Introduction to Artificial Intelligence

Exercise Sheet - Artificial Neural Network Architectures for Different Modalities

In this class, we will implement our own artificial neural network library. The library can be used to build artificial neural networks involving a variety of layers, such as dense, convolution and pooling. The models that can be created will be able to load (but not calibrate) weights and use them for prediction purposes. We will build three models to classify handwritten digits. These models are a dense, a convolutional and a recurrent neural network.

To help you with the heavy lifting, we provide you with our code template. Familiarize yourself with the structure of this template. In particular, make sure you look at the files:

- *layers.py*: This file contains the code for layer classes such as the Dense or Convolution2D. Here, you will be entering most of your code.
- *test_layers.py*: This file contains unit tests that you can use to test your implementations for the layers. It relies on some data structures stored in the unit test data folder. Make sure you register the unit tests with your code editor such that you can simply click a "run" button for each test that you want to execute. Note that it is not an exhaustive list of tests and you might want to add more tests in case you want to test your layers more rigorously.
- *driver.py*: This file contains the code to instantiate layers and compose them into models. It is also used to visualize the workings of your models.

IMPORTANT: In order to run this code you will need to have python installed along with the libraries *numpy, matplotlib, tabulate, pickle* and *pylab*. In your terminal, run *pip list* to see the list of pre-installed packages. You will probably only need to install *tabulate*. You can install libraries with the command *pip install name-of-library*. If pip does not work, you can also try to manage your packages with anaconda.

Below, you can find a list of exercises that require you to complete code in the accompanying code template. The final exercise explains how you should deliver your work for grading. Have fun!

1 Case Study: Dense Neural Networks

The first model we are going to build will be a dense neural network that shall classify images of handwritten digits. We will need to flatten the images into vectors and then pass them through a sequence of dense layers that outputs a probability distribution over the 10 possible digits. To achieve this in code, we need to do the following:

1.1 Layers

In *layers.py*, navigate to the top class. It contains a base class definition for layers. All other layers listed below will inherit from this base class. Familiarize yourself with the requirements of each method shown in the corresponding documentation comments. Also look at the implementations of the methods to check whether they are correct.

1.2 Flatten

In *layers.py*, navigate to the *Flatten* class and check the documentation comments to see how it works. Also familiarize yourself with the *__call__* method and try to understand how it flattens the incoming images. In *test_layers.py*, navigate to the *FlattenLayerTests* class and run the unit test to verify that the function is working as intended. If it works, great! If not, feel free to ask a teaching assistant for help.

1.3 Dense

Now that we have a way to flatten images, we can feed them into a dense layer. In *layers.py*, navigate to the *Dense* class. Make sure you read the documentation comments of the layer. Check the requirements of the *__call__* method and its implementation. Do you think the implementation is correct ? To test whether it is correct, run the unit test of the *DenseLayerTests* class found in *test_layers.py*.

1.4 Model

The model class brings our previous work together. Navigate to its class in *layers.py* and read the documentation comments of each of its methods to see how it works. Note that it offers a *print_summary* function that presents you with a convenient model overview in your terminal. The *test_layers.py* file also contains a unit test for the model. Run the test to verify that it is working.

In the file *driver.py*, navigate to the first section of the *__main__* script. This section loads images of handwritten digits and creates a model in which the flatten and dense layers are used to classify these images. You will then see that it loads pretrained model weights (the kernel and bias for each dense layer). If you run this code section, you should see the applet of figure 1 popping up. Great job! You have worked your way through the baseline model. Next, you will get the chance to write your own code.

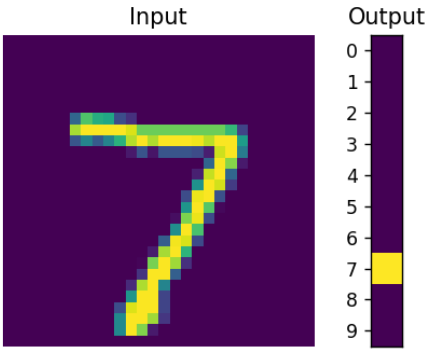

Figure 1: Example of how a neural network maps an image of a handwritten digit to a probability distribution over 10 digits.

2 Case Study: Convolutional Neural Networks

The dense neural network is a sort of 'one size fits all' approach in the sense that it works with many different data representations. Yet, depending on your data representation, there are neural network architectures that are more better suited. For images, convolution makes sense as its kernel 'scan' an image to filter to certain spatial patterns. The kernels are smaller than those of the dense network which makes them more interpretable. Also, our dense network from above, uses ca. 100K model parameters. We can build a convolutional neural network that solves the same task but with only 40K parameters (see below). To build such a model, we need a convolution layer, a max pooling layer and the flatten and dense layers from before.

2.1 Convolution

Review the lecture slides to make sure you understand the principle behind the convolution operation. Try to find out how kernel size, stride and padding affect it. Then, in *layers.py*, navigate to the *Convolution2D* class. Read the documentation comments of all methods of this class. The document comments of the *__call__* and *__iterate__* methods specify the methods' requirements. Make sure you understand them. Ask a teaching assistant if you are unsure. Then, write an implementation for the *__iterate__* method. To test whether your implementation is correct, you have two options:

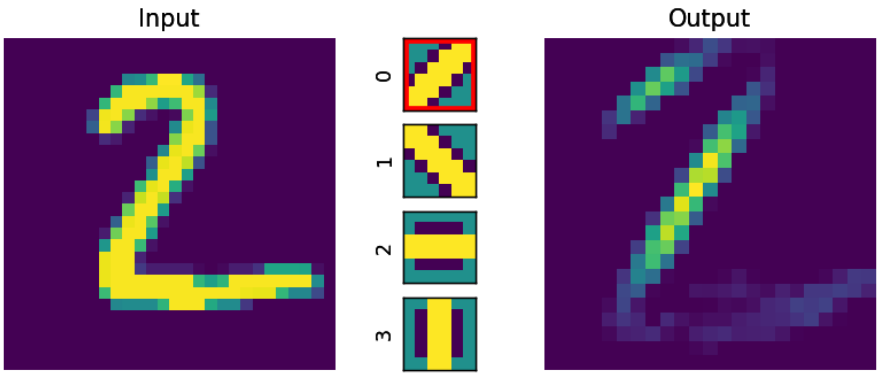

Figure 2: Visualization of 1 step of 2D convolution on a handwritten digit. Here, one of 4 hard-coded kernels is selected.

- Run the unit tests of the *Convolution2DLayerTests* class found in *test_layers.py*. Make sure your convolution implementation satisfies its requirements to ensure that the unit tests will run.
- Navigate to *driver.py* and instead of running the first section on dense networks, run the next section on convolution. This will pass the images of handwritten digits through your convolution layer and then plot how it processes the images using 4 given kernels (see figure 2).

2.2 Max Pooling

Max pooling is similar to convolution, yet it has one important difference. In *layers.py*, navigate to the *MaxPooling2D* class and dive into the documentation comments to find out how max pooling works. If you are unsure, ask a teaching assistant. Then, implement the *__call__* method. Hint, this simply involves calling the *__iterate__* function of *Convolution2D* with the right inputs. Hint: for the 'function' argument to *__iterate__*, you can use a lambda expression that uses *np.max* . Run the unit test of the *MaxPooling2DLayerTests* class found in *test_layers.py* to ensure your pooling works correctly.

2.3 Model

Now that you have all necessary layers, we will build a convolutional neural network (CNN) for handwritten digits classification. To do this, we instantiate a model using our custom layers and load it with a set of pre-trained weights. Navigate to the *driver.py* script. Inspect the next part of *__main__* that builds the CNN, loads weights and runs the plotting function. You will see that the CNN's layers are not yet specified. Use the below instructions to set up an array of layers that you can pass to the constructor of the *model* class.

1. A *Convolution2D* layer named 'conv2d' with 1 input_channel (since our input images only have 1 color channel), 32 output channels (i.e. kernels), a kernel size of 3, stride of 1, padding of 1 and a relu activation function.
2. A *MaxPooling2D* layer named 'pool2d' with pooling size 2, stride 2 and padding 0 and no activation function.
3. *Convolution2D* layer named 'conv2d_1' with ? input_channels (can you find out how many input channels are needed here?), 64 output channels (i.e. kernels), a kernel size of 3, stride of 1, padding of 0 and a relu activation function.
4. A *MaxPooling2D* layer named 'pool2d_1' with pooling size 2, stride 2 and padding 0 and no activation function.
5. A *Flatten* layer with name 'flat'.
6. A *Dense* layer with ? input dimensions and ? output dimensions. Can you find out which dimensions are needed here ? Hint: the *Model.print_summary()* function can be useful. Then, specify the softmax activation function for this layer.

Now, run the script. You should be able to browse through hand-written digits as shown in figure 1 and see the model's output. Good job!

3 Case Study: Recurrent Neural Networks

As you might remember from the accompanying lecture, different modalities pair with different neural network architectures. While images can typically be well recognized with convolutional neural networks (RNNs), temporal sequences are often modelled with recurrent artificial neural networks. In the next step, we will try to see whether the handwritten digits can be classified by looking at the *sequences of pen strokes* that led to them. Figure 3 shows the corresponding input-output mapping. You can see that the number 6 is represented as a sequence of dots, numbered 0 to 52. The gated recurrent unit of the RNN iterates the sequence of pen stroke coordinates. It accumulates information about these coordinates in the sequence of memory vectors (h). The final memory vector will then be used for classification by a dense layer.

3.1 Gated Recurrent Unit

In *layers.py*, navigate to the *GatedRecurrentUnit* class and inspect the documentation comments. Head over to the accompanying lecture slides and review the formulas needed to maintain memory h. Make sure you understand the principle behind the reset gate, the update gate and the candidate vector $\tilde{h}$. Also ask yourself why the gates use the sigmoid function σ while $\tilde{h}$ uses the tanh function ϕ. In the *__call__* method, complete the missing lines of code using the formulas you found in the slides. To test your implementation, you can run the corresponding unit test in *test_layers.py*.

3.2 Model

Now that you have a working recurrent layer, let us see how it can be used inside an artificial neural network to classify handwritten digits. Navigate to the driver script. Make sure you disable the code snippets on convolution from the earlier exercises and enable the section for the recurrent neural network. You will see that the layers of the model are not yet complete. Create the following two layers:

1. A *GatedRecurrentUnit* layer named 'gru' with 2 input dimensions (one for each coordinate) and 32 output dimensions.
2. A *Dense* layer with ? input dimensions and ? output dimensions. Can you find out which dimensions are needed here ? Then, specify the softmax activation function for this layer.

Run the script. You should now be able to browse through hand-written digits and see the model's output as in figure 3. Good job!

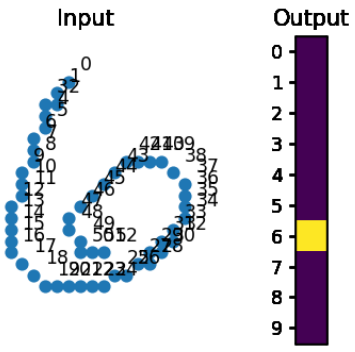

Figure 3: An example of how a recurrent neural network classifies handwritten digits based on a sequence of pen strokes.

4 Submission

Write a **report of 2-4 pages** in which you:

1. Give a brief conceptual overview for each of the three artificial neural networks using a hand-drawn sketch.
2. For the CNN explain how kernel_size, stride and padding affect how an input image is iterated. Use visualizations to aid your explanations, e.g. a hand-drawn sketch or the convolution plotting tool that is provided with the exercise sheet. Explain why you think that convolution is suited for processing images.
3. For the RNN, explain how the input to the recurrent network differs from that given to the convolutional network. Also explain how the recurrent neural network works, meaning, the role of the gates and the memory h and $\tilde{h}$. Make sure you explain why the sigmoid activation function is used for the gates but the tanh function is used for $\tilde{h}$.
4. Aid your explanations by displaying the relevant snippets of your code.

IMPORTANT: Your submission is graded on a pass/ fail basis. To pass the assignment we want to see that you **understand the principles** discussed in the lab. Simple googling or chat-GPT will not get you very far here. You need to make sense of your findings and explain them to us. A good side effect of practicing your critical thinking skills is that you will be prepared well for the final exam. Success.

5 Resources

- Standard Handwritten digit classification dataset (MNIST): https://en.wikipedia.org/wiki/MNIST_database
- Pen-stroke sequence version of MNIST: https://edwin-de-jong.github.io/blog/mnist-sequence-data/