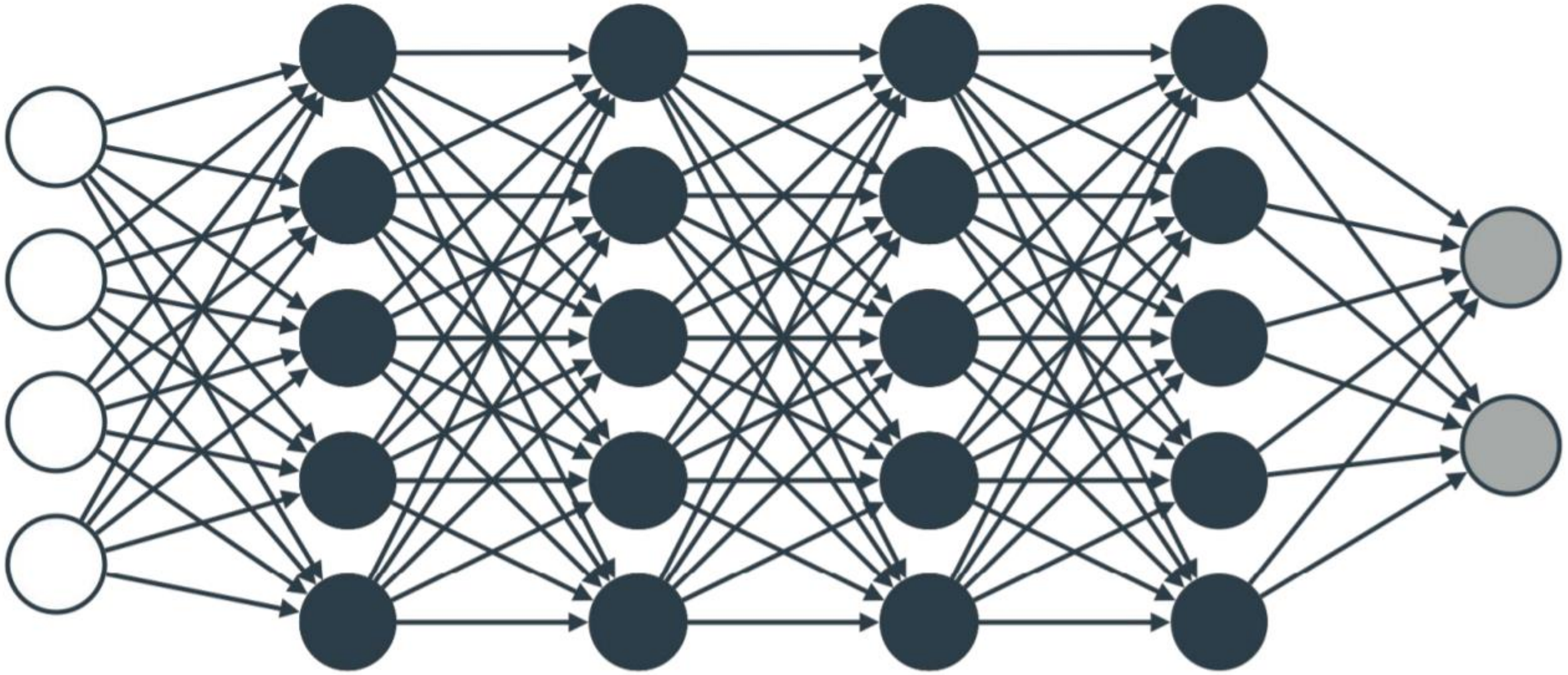
Combining building blocks to gain more power: Neural Networks

Neural Networks

This lesson covers

- What is a neural network?
- What is a perceptron?
- Using neural networks in a simple application: sentiment analysis.
- Training neural networks using backpropagation.
- Potential problems in training neural networks, and techniques that can be used to avoid these problems.
- How to code the linear regression algorithm in Keras.

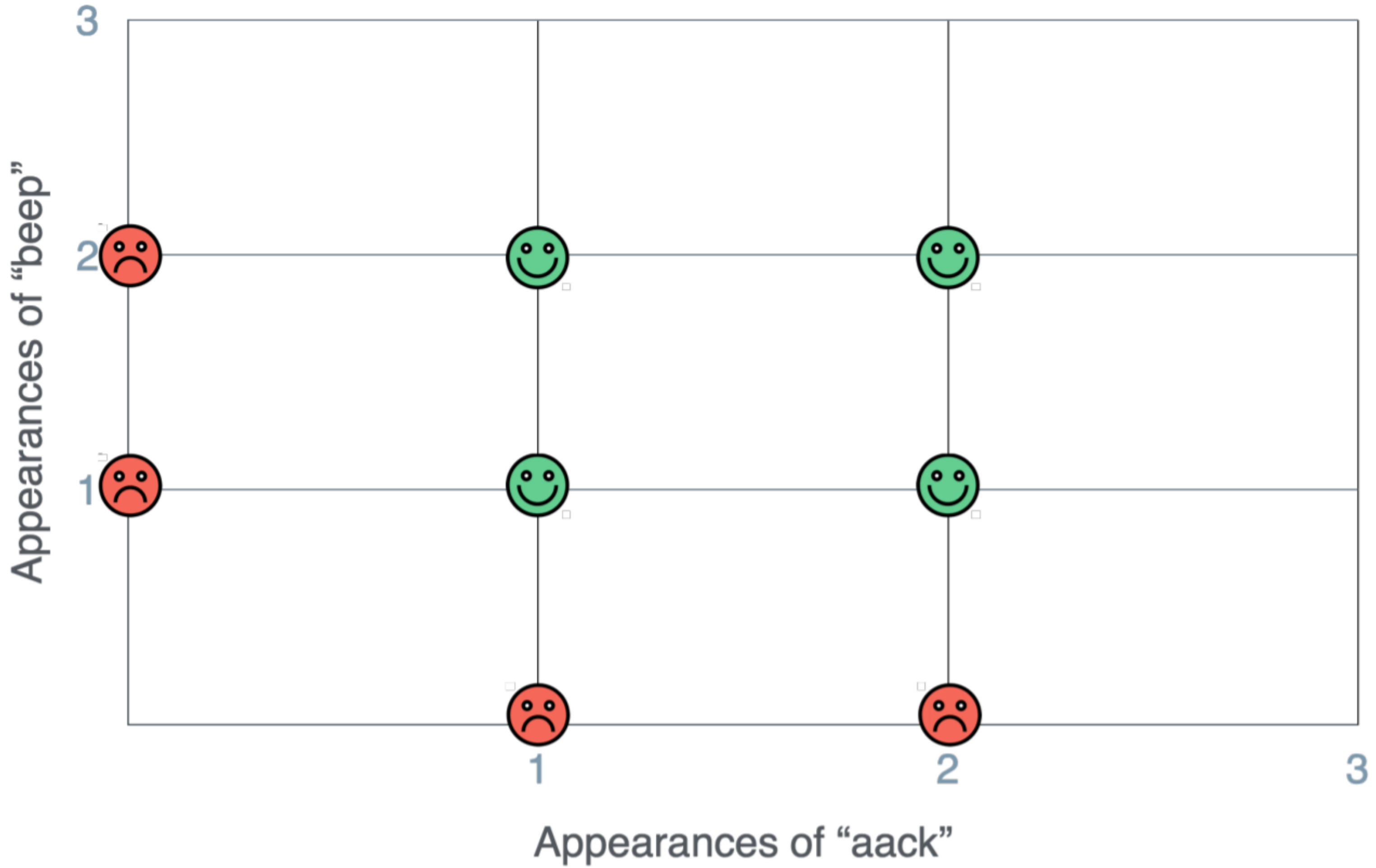
Neural Networks



The problem - A more complicated alien planet!

- In this lesson we will continue with the example from lessons 4 and 5, when we learned the perceptron algorithm and logistic regression.
- The scenario was the following: We find ourselves in a distant planet populated by aliens.
- They seem to speak a language formed by two words, ‘aack’ and ‘beep’, and we want to build a machine learning model that helps us determine if an alien is happy or sad based on the words they say.

Sentence	Aack	Beep	Mood
“Aack”	1	0	Sad
“Aack aack”	2	0	Sad
“Beep”	0	1	Sad
“Beep beep”	0	2	Sad
“Aack beep”	1	1	Happy
“Aack aack beep”	1	2	Happy
“Beep aack beep”	2	1	Happy
“Beep aack beep aack”	2	2	Happy


3
2
1
Appearances of "beep"
1
2
3
Appearances of "aack"

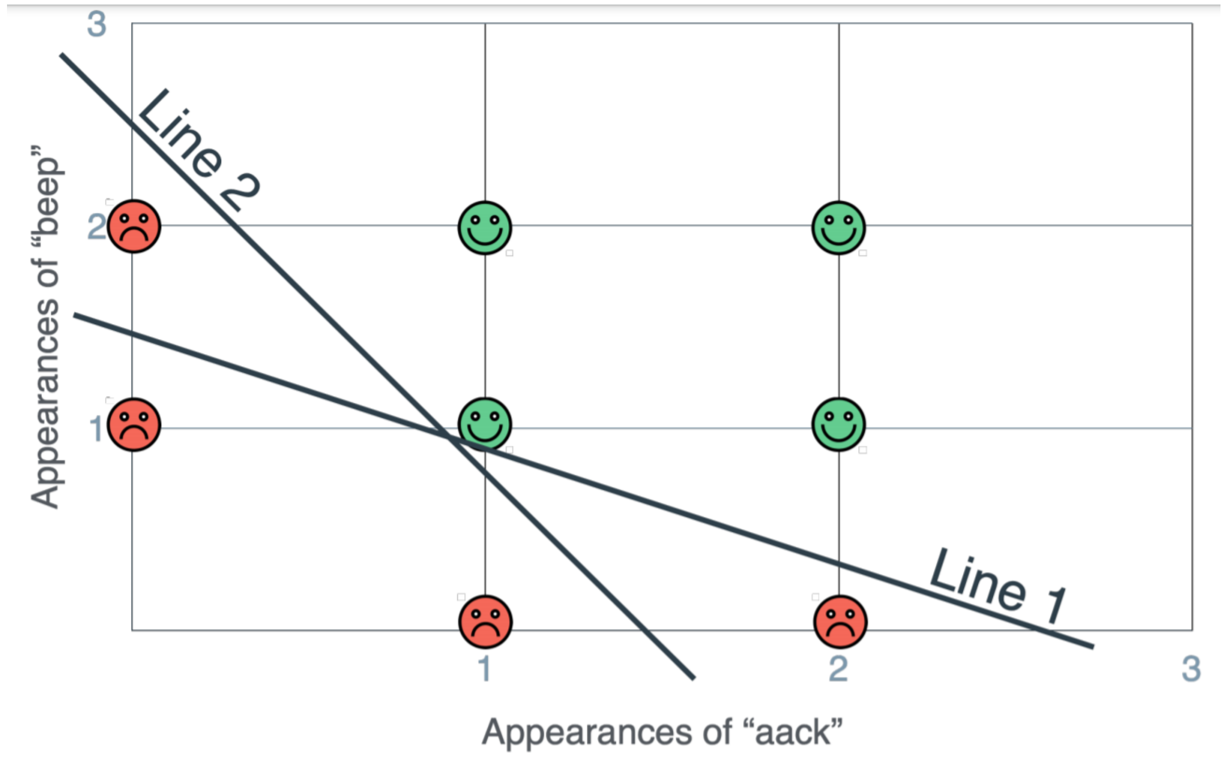
The problem - A more complicated alien planet!

If our goal is to separate the points in previous Figure, clearly one line won’t do it. What is better than one line? I can think of two things:

1. Two lines.

2. A curve.

Solution - If one line is not enough, use two lines to classify your dataset



Solution - If one line is not enough, use two lines to classify your dataset

Now, let’s throw in some math. Can you help me think of two equations for these lines?
Many equations would work, but I’ve thought about the following two (where #aack is the number of times the word ‘aack’ appears in the sentence, and #beep is the number of time ‘beep’ appears.):

- Line 1: 6*(#aack) + 10*(#beep) = 15
- Line 2: 10*(#aack) + 6*(#beep) = 15

Solution - If one line is not enough, use two lines to classify your dataset

Classifier: A sentence is classified as happy if both of the following two inequalities hold.

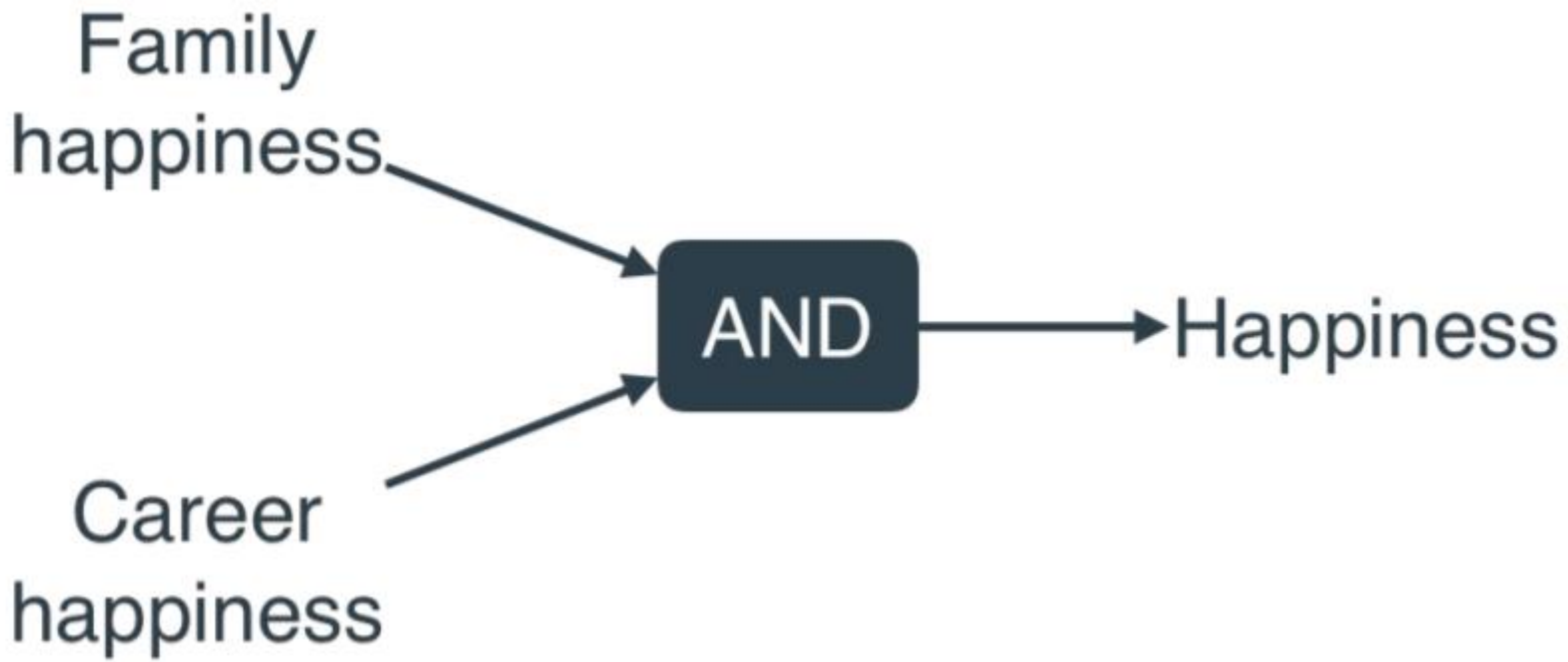
- Inequality 1: $6*(\#aack) + 10*(\#beep) \geq 15$
- Inequality 2: $10*(\#aack) + 6*(\#beep) \geq 15$

Sentence	Aack	Beep	Equation 1	*Eq 1 ≥ 15?*	Equation 2	*Eq 2 ≥ 15?*	*Both eqs. ≥ 15*
"Aack"	1	0	6	no	10	no	no
"Aack aack"	2	0	12	no	20	yes	no
"Beep"	0	1	10	no	6	no	no
"Beep beep"	0	2	20	yes	12	no	no
"Aack beep"	1	1	16	yes	16	yes	yes
"Aack aack beep"	1	2	26	yes	22	yes	yes
"Beep aack beep"	2	1	22	yes	26	yes	yes
"Beep aack beep aack"	2	2	32	yes	32	yes	yes

Why two lines? Is happiness not linear?

- In lessons 4 and 5 we managed to infer things about the language based on the equations in the classifiers.
- For example, if the weight of the word ‘aack’ was positive, we concluded that it was likely a happy word. What about now?
- Could we infer anything about the language in this classifier that contains two equations?

Why two lines? Is happiness not linear?



Perceptrons and how to combine them

- In this lesson we have seen two classifiers that are very similar to those in lesson 4: The family happiness and the career happiness classifier.

- Let’s study them more carefully. We’ll start with the family happiness classifier.

- It was given by Line 1, which had the equation

$$6*(\#aack) + 10*(\#beep) \geq 15$$

Perceptrons and how to combine them

Classifier 1 (family happiness)

Scores:

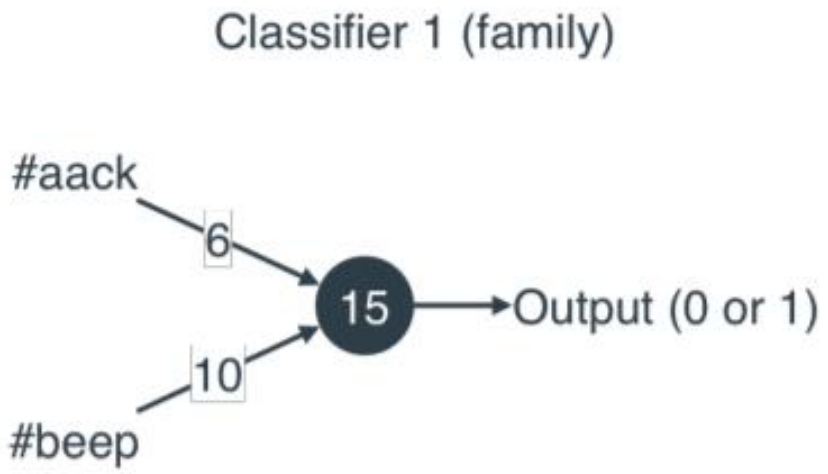
- Aack: 6 points
- Beep: 10 points
- Threshold: 15

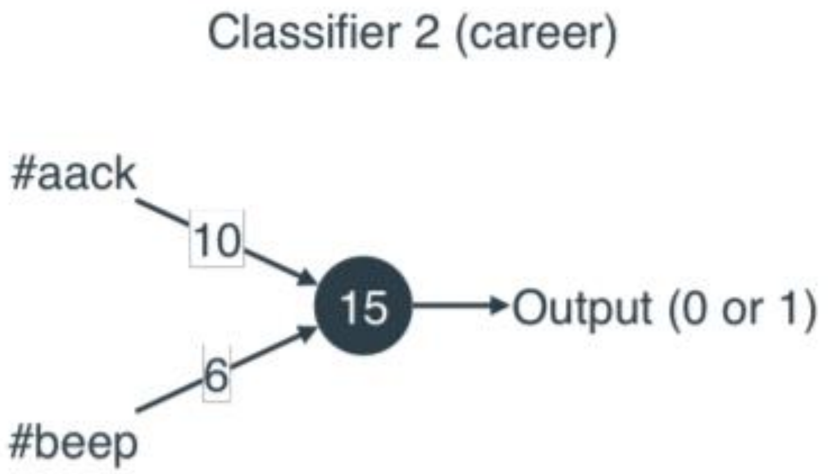
Rule:

Add the scores of all the words.

- If the score is larger than or equal to the threshold of 15, predict that the alien has a happy family life.
- If the score is smaller than the threshold of 15, predict that the alien doesn't have a happy family life.

Perceptrons and how to combine them





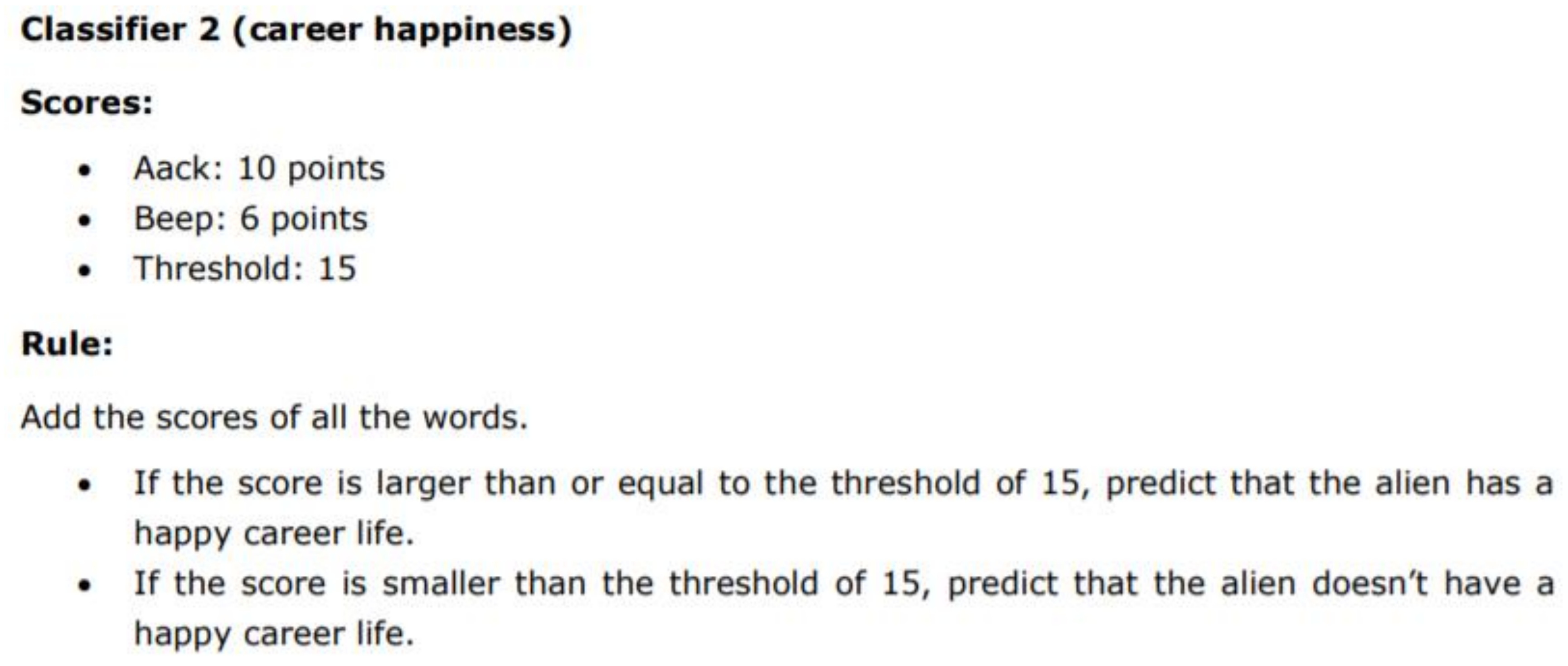
Perceptrons and how to combine them

- In lesson 5 we learned activation functions, in particular the step function, which returns a 1 if the input is positive or zero, and a 0 if the input is negative.

- In this case, we can write the output of the classifier as the following:

Output = Step(6*(#aack) + 10*(#beep) - 15)

Perceptrons and how to combine them



Classifier 2 (career happiness)

Scores:

- Aack: 10 points
- Beep: 6 points
- Threshold: 15

Rule:

Add the scores of all the words.

- If the score is larger than or equal to the threshold of 15, predict that the alien has a happy career life.
- If the score is smaller than the threshold of 15, predict that the alien doesn't have a happy career life.

Perceptrons and how to combine them

- The diagram of this perceptron is in the right of figure 8.x, and its output is given by the following formula:

Output = Step(10*(#aack) + 6*(#beep) - 15)

Sentence	Aack	Beep	Family Happiness	Career Happiness
“Aack”	1	0	0	0
“Aack aack”	2	0	0	1
“Beep”	0	1	0	0
“Beep beep”	0	2	1	0
“Aack beep”	1	1	1	1
“Aack aack beep”	1	2	1	1
“Beep aack beep”	2	1	1	1
“Beep aack beep aack”	2	2	1	

Family Happiness (feature)	Career Happiness (feature)	Happiness (label)
0	0	0
0	1	0
0	0	0
1	0	0
1	1	1
1	1	1
1	1	1
1	1	1

Classifier 3 (happiness)

Scores:

- Family happiness: 1 point
- Career happiness: 1 point
- Threshold: 1.5

Rule:

Add the scores of all the words.

- If the score is larger than or equal to the threshold of 1.5, predict that the alien has a happy career life.
- If the score is smaller than the threshold of 1.5, predict that the alien doesn't have a happy career life.

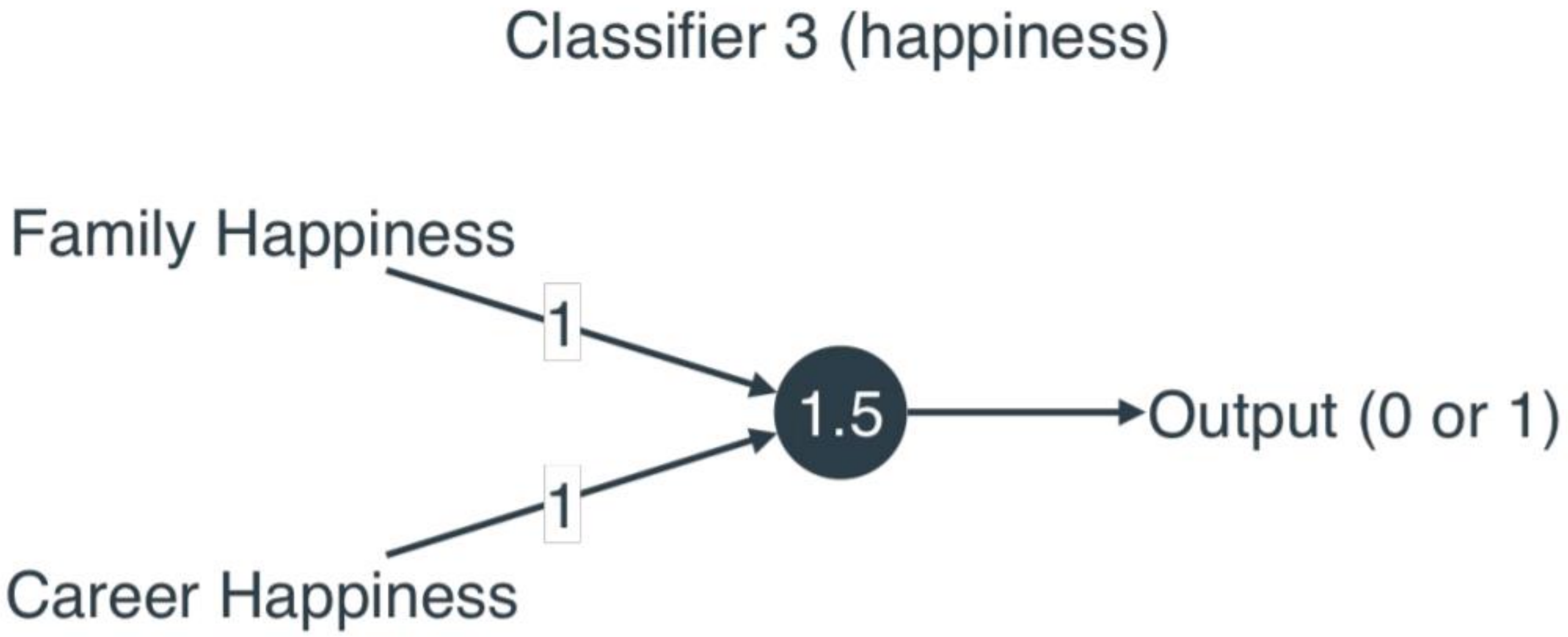
Perceptrons and how to combine them

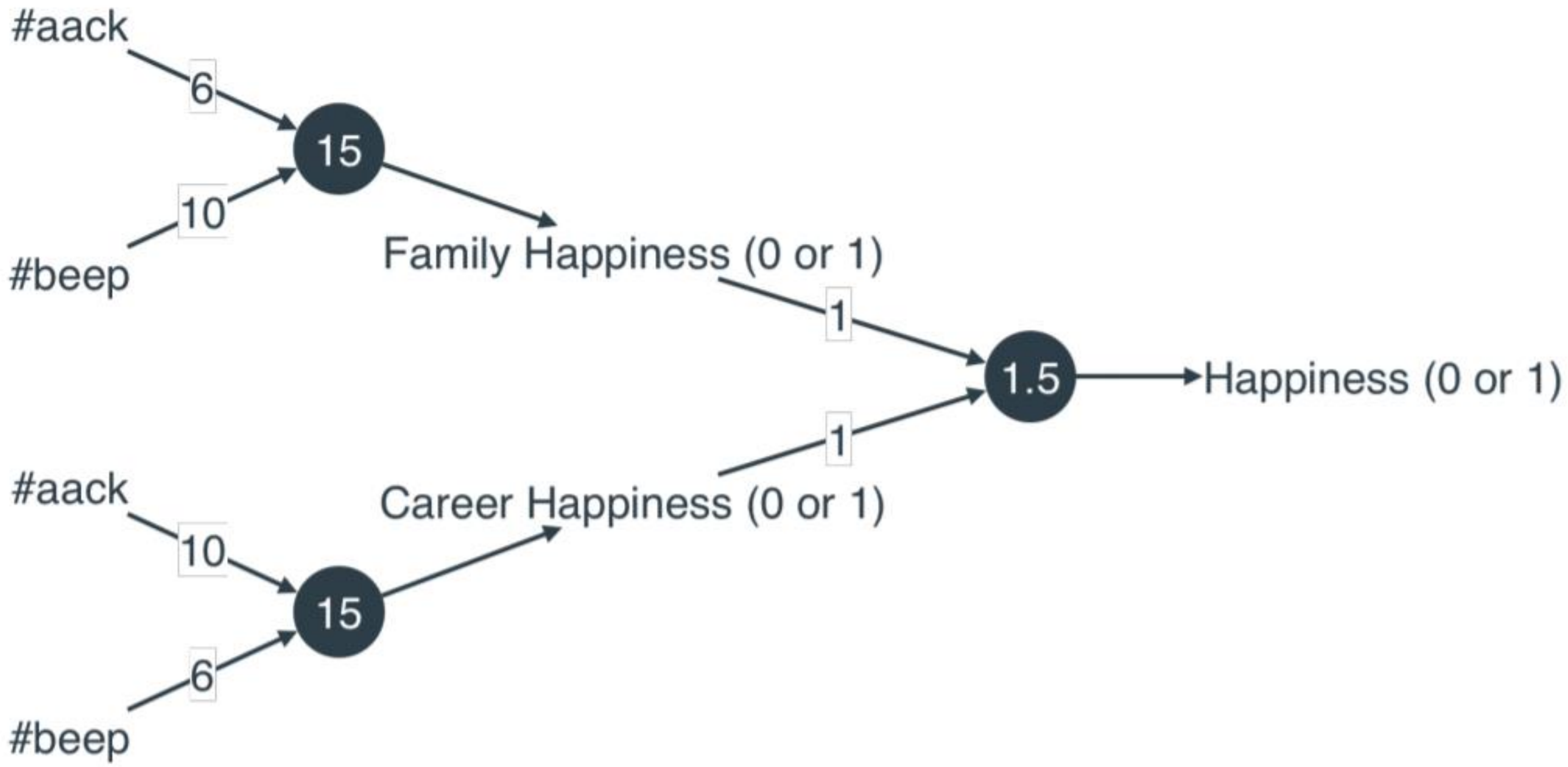
- The formula for the output of this classifier is given by

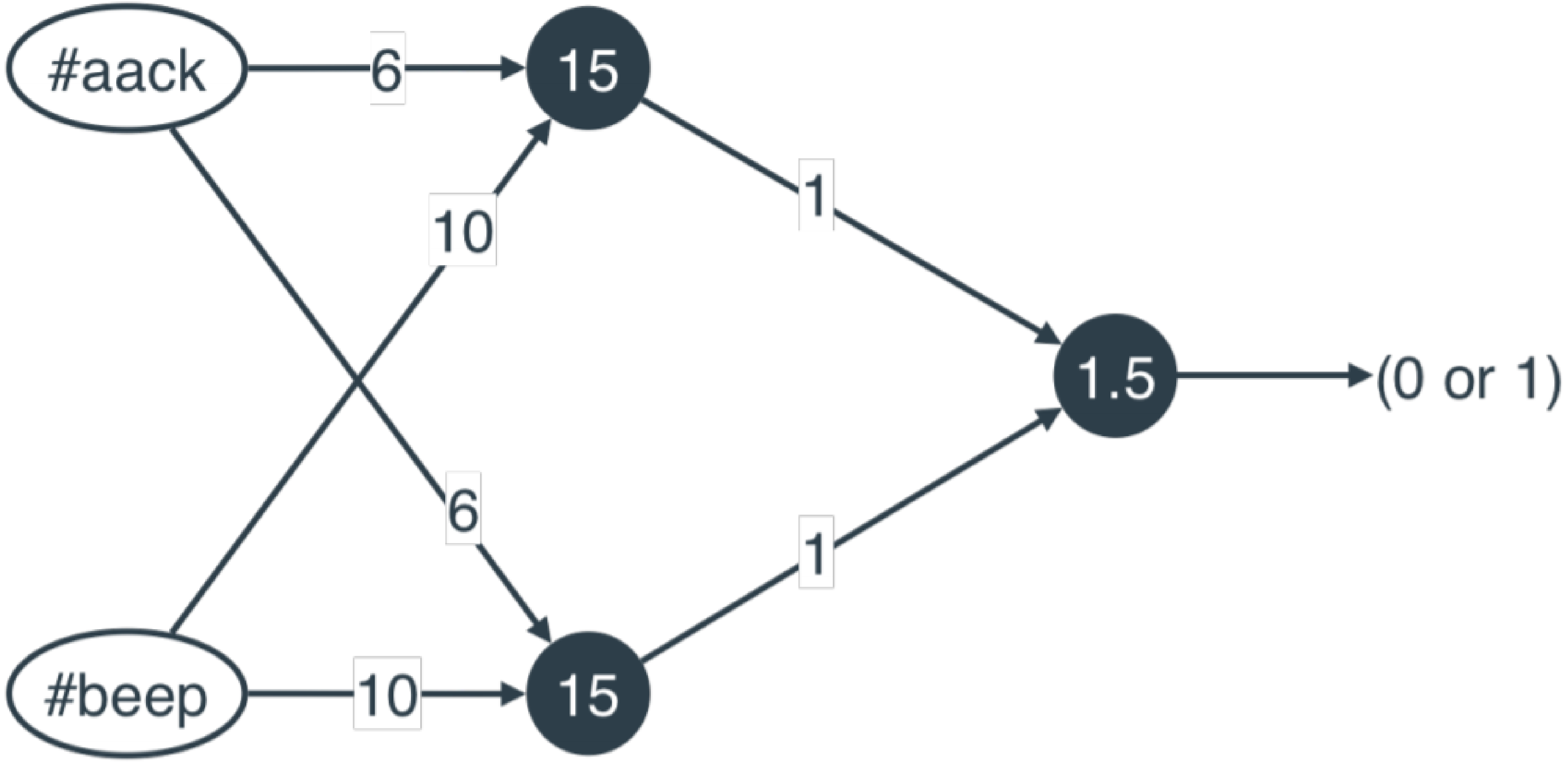
Output = Step(1*(Family happiness) + 1*(Career happiness) - 1.5)

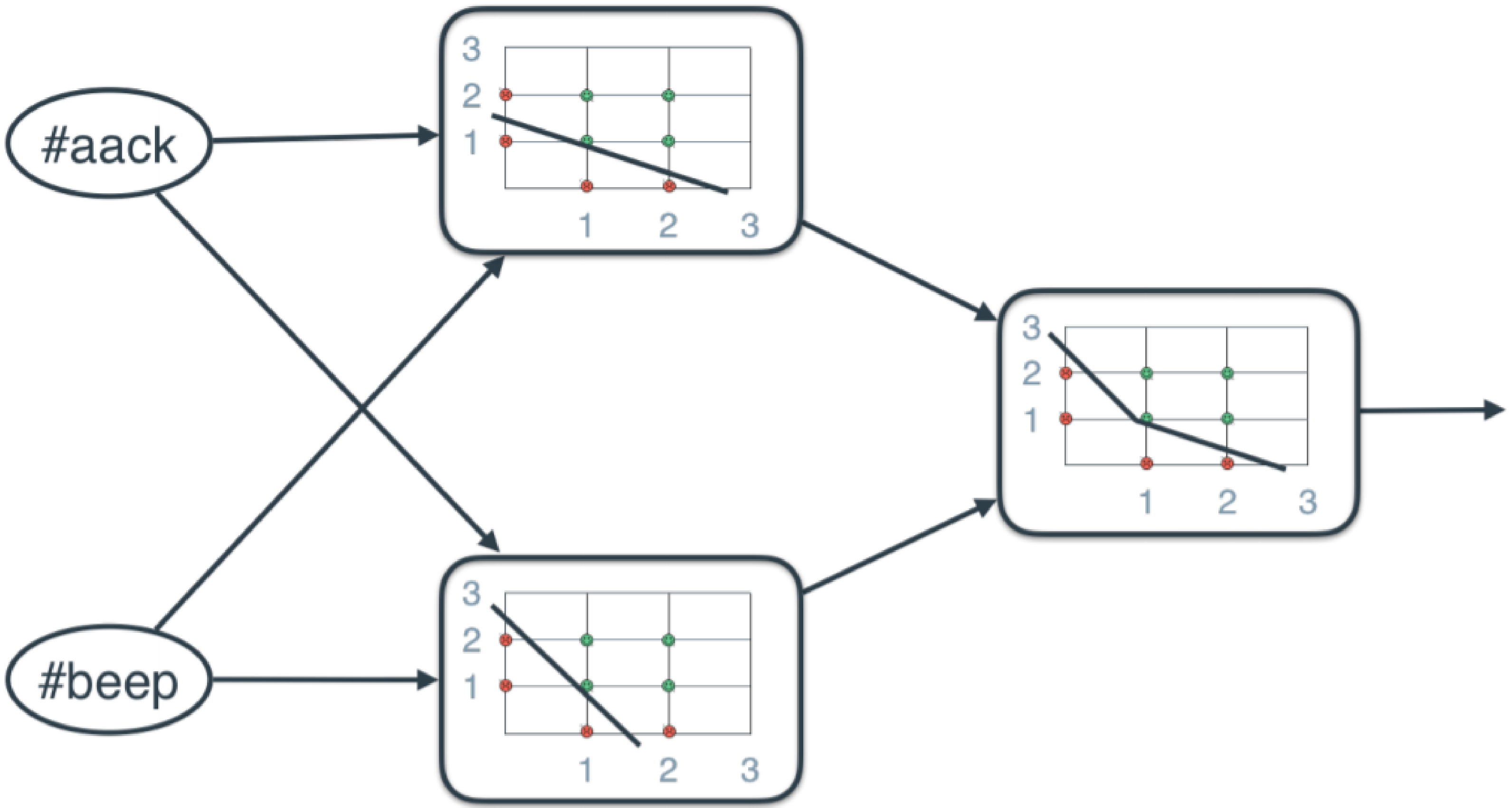
- Where (Family happiness) and (Career happiness) are the outputs of classifiers 1 and 2, respectively

Perceptrons and how to combine them




#aack
6
15
10
#beep
Family Happiness (0 or 1)
1
1.5
Happiness (0 or 1)
1
#aack
Career Happiness (0 or 1)
10
15
6
#beep


#aack
#beep
6
10
6
10
15
15
1
1
1.5
(0 or 1)


#aack
#beep
3
2
1
1
2
3

A trick to improve our training

- In this section, we’ll develop continuous perceptrons, which use the sigmoid function to output any number between 0 and 1.
- The development is the same, except the equation that defines the output is now the following (for classifier 1, the family happiness classifier):

Output = $\sigma\sigma$(6 × (#$aaaaaaaa$) + 10 × (#$bbbbbbbb$) − 15).

A trick to improve our training

- Let's look at a simple calculation as an example. We'll calculate the output for the sentence "Aack beep", where #aack = 1 and #beep = 1.

- This output is

$$\text{Output} = \sigma\sigma(6 \times (1) + 10 \times (1) - 15) = \sigma\sigma(1) = 0.73.$$

A trick to improve our training

- Similarly, the equation for classifier 2, the career happiness classifier, is

  Output = $\sigma\sigma$(10 × (#$aaaaaaaa$) + 6 × (#$bbbbbbbb$) − 15),

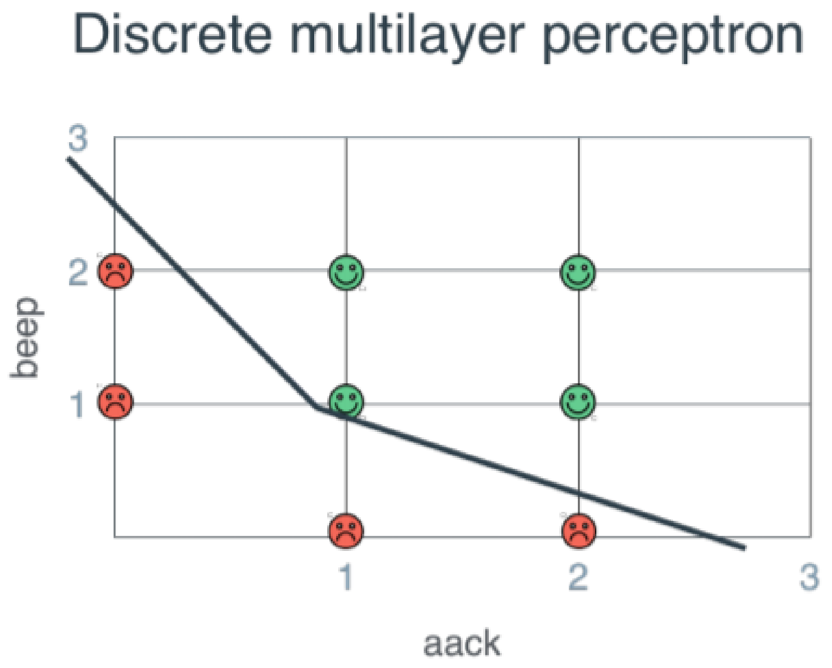
- and that of classifier 3, the happiness classifier, is

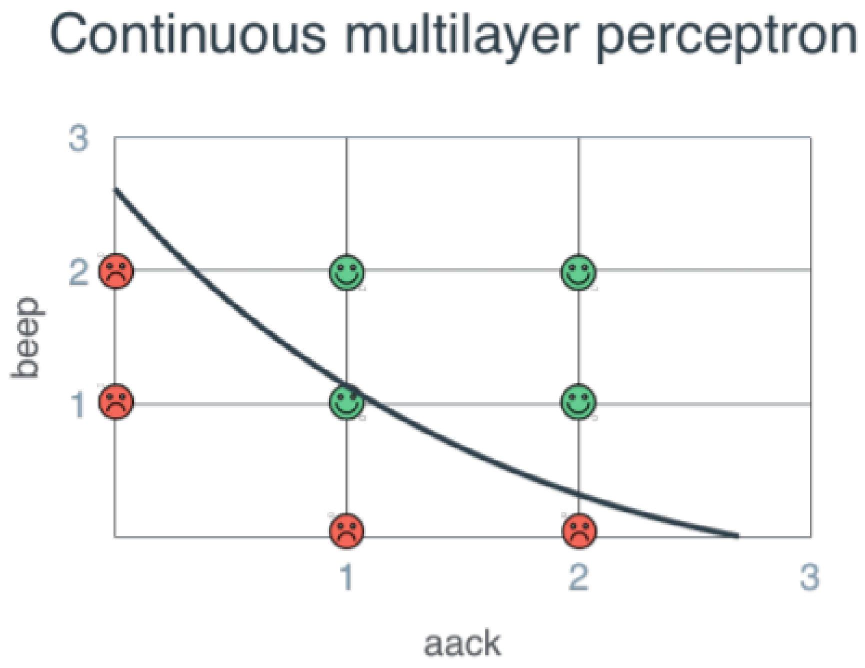
  Output = $\sigma\sigma$(1 × ($CCCCCCCCCCCC$) + 1 × (#$bbbbbbbb$) − 1.5),

- where Career and Family are the outputs of classifiers 1 and 2, respectively.

Sentence	Aack	Beep	Career Happiness	Family Happiness	Happiness
“Aack”	1	0	0.000	0.007	0.183
“Aack aack”	2	0	0.047	0.993	0.387
“Beep”	0	1	0.007	0.000	0.183
“Beep beep”	0	2	0.993	0.047	0.387
“Aack beep”	1	1	0.731	0.731	0.491
“Aack aack beep”	1	2	1.000	0.999	0.622
“Beep aack beep”	2	1	0.999	1.000	0.622
“Beep aack beep aack”	2	2	1.000	1.000	0.622

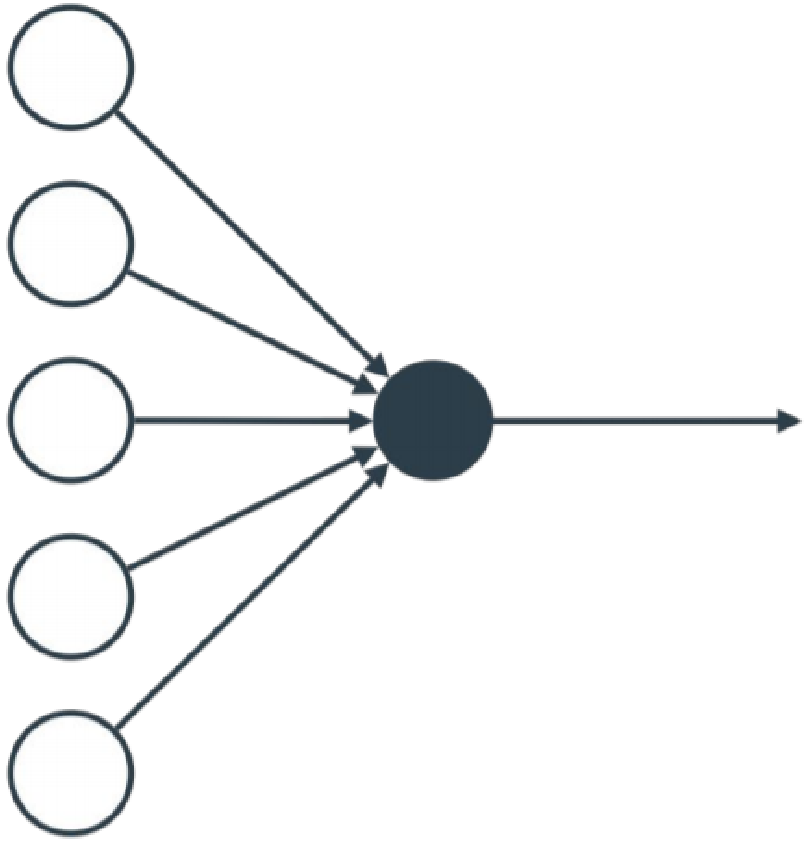
A trick to improve our training





The general scenario - Neural networks

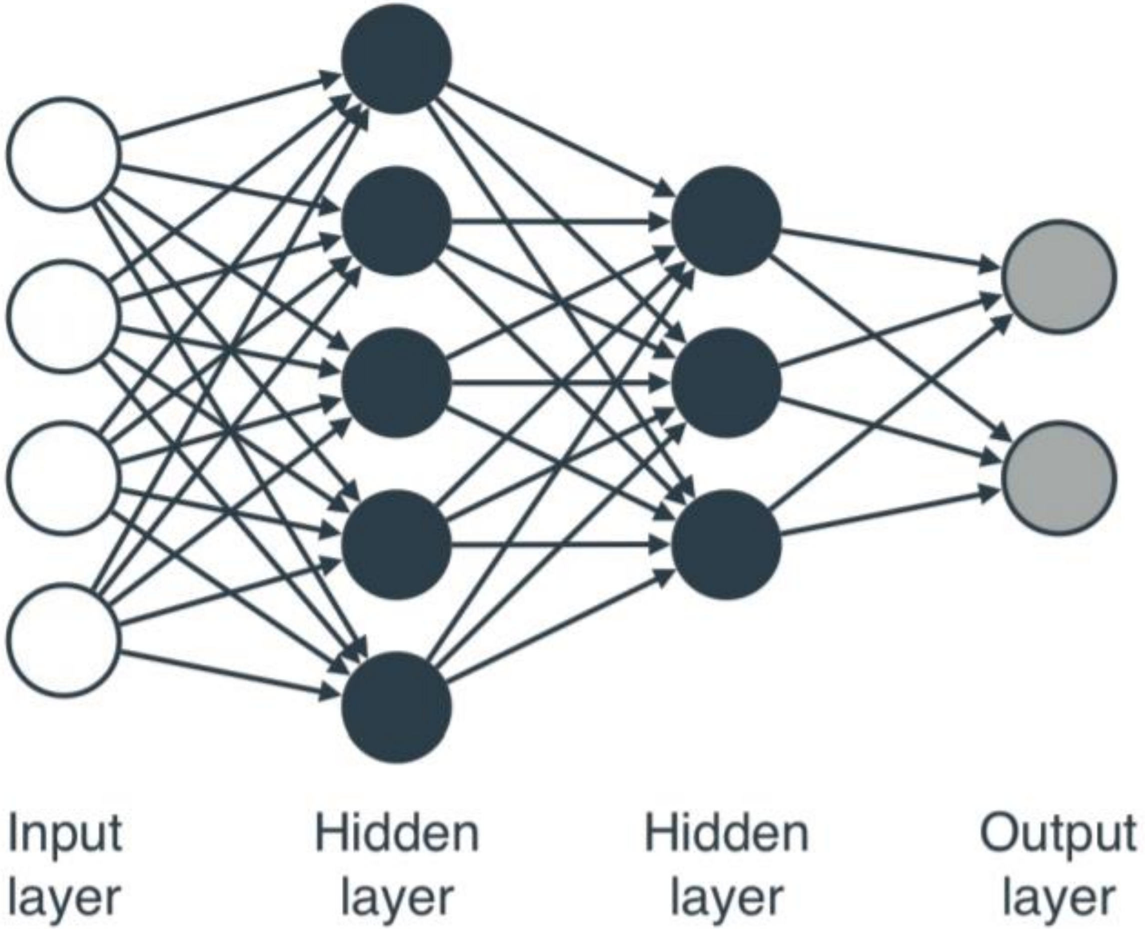
- First things first. Why are these objects called neural networks?
- The idea is that in a vague way, neural networks resemble how the brain works. If we look at a perceptron, it resembles a neuron.
- A general perceptron will have several inputs and one output, and the output is 0 or 1 depending on what the inputs are.



The architecture of a neural network

In real life, neural networks tend to be much larger, and the way we build them is by adding many more hidden layers. Previous figure shows a larger neural network with two hidden layers. The size of a hidden layer is simply the number of nodes in it. The architecture of this neural network is as follows:

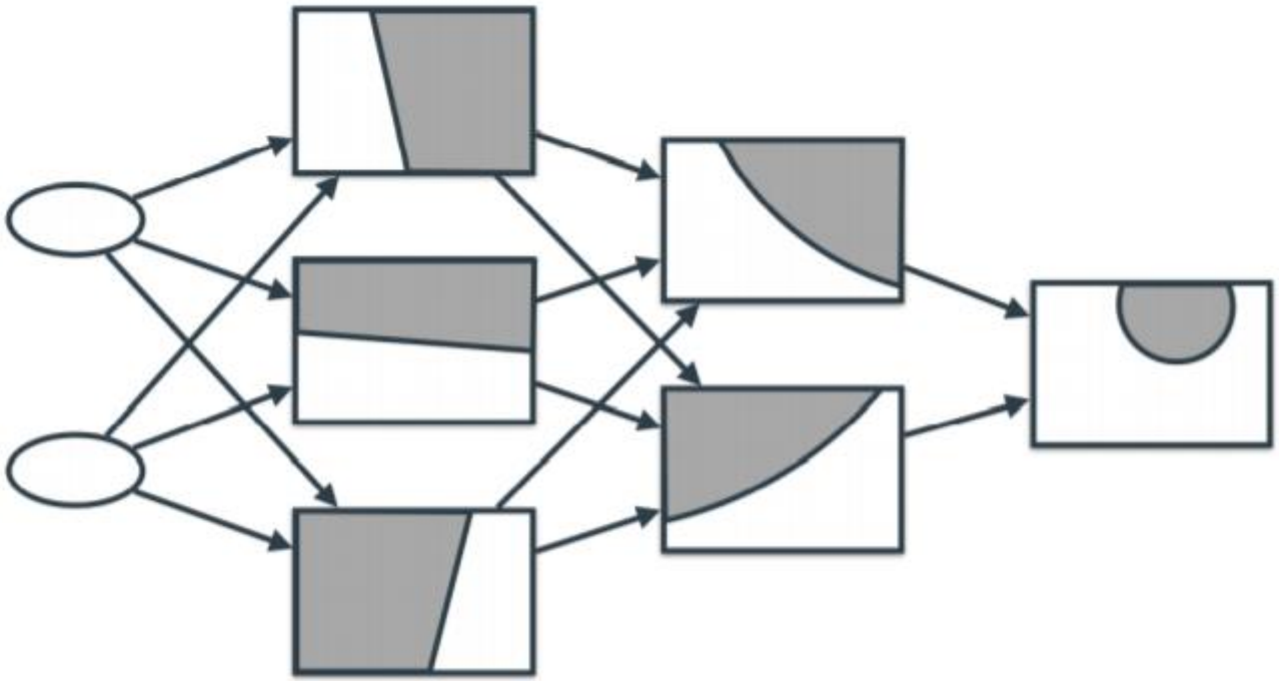
- An input layer of size 4.
- A hidden layer of size 5.
- A hidden layer of size 3.
- An output layer of size 2.


Input layer
Hidden layer
Hidden layer
Output layer

The architecture of a neural network

- What does it mean to have a hidden layer of size 2 or more? This is a great property of neural networks.
- We are used to classifiers that return one output, or answer, such as a ‘yes’ or a ‘no’.
- However, in neural networks we can output as many answers as we want.
- For example, if we want to train a neural network that will recognize images and tell us if the image contains a dog, cat, bird, or aardvark, we simply make sure the output layer has 4 nodes, one for each animal

The architecture of a neural network

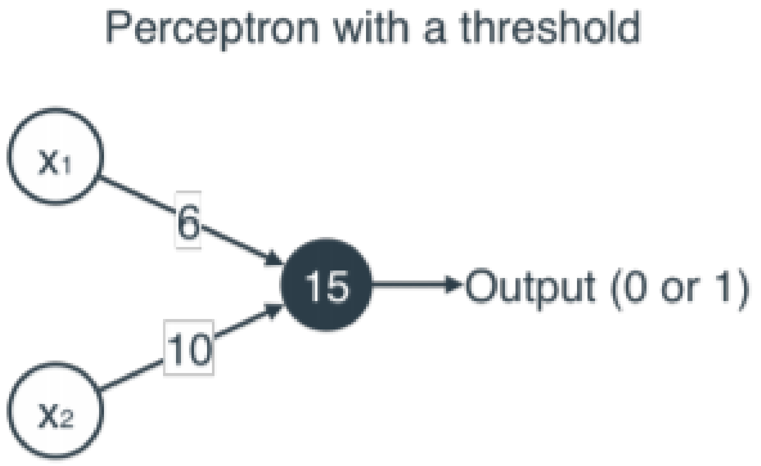


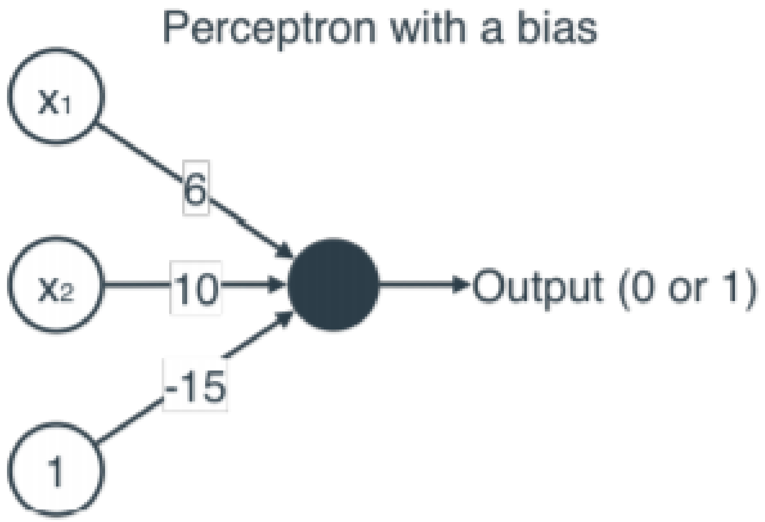
Bias vs Threshold

- Let’s say we want the threshold to always be zero, by convention.
- In that case, we need to subtract 15 from the score, and compare this score with a zero, in order to get the same effect.
- We call this -15 the bias. Now, the score is 6a + 10b - 15.
- We can write the classifier as follows:

Score of ‘aack’: 6
Score of ‘beep’: 10
Bias: 15

Bias vs Threshold





Training neural networks

- In this section we’ll take a closer look at how to train neural networks.

- The training is not very different from other algorithms such as linear or logistic regression: First we define an error function, then we start with random weights, and finally we iterate over these weights many times in order to reduce the error function

Training neural networks

Error function - A way to measure how our neural network is performing

- The training process in most machine learning models makes use of an error function which constantly informs us how the model is doing at every step this is the case for neural networks as well.
- The error function we use here is actually very similar from the one we used for logistic regression in lesson 5; in fact, it is the same log loss function

Training neural networks

function is defined as follows:

- If the label is 0:
  - log loss = -ln(1 - prediction)
- If the label is 1:
  - log loss = -ln(prediction).

These can be summarized into one formula as:

log loss = (-label) ln(prediction) - (1-label) ln(1 - prediction)

- **Example 1**: Label close to the prediction, so the error is small.

Label = 0

Prediction = 0.1

log loss = -ln(1-0.1) = -ln(0.9) = -0.105

- **Example 2:** Label far from the prediction, so the error is large.

Label = 0

Prediction = 0.9

log loss = -ln(1-0.9) = -ln(0.1) = -2.303

- **Example 3:** Label far from the prediction, so the error is large.

Label = 1

Prediction = 0.1

log loss = -ln(0.1) = -2.303

- **Example 4:** Label close to the prediction, so the error is small.

Label = 1

Prediction = 0.9

log loss = -ln(0.9) = -0.105

Training neural networks

Gradient descent:

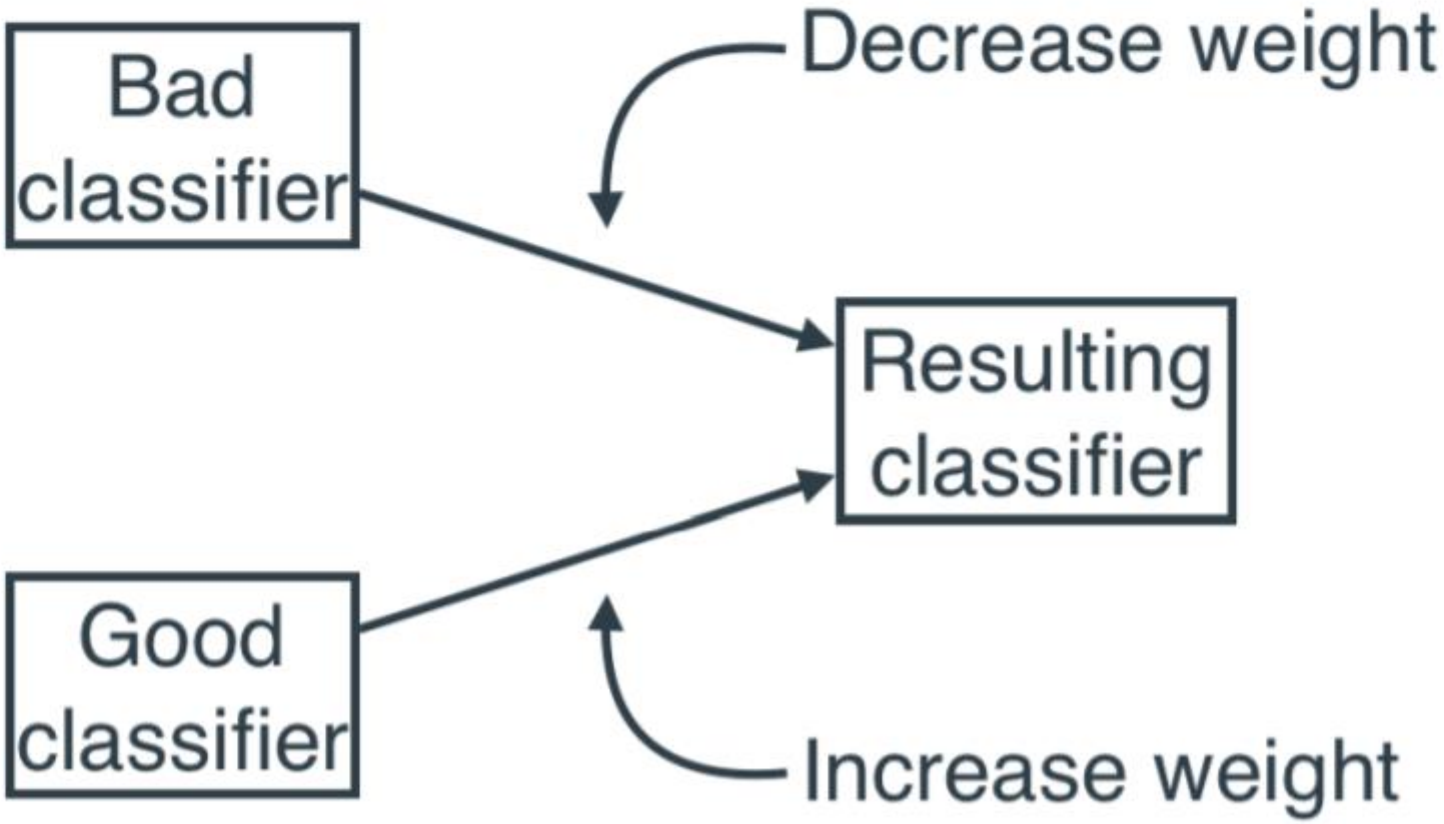
1. Start with random weights.
2. Repeat many times:

a) Calculate the loss function.
b) Take a small step in the direction of the gradient in order to decrease the loss function by a small amount.

3. The weights you obtain correspond to a neural network that (hopefully) fits the data well

Training neural networks

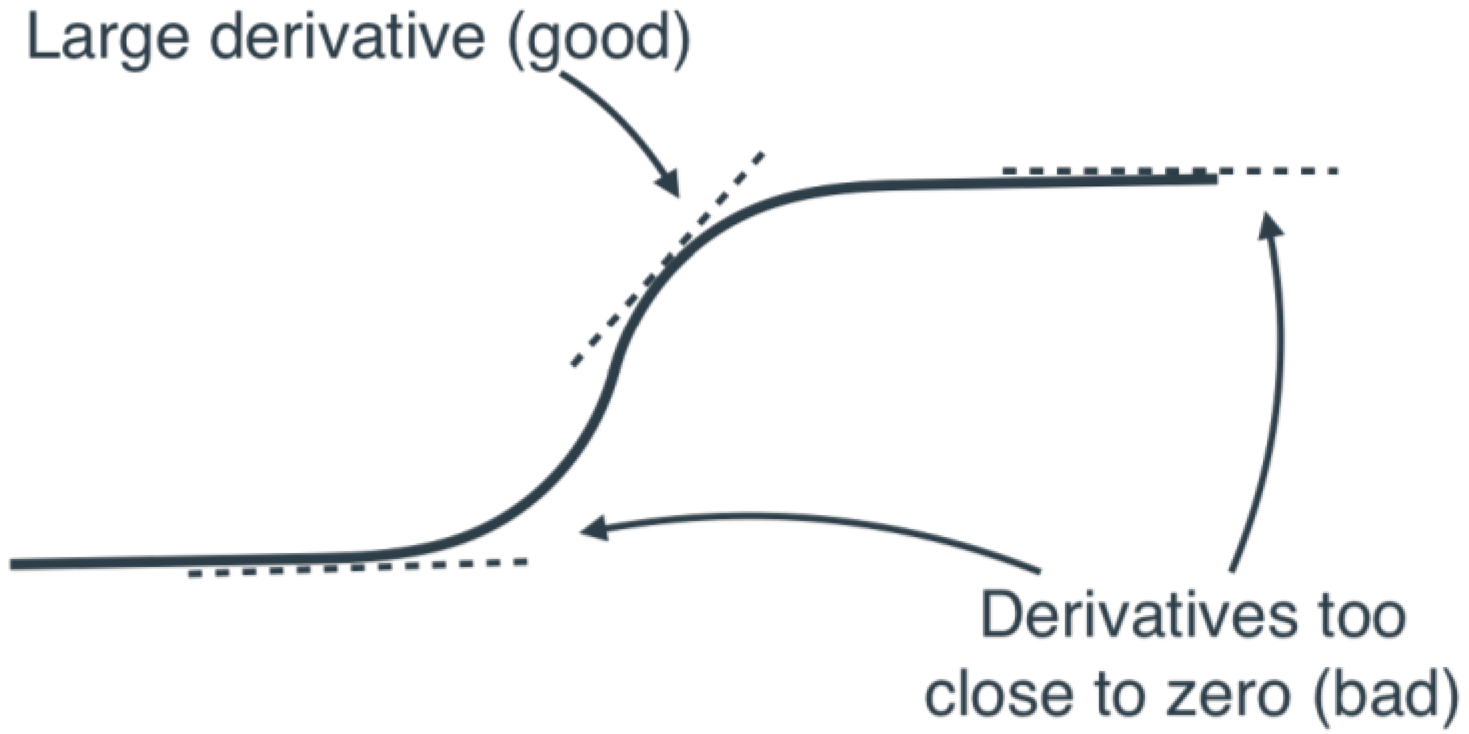
- When we talk about the gradient of the loss function, we are thinking of a derivative.
- The loss function of a neural network is complicated, since it involves the logarithm of the prediction, and the prediction itself is a complicated function (it contains several linear equations and sigmoids).
- Furthermore, we need to calculate the derivative with respect to many variables, one corresponding to each of the weights and biases of the neural network.


Bad classifier
Decrease weight
Resulting classifier
Good classifier
Increase weight

Training neural networks

- In practice, neural networks work very well, But as you can imagine, due to their complexity, many problems arise with their training.
- Luckily, we can put a solution to the most pressing ones, One problem that neural networks have is overfitting, as really big architectures can potentially memorize our data without generalizing it well.
- Another problem they can have is vanishing gradients, in which the values used to update the weights get so small, that training gets stuck.

Training neural networks



Training neural networks

Techniques for training your neural network - Dropout, regularization

- As I mentioned before, neural networks are very prone to overfitting.
- In this section I'll teach you some techniques to decrease the amount of overfitting during the training of neural networks.
- The first question you may have is: How do I pick the correct architecture? This is a very difficult question, and there is no concrete answer

Training neural networks

REGULARIZATION - A WAY TO REDUCE OVERFITTING BY PUNISHING HIGHT WEIGHTS

- As we've learned in this course, L1 and L2 regularization are techniques that we can use to decrease overfitting in many algorithms such as linear and logistic regression, and neural networks are not the exception.

Training neural networks

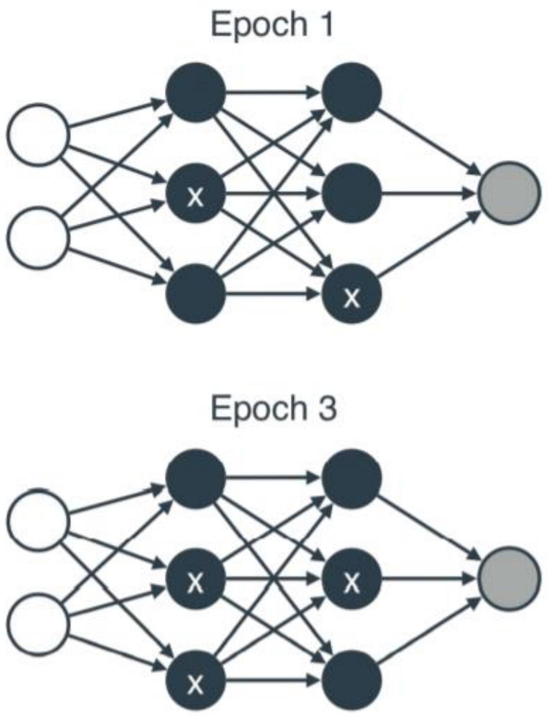
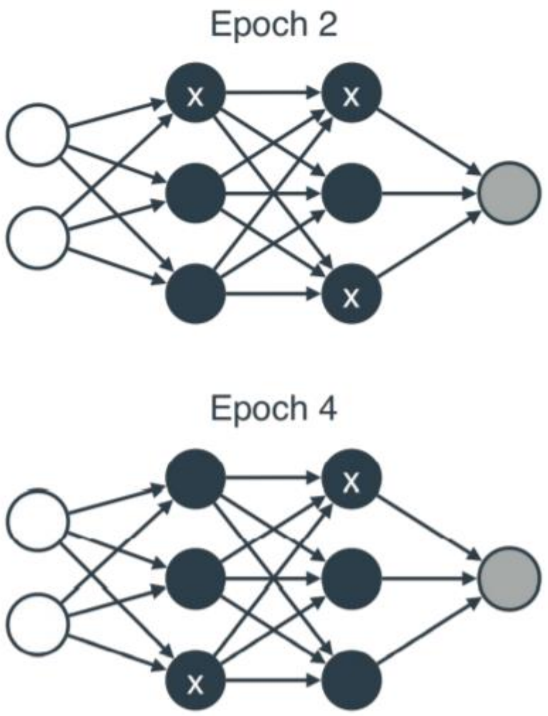
DROPOUT - A WAY TO MAKE SURE A FEW STRONG NODES ARE NOT DOMINATING THE TRAINING

- Dropout is a very interesting technique used to reduce overfitting in neural networks, and to understand it I like to think of the following analogy.
- Let’s say that we are right-handed, and we like to go to the gym. After some time we start noticing that after a while in the gym, our right bicep is growing a lot, but our left one is not growing at all.

Training neural networks

- Dropout uses this, except instead of arms, we train the weights in the neural network.
- One problem neural networks have is that if a neuron has very large weights, that neuron dominates the prediction step and drowns the smaller neurons around it, not allowing them to train properly.
- The dropout process attaches a small probability p to each of the neurons.

Training neural networks

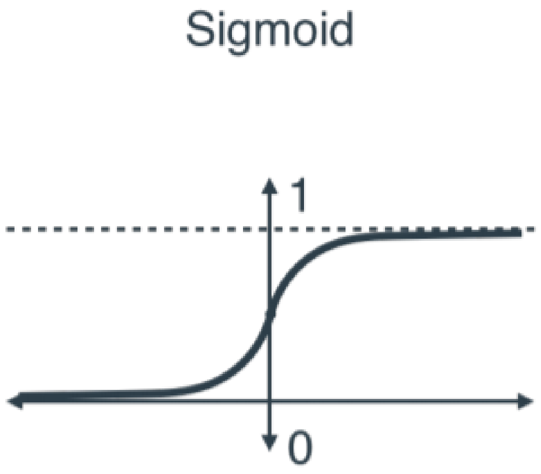
Training neural networks

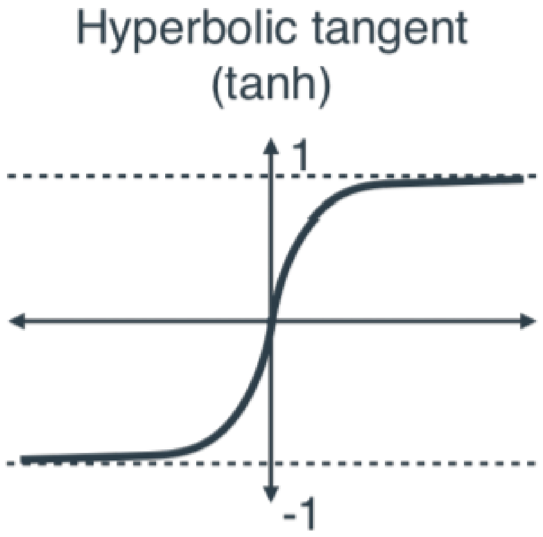
HYPERBOLIC TANGENT (TANH)

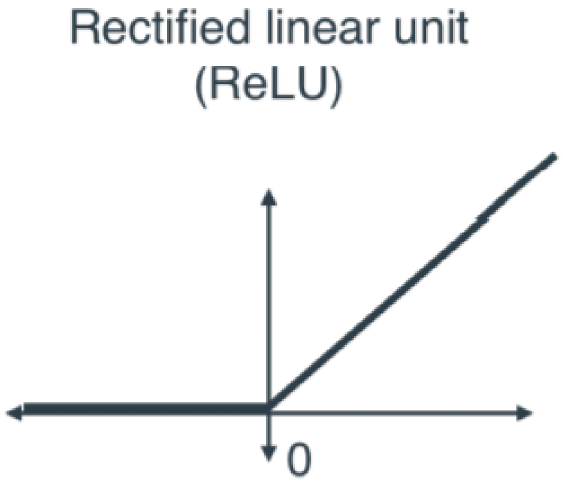
- The hyperbolic tangent function tends to work better than the sigmoid function in the practice,
- due to its shape.
- This one is given by the following formula:

$$tanh\,(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}.$$

Training neural networks







Training neural networks

RECTIFIED LINEAR UNIT (RELU)

- A much more popular activation that is commonly used in neural networks is the Rectified Linear Unit, or ReLU.
- This one is very simple, if the input is negative, the output is zero, while if the input is positive, the output is the same input.
- In other words, if $x \geq 0$, the output is x, and if $x \leq 0$, the output is 0.

Training neural networks

More than one input? No problem, the softmax function is here to help

- So far, neural networks have solved binary classification problems, namely, problems in which our outputs consist of two classes.
- However, if our data has more than 2 labels, we can do a simple modification to our neural network architecture to allow more than two classes.

Training neural networks

- Let’s look at a small example. We want to build a neural network that recognizes images in a dataset which contains images of dogs, cats, and birds.
- Our goal is for our network to output information on how likely an image is to be of a dog, cat, or bird.
- The first question we may ask is, how do we want this output to look? Since we want probabilities, an answer that makes sense is the following.

Training neural networks

For example, that it outputs the numbers 1, 2, and 3. This means that the neural network has given the dog a score of 1, the cat a score of 2, and the bird a score of 3. Let’s record that.

- Score(dog) = 1
- Score(cat) = 2
- Score(bird) = 3.

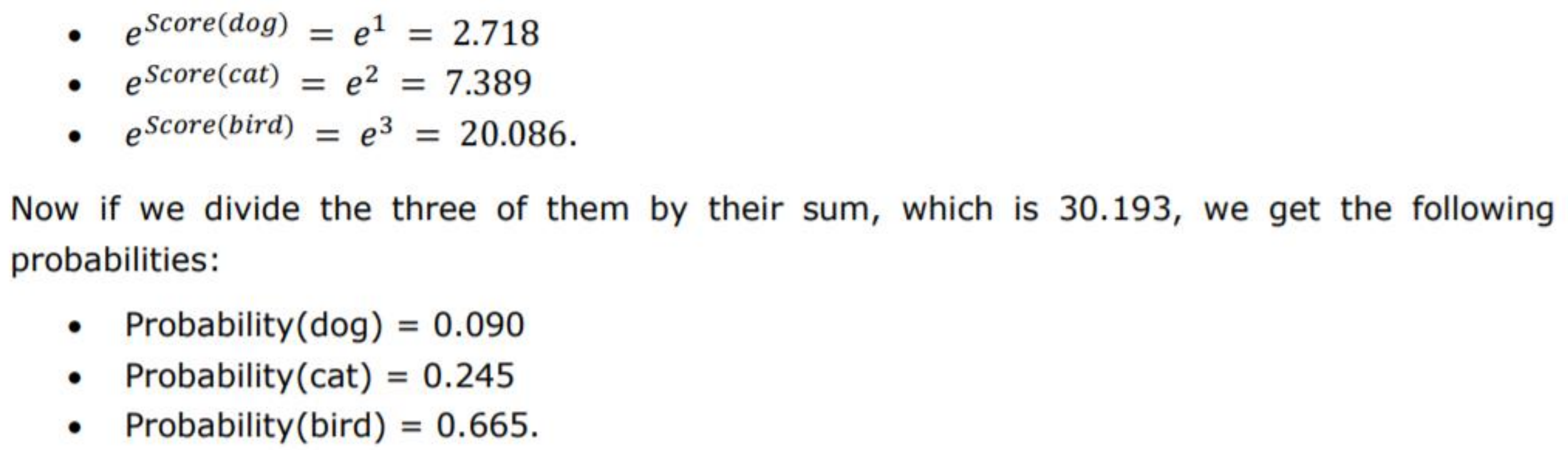
Training neural networks

the image seems to be more likely to be of a bird. However, what are the probabilities? One simple technique to turn numbers into probabilities is to divide by their sum. In this case, the sum is 6, so by dividing everything by 6, we get the following:

- Probability(dog) = ⅙
- Probability(cat) = 2/6 = ⅓
- Probability(bird) = 3/6 = ½

Training neural networks

- I can think of one, the exponential function! What if we raise e to the power of the scores?



- $e^{Score(dog)} = e^1 = 2.718$
- $e^{Score(cat)} = e^2 = 7.389$
- $e^{Score(bird)} = e^3 = 20.086.$

Now if we divide the three of them by their sum, which is 30.193, we get the following probabilities:

- Probability(dog) = 0.090
- Probability(cat) = 0.245
- Probability(bird) = 0.665.

Training neural networks

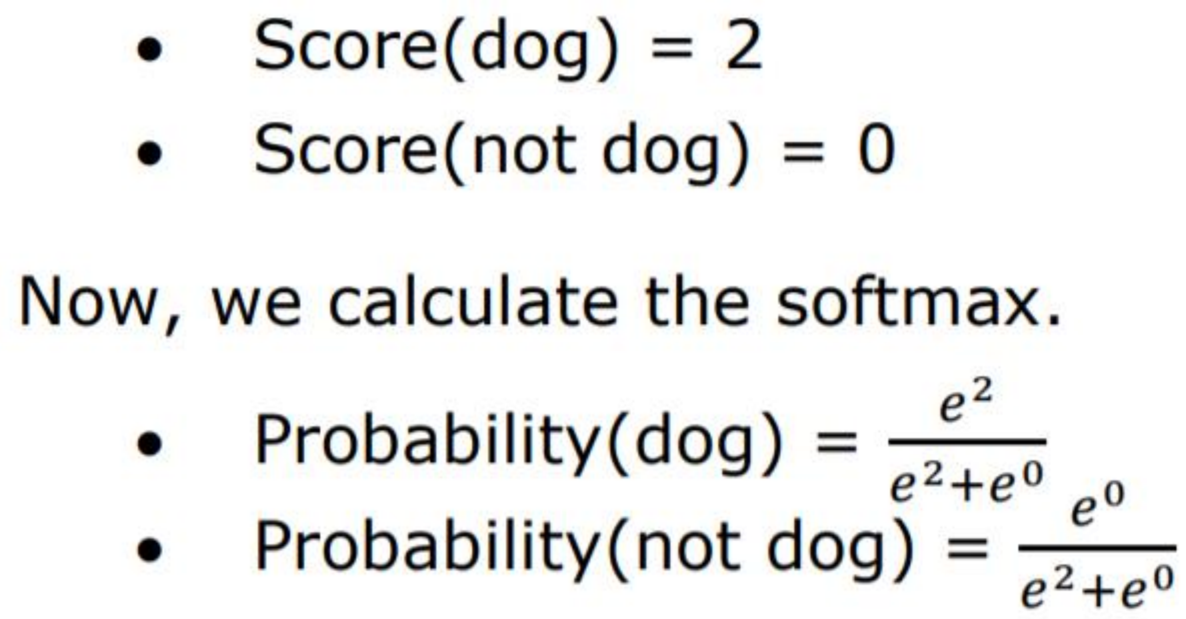
SOFTMAX FUNCTION

- Given n scores, aa1, aa2, … , $aann$, the softmax function will output the n numbers (pp1, pp2, … , $ppnn$) which add to 1, and correspond to probabilities, where

$$p_i = \frac{e^{a_i}}{e^{a_1}+e^{a_2}+\cdots+e^{a_n}}.$$

Training neural networks

- We have the following:



- Score(dog) = 2
- Score(not dog) = 0

Now, we calculate the softmax.

- Probability(dog) = $\frac{e^2}{e^2+e^0}$
- Probability(not dog) = $\frac{e^0}{e^2+e^0}$

Training neural networks

Remembering that $e^0 = 1$, we get that the probability that the image is a dog is $\frac{e^2}{e^2+1}$. Multiplying the numerator and the denominator by e^2, we get that

$$P(dog) = \frac{1}{1+e^{-2}} = \sigma(2).$$

Training neural networks

Hyperparameters - what we fine tune to improve our training

- Learning rate: the size of the step that we use during our training.
- Number of epochs: The number of steps we use for our training.
- Batch vs mini-batch vs stochastic gradient descent: How many points at a time enter the training process. Namely, do we enter the points one by one, in batches, or all at the same time?

Training neural networks

LEARNING RATE - THE LENGTH OF THE STEP THAT WE USE DURING OUR TRAINING

- Just like in linear or logistic regression, neural networks use the learning rate as the size of each step that it takes during the training process.
- This is a very important hyperparameter, because a learning rate that is too big will give very large steps to get to the solution, but it may accidentally miss it.

Training neural networks

NUMBER OF EPOCHS - THE NUMBER OF STEPS WE USE FOR OUR TRAINING

Normally, the more computational power we have, the more epochs we should use. However, there are other ways to tell when to stop the training, such as the following:

- When the loss function reaches a certain value that we have predetermined.
- When the loss function doesn't decrease during several epochs.

Training neural networks

BATCH VS MINI-BATCH VS STOCHASTIC GRADIENT DESCENT - HOW MANY POINTS AT A TIMEENTER THE TRAINING PROCESS

- The training process of a neural network can be very slow if our dataset is very large. Luckily, there are tricks to speed this process up.
- In the same way that we eat a large sandwich by taking small bites, we can train a model on a very large dataset by dividing it into many batches (chosen at random), and applying backpropagation in each one of these batches separately.

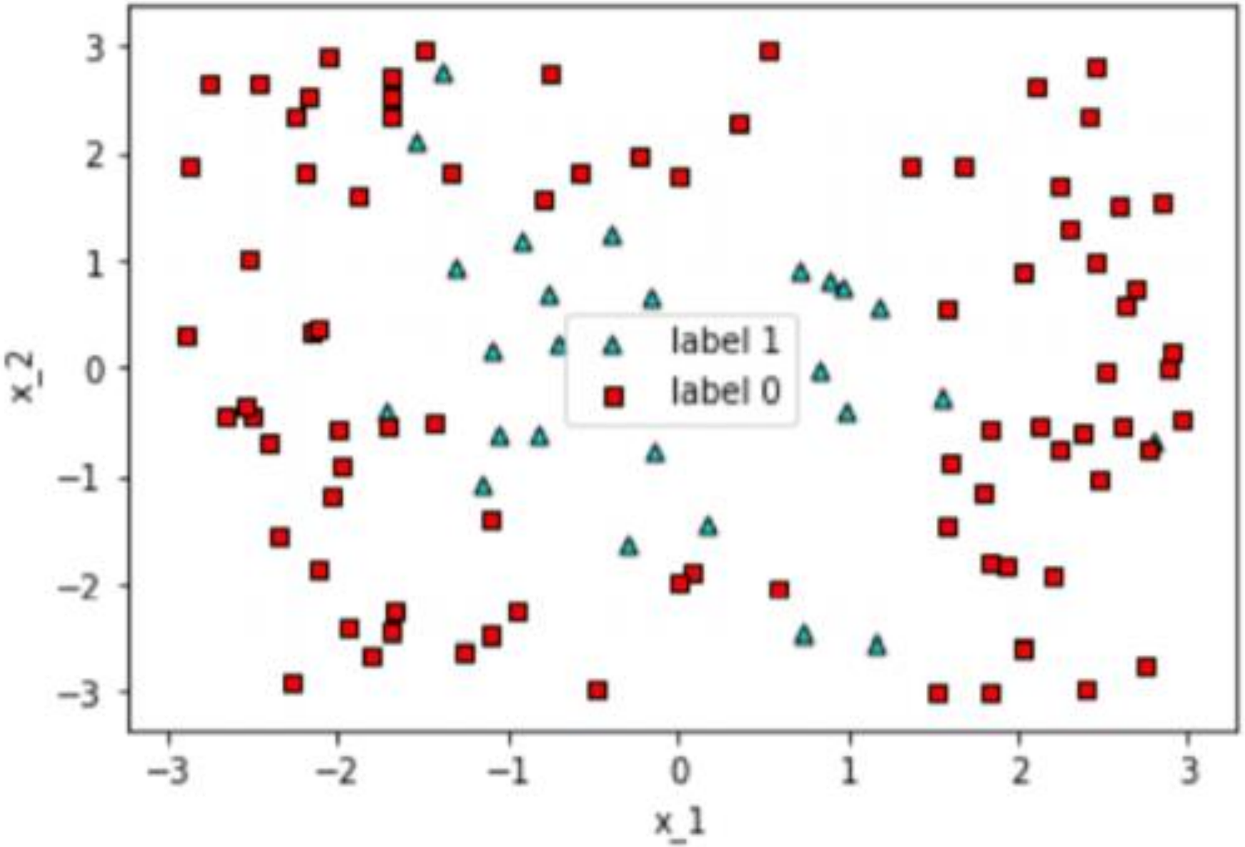
Training neural networks

- There are pros and cons to both. Batch gradient descent is more exact, since each step that we take in the training process considers all the data, so the steps are very accurate.

- Stochastic gradient descent is much faster, but since each step doesn't use the whole dataset, the steps are more chaotic.

How to code a neural network in Keras

- Now that we’ve learned how neural networks operate, it’s time to train a real neural network!
- In this section I will show you how to code a neural network in Keras.
- We will start by loading a sample dataset with two classes.

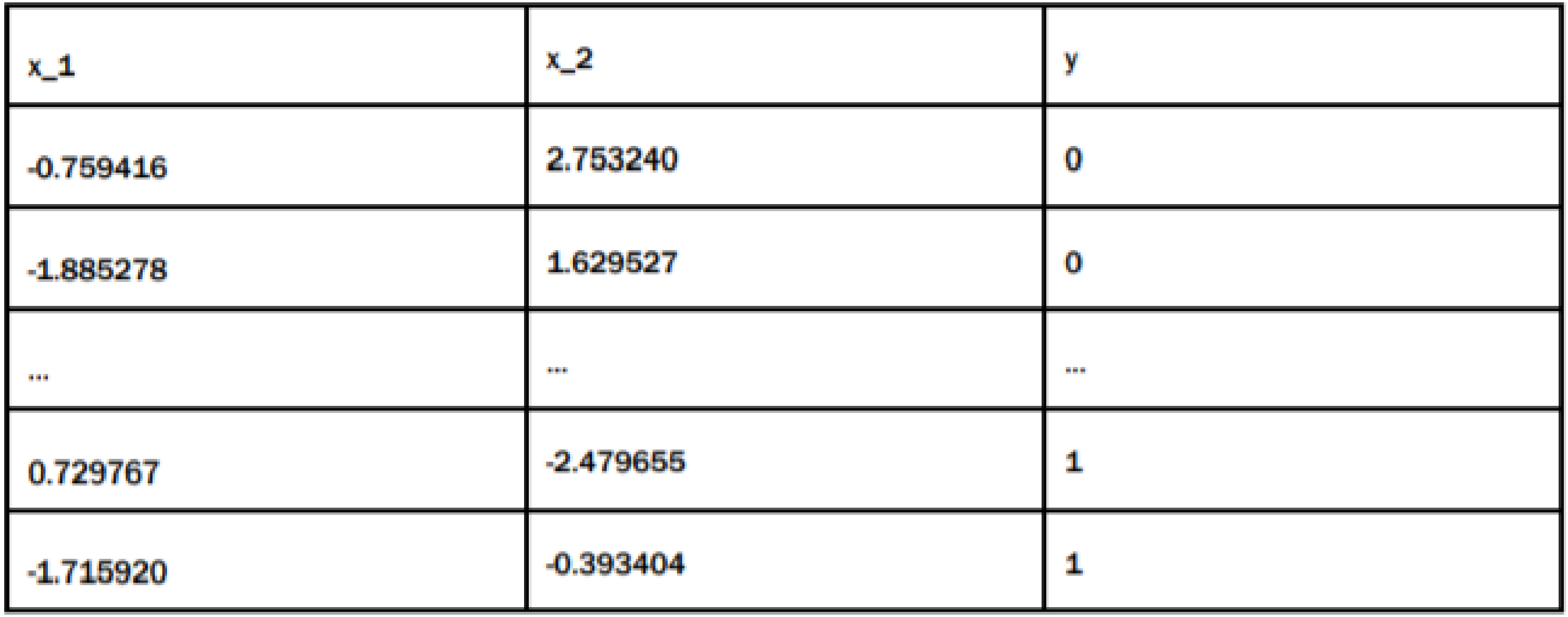

label 1
label 0
x_1
x_2
−3
−2
−1
0
1
2
3

How to code a neural network in Keras

- There are many very good packages to code neural networks. Some of the most popular are Keras, TensorFlow, and Pytorch.

- In this lesson I’ll teach you Keras, but I encourage you to check out all of them, as they are very useful and powerful.

How to code a neural network in Keras



x_1	x_2	y
-0.759416	2.753240	0
-1.885278	1.629527	0
...	...	...
0.729767	-2.479655	1
-1.715920	-0.393404	1

How to code a neural network in Keras

Categorizing our data - a way to turn categorical features into numbers

- Preprocessing data is one of the most important parts of the work of a data scientist.
- There is a particular preprocessing step which is recommended when training neural networks, called categorizing the data, which I'll show you in this section.

How to code a neural network in Keras

- We’d like to one-hot encode this data, in such a way that the labels are two columns.
- The labels of these two columns are called y_0 and y_1, and they are the following.

If y = 0, then y_0 = 1 and y_1 = 0.
If y = 1, then y_0 = 0 and y_1 = 1.

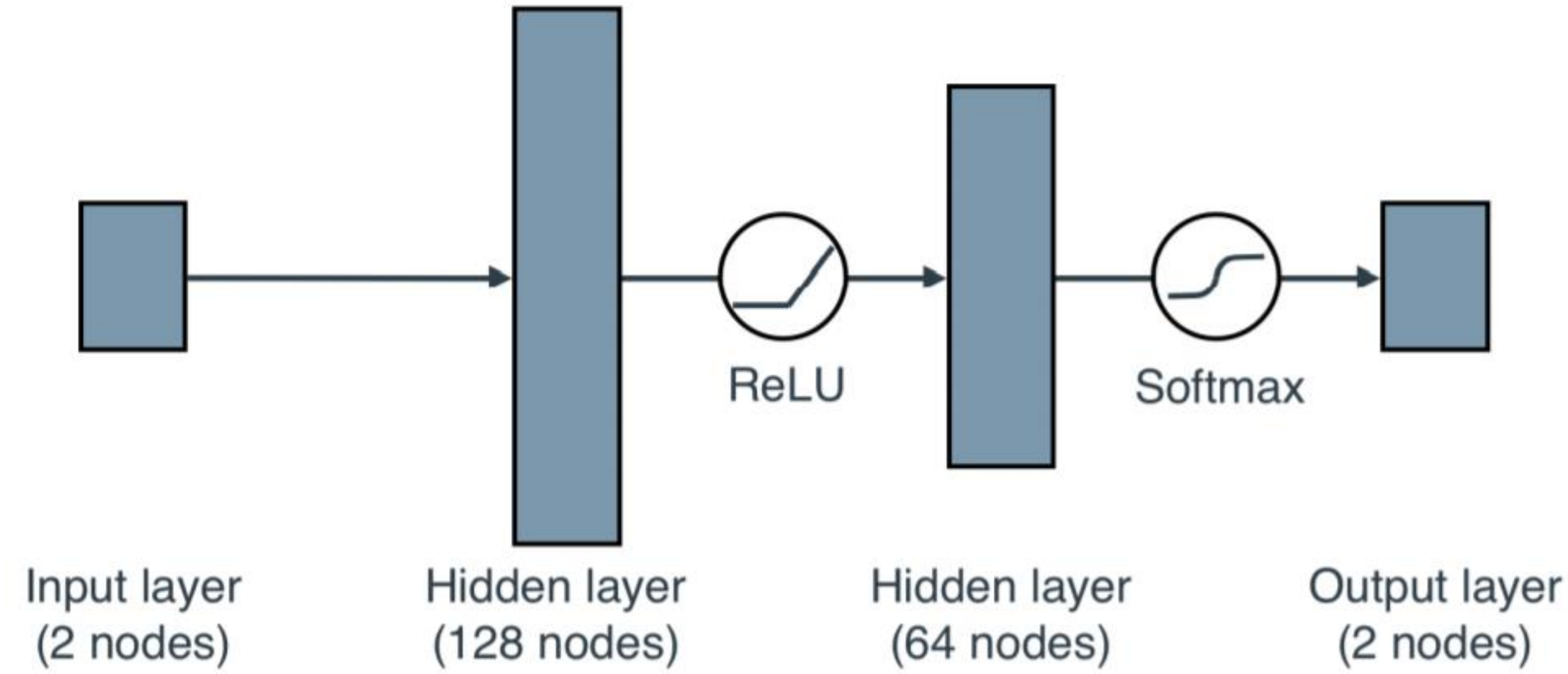
- This is easily done with the following command

```
from tensorflow.keras.utils import to_categorical
y = np.array(to_categorical(y, 2))
```

The architecture of a neural network that we'll use to train this dataset

- Input layer
  - Size: 2
- First hidden layer
  - Size:128
  - Activation function: ReLU
- Second hidden layer
  - Size: 64
  - Activation function: ReLU
- Output layer (size = 2)
  - Size: 2
  - Activation function: Softmax

How to code a neural network in Keras



How to code a neural network in Keras

- Why this architecture? Why not more layers, or less, or a different number of nodes? There is no answer to that.
- I normally use layers of size equal to a power of two, such as 64, 128, 256, or 512, and architectures of 2 or 3 layers are common for small datasets like this one.
- You normally want an architecture that is big enough to handle your data, but not that bit that it uses too many computational resources.

How to code a neural network in Keras

The next step in training our model is to define the architecture that we'll use. This comprises the following:

- The number of layers.
- The size (number of nodes) of each layer.
- The activation functions used at each layer.
- Other options, such as using dropout in that particular layer during the training

How to code a neural network in Keras

- This can all be done in Keras in only a few lines of code.
- The first thing we have to do is some useful imports.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation
```

How to code a neural network in Keras

```
model = Sequential()
```

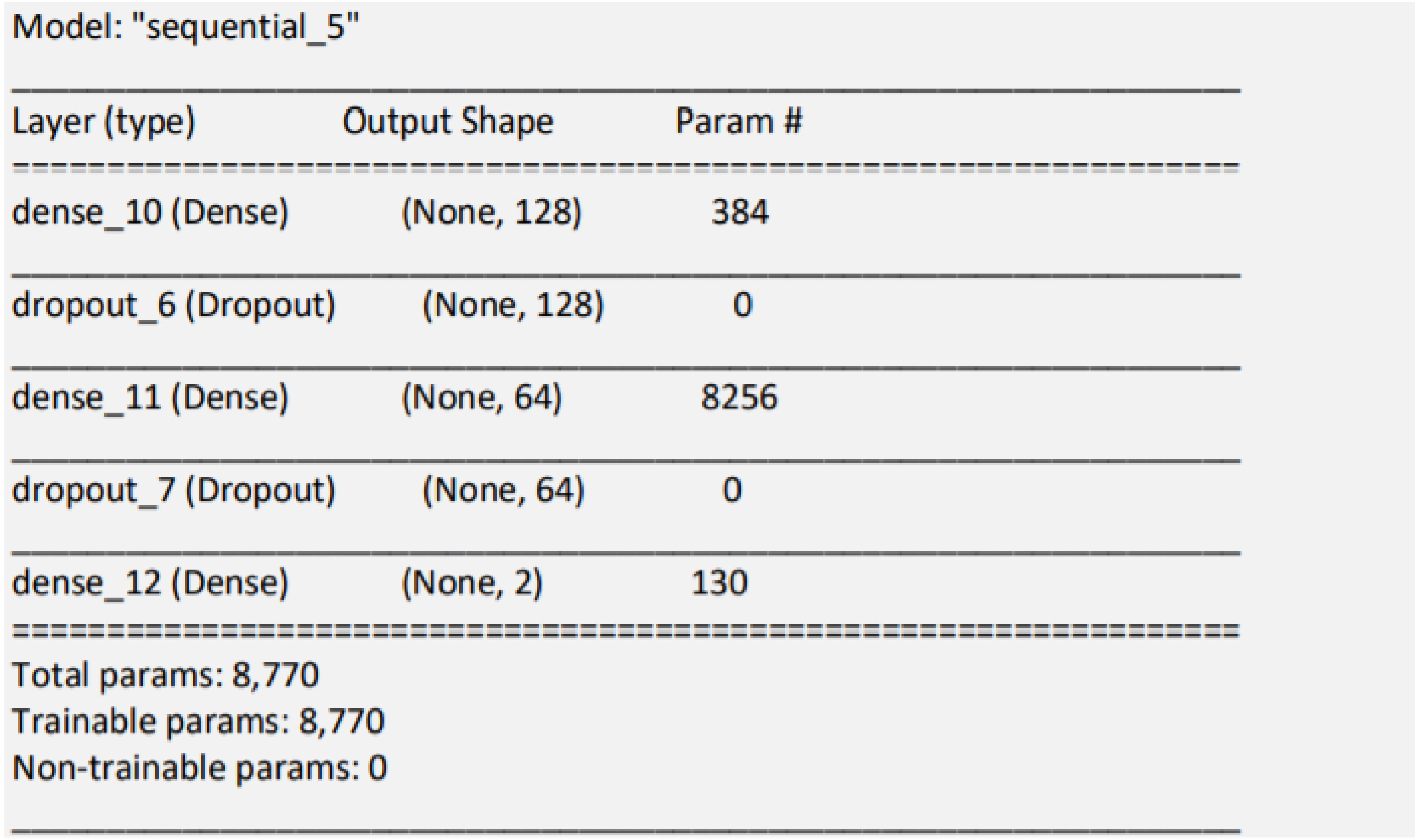
- Now, we proceed to add the hidden layers and the output layer, as follows:

```
model.add(Dense(128, activation='relu',
input_shape=(2,))) #A
model.add(Dropout(.2)) #B
model.add(Dense(64, activation='relu')) #C
model.add(Dropout(.1))
model.add(Dense(2, activation='softmax')) #D
```

How to code a neural network in Keras

- Once the model is defined, we need to compile it with the following line of code.

```
model.compile(loss = 'categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
```



```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 128)               384
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_11 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_7 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_12 (Dense)             (None, 2)                 130
=================================================================
Total params: 8,770
Trainable params: 8,770
Non-trainable params: 0
_________________________________________________________________
```

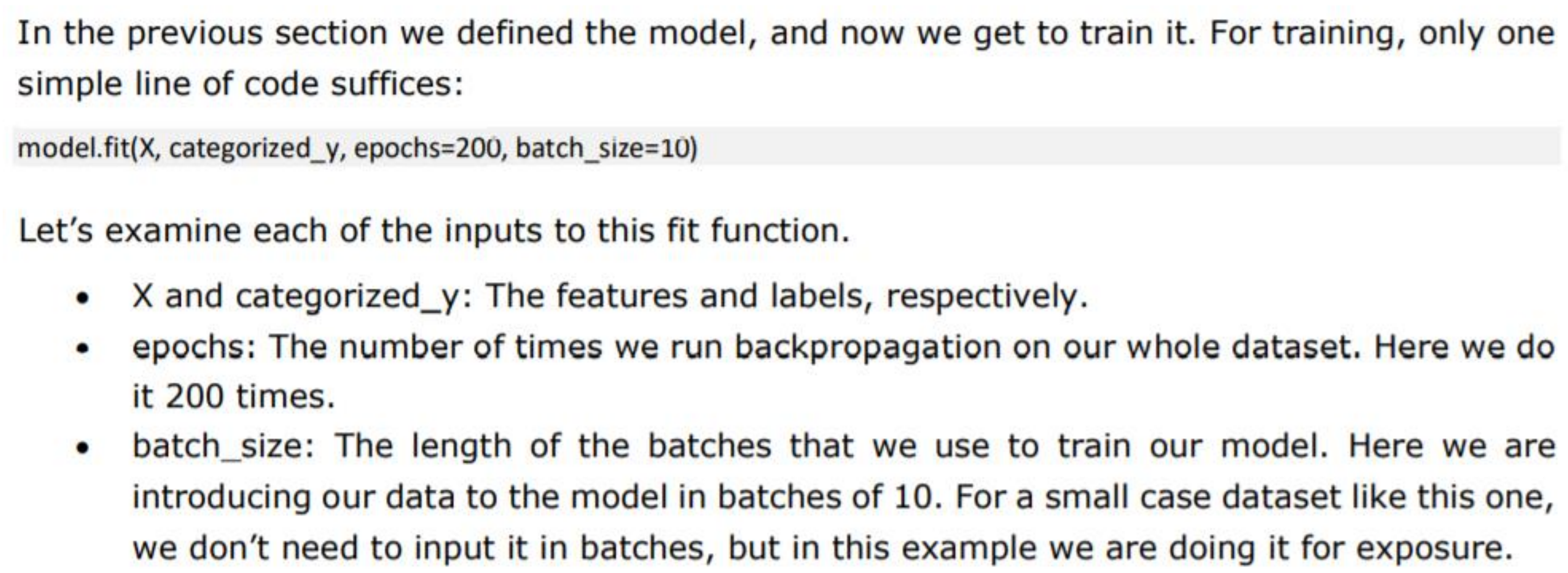
How to code a neural network in Keras

- Each row in the previous output is a layer (dropout layers are treated as separate layers for description purposes).

- The columns correspond to the type of the layer, the shape (number of nodes), and the number of parameters, which is precisely the number of weights plus the number of biases.

How to code a neural network in Keras

Training the model in Keras



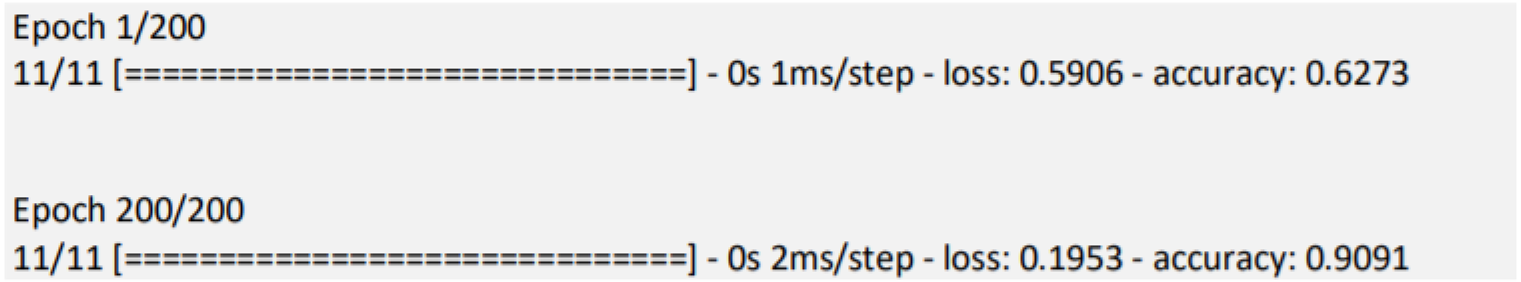
In the previous section we defined the model, and now we get to train it. For training, only one simple line of code suffices:

```
model.fit(X, categorized_y, epochs=200, batch_size=10)
```

Let's examine each of the inputs to this fit function.

- X and categorized_y: The features and labels, respectively.
- epochs: The number of times we run backpropagation on our whole dataset. Here we do it 200 times.
- batch_size: The length of the batches that we use to train our model. Here we are introducing our data to the model in batches of 10. For a small case dataset like this one, we don't need to input it in batches, but in this example we are doing it for exposure.

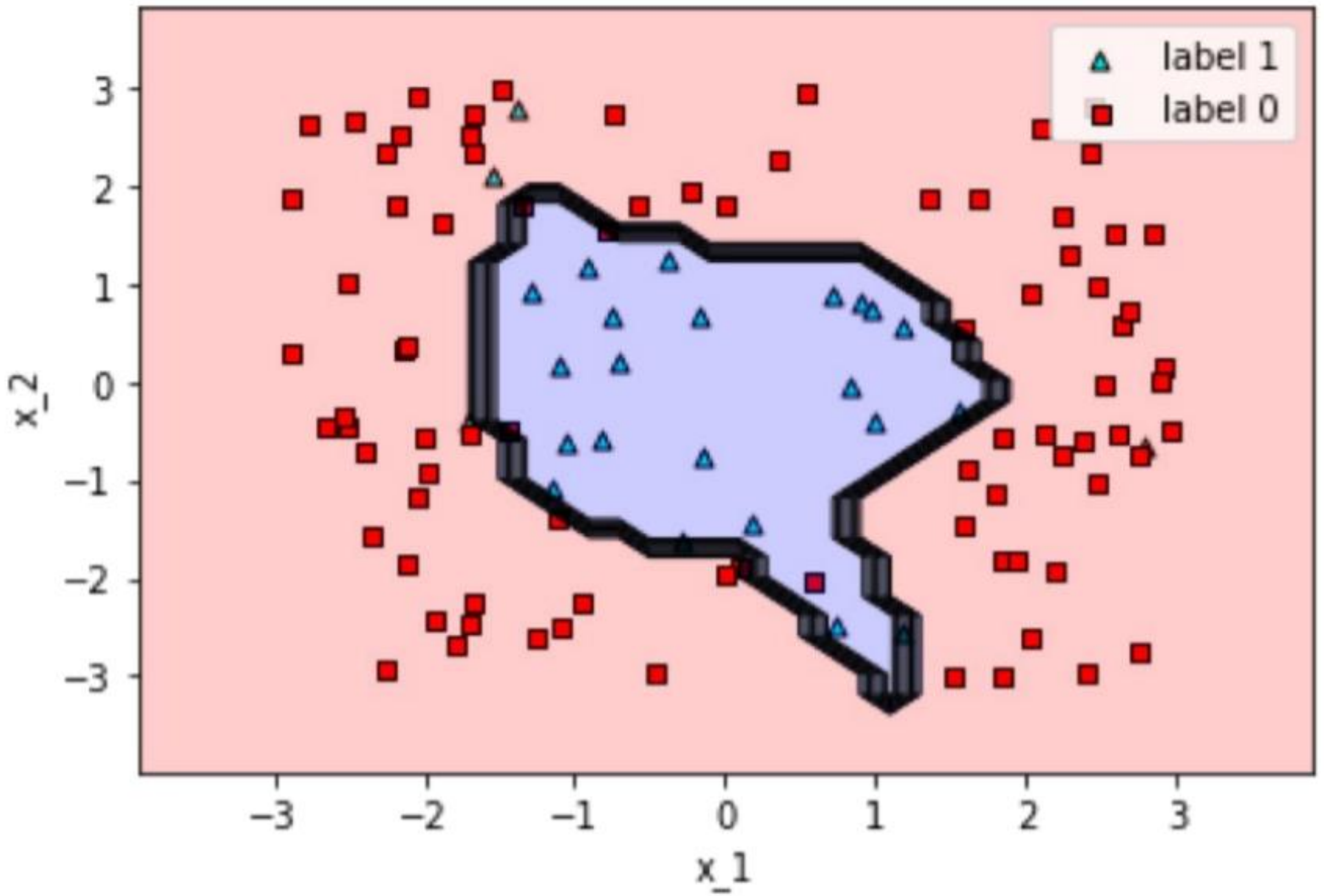
How to code a neural network in Keras



```
Epoch 1/200
11/11 [==============================] - 0s 1ms/step - loss: 0.5906 - accuracy: 0.6273


Epoch 200/200
11/11 [==============================] - 0s 2ms/step - loss: 0.1953 - accuracy: 0.9091
```

- The final accuracy of the model is 0.9091, which is pretty good.
- Now, let’s plot the boundary with our plot_model function to have a visual of how the neural network performed


label 1
label 0
x_1
x_2
−3
−2
−1
0
1
2
3

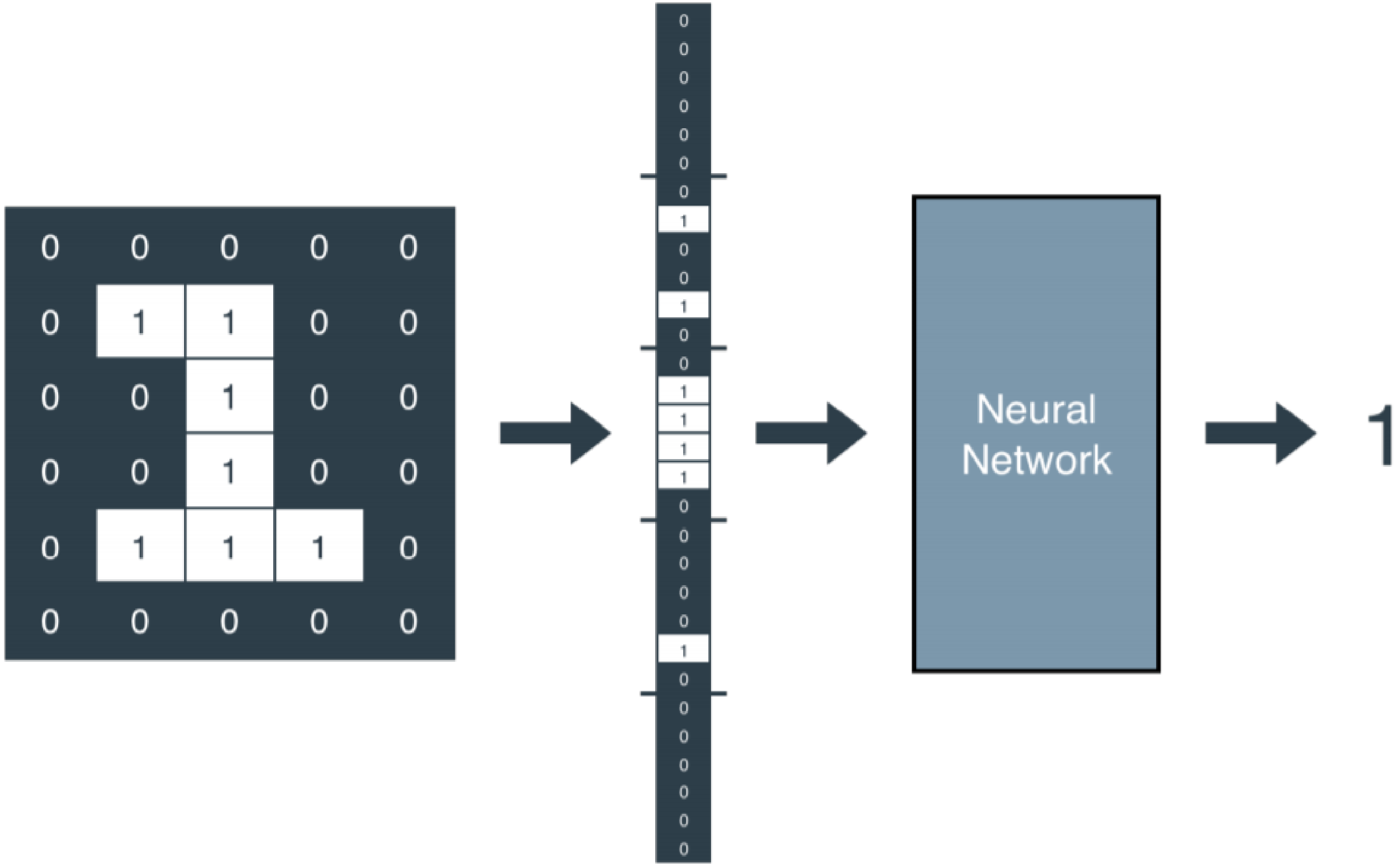
Other more complicated architectures and some sci-fi applications

- Neural networks are useful in many applications, perhaps more so than any other machine learning algorithm currently.
- One of the most important qualities of neural networks is their versatility.
- We can modify the architectures in very interesting ways in order to better fit our data and solve our problem.

Other more complicated architectures and some sci-fi applications

How neural networks see - Image recognition

- Neural networks are great with images, and there are many applications where one can use this, such as the following:
- Image recognition: The input is an image, and the output is the label on the image.
- Some famous datasets used for image recognition are the following:
  - o MNIST: Handwritten digits in 28 by 28 gray scale images.


0 0 0 0 0
0 1 1 0 0
0 0 1 0 0
0 0 1 0 0
0 1 1 1 0
0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0
Neural Network
1

Other more complicated architectures and some sci-fi applications

- This method works well for simple images, such as handwritten digits as in the MNIST dataset.

- However, for more complicated images such as pictures, faces, etc., the neural network won’t do very well.

- This is because turning the image into a long vector loses a lot of information.

Other more complicated architectures and some sci-fi applications

How neural networks talk - Natural language processing

- One of the most fascinating applications of neural networks is when we can get them to talk to us.
- This involves listening to what we say or reading what we write, analyzing it, and being able to respond or take action.
- The ability for computers to understand and process language is called natural language processing. Neural networks have had a lot of success in natural language processing

Other more complicated architectures and some sci-fi applications

How neural networks generate faces that look real – Generative adversarial networks

- In my opinion, the most fascinating among all the current applications of neural networks is in generation.
- So far, neural networks have worked well in predictive machine learning, namely, being able to answer questions successfully.
- For example, a machine learning model can answer questions such as “how much is that?”, or “is this A or B?”.

Summary

- Neural networks are a very powerful algorithm used for classification and regression.
- A neural network consists of a set of perceptrons organized in layers, where the output of one layer serves as input to the next layer.
- Their complexity allows them to achieve great success in applications that are very difficult for other machine learning models.
- Neural networks have cutting edge applications in many areas, including image recognition and text processing.