1. Tensors, layers and autoencoders

Now that you know how to tune your models, it's time to better understand how they work internally and to explore newer network architectures.

2. Accessing Keras layers

Model layers are easily accessible, we just need to call layers on a built model and access the index of the layer we want. From a chosen layer we can print its inputs, outputs, and weights. You can see inputs and outputs are tensors of a given shape built with TensorFlow tensor objects, weights are TensorFlow variable objects, which are just tensors that change their value as the neural network learns the best weights.

3. What are tensors?

Tensors are the main data structures used in deep learning, inputs, outputs, and transformations in neural networks are all represented using tensors and tensor multiplication. A tensor is a multi-dimensional array of numbers. A 2 dimensional tensor is a matrix, a 3 dimensional tensor is an array of matrices.

4. Keras backend

If we import the Keras backend we can build a function that takes in an input tensor from a given layer and returns an output tensor from another or the same layer. Tensorflow is the backend Keras is using in this course, but it could be any other, like Theano. To define the function with our backend K we need to give it a list of inputs and outputs, even if we just want 1 input and 1 output. Then we can use it on a tensor with the same shape as the input layer given during its definition. If the weights of the layers between our input and outputs change the function output for the same input will change as well. We can use this to see how the output of certain layers change as weights are adjusted during training, we will check this in the exercises!

5. Introducing autoencoders

It's time to introduce a new architecture.

6. Autoencoders!

Autoencoders! Autoencoders are models that aim at producing the same inputs as outputs.

7. Autoencoders!

This task alone wouldn't be very useful, but since along the way we decrease the number of neurons, we are effectively making our network learn to compress its inputs into a small set of neurons.

8. Autoencoder use cases

This makes autoencoders useful for things like: Dimensionality reduction, since we can obtain a smaller dimensional space representation of our inputs. De-noising, if trained with clear data and then fed with noisy data they will be able to decode back a good representation of the input data without noise. Anomaly detection, if you train an autoencoder to map inputs to outputs with data but you then pass in strange values, the network will fail at giving accurate output values. And we can measure this as a loss. Many other applications can also benefit from this architecture.

9. Building a simple autoencoder

To make an autoencoder that maps a hundred inputs to a hundred outputs, encoding the inputs into a layer of 4 neurons, we would do the following: Instantiate a sequential model, add a dense layer of 4 neurons with an input_shape of a hundred and end with an output layer of 100 neurons. We use activation sigmoid because we assume that our output can take a value between 0 and 1, we end compiling our model with adam optimizer and binary_crossentropy loss since we used sigmoid.

10. Breaking it into an encoder

Once you've built and trained your autoencoder you might want to encode your inputs. To do this, you just have to build a new model which just uses the first layer of your previously trained autoencoder. This new model predictions returns the 4 numbers given by the 4 neurons of the hidden layer. It's effectively returning a 4 number encoding of each observation in the input dataset.

11. Let's experiment!

Time for you to use this new knowledge to carry out a couple of experiments!