Neural Networks KU WS17

708.076 16W

Exercise Sheet 3 (Update: 27.11.2017)

Problems marked with * are optional.

Neural Networks [5P + 2*P]

Use TensorFlow in Python 3 to train a neural network for classification of a variant of the isolet dataset. The data set contains features extracted from the utterances of four human speakers saying the characters "a" to "z" several times. Hence, there are 26 classes, one for each character. The test set contains the same features extracted from the utterances of a fifth speaker. Note that the data samples are sorted by speaker identity. From the original data set, we only use the first 300 features to make the task more challenging.

Download the dataset from the homepage. The file `isolet_crop_train.pkl` contains the training set and `isolet_crop_test.pkl` contains the test set. A small script that loads the dataset is provided.

You should implement the network specifics by hand, i.e. using `tensorflow.layer` and `tensorflow.estimator` is not allowed.

Task details [5P]:

a) Make yourself familiar with the dataset. Normalize each feature to zero mean and unit variance.

   The class labels are in the range $1 \ldots 26$. Convert these labels to a one-out-of-K coding (K is the number of classes, i.e., 26) to allow us to train a multiclass classifier.

We now make some initial experiments with a neural network to obtain a good setup. For the next two tasks, use a simple architecture (e.g. one hidden layer with 20 units). The output layer should have K output units with softmax activation.

During this stage, we will not use the test set (which is only used to estimate the generalization to unseen data and should never be used during any kind of validation). Instead, split the provided training data into three sets: training ($\mathcal{T}$, 70%), validation ($\mathcal{V}$, 15%), and early stopping ($\mathcal{E}$, 15%). Errors should be reported in terms of the misclassification rate on $\mathcal{V}$.

You should use early stopping for all network training. Early stopping is a regularization technique which aims to prevent overfitting by evaluation the network on separate data (the early stopping validation set $\mathcal{E}$) during training. During each iteration, the error on this data is computed. After training for a number of iterations, we use the network weights which

lead to the lowest error on $\mathcal{E}$ as final weights. This should lead to less overfitting and thus better generalization.

b) Find a good training algorithm. Compare standard stochastic gradient descent with RMSprop and ADAM in terms of convergence speed and final misclassification rate. Train using minibatches (e.g. using batches of 40 examples per weight update). Use early stopping on set $\mathcal{E}$. The validation error should be determined using set $\mathcal{V}$. Give the mean result over a number of independent training runs (e.g. 10). Furthermore, plot error convergence curves, discuss your experiences and motivate your final choice. You will also have to choose a good learning rate for each of the methods.

c) Find a good activation function to use in the hidden layers. Test the following:

- Logistic sigmoid: $h(x) = (1 + \mathrm{e}^{-x})^{-1}$
- Tanh: $h(x) = \tanh(x)$
- ReLU: $h(x) = \max(0, x)$

Use random weights uniformly distributed in $\left[+1/\sqrt{d}, -1/\sqrt{d}\right]$, where d is the number of inputs per unit (use zeros for bias initialization). Again, use early stopping on set $\mathcal{E}$, and give the mean results on $\mathcal{V}$ over a number independent training runs (e.g. 10) where the weights are randomly re-initialized for each run. Discuss your final choice.

d) Find a good architecture. Test a number of architectures with varying number of hidden units and hidden layers. Use early stopping on set $\mathcal{E}$. Report the mean results on $\mathcal{V}$ over a number of runs (e.g. 10) for each architecture. Again, discuss your experiences and motivate your final choice.

Now fix the architecture. Since we have completed the validation procedure, you can now use the examples in the validation set $\mathcal{V}$ for training (i.e. add them to set $\mathcal{T}$).

e) Train the networks with your choice of training algorithm, activation function, and architecture. Use early stopping on $\mathcal{E}$. Repeat the training and report the final mean test error on the test set over a number of trials.

Bonus tasks [2*P]

f) Use a larger network and try out dropout for regularization. When using dropout, each hidden unit is dropped with some probability $1 - p_{\text{keep}}$ during each training iteration. This aims to prevent the co-adaptation of neurons and thus prevent overfitting. (For more information, see [Hinton et al., 2012]. Check the TensorFlow documentation for how to implement dropout.) Find a good value for the keep probability p_{keep} within $[0.5, 0.9]$. Can you increase the network performance in comparison to your results in e)?

g) Test two additional activation functions which have recently been proposed:

- CReLU: $h_1(x) = \max(0, x)$, $h_2(x) = \max(0, -x)$ (each hidden units has two outputs instead of one). Check the tensorflow library for an implementation.
- Swish: $h(x) = x \cdot \sigma(x)$, where $\sigma(x)$ is the logistic sigmoid

Use the same weight and bias initialization as before. Provide a plot of the activation functions and report your final misclassification rates for each.

Your approach and results for both optional tasks should be described in the report. Also describe how you determined the hyperparameters needed for these approaches.

Additional remarks

The initial weights can have a significant impact on the network performance. A number of different weight initializations have been proposed based on theoretical considerations, see [Glorot and Bengio, 2010] for logistic sigmoid and tanh activations and [He et al., 2015] for ReLUs. The DropOut regularization technique was introduced in [Hinton et al., 2012]. For more on to the activation functions from the last bonus task, see [Shang et al., 2016] and [Ramachandran et al., 2017].

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 249–256.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034.

[Hinton et al., 2012] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

[Ramachandran et al., 2017] Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Swish: a self-gated activation function. *arXiv preprint arXiv:1710.05941*.

[Shang et al., 2016] Shang, W., Sohn, K., Almeida, D., and Lee, H. (2016). Understanding and improving convolutional neural networks via concatenated rectified linear units. In *International Conference on Machine Learning*, pages 2217–2225.

Submit the code until 8:00 AM of the day of submission to `mueller@igi.tugraz.at` and `aid.ahmetovic@student.tugraz.at`. Use "NeuralNetworks HW3, ⟨name team member 1⟩ ⟨name team member 2⟩" as email subject. Only one email per team is necessary.