# Homework 3: Ensemble Methods and Recurrent Neural Networks

Large-Scale Data Analysis

Stefan Oehmcke and Christian Igel

**Submission Deadline:**
*June 2, 2020, 10:00*

## 1 AdaBoost (30 pts.)

Prove that in AdaBoost as introduced in the lecture the weight update can be rewritten as:

$$w_i^{(t+1)} = w_i^{(t)} \frac{\exp\left(-\alpha^{(t)} h^{(t)}(\mathbf{x}_i) y_i\right)}{2\sqrt{\varepsilon^{(t)}\left(1-\varepsilon^{(t)}\right)}} \qquad \text{(step 1)}$$

$$= \frac{\exp\left(-f^{(t)}(\mathbf{x}_i) y_i\right)}{n \prod_{\tau=1}^{t} 2\sqrt{\varepsilon^{(\tau)}\left(1-\varepsilon^{(\tau)}\right)}} \qquad \text{(step 2)}$$

$$= \frac{\exp\left(-f^{(t)}(\mathbf{x}_i) y_i\right)}{\sum_{l=1}^{n} \exp\left(-f^{(t)}(\mathbf{x}_l) y_l\right)} \qquad \text{(step 3)}$$

Here $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\} \in (\mathbb{R}^d \times \{-1, 1\})^n$ are the training data, $t = 1, \dots, T$ denotes the boosting rounds. At round $t$, $w_i^{(t)}$ is the weight of training pattern $i$, $h^{(t)}$ is the base classifier from round $t$, $\alpha^{(t)}$ is the importance of this classifier, $\varepsilon^{(t)} = \sum_{j=1}^{n} w_j \mathbb{I}(h^{(t)}(\mathbf{x}_j) \neq y_j)$, and the aggregated classifier is $f^{(t)} = \sum_{i=1}^{t} \alpha^{(i)} h^{(i)}$.

Split your formal proof into the three steps, and proof each step individually. Provide detailed intermediate steps and – for non-trivial steps – give arguments why they hold. If you use research papers, books, or tutorials (we recommend to try yourself), cite them properly.

## 2 Gradient Boosting (10 pts.)

When introducing gradient boosting (see slide “Gradient Boosting I”, given a training data set where the training points are drawn i.i.d. (if you have forgotten what that means, look it up) we assumed that the predictions of the model $\hat{y}_i^{(t)}$, for some round $t$ and $i = 1, \dots, n$ are independent. Is this, in general, true?

Answer the question and provide a proof. This question can be answered briefly, do not write more than three sentences.

## 3 Recurrent Neural Networks (30 pts.)

Replace the simple recurrent layer in the `LSDA2020_RNN1.ipynb` notebook with a LSTM layer. Follow the equations from `LSDA2020_RNN2.ipynb` to implement the LSTM. You will need to:

- **(10 pts.)** add all variables (`tf.Variable`) from the LSTM definition

- **(10 pts.)** extend the `step` and `iterate_series` function
- **(10 pts.)** pass on the hidden state $\boldsymbol{h}_t$ and the cell state $\boldsymbol{c}_t$ in the training loop and the `iterate_series` function

Report the new code in the report (insert all changed code cells from the notebook into the report). Train the new model, report the training loss and also attach the figure of the learned prediction.

Note: You can initialize the cell state $\boldsymbol{c}_0$ to zero (same as for $\boldsymbol{h}_0$) and do not need to learn it.

## 4 Recurrent Neural Networks in Keras (30 pts.)

Add different components to the RNN from `LSDA2020_RNN2.ipynb` and test the new models (each bullet point is a new separate model). The number of neuron can stay the same for all layers (default was 64). Report the mean validation error **and** the changed parts with a bit of context in the code (e.g. complete model or optimizer definition). Also, provide a model overview with `model.summary()` for each new model (not necessary for gradient clipping).

1. **(7 pts.)** Replace the LSTM layer with a Bidirectional-LSTM. Bidirectional sequence processing is possible with `tf.keras.layers.Bidirectional`.
2. **(4 pts.)** Stack 2 LSTM layers (you may need to use the `return_sequences` parameter).
   - **(4 pts.)** Detail the difference to bidirectional processing (max. 2-3 sentences).
3. **(7 pts.)** Add a 1-d convolution layer (`tf.keras.layers.Conv1D`) with a kernel size of 3 and stride of 1 before the recurrent part. You will need to reshape the data with `tf.keras.layers.Reshape`.
   - **(5 pts.)** Explain the difference between how a convolutional layer process time series and how a recurrent model does it (max 4-5 sentences).
4. **(3 pts.)** Gradient clipping can be a helpful to train recurrent networks. Keras offers to clip gradients directly through the optimizer. Change the optimizer to clip the gradients to 1.

It is more important here to show that you correctly changed the model (by showing the corresponding code in the report) or optimizer than improving the results.