Machine Learning and Applications: Logistic Regression

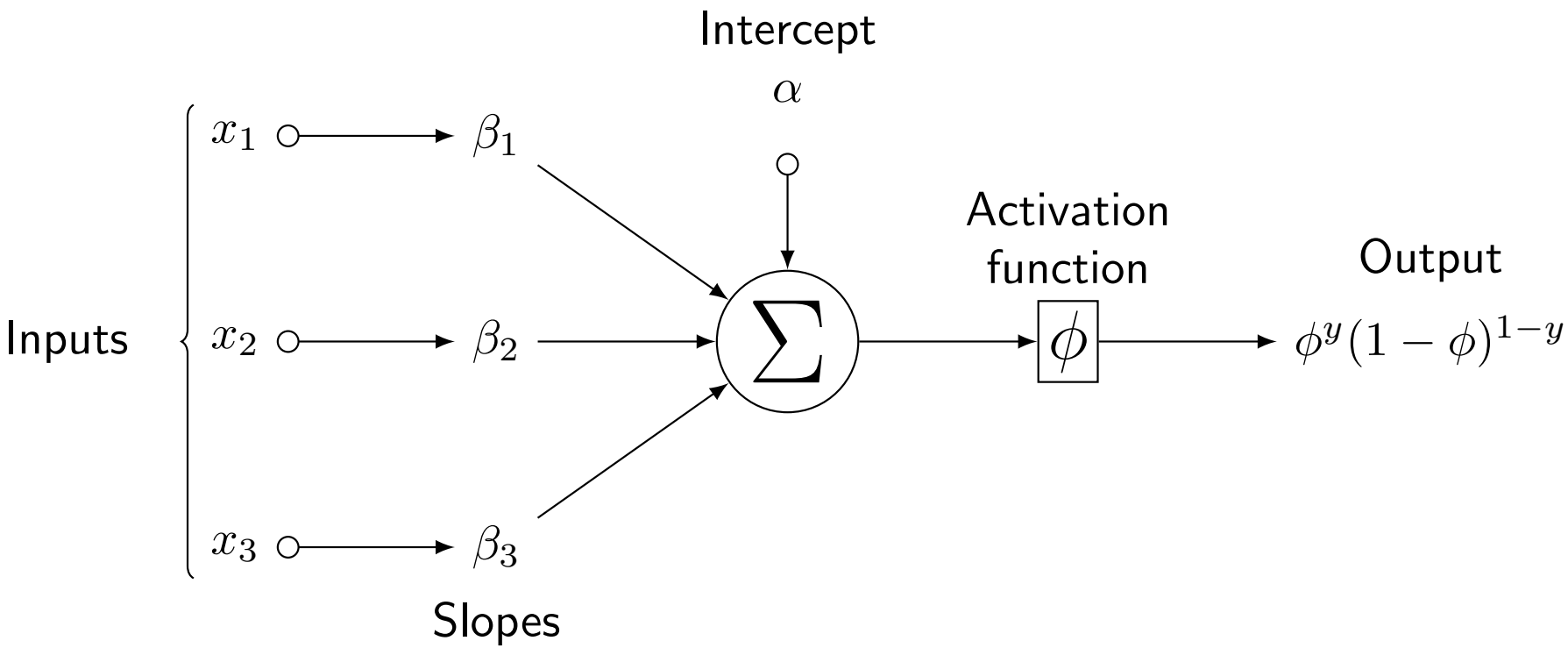
Alessio Benavoli

CSIS
University of Limerick

Logistic Regression with multiple inputs

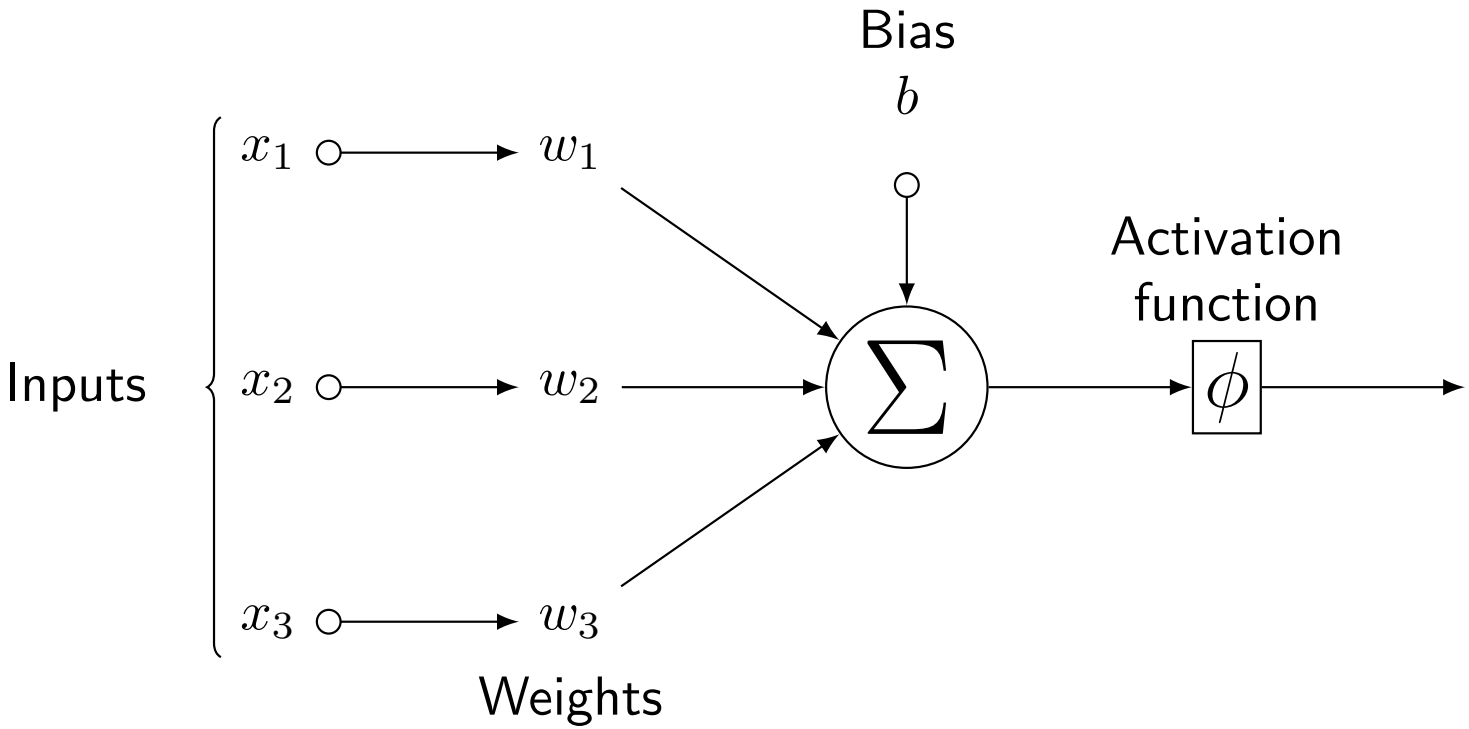
In case, we have three (or more) input variables we can generalize logistic regression as

$$\phi(\alpha + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3) = \frac{1}{1 + e^{-\alpha - \beta_1 x_1 - \beta_2 x_2 - \beta_3 x_3}}$$

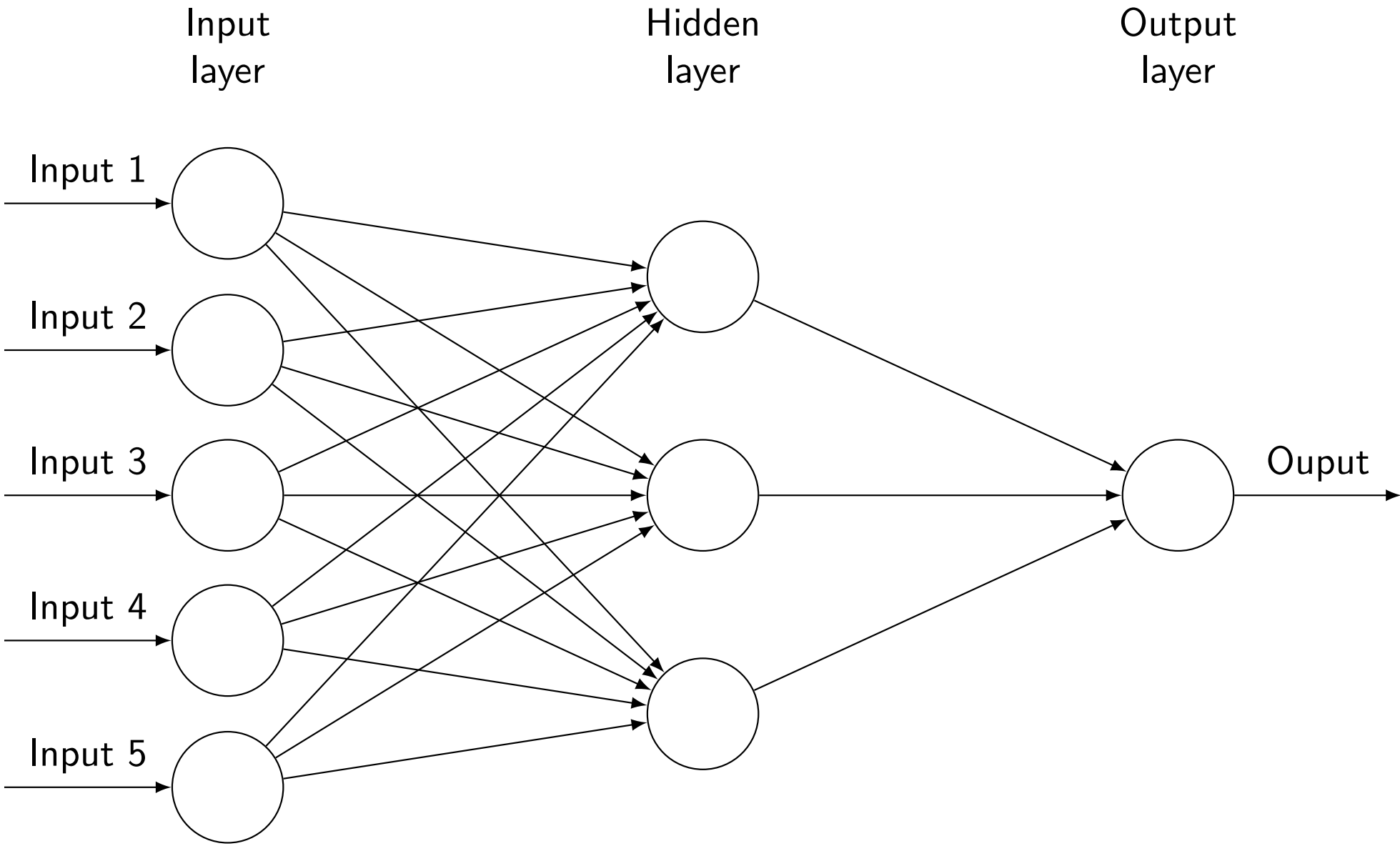


The output $\phi^y(1-\phi)^{1-y}$ means we observe $y = 1$ with probability ϕ and $y = 0$ with probability $1 - \phi$

This is an artificial neuron

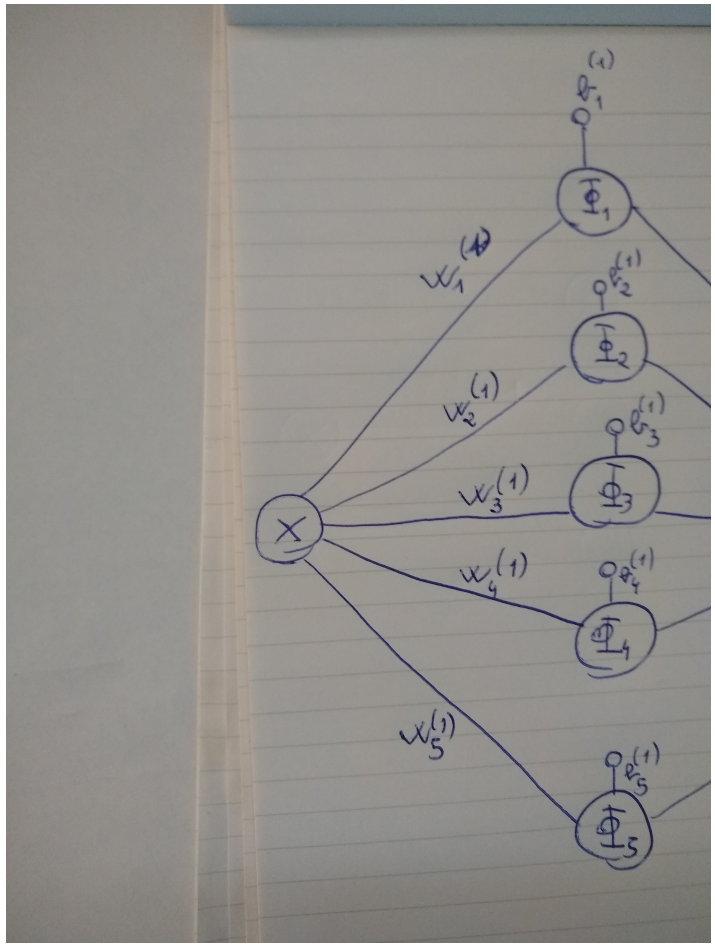


Artificial Neural Network



Artificial Neural Network

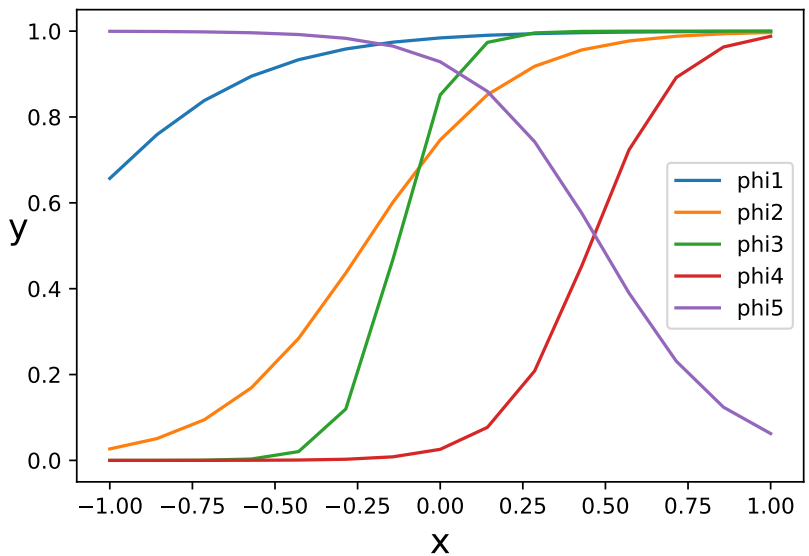
Consider the case we have only one input (x) and five hidden nodes:



Artificial Neural Network

Assume the five activation functions are:

$$\phi(w_1^{(1)}x + b_1^{(1)}) = \phi(-15.73x - 13.22) = \frac{1}{1+e^{15.73x+13.22}}$$
$$\phi(w_2^{(1)}x + b_2^{(1)}) = \phi(-4.76x - 2.66) = \frac{1}{1+e^{4.76x+2.66}}$$
$$\phi(w_3^{(1)}x + b_3^{(1)}) = \phi(6.56x - 4.18) = \frac{1}{1+e^{-6.56x+4.18}}$$
$$\phi(w_4^{(1)}x + b_4^{(1)}) = \phi(-12.03x - 3.90) = \frac{1}{1+e^{12.03x+3.90}}$$
$$\phi(w_5^{(1)}x + b_5^{(1)}) = \phi(17.41x + 5.99) = \frac{1}{1+e^{-17.41x-5.99}}$$



Artificial Neural Network

We now scale them with weights

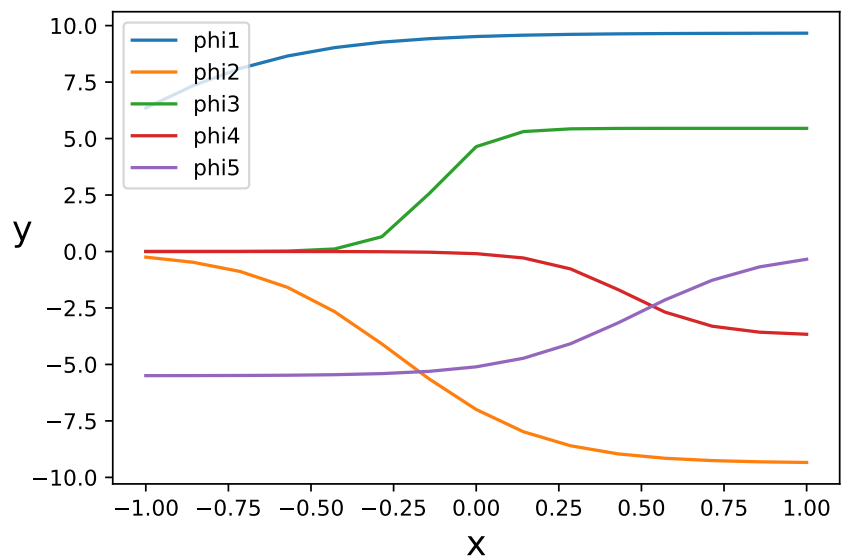
$$w_1^{(2)}\phi(w_1^{(1)}x + b_1^{(1)}) = -1.93\,\phi(-15.73x - 13.22) = -1.93\frac{1}{1+e^{15.73x+13.22}}$$

$$w_2^{(2)}\phi(w_2^{(1)}x + b_2^{(1)}) = -5.68\,\phi(-4.76x - 2.66) = -5.68\frac{1}{1+e^{4.76x+2.66}}$$

$$w_3^{(2)}\phi(w_3^{(1)}x + b_3^{(1)}) = +5.36\,\phi(6.56x - 4.18) = +5.36\frac{1}{1+e^{-6.56x+4.18}}$$

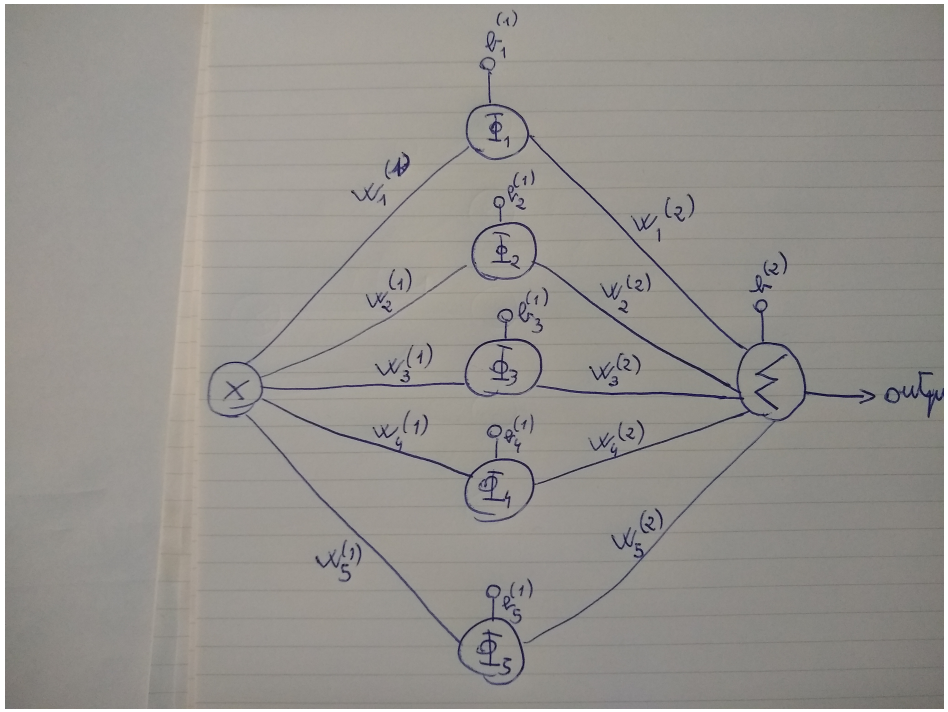
$$w_4^{(2)}\phi(w_4^{(1)}x + b_4^{(1)}) = -4.93\,\phi(-12.03x - 3.90) = -4.93\frac{1}{1+e^{12.03x+3.90}}$$

$$w_5^{(2)}\phi(w_5^{(1)}x + b_5^{(1)}) = -4.92\,\phi(17.41x + 5.99) = -4.92\frac{1}{1+e^{-17.41x-5.99}}$$



Artificial Neural Network

We sum them to get the output by also adding the bias $b^{(2)}$



The $\sum$ node is called "linear" node.

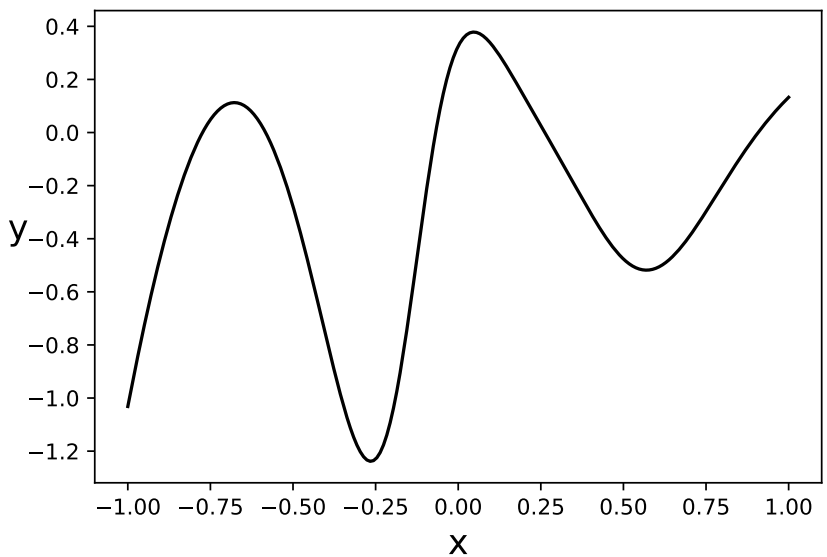
Note that in the image $^{(1)}$,$^{(2)}$ are not exponent, but are indexes denoting the first layer and the second layer.

Artificial Neural Network

Finally we sum them

$$\begin{aligned} output = &-1.93\,\phi_1(-15.73x - 13.22) - 5.68\,\phi_2(-4.76x - 2.66) \\ &+ 5.36\,\phi_3(6.56x - 4.18) - 4.93\,\phi_4(-12.03x - 3.90) \\ &- 4.92\,\phi_5(17.41x + 5.99) + 5.10 \end{aligned}$$

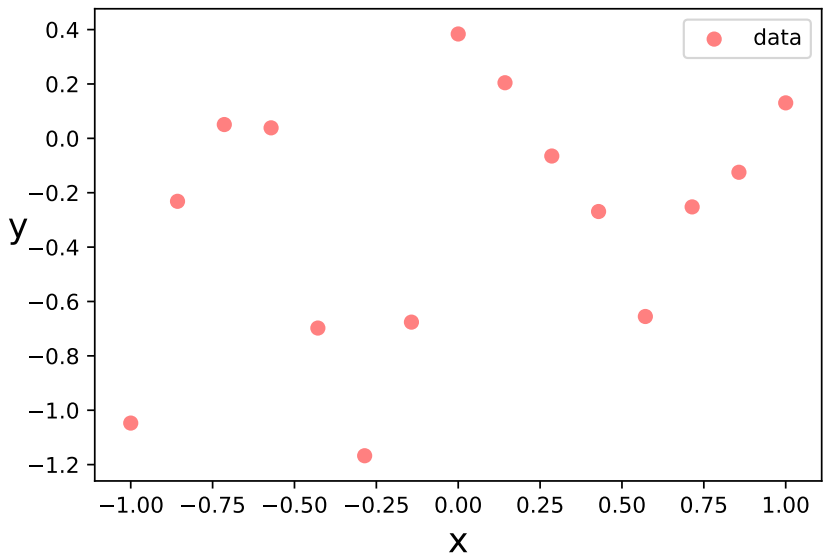
where $b^{(2)} = 5.10$ is the bias of the linear node.



We can then use ANN to solve nonlinear regression problems.

Nonlinear regression problem

Assume we have this dataset (x, y):



We cannot use linear-regression because there is not any linear trend in the data.
We could use either polynomial regression or ANN.

ANN

We choose 5 hidden nodes (as before), then the prediction of an ANN for the input x_i is:

$$\hat{y}_i = b^{(2)} + \sum_{j=1}^{5} w_j^{(2)} \, \phi(w_j^{(1)} x_i + b_j^{(1)})$$

the weights and bias (blue) are unknown, but we can estimate them from data by minimizing the squared error:

$$\arg \min_{w_j^{(1)}, b_j^{(1)}, w_j^{(2)}, b^{(2)}} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

where N is the number of observations.

ANN

Note the parallel between polynomial regression with degree $d = 5$:

$$\hat{y}_i = b^{(2)} + \sum_{j=1}^{5} w_j^{(2)} \, x_i^j$$

and ANN with 5 hidden-nodes:

$$\hat{y}_i = b^{(2)} + \sum_{j=1}^{5} w_j^{(2)} \, \phi(w_j^{(1)} x_i + b_j^{(1)})$$

the weights and bias (blue) are unknown, but we can estimate them from data by minimizing the squared error:

$$\arg \min_{\text{blue parameters}} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

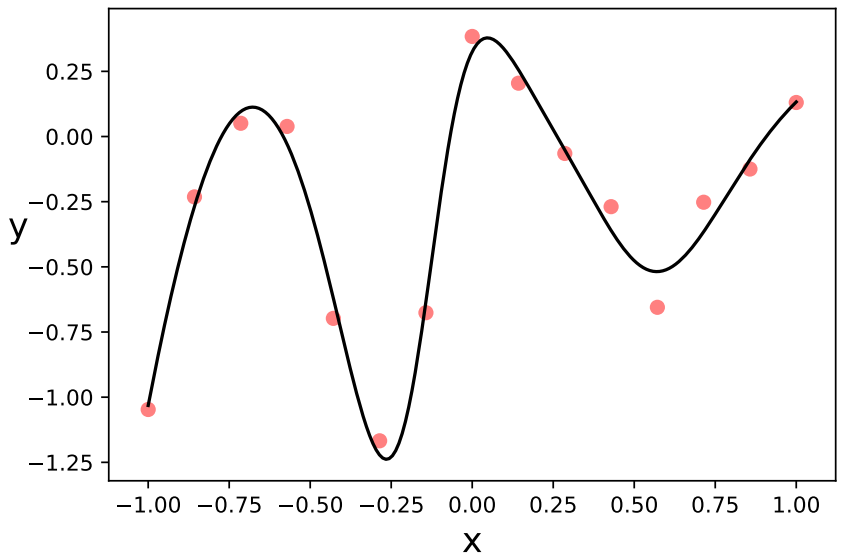
where N is the number of observations.

Sklearn

```
from sklearn.neural_network import MLPRegressor
mlp = MLPRegressor(hidden_layer_sizes=(5,),max_iter=300000,
                                        activation="logistic",solver="
                                        lbfgs")
mlp.fit(x.reshape(-1,1),y)
y_pred = mlp.predict(xtest.reshape(-1,1))
```

In ANN, we need to use a numerical optimizer (called "lbfgs" in the above example) to fit the data.

Solution



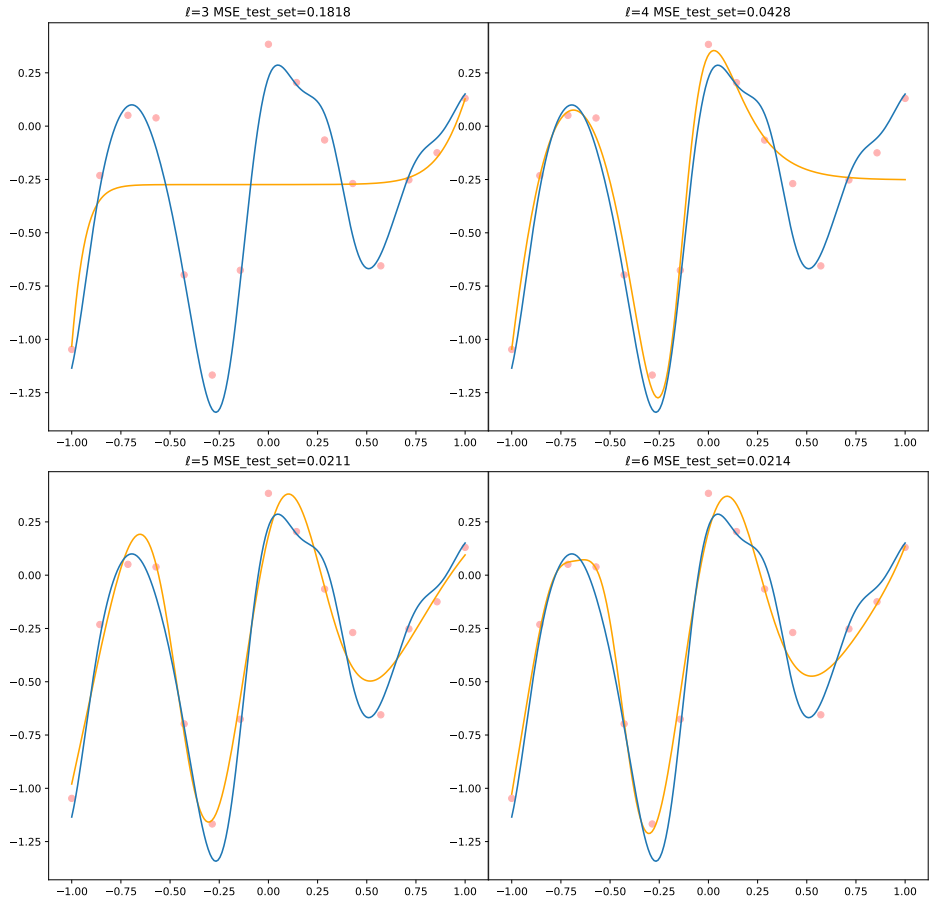
Why

- $l = 5$?

this is a question similar why "degree=5" in polynomial regression.

Increase of hidden nodes leads to overfitting

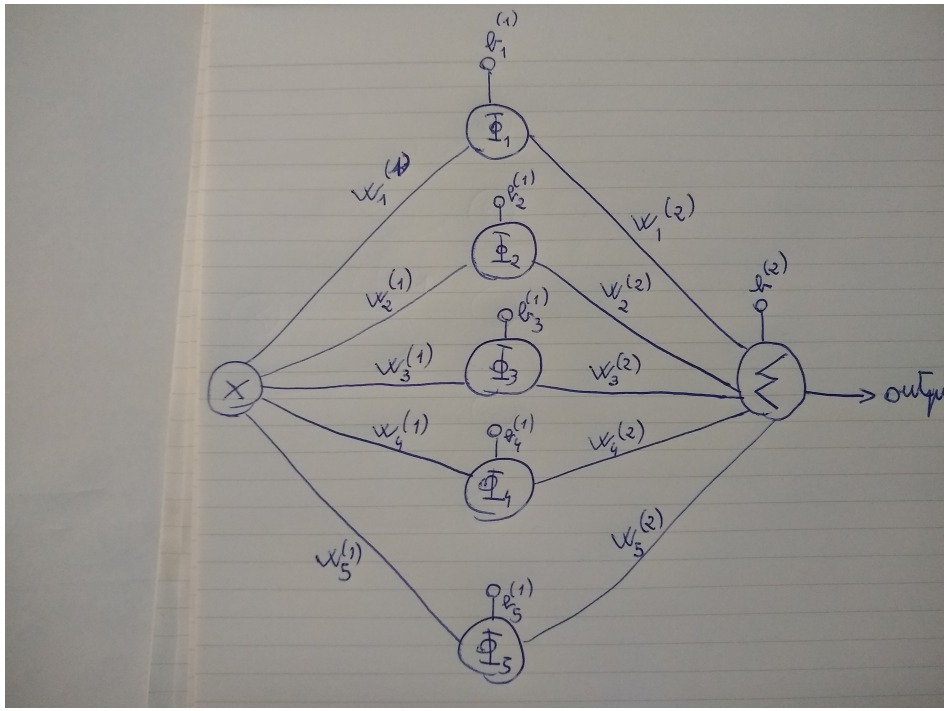
Let's use only 15 observations for training and 200 for testing. We fit 4 different NNs by using $l = 4, 5, 6, 7$:



We can see that the optimal value of l is 5, because it has minimum MSE in the test-set.

A better way to find the best l is to use 10-fold cross-validation.

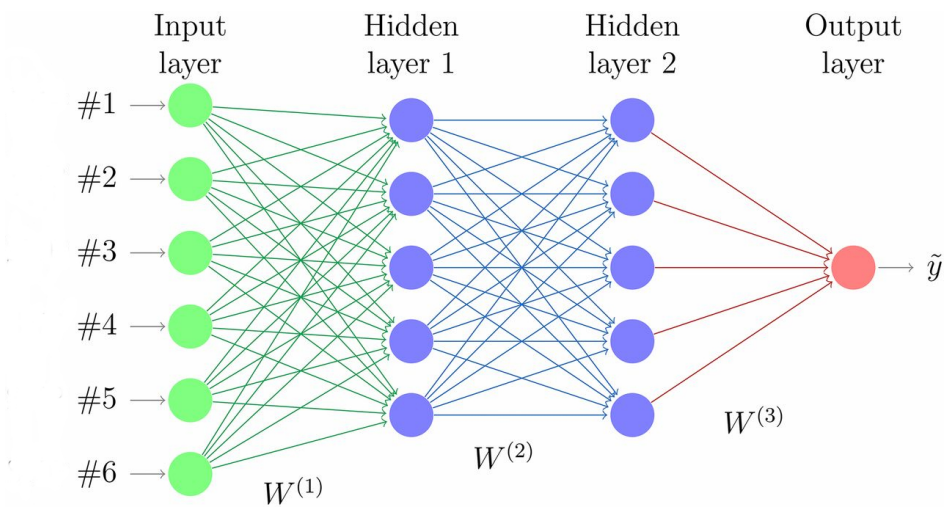
ANN for classification



Yes we can use ANN for classification, it is enough we add another logistic function after the $\sum$, so that we squeeze the output between 0 and 1.

```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(hidden_layer_sizes=(5,),max_iter=500000,
                                    activation="logistic",solver="
                                    lbfgs", random_state=42)
clf.fit(X_train_n, y_train)
y_test_pred = clf.predict(X_test_n)
```

Deep NN for regression/classification




```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(hidden_layer_sizes=(5,5),max_iter=500000,
                                    activation="logistic",solver="
                                    lbfgs", random_state=42)
clf.fit(X_train_n, y_train)
y_test_pred = clf.predict(X_test_n)
```