Deep Learning for Renewable Energy Forecasting

26.03.2026

Shiliang Zhang | Postdoc, University of Oslo, Norway

Learning objectives

Throughout this lecture, you will learn:



Neural Network

Learn neural network technique in plain language, mathematical language, and programming language



Application

Apply neural network for wind energy forecasting

Industry Invited Talk Today



Eirik Mathiesen

Director, Power Systems - Energy controls E&C Division

Kongsberg Maritime AS

Intelligent Energy Management System (iEMS)

KONGSBERG

Kongsberg Maritime empowers all maritime sectors—from offshore energy to fishing to cargo and naval—with cutting-edge technology and solutions to accelerate the energy transition. They have been at the forefront of innovation for over 200 years and support more than 30,000 vessels worldwide, ensuring safer, more secure, and sustainable operations.

Outline



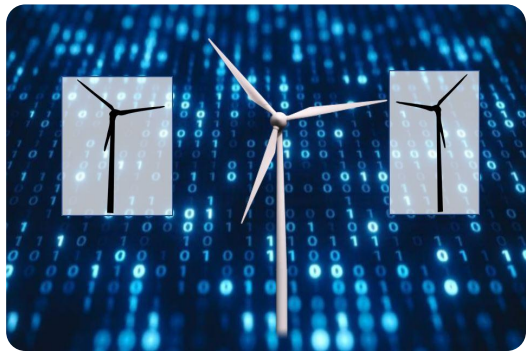
Neural Networks

Concept
Principles



House Price

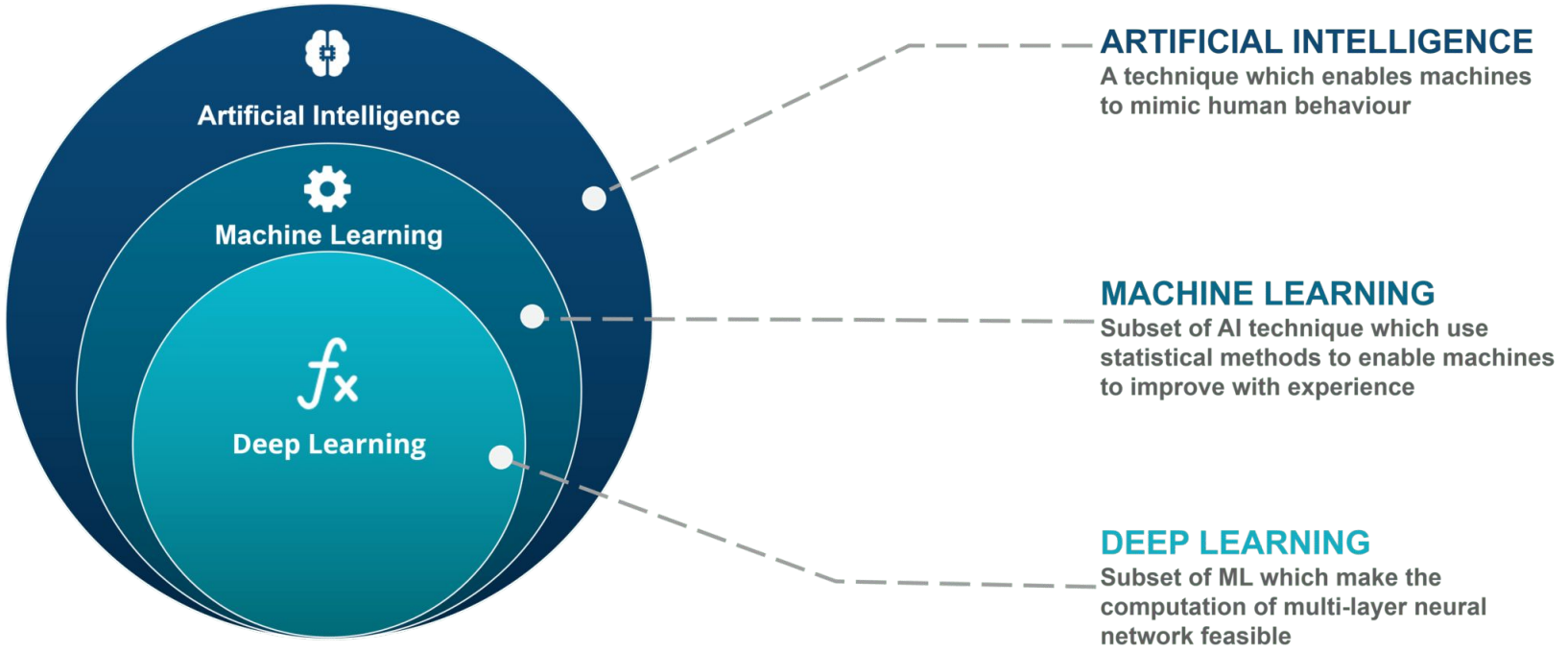
Neural Network: applications in Boston House Price Forecasting



Wind Energy

Neural Network: applications in Wind Energy Forecasting

Deep learning? Machine learning? Artificial intelligence?



Data

More data + **neural networks**

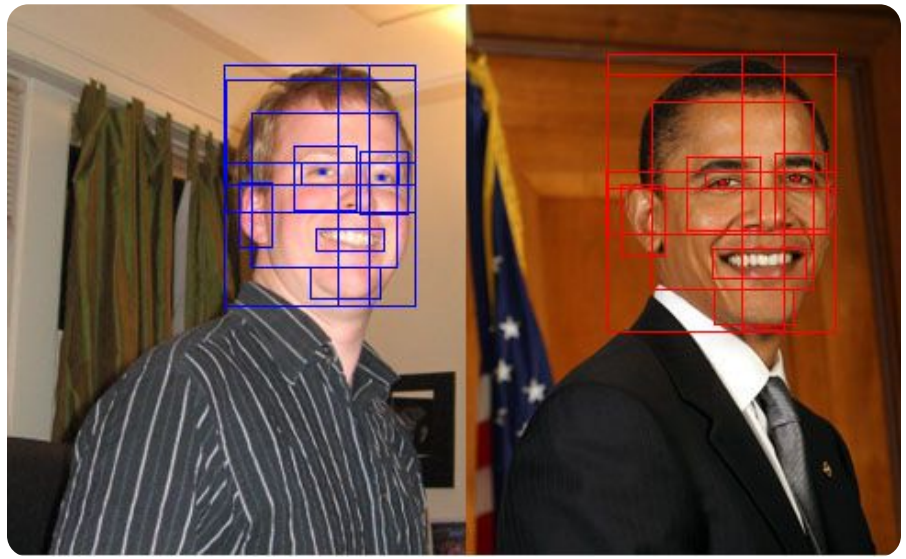
Figure source: https://www.edureka.co/blog/ai-vs-machine-learning-vs-deep-learning/

01

NEURAL NETWORK AND ITS APPLICATION

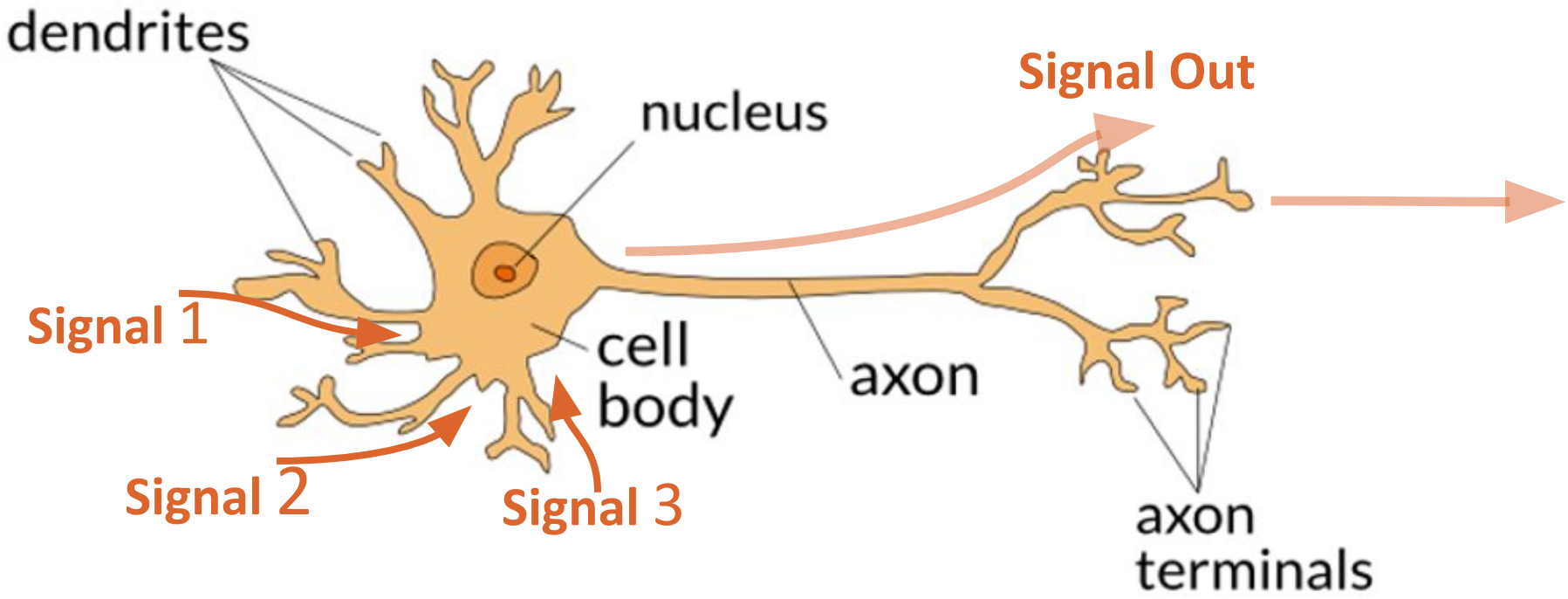
Artificial Neural Network

Q: The same person in two pictures?



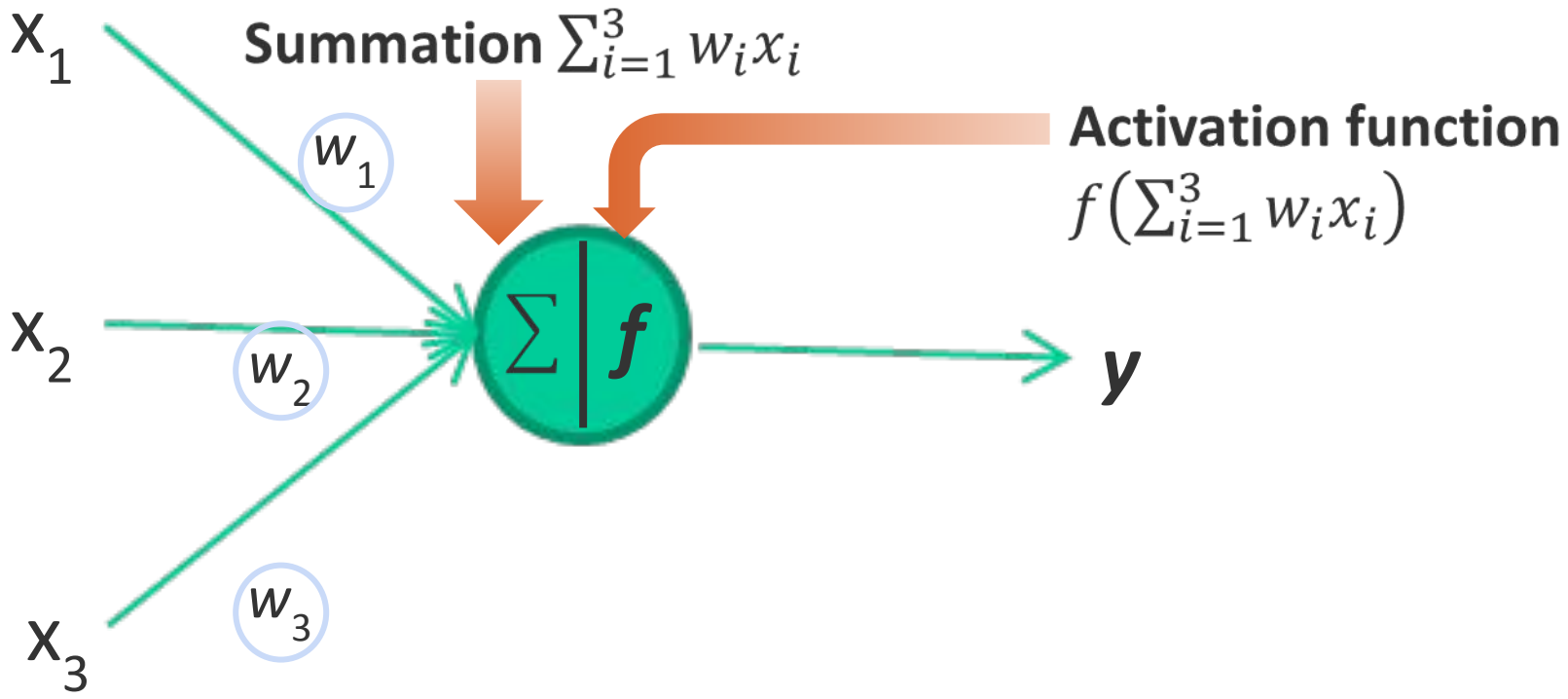
- Computers are great at solving problems, but many problems can't be mathematically defined easily. Facial recognition is a typical example.
- Artificial neural network's main idea: Enables computers to process information in a similar way as our own biological brains and our own nervous system.

Biological model for a single neuron



- Incoming signals received by dendrites. Signals are weighted according to its relative importance.
- Signals accumulate and a threshold is reached. Then the cell produces the output signal, transmitted down to the axon.
- At the axon terminals, the output signal is passed to neighboring neurons for further processing.

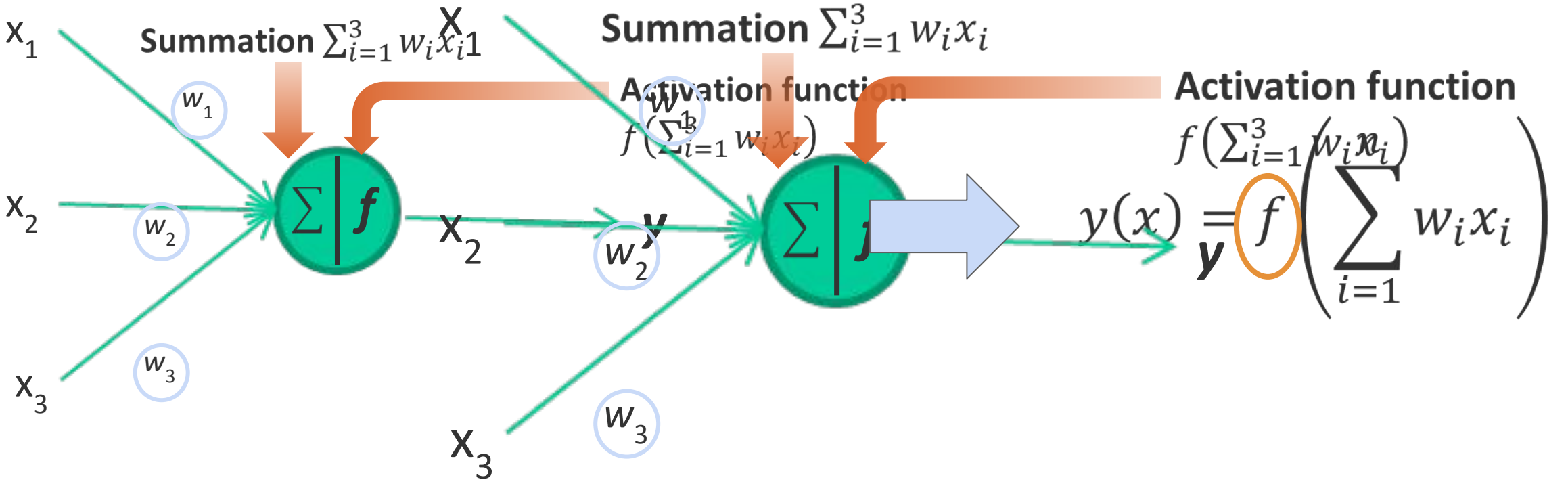
From biological model to a single artificial neuron



- Map from input signals (x variables) received by the dendrites to output signal (y variable).

- Each input signal weighted (w values) according to their importance.
- Input signals summed, then activated by an activation function (f).
- Output signal produced and passed to the next processing loop

From biological model to a single artificial neuron

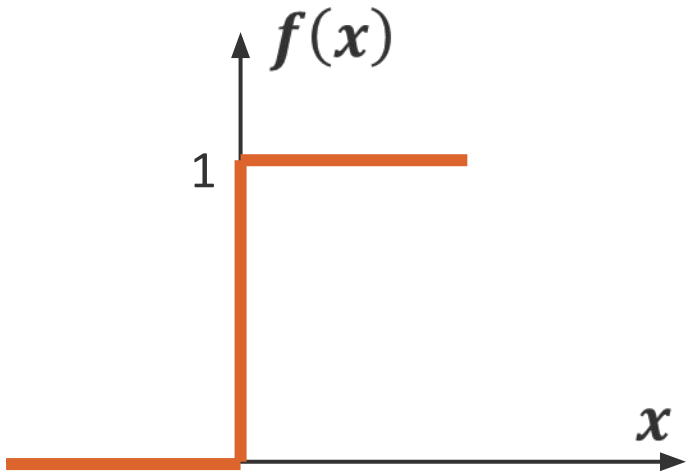


Activation function f(.): can be threshold activation function, rectified linear unit (ReLU), sigmoid function, hyperbolic tangent, or others.

Activation function – threshold activation

Threshold Activation Function: It results in an output signal of "1" only when the weighted-sum of the input signals is no less than zero.

$$f(x) = 1_{x \geq 0} = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$$

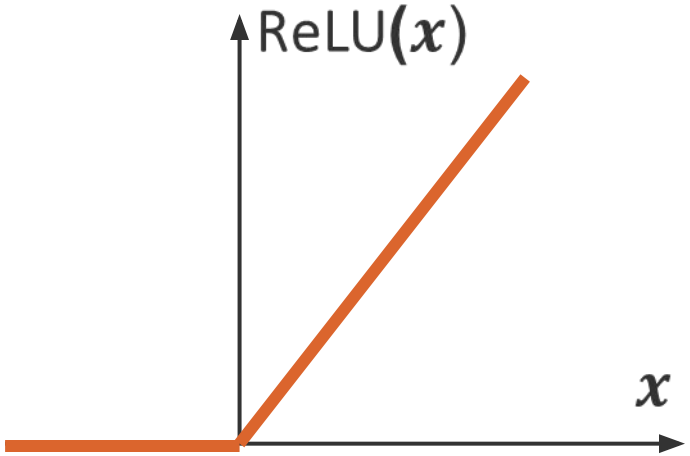


This activation function is rarely used, since this function is not derivable that can be a problem in updating neural network parameters.

Activation function – Rectified Linear Unit (ReLU)

ReLU: It returns 0 if the input x is no larger than 0, otherwise it returns x

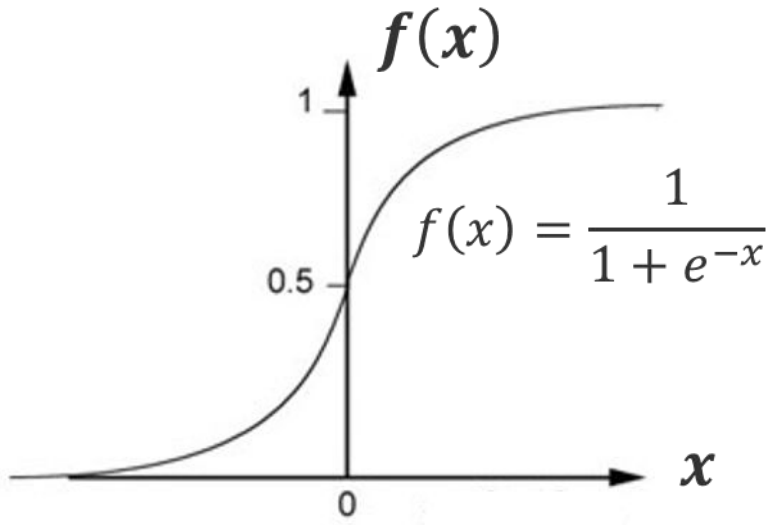
$$f(x) = max(x, 0) = \begin{cases} x, x \geq 0 \\ 0, x < 0 \end{cases}$$



ReLU is computationally efficient, continuous, and differentiable (except when x=0), and is widely used in neural networks. ReLU is also a non-linear activation function that can capture non-linear relationships between input and output.

Activation function – sigmoid

Sigmoid activation function: continuous and differentiable across the entire range of the input



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

- The derivative function and the 2nd derivative can be easily computed and expressed in terms of $f(x)$.

$$f'(x) = \left(\frac{1}{1+e^{-x}}\right)' = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{(1+e^{-x})-1}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} - \frac{1}{(1+e^{-x})^2} = f(x) - f^2(x) = f(x)[1 - f(x)]$$

Simplifies the neural network parameter update

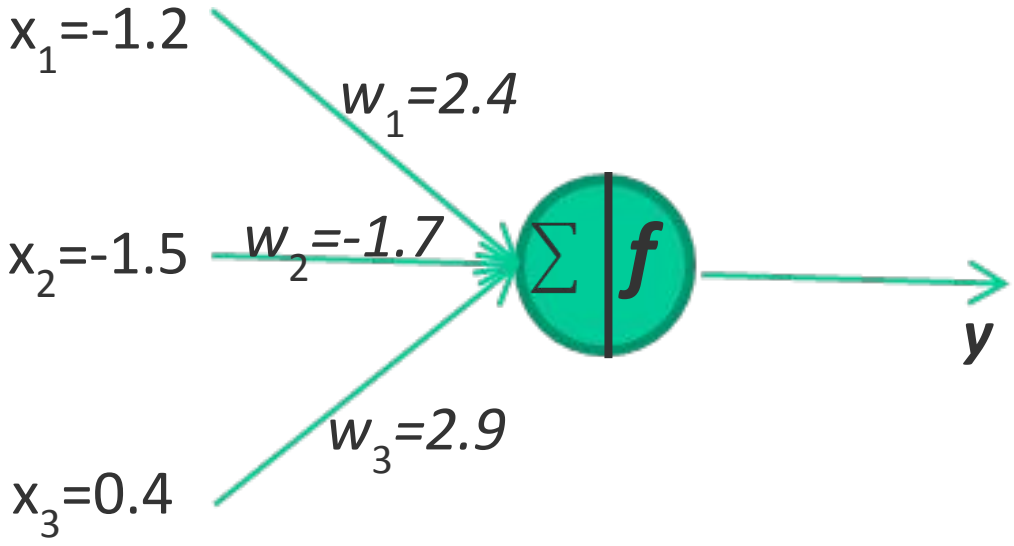
Example: Calculate NN output with given input

1. First calculate the weighted sum of input signals:

$$x = \sum_{i=1}^{3} w_i x_i$$

$$= (-1.2) \times 2.4 + (-1.5) \times (-1.7) + 0.4 \times 2.9$$

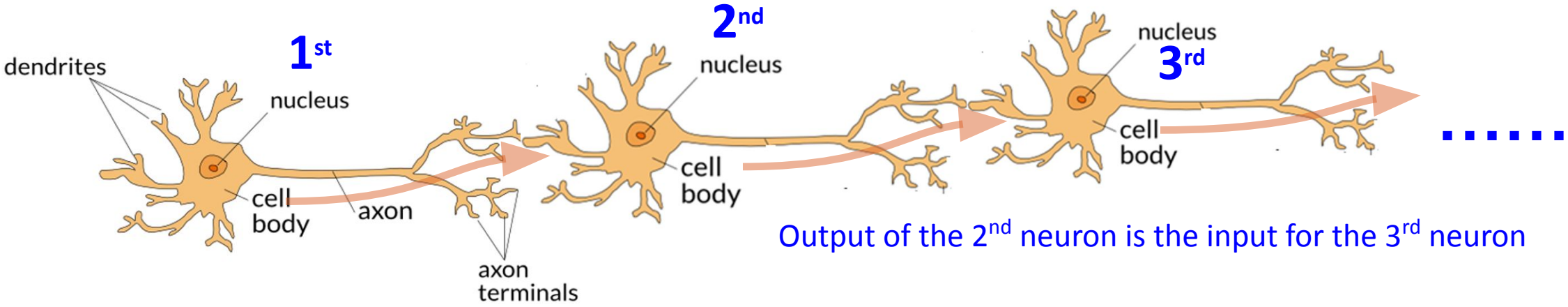
$$= 0.83$$



2. Then, we calculate the output signal based on the activation function.
   - If we use threshold activation function, the output signal is given by: $\boldsymbol{y = f(0.83) = 1}$
   - If we use ReLU activation function, the output signal is given by: $\boldsymbol{y = f(0.83) = max(0.83, 0) = 0.83}$
   - If we use sigmoid activation function, the output signal is given by:

$$\boldsymbol{y = f(0.83) = \frac{1}{1 + e^{-0.83}} = 0.696}$$

From a single neuron to neural networks

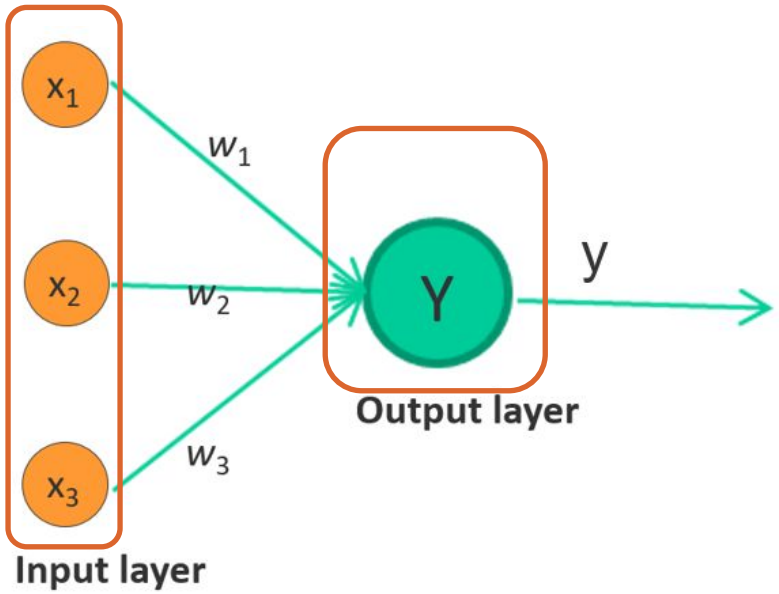


Output of the 2nd neuron is the input for the 3rd neuron

Artificial neural networks: Using multiple artificial neurons to construct complex models

Network topology: number of neurons in the network, number of layers, how layers are connected

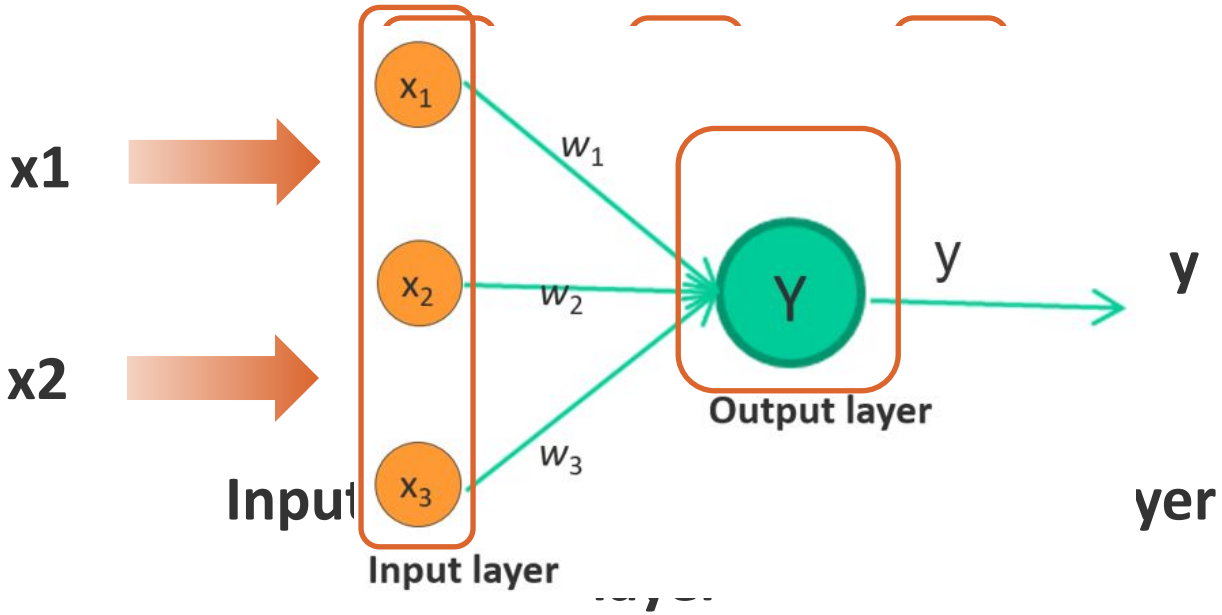
Network topology: A single layer neural network



Input layer: A set of neurons that receives the input data

Output layer: The layer with neuron(s) that generates the output

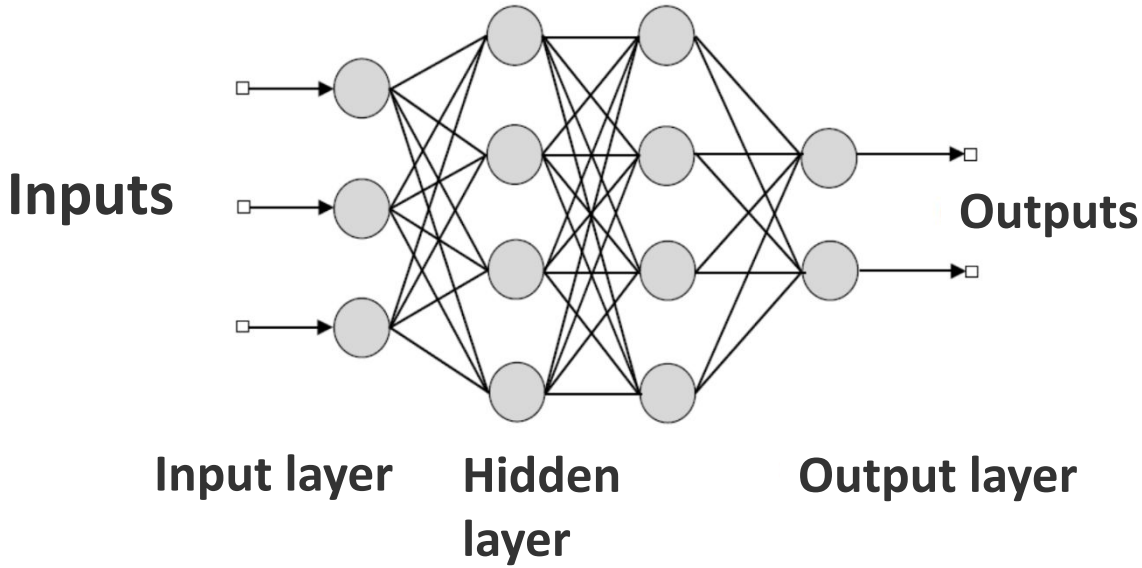
Network topology: a neural network with one hidden layer



Q: Why we need hidden layer(s)?

De facto standard topology

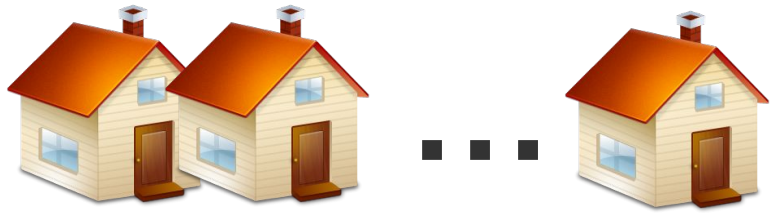
This is the de facto standard neural network topology. This topology is also called Feedforward Neural Network since the information flows unidirectionally from the input layer to the hidden layer and then to the output layer.



Q: How this neural network (NN) can work for us?

Make predictions using NN!
For a given input, we want to predict the output that, the predicted output is close to its ground truth.

De facto standard topology

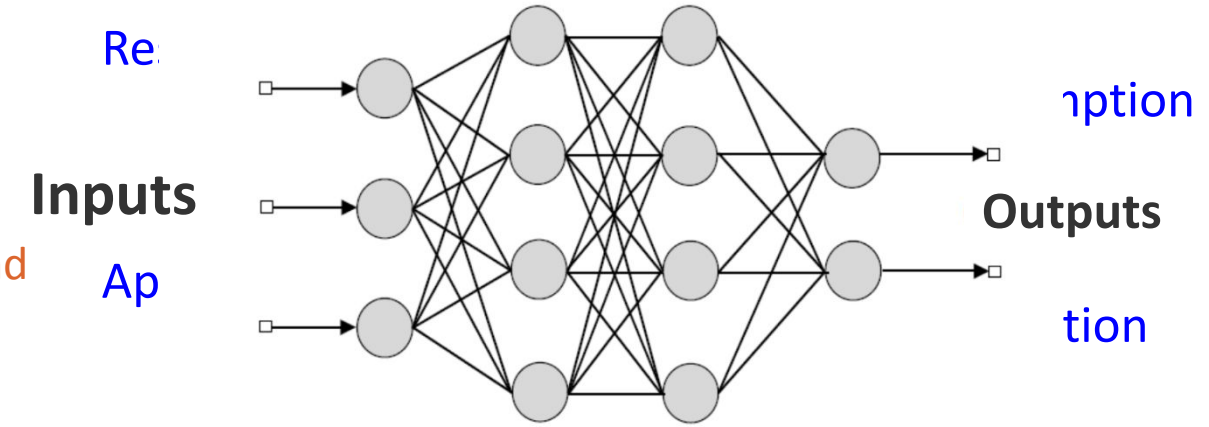


Input: Number of residents, PV panels, electricity appliances

Output: Energy consumption and generation for house 1, 2,…, 999, 1000



1001# house consumption and generation?



Q: How this neural network (NN) can work for us?

Make predictions using NN!
For a given input, we want to predict the output that, the predicted output is close to its ground truth.

To make good predictions, we need to train the neural network, or to update the NN parameters (weights) using training data

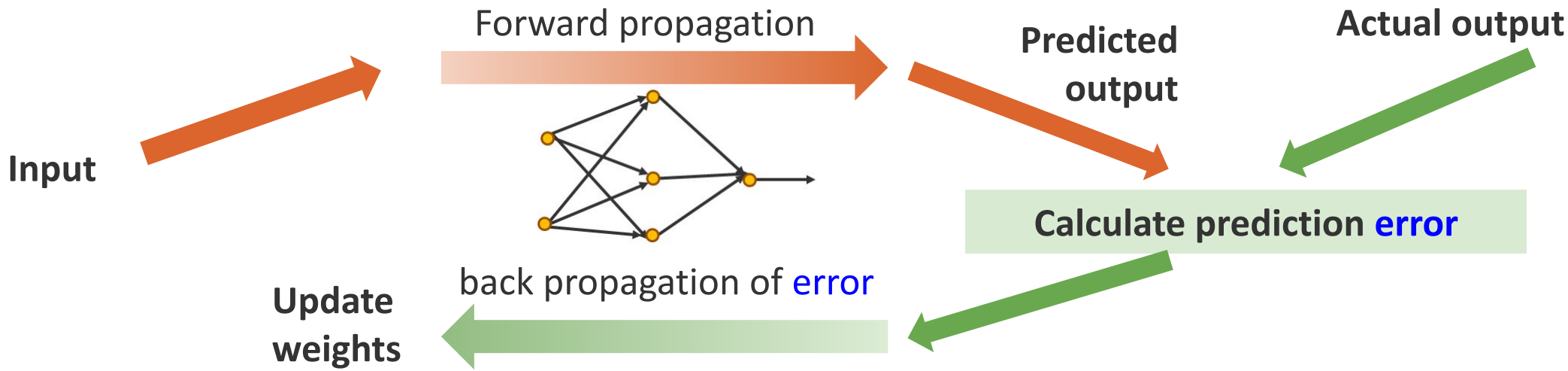
How to train? Using what method to train?

Train a neural network: An example

How to train: There are many algorithms. They train the network by iteratively modifying the weights until the error between the output predicted by NN and its ground truth is minimized.

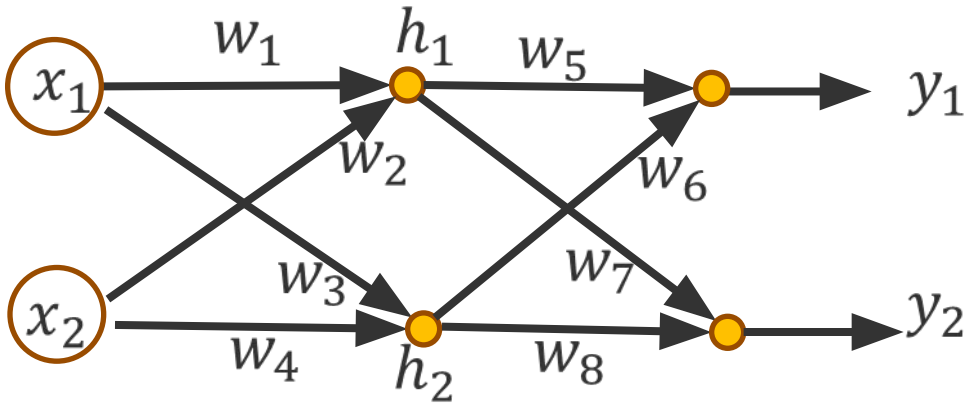
Steps:

- Forward propagation: neurons are activated in sequence from input layer to output layer, applying each neuron's weights and activation function along the way. Upon reaching the final layer, an predicted output is produced.
- Backpropagation (BP) algorithm: uses **gradient descent** to update weights. Starting from random weights, BP algorithm adjusts the weights according to the error between the output produced by the neural network and its ground truth. BP algorithm applies error propagation from output layer to input layer, and gradually fine tunes the network weights to minimize the error.



Forward propagation

- Input data: x_1=0.05, x_2=0.10
- Output data: y_1=0.20, y_2=0.80
- Initial random weight: w_1=0.10, w_2=0.15, w_3=0.20, w_4=0.25, w_5=0.30, w_6=0.35, w_7=0.40, w_8=0.45
- *Goal*: for the input, training neural networks to find the weight $w_i (i = 1,2 \ldots 8)$ to produce the output that is as close as possible to the actual output



- Forward propagation from the input layer to the hidden layer. We calculate the weighted summation of the nodes in the hidden layer, and then calculate the output of the hidden nodes.

$$sum_{h1} = w_1 x_1 + w_2 x_2 = 0.10*0.05 + 0.15*0.1 = 0.02$$
$$sum_{h2} = w_3 x_1 + w_4 x_2 = 0.20*0.05 + 0.25*0.1 = 0.035$$

$$out_{h1} = sigmoid(sum_{h1}) = 0.5049998$$
$$out_{h2} = sigmoid(sum_{h2}) = 0.5087491$$

- Calculate the predicted output at the output layer

$$sum_{y1} = w_5 out_{h1} + w_6 out_{h2}$$
$$= 0.3*0.5049998 + 0.35*0.5087491 = 0.329562125$$
$$sum_{y2} = w_7 out_{h1} + w_8 out_{h2}$$
$$= 0.4*0.5049998 + 0.45*0.5087491 = 0.430937015$$

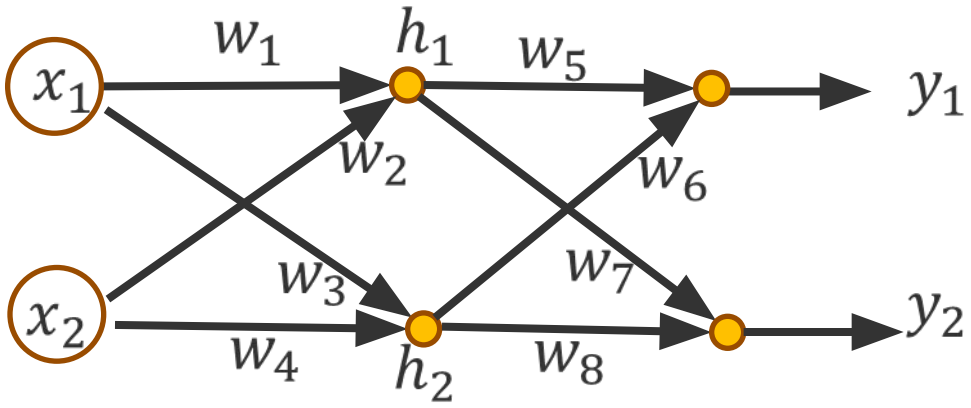
$$out_{y1} = sigmoid(sum_{y1}) = 0.5816528$$
$$out_{y2} = sigmoid(sum_{y2}) = 0.6060974$$

Update NN parameters (weights) – start from collecting errors

y_1=0.20
y_2=0.80

$$out_{y1} = sigmoid(sum_{y1}) = 0.5816528$$

$$out_{y2} = sigmoid(sum_{y2}) = 0.6060974$$



The error is defined as sum of squared error between the predicted and actual outputs

$$E_{total} = \frac{1}{2}\sum(y_i - out_{yi})^2 = \frac{1}{2}\sum(actual\ output - predicted\ output)^2$$

In our example, the error is

$$E_{out1} = \frac{1}{2}(actual\ output - predicted\ output)^2 = \frac{1}{2}(0.2 - 0.5816528)^2 = 0.0728294299$$

$$E_{out2} = \frac{1}{2}(0.8 - 0.6060974)^2 = 0.0187991091$$

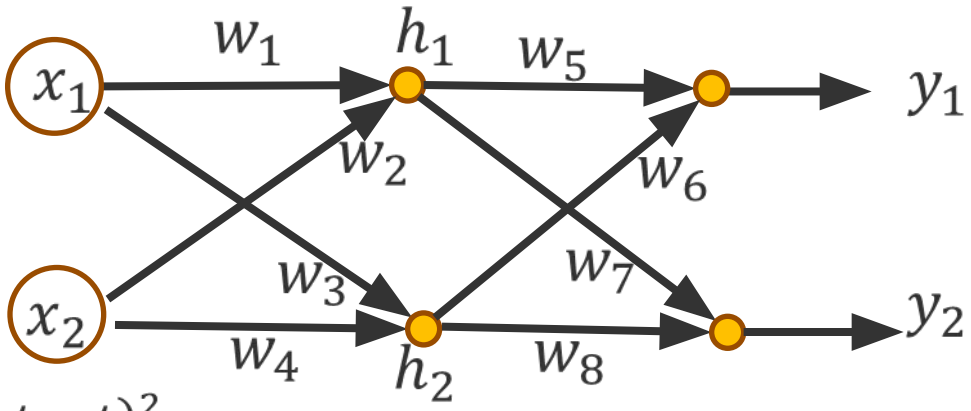
$$E_{total} = E_{out1} + E_{out2} = 0.0728294299 + 0.0187991091 = 0.091628539$$

Update NN parameters (weights) – start from collecting errors

y_1=0.20
y_2=0.80

$$out_{y1} = sigmoid(sum_{y1}) = 0.5816528$$

$$out_{y2} = sigmoid(sum_{y2}) = 0.6060974$$

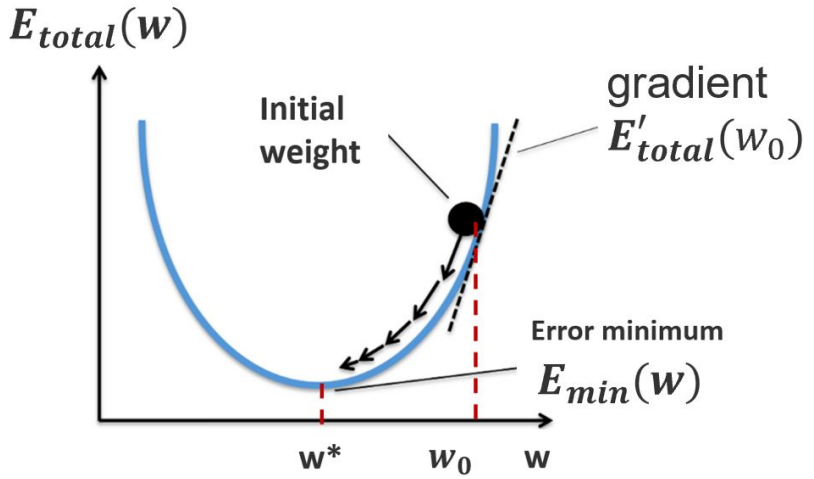


$$E_{total} = \frac{1}{2}\sum(y_i - out_{yi})^2 = \frac{1}{2}\sum(actual\ output - predicted\ output)^2$$

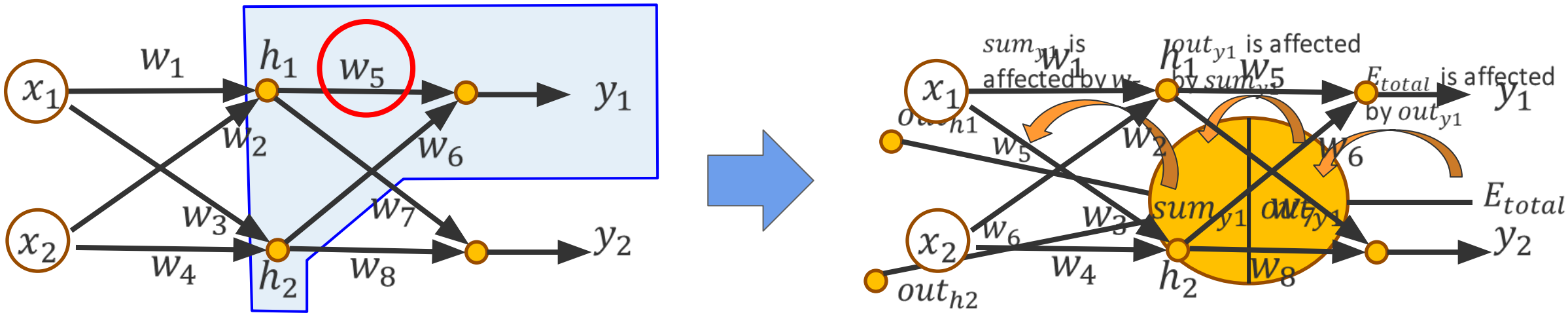
The error is an **function of the weights**: $E_{total}(w)$

Now we use the error to update the weight via **backpropagation**

Gradient descent: The gradient – calculated by the first-order derivative – of the error indicates in which direction the error increase fastest. To reduce the error, weight should be updated in the opposite direction of the gradient



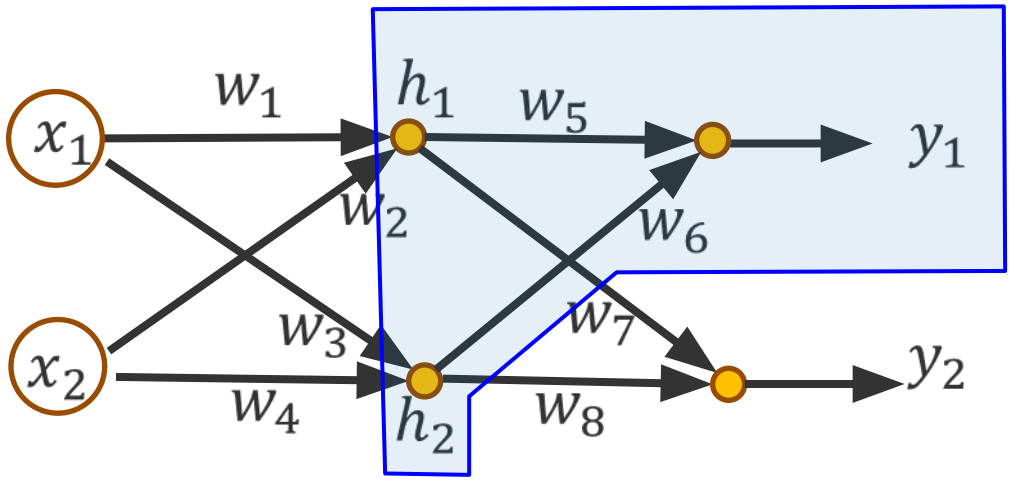
Gradient descent calculation: Taking example of W_5



We calculate gradient descent for W_5, calculated by partial derivative of the error with respect to W_5, following the **Chain Rule**:

$$\frac{\partial E_{total}}{\partial w_5} = \boxed{\frac{\partial E_{total}}{\partial out_{y1}}} * \boxed{\frac{\partial out_{y1}}{\partial sum_{y1}}} * \boxed{\frac{\partial sum_{y1}}{\partial w_5}}$$

Gradient descent calculation: Taking example of W_5



y_1=0.20
y_2=0.80

$out_{h1} = sigmoid(sum_{h1}) = 0.5049998$

$out_{y1} = sigmoid(sum_{y1}) = 0.5816528$

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{y1}} * \frac{\partial out_{y1}}{\partial sum_{y1}} * \frac{\partial sum_{y1}}{\partial w_5}$$

$$E_{total} = \frac{1}{2}(y_1 - out_{y1})^2 + \frac{1}{2}(y_2 - out_{y2})^2 \rightarrow \frac{\partial E_{total}}{\partial out_{y1}} = 2 * \frac{1}{2}(y_1 - out_{y1}) * (-1) + 0 = -(y_1 - out_{y1})$$

$$= -(0.2 - 0.5816528) = 0.3816528$$

$$out_{y1} = sigmoid(sum_{y1}) \rightarrow \partial out_{y1}/\partial sum_{y1} = out_{y1}(1 - out_{y1}) = 0.5816528 * (1 - 0.5816528) = 0.24333282$$

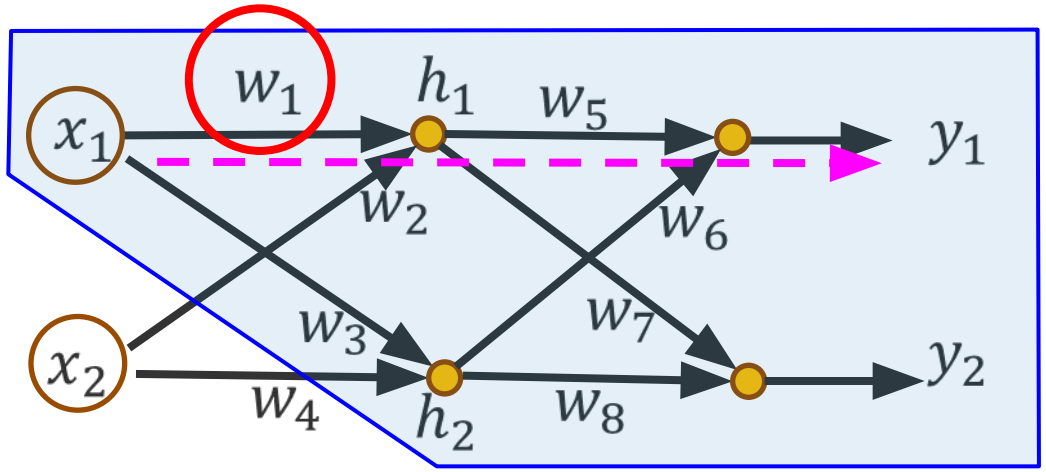
$$sum_{y1} = w_5 out_{h1} + w_6 out_{h2} \rightarrow \partial sum_{y1}/\partial w_5 = out_{h1} = 0.5049998$$

$$f'(x) \equiv \left(\frac{1}{1+e^{-x}}\right)' = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{(1+e^{-x})-1}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} - \frac{1}{(1+e^{-x})^2} = f(x) - f^2(x) = f(x)[1 - f(x)]$$

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{y1}} * \frac{\partial out_{y1}}{\partial sum_{y1}} * \frac{\partial sum_{y1}}{\partial w_5} = 0.3816528 * 0.24333282 * 0.5049998 = 0.046898651$$

New weight: $w_5^+ = w_5 - \alpha \frac{\partial E_{total}}{\partial w_5} = 0.3 - 0.5 * 0.046898651 = 0.276550674$; where α is learning rate ($\alpha \in (0,1)$)

Gradient descent calculation: Taking example of W_1



y_1=0.20
y_2=0.80

$out_{h1} = sigmoid(sum_{h1}) = 0.5049998$

$out_{h2} = sigmoid(sum_{h2}) = 0.5087491$

$out_{y1} = sigmoid(sum_{y1}) = 0.5816528$

$out_{y2} = sigmoid(sum_{y2}) = 0.6060974$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial (E_{out1}+E_{out2})}{\partial w_1} = \frac{\partial E_{out1}}{\partial w_1} + \frac{\partial E_{out2}}{\partial w_1} \Rightarrow \frac{\partial E_{out1}}{\partial w_1} = \frac{\partial E_{out1}}{\partial out_{y1}} * \frac{\partial out_{y1}}{\partial sum_{y1}} * \frac{\partial sum_{y1}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial sum_{h1}} * \frac{\partial sum_{h1}}{\partial w_1}$$

$E_{out1} = \frac{1}{2}(y_1 - out_{y1})^2 \rightarrow \frac{\partial E_{out1}}{\partial out_{y1}} = 2 * \frac{1}{2}(y_1 - out_{y1}) * (-1) = -(y_1 - out_{y1}) = -(0.2 - 0.5816528) = 0.3816528$

$out_{y1} = sigmoid(sum_{y1}) \rightarrow \partial out_{y1}/\partial sum_{y1} = out_{y1}(1 - out_{y1}) = 0.5816528 * (1 - 0.5816528) = 0.24333282$

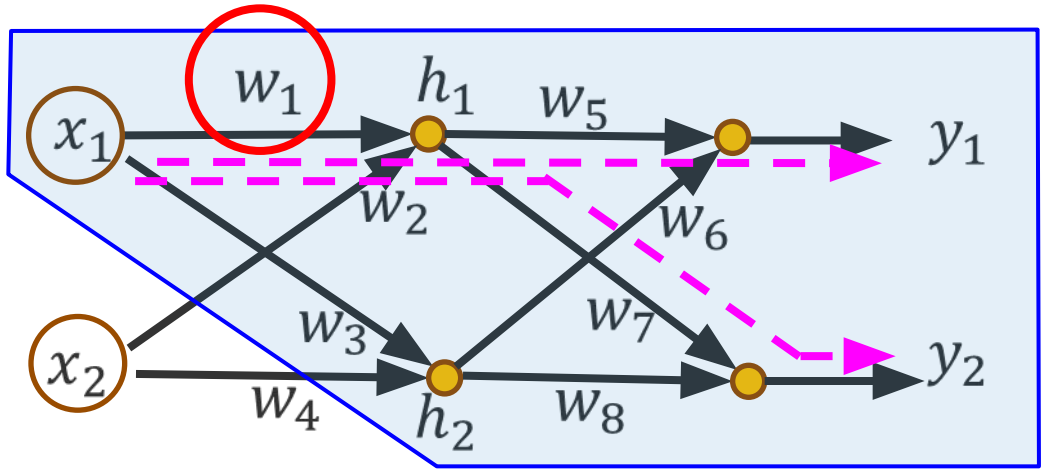
$sum_{y1} = w_5 out_{h1} + w_6 out_{h2} \rightarrow \partial sum_{y1}/\partial out_{h1} = w_5 = 0.3$

$out_{h1} = sigmoid(sum_{h1}) \rightarrow \partial out_{h1}/\partial sum_{h1} = out_{h1}(1 - out_{h1}) = 0.5049998 * (1 - 0.5049998) = 0.249975002$

$sum_{h1} = w_1 x_1 + w_2 x_2 \rightarrow \partial sum_{h1}/\partial w_1 = x_1 = 0.05$

$$\frac{\partial E_{out1}}{\partial w_1} = 0.3816528 * 0.24333282 * 0.3 * 0.249975002 * 0.05 = 0.000348223$$

Gradient descent calculation: Taking example of W_1



y_1=0.20
y_2=0.80

$out_{h1} = sigmoid(sum_{h1}) = 0.5049998$
$out_{h2} = sigmoid(sum_{h2}) = 0.5087491$
$out_{y1} = sigmoid(sum_{y1}) = 0.5816528$
$out_{y2} = sigmoid(sum_{y2}) = 0.6060974$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial (E_{out1}+E_{out2})}{\partial w_1} = \frac{\partial E_{out1}}{\partial w_1} + \boxed{\frac{\partial E_{out2}}{\partial w_1}} \Rightarrow \frac{\partial E_{out2}}{\partial w_1} = \boxed{\frac{\partial E_{out2}}{\partial out_{y2}}} * \boxed{\frac{\partial out_{y2}}{\partial sum_{y2}}} * \boxed{\frac{\partial sum_{y2}}{\partial out_{h1}}} * \boxed{\frac{\partial out_{h1}}{\partial sum_{h1}}} * \boxed{\frac{\partial sum_{h1}}{\partial w_1}}$$

$E_{out2} = \frac{1}{2}(y_2 - out_{y2})^2 \rightarrow \frac{\partial E_{out2}}{\partial out_{y2}} = 2 * \frac{1}{2}(y_2 - out_{y2}) * (-1) = -(y_2 - out_{y2}) = -(0.8 - 0.6060974) = -\ 0.1939026$

$out_{y2} = sigmoid(sum_{y2}) \rightarrow \partial out_{y2}/\partial sum_{y2} = out_{y2}(1 - out_{y2}) = 0.6060974 * (1 - 0.6060974) = 0.238743342$

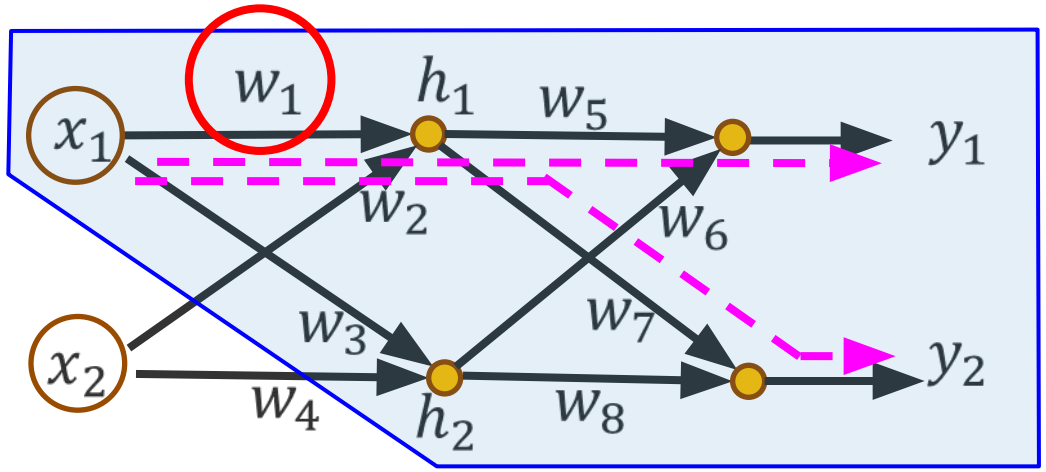
$sum_{y2} = w_7 out_{h1} + w_8 out_{h2} \rightarrow \partial sum_{y2}/\partial out_{h1} = w_7 = 0.4$

$out_{h1} = sigmoid(sum_{h1}) \rightarrow \partial out_{h1}/\partial sum_{h1} = out_{h1}(1 - out_{h1}) = 0.5049998 * (1 - 0.5049998) = 0.249975002$

$sum_{h1} = w_1 x_1 + w_2 x_2 \rightarrow \partial sum_{h1}/\partial w_1 = x_1 = 0.05$

$\frac{\partial E_{out2}}{\partial w_1} = (-0.1939026) * 0.238743342 * 0.4 * 0.249975002 * 0.05 = -0.000231442$

Gradient descent calculation: Taking example of W_1



y_1=0.20
y_2=0.80

$out_{h1} = sigmoid(sum_{h1}) = 0.5049998$
$out_{h2} = sigmoid(sum_{h2}) = 0.5087491$
$out_{y1} = sigmoid(sum_{y1}) = 0.5816528$
$out_{y2} = sigmoid(sum_{y2}) = 0.6060974$

$$\frac{\partial E_{total}}{\partial w_1} = \boxed{\frac{\partial E_{out1}}{\partial w_1}} + \boxed{\frac{\partial E_{out2}}{\partial w_1}} = 0.000348223 - 0.000231442 = 0.000116781$$

New weight: $w_1^+ = w_1 - \alpha \frac{\partial E_{total}}{\partial w_1} = 0.1 - 0.5 * 0.000116781 = 0.0999416095$

02

NEURAL NETWORK FOR HOUSE PRICE FORECASTING

An example of prediction using NN in R

Software settings in R

- Install the software package in R 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

Predict Boston house price using NN

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.
- **Inputs: 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);
- **Output: MEDV**: median price of owner-occupied homes.

Data partition and NN model structure

- We divide the 506 records into two parts: the first 75% records (1 - 380) as training data to train a neural network, the last 25% (381 - 506) as test data.

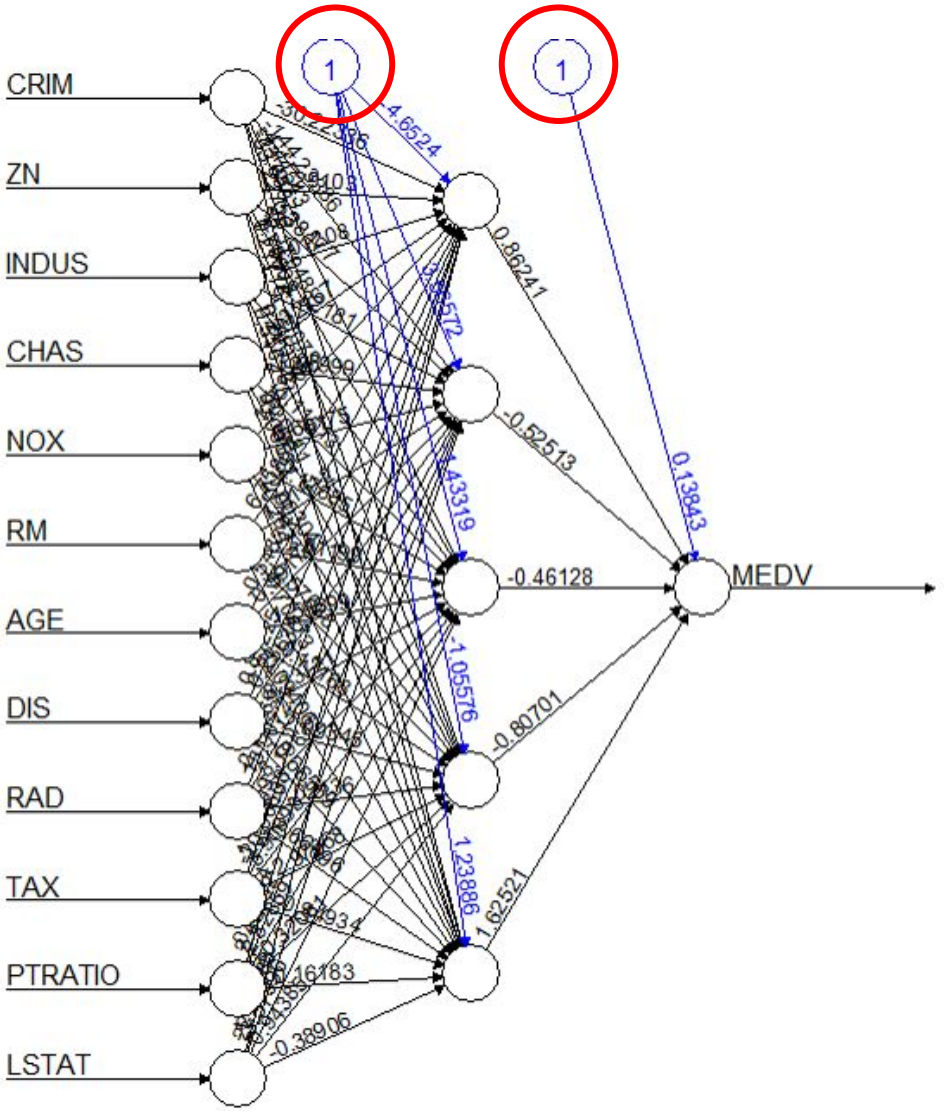
- We try two NN models. The first one is with one hidden layer that has 5 neurons/nodes

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

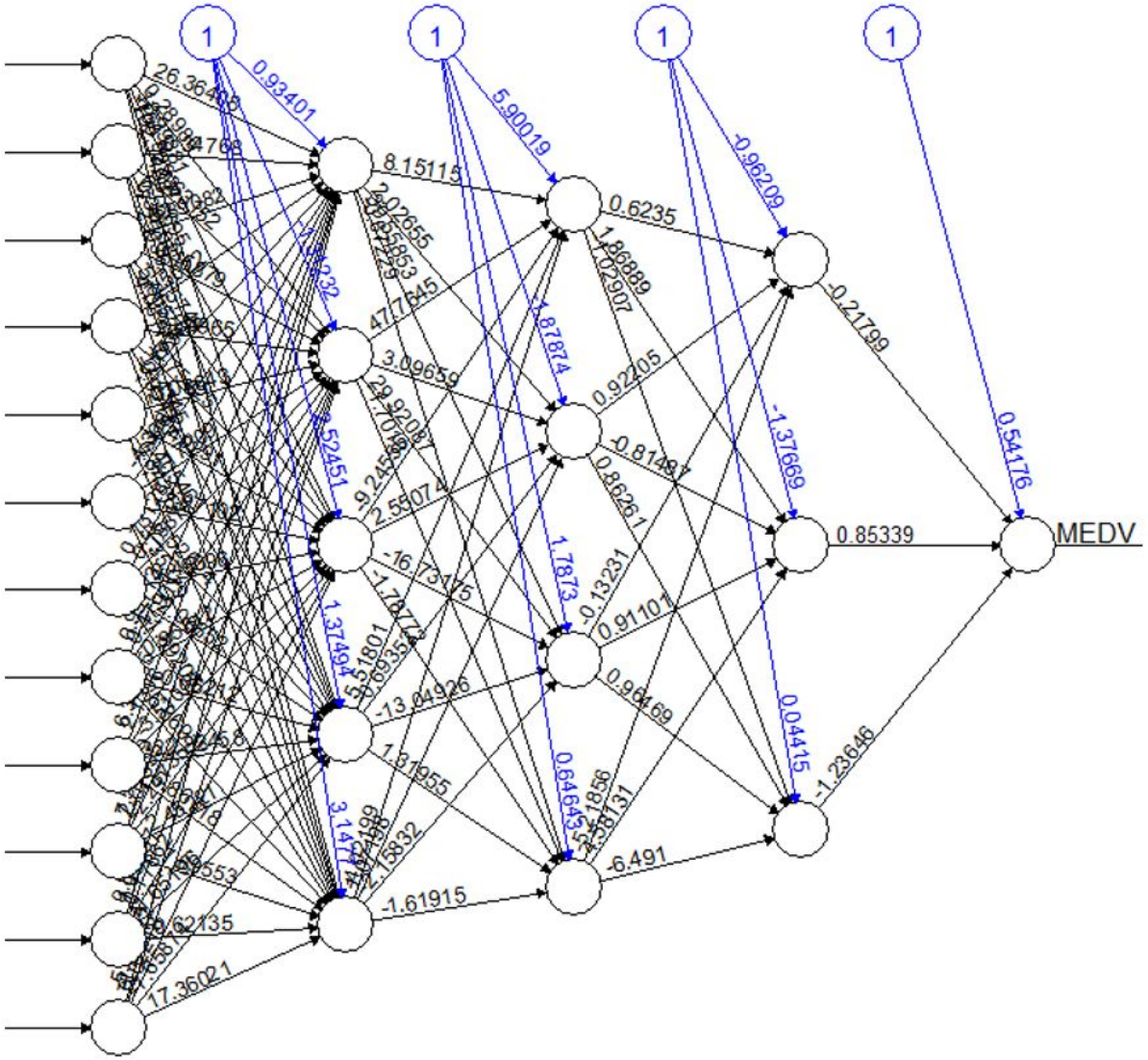
- The second neural network is with three hidden layers with 5 nodes in the first hidden layer; 4 nodes in the second hidden layer; 3 nodes in the third hidden layer

```
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 or three hidden layer(s)



Error metric RMSE: **0.246**



Error metric RMSE: **0.137**

03

NEURAL NETWORK FOR WIND ENERGY FORECASTING

Neural network for wind energy forecasting

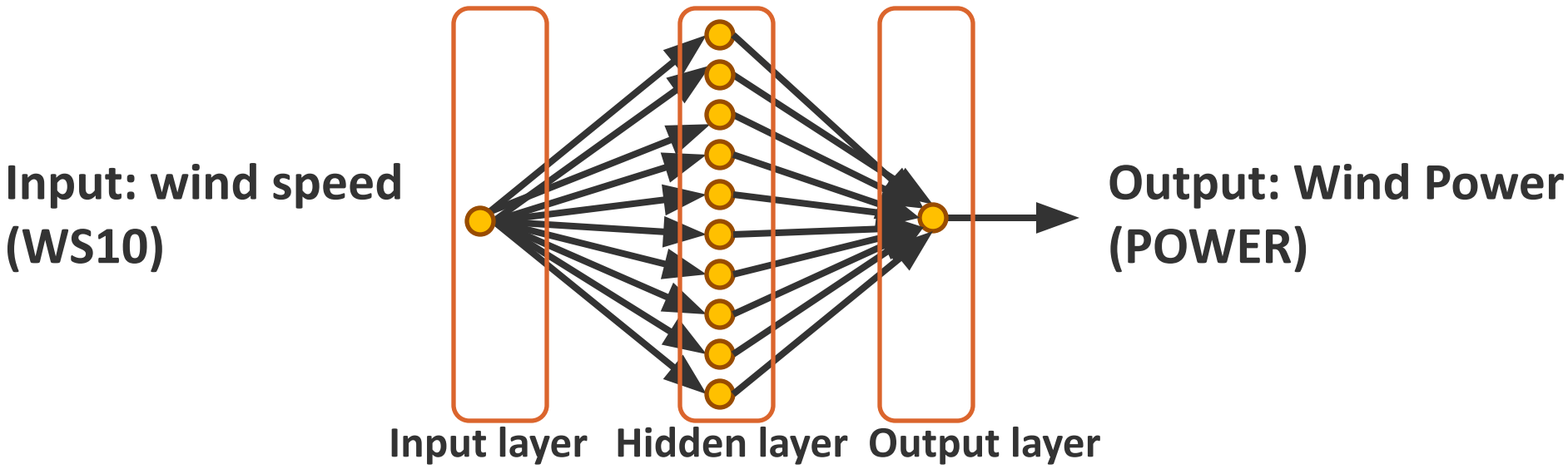
	A	B	C	D	E	F	G
1	TIMESTAMP	POWER	U10	V10	WS10		
2	20120101 1:00	0.2736781568	0.5348940035	-3.6602432799	3.6991204986		
3	20120101 2:00	0.0867959455	0.3308130989	-2.6764297561	2.6967969048		
4	20120101 3:00	0.0068114015	-0.0658387244	-2.0290719396	2.0301398163		
5	20120101 4:00	0.0186459868	-0.4195494463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

Wind power

Wind speed

- Real wind power data from a wind farm in Australia
- Hourly data recorded 2012.01.01-2013.10.01, with 15,336 records

Neural Network for Wind Energy Forecasting



- One node/neuron in the input layer
- One hidden layer with 10 nodes
- One node in the output layer

R code for the wind energy forecasting

We take 3000 records (12,001 - 15,000) as training data, the last 336 records (15,001 - 15,336) as test data.

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

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

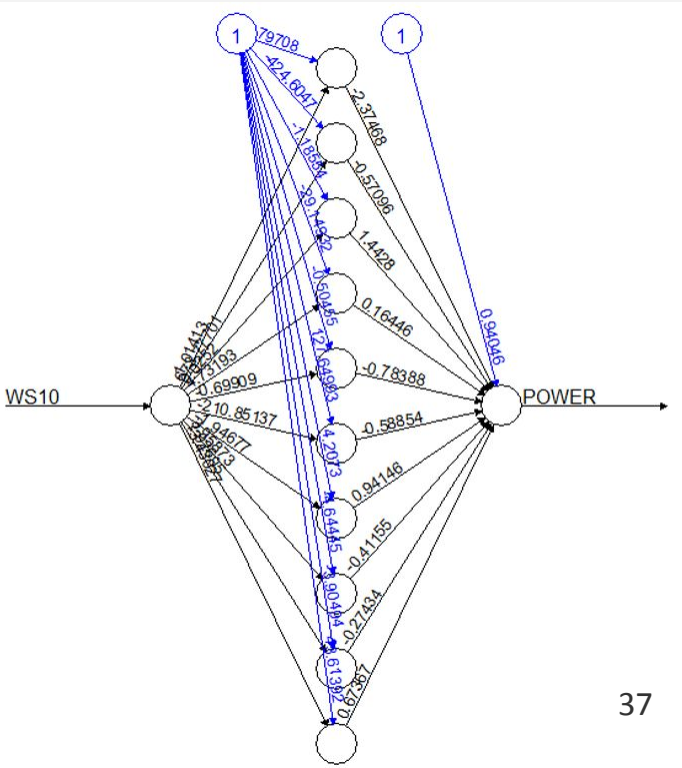
#generate the prediction on the test data
modelResults <- compute(powerModel, testData$WS10)
powerPredictedNN = modelResults$net.result

# plot the neural networks
plot(powerModel)

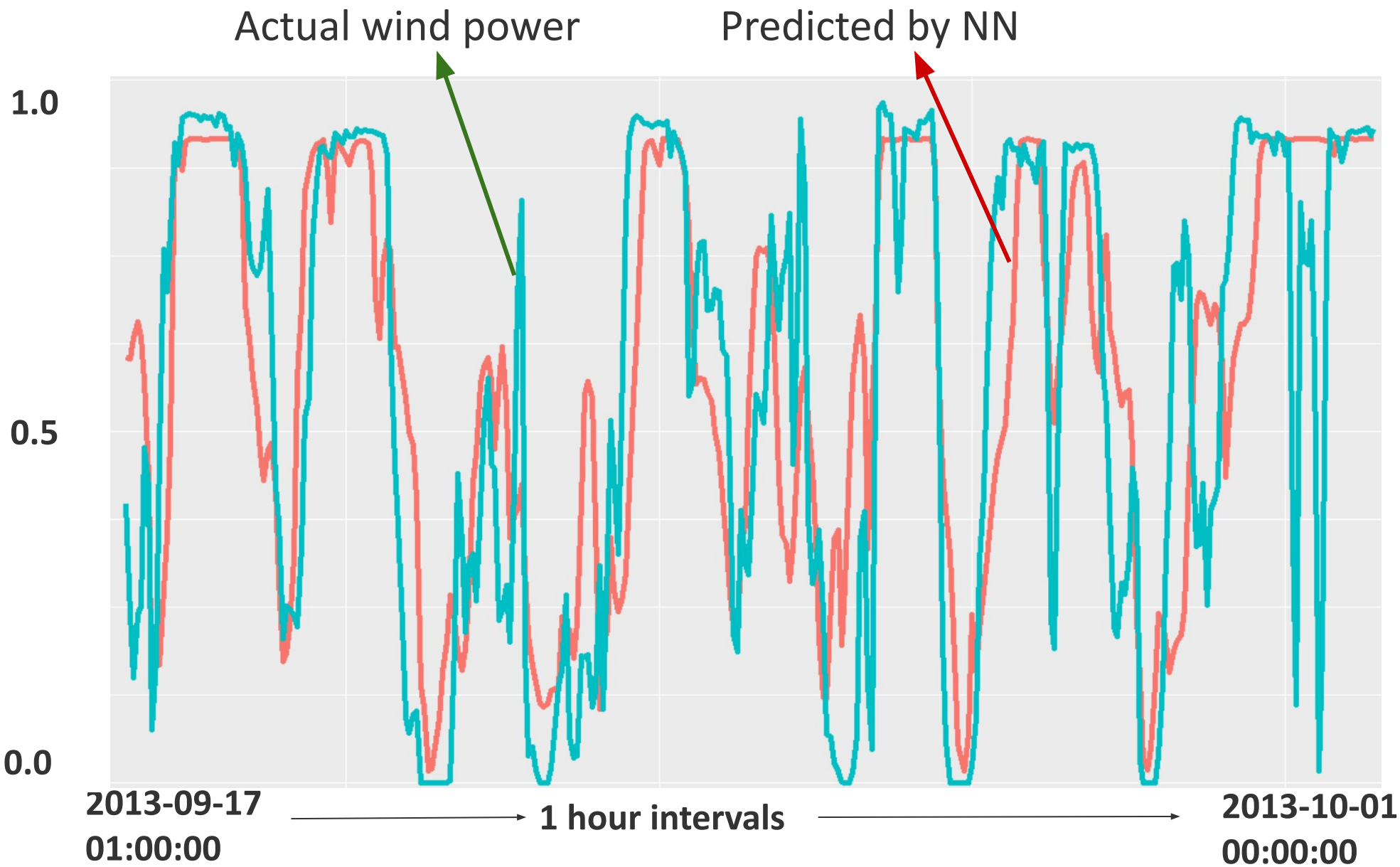
# Calculate error
errorNN <- powerPredictedNN - testData$POWER
rmse(errorNN)
```

Train the NN

Q: can we use all 15336 records as training data?



Actual & Predicted Wind Power



RMSE=0.2277

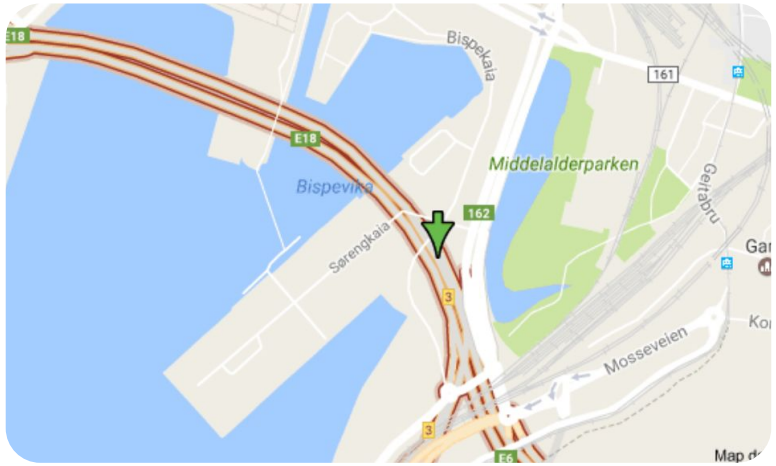
04

MORE CONSIDERATIONS

Interested to forecast something new?

What else you can forecast in energy sector?

- **Solar power generation**
- **Energy prices**
- **Electricity load**
- **Battery health prediction**





Reference

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

41