NEURAL NETWORKS

Clayton Leite

Overview

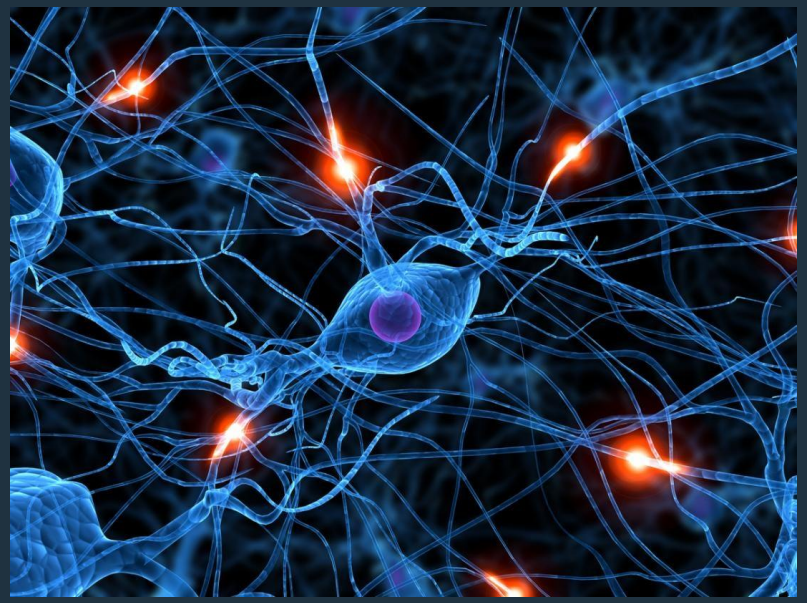
- Basics of neural networks (architecture, weights, loss function, optimizer, etc.)
- Basics of deep learning (automatic feature extraction, convolutions, long short-term memory)

01 WHAT ARE NEURAL NETWORKS?

We will talk about the basic concept of neural networks and what they can be used for

Neural Networks

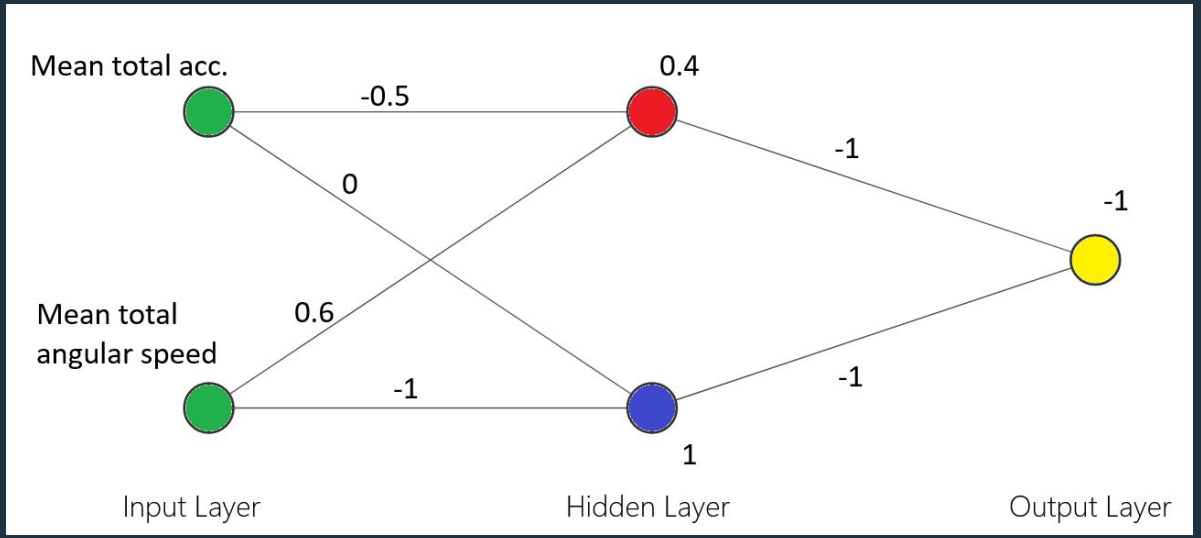
- (Artificial) neural networks consist of a machine learning method.
- They are inspired in how our brains work.
- Just like in our brains, (artificial) neural networks are composed of interconnected neurons



Neural Networks

- Let’s start looking at the simplest possible neural network
- And let’s consider again the case of classifying “walking” and “running”
- The neural network on the right classifies the samples on the left with 80% success rate (accuracy).

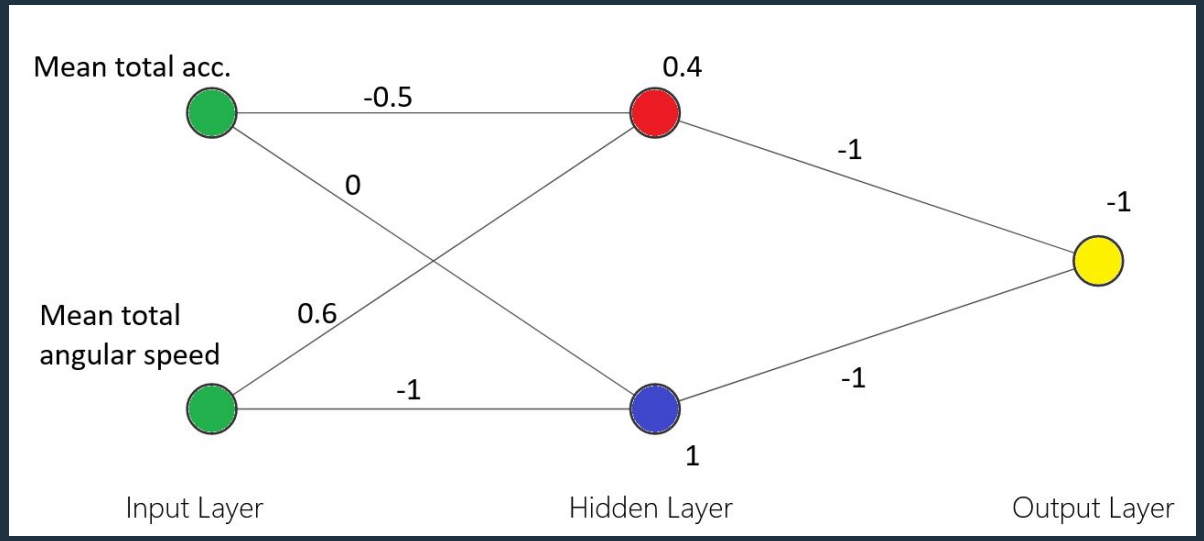
Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running



Neural Networks

- Let’s see how it works step by step.
- Let’s start classifying the first sample *(3.2, 1.8)*
- We will plug the values 3.2 and 1.8 at the input layer and perform some calculations according to the diagram.

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running

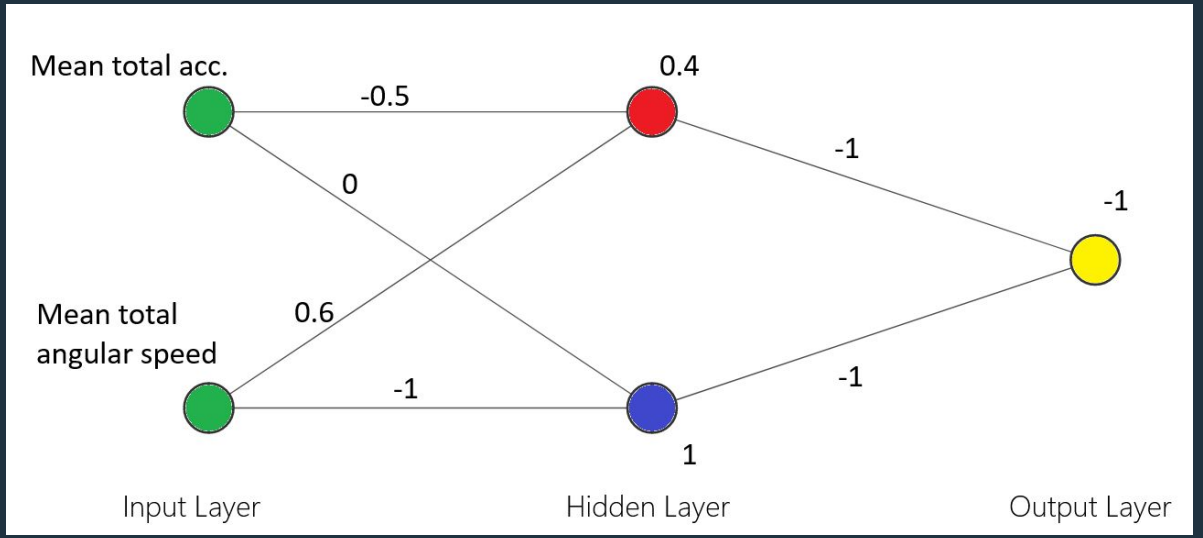


Neural Networks

- Our goal is to find what comes out of the yellow neural (the output – the end).

- We start from the left to calculate what comes out of the red and blue neurons.

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running



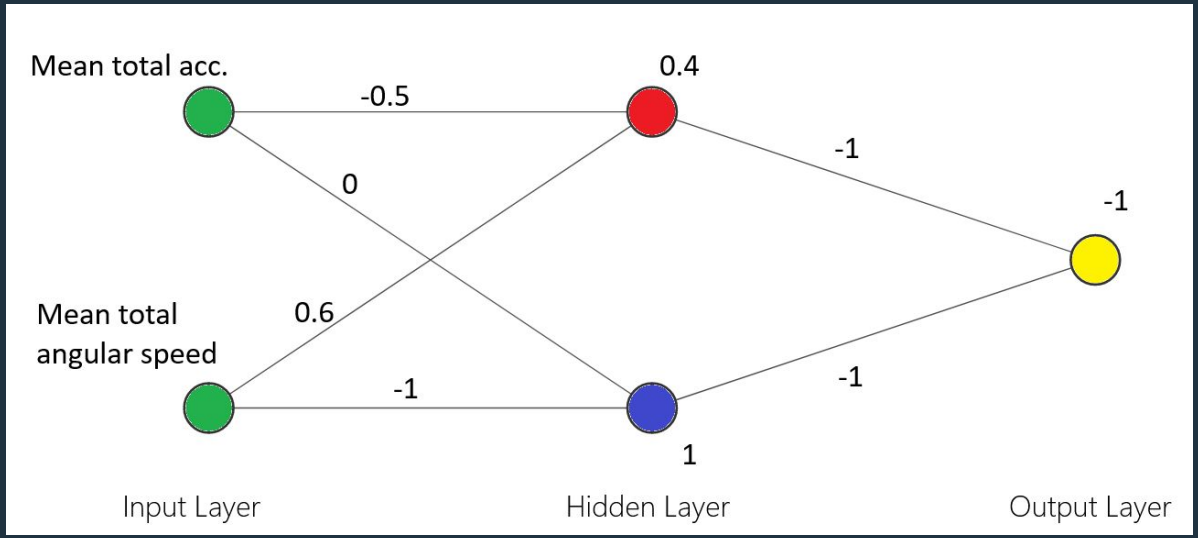
Neural Networks

- Sample: *(3.2, 1.8)*
- Red neuron gives: 3.2*(-0.5) + 1.8*(0.6) + 0.4 = -0.12

  weight weight bias
- Blue neuron gives: 3.2*(0) + 1.8*(-1) + 1 = -0.8

  weight weight bias

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running

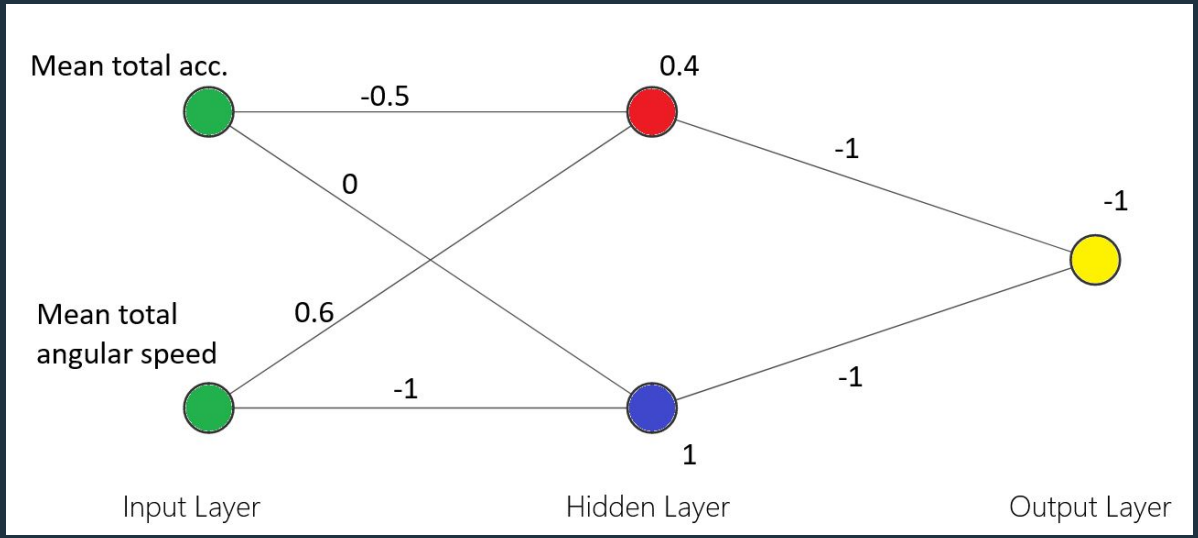


Neural Networks

- What comes out of the red neuron is -0.12 and what comes out of the blue one is -0.8
- Now, we can determine what comes out of the yellow neuron
- Yellow neuron gives: -0.12*(-1) + -0.8*(-1) + (-1) = -0.08

weight weight bias

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running



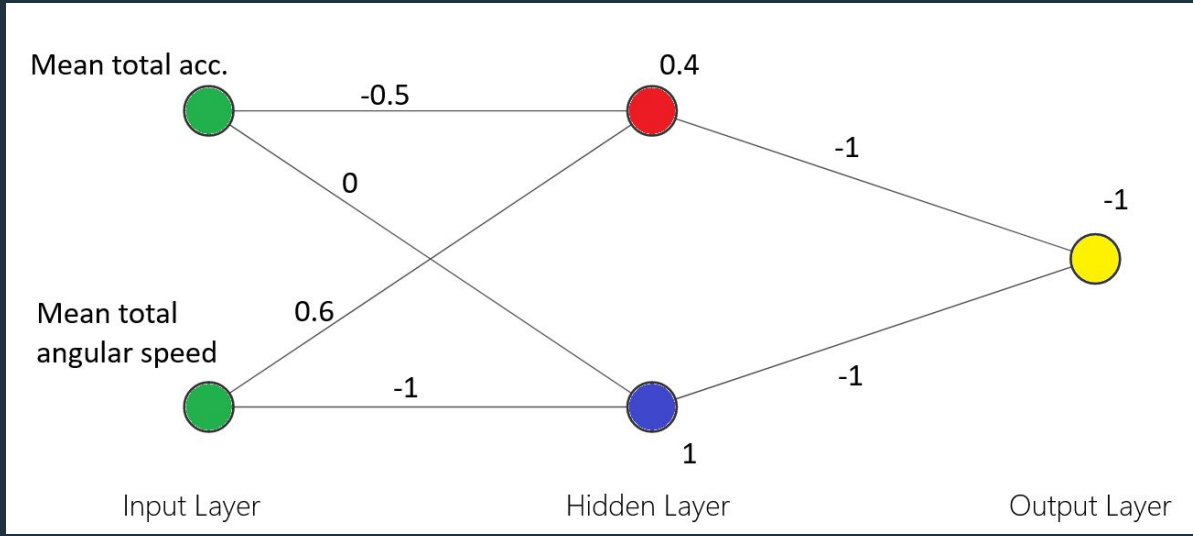
Neural Networks

- How to make sense of the value -0.08?

- We will see how neural networks are built. But in the way this one was built, negative values represent the class “walking”, whereas positive values represent “running”

- Hence, this neural network correctly classifies the sample *(3.2, 1.8)*

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running



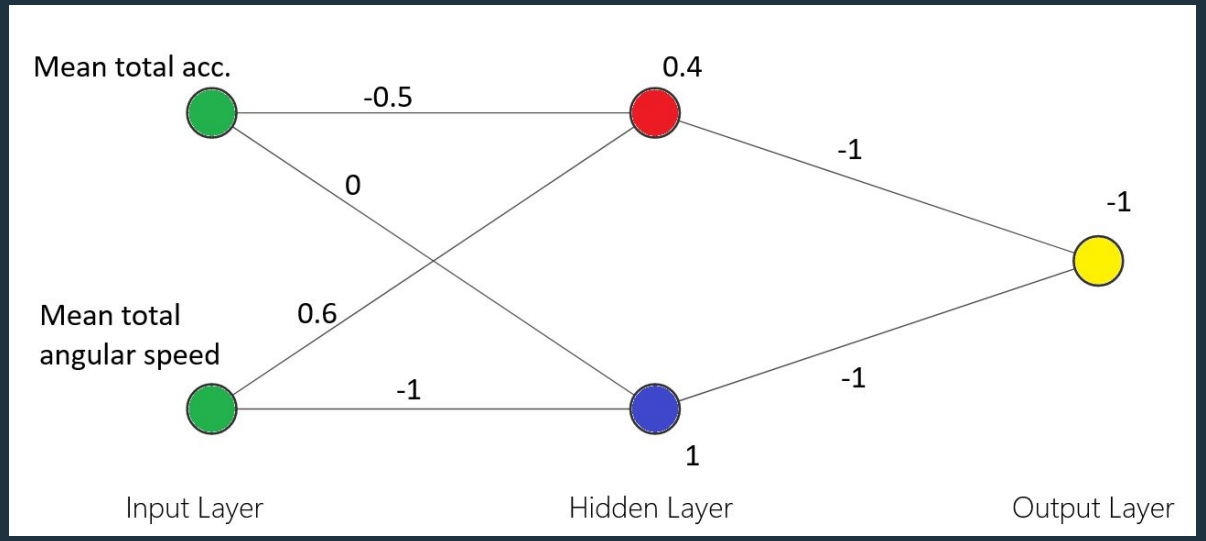
Neural Networks

- Sample: *(2.7, 3.6)*
- Red neuron gives: 2.7*(-0.5) + 3.6*(0.6) + 0.4 = 1.21

  weight weight bias
- Blue neuron gives: 2.7*(0) + 3.6*(-1) + 1 = -2.6

  weight weight bias

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running

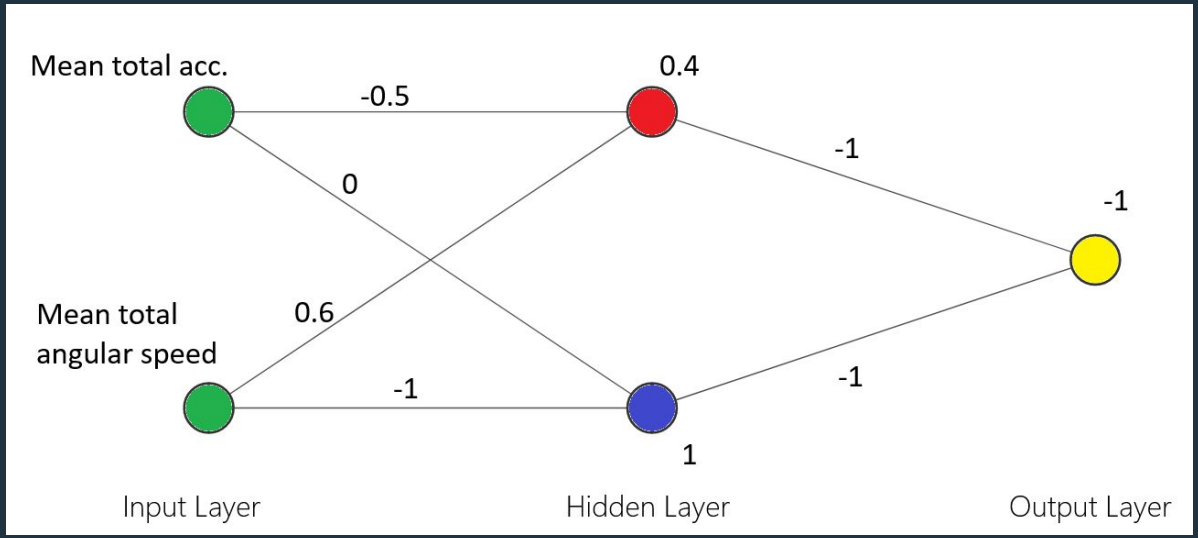


Neural Networks

- What comes out of the red neuron is 1.21 and what comes out of the blue one is -2.6
- Now, we can determine what comes out of the yellow neuron
- Yellow neuron gives: 1.21*(-1) + -2.6*(-1) + (-1) = 0.39

weight weight bias

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running



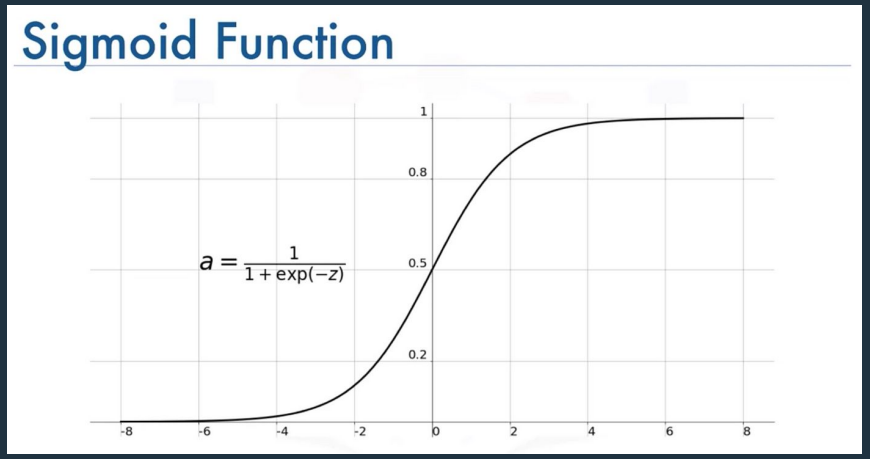
Neural Networks

- And what does 0.39 represent? The “running” class

- Instead of having values ranging from -inf to inf, we could have values ranging from 0 to 1. These values could represent the probability.

- We can do this by applying the **sigmoid** function.

```
probability = 1/(1 + exp(-output))
```



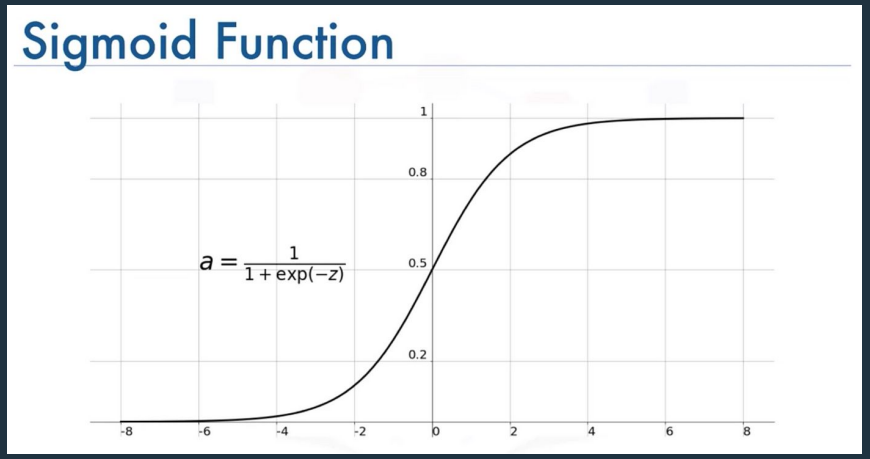
Neural Networks

- Hence, for the sample (3.2, 1.8) instead of -0.08, we have:

  probability = 1/(1 + exp(0.08)) = 48.0%

- For the sample (2.7, 3.6) instead of 0.39, we have:

  probability = 1/(1 + np.exp(-0.39)) = 59.6%



Neural Networks

- Hence, for the sample *(3.2, 1.8)* instead of -0.08, we have:

  probability = 1/(1 + exp(0.08)) = 48.0%

  Interpretation: the sample *(3.2, 1.8)* has a probability of 48.0% of being of class “running” (the positive class). Alternatively, 52% of being of class “walking”

- For the sample *(2.7, 3.6)* instead of 0.39, we have:

  probability = 1/(1 + np.exp(-0.39)) = 59.6%

  Interpretation: the sample *(2.7, 3.6)* has a probability of 59.6% of being of class “running” (the positive class). Alternatively, 40.6% of being of class “walking”. The probability is a confidence score!

Neural Networks

- What if we have more than two classes?

- Consider the additional class “sitting” in our example

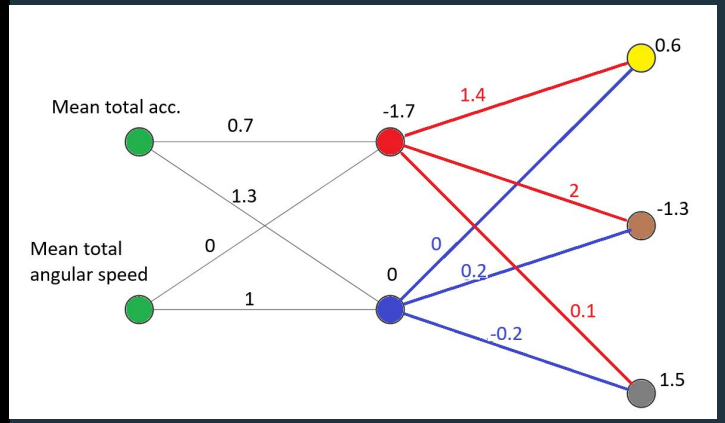
- When we have more than two classes, the **number of output nodes** are equal to the **number of classes**.

Mean Total Acceleration	Mean Total Angular Speed	Class
2.4	0.7	walking
3.8	1.5	running
3.2	1.8	walking
1.1	2.7	walking
4.1	2.0	walking
4.7	3.1	running
2.7	3.6	running
0.5	4.2	running
0.9	4.4	walking
4.3	3.9	running
0.2	0.3	sitting
0.1	0.4	sitting
0.3	0.2	sitting
0.2	0.1	sitting
0.1	0.3	sitting

Neural Networks

- A neural network that works well classifying these samples is shown below.
- **Note**: we are using colors only for the purpose of illustrating better.
- Now let's try one sample of each class.

Mean Total Acceleration	Mean Total Angular Speed	Class
2.4	0.7	walking
3.8	1.5	running
3.2	1.8	walking
1.1	2.7	walking
4.1	2.0	walking
4.7	3.1	running
2.7	3.6	running
0.5	4.2	running
0.9	4.4	walking
4.3	3.9	running
0.2	0.3	sitting
0.1	0.4	sitting
0.3	0.2	sitting
0.2	0.1	sitting
0.1	0.3	sitting

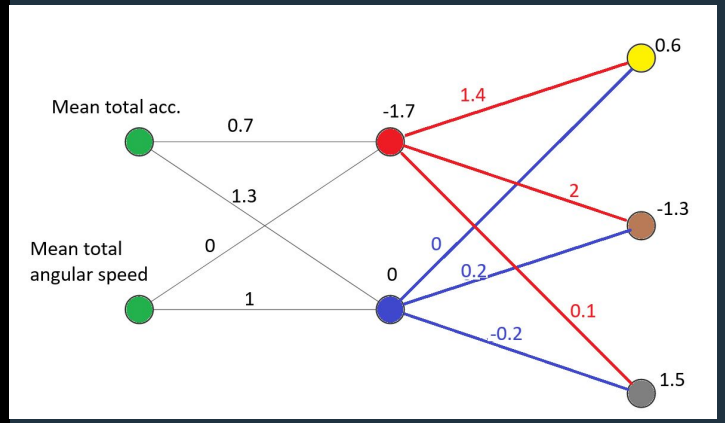


Neural Networks

- Each neuron is associated with a class

- Yellow: walking
- Brown: running
- Gray: sitting

Mean Total Acceleration	Mean Total Angular Speed	Class
2.4	0.7	walking
3.8	1.5	running
3.2	1.8	walking
1.1	2.7	walking
4.1	2.0	walking
4.7	3.1	running
2.7	3.6	running
0.5	4.2	running
0.9	4.4	walking
4.3	3.9	running
0.2	0.3	sitting
0.1	0.4	sitting
0.3	0.2	sitting
0.2	0.1	sitting
0.1	0.3	sitting



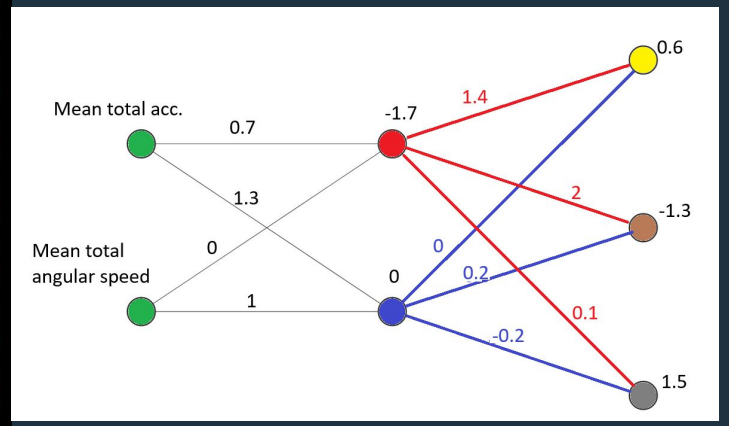
Neural Networks

- Sample: *(3.2, 1.8) walking*
- Red neuron gives: 3.2*(0.7) + 1.8*(0) + (-1.7) = 0.54

  weight weight bias
- Blue neuron gives: 3.2*(1.3) + 1.8*(1) + 0 = 5.96

  weight weight bias

Mean Total Acceleration	Mean Total Angular Speed	Class
2.4	0.7	walking
3.8	1.5	running
3.2	1.8	walking
1.1	2.7	walking
4.1	2.0	walking
4.7	3.1	running
2.7	3.6	running
0.5	4.2	running
0.9	4.4	walking
4.3	3.9	running
0.2	0.3	sitting
0.1	0.4	sitting
0.3	0.2	sitting
0.2	0.1	sitting
0.1	0.3	sitting



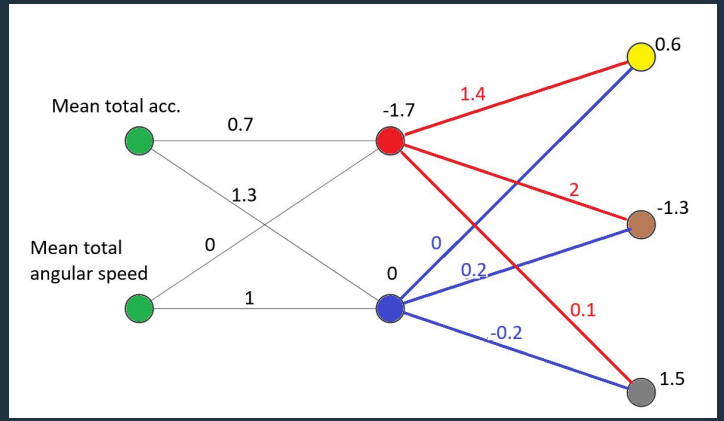
Neural Networks

- Sample: *(3.2, 1.8) walking*
- Yellow neuron gives: 0.54*(1.4) + 5.96*(0) + (0.6) = **1.35**

  weight weight bias
- Brown neuron gives: 0.54*(2) + 5.96*(0.2) + (-1.3) = 0.97

  weight weight bias
- Gray neuron gives: 0.54*(0.1) + 5.96*(-0.2) + (1.5) = 0.36

  weight weight bias



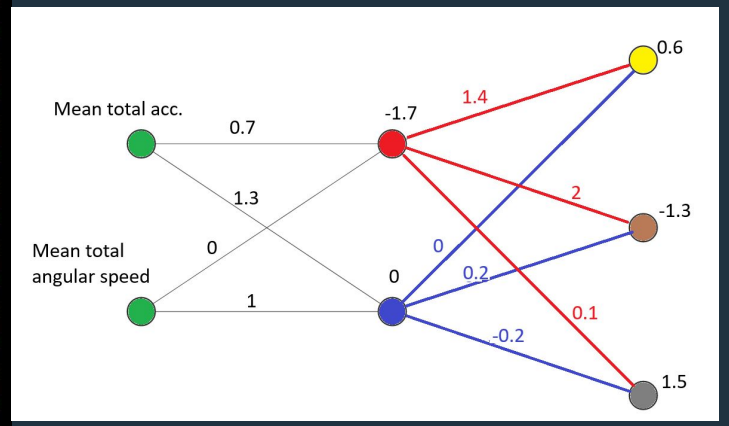
Neural Networks

- Sample: *(4.7, 3.1) running*
- Red neuron gives: 4.7*(0.7) + 3.1*(0) + (-1.7) = 1.59

  weight weight bias
- Blue neuron gives: 4.7*(1.3) + 3.1*(1) + 0 = 9.21

  weight weight bias

Mean Total Acceleration	Mean Total Angular Speed	Class
2.4	0.7	walking
3.8	1.5	running
3.2	1.8	walking
1.1	2.7	walking
4.1	2.0	walking
4.7	3.1	running
2.7	3.6	running
0.5	4.2	running
0.9	4.4	walking
4.3	3.9	running
0.2	0.3	sitting
0.1	0.4	sitting
0.3	0.2	sitting
0.2	0.1	sitting
0.1	0.3	sitting



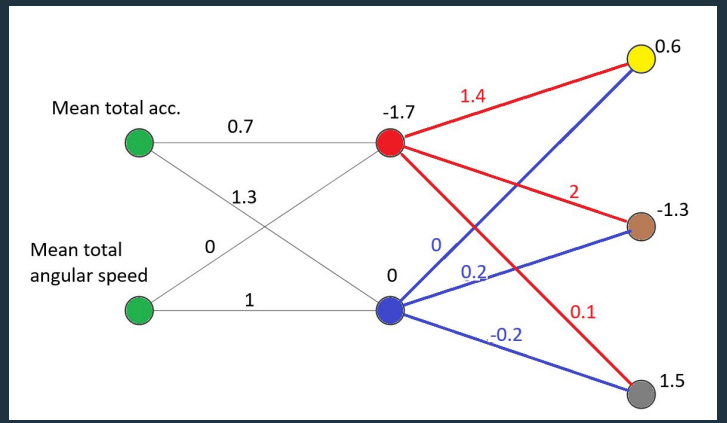
Neural Networks

- Sample: *(4.7, 3.1) running*
- Yellow neuron gives: 1.59*(1.4) + 9.21*(0) + (0.6) = 2.82

  weight weight bias
- Brown neuron gives: 1.59*(2) + 9.21*(0.2) + (-1.3) = 3.72

  weight weight bias
- Gray neuron gives: 1.59*(0.1) + 9.21*(-0.2) + (1.5) = -0.18

  weight weight bias



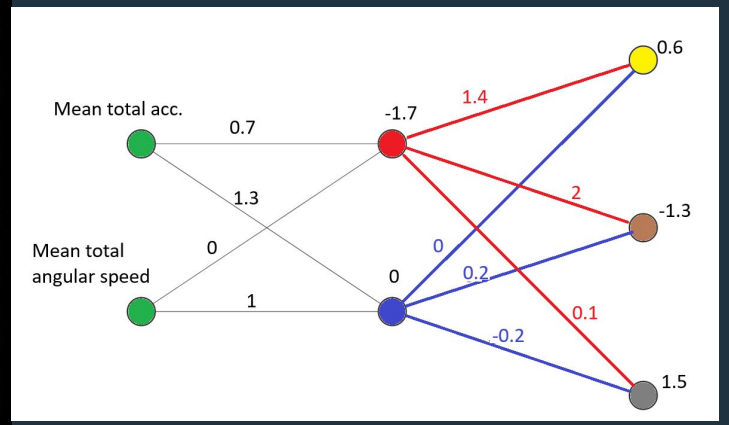
Neural Networks

- Sample: *(0.2, 0.1) sitting*
- Red neuron gives: 0.2*(0.7) + 0.1*(0) + (-1.7) = -1.56

  weight weight bias
- Blue neuron gives: 0.2*(1.3) + 0.1*(1) + 0 = 0.36

  weight weight bias

Mean Total Acceleration	Mean Total Angular Speed	Class
2.4	0.7	walking
3.8	1.5	running
3.2	1.8	walking
1.1	2.7	walking
4.1	2.0	walking
4.7	3.1	running
2.7	3.6	running
0.5	4.2	running
0.9	4.4	walking
4.3	3.9	running
0.2	0.3	sitting
0.1	0.4	sitting
0.3	0.2	sitting
0.2	0.1	sitting
0.1	0.3	sitting



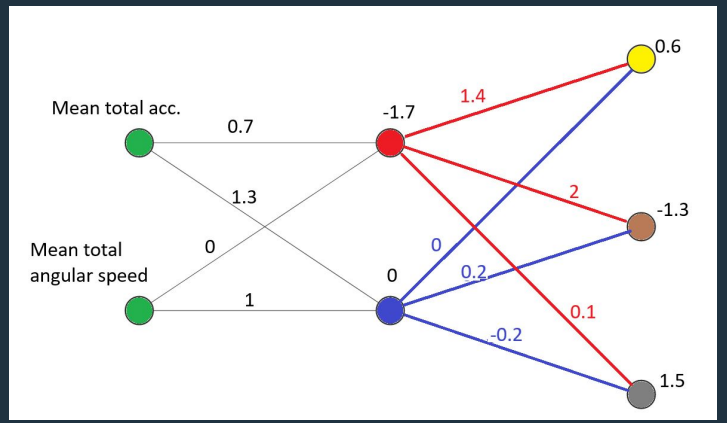
Neural Networks

- Sample: *(0.2, 0.1) sitting*
- Yellow neuron gives: -1.56*(1.4) + 0.36*(0) + (0.6) = -1.584

  weight weight bias
- Brown neuron gives: -1.56*(2) + 0.36*(0.2) + (-1.3) = -4.34

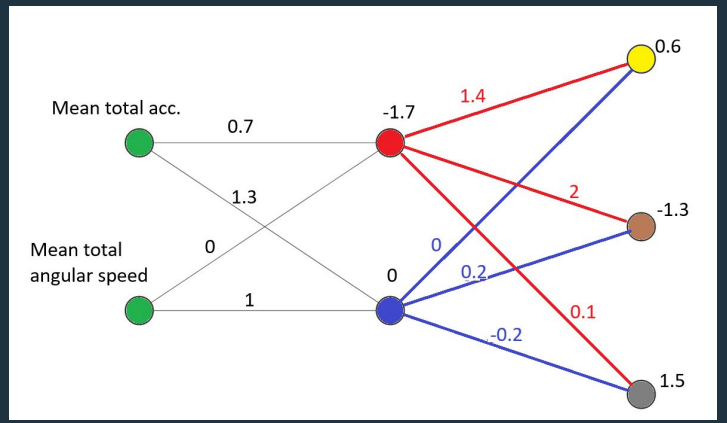
  weight weight bias
- Gray neuron gives: -1.56*(0.1) + 0.36*(-0.2) + (1.5) = 1.27

  weight weight bias



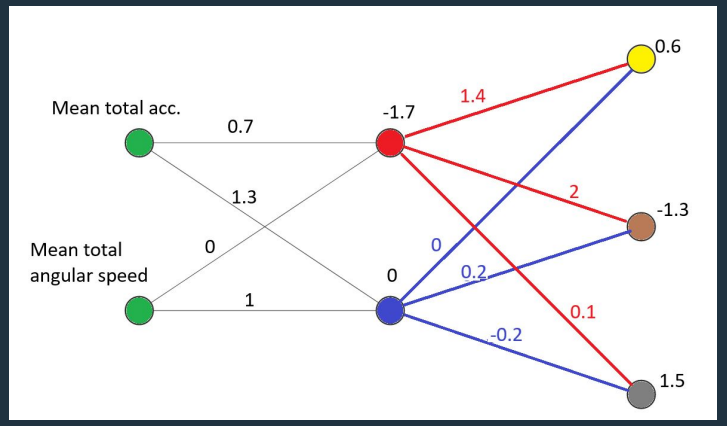
Neural Networks

- How do we go from the scores to probabilities: **softmax function**
- **Softmax function** is an expansion of the sigmoid function
- probability of a certain class = exp(*output of the class*)/(exp(*class 1*) + exp(*class 2*) + … + exp(*class n*))



Neural Networks

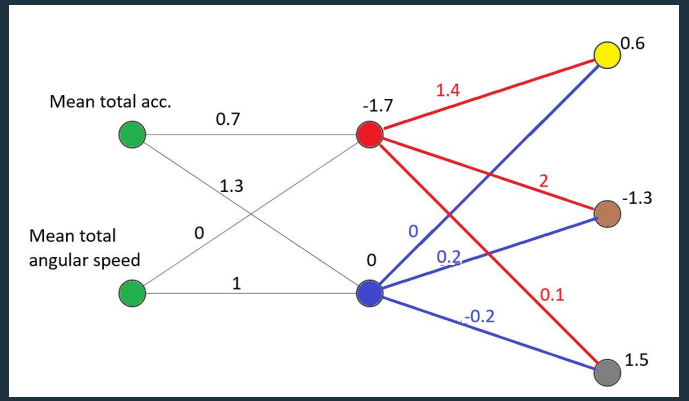
- probability of a certain class = exp(*output of the class*)/(exp(*class 1*) + exp(*class 2*) + … + exp(*class n*))
- Sample: *(0.2, 0.1) sitting*. First, we calculate:
- Yellow neuron: -1.584 and exp(-1.584) = 0.205
- Brown neuron: -4.34 and exp(-4.34) = 0.013
- Gray neuron: 1.27 and exp(1.27) = 3.560



Neural Networks

- probability of a certain class = exp(*output of the class*)/(exp(*class 1*) + exp(*class 2*) + … + exp(*class n*))
- Sample: *(0.2, 0.1) sitting*. First, we calculate:
- Yellow neuron: -1.584 and exp(-1.584) = 0.205
- Brown neuron: -4.34 and exp(-4.34) = 0.013
- Gray neuron: 1.27 and exp(1.27) = 3.560

- Yellow neuron (walking): 0.205/(0.205 + 0.013 + 3.560) = 0.0542 = 5.42%
- Brown neuron (running): 0.013/(0.205 + 0.013 + 3.560) = 0.00344 = 0.344%
- Gray neuron (sitting): 3.560/(0.205 + 0.013 + 3.560) = **0.9422 = 94.22%**

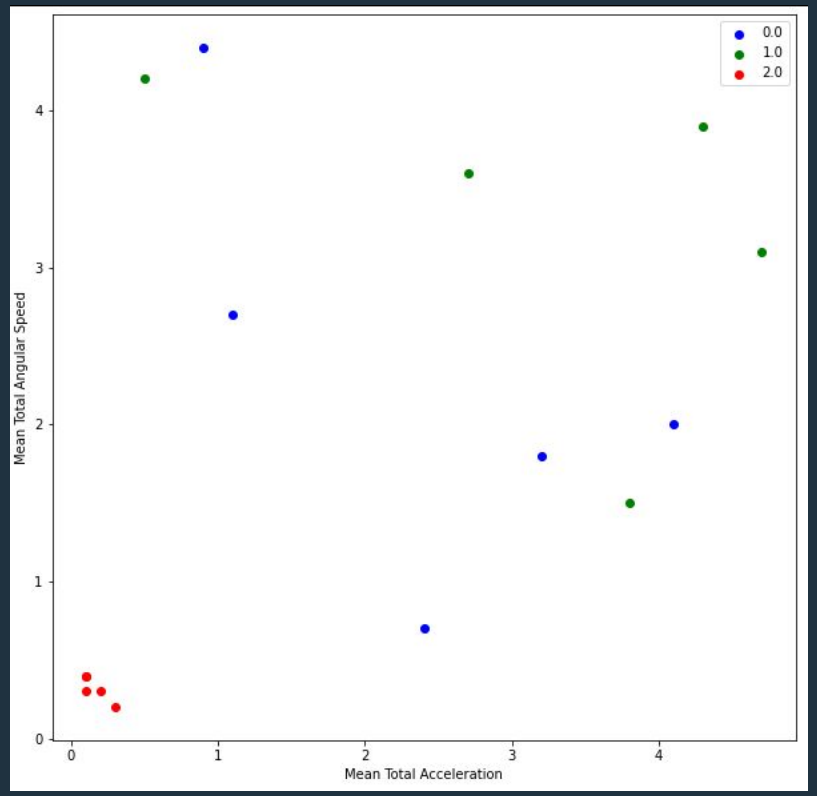


Neural Networks

- The neural network examples were pretty simple.

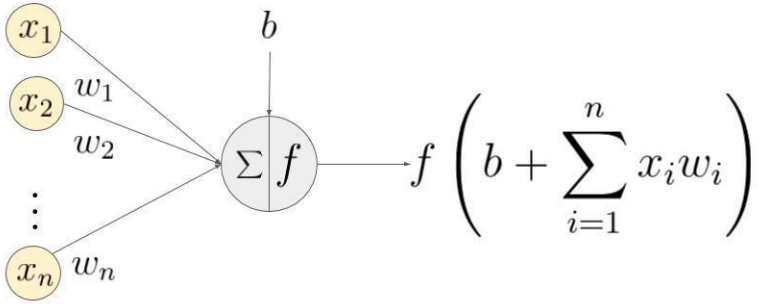
- We only had linear operations (sum and multiplication), few neurons (2) and only one hidden layer.

- In more complex cases (more classes and non-linearly separable data), we need more hidden layers, neurons, and nonlinear operations.



Neural Networks

- How can neural networks deal with non-linearly separable data?
- Utilizing a so-called **activation function**
- The activation function is a non-linear function applied right after the bias

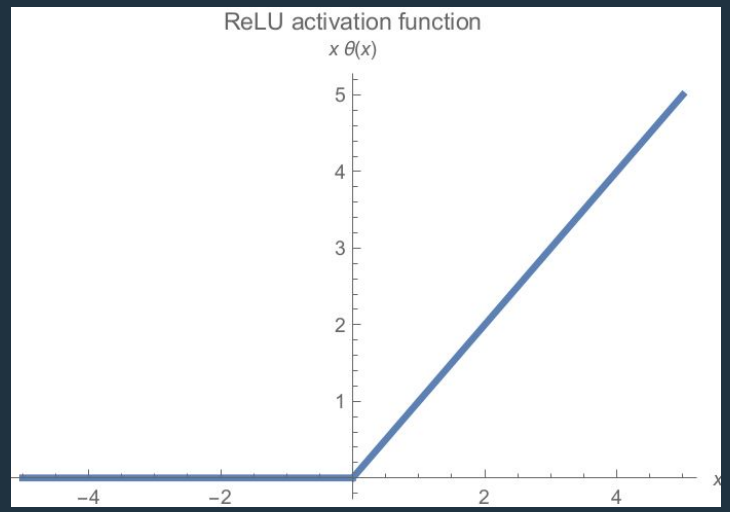


An example of a neuron showing the input (x_1 - x_n), their corresponding weights (w_1 - w_n), a bias (b) and the activation function f applied to the weighted sum of the inputs.

Neural Networks

- A very simple and widely used activation function is the Rectified Linear Unit function or **ReLU**

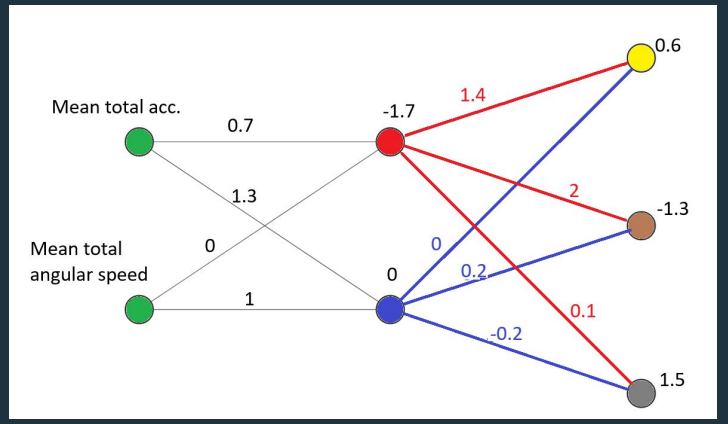
- ReLU(x) = max(0, x)
- ReLU(-1) = max(0, -1) = 0
- ReLU(2) = max(0, 2) = 2



Neural Networks

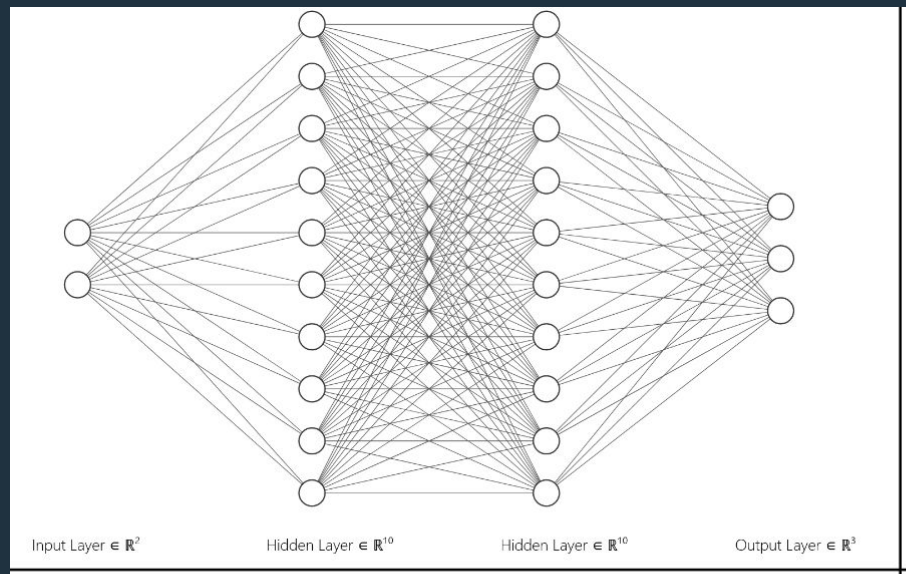
- How do we build a neural network? That is, how do we find the weights and biases from the data + labels?

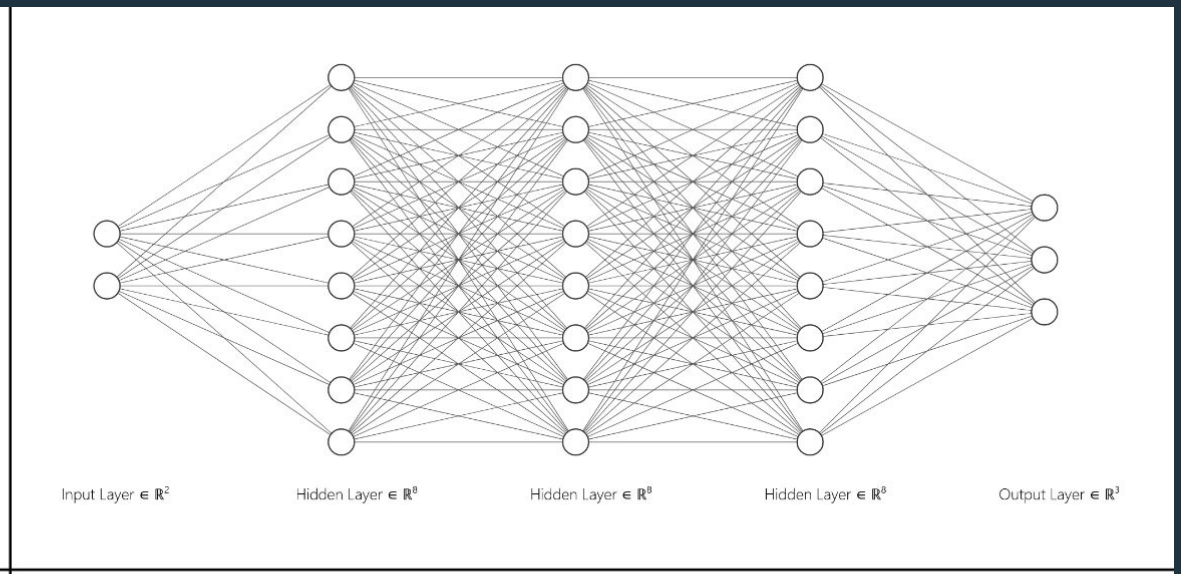
- Let’s go step by step!

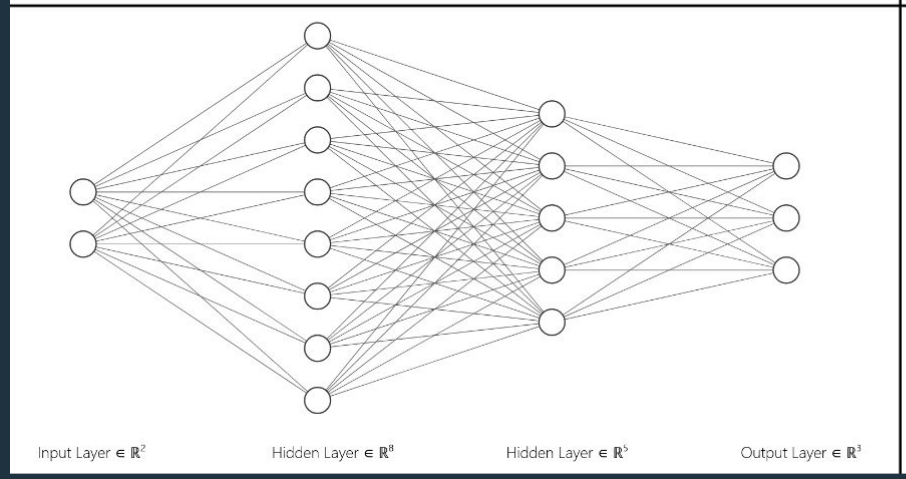


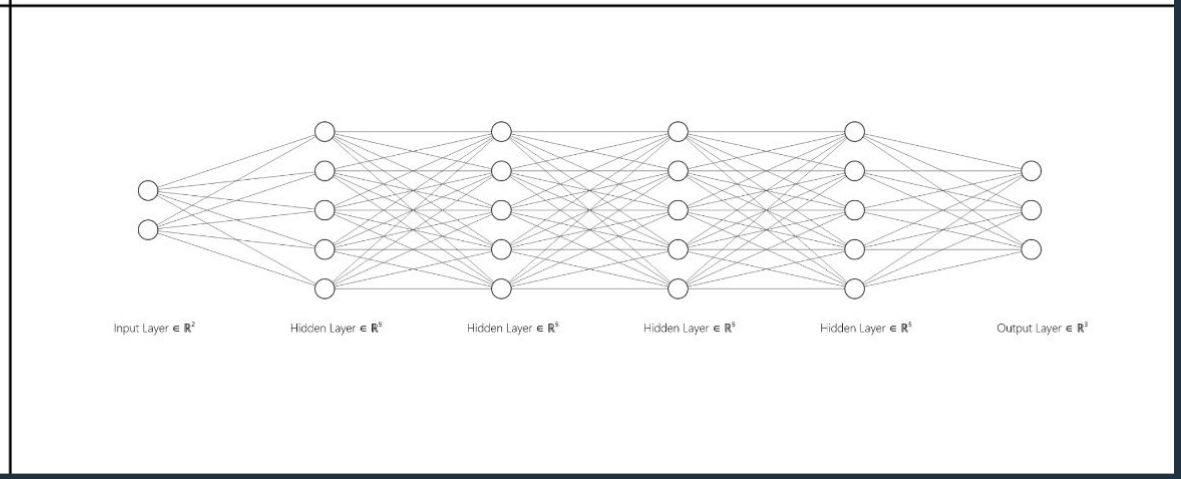
Neural Networks

- Consider we have input size of 2 and 3 classes.
- **STEP 1.** define the numbers of hidden layers and their neurons



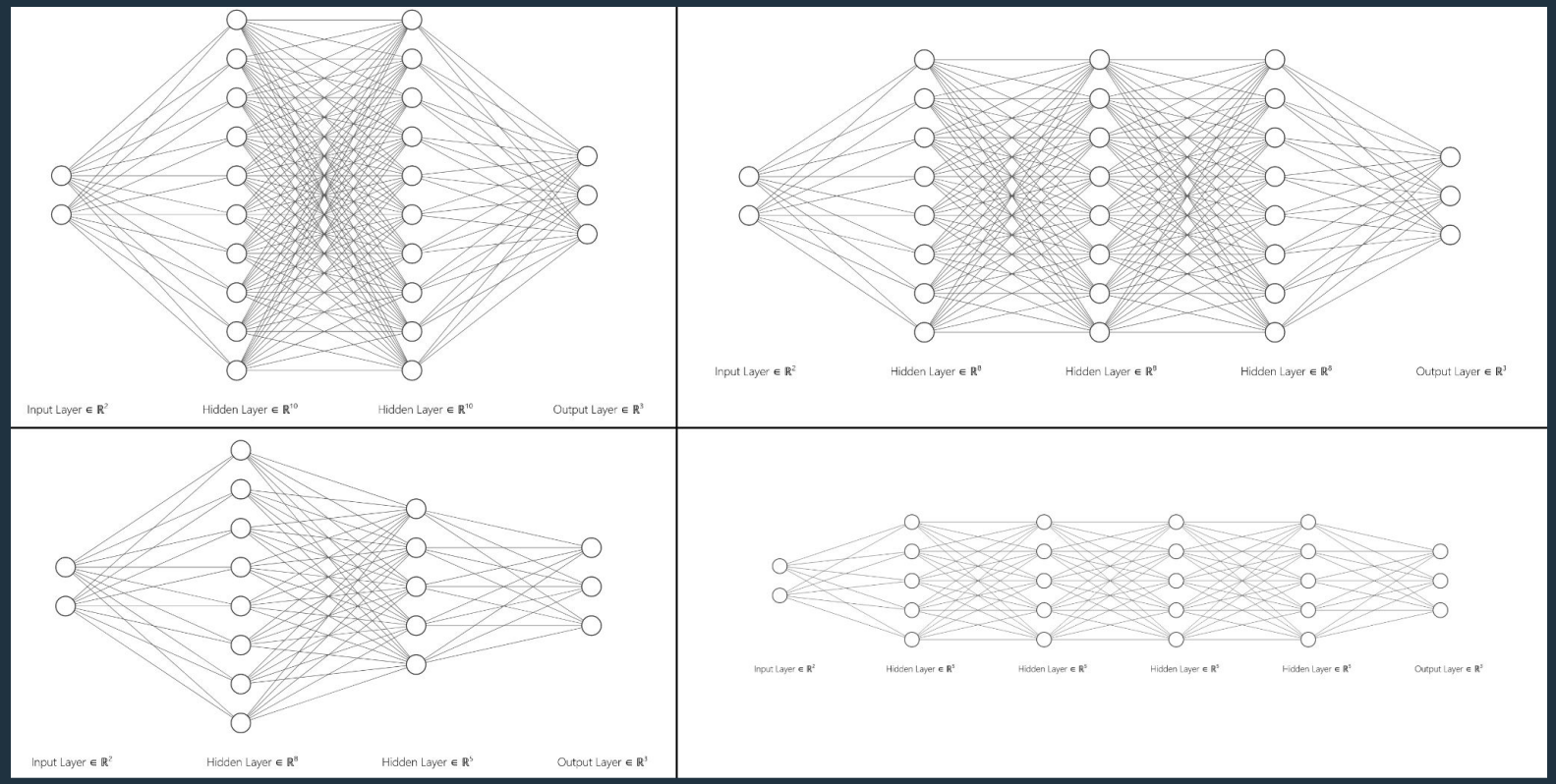






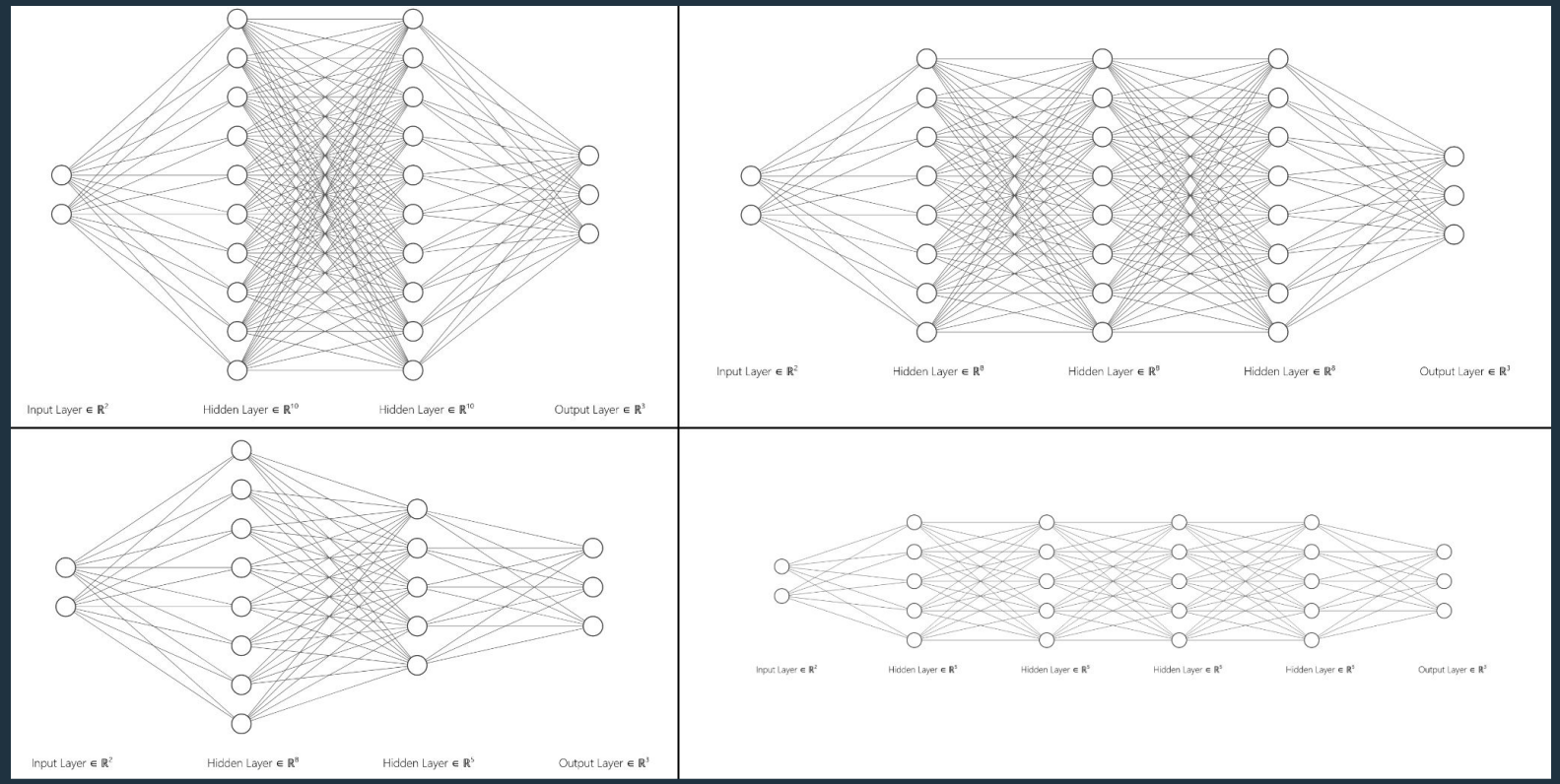
Neural Networks

- There isn't a specific guideline for choosing these **hyper-parameters**
- **High number of layers and neurons** → more complex neural networks → possible to overfit
- **Low number of layers and neurons** → less complex neural networks → possible to underfit



Neural Networks

- **Trial and error**: try certain complexity, train the neural network, and see how the neural network performs on the validation set. Choose the complexity that gives the best performance on the validation set.



Neural Networks

- **STEP 2**: initializing weights and biases

- The **parameters** (weights and biases) can be initialized all to zero, for example.

- Or, better, they can be initialized according to certain rules (for example, the Xavier Glorot's initialization). But we won't go into details. Scikit-learn uses Glorot's method for initializing parameters.

- **Terminology note:** we often denote the weights and biases of the neural network as just weights.

Neural Networks

- The Adam optimizer is the most used one. The details on how it operates require university-level calculus and are beyond the scope of this course.

- The loss function is a measure on how good the weights and biases of the neural network are. The Adam optimizer utilizes the loss function to update the weights and biases.

- In the optimization, we also have something called **learning rate**.

Neural Networks

- But, if the parameters are initialized randomly, won't the neural network perform bad?

- Yes, but the parameters will undergo a process named **optimization** (i.e., learning or training) by an **optimizer.**

- The goal of the optimization is to improve the weights (and biases) gradually with the help of a function named **loss function**.

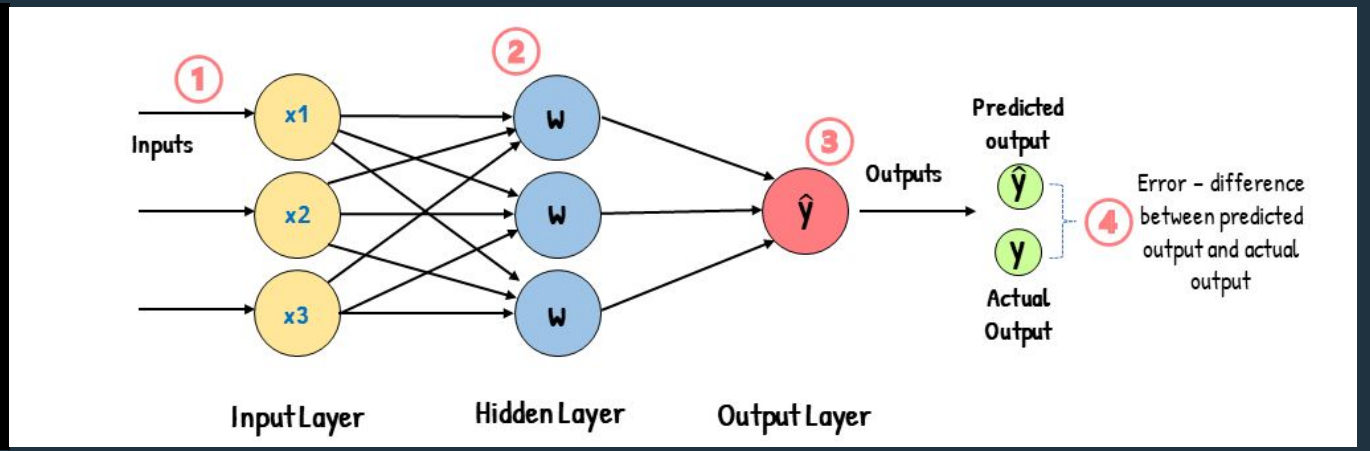
Neural Networks

- Let’s learn a little bit more about the training.

- The training starts by dividing the training set into batches. In the image below, each batch (in blue) has two samples.

- We put each sample of the batch through the neural network and find the predictions (output).

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running



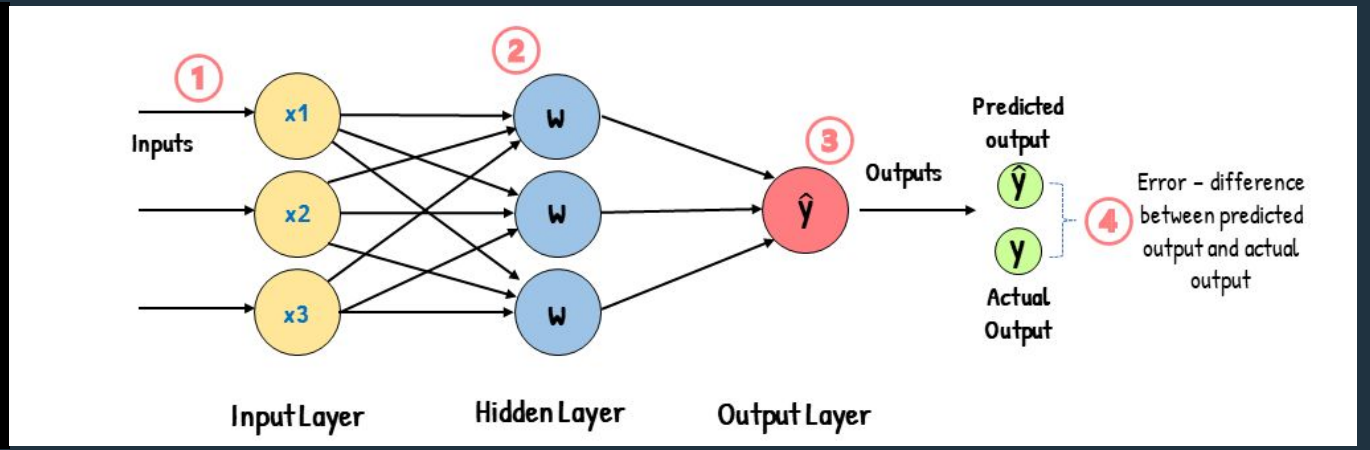
Neural Networks

- An error is calculated. This error tells how much the output differs from the desired output.

- Suppose we give the sample *(3.2, 1.8)* and the neural network predicts it “walking” with 5% probability. The error is big because we want the neural network to predict it with 100% probability.

- The error is calculated with the **loss function**.

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running



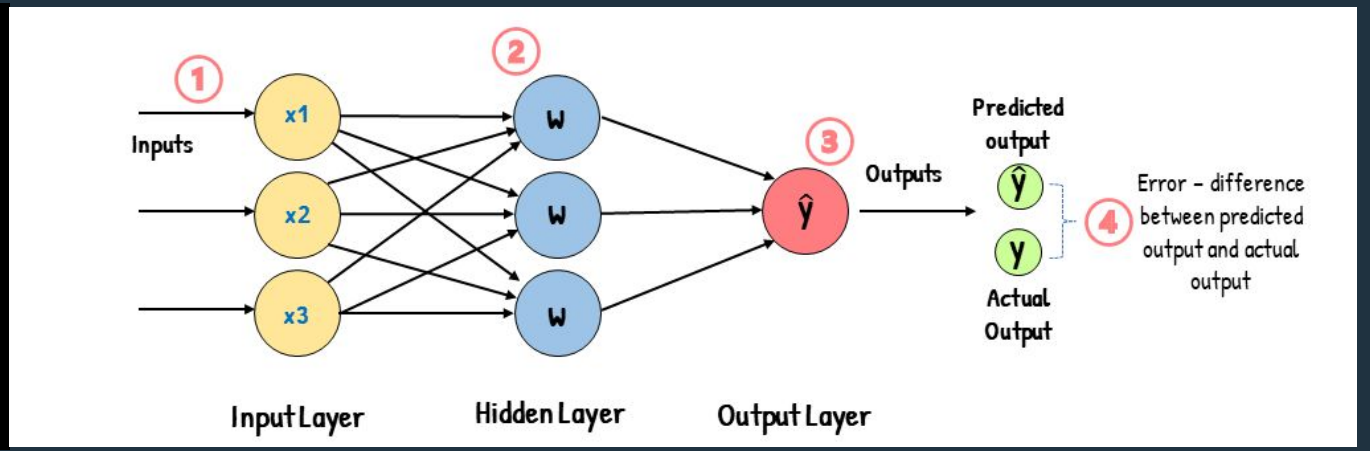
Neural Networks

- The optimizer calculates how much the weights should be adjusted to reduce the error.

- The weights are updated and a new batch is processed.

- We call an **epoch** once we complete this procedure with all the batches.

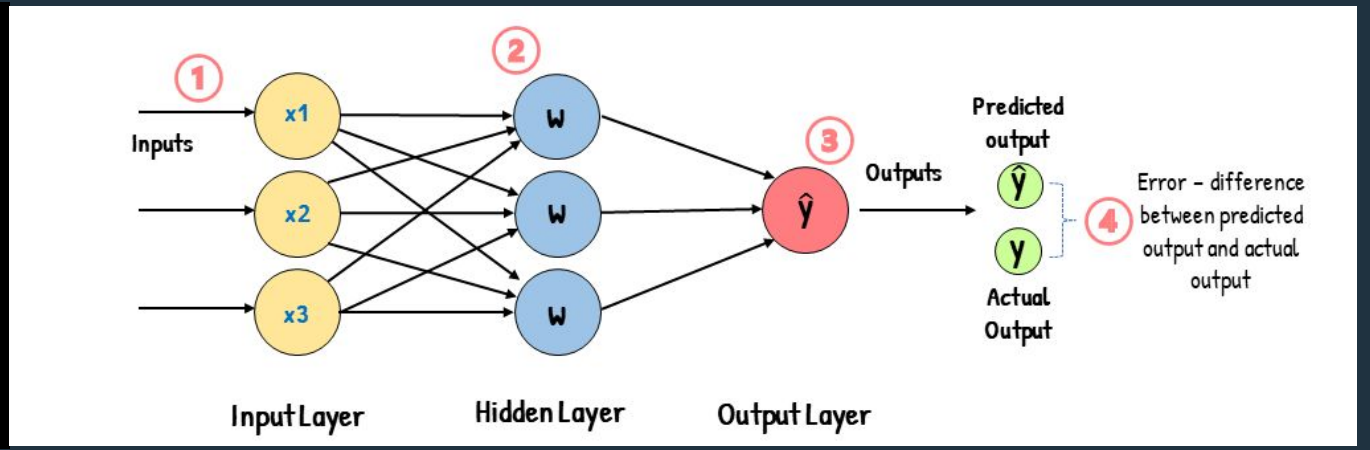
Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running



Neural Networks

- The training proceeds by using the batches all over again.
- We will talk later about when the training stops.
- **Question**: why don't we use the entire dataset as a batch? Why do we have to split?

Mean Total Acceleration	Mean Total Angular Speed	Class
3.2	1.8	walking
2.7	3.6	running
4.1	2.0	walking
0.9	4.4	walking
3.8	1.5	running
1.1	2.7	walking
4.3	3.9	running
2.4	0.7	walking
0.5	4.2	running
4.7	3.1	running



Neural Networks

- The optimizer has something called **learning rate**. The learning is a positive number (usually small like 0.001)

- It determines how much the weights are adjusted during the optimization.

- So, should we set the learning rate to a large number to learn as fast as possible?

- No. The reason is mathematical. But imagine learning to ride a bike. Rushing will lead you to commit mistakes that can cause injuries and hinder you from learning.

Neural Networks

- A very small learning rate will slow down the process of learning.

- Using the analogy of learning to ride a bike. Being too carefully will make the process of learning too slow.

- Usually a learning rate in the range from 0.0001 to 0.01 will work well.

Neural Networks

- **STEP 1.** We define the number of layers and their neurons.
- **STEP 2.** We initialize the weights and biases.
- **STEP 3.** We choose an optimizer, loss function, and learning rate. These three things will work together to gradually improve the weights.
- **STEP 4.** We start the learning (training or optimization).
- **STEP 5.** The training ends according to certain criteria

Neural Networks

- When to stop the training? Ideas?

Neural Networks

- When to stop the training?

- **Maximum Epochs:** Set a maximum number of training epochs.
- **Loss Convergence:** Monitor the training and validation loss (error) during training. Training is typically stopped when the loss reaches a plateau or starts increasing on the validation set.
- **Validation Performance**: Monitor accuracy on the validation set at the end of each epoch. Stop training when it no longer improves or starts degrading.

Neural Networks

- Sounds like a lot? No worries. Scikit-learn implements all this. We just have to use the already-made functions.

- But that is not all, prior to the training (learning or optimization), we need to do something to the data. Look at the table below and try to notice something that distinguishes “heart rate”, “acceleration”, and “angular speed”.

Heart Rate	Acceleration	Angular Speed	Class
65	0.5	1.0	Sitting
75	0.6	1.2	Sitting
80	0.7	1.3	Sitting
100	2.5	2.5	Running
110	3.0	2.8	Running
120	2.7	2.9	Running
70	0.4	1.1	Sitting
130	3.2	3.0	Running
140	3.5	3.2	Running
60	0.3	0.9	Sitting

Neural Networks

- The heart rate readings (in BPM) are much larger than those of acceleration and angular speed.

- Such a vastly different scale means that the heart rate will influence a lot more in the output. (Remember, the neural network does multiplications and sums from the inputs)

Heart Rate	Acceleration	Angular Speed	Class
65	0.5	1.0	Sitting
75	0.6	1.2	Sitting
80	0.7	1.3	Sitting
100	2.5	2.5	Running
110	3.0	2.8	Running
120	2.7	2.9	Running
70	0.4	1.1	Sitting
130	3.2	3.0	Running
140	3.5	3.2	Running
60	0.3	0.9	Sitting

Neural Networks

- **Vastly different scales** -> slower training and lower performance.
- Data normalization aims at solving this.
- Ideas on how we can normalize the data?

Heart Rate	Acceleration	Angular Speed	Class
65	0.5	1.0	Sitting
75	0.6	1.2	Sitting
80	0.7	1.3	Sitting
100	2.5	2.5	Running
110	3.0	2.8	Running
120	2.7	2.9	Running
70	0.4	1.1	Sitting
130	3.2	3.0	Running
140	3.5	3.2	Running
60	0.3	0.9	Sitting

Neural Networks

- **Min-max normalization**

1. Find the min and max of each feature
2. new value = (old value - min of the feature)/(max of the feature - min of the feature)

Min: 60, 0.3, 0.9

Max: 140, 3.5, 3.2

Heart Rate	Acceleration	Angular Speed	Class
65	0.5	1.0	Sitting
75	0.6	1.2	Sitting
80	0.7	1.3	Sitting
100	2.5	2.5	Running
110	3.0	2.8	Running
120	2.7	2.9	Running
70	0.4	1.1	Sitting
130	3.2	3.0	Running
140	3.5	3.2	Running
60	0.3	0.9	Sitting

Heart Rate (Normalized)	Acceleration (Normalized)	Angular Speed (Normalized)
0.0625	0.0625	0.04347826
0.1875	0.09375	0.13043478
0.25	0.125	0.17391304
0.5	0.6875	0.69565217
0.625	0.84375	0.82608696
0.75	0.75	0.86956522
0.125	0.03125	0.08695652
0.875	0.90625	0.91304348
1.	1.	1.
0.	0.	0.

Neural Networks

- **Standardization**

1. Find the mean and std of each feature
2. new value = (old value - mean of the feature)/(std of the feature)

Mean: 95, 1.74, 1.99

Std: 27.38, 1.27, 0.91

Heart Rate	Acceleration	Angular Speed	Class
65	0.5	1.0	Sitting
75	0.6	1.2	Sitting
80	0.7	1.3	Sitting
100	2.5	2.5	Running
110	3.0	2.8	Running
120	2.7	2.9	Running
70	0.4	1.1	Sitting
130	3.2	3.0	Running
140	3.5	3.2	Running
60	0.3	0.9	Sitting

Heart Rate (Normalized)	Acceleration (Normalized)	Angular Speed (Normalized)
-1.09544512,	-0.97713553,	-1.08738697,
-0.73029674,	-0.89833428,	-0.86771283,
-0.54772256,	-0.81953302,	-0.75787577,
0.18257419,	0.59888952,	0.56016905,
0.54772256,	0.99289578,	0.88968025,
0.91287093,	0.75649202,	0.99951732,
-0.91287093,	-1.05593678,	-0.9775499 ,
1.2780193 ,	1.15049828,	1.10935438,
1.64316767,	1.38690204,	1.32902852,
-1.2780193 ,	-1.13473803,	-1.19722404,

Attention!

- We not only need to normalize the training data, but also the validation and test data!
- Moreover, we use the min, max, std, mean of the training data when normalizing the validation or test data.

Can you tell why?

Heart Rate	Acceleration	Angular Speed	Class
65	0.5	1.0	Sitting
75	0.6	1.2	Sitting
80	0.7	1.3	Sitting
100	2.5	2.5	Running
110	3.0	2.8	Running
120	2.7	2.9	Running
70	0.4	1.1	Sitting
130	3.2	3.0	Running
140	3.5	3.2	Running
60	0.3	0.9	Sitting

Heart Rate (Normalized)	Acceleration (Normalized)	Angular Speed (Normalized)
-1.09544512,	-0.97713553,	-1.08738697,
-0.73029674,	-0.89833428,	-0.86771283,
-0.54772256,	-0.81953302,	-0.75787577,
0.18257419,	0.59888952,	0.56016905,
0.54772256,	0.99289578,	0.88968025,
0.91287093,	0.75649202,	0.99951732,
-0.91287093,	-1.05593678,	-0.9775499 ,
1.2780193 ,	1.15049828,	1.10935438,
1.64316767,	1.38690204,	1.32902852,
-1.2780193 ,	-1.13473803,	-1.19722404,

Neural Networks

- We need to randomly shuffle the training data before starting each epoch.
- Shuffling means altering the order of the data
- This randomization helps prevent the model from learning the order of the data and potentially overfitting to it.

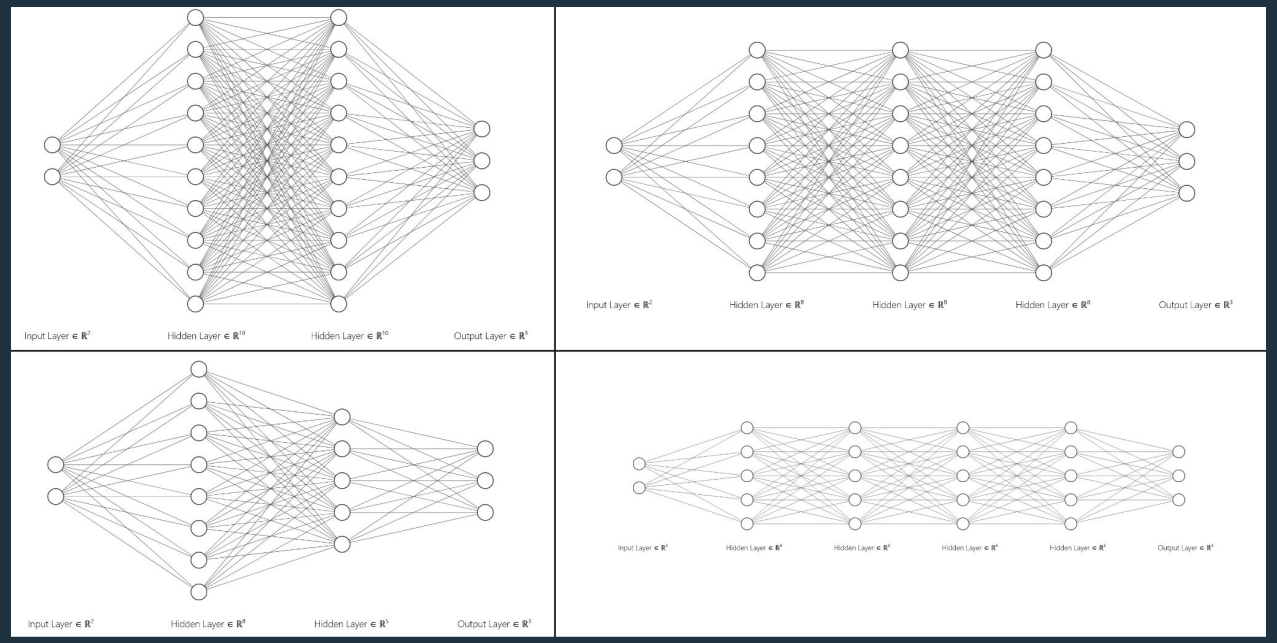
Heart Rate (Normalized)	Acceleration (Normalized)	Angular Speed (Normalized)	Class
-1.09544512,	-0.97713553,	-1.08738697,	Sitting
-0.73029674,	-0.89833428,	-0.86771283,	Sitting
-0.54772256,	-0.81953302,	-0.75787577,	Sitting
0.18257419,	0.59888952,	0.56016905,	Running
0.54772256,	0.99289578,	0.88968025,	Running
0.91287093,	0.75649202,	0.99951732,	Running
-0.91287093,	-1.05593678,	-0.9775499 ,	Sitting
1.2780193 ,	1.15049828,	1.10935438,	Running
1.64316767,	1.38690204,	1.32902852,	Running
-1.2780193 ,	-1.13473803,	-1.19722404,	Sitting

Heart Rate (Normalized)	Acceleration (Normalized)	Angular Speed (Normalized)	Class
0.18257419,	0.59888952,	0.56016905,	Running
-0.54772256,	-0.81953302,	-0.75787577,	Sitting
0.54772256,	0.99289578,	0.88968025,	Running
1.2780193 ,	1.15049828,	1.10935438,	Running
1.64316767,	1.38690204,	1.32902852,	Running
-1.2780193 ,	-1.13473803,	-1.19722404,	Sitting
-1.09544512,	-0.97713553,	-1.08738697,	Sitting
-0.73029674,	-0.89833428,	-0.86771283,	Sitting
0.91287093,	0.75649202,	0.99951732,	Running
-0.91287093,	-1.05593678,	-0.9775499 ,	Sitting

Deep Learning

Deep Learning

- What we have seen so far is shallow learning.
- The term "deep learning" was coined in the early 2000s.
- A deep neural network is a neural network with many layers.
- How many layers? there is no universal agreement on when “shallow” learning becomes “deep” learning.



Deep Learning

- But there are other characteristics that separate deep learning from shallow learning. First, let's talk about **feature learning**. Suppose we choose our sliding windows to have 32 time steps and 2 channels (total acceleration and total angular speed).

  Shallow Learning: we extract features from the channels across the entire sliding window. So each sliding window is transformed into 2 values (for example, the mean total acceleration and the mean total angular speed)

  Deep Learning: we do not extract features, but give the whole data of the sliding window at once. That is, 32*2 (64) values. The deep neural network is supposed to learn features from the "raw" data.

Deep Learning

- Deep Learning has permitted a revolution in terms of accuracy (performance).
- So, does this mean we should always use deep learning? No!
    - Deep learning requires **a lot** of data.
    - Deep learning uses **a lot** of computational effort (memory and processing)

Can you explain why?

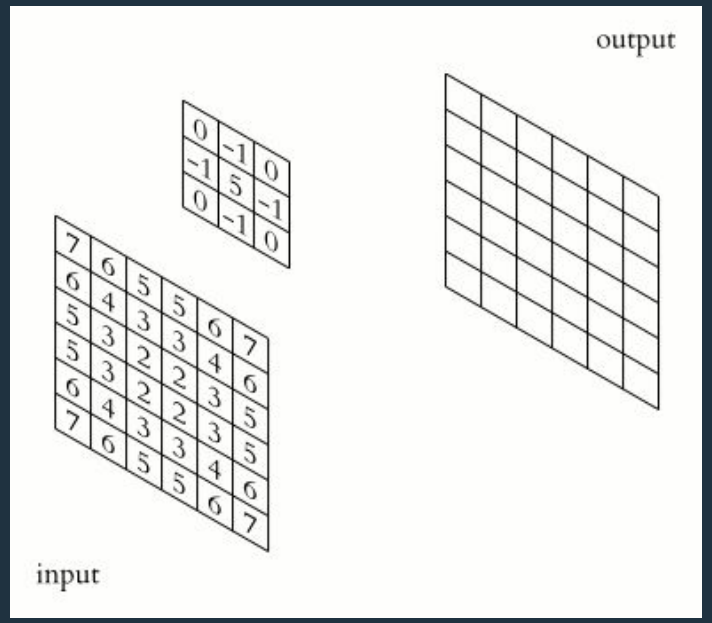
Deep Learning

- Deep Learning has permitted a revolution in terms of accuracy (performance).
- So, does this mean we should always use deep learning? No!
    - Deep learning requires **a lot** of data.
    - Deep learning uses **a lot** of computational effort (memory and processing)

Can you explain why?

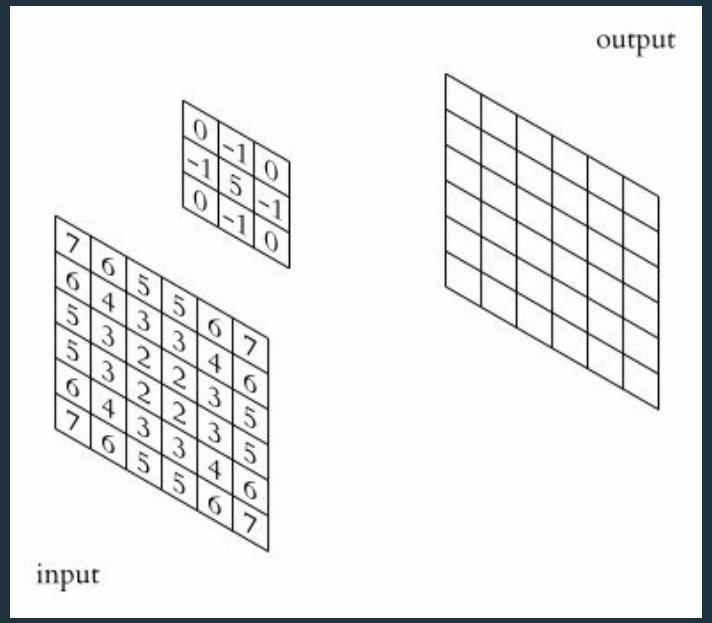
Deep Learning

- What we have seen was only one type of **architecture** named MLP (multi-layer perceptron), which is made of fully-connected (or dense) layers.
- Let’s quickly have a look at another type: **convolutional neural networks**
- The basic operation is the **convolution**



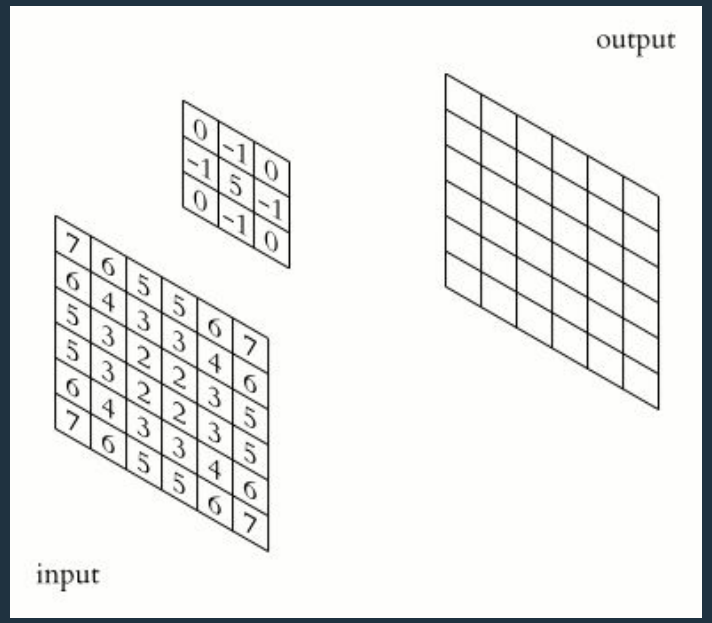
Deep Learning

- Convolutional neural networks are primarily used for images. But let’s have a look considering time-series data.
- 2 dimensions (number of time steps, number of sensor channels)
- The input undergoes a convolution with a **convolutional kernel**

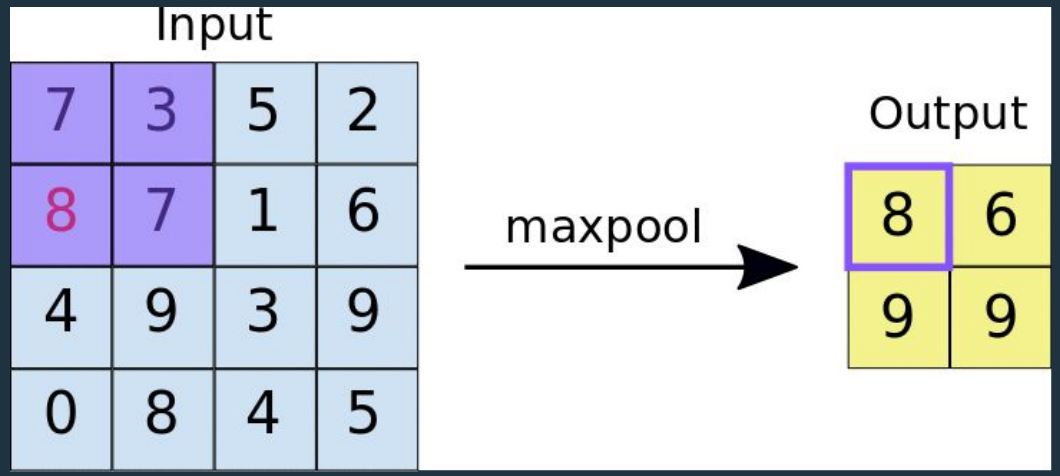


Deep Learning

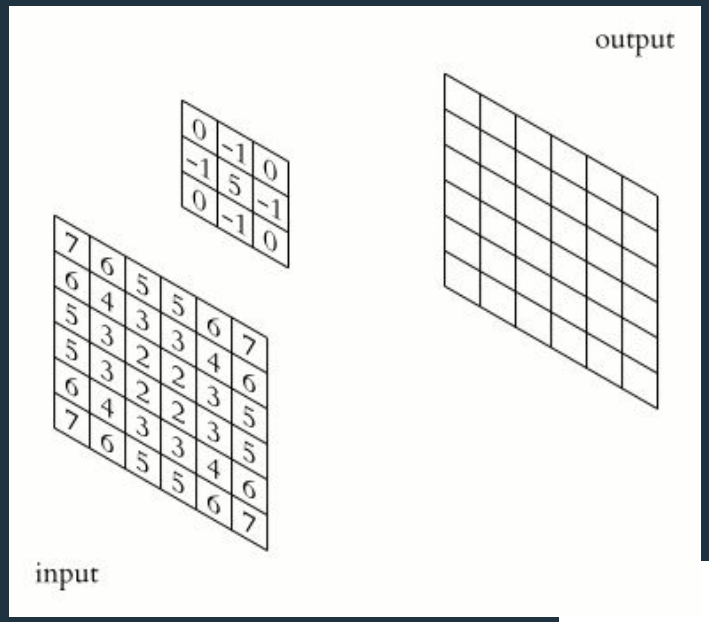
- The convolutional kernel has several weights that are learnable.
- In the animation below, we see a convolutional kernel of size 3 x 3 and stride 1 x 1

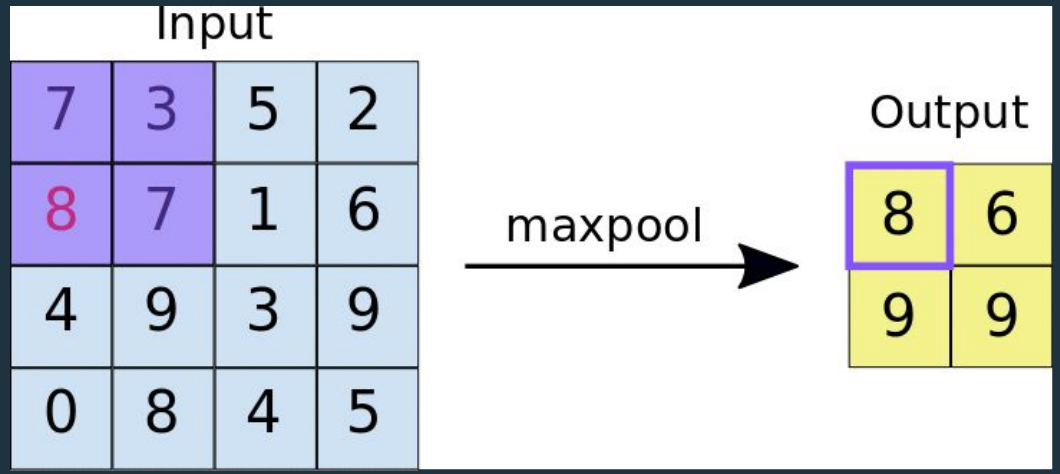


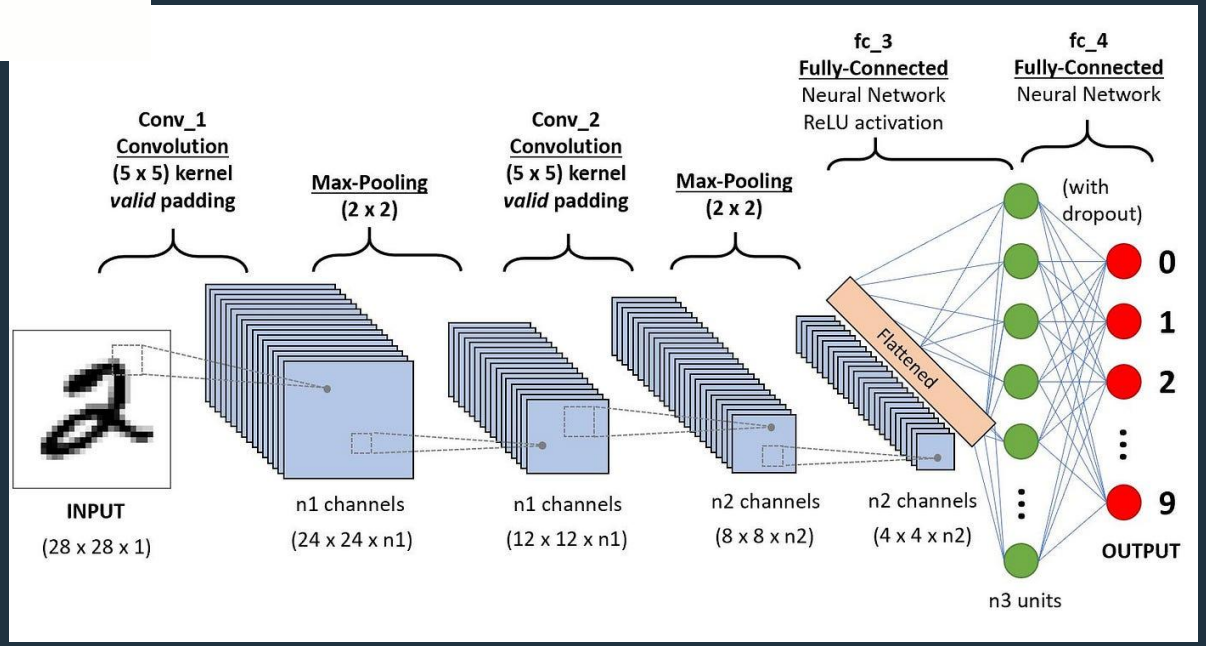
- Typically, after a convolution, ReLU and max pooling are performed
- In the animation below, we see a max pooling operation with kernel size 2 x 2 and stride 2 x 2



Deep Learning

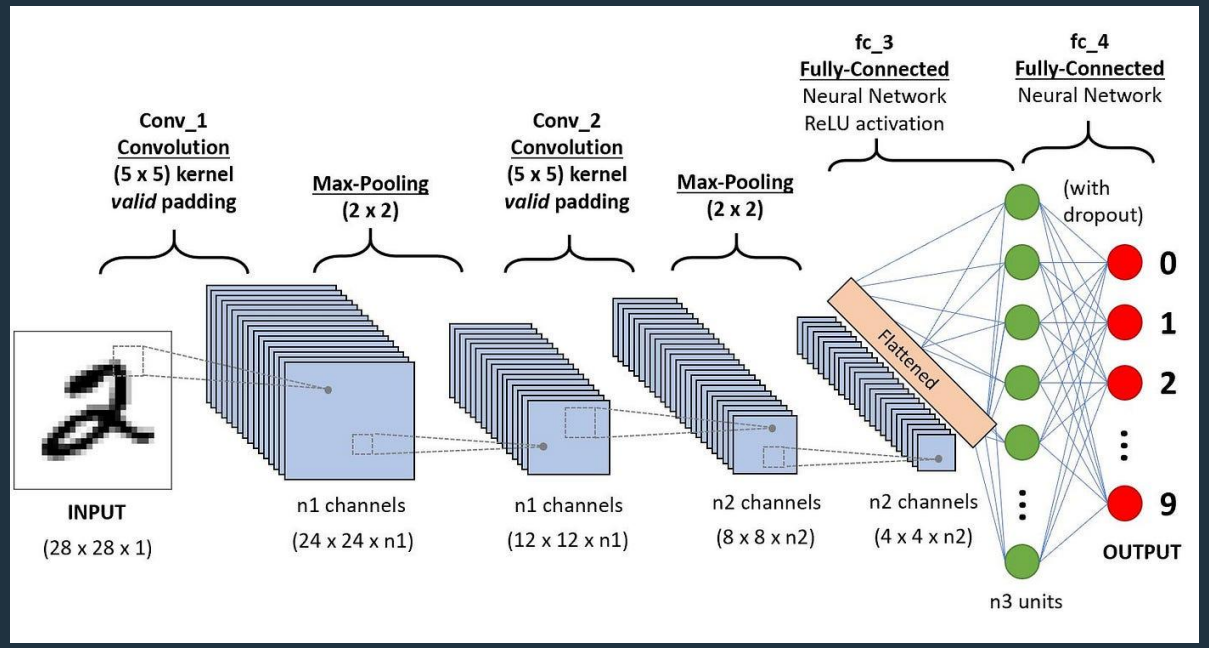






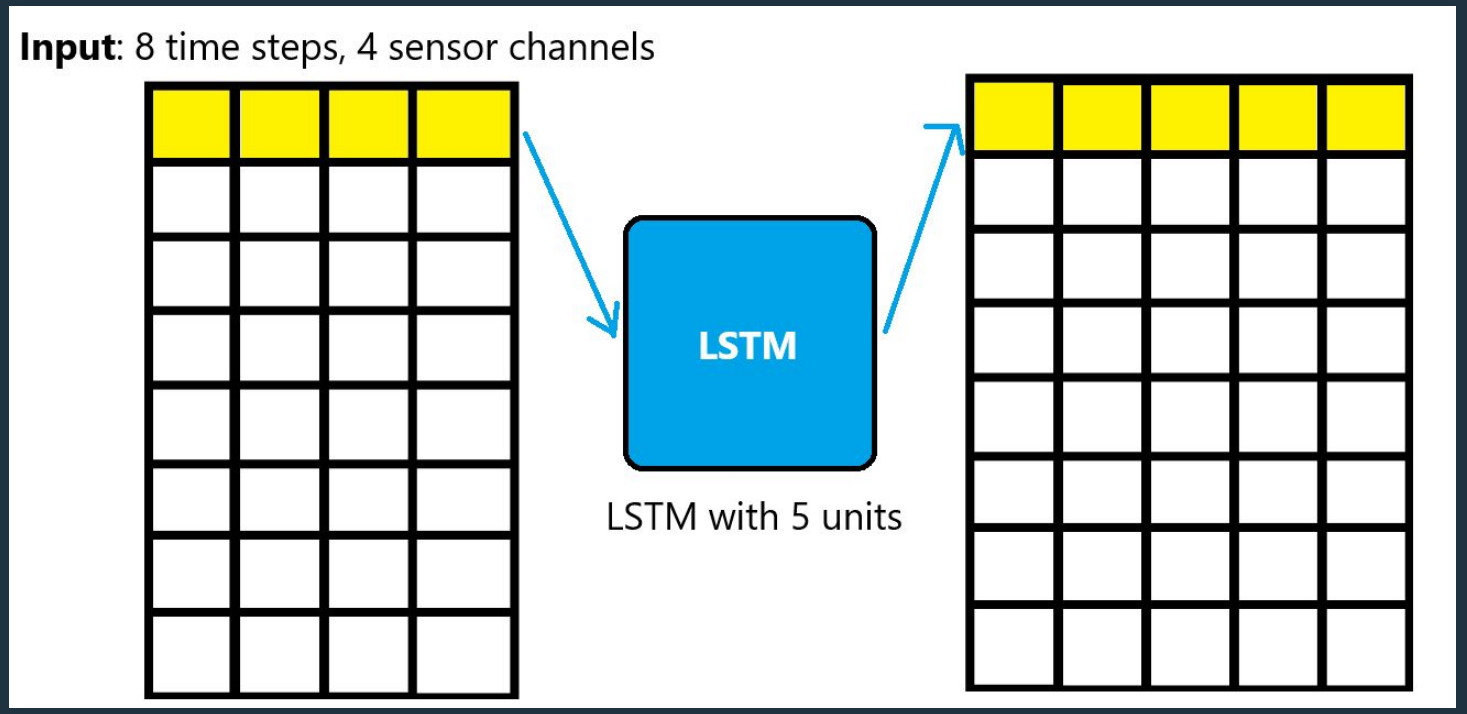
Deep Learning

- Notice that, at the end of the convolutions and max pooling operations, we have a 3D tensor. To get a prediction (classification) from such a 3D tensor, we need to flatten it. Flatten transforms it into 1D and passes it to a fully-connected layer.

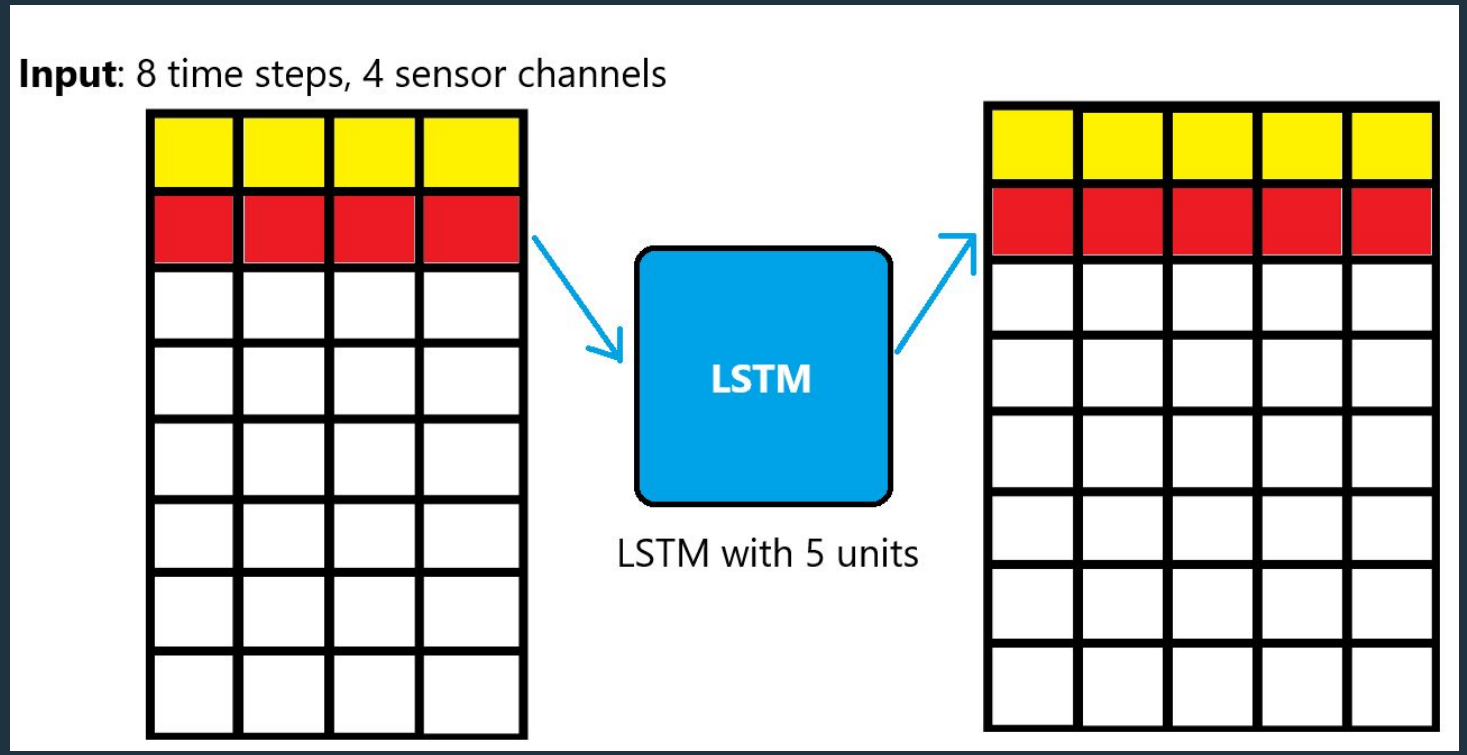


Deep Learning

- LSTMs (Long Short-Term Memory)
- An LSTM layer processes the input (time-series data) step by step. Step *n* depends on all previous steps.

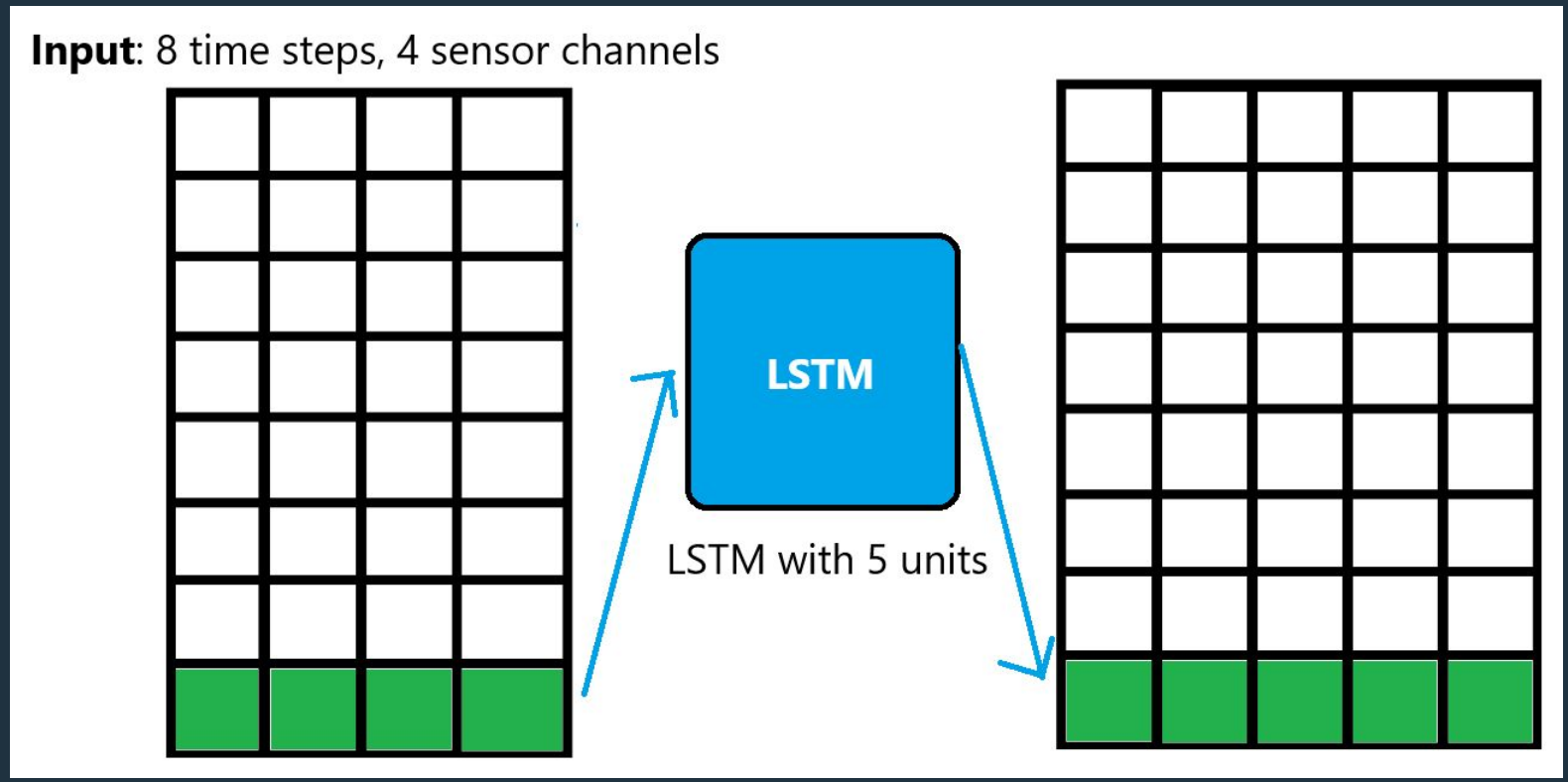


- The output at each step has the dimension of 5 (the chosen number of units for LSTM)
- Eventually, the output after processing all steps with have dimension (8, 5) which is the number of time steps and LSTM units



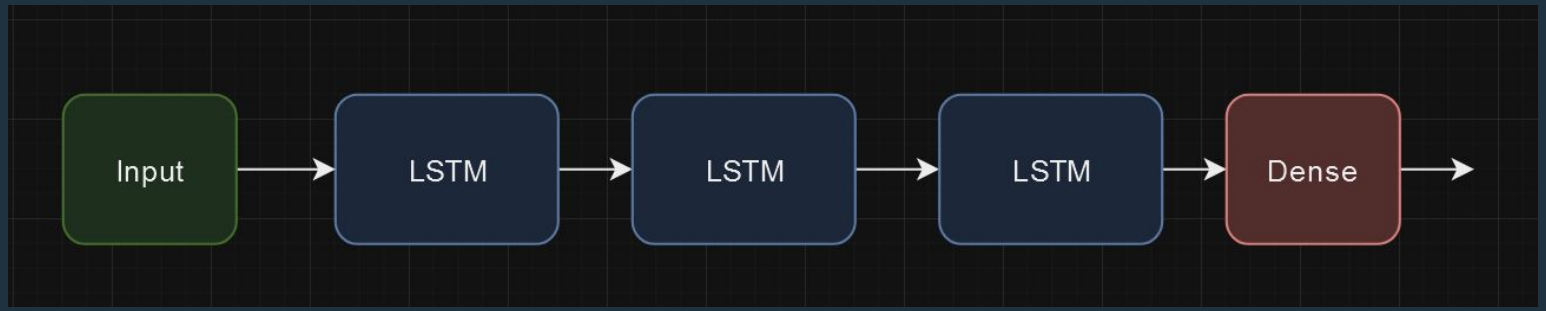
Deep Learning

- From the output, shape (8, 5), we have two options:
  - Keep everything (8, 5)
  - Or consider only the last (1, 5). Do you know why?



Deep Learning

- A neural network with LSTM layers can look like this.
- Can you guess the output of each LSTM layer? If the output contains the entire sequence or just the last time step?



Deep Learning

- A neural network with LSTM layers can look like this.
- Can you guess the output of each LSTM layer? If the output contains the entire sequence or just the last time step?
  - 1st: entire output
  - 2nd: entire output
  - 3rd: entire output (plus flattening) or just the last time step