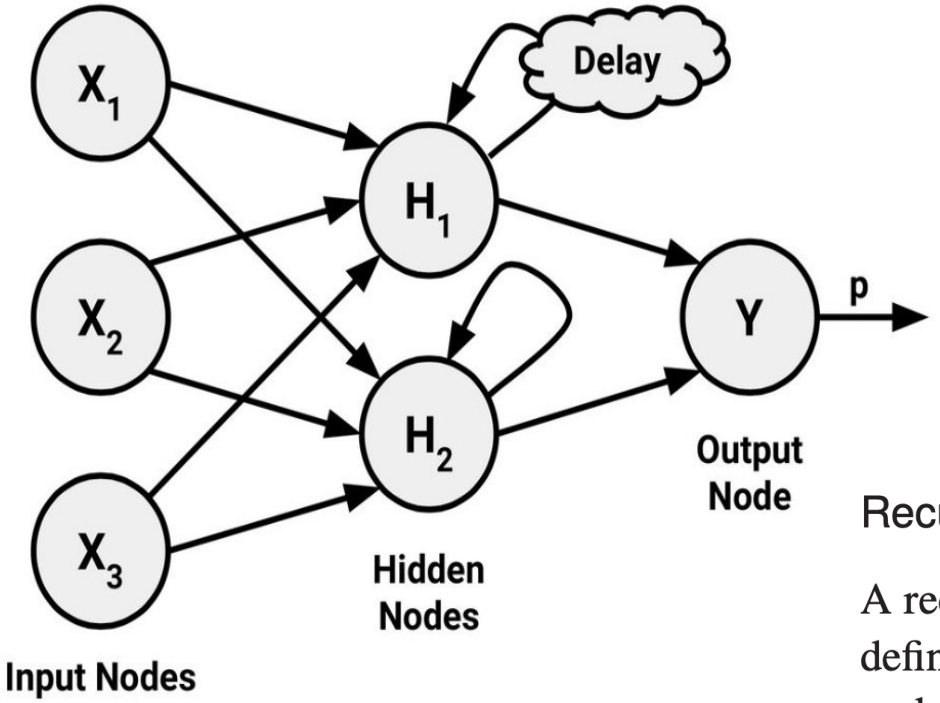
- Back-propagation is just **a way of propagating the total loss back into the neural network** to know how much of the loss every node is responsible for
- and subsequently updating the weights in such a way that minimises the loss by giving the nodes with higher error rates lower weights and vice versa.

Learning by Backpropagation

Backpropagation is the most widely used and most efficient algorithm for computing gradients in neural networks, and thus for learning by gradient descent. The algorithm is a straightforward application of the chain rule for computing derivatives.

Source: chapter 6, Baldi's book

Recurrent NN



Recurrent Architectures

A recurrent architecture is any architecture that contains at least one directed cycle. To define how the architecture operates, one must also specify in which order the neurons are updated (e.g. synchronously, stochastically, according to a fixed order). By discretizing time and using synchronous updates, a recurrent architecture with n neurons can always be "unfolded in time" into a feedforward layered architecture (see Chapter 9), where each layer has n neurons, and the depth of the unfolded architecture is equal to the number of time steps.

Read more on RNN in Baldi's and Kektar's book

Source: ch2, Baldi's book

Convolutional Neural Networks (CNN)

- Chapter 5 of Ketkar's book
- Chapter 8, section 8.2.3 of Jiang's book, CNN Case study

- CNNs are the dominant machine learning models to handle visual data, such as images and videos.
- CNNs are also applied to many other applications involving sequential data, such as speech, audio, and text.

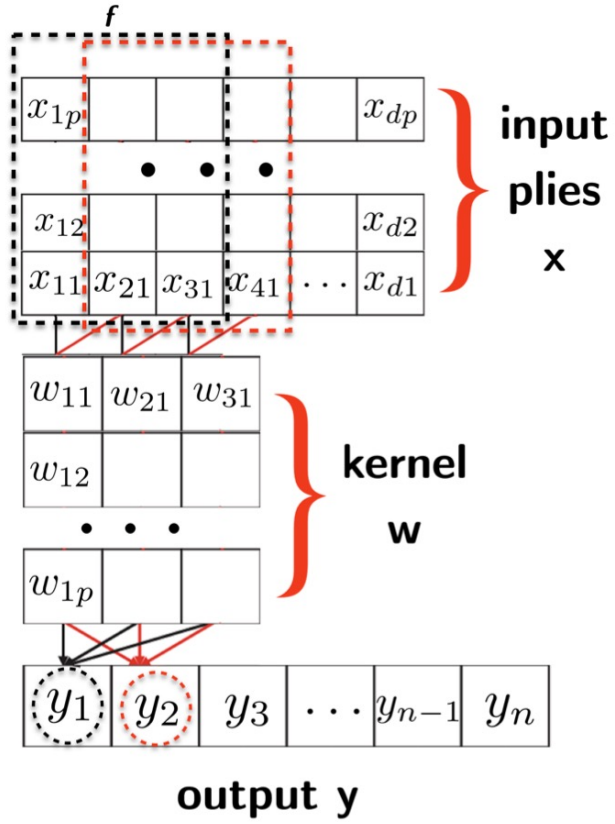


Figure 8.20: An illustration of the 1D convolution sum involving multiple input feature plies.

Source: ch 8, Jiang's book