COLUMBIA UNIVERSITY
School of Professional Studies

Applied Analytics: Frameworks and Methods 2

Neural Networks

Outline

- Introduction to Neural Networks
- Artificial Neuron
- Multiple Layer Neural Networks
- Network Architecture
- Types of Networks
- Applications
- Using Deep Learning at Scale
- Illustration of Neural Networks on MNIST

Introduction to Neural Networks

Deep Learning

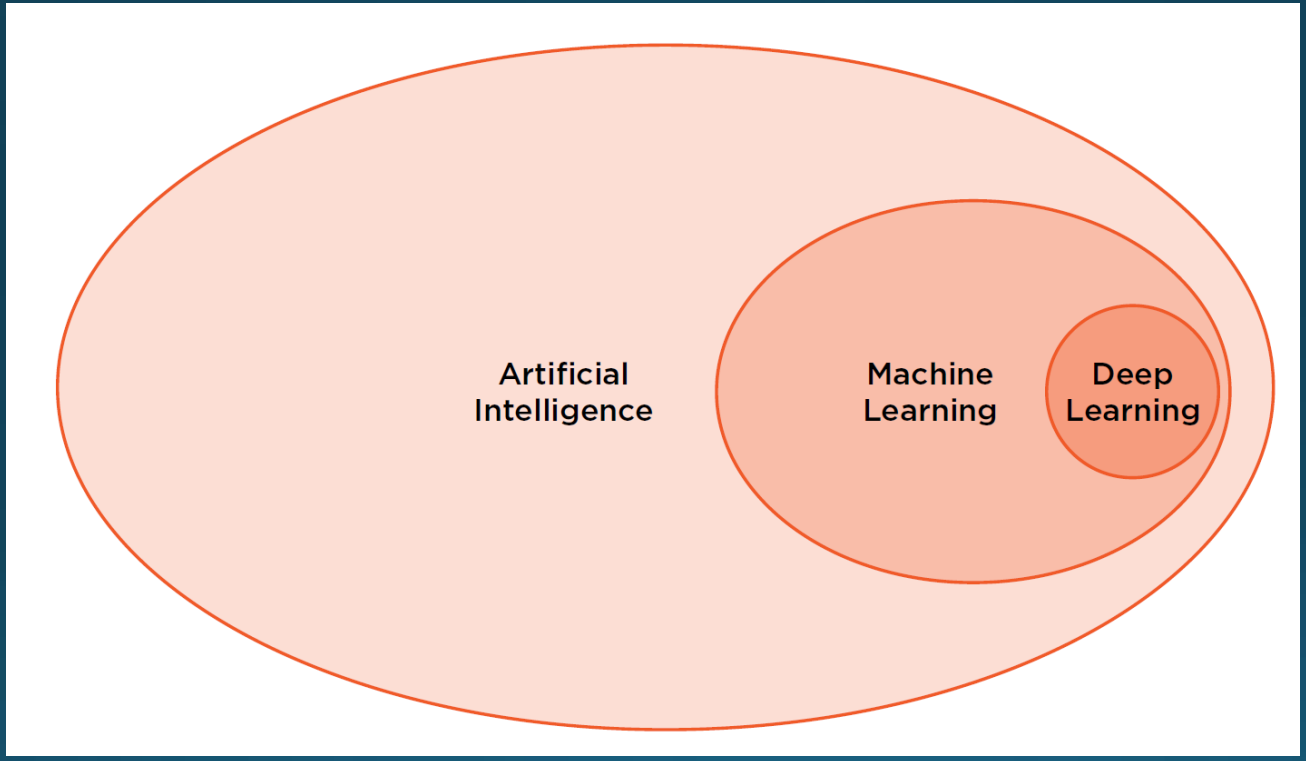
- Artificial Neural Networks, conceived in the 1950s as a crude approximation of how the human brain works, are the basis of what is now referred to as Deep Learning.

- Artificial Neural Networks used to address machine learning problems are referred to as Deep Learning. Since our goal is to apply neural networks to solve machine learning problems, we will use the terms Deep Learning and Neural Networks interchangeably.

- Deep Learning is a form of *Artificial Intelligence* that uses a type of machine learning called an *artificial neural network* with *multiple hidden layers* that learns *hierarchical representations* of the underlying data in order to make *predictions* given new data.

Reference free online book: http://neuralnetworksanddeeplearning.com/

Deep Learning
is a Form of Artificial Intelligence



Artificial Neural Network

Input:	0,	8,	15,	22,	38
Output:	32,	46.4,	59,	71.6,	?

Question: What will be the Output value for an Input value of 38?

Artificial Neural Network

Input:	**0,**	**8,**	**15,**	**22,**	**38**
Output:	**32,**	**46.4,**	**59,**	**71.6,**	**100.4**

$$F = C * 1.8 + 32$$

F = *Fahrenheit*
C = *Celsius*

Evaluation of Deep Learning

Pros

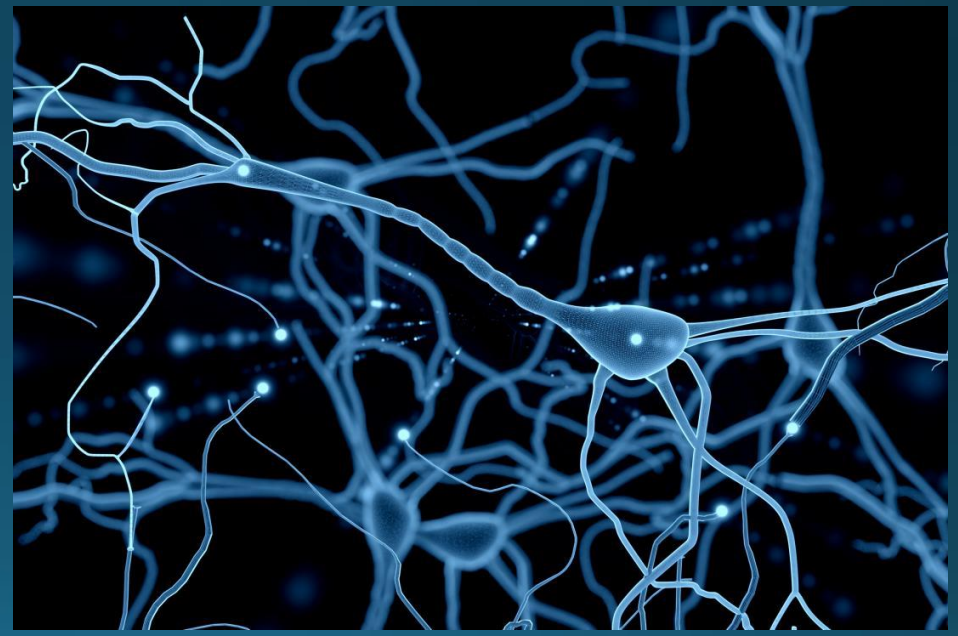
- Excels at tasks such as computer vision, natural language processing, speech recognition.
- Powers many recommender systems, and fraud detection systems.
- Algorithms are very general and adaptive
- Work directly on raw data. No feature engineering required.

Cons

- Computational resource intensive
- Tricky hyper parameterization
- Non-optimal methods

Artificial Neural Network

- Inspired by the biological neuron but they work very differently.
- Consists of a series of Neurons connected together in a network.
- Lets first examine an Artificial Neuron.

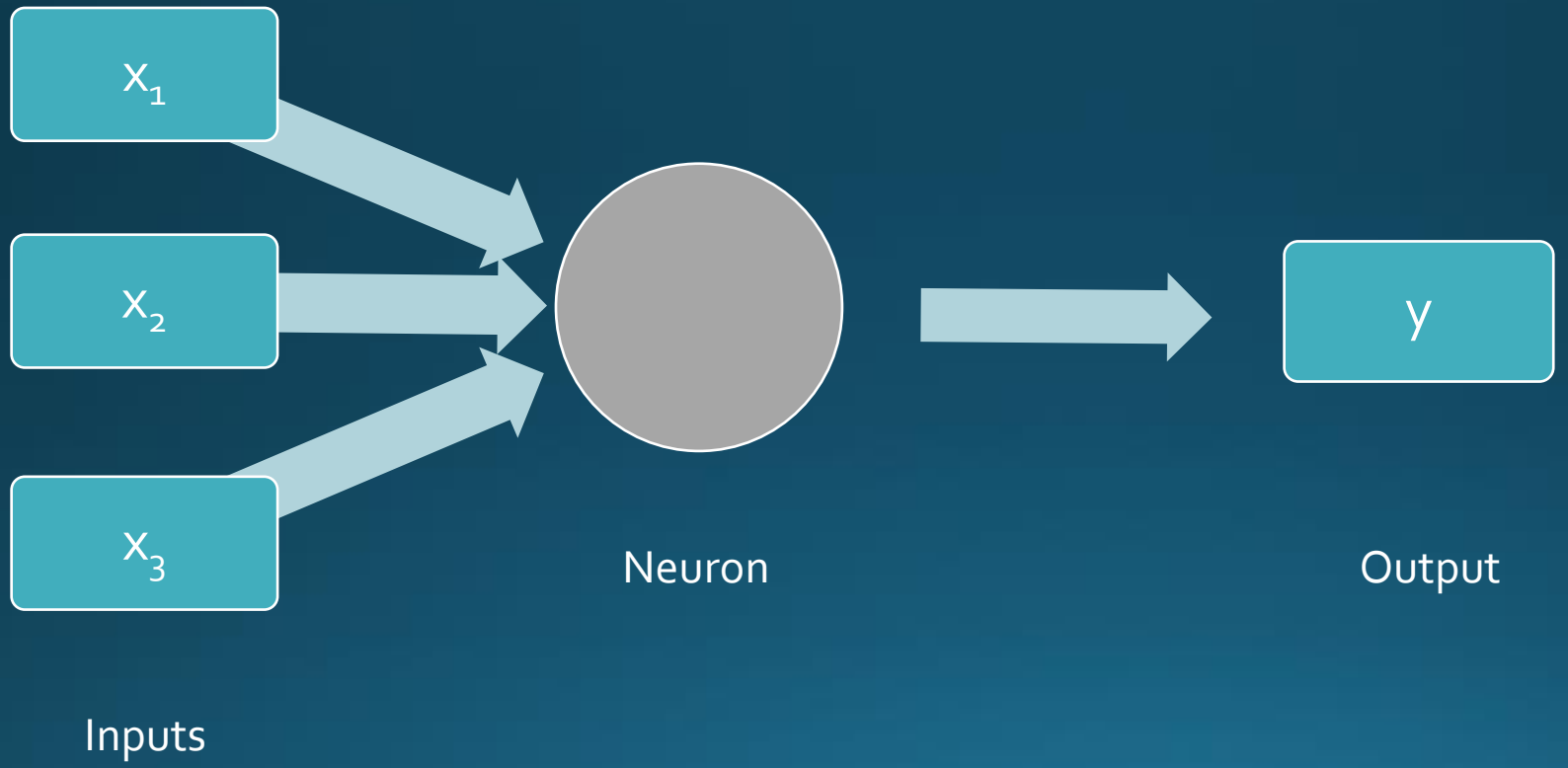


Artificial Neuron

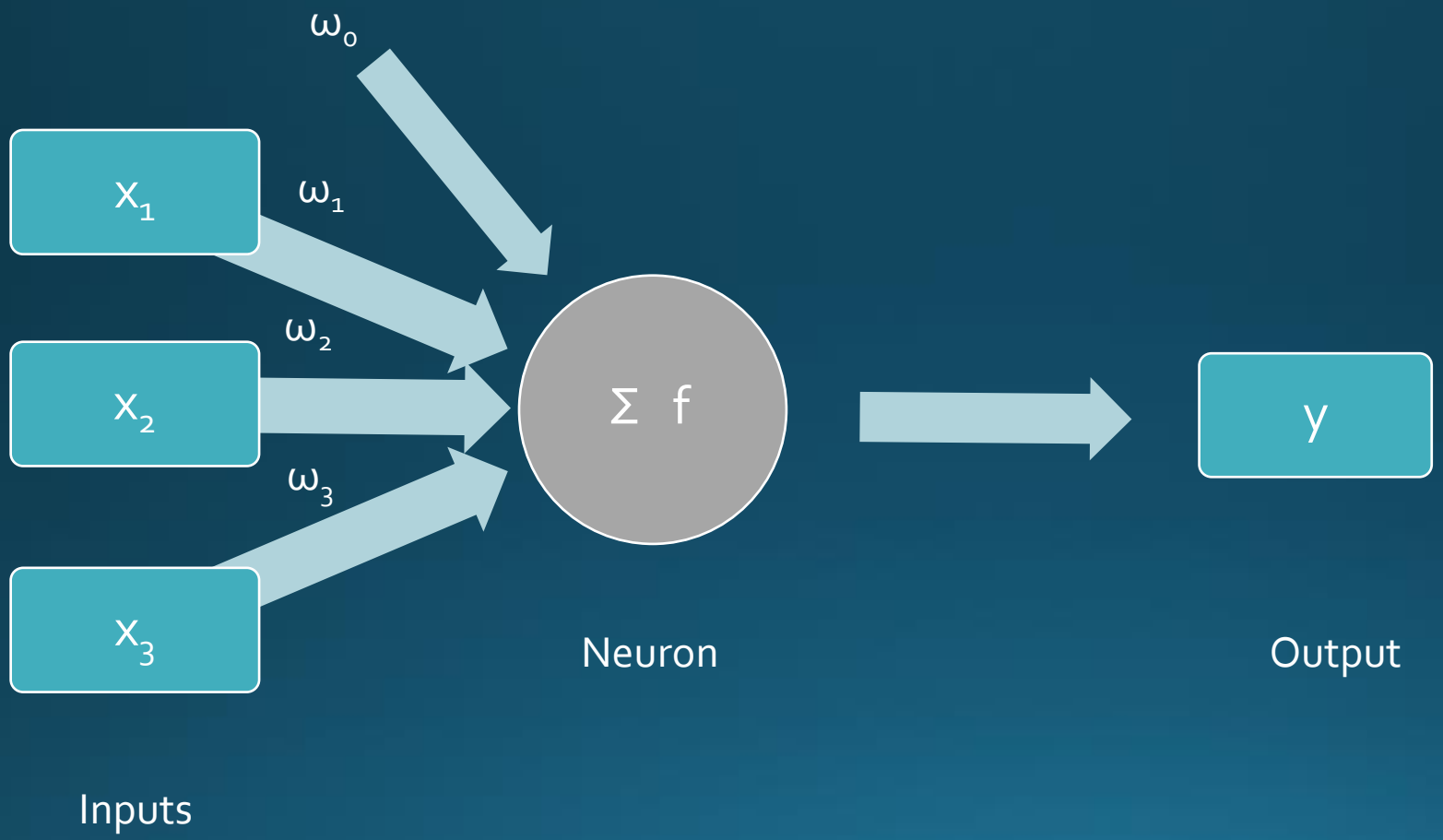
Artificial Neuron

- Input (x_1, x_2, x_3)
- Weights or parameters (ω_1, ω_2, ω_3)
- Bias (ω_0)
- Neuron
- Output (y)
- Activation function (e.g., tanh)

Artificial Neuron



Artificial Neuron



Activation Functions

- Linear
- Logistic (sigmoid)
- Hyperbolic Tangent (tanh)
- Rectified Linear Unit (ReLU)
- …

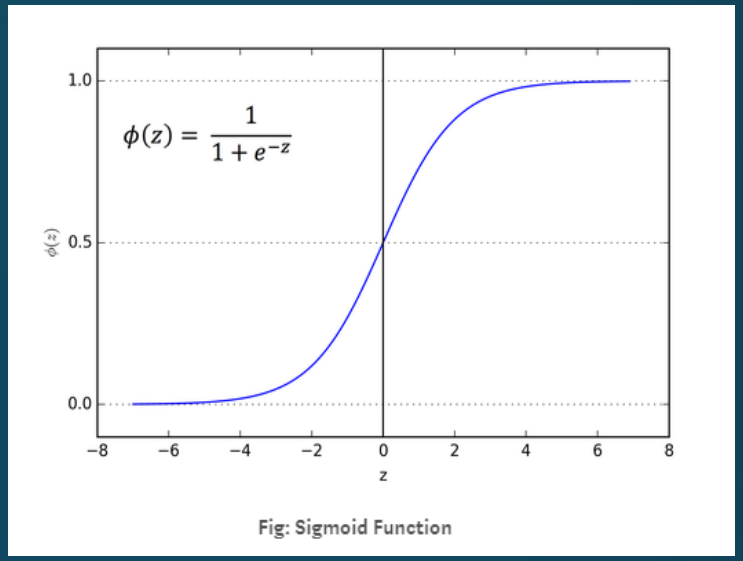


Fig: Sigmoid Function

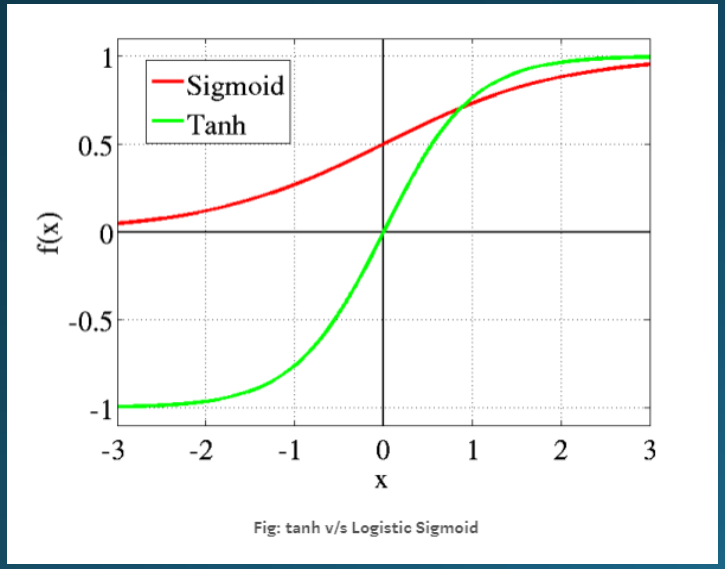


Fig: tanh v/s Logistic Sigmoid

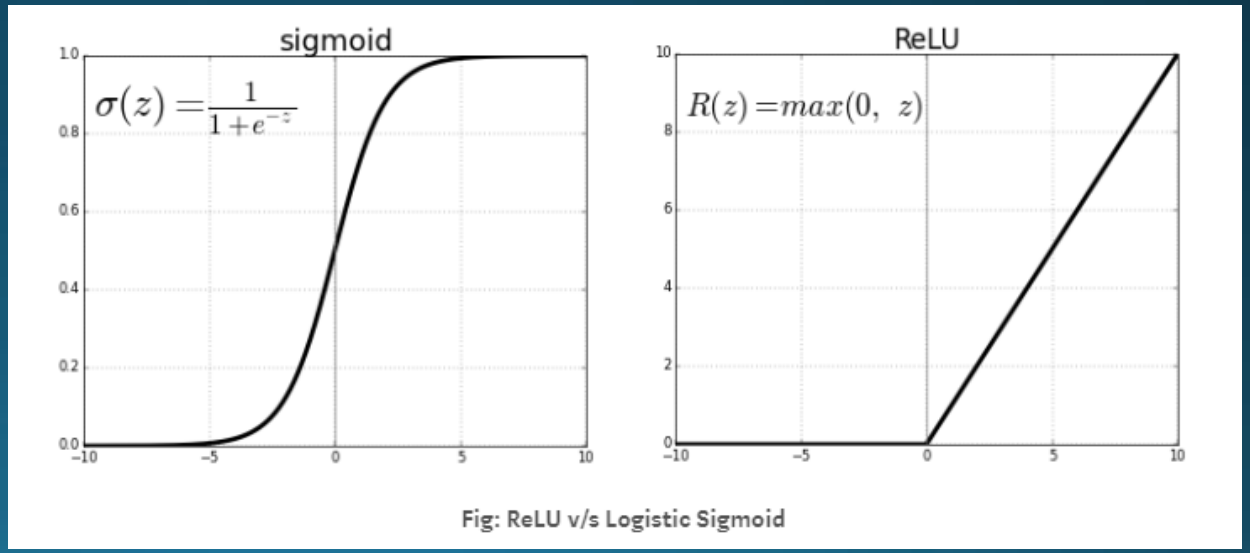
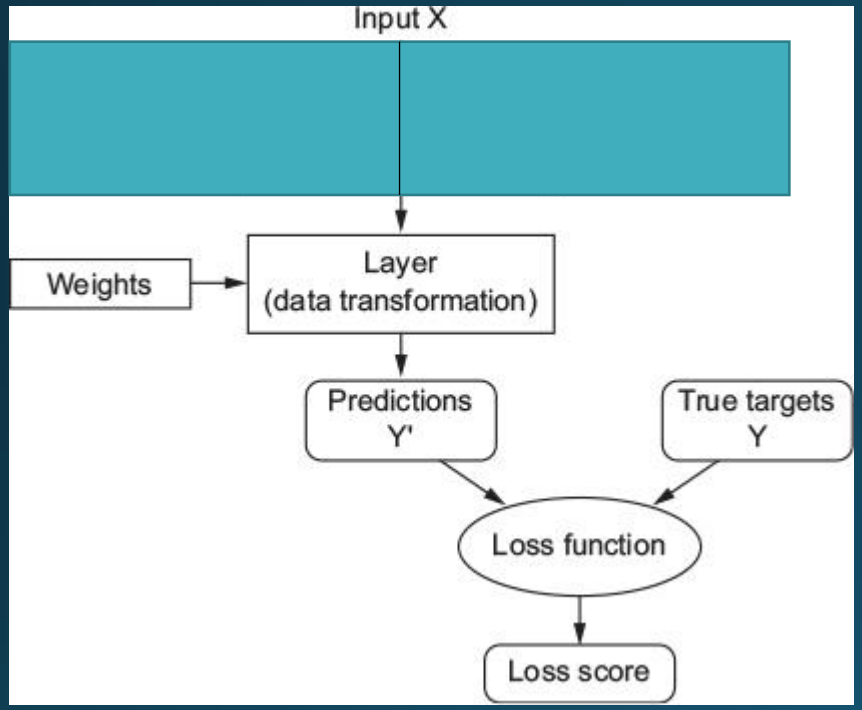
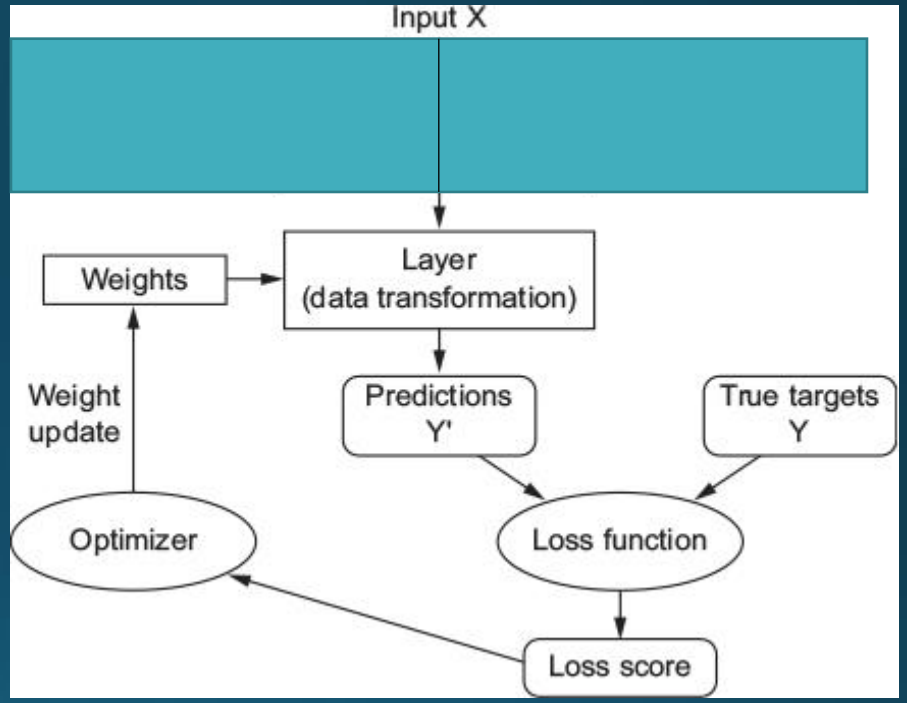


Fig: ReLU v/s Logistic Sigmoid

The loss function (e.g., mean square error, cross-entropy, Hinge) takes the predictions of the network and the true target and computes a distance score, capturing how well the network has done .

Use this score as a feedback signal to adjust the value of the weights a little, in a direction that will lower the loss score



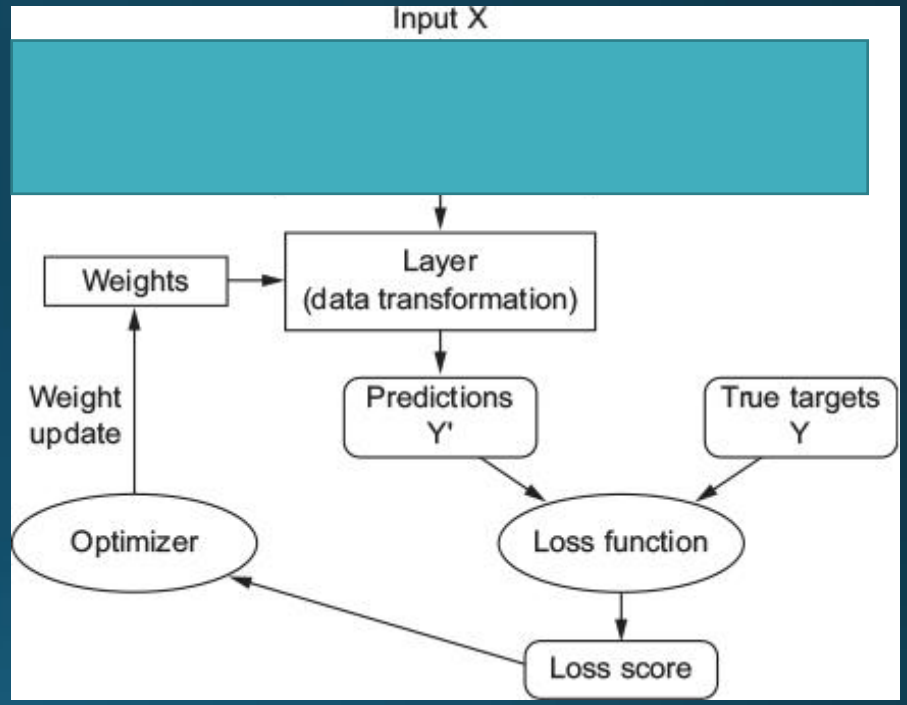


- Initially, the internal variables (weights and bias) of the network are assigned random values. Naturally, its output is far from what it should ideally be, and the loss score is accordingly very high. As the weights are adjusted a little in the correct direction, and the loss score decreases. This is the training loop, which, repeated a sufficient number of times to yields weight values that minimize the loss function.

- Determine parameter weights and bias by minimizing a loss function using gradient descent

Minimizing a Loss Function with Stochastic Gradient Descent

- Gradient Descent: An algorithm that changes the internal variables a bit at a time to gradually reduce the loss function.
- Gradient descent is computationally demanding for large datasets. Stochastic gradient descent or mini-batch gradient descent searches for minima using a small batches of the training set rather than the entire train sample.
- Learning rate: The "step size" for loss improvement during gradient descent.



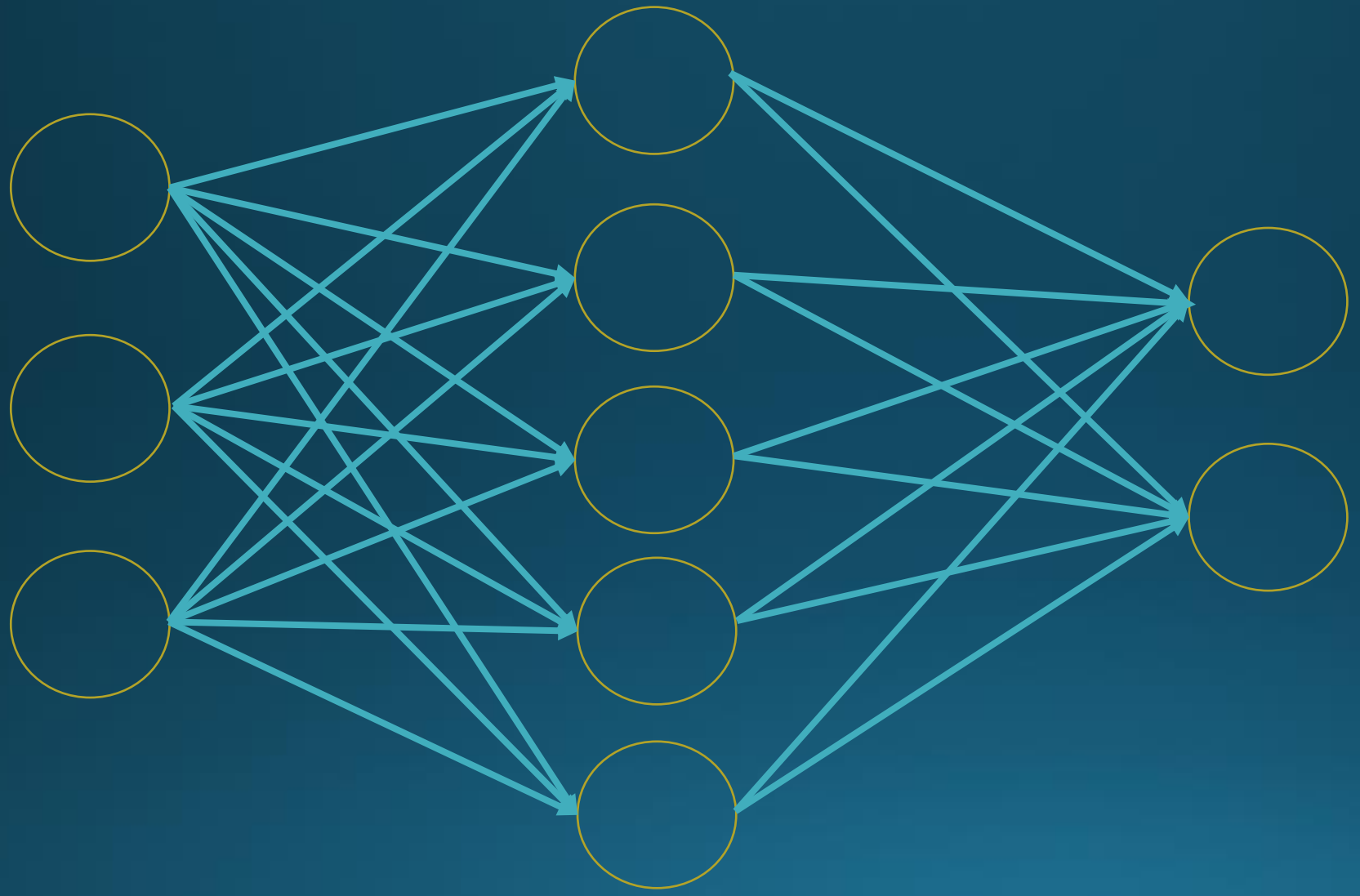
Multiple Layer Neural Networks

Artificial Neural Network

- Network of connected neurons

- Includes
  - Input Layer
  - One or more Hidden Layer(s)
  - Output Layer

Artificial Neural Network (with One Hidden Layer)



Inputs

Hidden Layer

Outputs

- Hidden Layer(s): 1
- Hidden Units/Neurons: 5

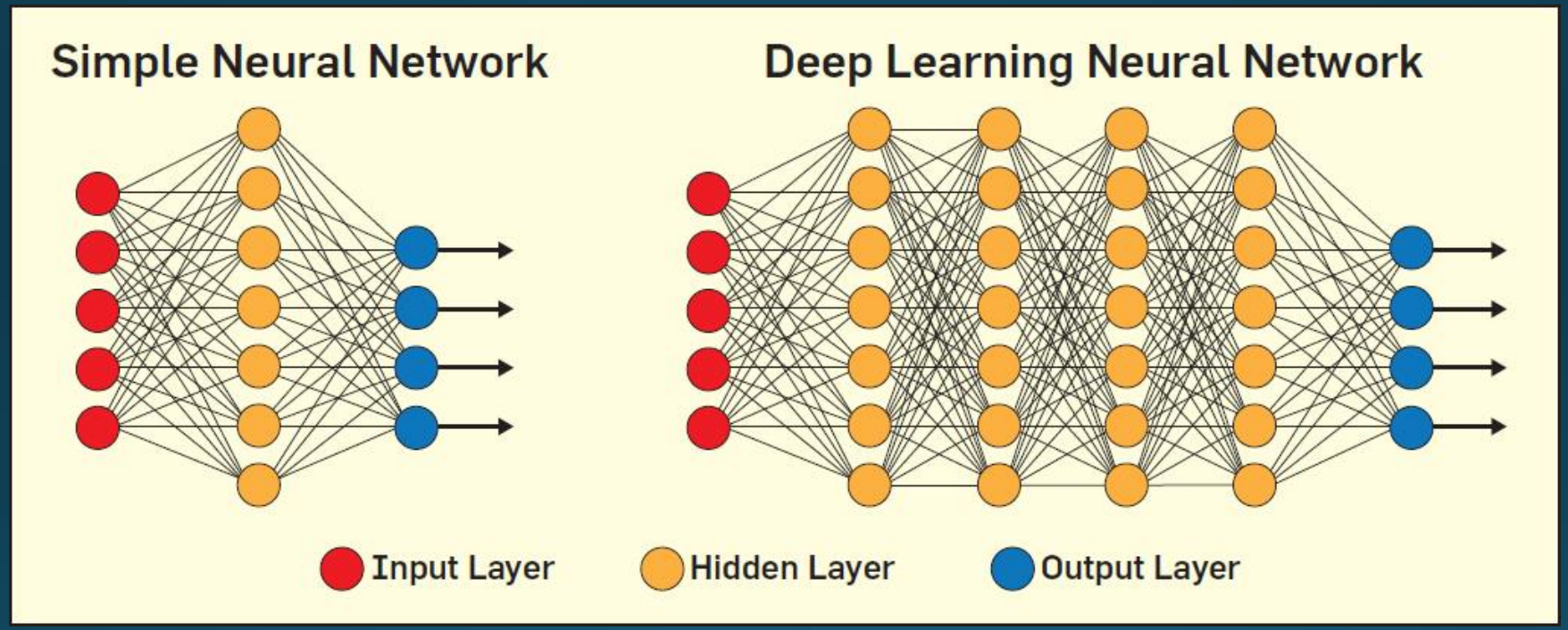
Artificial Neural Network

- In practice, each layer can be a
  - vector (one-dimensional)
  - matrix (two-dimensional array)
  - tensor (n-dimensional array)

Deep Learning Neural Network

- A neural network with more than one hidden layer
- Adding more hidden layers enables the network to model progressively more complex functions

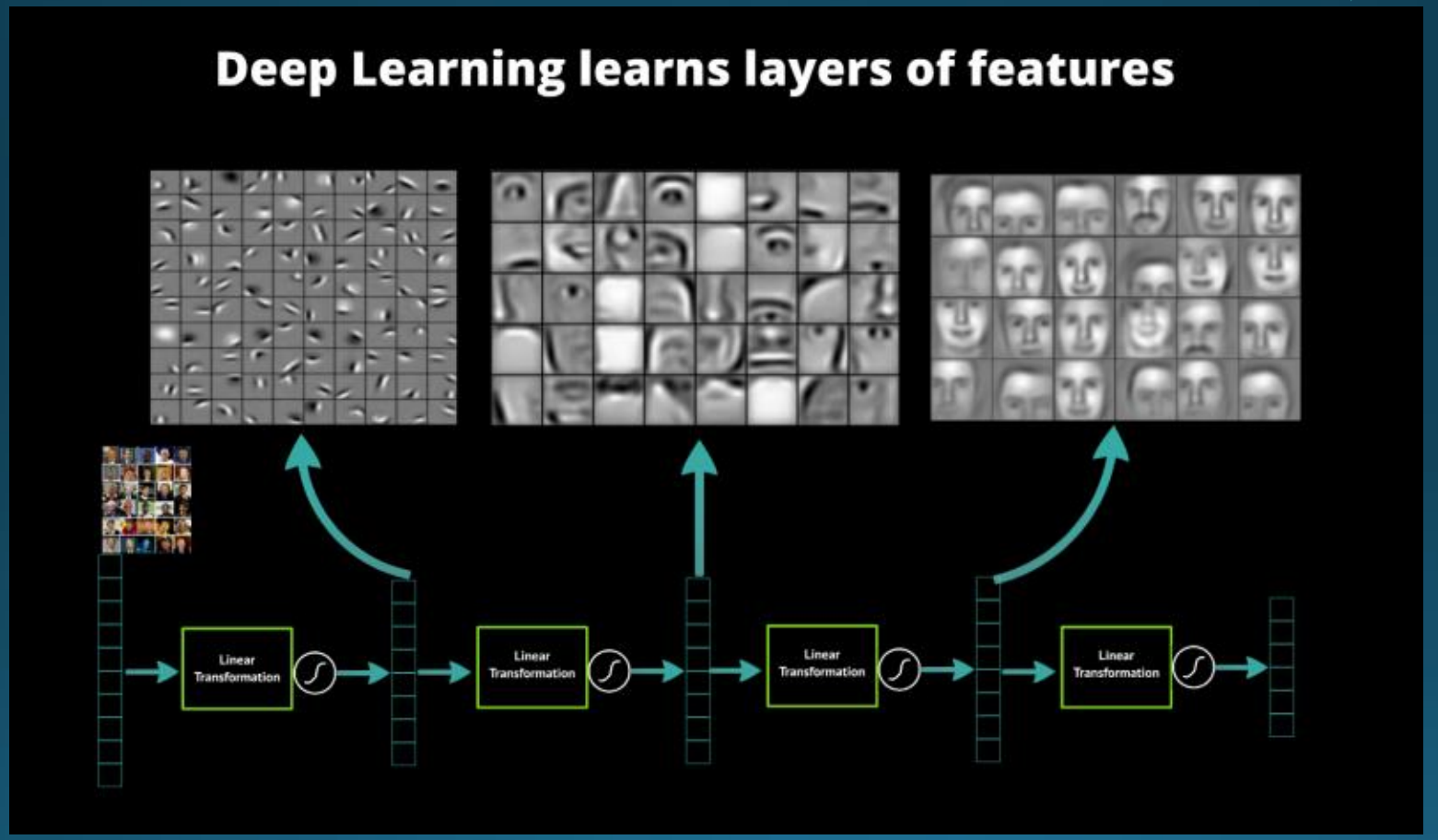
Deep Learning Neural Network



Source: Communications of the ACM

Hierarchical Representations

Greater Abstraction



Source: Data Science Central

Mechanics and Network Architecture

Mechanics of Neural Networks

- Specify Neural Network hyper-parameters
  - Number of hidden layers
  - Number of neurons per hidden layer
  - Activation function (tanh, softmax, sigmoid)
- Determine parameter weights by minimizing a loss function (e.g., mean square error, cross-entropy, Hinge) using stochastic gradient descent
  - Gradient descent is computationally demanding for large datasets. Stochastic gradient descent or mini-batch gradient descent searches for minima using a small batches of the training set rather than the entire train sample.
- Specify regularization terms (L1 and L2) to prevent overfitting

Network Architectures

- Various network architectures are possible by changing the
  - Number of hidden layers
  - Number of neurons per hidden layer
  - Activation function
- Adding more hidden layers makes the network deep
- Adding more neurons per layer makes the network wide

Network Architecture

- There is not a solid theoretical basis to determine the best Network Architecture
- Often a trial-and-error process
- Some approaches include
  - Using a network architecture for a similar problem
  - Progressively increasing complexity by adding more hidden units and layers until performance improvements become asymptotic

Tuning Hyper-Parameters

- The term hyper-parameters is used to distinguish them from standard model parameters
  - They define higher level concepts about the model like complexity or capacity to learn
  - They cannot be learned directly from the data in the standard model training process and need to be predefined
  - They are critical to model performance

Tuning Hyper-Parameters

- There are many hyper-parameters to tune
  - Number of hidden layers
  - Number of hidden units (number of neurons per hidden layer)
  - Number of training iterations
  - Activation function (tanh, sigmoid, etc)
  - Learning rate
  - Regularization
    - Specify regularization terms (L1 and L2) to prevent overfitting

Tuning Hyper-Parameters

Two approaches to tuning:

- Manual:
  - Quite a common approach
  - Without domain knowledge or experience with similar problems, this can take a long time
- Automatic:
  - Using Stochastic gradient descent for Hyper-Parameters means training the model from scratch each time! This is computationally expensive and only practical for small datasets. For small models, here are two approaches
    - Grid Search
    - Random Search

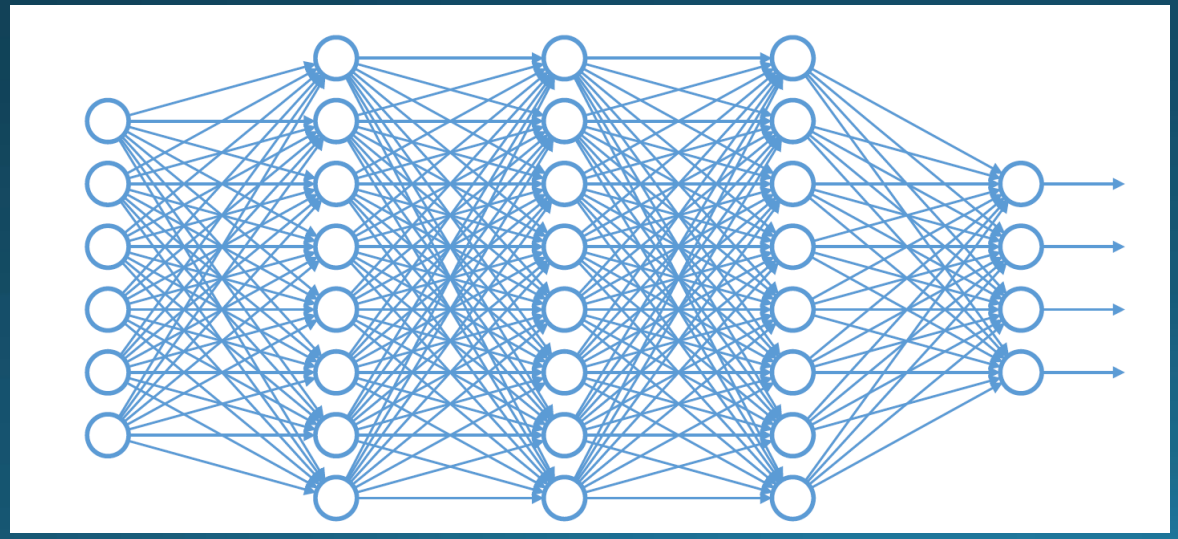
Types of Networks

Types

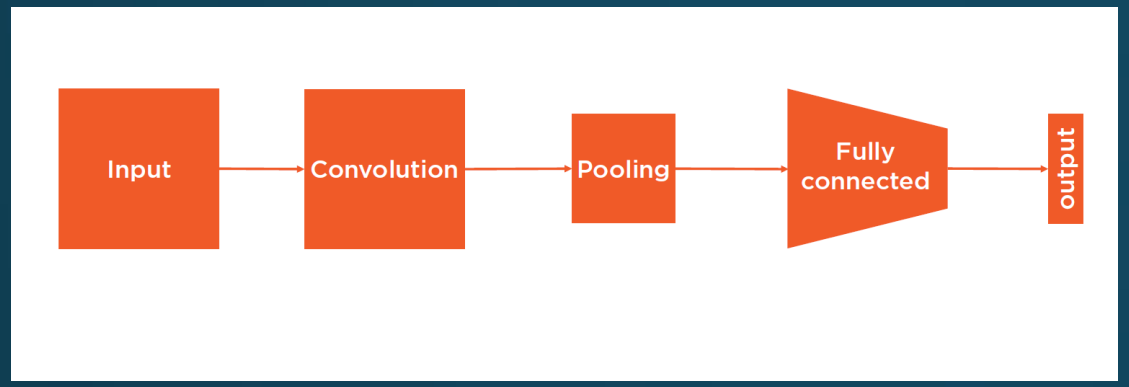
- Fully Connected Networks
- Convolutional Networks
- Recurrent Networks
- Generative Adversarial Networks
- Deep Reinforcement Learning

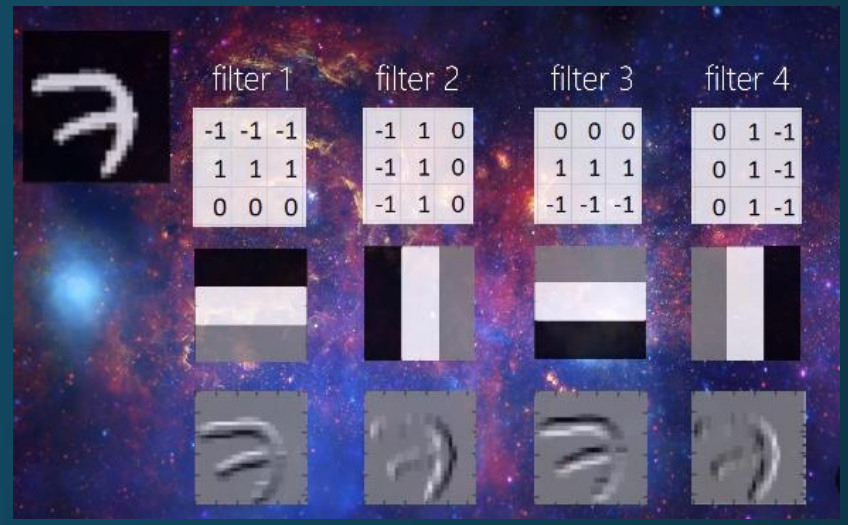
Full Connected Feed Forward Network

- Fully Connected:
  - Each neuron is connected to every neuron in the subsequent layer
- Feed Forward:
  - Neurons are only connected to neurons in a subsequent layer. There are no feed back loops.



Convolutional Neural Network



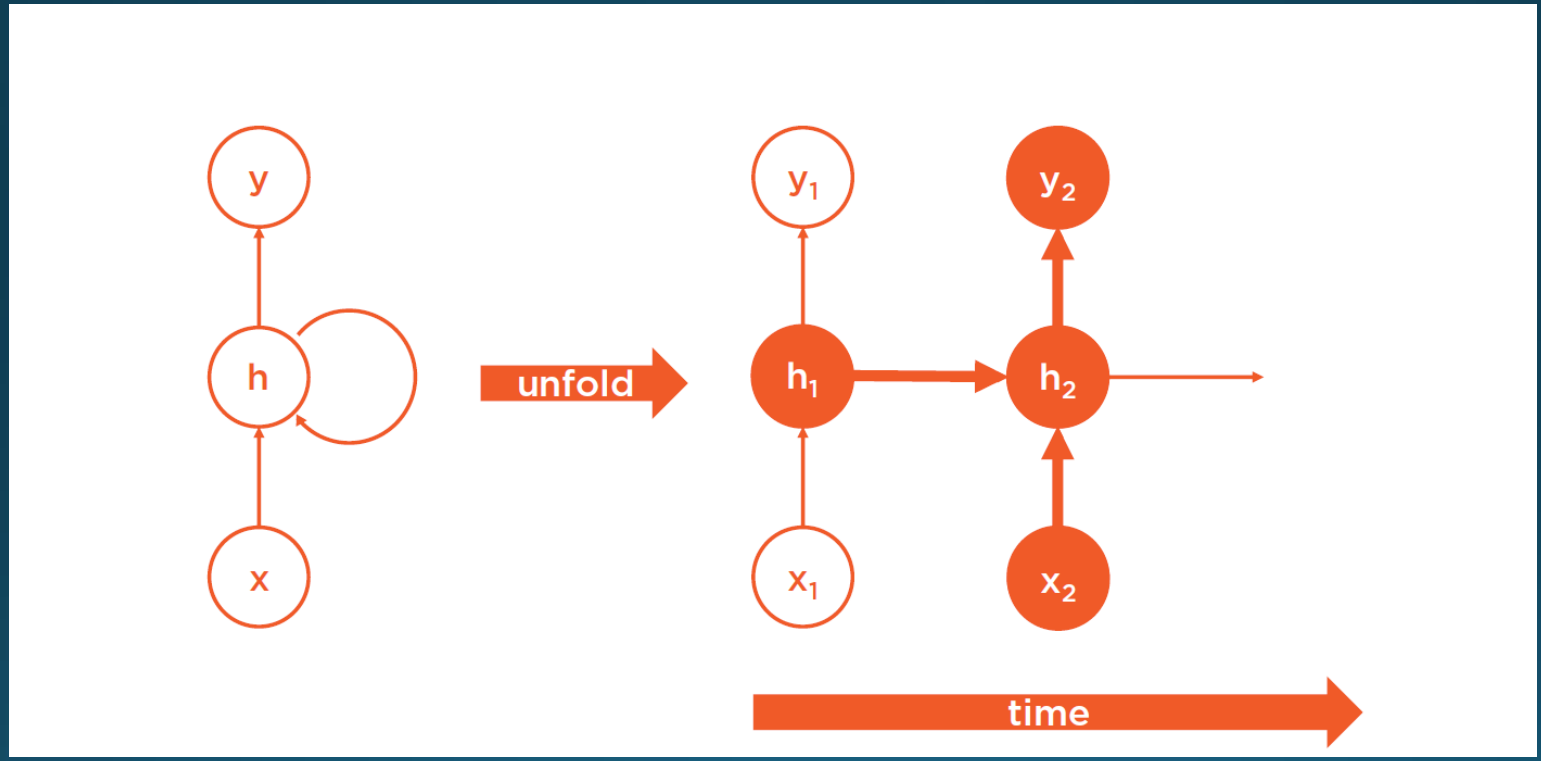


- Adding Hidden Layers and Neurons to a Fully Connected Network results in an exponential increase in system resources
- Convolution
  - Technique that allows us to extract visual features from an image in small chunks. Each neuron in a convolution layer is responsible for a small cluster of neurons in the preceding layer.
- Filter
  - Bounding box that defines the cluster of neurons. Also known as a kernel.
  - Filters help identify certain features of an image such as sharpen an image or detect edges.
- Pooling (also known as subsampling or downsampling)
  - Reduces the number of neurons in the previous layer while still retaining the most important information

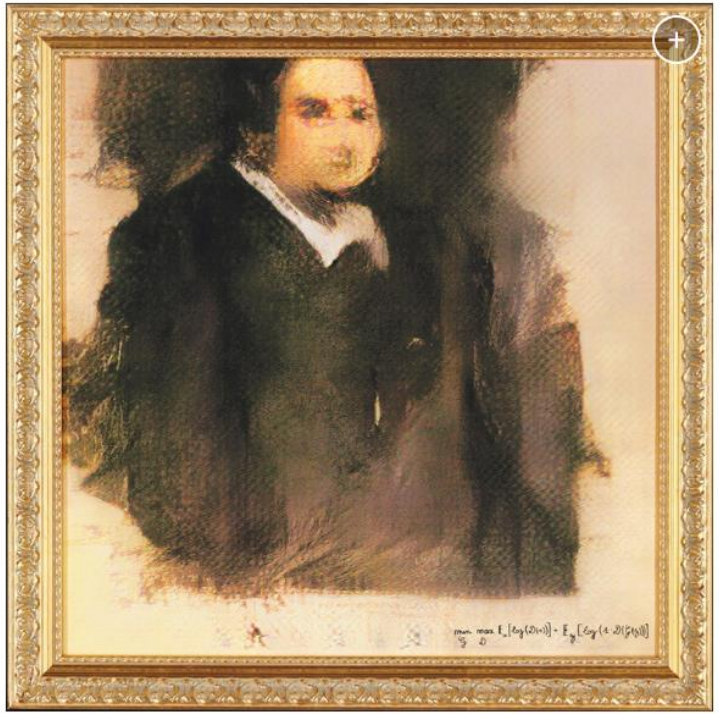
Convolutional Neural Networks

- Find application in
  - Image recognition
  - Image processing
  - Image segmentation
  - Video analysis
  - Language processing

Recurrent Neural Network

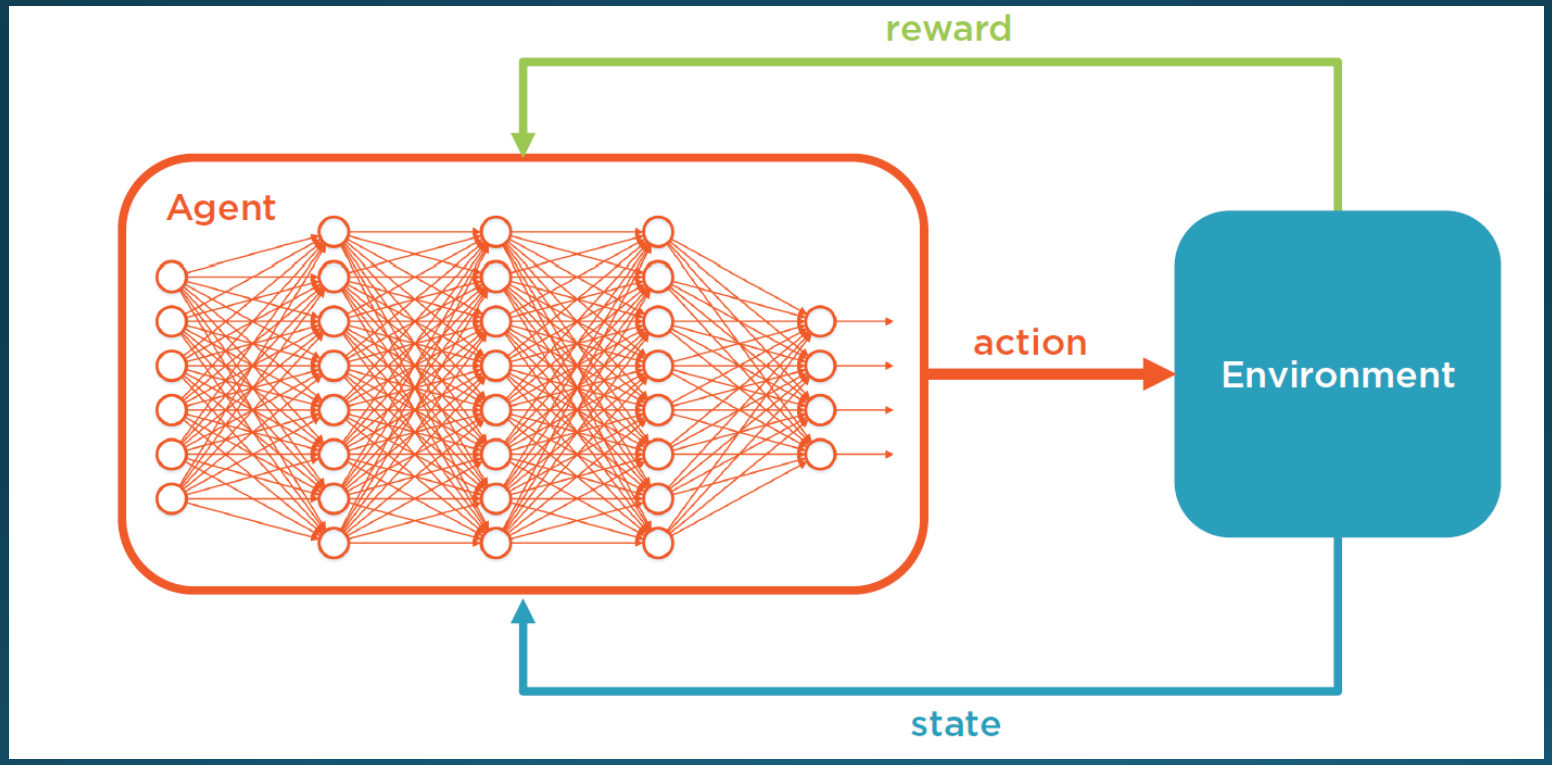


Generative Adversarial Network (GAN)



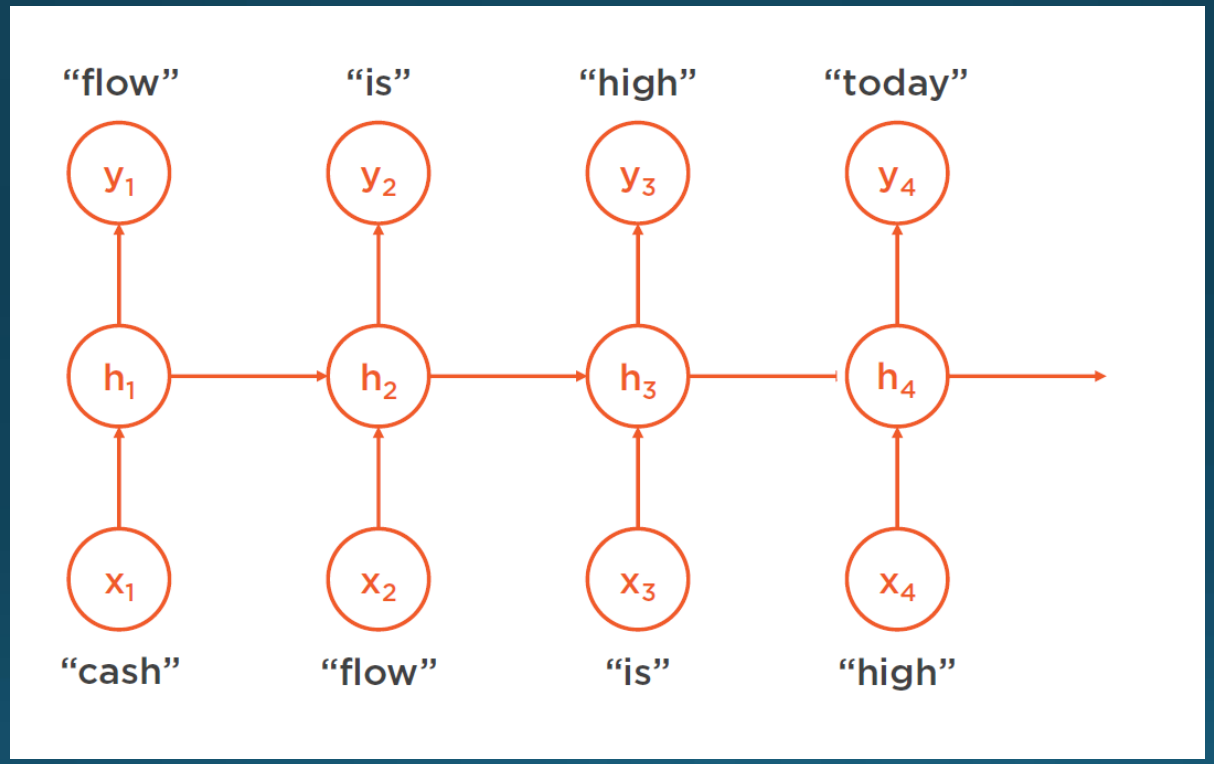
Portrait of Edmond Belamy, 2018, created by GAN (Generative Adversarial Network). Sold for $432,500 on 25 October at Christie's in New York. Image © Obvious

Deep Reinforcement Learning

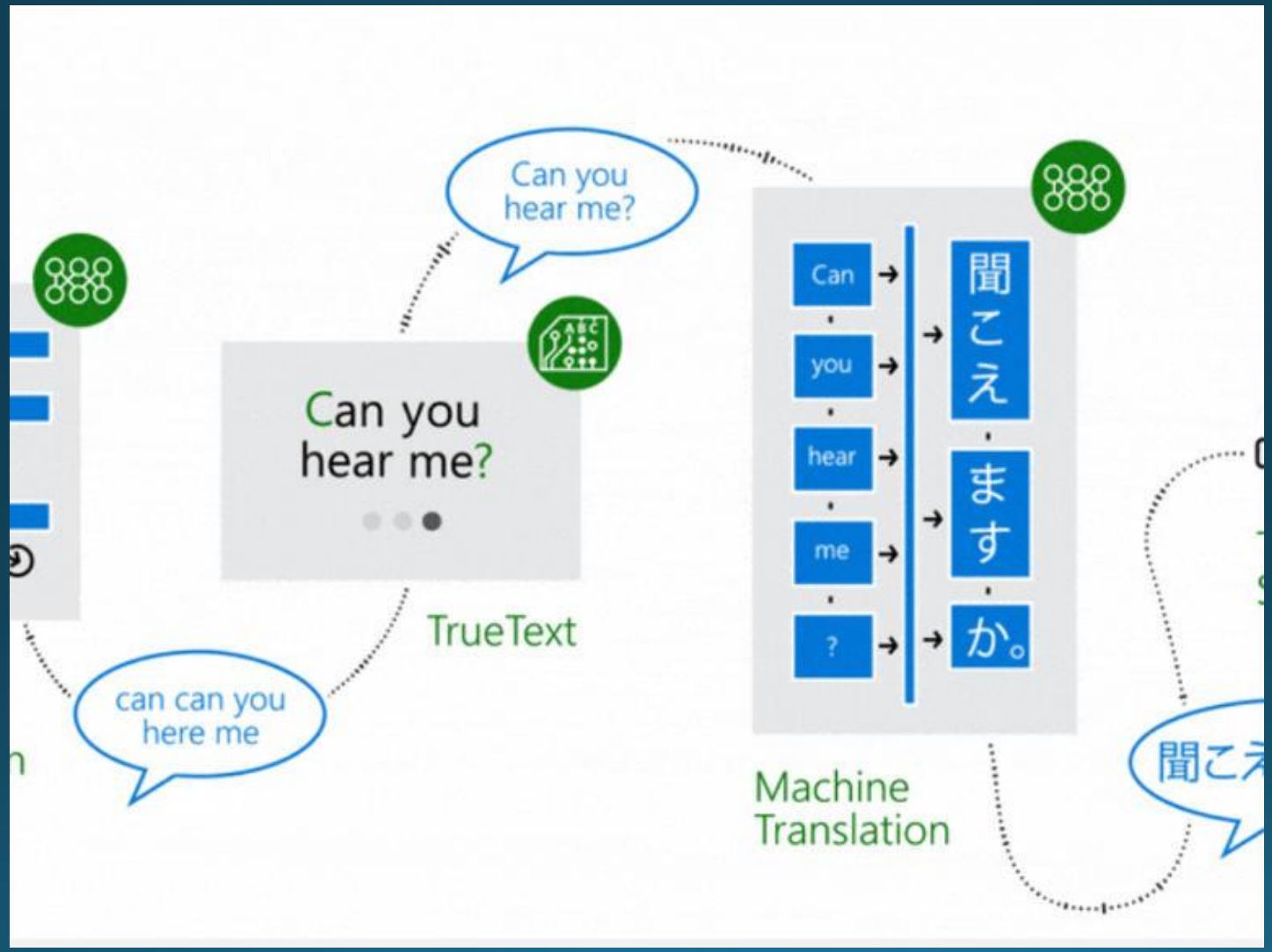


Applications

Sentence Completion



Translation

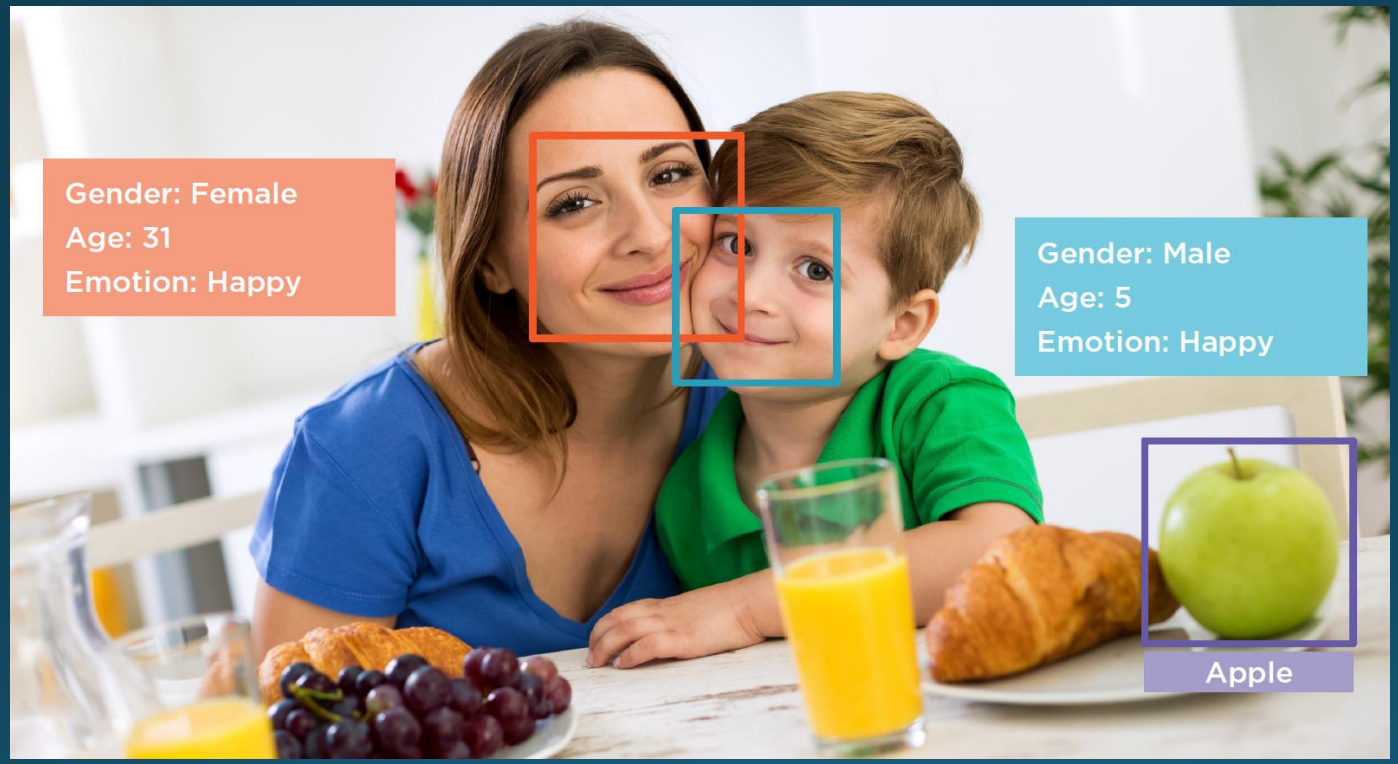


Summarizing

Article Text	Predicted Headline
At least 72 people died and scores more were hurt when a truck crowded with pilgrims plunged into a gorge in the desert state of Rajasthan on Friday, police told the press trust of India.	At least 72 dead in Indian road accident
One of the last remaining routes for Iraqis trying to flee their country has effectively been closed off by new visa restrictions imposed by Syria, the U. N. refugee agency said Tuesday.	U.N. refugee agency closes last routes to Iraq
Democratic presidential candidates said Thursday they would step up pressure on Pakistan's president Pervez Musharraf over democracy, and criticized White House policy towards Islamabad.	Democratic presidential hopefuls call for pressure on Musharraf

Original Source: https://nlp.stanford.edu/courses/cs224n/2015/reports/1.pdf

Auto-Tagging



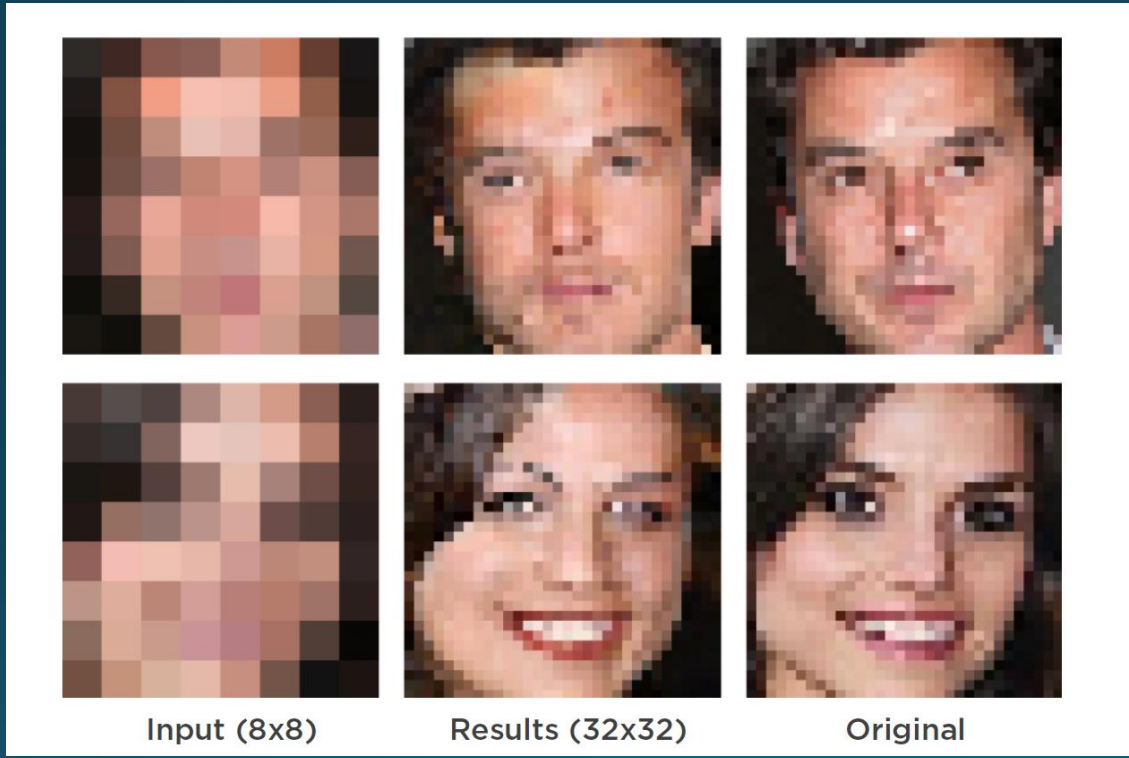
Summarizing Pictures

A person riding a motorcycle on a dirt road.

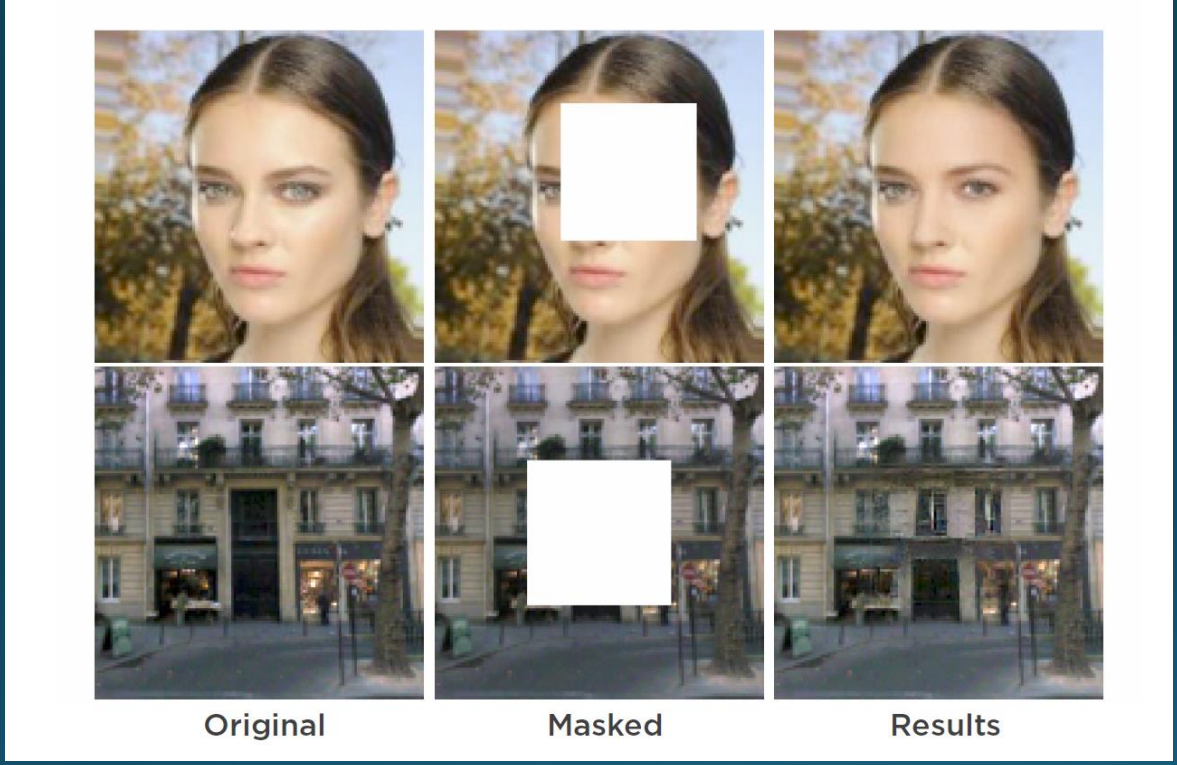
A group of young people playing a game of frisbee.

A herd of elephants walking across a dry grass field.

Filling in Pictures

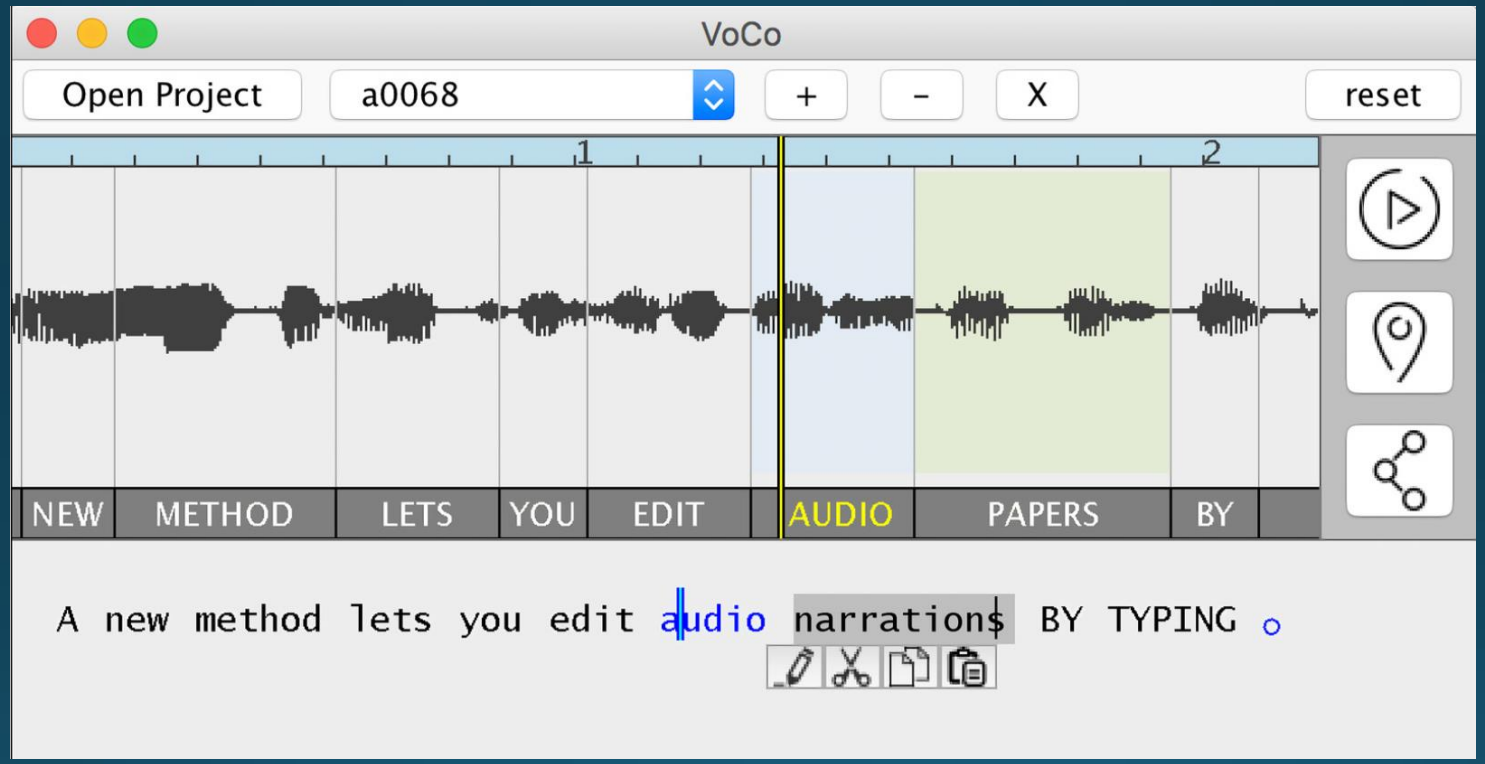


Completing Pictures

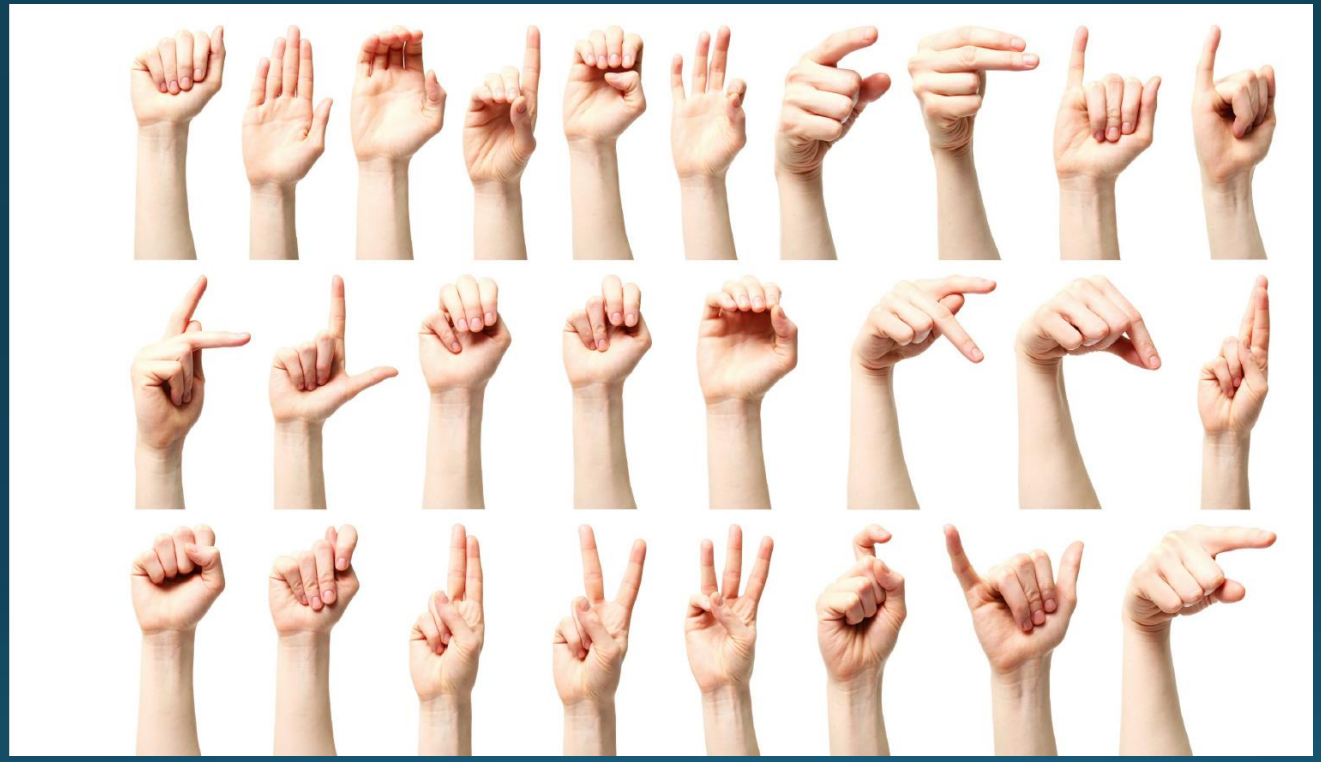


Do you recognize them?

Editing Audio

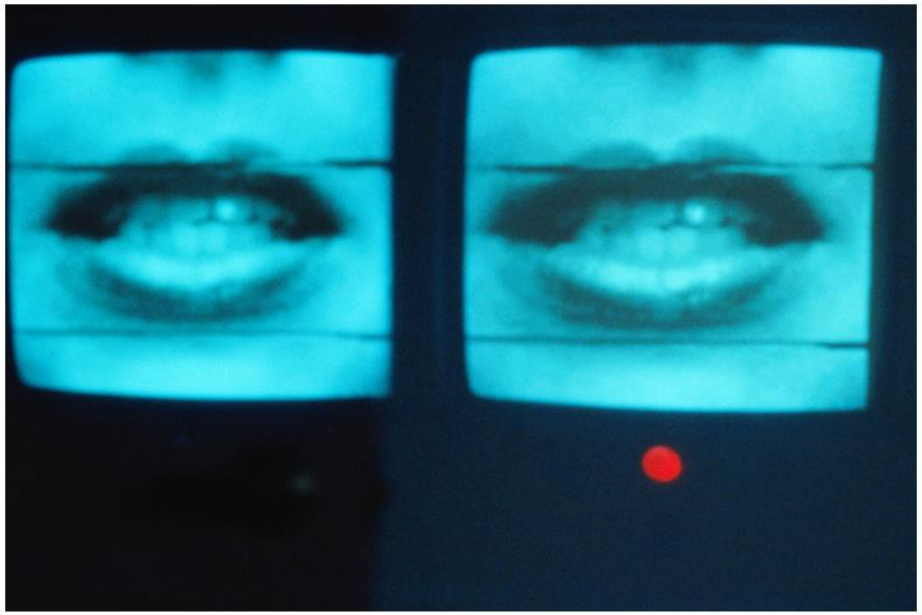


Sign Language



Lip Reading

Google's DeepMind AI can lip-read TV shows better than a pro



Using Deep Learning at Scale

Using Deep Learning with Large Datasets

- Deep Learning Services
- Deep Learning Platforms
- Deep Learning Frameworks

Deep Learning as a Service

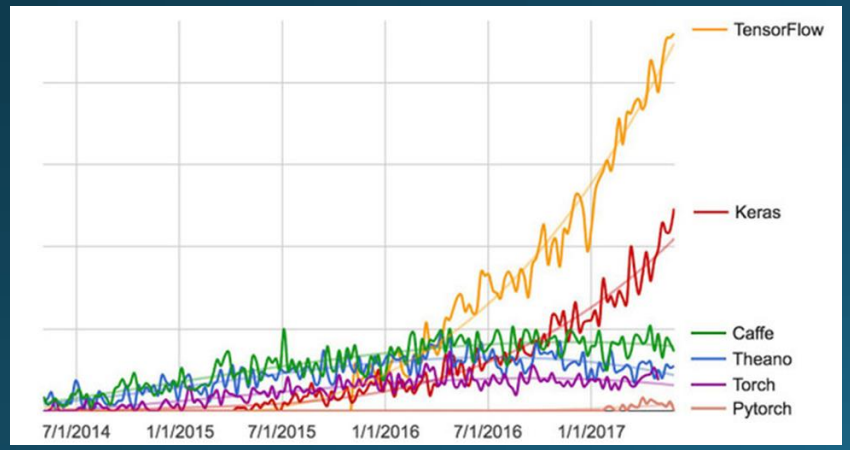
- Training Data
- Training Model
- Hosted on Server

Deep Learning Platforms

- You provide Data
- Training Model
- Hosted on Server

Deep Learning Frameworks

- Own Data
- Own Algorithm
- Host



TensorFlow Keras mxnet

Microsoft CNTK torch Caffe2

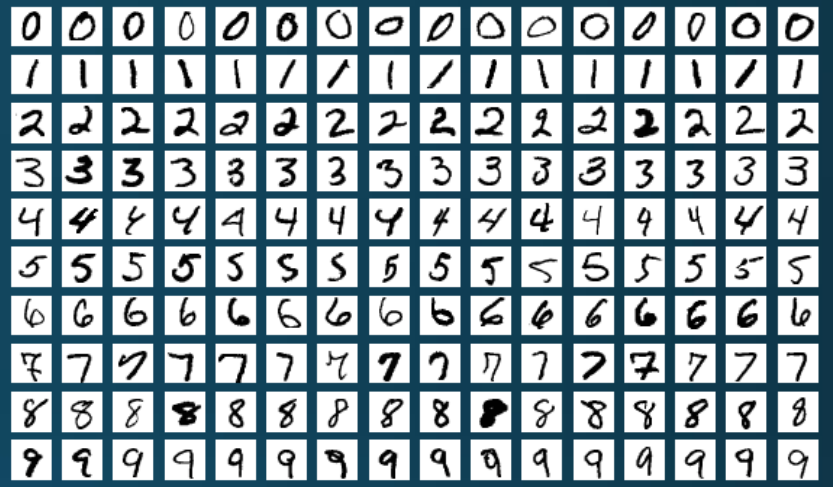
Illustration of Neural Networks on MNIST

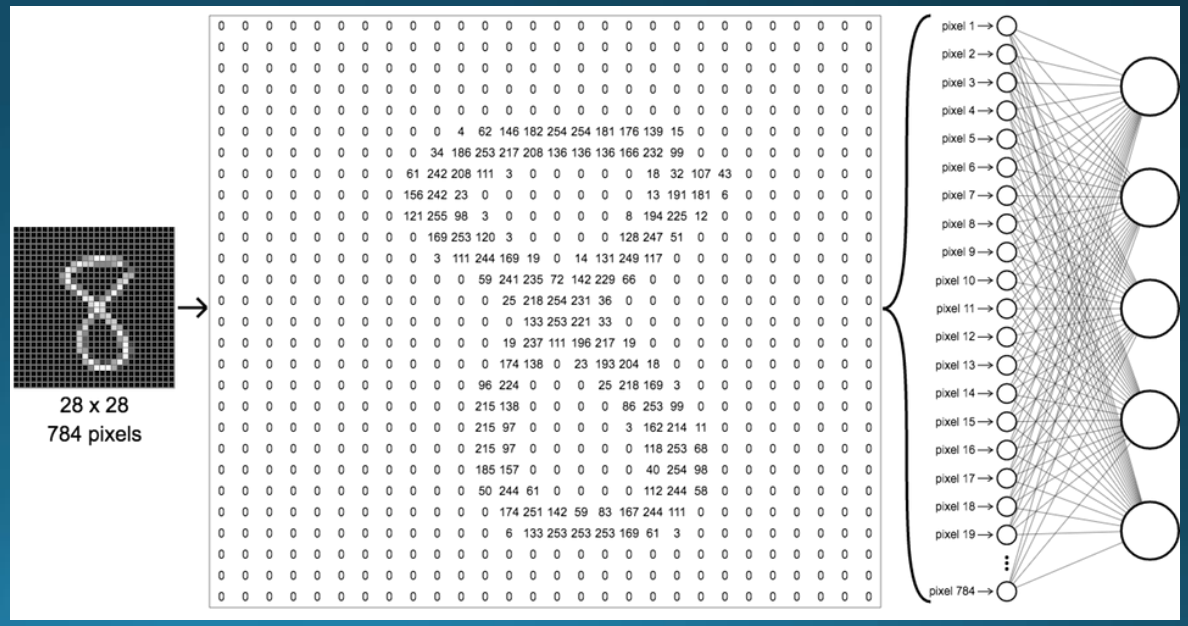
Tools

- R Packages
  - nnet
  - neuralnet
  - RSNNS
  - deepnet
  - darch
  - caret
  - RNN
  - Autoencoder
  - RcppDL
  - h2o
  - MXNetR ….
- Other General Frameworks
  - Caffe
  - Tensorflow
  - Theano
  - Torch
  - Deeplearning4j
  - CNTK

MNIST

- A large database of handwritten digits that is commonly used for training various image processing systems
- 42000 images.
- Each image is on a 28 x 28 pixel gray scale
- Thus, image has a 784 pixel descriptors





Source: Wikipedia and mc.ai

MNIST

- Basic
  - Hidden Layer(s): 1
  - Hidden Units or Neurons: 5
- Multiple Layers
  - Model A
    - Hidden Layer(s): 2
    - Hidden Units or Neurons: 5
  - Model B
    - Hidden Layer(s): 3
    - Hidden Units or Neurons: 10
  - Model C
    - Hidden Layer(s): 3
    - Hidden Units or Neurons: 50
  - Model D
    - Hidden Layer(s): 4
    - Hidden Units or Neurons: 10
- Specify More Hyper-Parameters
- Tune Hyper-Parameters Manually
- Hyper-Parameters Optimization
  - Use Random Search
  - Number of each of the following hyper-parameters explored
    - Activation Function: 6
    - Network architectures: 6
    - L1, L2
    - … etc

Neural Networks

Read MNIST Data

You must read the data before trying to run code on your own machine. To read data use the following code after setting your working directory. To set your working directory, modify the following to set the file path for the folder where the data file resides. setwd('c:/thatawesomeclass/neuralnetworks/)

```
digits = read.csv('mnist.csv')
```

Explore Data

```
dim(digits)
```

```
## [1] 42000   785
```

```
head(colnames(digits))
```

```
## [1] "label"  "pixel0" "pixel1" "pixel2" "pixel3" "pixel4"
```

```
tail(colnames(digits))
```

```
## [1] "pixel778" "pixel779" "pixel780" "pixel781" "pixel782" "pixel783"
```

```
head(digits[1:5,1:5])
```

label	pixel0	pixel1	pixel2	pixel3
1	0	0	0	0
0	0	0	0	0
1	0	0	0	0
4	0	0	0	0
0	0	0	0	0

```
digits$label[1:100]
```

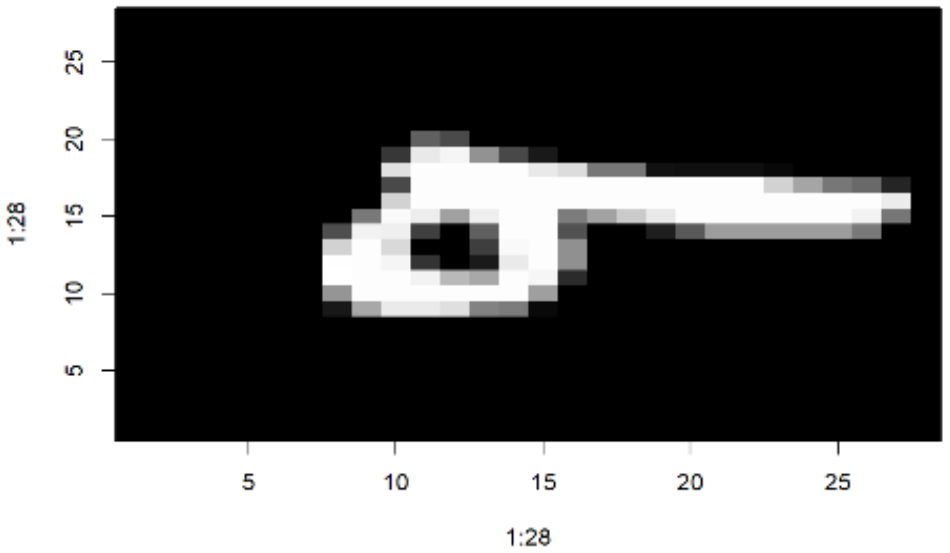
```
##   [1] 1 0 1 4 0 0 7 3 5 3 8 9 1 3 3 1 2 0 7 5 8 6 2 0 2 3 6 9 9 7 8 9 4 9 2 1 3
##  [38] 1 1 4 9 1 4 4 2 6 3 7 7 4 7 5 1 9 0 2 2 3 9 1 1 1 5 0 6 3 4 8 1 0 3 9 6 2
##  [75] 6 4 7 1 4 1 5 4 8 9 2 9 9 8 9 6 3 6 4 6 2 9 1 2 0 5
```

Prepare Data

```
digits$label = factor(digits$label,levels = 0:9)
set.seed(1031)
split = sample(x = c('train','validation','test'),size = nrow(digits),replace = T,prob = c(0.4,0.4,0.2))
train = digits[split=='train',]
validation = digits[split=='validation',]
test = digits[split=='test',]
```


See Digits

```
viewDigit = function(num,flip=F){
  img = matrix(as.numeric(train[num,2:785]),ncol=28,byrow = flip)
  return(image(x = 1:28,y = 1:28,z = img,col=gray((0:255)/255)))
}
viewDigit(101,flip = T)
```

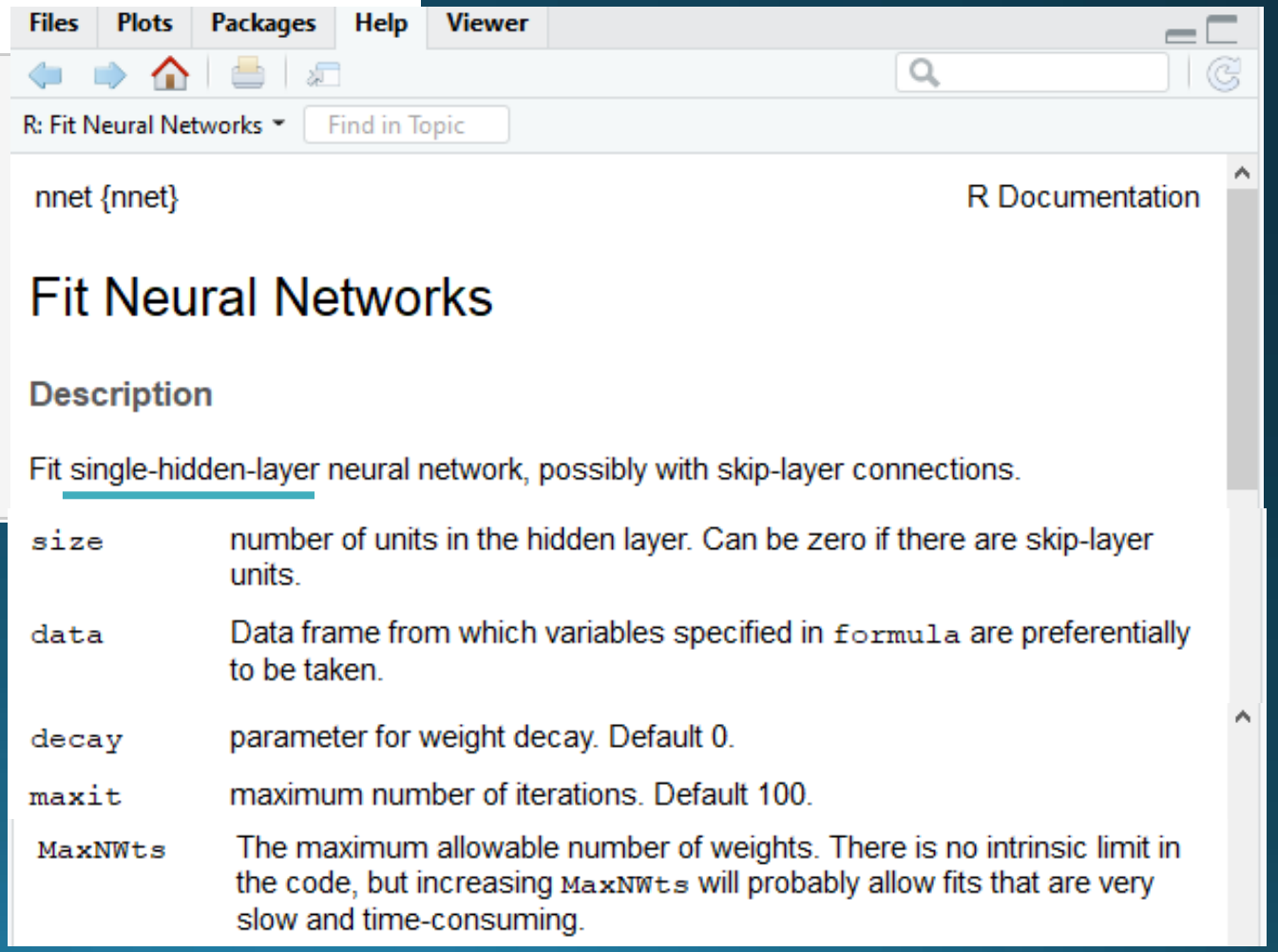


Basic Neural Network

Hidden Layer(s): 1
Hidden Units or Neurons: 5
Using library(nnet).
For more on library(nnet) see the documentation

- Using nnet package

```
library(nnet)
set.seed(1031)
model1 = nnet(label~.,
              data = train,
              size=5,
              decay=0.1,
              MaxNWts=10000,
              maxit=100)
```



```
## # weights:  3985
## initial  value 41246.477851
## iter  10 value 24114.010031
## iter  20 value 22327.480616
## iter  30 value 22022.594792
## iter  40 value 21789.544884
## iter  50 value 21254.902462
## iter  60 value 20826.286215
## iter  70 value 20413.541371
## iter  80 value 20311.221087
## iter  90 value 20096.969733
## iter 100 value 20004.010807
## final  value 20004.010807
## stopped after 100 iterations
```

Performance on Train sample

```
pred_train_prob = predict(model1)
pred_train_prob[1,]
```

```
##            0            1            2            3            4            5
## 4.334108e-09 9.533198e-01 1.787669e-02 1.376767e-02 3.512015e-07 7.769479e-03
##            6            7            8            9
## 5.957880e-11 2.652715e-04 6.894980e-03 1.057805e-04
```

```
sum(pred_train_prob[1,])
```

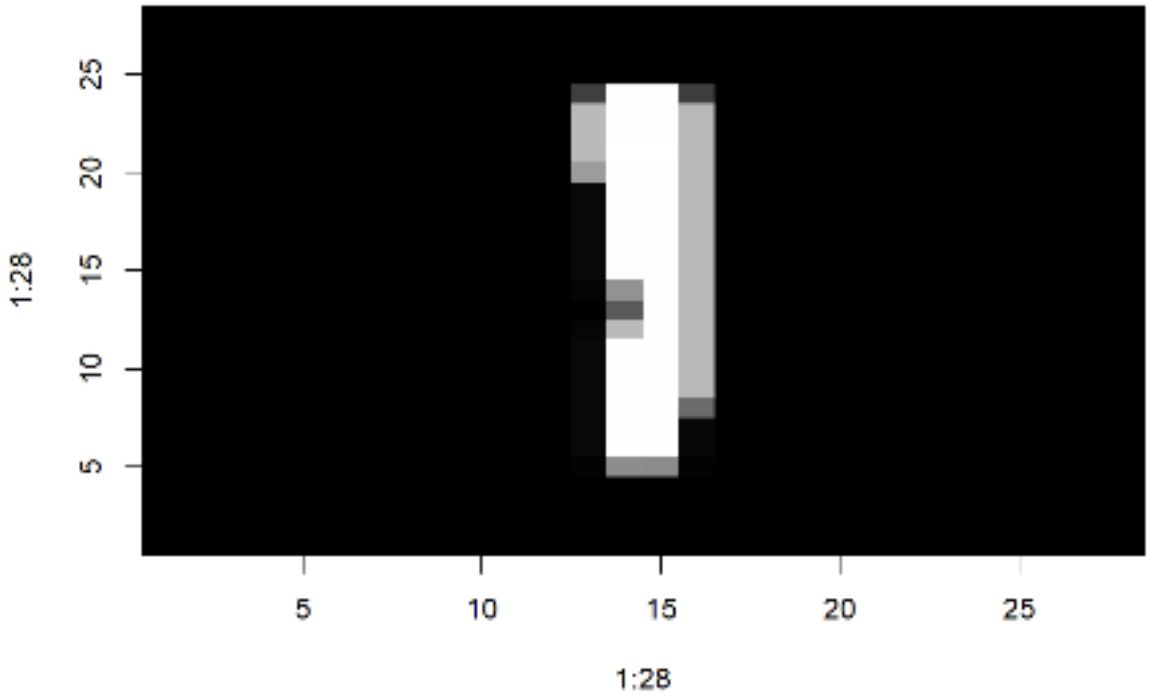
```
## [1] 1
```

```
pred_train = predict(model1,type='class')
train$label[1]
```

True Output

```
## [1] 1
## Levels: 0 1 2 3 4 5 6 7 8 9
```

```
viewDigit(1)
```




```
pred_train[1]
```

Predicted Output

```
## [1] "1"
```

```
table(train$label,pred_train)
```

```
##    pred_train
##        0    1    2    3    4    5    6    7    8    9
##   0 1312    3  163    6    3   36  112    0    2    5
##   1    3 1767   23   13    1   14    4   18   13    5
##   2  106  260 1000   39    8   30   88   22   26   57
##   3   16   61   34  901   31  533   46   25   36   68
##   4    2    7   35    2  608   15   19   35   10  908
##   5   13  115   46   41  113  971   80    5  124   62
##   6   51    7  193    7   35   89 1243    0   10   19
##   7    1   49   27   10   24   17    0 1433    4  161
##   8    3  170   77   50    6  346   49   10  865   73
##   9    2    7   36   17  334   18   15  212   17 1026
```

```
mean(pred_train==train$label) # Accuracy
```

```
## [1] 0.6617105
```

Performance on Validation sample

```
pred = predict(model1,newdata=validation,type='class')
table(validation$label,pred)
```

```
##    pred
##        0    1    2    3    4    5    6    7    8    9
##   0 1345    3  148    1    4   35  131    0    9    6
##   1    1 1817   20   22    0   18    1   18   20    2
##   2   96  256 1089   44    9   35   83   16   30   62
##   3   22   62   40  853   33  540   46   27   31   60
##   4    3    7   37    2  593    6   22   31    7  913
##   5   19  119   57   45  128  881   56    6  124   68
##   6   62    8  223    3   36  106 1229    0    6   17
##   7    1   49   32   14   24   22    4 1458    7  179
##   8    2  196   63   51   17  341   48    9  801   67
##   9    4    9   34   21  330   35   13  222   15  973
```

```
mean(pred == validation$label)
```

```
## [1] 0.653582
```

Performance on Test sample

```
pred = predict(model1,newdata=test,type='class')
table(test$label,pred)
```

```
##    pred
##        0   1   2   3   4   5   6   7   8   9
##   0  646   3  74   4   1  18  56   0   2   4
##   1    2 853  18   6   1  10   0   7   5   2
##   2   50 143 499  19   6  14  43   9  11  27
##   3    9  41  14 442  15 270  22  16  18  39
##   4    3   3  16   3 264   4  11  18   8 480
##   5   11  63  22  20  55 444  26   2  61  18
##   6   24   5  96   3   9  46 591   1   4  14
##   7    0  21  12   7  15   2   4 739   2  83
##   8    0 110  31  23   3 169  17   2 420  44
##   9    1   3  19  14 158  14   1 109   7 522
```

```
mean(pred == test$label)
```

```
## [1] 0.6533269
```

```
library(caret); confusionMatrix(factor(pred),test$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 646   2  50   9   3  11  24   0   0   1
##          1   3 853 143  41   3  63   5  21 110   3
##          2  74  18 499  14  16  22  96  12  31  19
##          3   4   6  19 442   3  20   3   7  23  14
##          4   1   1   6  15 264  55   9  15   3 158
##          5  18  10  14 270   4 444  46   2 169  14
##          6  56   0  43  22  11  26 591   4  17   1
##          7   0   7   9  16  18   2   1 739   2 109
##          8   2   5  11  18   8  61   4   2 420   7
##          9   4   2  27  39 480  18  14  83  44 522
##
## Overall Statistics
##
##                Accuracy : 0.6533
##                  95% CI : (0.643, 0.6636)
##     No Information Rate : 0.109
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6147
##
##  Mcnemar's Test P-Value : NA
```

```
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.79950   0.9436  0.60780  0.49887  0.32593  0.61496
## Specificity           0.98665   0.9470  0.95960  0.98664  0.96487  0.92778
## Pos Pred Value        0.86595   0.6851  0.62297  0.81701  0.50095  0.44803
## Neg Pred Value        0.97854   0.9928  0.95704  0.94275  0.92972  0.96194
## Prevalence            0.09740   0.1090  0.09896  0.10680  0.09764  0.08703
## Detection Rate        0.07787   0.1028  0.06015  0.05328  0.03182  0.05352
## Detection Prevalence  0.08992   0.1501  0.09655  0.06521  0.06352  0.11946
## Balanced Accuracy     0.89308   0.9453  0.78370  0.74276  0.64540  0.77137
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.74527  0.83503  0.51282  0.61557
## Specificity           0.97601  0.97787  0.98422  0.90454
## Pos Pred Value        0.76654  0.81838  0.78067  0.42336
## Neg Pred Value        0.97316  0.98025  0.94857  0.95384
## Prevalence            0.09559  0.10668  0.09872  0.10222
## Detection Rate        0.07124  0.08908  0.05063  0.06292
## Detection Prevalence  0.09294  0.10885  0.06485  0.14863
## Balanced Accuracy     0.86064  0.90645  0.74852  0.76005
```

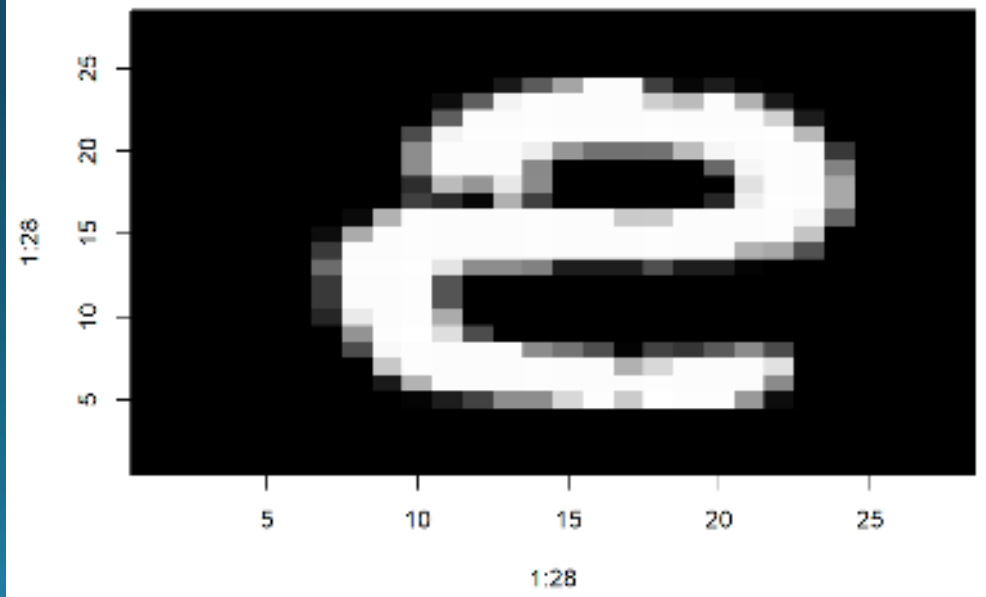
Digits that Model got wrong

Let us see if the human eye and get read what nnet failed at. Here is a list of all the digits that the model got wrong. Let us see the first digit the model got wrong.

```
# Digits that the model got wrong
which(pred!=test$label)
```

```
##    [1]    7   11   13   15   16   17   18   20   21   23   26   27   31   33
##   [15]   34   35   39   41   45   47   52   54   63   70   73   74   76   80
##   [29]   81   84   85   91   92  101  102  108  110  113  115  117  121  123
##   [43]  125  126  130  132  134  135  137  145  147  151  157  158  162  170
##   [57]  171  172  175  185  186  192  193  197  199  204  205  206  207  209
##   [71]  210  211  212  219  222  223  228  235  236  237  242  244  248  254
##   [85]  256  259  261  263  267  274  275  278  289  293  297  300  301  303
##   [99]  308  310  317  321  324  325  329  333  338  345  348  350  355  358
##  [113]  359  360  361  362  364  367  370  372  374  377  387  388  392  393
```

```
# Examine the second digit the model got wrong
viewDigit(11)
```



Neural Networks with Multiple Layers

We are going to explore a popular deep learning framework called h2o. For more on this, see the documentation. h2o is a powerful package useful for not only deep learning but also machine learning. It is designed for scalability and is more flexible than nnet.

Convert data to h2o objects

```
library(h2o)
h2o.init()
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     C:\Users\Kitty\AppData\Local\Temp\Rtmp8SDZ1W\file40907f393c79/h2o_Kitty_started_from_r.out
##     C:\Users\Kitty\AppData\Local\Temp\Rtmp8SDZ1W\file4090575a15af/h2o_Kitty_started_from_r.err
##
##
## Starting H2O JVM and connecting:  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         2 seconds 765 milliseconds
##     H2O cluster timezone:       America/New_York
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.36.0.3
##     H2O cluster version age:    1 month and 7 days
##     H2O cluster name:           H2O_started_from_R_Kitty_bxn282
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   7.90 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.1.2 (2021-11-01)
```

- Using h2O package

COLUMBIA UNIVERSITY
School of Professional Studies

```
train_h2o = as.h2o(train)
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
validation_h2o = as.h2o(validation)
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
test_h2o = as.h2o(test)
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

Single Layer Network with h20

This is simply a replication of the single layer network created before using nnet()

Hidden Layer: 1

Hidden Units: 5

```
model2 = h2o.deeplearning(x=2:785,
                          y = 1,
                          training_frame = train_h2o,
                          hidden = c(5),
                          seed=1031)
```

```
##
  |
  |                                                                      |   0%
  |
  |==================================                                    |  50%
  |
  |=============================================================         |  90%
  |
  |======================================================================| 100%
```

Performance on validation sample


```
pred = h2o.predict(model2,newdata = validation_h2o)
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
mean(pred[1]==validation_h2o$label) # Accuracy
```

```
## [1] 0.7415038
```

```
h2o.confusionMatrix(model2,validation_h2o) # Using a built-in function
```

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	1444	3	142	11	0	44	26	4	7	1	0.1414982	238 / 1,682
1	42	1605	159	9	2	7	1	16	77	1	0.1636269	314 / 1,919
2	56	54	1385	93	5	10	17	47	52	1	0.1947674	335 / 1,720
3	5	8	330	1196	3	46	26	42	45	13	0.3022170	518 / 1,714
4	6	1	100	8	1212	79	1	9	42	163	0.2523134	409 / 1,621
5	61	15	373	91	18	800	17	18	15	95	0.4677312	703 / 1,503
6	150	15	130	194	1	12	1132	18	37	1	0.3301775	558 / 1,690
7	1	30	41	71	11	77	3	1464	30	62	0.1821229	326 / 1,790
8	77	76	196	60	60	33	29	10	1016	38	0.3630094	579 / 1,595
9	9	2	26	16	131	99	7	66	30	1270	0.2330918	386 / 1,656
Totals	1851	1809	2882	1749	1443	1207	1259	1694	1351	1645	0.2584962	4,366 / 16,890

Performance on test sample


```
pred = h2o.predict(model2,newdata = test_h2o)
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
mean(pred[1]==test_h2o$label) # Accuracy
```

```
## [1] 0.7268563
```

```
h2o.confusionMatrix(model2,test_h2o) # Using a built-in function
```

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	690	4	62	10	0	24	13	3	2	0	0.1460396	118 / 808
1	28	732	93	2	2	5	0	11	31	0	0.1902655	172 / 904
2	23	39	622	56	4	5	4	29	39	0	0.2423873	199 / 821
3	4	6	142	600	4	42	10	35	32	11	0.3227991	286 / 886
4	6	0	46	6	590	43	0	6	28	85	0.2716049	220 / 810
5	35	9	181	49	9	385	6	10	5	33	0.4667590	337 / 722
6	72	11	61	80	2	9	538	11	9	0	0.3215637	255 / 793
7	3	10	22	35	7	31	0	721	20	36	0.1853107	164 / 885
8	52	28	105	31	33	16	11	6	518	19	0.3675214	301 / 819
9	7	2	13	8	79	43	0	49	13	634	0.2523585	214 / 848
Totals	920	841	1347	877	730	603	582	881	697	818	0.2731437	2,266 / 8,296

Neural Network with Two Hidden Layer

Hidden Layers: 2

Hidden Units: 5 in each layer

```
model3 = h2o.deeplearning(x=2:785,
                          y = 1,
                          training_frame = train_h2o,
                          hidden = c(5,5),
                          seed=1031)
```

```
##
  |
  |                                                                      |   0%
  |
  |==============                                                        |  20%
  |
  |============================================                          |  70%
  |
  |======================================================================| 100%
```

Performance on Validation Sample

```
pred = h2o.predict(model3,validation_h2o)
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
mean(pred[1]==validation_h2o$label) # Accuracy
```

```
## [1] 0.8309059
```

```
h2o.confusionMatrix(model3,validation_h2o) # Using a built-in function
```

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	1557	2	9	9	1	58	36	5	4	1	0.0743163	125 / 1,682
1	4	1813	7	34	13	7	9	2	23	7	0.0552371	106 / 1,919
2	17	36	1303	171	23	41	13	6	108	2	0.2424419	417 / 1,720
3	12	31	167	1322	7	112	4	47	8	4	0.2287048	392 / 1,714
4	2	6	2	18	1422	16	41	1	34	79	0.1227637	199 / 1,621
5	20	5	39	46	36	1153	57	45	98	4	0.2328676	350 / 1,503
6	109	7	3	7	32	57	1449	0	20	6	0.1426036	241 / 1,690
7	5	21	7	106	16	11	4	1520	6	94	0.1508380	270 / 1,790
8	8	62	126	45	41	86	58	7	1124	38	0.2952978	471 / 1,595
9	3	1	0	25	78	24	27	117	10	1371	0.1721014	285 / 1,656
Totals	1737	1984	1663	1783	1669	1565	1698	1750	1435	1606	0.1690941	2,856 / 16,890

Neural Network with Three Hidden Layers

Hidden Layers: 3

Hidden Units: 10 in each layer

```
model4 = h2o.deeplearning(x=2:785,
                          y = 1,
                          training_frame = train_h2o,
                          hidden = c(10,10,10),
                          seed=1031)
```

```
##
  |
  |                                                                      |   0%
  |
  |=======                                                               |  10%
  |
  |=================================                                     |  50%
  |
  |===================================================                   |  80%
  |
  |======================================================================| 100%
```

Performance on Validation Sample

```
pred = h2o.predict(model4,validation_h2o)
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
mean(pred[1]==validation_h2o$label) # Accuracy
```

```
## [1] 0.9071048
```

```
h2o.confusionMatrix(model4,validation_h2o)
```

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	1618	0	11	2	1	27	11	2	10	0	0.0380499	64 / 1,682
1	0	1872	10	9	1	4	3	5	11	4	0.0244919	47 / 1,919
2	5	21	1529	51	16	15	23	30	27	3	0.1110465	191 / 1,720
3	10	6	39	1502	0	92	2	27	27	9	0.1236873	212 / 1,714
4	4	4	17	5	1510	5	11	12	9	44	0.0684762	111 / 1,621
5	10	10	6	43	9	1344	38	12	24	7	0.1057884	159 / 1,503
6	20	4	24	3	8	69	1539	1	22	0	0.0893491	151 / 1,690
7	1	8	19	13	12	12	0	1682	1	42	0.0603352	108 / 1,790
8	14	56	13	46	33	72	23	11	1296	31	0.1874608	299 / 1,595
9	5	6	4	28	64	10	0	88	22	1429	0.1370773	227 / 1,656
Totals	1687	1987	1672	1702	1654	1650	1650	1870	1449	1569	0.0928952	1,569 / 16,890

Neural Network with Three Hidden Layers

Hidden Layers: 3
Hidden Units: 50 in each layer

```
model5 = h2o.deeplearning(x=2:785,
                          y = 1,
                          training_frame = train_h2o,
                          hidden = c(50,50,50),
                          seed=1031)
```

Performance on Validation Sample

```
pred = h2o.predict(model5,validation_h2o)
```

```
mean(pred[1]==validation_h2o$label) # Accuracy
```

```
## [1] 0.9462996
```

```
h2o.confusionMatrix(model5,validation_h2o)
```

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	1643	1	3	2	3	5	15	4	3	3	0.0231867	39 / 1,682
1	0	1887	3	9	0	2	2	5	8	3	0.0166754	32 / 1,919
2	8	9	1597	38	18	4	6	29	7	4	0.0715116	123 / 1,720
3	6	7	19	1616	1	17	0	14	19	15	0.0571762	98 / 1,714
4	9	4	7	2	1532	3	13	13	6	32	0.0549044	89 / 1,621
5	7	4	2	41	8	1387	14	8	12	20	0.0771790	116 / 1,503
6	10	2	2	0	15	13	1641	1	3	3	0.0289941	49 / 1,690
7	4	11	11	10	6	1	0	1724	4	19	0.0368715	66 / 1,790
8	5	30	11	36	5	22	16	11	1439	20	0.0978056	156 / 1,595
9	8	6	2	26	27	5	2	61	2	1517	0.0839372	139 / 1,656
Totals	1700	1961	1657	1780	1615	1459	1709	1870	1503	1636	0.0537004	907 / 16,890

Neural Network with Four Hidden Layers

Hidden Layers: 4
Hidden Units: 10 in each layer

```
model6 = h2o.deeplearning(x=2:785,
                          y = 1,
                          training_frame = train_h2o,
                          hidden = c(10,10,10,10),
                          seed=1031)
```

Performance on Validation Sample

```
pred = h2o.predict(model6,validation_h2o)
```

```
mean(pred[1]==validation_h2o$label) # Accuracy
```

```
## [1] 0.9027827
```

```
h2o.confusionMatrix(model6,validation_h2o)
```

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	1620	0	7	5	4	11	10	4	9	12	0.0368609	62 / 1,682
1	1	1821	10	16	0	17	1	13	40	0	0.0510683	98 / 1,919
2	17	17	1511	43	20	13	32	23	31	13	0.1215116	209 / 1,720
3	7	5	36	1523	1	61	4	15	43	19	0.1114352	191 / 1,714
4	7	2	22	1	1455	1	26	2	20	85	0.1024059	166 / 1,621
5	8	7	7	82	4	1306	31	1	50	7	0.1310712	197 / 1,503
6	20	2	11	3	14	52	1572	3	8	5	0.0698225	118 / 1,690
7	6	28	16	9	4	1	5	1622	11	88	0.0938547	168 / 1,790
8	5	23	12	61	14	57	11	12	1379	21	0.1354232	216 / 1,595
9	13	2	11	35	53	8	5	71	19	1439	0.1310386	217 / 1,656
Totals	1704	1907	1643	1778	1569	1527	1697	1766	1610	1689	0.0972173	1,642 / 16,890

Fun Exercise

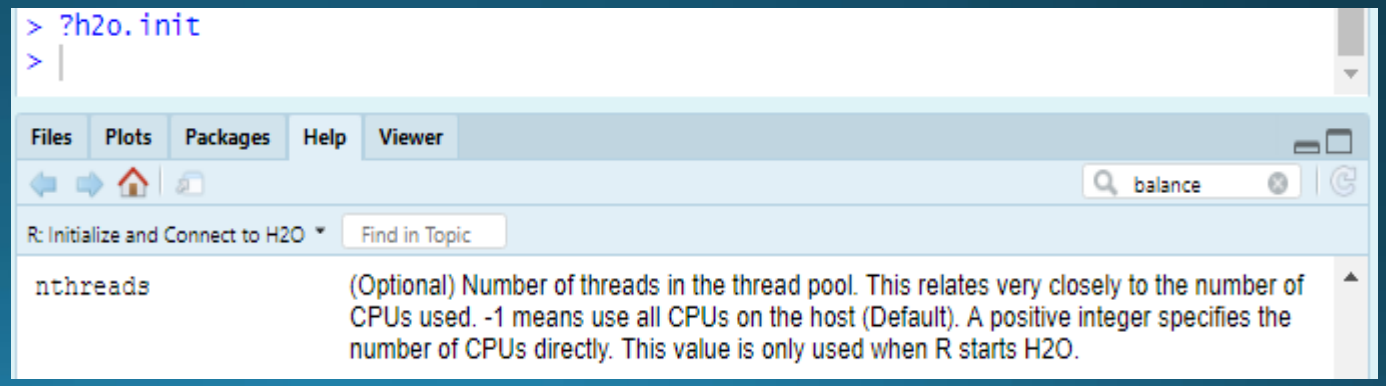
Try out Network architectures with different number of hidden layers and hidden units.

Specify more Hyperparameters

Hidden Layers: 2
Hidden Units: 10 in each layer
Activation Function: ReLU with dropout
Multi-threaded process

```
h2o.init(nthreads = -1) # Use all CPUs
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         1 minutes 10 seconds
##     H2O cluster timezone:       America/New_York
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.36.0.3
##     H2O cluster version age:    1 month and 7 days
##     H2O cluster name:           H2O_started_from_R_Kitty_bxn282
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   7.77 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.1.2 (2021-11-01)
```



```
model7 = h2o.deeplearning(x=2:785,
                          y=1,
                          training_frame=train_h2o,
                          activation = 'RectifierWithDropout',
                          input_dropout_ratio = 0.05,
                          balance_classes = TRUE,
                          hidden=c(10,10),
                          hidden_dropout_ratios = c(0.05,0.05),
                          epochs = 10,
                          seed=1031)
```

Performance on Validation Sample

```
pred = h2o.predict(model7,validation_h2o)
```

```
mean(pred[1]==validation_h2o$label) # Accuracy
```

```
## [1] 0.8953227
```

```
> ?h2o.deeplearning
>
```

Files | Plots | Packages | Help | Viewer

R: Build a Deep Neural Network model using CPUs — Find in Topic

input_dropout_ratio	Input layer dropout ratio (can improve generalization, try 0.1 or 0.2). Defaults to 0.
hidden_dropout_ratios	Hidden layer dropout ratios (can improve generalization), specify one value per hidden layer, defaults to 0.5.
epochs	How many times the dataset should be iterated (streamed), can be fractional. Defaults to 10.
balance_classes	Logical. Balance training data class counts via over/under-sampling (for imbalanced data). Defaults to FALSE.
activation	Activation function. Must be one of: "Tanh", "TanhWithDropout", "Rectifier", "RectifierWithDropout", "Maxout", "MaxoutWithDropout". Defaults to Rectifier.

```
h2o.confusionMatrix(model7,validation_h2o)
```

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	1620	3	10	6	5	13	14	2	6	3	0.0368609	62 / 1,682
1	0	1855	8	12	4	13	1	5	21	0	0.0333507	64 / 1,919
2	27	25	1474	69	17	7	38	20	29	14	0.1430233	246 / 1,720
3	10	17	58	1498	0	65	4	9	37	16	0.1260210	216 / 1,714
4	5	13	10	1	1467	0	35	10	12	68	0.0950031	154 / 1,621
5	19	30	10	99	28	1216	19	5	66	11	0.1909514	287 / 1,503
6	26	10	14	3	11	25	1576	1	17	7	0.0674556	114 / 1,690
7	3	30	25	14	15	0	1	1649	3	50	0.0787709	141 / 1,790
8	17	38	18	40	27	57	12	4	1348	34	0.1548589	247 / 1,595
9	12	7	1	30	79	6	4	72	26	1419	0.1431159	237 / 1,656
Totals	1739	2028	1628	1772	1653	1402	1704	1777	1565	1622	0.1046773	1,768 / 16,890

Tune Hyperparameters manually

Hidden Layers: 3
Hidden Units: 10 in each layer
Activation Function: Hyperbolic tangent with dropout
Multi-threaded process

```
h2o.init(nthreads = -1) # Use all CPUs
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         1 minutes 21 seconds
##     H2O cluster timezone:       America/New_York
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.36.0.3
##     H2O cluster version age:    1 month and 7 days
##     H2O cluster name:           H2O_started_from_R_Kitty_bxn282
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   7.78 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.1.2 (2021-11-01)
```

```
model8 = h2o.deeplearning(x=2:785,
                          y=1,
                          training_frame=train_h2o,
                          activation = 'TanhWithDropout',
                          input_dropout_ratio = 0.1,
                          balance_classes = TRUE,
                          hidden=c(10,10,10),
                          hidden_dropout_ratios = c(0.2,0.2,0.2),
                          epochs = 10,
                          seed=1031)
```

Performance on Validation Sample

Without domain knowledge or experience with similar problems, this can take a long time.

```
pred = h2o.predict(model8,validation_h2o)
```

```
mean(pred[1]==validation_h2o$label) # Accuracy
```

```
## [1] 0.6723505
```

```
h2o.confusionMatrix(model8,validation_h2o)
```

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	1449	6	9	160	10	10	28	2	7	1	0.1385256	233 / 1,682
1	4	1877	11	8	0	0	3	7	1	8	0.0218864	42 / 1,919
2	79	390	962	38	21	0	154	23	6	47	0.4406977	758 / 1,720
3	149	119	22	1208	4	15	8	135	20	34	0.2952159	506 / 1,714
4	12	7	41	1	444	2	19	27	2	1066	0.7260950	1,177 / 1,621
5	568	66	22	280	36	386	48	50	7	40	0.7431803	1,117 / 1,503
6	74	50	66	1	9	2	1473	3	3	9	0.1284024	217 / 1,690
7	5	75	8	2	43	26	1	1388	1	241	0.2245810	402 / 1,790
8	65	160	19	193	44	24	49	21	824	196	0.4833856	771 / 1,595
9	6	11	13	20	76	21	2	161	1	1345	0.1878019	311 / 1,656
Totals	2411	2761	1173	1911	687	486	1785	1817	872	2987	0.3276495	5,534 / 16,890

Hyperparameters Optimization

Using Random search to automatically tune Hyperparameters. Implementing random search using H2o which comes with a set of utility functions. h2o.grid() is

- very general and adaptive
- efficiently and distributed implementtation of random search
- simple, requiring just a few lines of code

Specify hyper-parameters to examine

```
hyper_parameters = list(activation=c('Rectifier','Tanh','Maxout',
                                     'RectifierWithDropout','TanhWithDropout',
                                     'MaxoutWithDropout'),
                        hidden=list(c(20,20),c(50,50),c(100,100,100)
                                    ,c(30,30,30),c(50,50,50,50),c(25,25,25,25)),
                        l1=seq(0,1e-4,1e-6),
                        l2=seq(0,1e-4,1e-6))
```

```
> ?h2o.deeplearning
>
```

Files | Plots | Packages | Help | Viewer

R: Build a Deep Neural Network model using CPUs ▾ Find in Topic

l1	L1 regularization (can add stability and improve generalization, causes many weights to become 0). Defaults to 0.
l2	L2 regularization (can add stability and improve generalization, causes many weights to be small. Defaults to 0.

Specify search criteria


```
search_criteria = list(strategy='RandomDiscrete',
                       max_runtime_secs=360,
                       max_models=100,
                       stopping_rounds=5,
                       stopping_tolerance=1e-2,
                       stopping_metric='logloss')
                       # stop when logloss does not improve by more than 1%
```

search_criteria	(Optional) List of control parameters for smarter hyperparameter search.

the search will stop when logloss does not improve by more than 5 scoring events

Tune

```
h2o.grid(algorithm='deeplearning',
                grid_id='dl_grid_random',
                training_frame = train_h2o,
                validation_frame=validation_h2o,
                x=2:785,
                y=1,
                epochs=10,
                seed=1031,
                hyper_params = hyper_parameters,
                search_criteria = search_criteria)
```

Examine performance of hyper-parameters tested

```
grid = h2o.getGrid("dl_grid_random",sort_by="logloss",decreasing=FALSE)
grid
```

```
## H2O Grid Details
## ================
##
## Grid ID: dl_grid_random
## Used hyper parameters:
##   -  activation
##   -  hidden
##   -  l1
##   -  l2
## Number of models: 10
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by increasing logloss
##              activation           hidden      l1      l2
## 1  RectifierWithDropout  [100, 100, 100] 0.00003 0.00005
## 2               Maxout          [20, 20] 0.00009 0.00001
## 3       TanhWithDropout  [100, 100, 100] 0.00001 0.00006
## 4       TanhWithDropout          [20, 20] 0.00000 0.00002
## 5  RectifierWithDropout [25, 25, 25, 25] 0.00004 0.00007
## 6     MaxoutWithDropout          [50, 50] 0.00000 0.00010
## 7     MaxoutWithDropout  [100, 100, 100] 0.00006 0.00003
## 8     MaxoutWithDropout          [50, 50] 0.00005 0.00001
## 9       TanhWithDropout [25, 25, 25, 25] 0.00001 0.00006
## 10    MaxoutWithDropout          [20, 20] 0.00008 0.00009
##                  model_ids logloss
## 1   dl_grid_random_model_5 0.22094
## 2   dl_grid_random_model_2 0.25575
## 3   dl_grid_random_model_1 0.39737
## 4   dl_grid_random_model_3 0.83853
## 5   dl_grid_random_model_4 1.13984
## 6   dl_grid_random_model_6 1.31656
## 7   dl_grid_random_model_9 1.39691
## 8  dl_grid_random_model_10 1.40637
## 9   dl_grid_random_model_7 2.04131
## 10  dl_grid_random_model_8 2.43274
```

```
grid@summary_table[1,]
```

activation	hidden	l1	l2	model_ids	logloss
RectifierWithDropout	[100, 100, 100]	2.7e-05	5.1e-05	dl_grid_random_model_5	0.2209446

Performance of best model on validation sample

```
best_model <- h2o.getModel(grid@model_ids[[1]])
# model with lowest logloss (on validation, since it was available during training)
h2o.confusionMatrix(best_model,valid=T)
```

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	1650	0	1	2	4	5	14	1	4	1	0.0190250	32 / 1,682
1	0	1871	13	8	2	7	1	5	10	2	0.0250130	48 / 1,919
2	9	6	1598	18	15	9	15	18	30	2	0.0709302	122 / 1,720
3	7	4	50	1548	0	49	3	13	30	10	0.0968495	166 / 1,714
4	3	5	8	2	1517	7	17	4	7	51	0.0641579	104 / 1,621
5	14	4	3	48	8	1371	14	3	28	10	0.0878244	132 / 1,503
6	15	5	5	1	8	22	1630	0	4	0	0.0355030	60 / 1,690
7	8	10	17	8	10	3	1	1695	0	38	0.0530726	95 / 1,790
8	9	23	7	23	8	51	8	4	1447	15	0.0927900	148 / 1,595
9	11	4	1	23	50	17	0	48	12	1490	0.1002415	166 / 1,656
Totals	1726	1932	1703	1681	1622	1541	1703	1791	1572	1619	0.0635287	1,073 / 16,890

Hyper-parameters of best model

```
best_params <- best_model@allparameters
best_params
```

```
## $model_id
## [1] "dl_grid_random_model_5"
##
## $training_frame
## [1] "train_sid_aac4_1"
##
## $validation_frame
## [1] "validation_sid_aac4_3"
##
## $nfolds
## [1] 0
##
## $keep_cross_validation_models
## [1] TRUE
##
```

```
best_params$activation
```

```
## [1] "RectifierWithDropout"
```

```
best_params$hidden
```

```
## [1] 100 100 100
```

```
best_params$l1
```

```
## [1] 2.7e-05
```

```
best_params$l2
```

```
## [1] 5.1e-05
```

Performance of best model on test sample

Finally, how does performance of the best model compare to previous models.

```
h2o.confusionMatrix(best_model,newdata=test_h2o)
```

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	785	0	2	4	1	3	9	2	2	0	0.0284653	23 / 808
1	0	876	10	1	1	2	0	3	10	1	0.0309735	28 / 904
2	5	5	755	10	11	1	7	12	14	1	0.0803898	66 / 821
3	1	0	22	798	0	30	2	8	16	9	0.0993228	88 / 886
4	1	0	2	1	761	6	12	4	2	21	0.0604938	49 / 810
5	6	4	5	20	5	656	8	2	8	8	0.0914127	66 / 722
6	6	2	5	1	7	11	755	1	5	0	0.0479193	38 / 793
7	2	13	10	3	3	2	4	830	0	18	0.0621469	55 / 885
8	7	12	11	10	4	22	7	6	732	8	0.1062271	87 / 819
9	4	3	2	12	36	3	0	33	9	746	0.1202830	102 / 848
Totals	817	915	824	860	829	736	804	901	798	812	0.0725651	602 / 8,296

Accuracy

```
pred = h2o.predict(best_model,newdata = test_h2o)
```

```
mean(pred[,1]==test_h2o$label)
```

```
## [1] 0.9274349
```

Summary

- This module addressed the following topics
  - Introduction to Neural Networks
  - Artificial Neuron
  - Multiple Layer Neural Networks
  - Network Architecture
  - Types of Networks
  - Applications
  - Using Deep Learning at Scale
  - Illustration of Neural Networks on MNIST