Machine Intelligence 1 **Exercise Sheet 6**
WS 2017/18, Obermayer/Augustin due: 06.12.2017 at 23:55

Deep networks: convolutional & recurrent architectures

Exercise T6.1: Deep convolutional neural network (tutorial)

Exercise T6.2: Recurrent neural network: long short-term memory (tutorial)

Exercise H6.1: Convolutional neural network (homework, 7 points)

The goal of this exercise sheet is to get familiar with a deep learning framework of your choice, e.g., Tensorflow, Keras or PyTorch. You are free to choose any framework/library you want, but in the following we give some (optional) hints only for Tensorflow or Keras.

In this exercise we work with the publicly available dataset MNIST, which contains black and white images of handwritten digits from 0 to 9. The images consist of 28x28 pixels. The data set is comprised of a training and a separate test set such that the performance of different models can be compared on the same test set (lecture: “validation set”).

Hint: The MNIST data can be loaded easily in many machine learning libraries. For example in Tensorflow the only Python command required to access that data set is given by `from tensorflow.examples.tutorials.mnist import input_data`. Note that most deep learning frameworks require class labels in a one-hot one-hot encoding, i.e., for MNIST you get a 10-dimensional sparse vector with entry one at the specific class, e.g. class of digit 3 has a one at the fourth index $\underline{\mathbf{y}}_T = (0,0,0,1,0,0,0,0,0,0)^T$ (as digit 0 is at the first index). Note furthermore that each image of size 28x28 pixels has to be flattened yielding a 784-dimensional vector $\underline{\mathbf{x}}$.

(a) As a start implement a linear model (remember: $\underline{\mathbf{h}} = \underline{\mathbf{W}}\,\underline{\mathbf{x}} - \underline{\boldsymbol{\theta}}$). Initialize $\underline{\mathbf{W}} \in \mathbb{R}^{10,784}$ and $\underline{\boldsymbol{\theta}} \in \mathbb{R}^{10}$ with zeros. Use the softmax as output to determine the predictions $\underline{\mathbf{y}}(\underline{\mathbf{h}})$. Use the cross entropy as cost function (often instead of cost it is called loss). To train the model use a gradient descent algorithm with learning rate $\eta = 0.5$. Calculate the accuracy (the fraction of correctly classified digits) of the model on the test dataset.
The data shall be processed in mini-batches and the network is trained on one (whole) mini-batch at each gradient descent step (one iteration). For every model in this exercise we use a mini-batch size of 100. For the linear model we use 10000 iterations. Note that this can involve multiple runs over the data (i.e., multiple “epochs”).
Hint: In most deep learning frameworks mini-batches are just referred to as batches.

(b) Implement as second model a Multilayer perceptron (MLP) with 3 hidden layers containing 1500 neurons each. Initialize the elements of the weight matrices with (small) normally distributed random values with mean 0 and standard deviation 0.01, where each initial weight whose magnitude is bigger than two times the standard deviation is recalculated (in Tensorflow see: `truncated_normal`). The biases should be initialized with a constant of 0.1 (this weight and bias initialization should be used in every of the following models, too). Use Rectified Linear Units (ReLU) as activation function. As in the first model, the softmax should be used as output nonlinearity, and the cross entropy as loss function. To train the model use the Adam algorithm, which is often used in deep neural networks instead of standard gradient descent. Use the following parameters: $\eta = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1 * 10^{-8}$. Use 20000 iterations to train the network.

On ISIS you can find a text comparing the different variants of gradient descent which also contains a short explanation of Adam with the same notation as used in this exercise: "Sebastian Ruder - An overview of gradient descent optimization algorithms"

Hint: For prediction it is not important if you use the softmax or its argument, because the predicted label is determined by the maximal value, while softmax just acts as a normalization and does not change the size order of its argument components.

(c) As third model take the MLP from above but with dropout applied for regularization during training of the MLP with dropout rate equal to 0.5 to each of the hidden layers.

(d) As fourth and final model, instead of using fully connected layers combined with dropout regularization, you now have to use convolutional layers, which usually work well in image classification tasks. Note that explicit regularization is not needed for this (small) convoluational neural network.
We define our convolution operation as follows:

- Use a convolutional kernel of size 5x5 pixels and apply it to each image (2-dimensional convolution).
- Apply the convolutional kernel to every pixel in the image (means: stride/shift 1 in every dimension).
- Use zero-padding to obtain the same size of the images after applying the convolution. (In Tensorflow: `padding='same'`).

We define our pooling (downsampling) operation as follows:

- Use max-pooling with a size (window) of 2x2 pixels (2-dimensional max-pooling).
- Use every pixel of the image in just one max-pooling operation (means: stride/shift 2 in every spatial dimension).

The architecture of the network should be as follows:

- At first use a convolutional layer which produces 32 feature maps (in Tensorflow: 32 output channels). Add a bias to every feature map and apply a ReLU activation.
- Apply downsampling as defined before to the features.
- The second convolutional layer should produce 64 features. Proceed as in the first convolutional layer.
- Again apply downsampling as defined before.
- Use a fully connected layer from the downsampled features to the output nodes.

Use the same loss function (cross-entropy) and training algorithm (Adam) as for the multilayer perceptrons. Again use 20000 iterations to train the model.

(e) Compare the training and testing accuracies of the models and plot them over the training iterations (showing each 100th iteration should be sufficient). Can you detect any under-

or overfitting and explain why? What can you say about the use of regularization vs. convolutional layers with downsampling with respect to the overall performance of the models and their ability to prevent overfitting? Consider for example the approximate number of parameters that have to be optimized for each model.
Hint: You can plot the training and testing accuracies over the training e.g., by using two of Tensorflow's `summary.FileWriter` (one for training and one for testing) and `summary.scalar` (which logs the accuracy) in combination with tensorboard (loading multiple runs from different folders), alternatively use the values of accuracy directly by returning them via `sess.run([accuracy])`. When using tensorboard it is sufficient to include a screenshot of the plot into your jupyter notebook. For Keras you may use `history` which is returned by `fit(...)`.

Exercise H6.2: Long short-term memory (LSTM) (homework, 3 points)

In this exercise an LSTM shall be used to classify a simple number series. The goal is to detect whether the sum of the series' elements is greater or equal to 100 (=class 1) or less (=class 0).

(a) Create the train and test data as follows:

- Draw 10000 different series consisting of 30 integer numbers between 0 and 9, where each number is uniformly distributed and independent from the others, that is, $(x_1^{(\alpha)}, \ldots, x_{30}^{(\alpha)}) \in \{0, 1, \ldots, 9\}^{30}$ with $x_t^{(\alpha)} \overset{\text{iid}}{\sim} \mathcal{U}_{\text{int}}(0, 9)$ for $t = 1, \ldots, 30,\ \alpha = 1, \ldots, 10000$.
- A series gets the label 1 if its sum is greater or equal to 100 and the label 0 else.
- Use 8000 series as training set and 2000 series as test set.

(b) Build a recurrent network for number series classification as follows:

- The network is composed of 200 LSTM cells yielding an output vector $\underline{\mathbf{h}}(t)$ in each time step t of the overall 30 time steps.
- On top of the LSTM layer the network should have a single linear output neuron (receiving input from the 200 LSTM cells) with logistic sigmoidal as nonlinearity, i.e, $y(\underline{\mathbf{h}}) \in (0, 1)$. $y(\underline{\mathbf{h}}(30))$ should be interpreted as the probability that the sum of the number series is greater or equal to 100.
Hints using Keras: You have to apply a `Dense` layer with one output neuron and sigmoid activation function after the `LSTM` layer.
- Use cross-entropy between the labels of the training data set and the output y of the network *after the last time step* as loss function for the learning process, and the classification accuracy as performance measure to compare test set and training set result for the trained model.
Hint: Then you can use the cross-entropy for the binary case as loss function as the labels are either 1 or 0.

(c) As many frameworks differ in their implementation of LSTM and allow for different levels of tweaking the parameters you are free to choose parameters which seem reasonable to you (maybe the default values). Also decide on the architecture of the network on your own.
Hint: You can look up a different implementation of LSTM than in the slides at the following article: http://tinyurl.com/q6dcybc. The implementation and notation used there seems to be more similar to most deep learning frameworks.

(d) For the training procedure iterate over the data 60 times in a random fashion (i.e. use 60 epochs) and use a mini-batch size of 50. As training algorithm use Adam with the same parameters as in Exercise H6.1.

(e) Evaluate the final accuracy of the model on the test data.

Total 10 points.