Deep Learning for Renewable Energy Forecasting

Yan Zhang

Professor, University of Oslo, Norway

Learning Objectives

Throughout this lecture, it is aimed for the students to be able to

- **Learn neural network technique in plain language, programming language, and mathematical language**
- **Apply neural network for wind energy forecasting**

Industry Invited Talk Today

- **Speakers**: Boris Tistan, *Group Manager Powel.AI, Powel AS*

- **Title**: Machine Learning Applications for Intelligent Energy Systems

- **Powel**: Founded in Norway in 1996, Powel has grown to be an international corporation and a leading supplier of software solutions to the energy, public and contracting sectors

powel

Outline

- **Neural Network: concepts and principles**
- **Neural Network: applications in Boston House Price Forecasting**
- **Neural Network: applications in Wind Energy Forecasting**

Deep learning

- **Deep Learning: not an n-layered neural network where n is a very big number. Even a simple two layered neural networks which consists of input and output layers only can be classified as a Deep Learning model. Deep learning breakthrough came in 2006, when methods were developed to overcome the difficulties in training deep neural networks.**

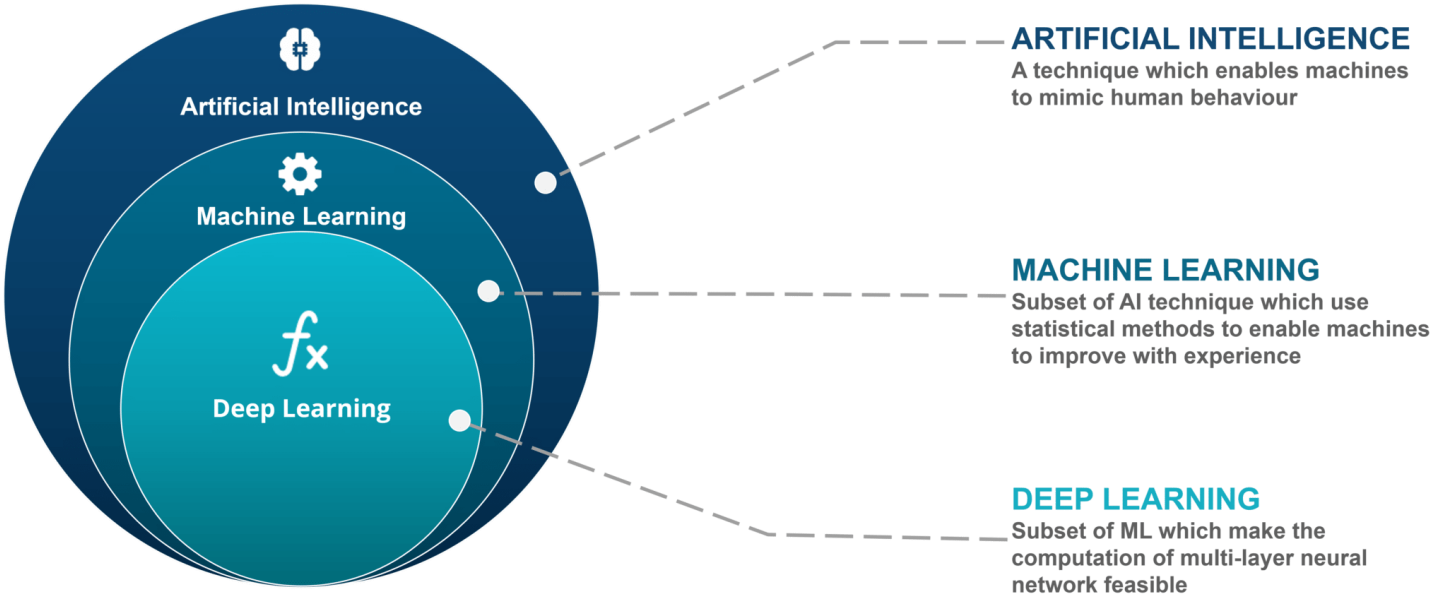
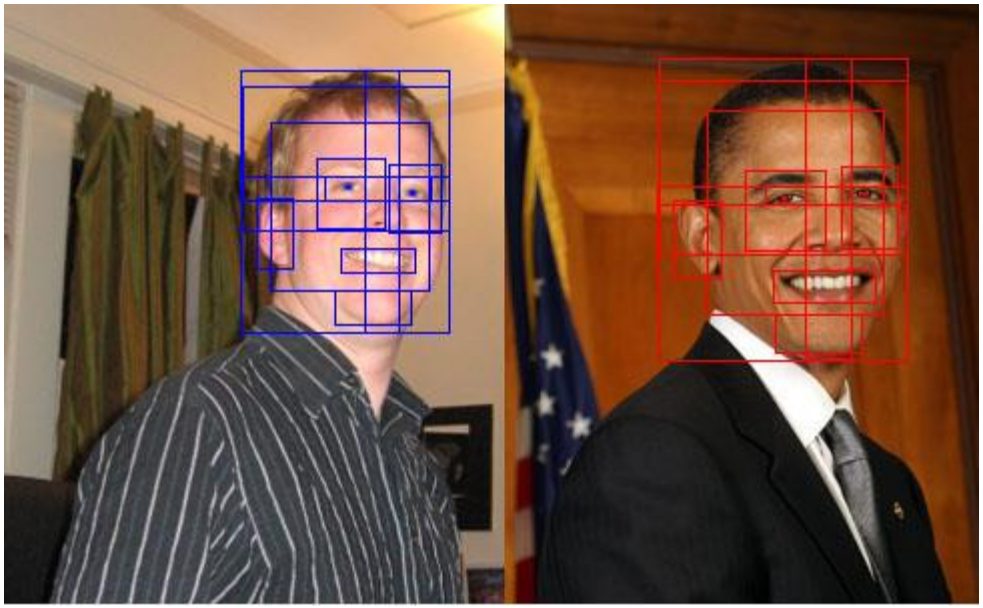


Figure source: https://www.edureka.co/blog/ai-vs-machine-learning-vs-deep-learning/

NEURAL NETWORK AND ITS APPLICATION

Artificial Neural Network (ANN)

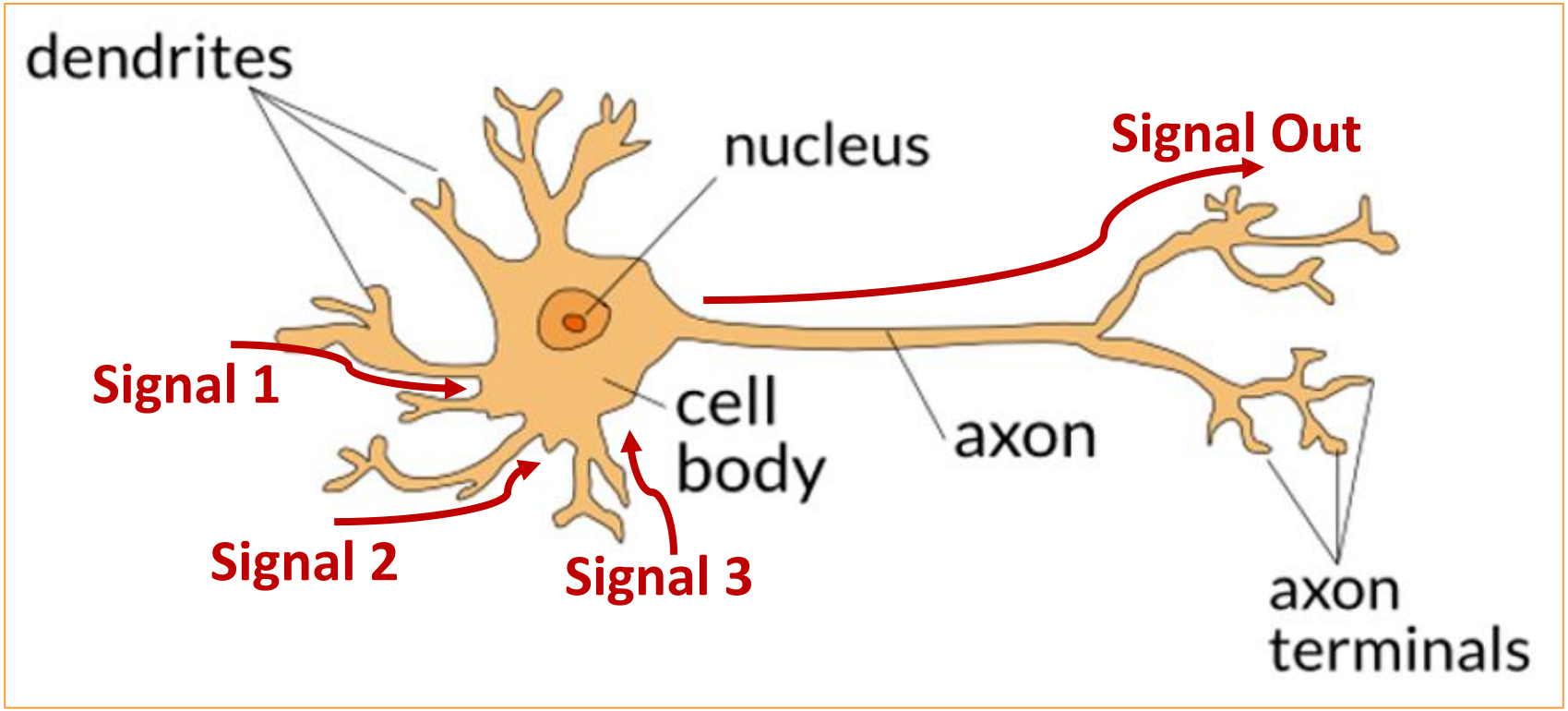


Q: who is Obama?

- **Computers are great at solving algorithmic and math problems, but many problems can't easily be defined with a mathematical algorithm. Facial recognition is a typical such example. Computers believe that the two photos show the same person. However, this is apparently incorrect and such tasks are trivial to humans.**

- **Artificial Neural Networks main idea: enables computers to process information in a similar way as our own biological brains and our own nervous system.**

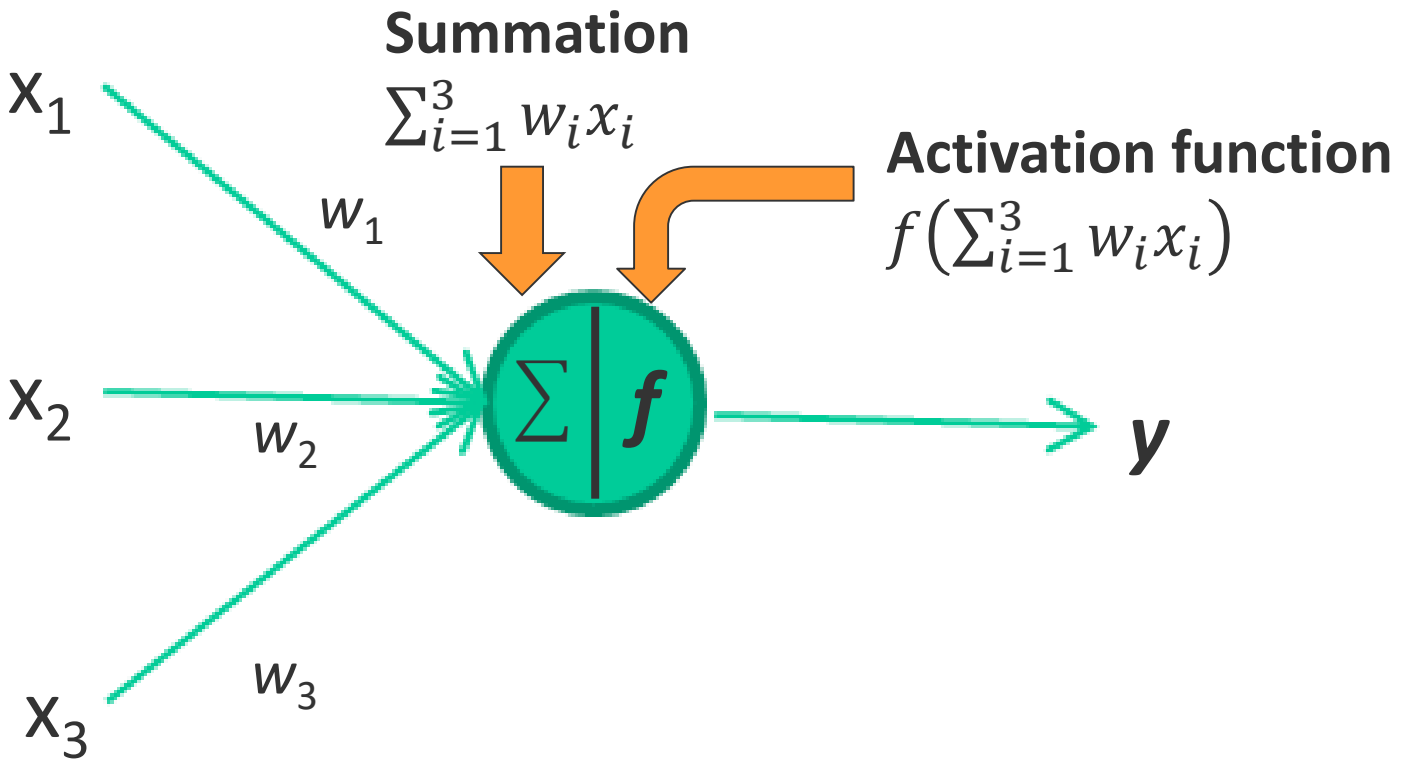
Biological Model for a Single Neuron



- Incoming signals are received by the cell's **dendrites** through a biochemical process. The process allows the impulse to be weighted according to its relative importance.

- As **the cell body** begins accumulating the incoming signals, a threshold is reached at which the cell fires and the output signal is transmitted via an electrochemical process down the **axon**.

From Biological Model to a Single Artificial Neuron (I)



- This figure defines the relationship between the input signals (**x variables**) received by the dendrites and the output signal (**y variable**).
- Each dendrite's signal is weighted (**w values**) according to importance.
- The input signals are summed by the cell body and the signal is passed on according to an **activation function denoted by f**.

From Biological Model to a Single Artificial Neuron (II)

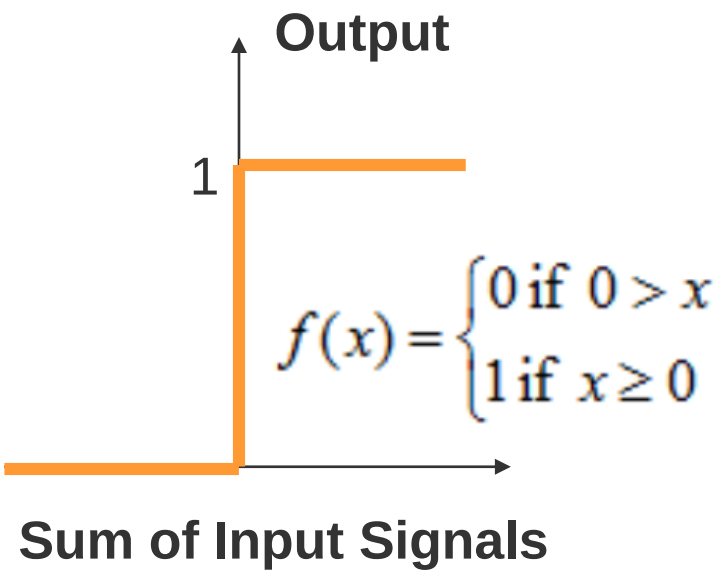
- **Model for a single artificial neuron**: A typical artificial neuron has n input dendrites. The w weights allow each of the n inputs (denoted by x_i) to contribute a greater or less amount to the sum of input signals. The total is used by the activation function $f(x)$, and the resulting signal y, is the output axon:

$$y(x) = f\left(\sum_{i=1}^{n} w_i x_i\right)$$

- **Activation function f(.)**: can be threshold activation function, sigmoid function or other different functions.

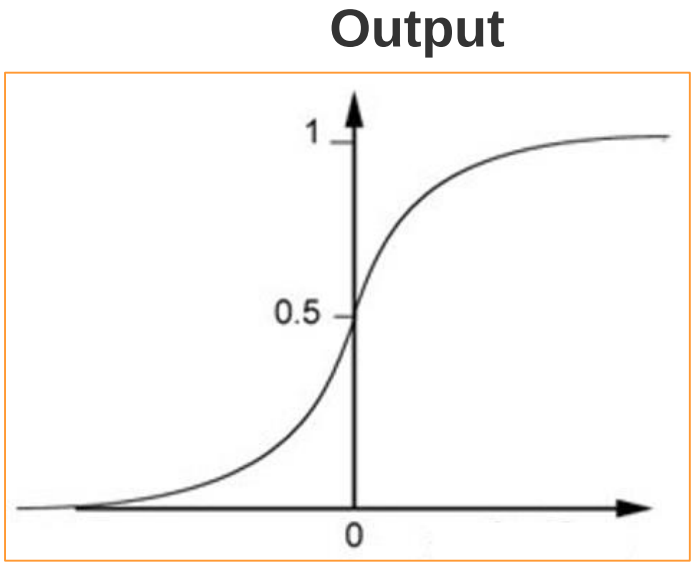
Activation Function (I) – threshold activation function

- In the biological case, the activation function is the process that involves summing the total input signal and determining whether it meets the firing threshold. If so, the neuron passes on the signal; otherwise, it does nothing.
- Threshold Activation Function: it results in an output signal only once a specified input threshold has been reached.
- The neuron fires when the sum of the input signals is at least zero.
- The threshold activation function parallels with biology, but it is rarely used in artificial neural networks. **(Q: why?)**



Activation Function (II) – sigmoid activation function

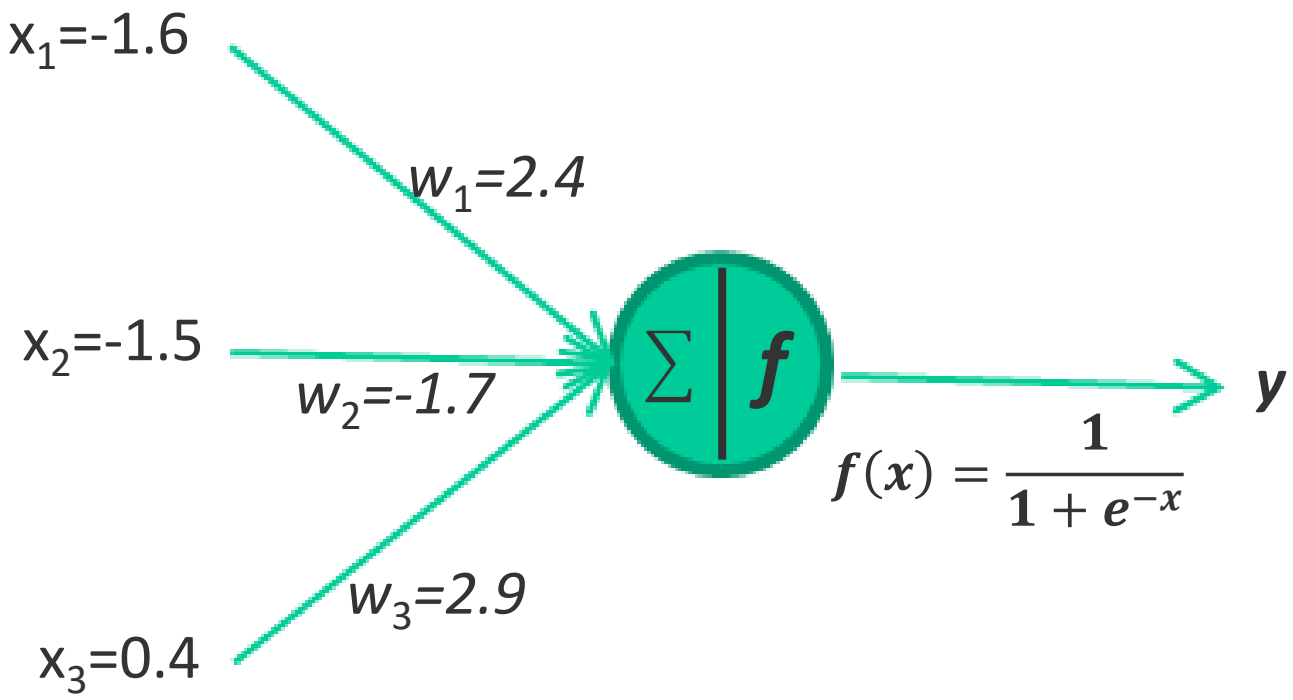
- Sigmoid activation function: the most commonly used activation function
- The output signal is no longer binary; the output values can be any value in the range from 0 to 1.
- The sigmoid is differentiable, and it is possible to calculate the derivative across the entire range of inputs. In addition, the derivative function can be easily computed and can significantly reduce the computation cost during training.



$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = f(x)[1 - f(x)]$$

Sigmoid Function: an example



1. **Calculate the weighted sum of input signals:**

$$x = \sum_{i=1}^{3} w_i x_i = (-1.6)\times 2.4 + (-1.5)\times(-1.7) + 0.4\times 2.9 = -0.13$$

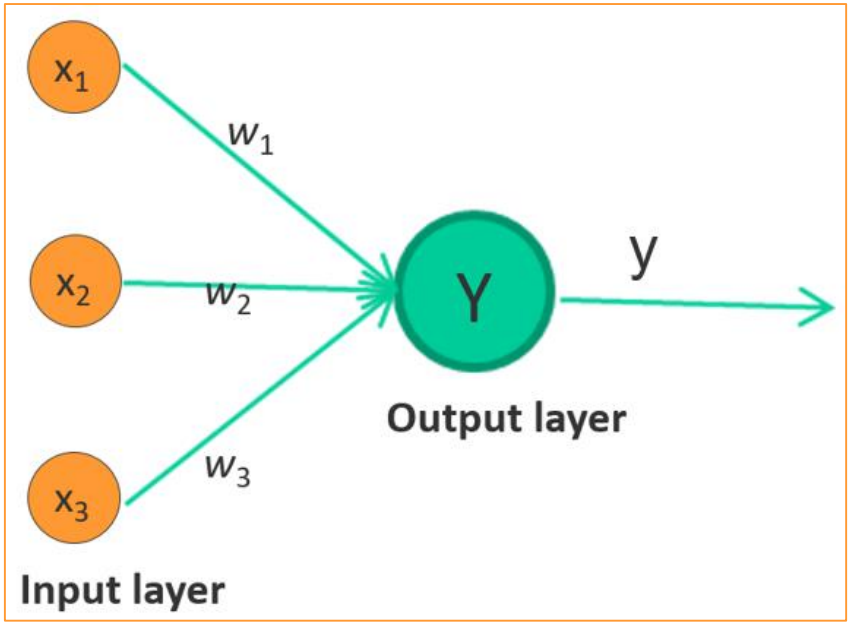
2. **Calculate the output signal based on sigmoid function:**

$$y = f(-0.13) = \frac{1}{1+e^{0.13}} = 0.468$$

From a Single Artificial Neuron to Neural Networks

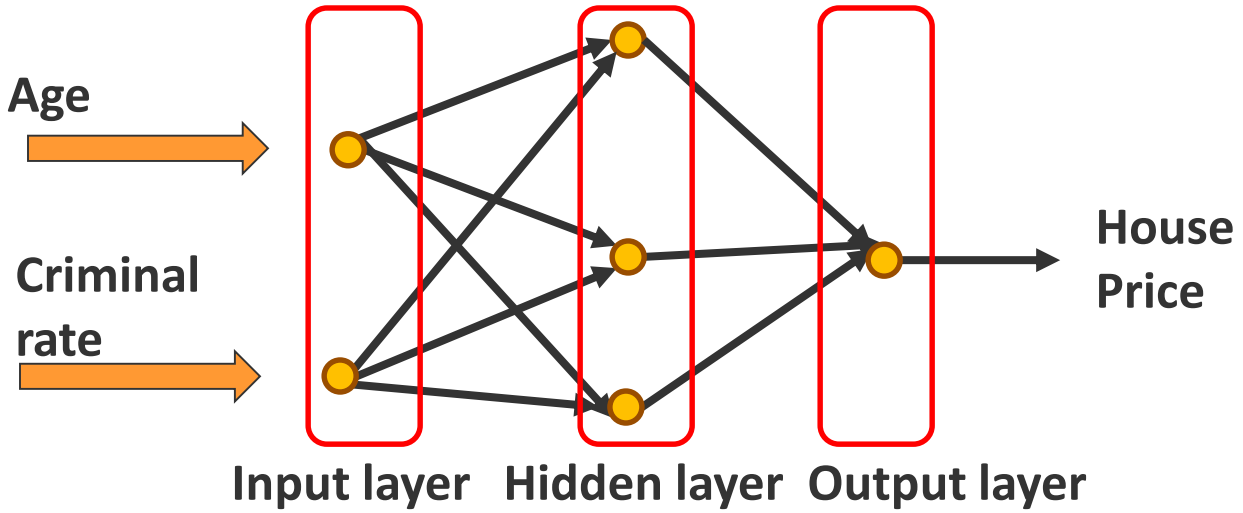
- Neural Networks: we use many neurons as building blocks to construct complex models of data.
- A neural network has three characteristics.
- Activation function $f(x)$: transforms a neuron's combined input signals into a single output signal to be broadcasted further in the networks
- Network topology: describe the number of neurons in the model, the number of layers, and the manner in which the layers are connected
- Training algorithm: specifies how weights w are set in order to excite neurons in proportion to the input signal.

Network Topology: a single layer neural network



- **Input layer**: A set of neurons receives the input data. Each input node is responsible for processing a single feature in the dataset. The feature's value will be transformed by the corresponding node's activation function.
- **Output layer**: The signals sent by the input nodes are received by the output layer, which uses its activation function to generate a final prediction **y**.
- This is a single-layer neural network: the input nodes process the incoming data, the network has only one set of weights (**w1,w2,w3**)

Network topology: a typical neural network with one hidden layer

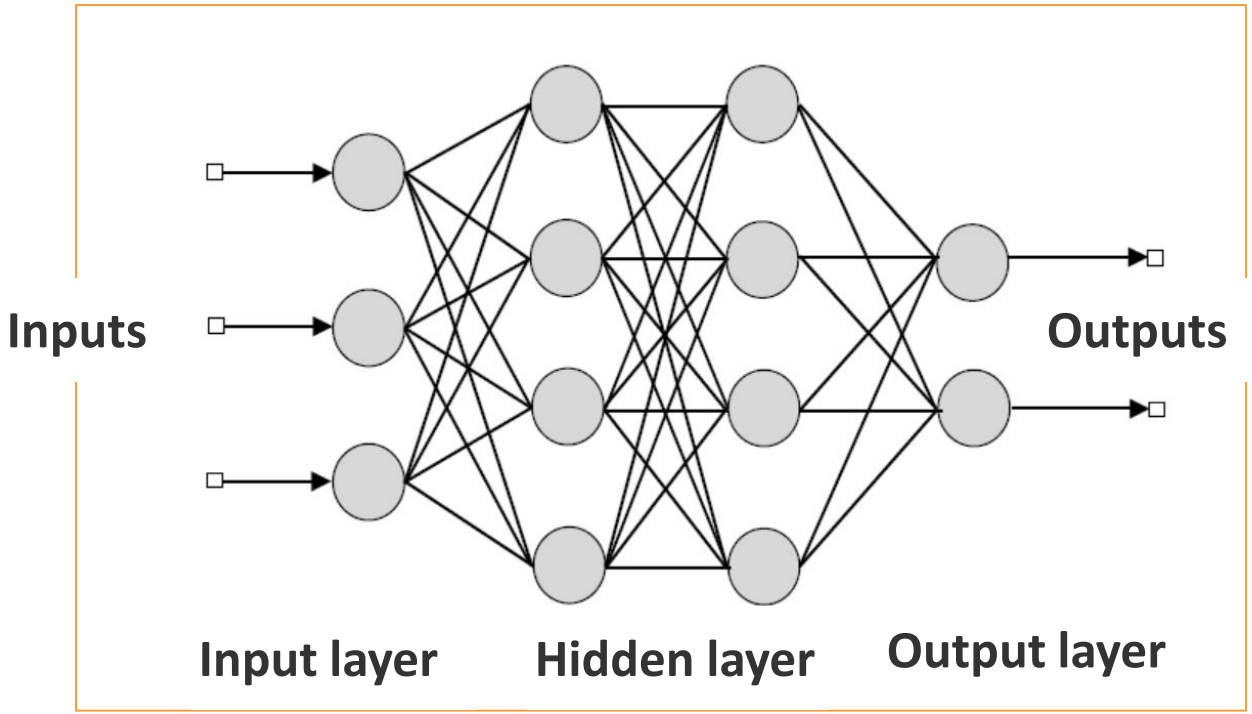


- A typical neural network topology used to predict the house price. It has 2 input nodes, 1 hidden layer with 3 nodes, and 1 output node. The information flows unidirectional to the hidden layer; and then to the output layer.

- The input nodes feed the attributes (Age, Criminal rate) into the network. There is one input node for each attribute. The output calculates a weighted sum on the received data to predict house price.

- **Q:** why do we need hidden layers?

De facto standard topology



- An artificial neural network has multiple layers and it is an interconnected group of nodes. One or more hidden layers are added that process the signals from the input nodes prior to it reaching the output node.
- This topology is also called **Feedforward Neural Network** since the information flows unidirectional from the input layer to the hidden layer; and then to the output layer.
- This is the de facto standard neural network topology.

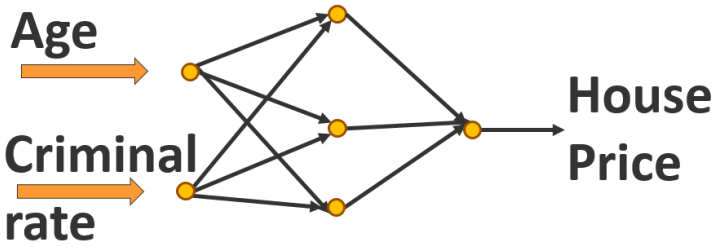
What can Neural Network work for us? - an example of house price

- We donot know the math relationship between “House Price” and “Age and “Criminal rate”. It is difficult to calculate the house price.
- Now, we have data for 1000 houses with data record (Age, Criminal Rate, House Price). Then, we need to predict a new house’s price when we know its age and its criminal rate.

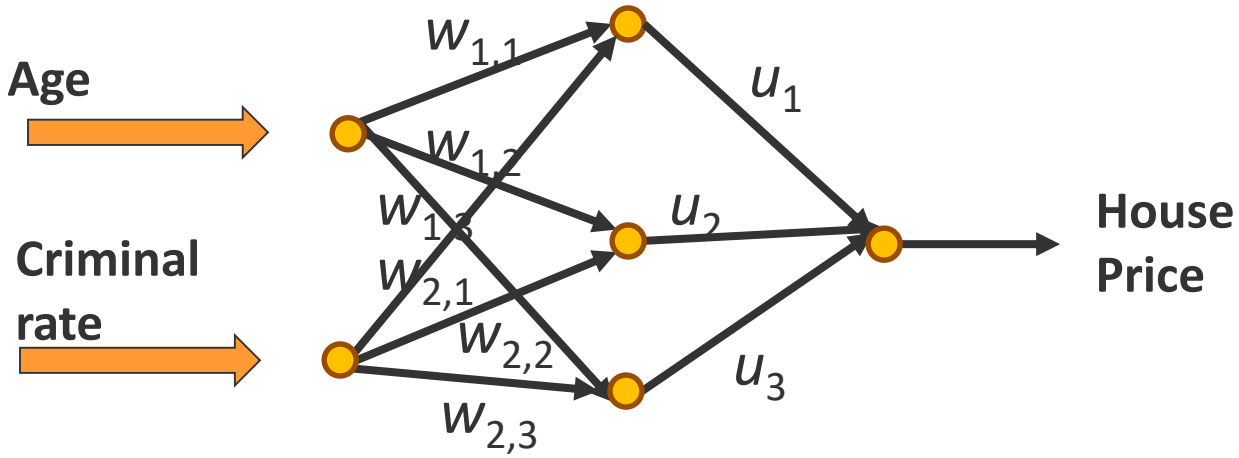
House 1, 2,3,…, 999, 1000

1001# house price?

Use 1000 houses data to build neural network model



What can Neural Network work for us? - an example of house price



- The 1000 houses training data will decide the weight ($w_{1,1}$, $w_{1,2}$ … $w_{2,3}$, u_1, u_2, u_3) between/in the hidden layers. Then, we have the neural network model to show the relationship between the output "House Price" and the inputs "Age" and "Criminal rate".

- After the weights are known, it is easy to get the price for a new house after feeding the age and criminal rate data of the new house.

- **Q:** for a new house, we only know its Age. Can we predict its price?

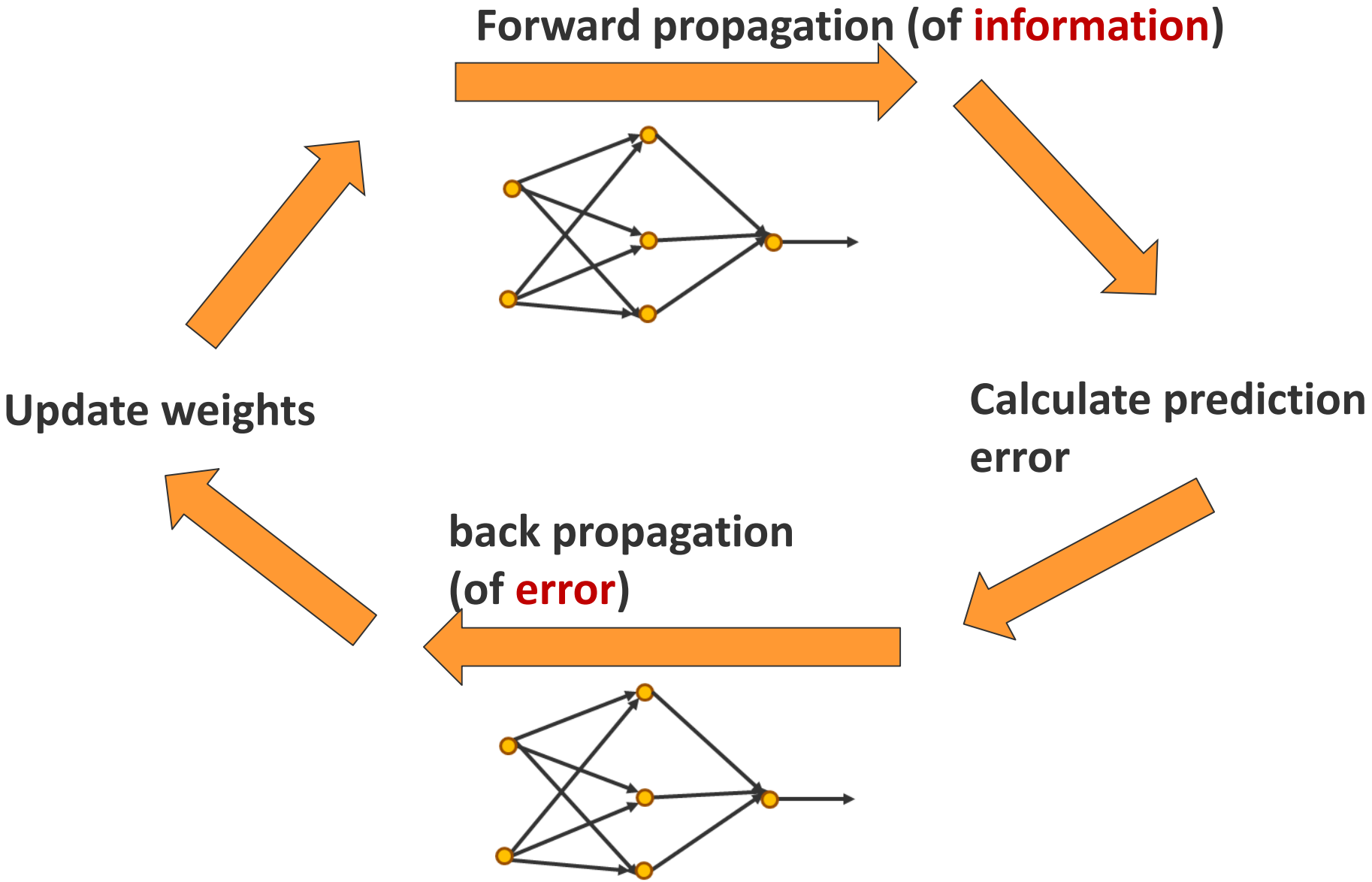
We need to decide the weight: Backpropagation Algorithm

- **Need to decide the weights**: There are many learning algorithms. They will train the network by iteratively modifying the weights until the error between the output produced by the network and the desired output falls below a specified threshold.

- **Backpropagation (BP) algorithm**: the first popular learning algorithm and is still widely used. It uses **gradient descent** as the learning mechanism. Starting from random weights, the backpropagation algorithm calculates the weights and gradually makes adjustments determined by the error between the result produced by the neural network and the real outcome.

- The algorithm applies error propagation from outputs to inputs, and gradually fine tunes the network weights to minimize the sum of error.

Neural network learning cycle



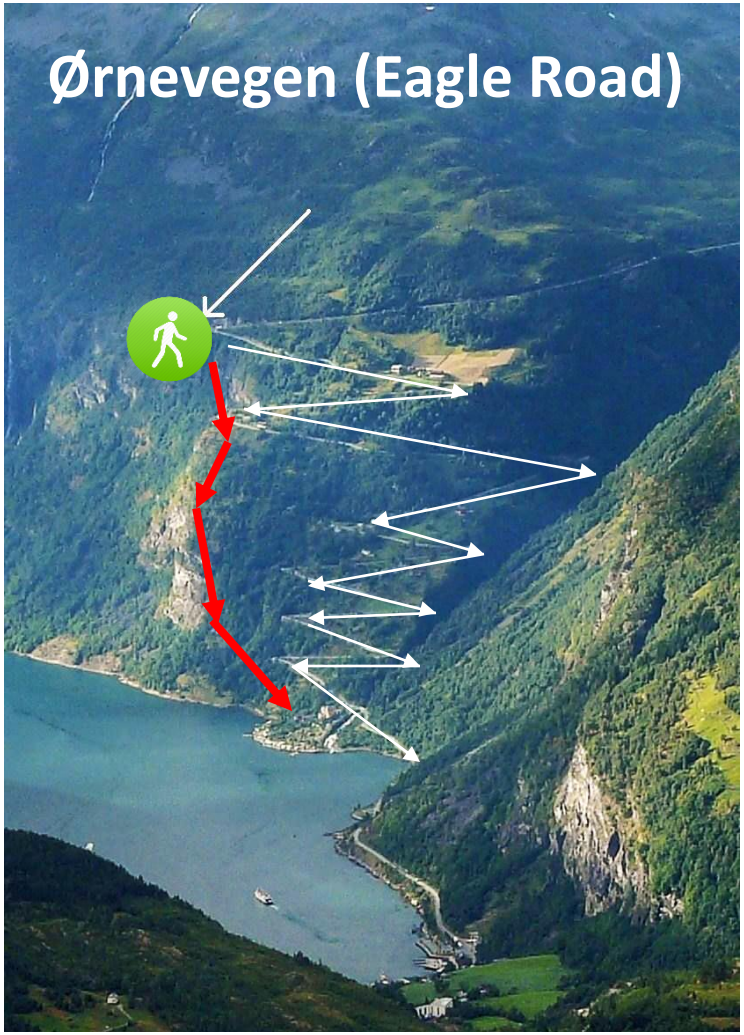
Training Neural Networks with Backpropagation

- **Forward phase**: neurons are activated in sequence from the input layer to the output layer, applying each neuron's weights and activation function along the way. Upon reaching the final layer, an output signal is produced.
  - Forward phase focuses on information forward propagation
- **Backward phase**: network's output signal resulting from the forward phase is compared to the true value in the training data. The difference between the output signal and the true value results in an error. This error is propagated backwards in the network to adjust the weight and reduce future errors.
  - Backward phase focuses on error backward propagation

Gradient Descent Technique for Adjusting Weight (I): general idea

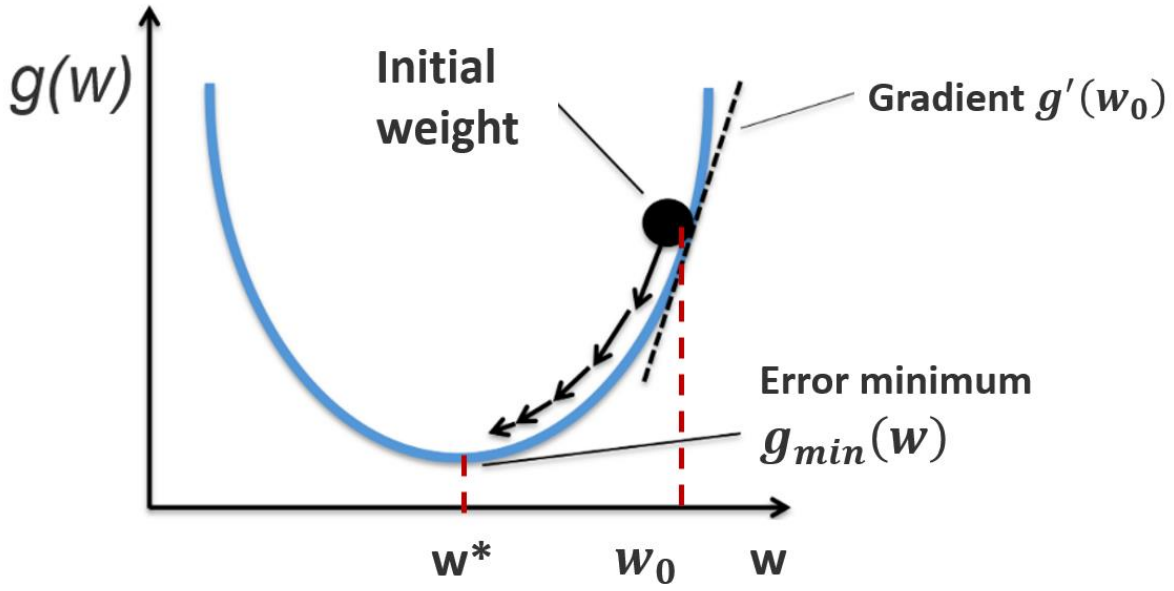
- **Gradient Descent**: the algorithm updates the weights in the direction of the gradient of the error function until a minimum is reached. We need to adjust weights in the direction of reducing error.

- This is analogous to going down a hill. When we are standing on top of a hill. We look around and decide the direction to go, then we can reach the bottom of the hill fastest. A simple strategy is to take a step in the direction where the direction is descending the fastest. Clearly the route with red color instead the route with white color.



The steepest stretch of road from Geiranger towards Eidsdal on road no. 63.

Gradient Descent Technique for Adjusting Weight (II): how to adjust?



- $\boldsymbol{w}$: weight; $\boldsymbol{g(w)}$: error function (summation of squared error)
- $\boldsymbol{g'(w)}$: first-order derivative function: importance of having a differentiable activation function
- $g'(w_0) > 0$: the weight is decreased in the direction from w_0 to w*; $g'(w_0) < 0$: the weight is increased
- $g'(w_0) = 0$: we arrive at the optimal weight w^*

Neural networks package neuralnet in R

- **You need to install it by typing** install.packages("neuralnet") **and load it with the** library(neuralnet) **command.**
- **Build the model:**
  - m <- neuralnet(target~predictors, data = MyData, hidden = 1)
  - target: outcome in the MyData data frame to be modeled
  - predictors: features in the MyData data frame to use for prediction
  - data: data set where the target and predictors variables can be found
  - hidden: the number of neurons in the hidden layer
- **Making predictions:**
  - p <- compute(m, test)
  - m: trained model by the neuralnet() function
  - test: test data with the same features as the training data
- **More info:** https://cran.r-project.org/web/packages/neuralnet/index.html

Boston House Price (we do not have Oslo house price)

1	CRIM,ZN,INDUS,CHAS,NOX,RM,AGE,DIS,RAD,TAX,PTRATIO,LSTAT,MEDV
2	0.00632,18,2.31,0,0.538,6.575,65.2,4.09,1,296,15.3,4.98,24
3	0.02731,0,7.07,0,0.469,6.421,78.9,4.9671,2,242,17.8,9.14,21.6
4	0.02729,0,7.07,0,0.469,7.185,61.1,4.9671,2,242,17.8,4.03,34.7

....

505	0.06076,0,11.93,0,0.573,6.976,91,2.1675,1,273,21,5.64,23.9
506	0.10959,0,11.93,0,0.573,6.794,89.3,2.3889,1,273,21,6.48,22
507	0.04741,0,11.93,0,0.573,6.03,80.8,2.505,1,273,21,7.88,11.9

- **The Boston data frame has 506 rows and 13 columns. For each column data,**
- **CRIM: per capita crime rate by town; ZN: proportion of residential land zoned for lots over 25,000 sq.ft; INDUS: proportion of non-retail business acres per town; CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise); NOX: Nitrogen oxides concentration (parts per 10 million); RM: average number of rooms per dwelling; AGE: proportion of owner-occupied units built prior to 1940; DIS: weighted mean of distances to five Boston employment centres; RAD: index of accessibility to radial highways; TAX: full-value property-tax rate per \$10,000; PTRATIO: pupil-teacher ratio by town; LSTAT: lower status of the population (percent);**
- **MEDV: median value of owner-occupied homes in $1000s.**

Training Data, Test Data, and Experiment

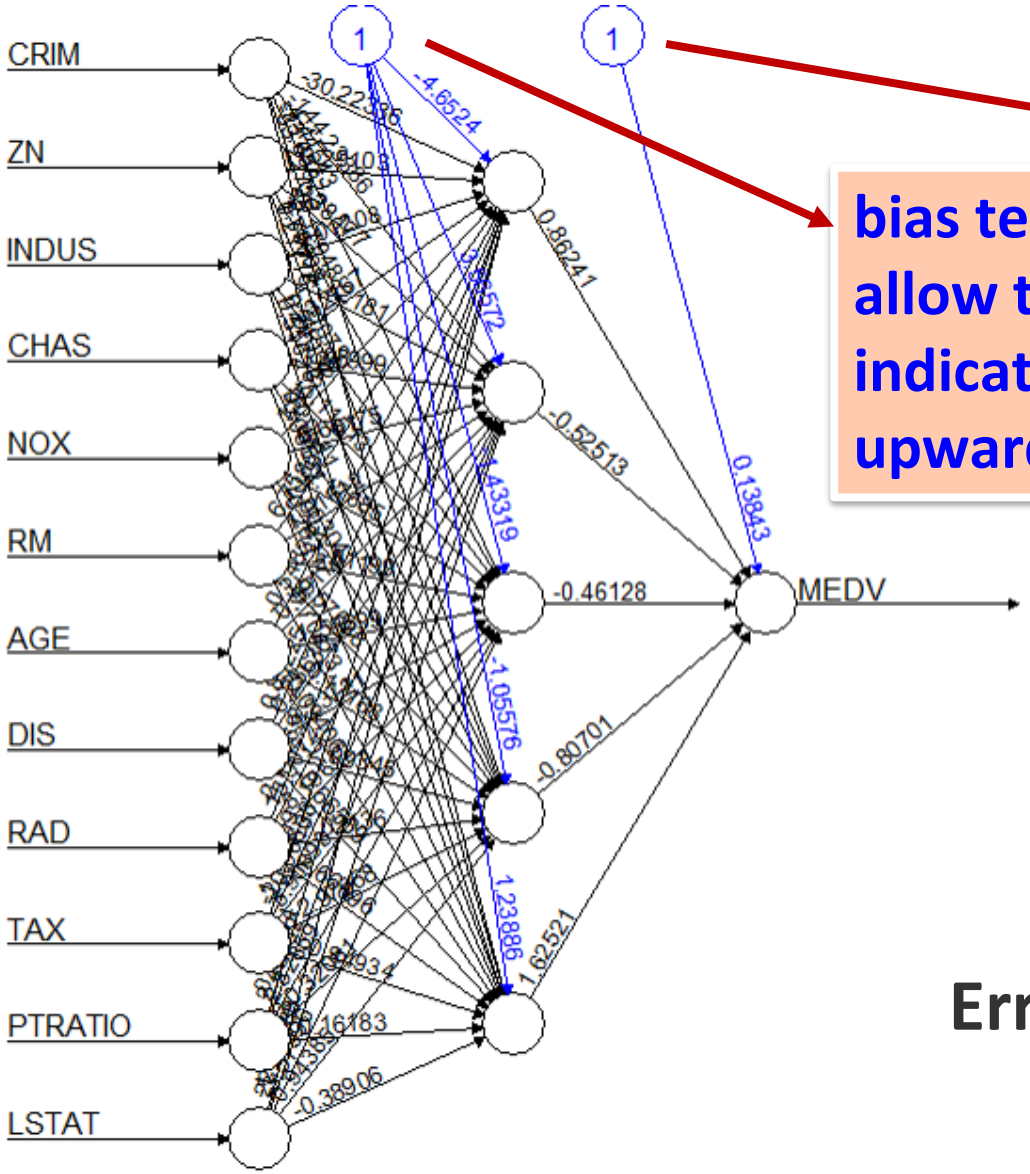
- For 506 data records, we divide them into two parts. We use 75% first data record (from 1 to 380 data records) are used as training data to build neural network model. The last 25% data records (from 381 to 506 data records) are used as test data.
- We try neural network with one hidden layer that has 5 nodes

```
bostonModel <- neuralnet(MEDV ~ CRIM + ZN + INDUS + CHAS + NOX + RM
+ AGE + DIS + RAD + TAX + PTRATIO + LSTAT, data = trainingData,
hidden = 5)
```

- We try neural network with three hidden layers with
  - 5 nodes in the first hidden layer
  - 4 nodes in the second hidden layer
  - 3 nodes in the third hidden layer

```
bostonModel <- neuralnet(MEDV ~ CRIM + ZN + INDUS + CHAS + NOX + RM
+ AGE + DIS + RAD + TAX + PTRATIO + LSTAT, data = trainingData,
hidden = c(5,4,3))
```

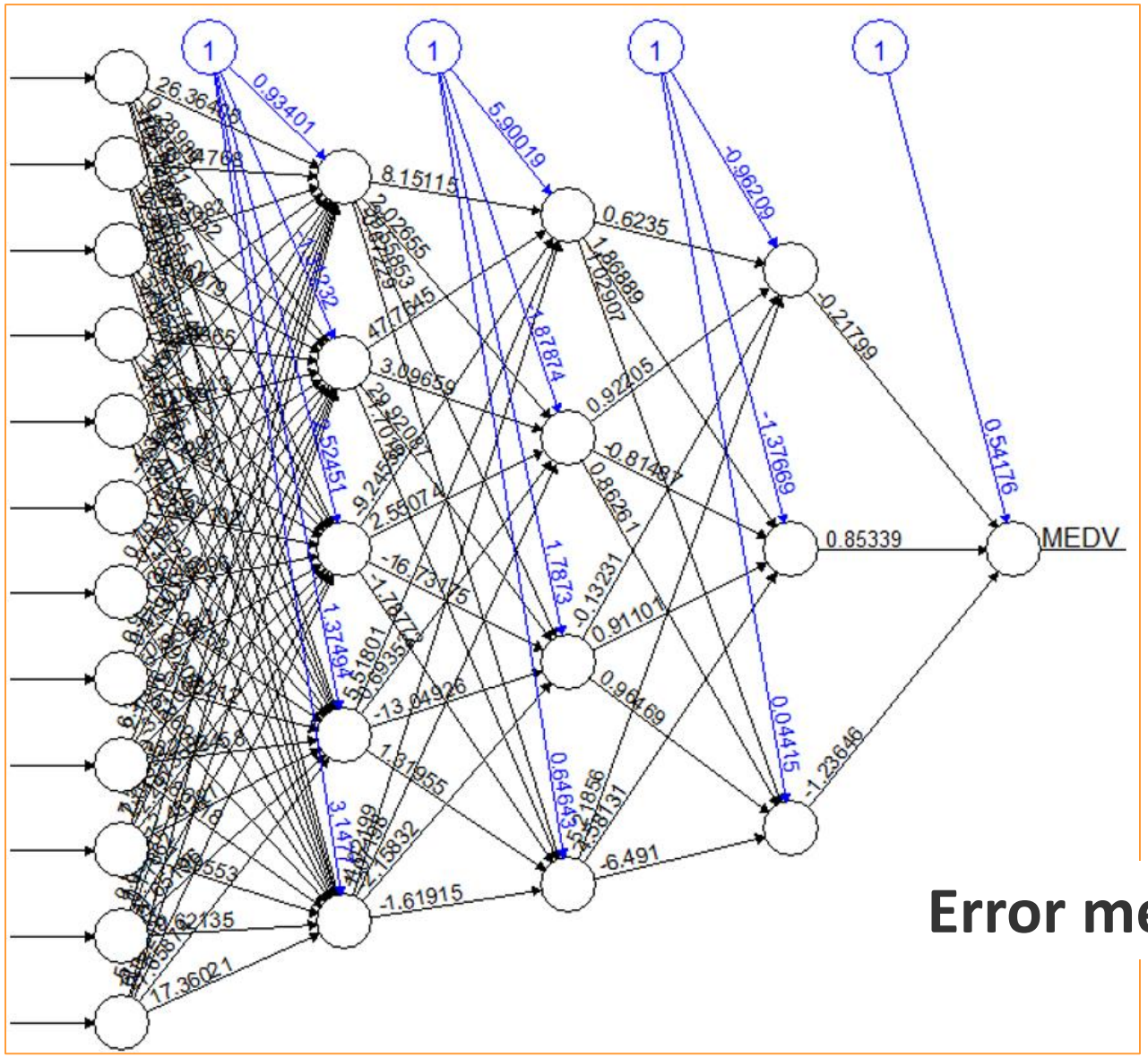
Neural network with one hidden layer and 5 hidden nodes in the hidden layer



bias terms: constant that allow the value at the indicated nodes to be shifted upward or downward

Error metric RMSE: 0.246

Neural network with three hidden layers (5 nodes in the 1st layer, 4 nodes in the 2nd layer, and 3 nodes in 3rd layer)



Error metric RMSE: 0.137

Neural Network for Wind Energy Forecasting

	A	B	C	D	E	F	G
1	TIMESTAMP	POWER	U10	V10	WS10		
2	20120101 1:00	0.2736781568	0.5348940035	-3.6602432799	3.6991204986		
3	20120101 2:00	0.0867959455	0.3308130989	-2.6764297561	2.6967969048		
4	20120101 3:00	0.0068114015	-0.0658387244	-2.0290719396	2.0301398163		
5	20120101 4:00	0.0186459868	-0.4195494463	-1.7990895574	1.847361625		
6	20120101 5:00	0.0348118328	-0.7542244222	-1.6615260214	1.8246981117		

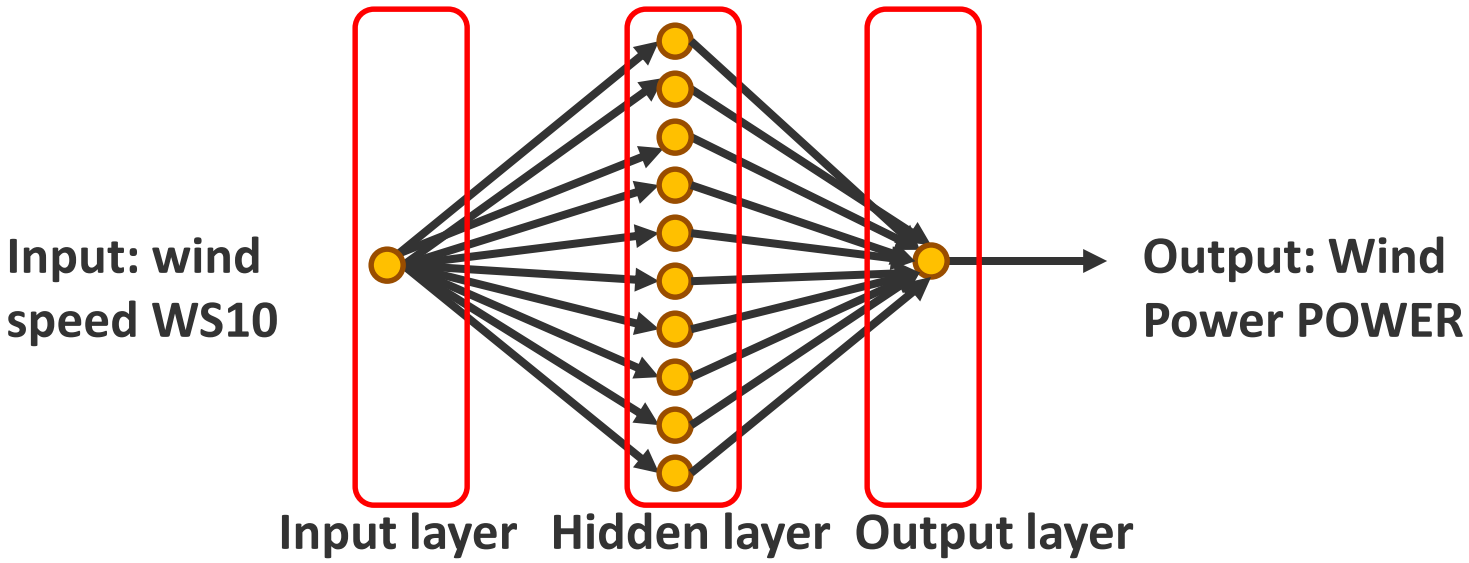
Normalized wind power

Wind speed above ground 10m

- The data is between 2012.01.01-2013.10.01, which has **15336** data records. The file contains hourly wind power measurements and wind speed
- The data includes real wind power data (**normalized to preserve anonymity**) for a wind farm in Australia

Neural Network for Wind Energy Forecasting

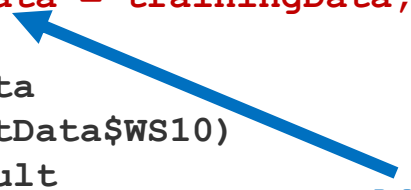
- One input node in the input layer
- One hidden layer, 10 hidden nodes in the hidden layer
- One output node



R code to build neural network model

- For 15336 data records, we divide them into two parts. We use 3000 data record (from 12001 to 15000 data records) are used as training data to build neural network model. The wind speed in the last 336 records (from 15001 to 15336 data records) are used as test data

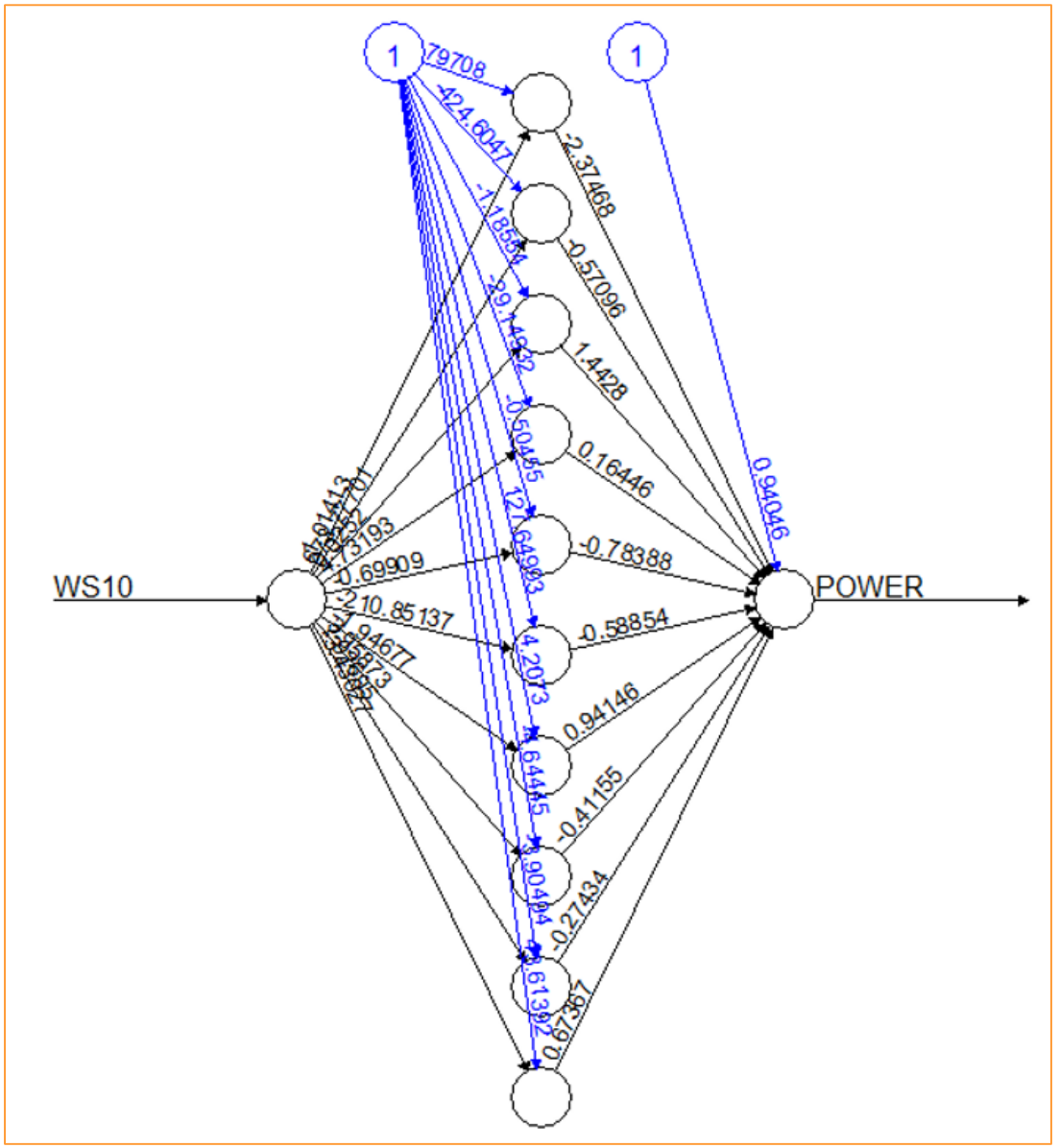
```
#calling neuralnet function to build the model.
#The function will return a neural network objec that can make predictions

powerModel <- neuralnet(POWER ~ WS10, data = trainingData, hidden = 10)

#generate the prediction on the test data
modelResults <- compute(powerModel, testData$WS10)
powerPredictedNN = modelResults$net.result

# plot the neural networks
plot(powerModel)

# Calculate error
errorNN <- powerPredictedNN - testData$POWER
rmse(errorNN)
```



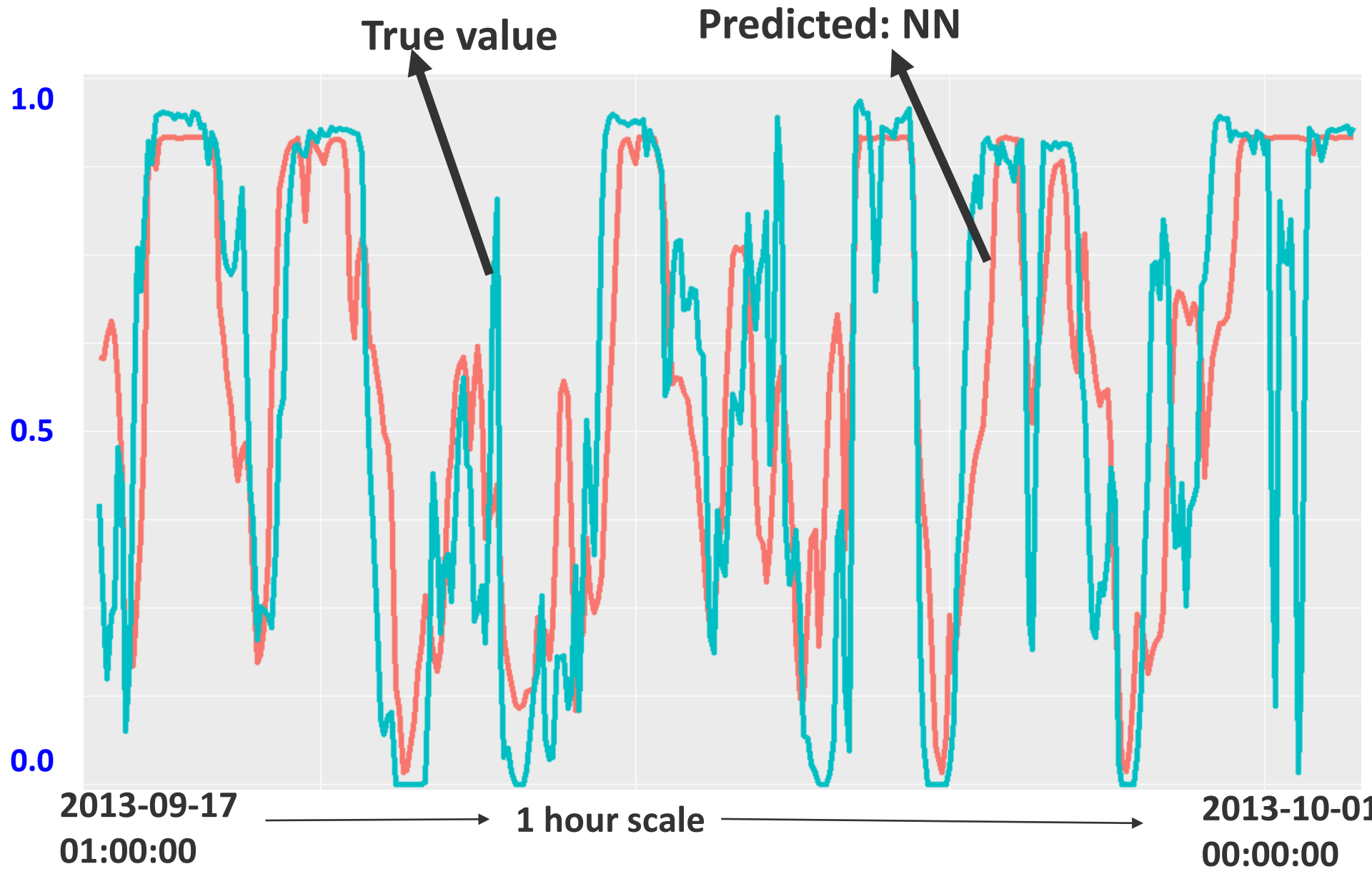
Neural network training

Neural network model with one hidden layer (10 hidden nodes)

- **Q: can we use all 15000 data records as training data?**



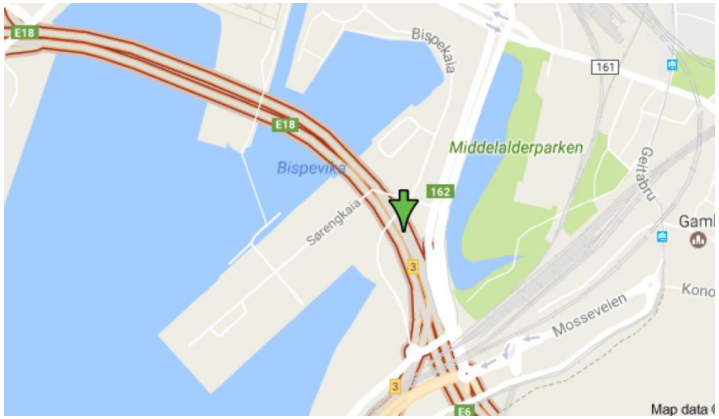
True Value & Predicted Wind Power. RMSE=0.2277



MORE CONSIDERATIONS…

Interested to forecast something new?

- What else may you need to forecast, in an electricity market context?
  - solar power generation
  - energy prices
  - electricity load
  - battery health prediction



- What else may you need to forecast, in general?
  - how many travelers in Oslo this summer?
  - traffic flow prediction in Operatunnel
  - Norway's wealth fund hit $1 trillion

TensorFlow

- **TensorFlow: An open-source software library for Machine Intelligence, developed by Google. The primary language in which TensorFlow machine learning models are created and trained is Python.**
- **It supports Python, R, and (experimental API) for Java**

- **Tensor: The central unit of data in TensorFlow. A tensor consists of a set of primitive values shaped into an array of any number of dimensions.**
- **Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them.**
- **More information: https://www.tensorflow.org/**

References

- **M. Nielsen, “Neural Network and Deep Learning” free online book.** https://neuralnetworksanddeeplearning.com/