Group number: 530

- Rahabu Mwang’amba rahabum@es.aau.dk
- Yurii Iotov yio@create.aau.dk
- Galadrielle Humblot-Renaux gegeh@create.aau.dk

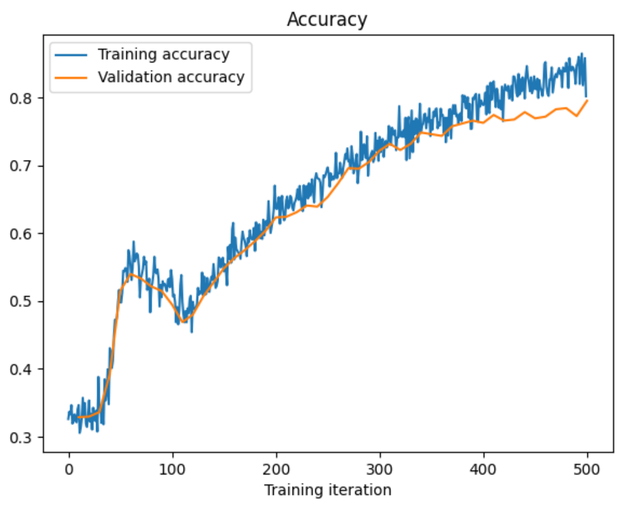
PhD course Deep learning - assignment 1

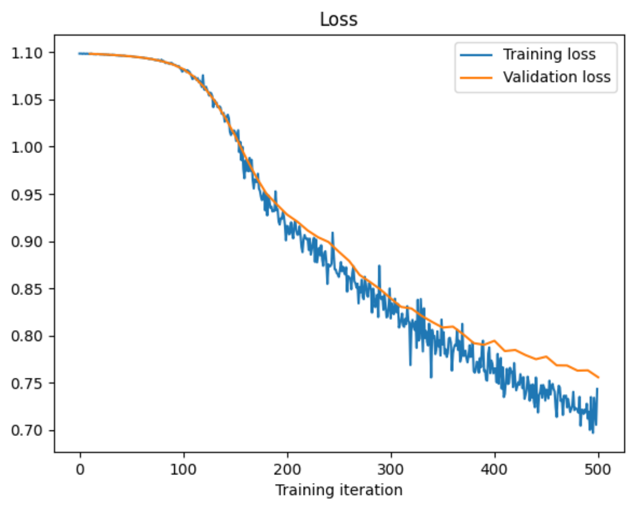
Notes about the data

Input samples have shape 101x40, labels are one-hot encoded vectors with 3 entries (1 per class)
Training set contains 9489 samples -> with a batch size of 1024, there are 10 iterations per epoch

Questions

1. **Input layer:** The size of the input layer depends on the dimensions of the input data. In our case, each input sample has a resolution of 101x40, therefore the input size of the 1st layer is 4040.
   Output layer activation: Since we are considering a neural network that solves the multi-class classification task, and in our case we have 3 classes, the output layer activation function has to be the softmax function which normalizes its input into a class probability distribution (consisting of 3 probabilities in our case), such that each entry in the vector is between 0 and 1, and the entries add up to 1.
   Loss function: We have chosen the cross-entropy loss function since it measures the error between probability distributions (between the one-hot label and the predicted softmax categorical distribution).
2. **Overfitting**: Analyzing the accuracy & loss plots below, we think there is overfitting since the validation loss and accuracy start to plateau after ~400 iterations, after which the performance gap between the training set vs. “unseen” data widens. This means that while the model performance on the training set keeps improving, this performance does not generalize to new data, i.e., it starts to "memorize" the training examples. Strategies to reduce overfitting include:
   a. Model complexity: Simplifying the model, i.e., decreasing the number of layers or the neurons per layer is one way to address overfitting.
   b. Early stopping: We also could try to reduce the number of epochs to train the model, based on when the validation loss and/or accuracy start to plateau.
   c. Regularization: Eg. by increasing weight decay, introducing dropout, or applying data augmentation.





3. **Performance**: Evaluating the model on the test set after 50 epochs, we get a test accuracy of ~77.8%.

4. The number of parameters per layer depends on the number of inputs n_{inputs} and the number of neurons $n_{neurons}$ in the layer. For each neuron, the number of parameters is equal to $n_{inputs} + 1$ (since there is one weight per input + a bias term). For a layer, the number of parameters is thus $(n_{inputs} + 1)\, n_{neurons}$

   Calculating the total number of parameters of the model:

   1) Number of parameters in the input layer: (4040 + 1)×128 = 517.248; where +1 takes into account biased per neuron
   2) …in the second layer: (128 + 1)×128 = 16.512
   3) …in the third layer: (128 + 1)×128 = 16.512
   4) …in the fourth layer: (128 + 1)×128 = 16.512
   5) …in the output layer (we have 3 output classes): (128 + 1)×3 = 387
   6) The total number of parameters is found by summing up all, which gives 567.171 parameters of the model