



ITU



MACHINE LEARNING WITH PYTHON FOR SPACE WEATHER APPLICATIONS

by ITU Upper Atmosphere and Space Weather Laboratory

Lecture 4: Artificial Neural Networks

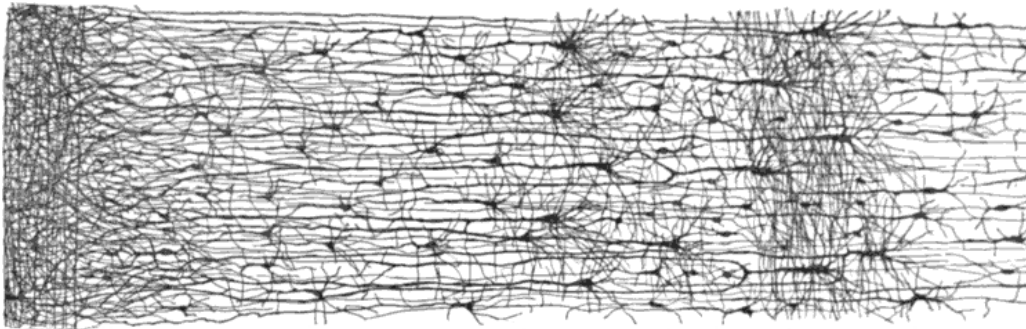
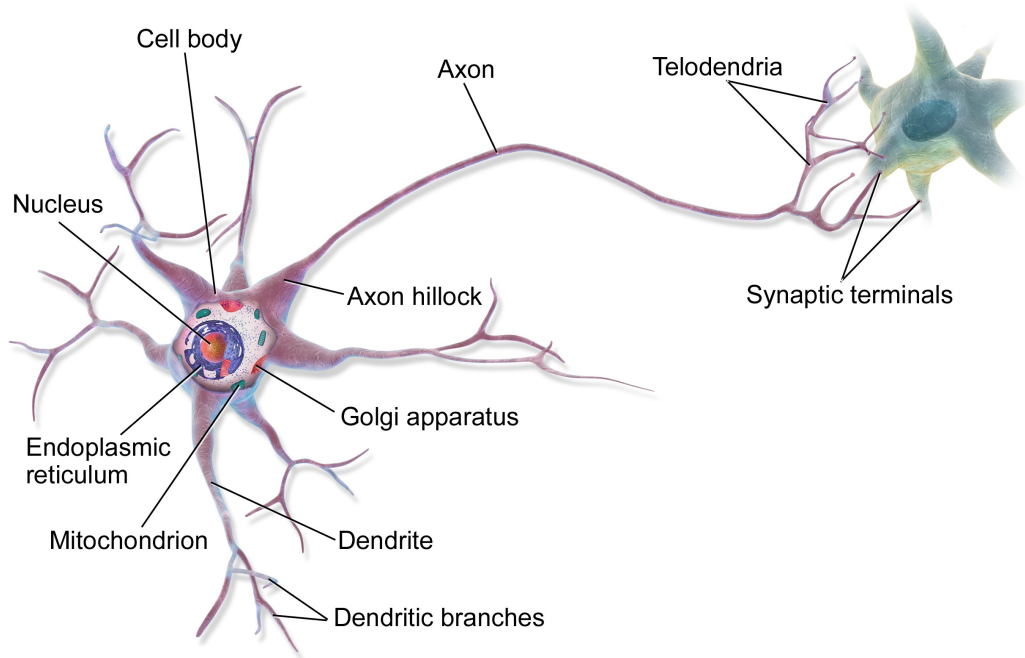


Instructor: Dogacan Su Ozturk

Contact: dsozturk@alaska.edu

11-12 May 2022 at Upper Atmosphere and Space Weather Laboratory

Neurons and Neural Networks



- A neuron is composed of a cell body, branching extensions called dendrites, and a long extension called axon.
- Axon splits into many branches called telodendria at its extremity.
- A neuron is connected to other neurons around its telodendria through its synaptic terminals.
- Biological neurons produce short electrical impulses called action potentials that travel through axons.
- As the signal travels along the axon it releases neurotransmitters necessary for communication between different neurons.
- A threshold/sufficient amount is necessary for other neurons to pick up the signal.

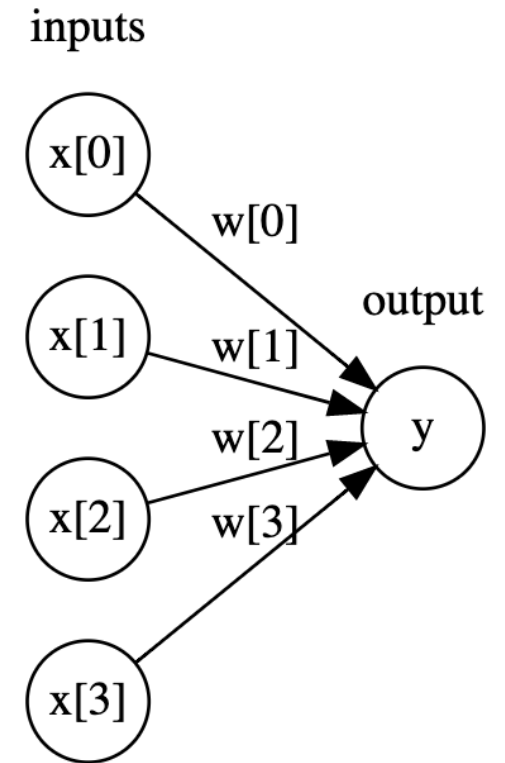
Remember Linear Regression

Remember the linear regression formula:

$$\hat{y} = w[0]x[0] + w[1]x[1] + \cdots + w[n]x[n] + c$$

If we were to visualize this relationship:

- Where each arrow defines a relationship between an input feature and the output, y .
- The relationships are prioritized based on the weights, w .
- The output, y , becomes a “weighted” sum of these relations.



The artificial neuron: Perceptron

- The artificial neuron has one or more binary inputs and one binary output.
- The artificial neuron only activates its output when a threshold is satisfied, just like the biological neuron having enough neurotransmitters.
- They can then mimic complex logical thinking
- In ML an artificial neuron is imitated through something called a threshold logic unit.
- It computes a weighted sum of its input, then applies a step function to the sum and outputs the result.
- What does this type of learning remind you of?
- A single layer of TLUs makes up a perceptron

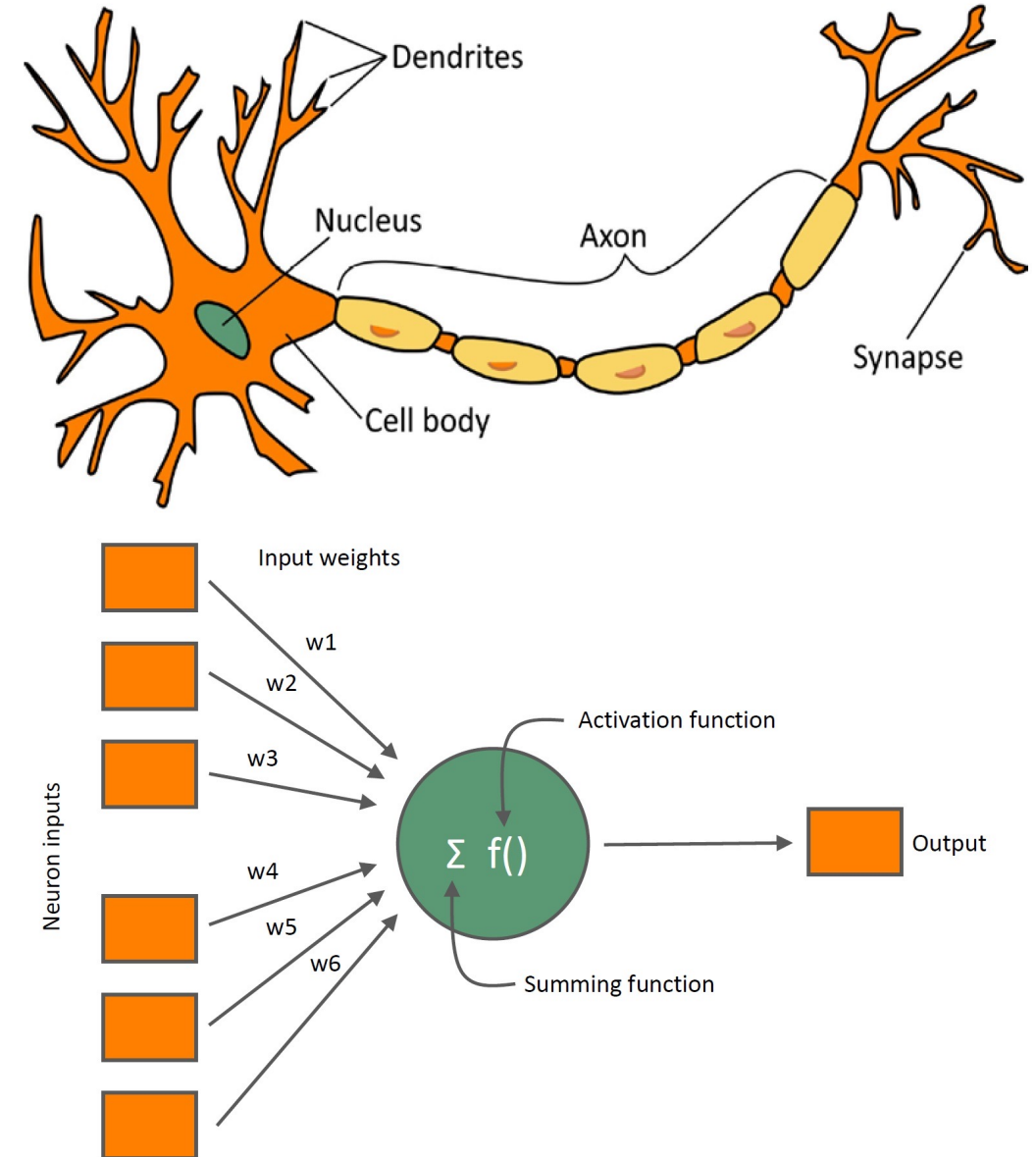
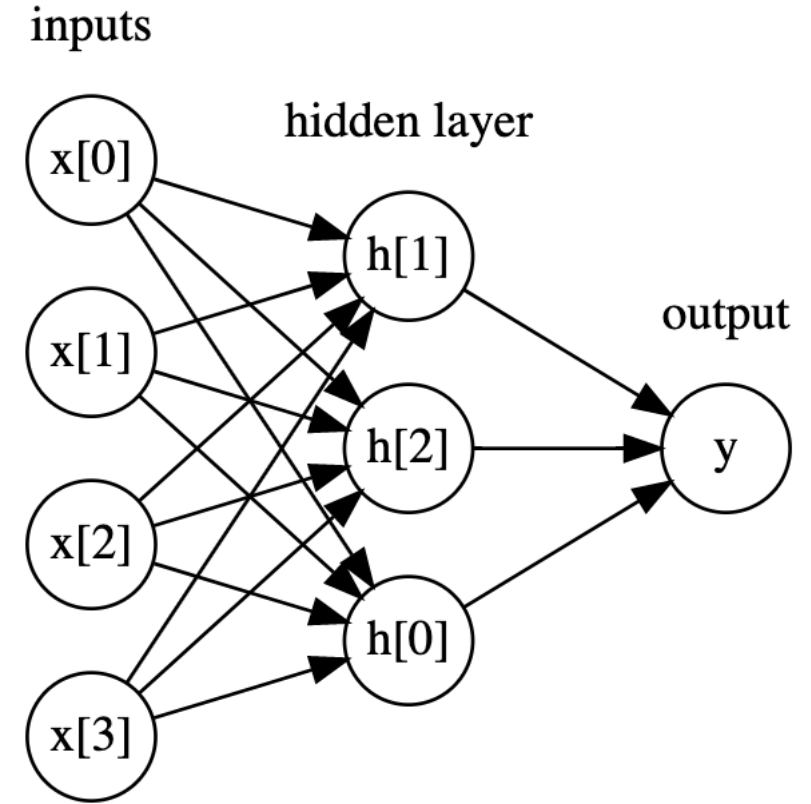


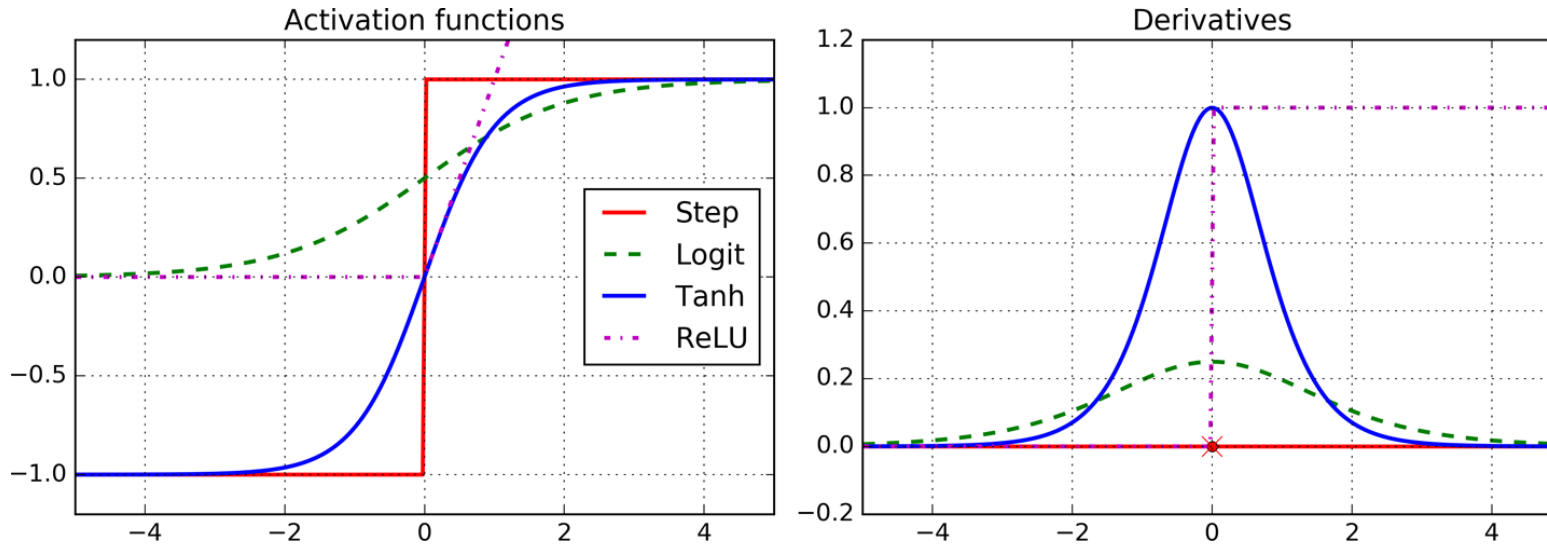
Image Credit: Francois Marais, John Thompson

A single layer perceptron model

- The neural network is when there is an extra layer added between input and output.
- The model now needs to repeat computing sums.
- There are now more coefficients (weights) and hidden layers.
- How do we make this “smarter” than a linear regression model?
- We add a decision making process, using “non-linearity”.
- These are called activation functions.



Activation Functions

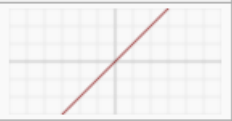





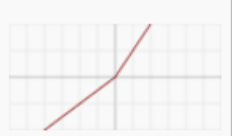
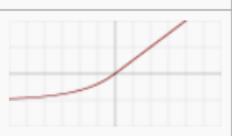
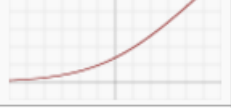


Historically, step function was used for decision making, but this resulted in problematic error propagations. Can you tell why?

Commonly used activation functions are:

- Logit/Sigmoid function: $\sigma(z) = \frac{1}{1+e^{-z}}$
- The hyperbolic tangent function: $\tanh(z)$
- Rectified Linear Unit (ReLU) function: $\max(0, z)$

Activation Functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Activation function cheat sheet
Courtesy of Sagar Sharma

A single layer perceptron model

- The formula for the relationship between the inputs and output:

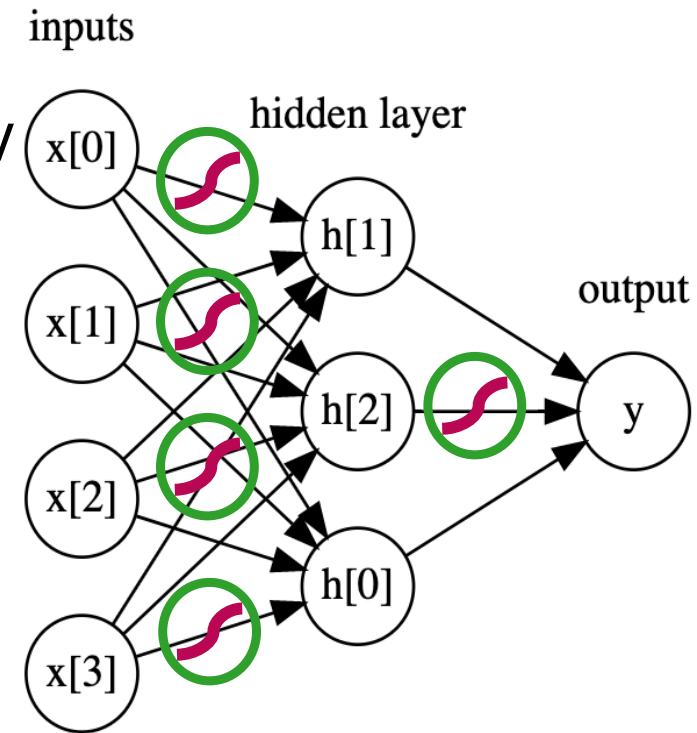
$$h[0] = \tanh(w[0,0] * x[0] + w[1,0] * x[1] + w[2,0] * x[2] + w[3,0] * x[3] + b[0])$$

$$h[1] = \tanh(w[0,1] * x[0] + w[1,1] * x[1] + w[2,1] * x[2] + w[3,1] * x[3] + b[1])$$

$$h[2] = \tanh(w[0,2] * x[0] + w[1,2] * x[1] + w[2,2] * x[2] + w[3,2] * x[3] + b[2])$$

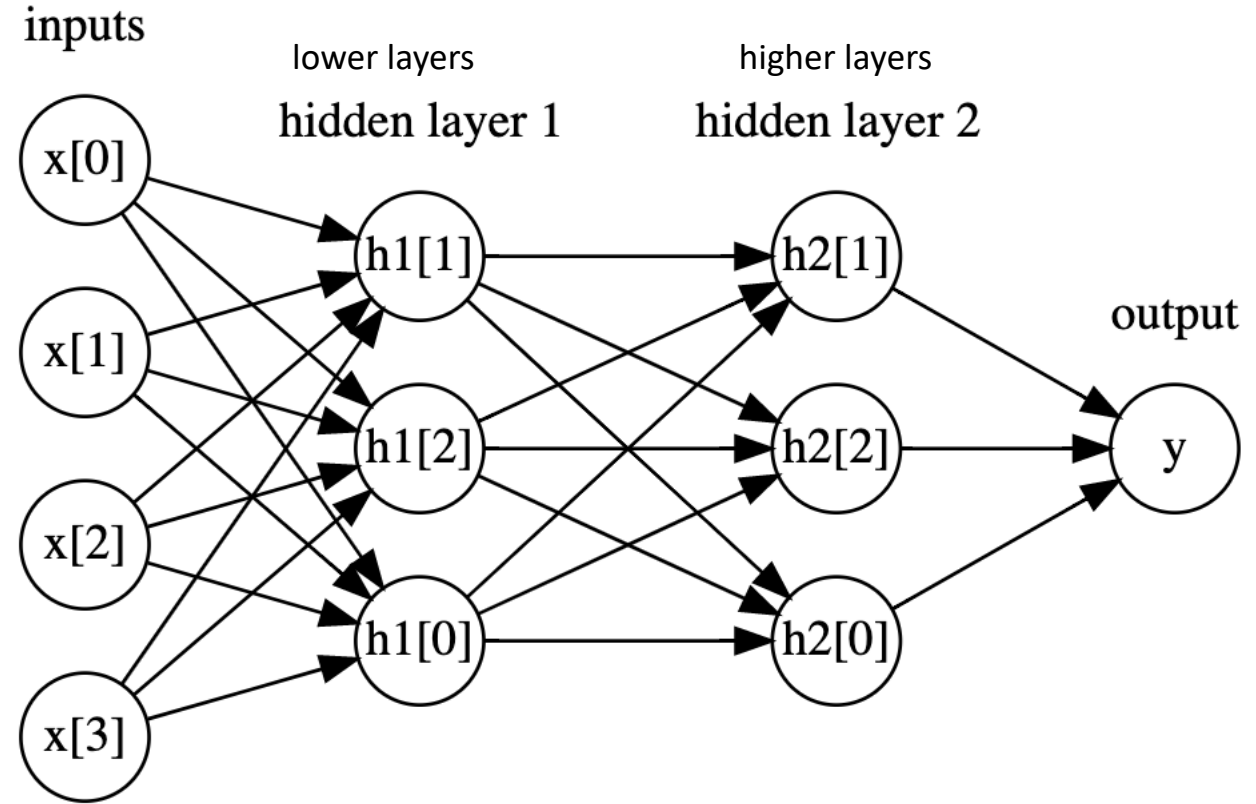
$$\hat{y} = v[0] * h[0] + v[1] * h[1] + v[2] * h[2] + b$$

- w are the weights between x and hidden layer, whereas v are the weights between hidden layer and y.
- The user needs to determine how many nodes and how many hidden layers are necessary.
- The user also needs to define which activation function would be most suitable for the problem.
- The last thing user needs to select is the solver.



A multi layer perceptron model

- An MLP composed of one input layer, one or more layers of TLUs called hidden layers, and an output layer.
- What does this type of learning remind you of?
- What does this learning look like without activation functions?



How do NNs learn?

After the weights are distributed randomly, an output value is calculated.

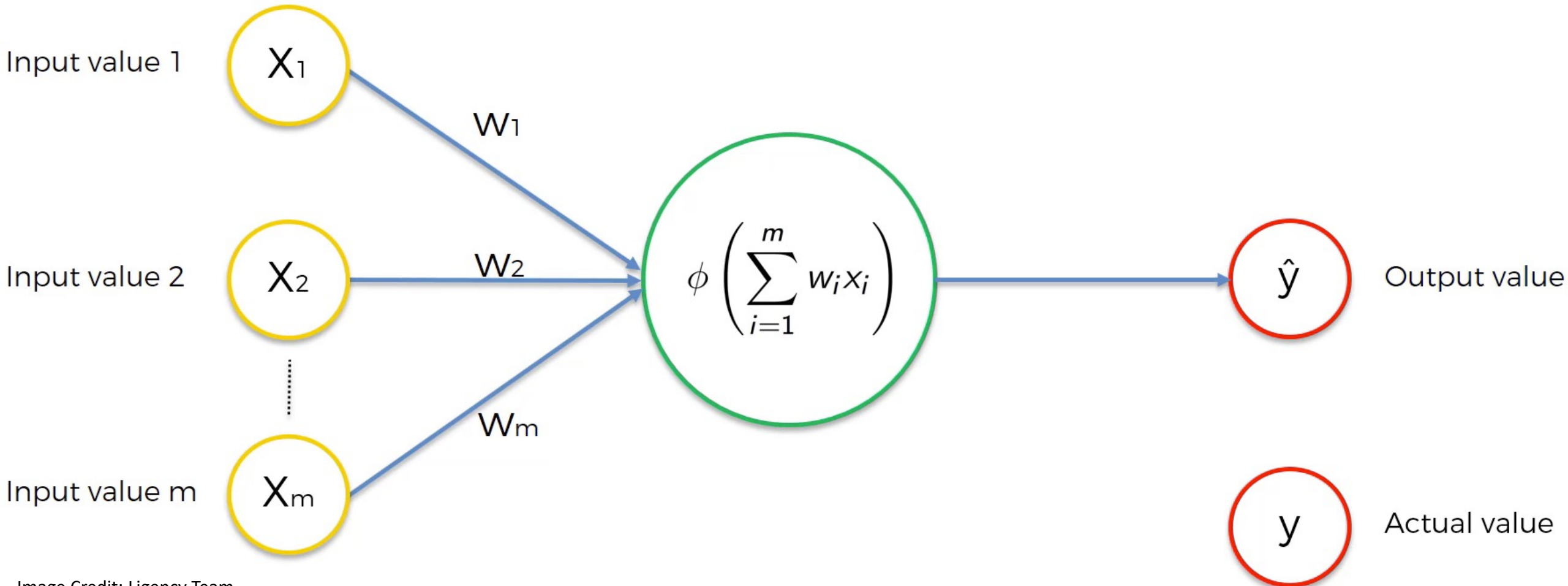


Image Credit: Ligency Team

How do NNs learn?

The cost function is calculated as the difference between the output value and the actual value.

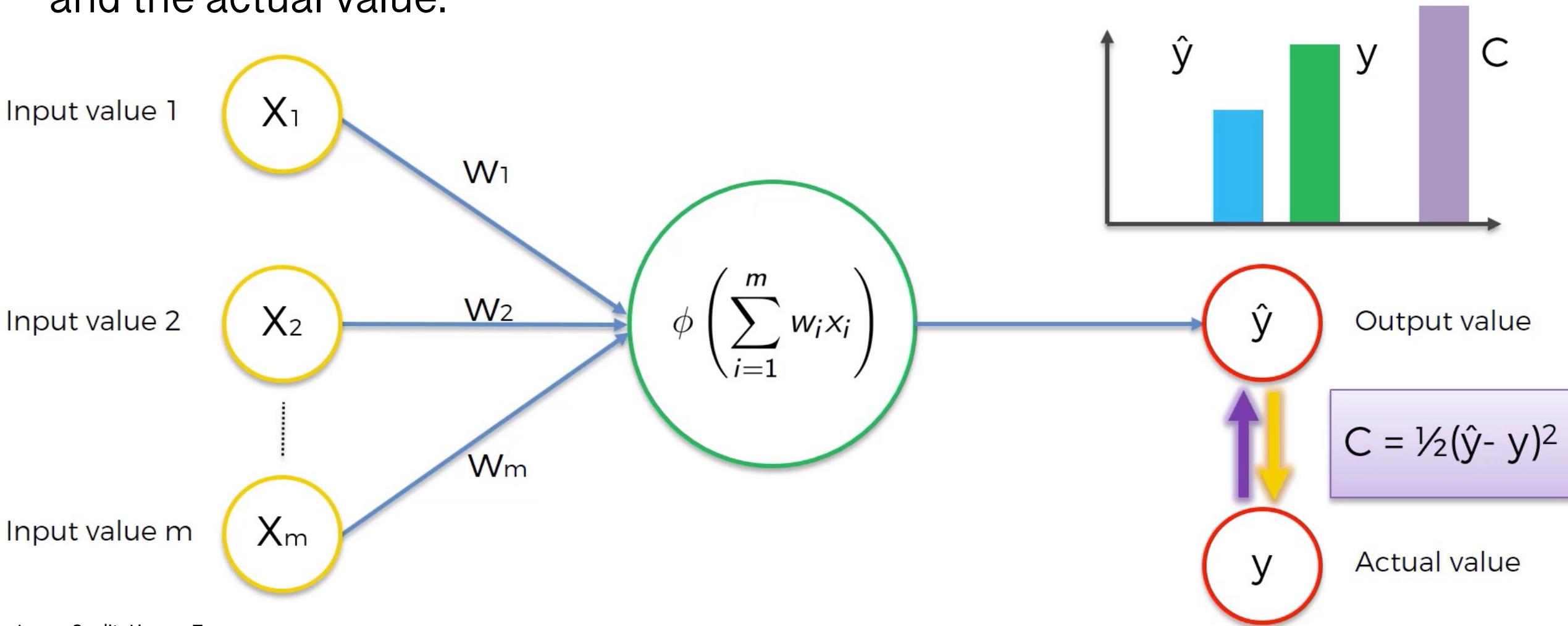


Image Credit: Ligency Team

How do NNs learn?

The weights for each input are then re-adjusted based on error.

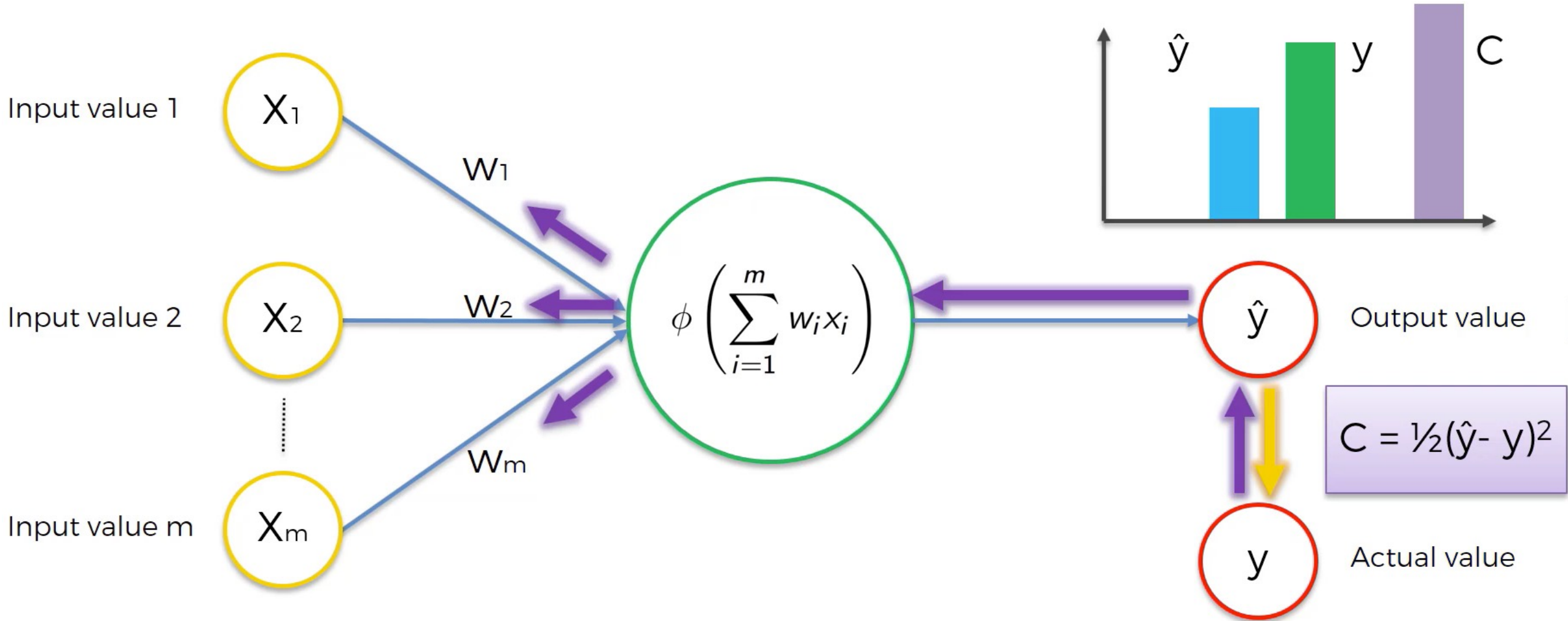


Image Credit: Ligency Team

How do NNs learn?

This process is done for each row, but not separately (batch).

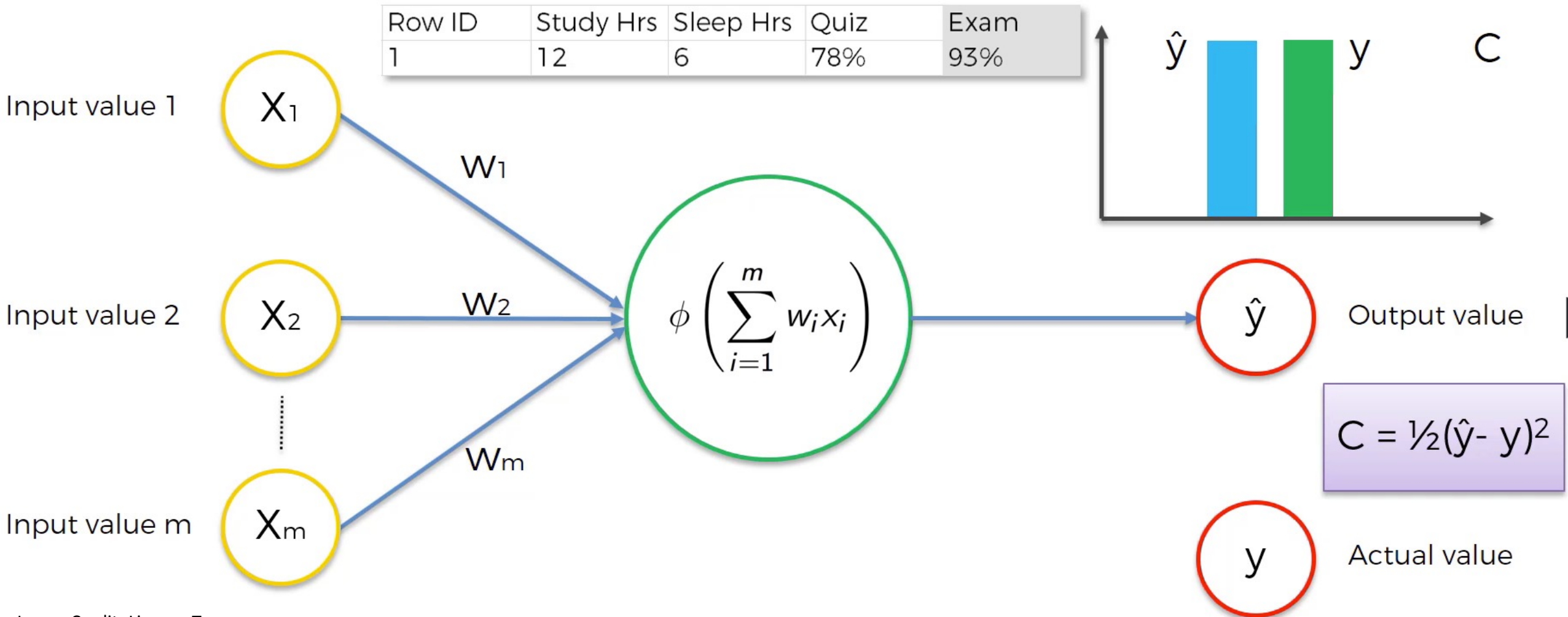
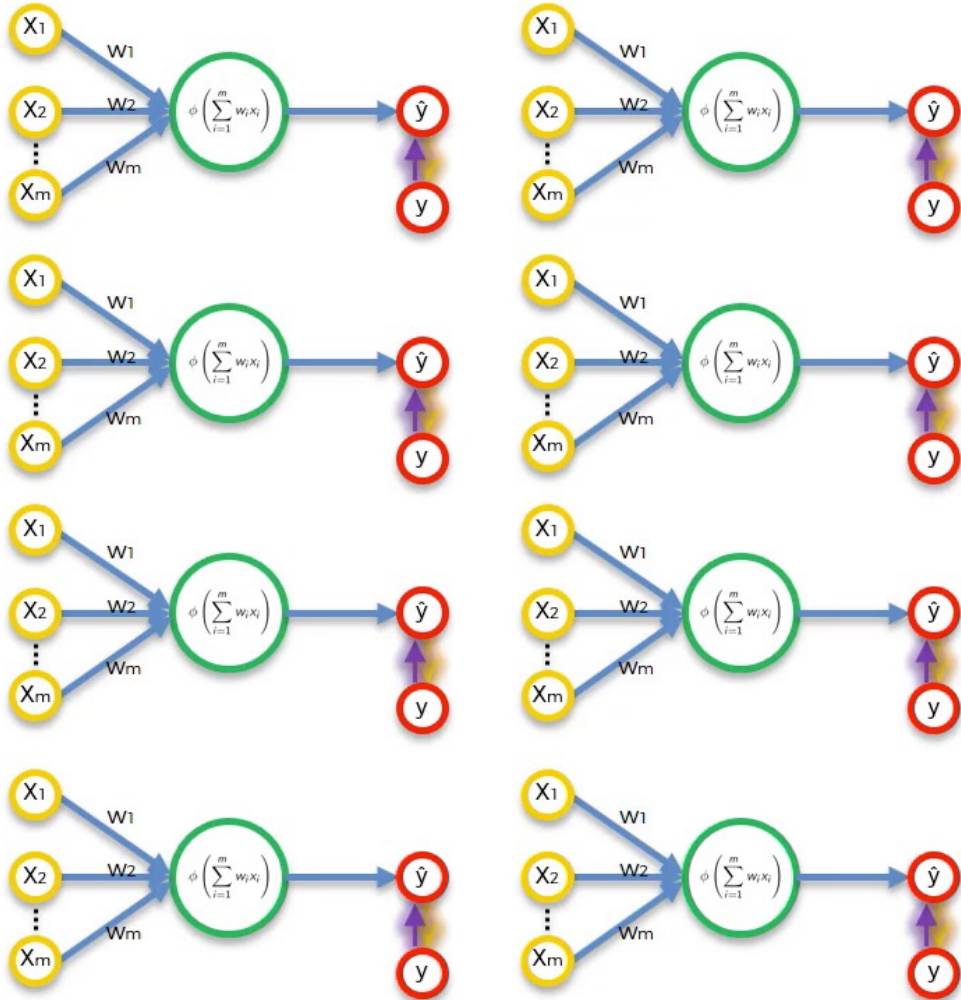


Image Credit: Ligency Team

How do NNs learn?

The sum from all rows in the training set is used for backpropagation.



Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

$$C = \sum \frac{1}{2}(\hat{y} - y)^2$$



Image Credit: Ligency Team

Sample, Batch, and Epoch

- Sample is one row of entry data, all features at single point.
- Batch is a number of samples that the algorithm will calculate the cost function on.
 - Batch size = n_rows , all of the training set is used for training. This is called “Batch Gradient Descent”.
 - Batch size = 1, all rows are used separately for training. This is called “Stochastic Gradient Descent”.
 - $1 < \text{Batch size} < n_rows$, the training set is divided into batch sizes for training. This is called “Mini Batch Gradient Descent”.
- Epoch is the number of times the algorithm will work through the training set.

For a training set of 100 samples, a batch size of 5 means the training will be made on 20 epochs with 20 batches.

Steps involved in NN learning

Step 1: Randomly initialize the weights to small numbers close to 0.

Step 2: Input the first observations from your dataset in the input layer, each in one input node.

Step 3: Forward-Propagation: From left to right, the neurons are activated in a way that is limited by the weights. Propagate the activations until the predicted result y .

Step 4: Compare predicted result to the actual result. Measure the error.

Step 5: Back-Propagation: From right to left, the error is back-propagated. Using the error, weights are updated. The learning rate decides how much the weights are updated.

Step 6: Repeat Steps 1 to 5 and update weights (Reinforcement Learning) or Repeat Steps 1 to 5 but update only after a batch (Batch Learning).

Step 7: When the whole training set is passed through it is one epoch. Redo for more epochs.

Deep Learning (Multi-layer perceptrons)

When an ANN contains a deep stack of hidden layers, it is called a deep neural network (DNN).

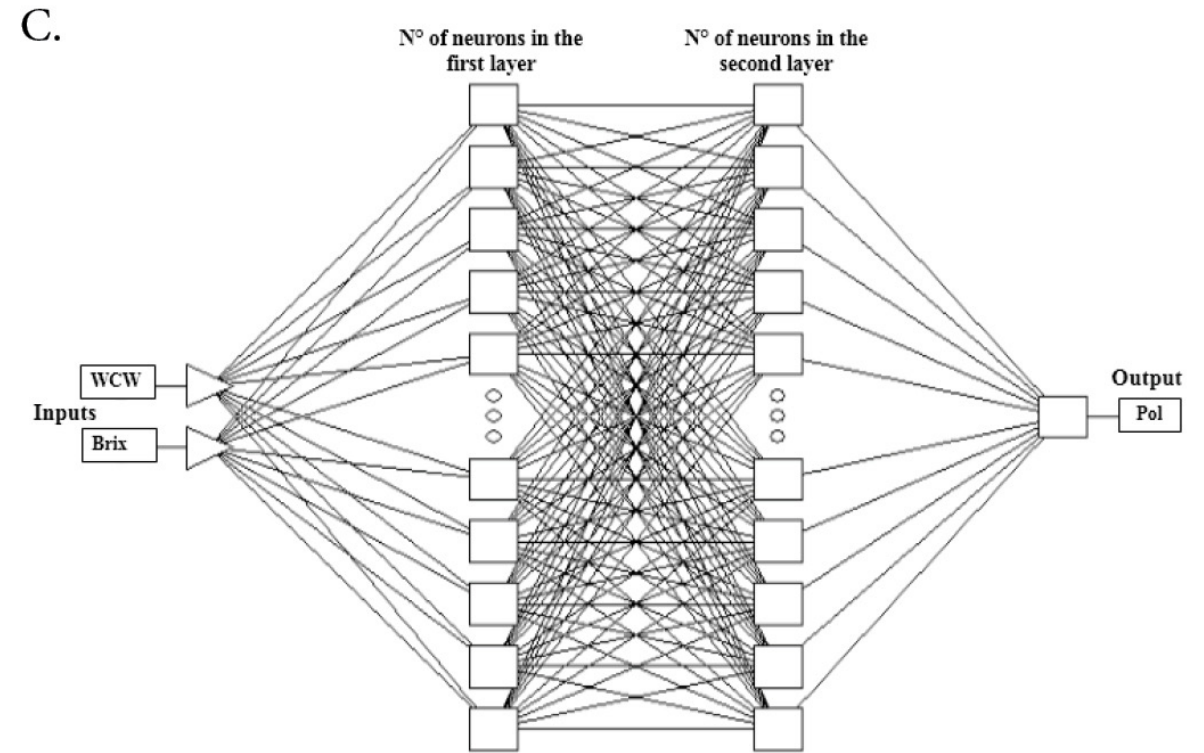
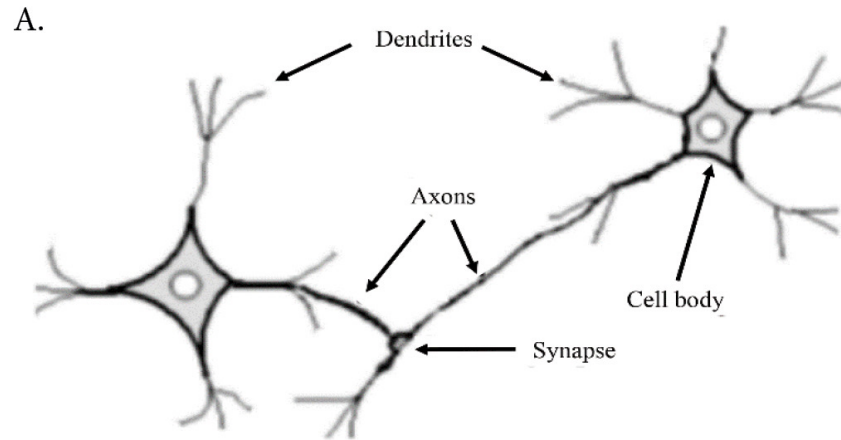


Image Credit: A. P. Coelho et al., 2018, Scielo

MLP for Regression

MLPs can be used for regression tasks:

- To predict single value, 1 output neuron.
- For multivariate regression, one output neuron per output dimension.
- No output activation is necessary, but still you can use them.

Hyperparameter	Default value
# input neurons	One per input feature
# hidden layers	Depends on the problem, 1 to 5
# neurons per hidden layers	Depends on the problem, 10 to 100
# output neurons	1 per prediction dimension
Hidden activation	ReLU
Output activation	ReLU/softplus (positive output) logistic/tanh (bounded output)
Loss function	MSE, MAE, or RMSE

MLP for Regression

from sklearn.neural_network import MLPRegressor

```
class sklearn.neural_network.MLPRegressor(hidden_layer_sizes=100, activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000)
```

- hidden_layer_sizes: tuple, length = n_layers - 2, default=(100,)
- activation: {'identity', 'logistic', 'tanh', 'relu'}, default='relu'
- solver: {'lbfgs', 'sgd', 'adam'}, default='adam'
- best_loss_: float The minimum loss reached by the solver throughout fitting.
- loss_curve_: list of shape (n_iter_,): Loss value evaluated at the end of each training step. The ith element in the list represents the loss at the ith iteration.
- coefs_: list of shape (n_layers - 1,): The ith element in the list represents the weight matrix corresponding to layer i.

You can learn more on MLPRegressor at: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPRegressor

MLP for Classification



Tip: Cross entropy is also known as log-loss function.

MLPs can be used for classification tasks:

- To predict single value or its probability, 1 output neuron.
- If sample has more than one label, one output per label.
- If each sample can belong to a distinct class, one output per class.

Hyperparameter	Binary Classification	Multilabel Binary Classification	Multiclass Classification
# input neurons	One per input feature	One per input feature	One per input feature
# hidden layers	Depends on the problem	Depends on the problem	Depends on the problem
# output neurons	1 per prediction dimension	1 per label	1 per class
Output activation	Logistic/tanh	Logistic/tanh	Softmax
Loss function	Cross entropy	Cross entropy	Cross entropy

MLP for Classification

from sklearn.neural_network **import** MLPClassifier

```
class sklearn.neural_network.MLPClassifier(hidden_layer_sizes=100, activation='relu', *,
solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001,
power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False,
warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False,
validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10,
max_fun=15000)
```


- hidden_layer_sizes: tuple, length = n_layers - 2, default=(100,)
- activation: {'identity', 'logistic', 'tanh', 'relu'}, default='relu'
- solver: {'lbfgs', 'sgd', 'adam'}, default='adam'

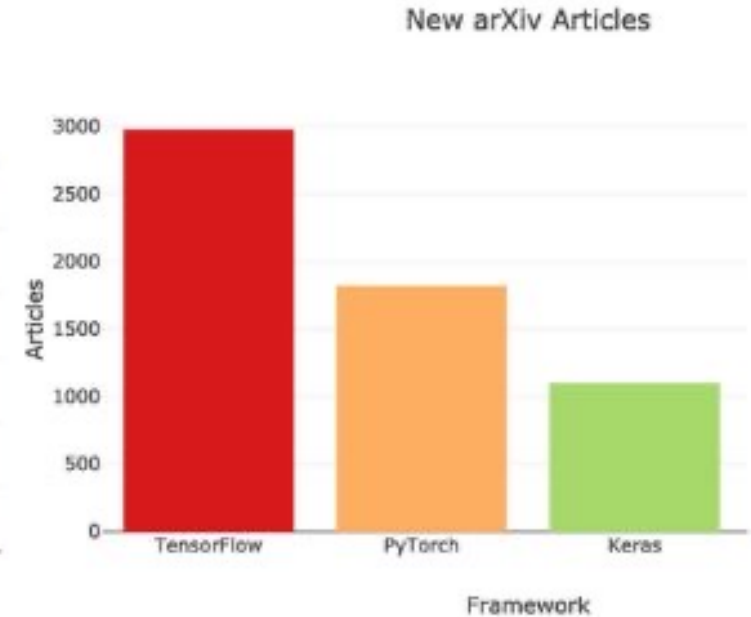
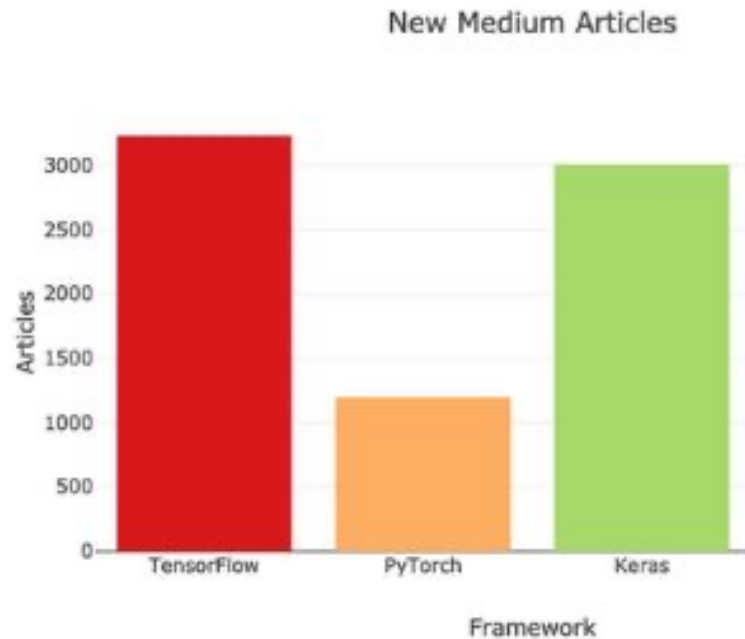
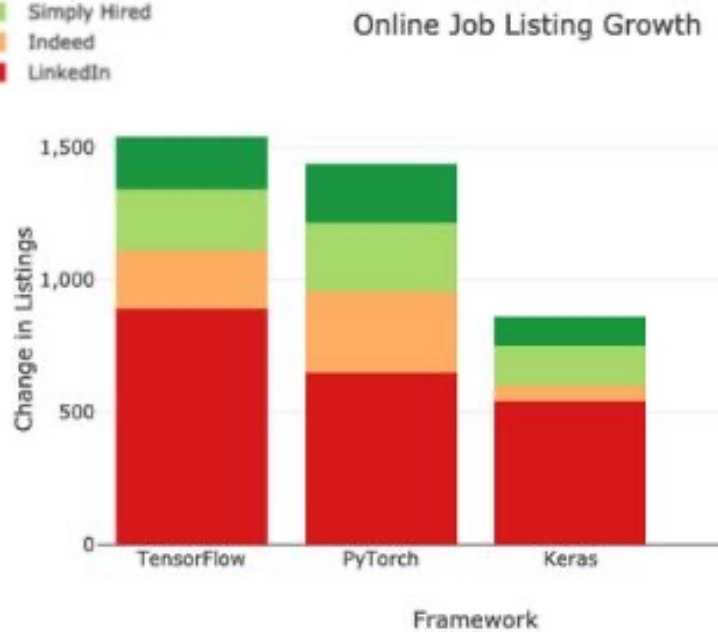
You can learn more on MLPClassifier at: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier

Strengths and Weaknesses

- Neural networks often beat other ML algorithms for regression and classification tasks when large amounts of data is present.
- They take a long time to train.
- They work best with homogeneous, well cleaned data.
- Tuning the neural network parameters is very arduous.
- Overfitting is very likely.
- There is a bias for/against neural networks.

Which library is right for you?

Library	Friendly	API Level	Speed	Size
TensorFlow				
Keras				
PyTorch				



Source: [KDNuggets](https://www.kdnuggets.com/)