

Neural Networks

W. Evan Johnson, Ph.D.
Professor, Division of Infectious Disease
Director, Center for Data Science
Rutgers University – New Jersey Medical School

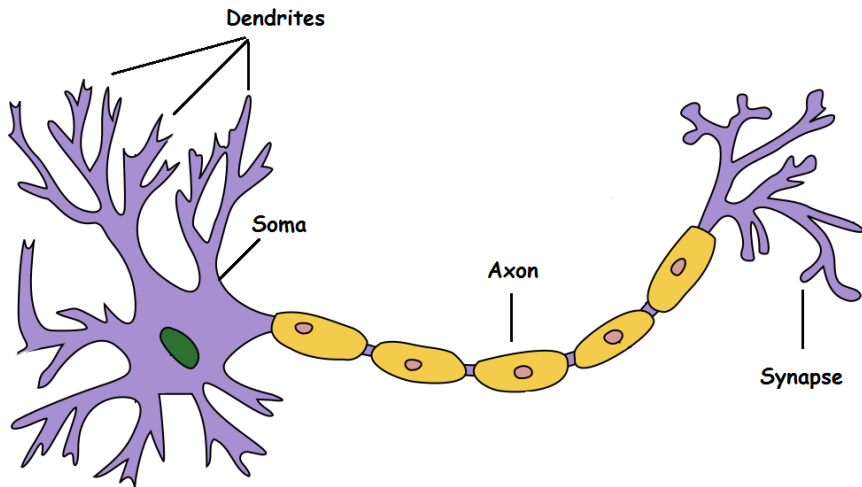
11/20/2023

Introduction to Neural Networks

The human brain consists of billions of neural cells that process information. Each neural cell considered a simple processing system. The interconnected web of neurons, also known as a **biological neural network**, transmits information through electrical signals from neuron to neuron.

Specifically, the dendrites of a neuron receive input signals from another neuron. The cell body, or soma, sums the inputs from multiple dendrites. Axons pass on information and output from the soma to the synapses, which are the *conjunction* points for other neurons.

Introduction to Neural Networks



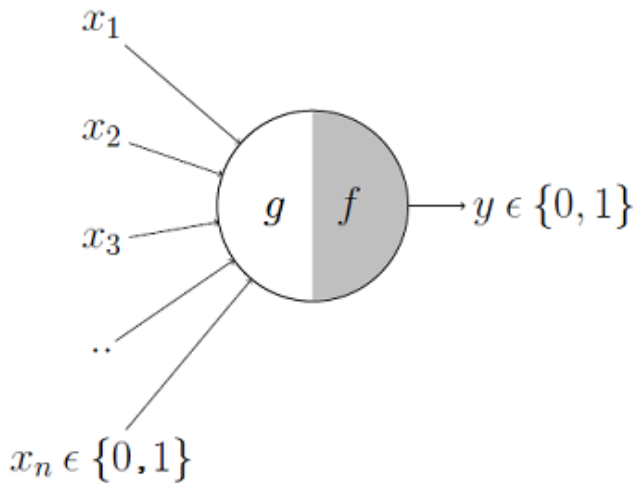
Introduction to Neural Networks

In 1944, Warren McCulloch and Walter Pitts developed the first mathematical model of a neuron.

In their research paper “A logical calculus of the ideas immanent in nervous activity”, they described the simple mathematical model for a neuron, which represents a single cell of the neural system that takes inputs, processes those inputs, and returns an output. This model is known as the McCulloch-Pitts neural model.

Introduction to Neural Networks

The McCulloch-Pitts neuron model looked something like this:



Introduction to Neural Networks

The neural nets described by McCullough and Pitts in 1944 had thresholds and weights, but they weren't arranged into layers, and the researchers didn't specify any training mechanism.

What McCullough and Pitts showed was that a neural net could, in principle, compute any function that a digital computer could. The result was more neuroscience than computer science: The point was to suggest that the human brain could be thought of as a computing device.

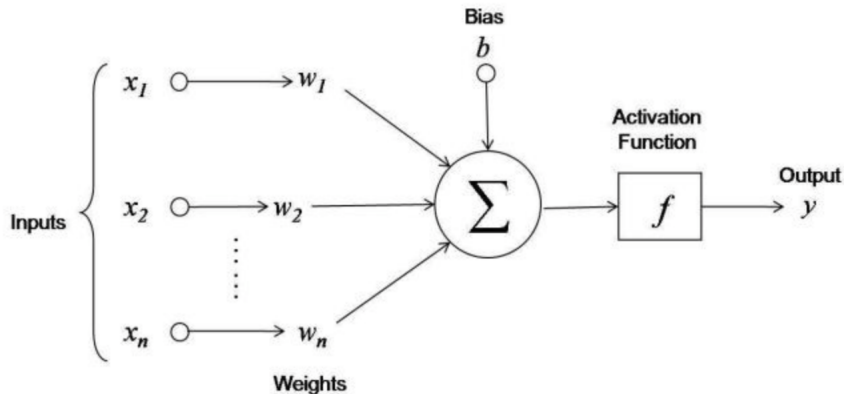
Introduction to Neural Networks

The first trainable neural network, the **Perceptron**, was demonstrated by the Cornell University psychologist Frank Rosenblatt in 1957. The Perceptron's design was much like that of the modern neural net, except that it had only one layer with adjustable weights and thresholds, sandwiched between input and output layers.

Perceptrons were an active area of research in both psychology and the fledgling discipline of computer science until 1959, when Minsky and Papert published a book titled "Perceptrons," which demonstrated that executing certain fairly common computations on Perceptrons would be *impractically time consuming*.

Introduction to Neural Networks

The Perceptron model looked something like this:



Introduction to Neural Networks

Here, x_1, x_2, \dots, x_N are input (predictor) variables, w_1, w_2, \dots, w_N are weights of respective inputs, b is the bias, which is summed with the weighted inputs to form the net inputs. Bias and weights are both adjustable parameters of the (computational) neuron.

Parameters are adjusted using learning rules trained on the x s. Once the weights and bias are determined, a mapping function or mechanism processes or connects the input and output of the neuron. This mechanism of mapping inputs to output is known as the **activation function**.

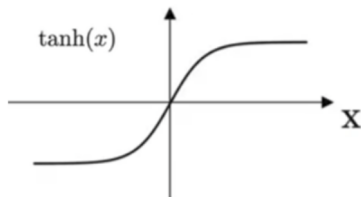
Introduction to Neural Networks

The **activation function** defines the output of a neuron in terms of a local induced field. The activation function (often a single line of code!) can give the neural net non-linearity and expressiveness. Here are some examples:

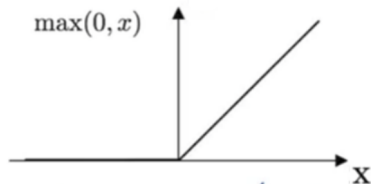
- Identity: maps input to the same output value. It is a linear operator in vector space.
- Binary Step: if the value of the (weighed) input signal is above a certain threshold, the output is true (or activated), and otherwise false (or not activated). It is very useful as a binary classifier.
- Sigmoid or S-shaped: Logistic and hyperbolic tangent functions are commonly used sigmoid functions, used for binary classification or continuous/probabilistic outputs
- Ramp or ReLU: Derived from the appearance of its graph, e.g.,. maps negative inputs to 0 and positive inputs to the same (identity) output. *ReLu* stands for 'Rectified Linear unit'.

Introduction to Neural Networks

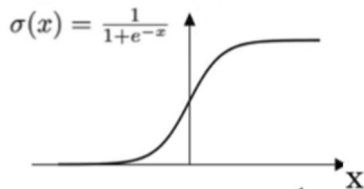
Hyper Tangent Function



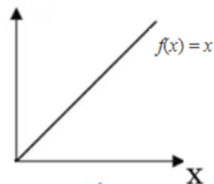
ReLU Function



Sigmoid Function

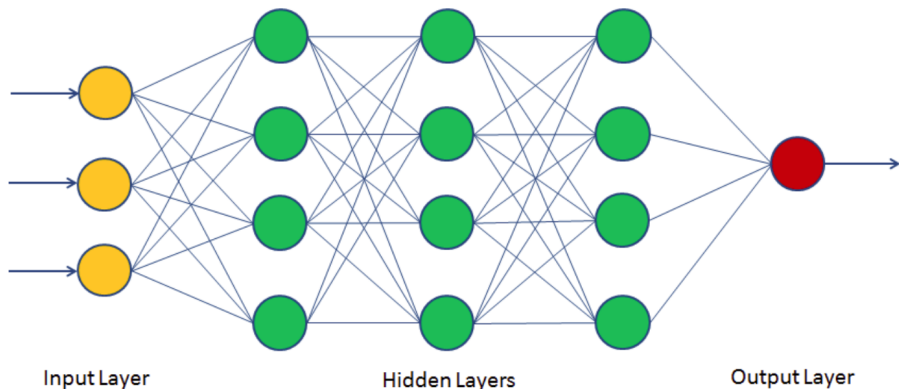


Identity Function



Introduction to Neural Networks

More generally, modern **neural networks** are algorithms further inspired by the human brain, comprising multiple layers of connected input/output units (neurons) in which each connection has a weight associated with it.



Introduction to Neural Networks

Neural networks (or **artificial neural networks**) have the ability to learn by examples, or from training data. Neural networks follow a *non-linear* path that processes information through a complex adaptive system—by adjusting weights of inputs using a complex *network* of artificial neurons.

Neural networks where designed to solve problems which are easy for humans and difficult for machines. For example distinguishing between pictures of cats and dogs. These problems are often referred to as pattern recognition. Current applications for neural network range from optical character recognition to object detection.

Introduction to Neural Networks

Two of the more common artificial neural networks are: **feedforward** and **feedback** neural networks.

A *feedforward neural network* is a network which is non-recursive: neurons in each layer are only connected to neurons in the next layer and do not form a cycle. Signals only travel in one direction towards the output layer.

Feedback neural networks contain cycles. Signals travel in both directions by introducing loops in the network. The feedback cycles can cause the network's behavior change over time based on its input. Feedback neural networks are also known as *recurrent neural networks*.

Introduction to Neural Networks

There are many other formulations of **deep learning** neural networks:

- Perceptron
- Multilayer Perceptron
- Feedforward Neural Network
- Recurrent Neural Network
- Convolutional Neural Network
- Radial Basis Functional Neural Network
- LSTM – Long Short-Term Memory
- Sequence to Sequence Models
- Modular Neural Network

Convolutional Neural Networks

In deep learning, a **convolutional neural network (CNN, or ConvNet)** is a class of artificial neural network (ANN), most commonly applied to analyze visual imagery.

CNNs are also known as **Shift Invariant** or **Space Invariant Artificial Neural Networks (SIANN)**, based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide **translation-equivariant** responses known as feature maps. Counter-intuitively, most convolutional neural networks are only **equivariant**, as opposed to **invariant**, to translation.

CNNs have applications in image and video recognition, recommender systems, image classification/segmentation, medical image analysis, natural language processing, brain–computer interfaces, and financial time series.

Convolutional Neural Networks

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer.

The “full connectivity” of these networks make them prone to overfitting data. Typical ways of regularization, or preventing overfitting, include: penalizing parameters during training (such as weight decay) or trimming connectivity (skipped connections, dropout, etc.)

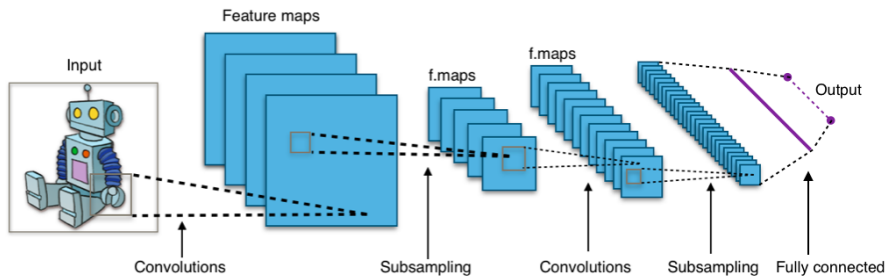
Convolutional Neural Networks

CNNs often take a different approach towards regularization: they utilize the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters. Therefore, on a scale of connectivity and complexity, CNNs are on the lower extreme.

Convolutional networks were inspired by biological processes: the connectivity pattern resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

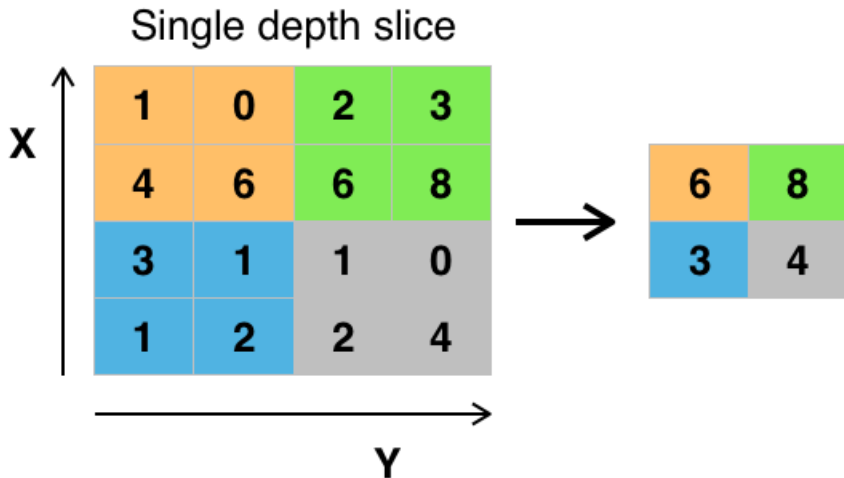
Convolutional Neural Networks

CNNs use relatively little pre-processing compared to other classification algorithms. The network learns to optimize the filters (or kernels) through automated learning, whereas in traditional algorithms these filters are hand-engineered. This independence from prior knowledge and human intervention in feature extraction can be a major advantage.



Convolutional Neural Networks

Maximum filter example (or **pooling**):



Convolutional Neural Networks

Sliding (overlapping) filter example (**kernel**, **stride** and **padding**):

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

1	1	1	0	0
0	1	1 _{x1}	1 _{x0}	0 _{x1}
0	0	1 _{x0}	1 _{x1}	1 _{x0}
0	0	1 _{x1}	1 _{x0}	0 _{x1}
0	1	1	0	0

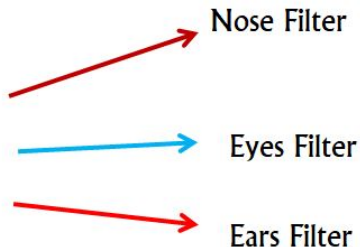
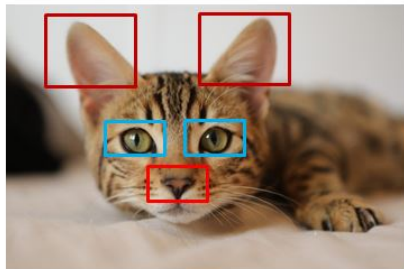
Image

4	3	4
2	4	3

Convolved
Feature

Convolutional Neural Networks

Now applying this to cats:



Convolutional Neural Networks

More realistically:



Cat



Cat



Representation

Convolutional Neural Networks

And as a classifier:



Multilayer Perceptron Neural Network in R

Suppose we have students' technical knowledge (TKS), communication skill score (CSS), and placement status (Placed):

```
TKS=c(20,10,30,20,80,30)
CSS=c(90,20,40,50,50,80)
Placed=c(1,0,0,0,1,1)
df=data.frame(TKS,CSS,Placed)
knitr::kable(df)
```

TKS	CSS	Placed
20	90	1
10	20	0
30	40	0
20	50	0
80	50	1
30	80	1

Multilayer Perceptron Neural Network in R

Fit the multilayer perceptron neural network:

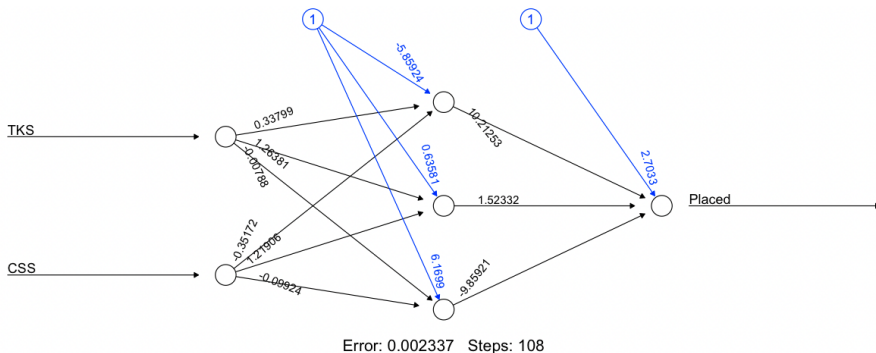
```
suppressPackageStartupMessages(library(neuralnet))
set.seed(0)
nn=neuralnet::neuralnet(Placed~TKS+CSS,data=df, hidden=3,
                        act.fct = "logistic", linear.output = FALSE)
names(nn)
```

##	[1]	"call"	"response"	"covariate"
##	[4]	"model.list"	"err.fct"	"act.fct"
##	[7]	"linear.output"	"data"	"exclude"
##	[10]	"net.result"	"weights"	"generalized.weights"
##	[13]	"startweights"	"result.matrix"	

Multilayer Perceptron Neural Network in R

We can plot our neural network:

```
plot(nn)
```



Multilayer Perceptron Neural Network in R

Creating a test set:

```
TKS=c(30,40,85)  
CSS=c(85,50,40)  
test=data.frame(TKS,CSS)  
knitr::kable(test)
```

TKS	CSS
30	85
40	50
85	40

Multilayer Perceptron Neural Network in R

Prediction using neural network:

```
Predict=compute(nn,test)
Predict$net.result
```

```
##           [,1]
## [1,] 0.4013943
## [2,] 0.4014894
## [3,] 0.4974918
```

Multilayer Perceptron Neural Network in R

Converting probabilities into binary classes setting threshold level 0.5:

```
prob <- Predict$net.result  
pred <- ifelse(prob>0.5, 1, 0)  
pred
```

```
##      [,1]  
## [1,]    0  
## [2,]    0  
## [3,]    0
```

Multilayer Perceptron Neural Network in R

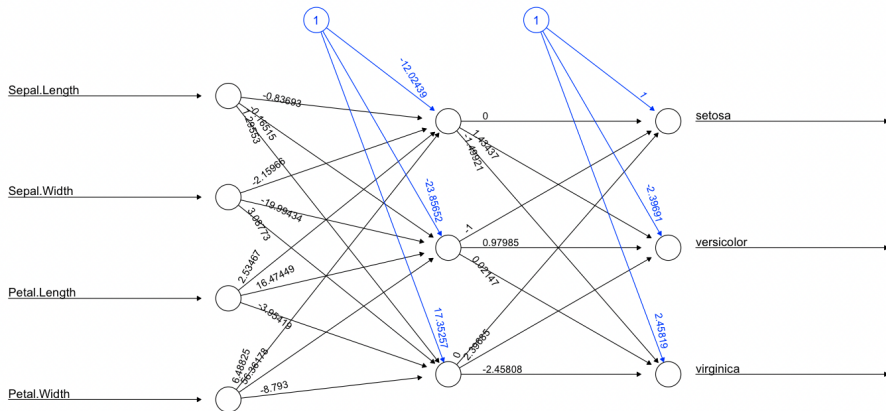
Now, using the iris dataset:

```
set.seed(0)
nn_iris <- neuralnet(Species ~ ., data=iris, hidden=3)
```

Multilayer Perceptron Neural Network in R

Plotting the Neural Network:

```
plot(nn_iris)
```



Error: 1.934716 Steps: 26654

Multilayer Perceptron Neural Network in R

Now, doing the same thing using 'caret':

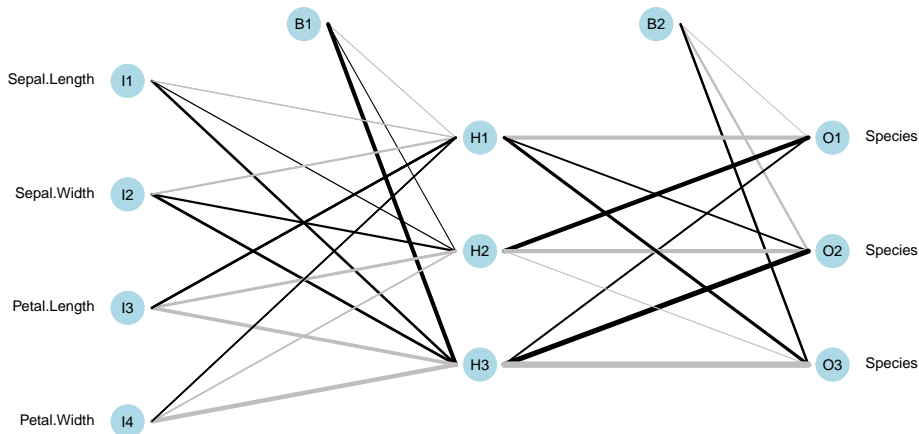
```
set.seed(0)
nn_caret <- caret::train(Species~., data = iris,
                        method = "nnet", linout = TRUE,
                        trace = FALSE)
ps <- predict(nn_caret, iris)
confusionMatrix(ps, iris$Species)$overall["Accuracy"]

## Accuracy
## 0.9733333
```

Multilayer Perceptron Neural Network in R

Plotting the 'caret' neural network:

```
NeuralNetTools::plotnet(nn_caret)
```



Session Info

```
sessionInfo()
```

```
## R version 4.3.2 (2023-10-31)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.5.1
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib; LAPACK version 3
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/Denver
## tzcode source: internal
##
## attached base packages:
## [1] stats      graphics  grDevices  utils      datasets  methods   base
##
## other attached packages:
## [1] neuralnet_1.44.2 lubridate_1.9.3 forcats_1.0.0 stringr_1.5.1
## [5] dplyr_1.1.3      purrr_1.0.2      readr_2.1.4     tidyr_1.3.0
## [9] tibble_3.2.1     tidyverse_2.0.0  caret_6.0-94    lattice_0.22-5
## [13] ggplot2_3.4.4    knitr_1.45
##
## loaded via a namespace (and not attached):
## [1] gtable_0.3.4      xfun_0.41         recipes_1.0.8
## [4] tzdb_0.4.0        vctrs_0.6.4       tools_4.3.2
## [7] generics_0.1.3    stats4_4.3.2      parallel_4.3.2
## [10] proxy_0.4-27      fansi_1.0.5       pkgconfig_2.0.3
## [13] ModelMetrics_1.2.2.2 Matrix_1.6-3      data.table_1.14.8
## [16] lifecycle_1.0.4   farver_2.1.1      compiler_4.3.2
```