

COMP 6721 Applied Artificial Intelligence (Winter 2021)

Lab Exercise #06: Artificial Neural Networks

Question 1 Given the training instances below, use `scikit-learn` to implement a perceptron classifier¹ that classifies students in two categories, predicting who will get an ‘A’ this year, based on an input feature vector \vec{x} . Here’s the training data again:

Student	Feature(x)				Output f(x)
	‘A’ last year?	Black hair?	Works hard?	Drinks?	‘A’ this year?
X1: Richard	Yes	Yes	No	Yes	No
X2: Alan	Yes	Yes	Yes	No	Yes
X3: Alison	No	No	Yes	No	No
X4: Jeff	No	Yes	No	Yes	No
X5: Gail	Yes	No	Yes	Yes	Yes
X6: Simon	No	Yes	Yes	Yes	No

Use the following Python imports for the perceptron:

```
import numpy as np
from sklearn.linear_model import Perceptron
```

All features must be numerical for training the classifier, so you have to transform the ‘Yes’ and ‘No’ feature values to their binary representation:

```
dataset = np.array([[1,1,0,1,0],
                    [1,1,1,0,1],
                    [0,0,1,0,0],
                    [0,1,0,1,0],
                    [1,0,1,1,1],
                    [0,1,1,1,0],])
```

For our feature vectors, we need the first four columns:

```
X = dataset[:, 0:4]
```

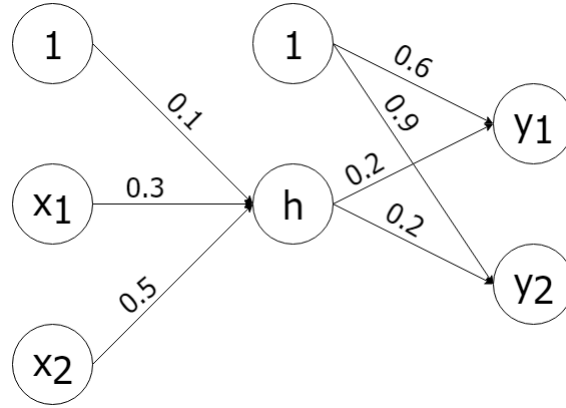
and for the training labels, we use the last column from the dataset:

```
y = dataset[:, 4]
```

¹https://scikit-learn.org/stable/modules/linear_model.html#perceptron

- (a) Now, create a perceptron classifier (same approach as in the previous labs) and train it.
- (b) Apply the trained model to all training samples and print out the prediction.

Question 2 Consider the neural network shown below. It consists of 2 input nodes, 1 hidden node, and 2 output nodes, with an additional bias at the input layer and a bias at the hidden layer. All nodes in the hidden and output layers use the sigmoid activation function (σ).



- Calculate the output of y_1 and y_2 if the network is fed $\vec{x} = (1, 0)$ as input.
- Assume that the expected output for the input $\vec{x} = (1, 0)$ is supposed to be $\vec{t} = (0, 1)$. Calculate the updated weights after the backpropagation of the error for this sample. Assume that the learning rate $\eta = 0.1$.

Question 3 Let's see how we can build multi-layer neural networks using `scikit-learn`.²

- (a) Implement the architecture from the previous question using `scikit-learn` and use it to learn the XOR function, which is not linearly separable.

Use the following Python imports:

```
import numpy as np
from sklearn.neural_network import MLPClassifier
```

Here is the training data for XOR function

```
dataset = np.array([[1,1,0],
                    [0,1,1],
                    [1,0,1],
                    [0,0,0]])
```

For our feature vectors, we need the first two columns:

```
X = dataset[:, 0:2]
```

and for the training labels, we use the last column from the dataset:

```
y = dataset[:, 2]
```

Now you can create a Multi-layer Perceptron using `scikit-learn`³'s MLP classifier. There are a lot of parameters you can choose to define and customize, here you need to define the `hidden_layer_sizes`. For this parameter, you pass in a tuple consisting of the number of neurons you want at each layer, where the n th entry in the tuple represents the number of neurons in the n th layer of the MLP model. You also need to set the activation to 'logistic', which is the logistic Sigmoid function. The bias and weight details are implicitly defined in the function definition.

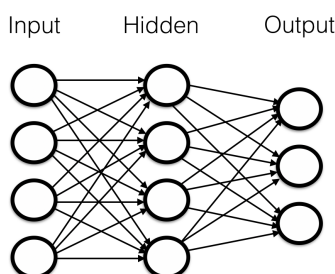
- (b) Now apply the trained model to all training samples and print out its prediction.

As you see, our single hidden layer with a single neuron doesn't perform well on learning XOR. It's always a good idea to experiment with different network configurations.

²https://scikit-learn.org/stable/modules/neural_networks_supervised.html

³https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

Question 4 Consider the neural network architecture below and use it to classify Iris flower dataset.



- (a) Load Iris from `scikit-learn`⁴'s builtin datasets. Like before, use `scikit-learn`⁵'s `train_test_split` helper function to split the Iris dataset into a training and testing subset and train the model.
- (b) Now run an evaluation to compute your model's accuracy using `scikit-learn`'s⁶ `accuracy_score`.
- (c) In binary classification, we score the model intuitively using precision and recall metrics. But for multi-class classification, it's different. For this case, the `scikit-learn` package provides the implementation of precision and recall scores based on *macro* and *micro* averaging: 'micro' calculate metrics globally by counting the total true positives, false negatives, and false positives. The 'macro' version calculates metrics for each label and find their unweighted mean.

Run an evaluation on your results and compute the precision and recall score with micro and macro averaging, using `scikit-learn`'s `precision_score`⁷ and `recall_score`.⁸

⁴https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html

⁵https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

⁶https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

⁷https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html

⁸https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html