Machine Learning - Homework 11 Report

Lukas Johannes Ruettgers (2023403372)

December 31, 2023

1 Introduction

Sequential data occurs widely in time-dependent signals such as spoken audio waves, video streams and written language. The early intuition to keep a hidden memory state inside the model that remembers important events in the prior signals quickly turned out to entail both vanishing and exploding gradients, because the chain rule is applied many times during gradient backpropagation. Schmidhuber et al. (2) therefrom obtained the Long Short Term Memory (LSTM), which does not multiply the prior hidden state to the new hidden state, but merely adds it. Despite their computational complexity in comparison with other recurrent neural network architectures such as the Gated Recurrent Unit (GRU), LSTM has proved a sound performance in many machine learning tasks on time-dependent data streams.

2 Experiment Objective

This experiment examines the impact specific hyperparameters play with regard to the text classification performance of Long Short Term Memory (LSTM). To that end, the experiments avail to the *AG_NEWS* dataset, which contains news ranging across four domains, namely World, Business, Sports, and Science and Technology. Specifically, I want to investigate:

1. what advantage the LSTM offers over normal encoder architectures,
2. how strongly a larger embedding, which is the input of the LSTM, improves the expressivity of the hidden state,
3. how the classification performance depends on the size of the hidden layer itself,
4. whether a stacked multi-layer LSTM exhibits a noticeable advantage over its classical one-layer counterpart,
5. and how sharply the training efficiency and prediction capability decreases when shrinking the dimensions of the embedding and hidden space.

To approach these five questions, I implemented an LSTM that receives embedding size, hidden size, LSTM layers, and the output size as input parameters. Then I created a baseline model by selecting parameters that achieved mediocre classification performance on the AG_NEWS dataset. These parameters were the following:

- Embedding space dimension: EMBEDDING_DIM=16,
- Hidden layer size: HIDDEN_DIM=16,
- Number of LSTM layers: LSTM_LAYERS=1,
- The input and output dimension according to the vocabulary size and the four classes respectively.

To directly express the output vector as a probability distribution over the possible classes, the values of the output vector were normalized to $[0, 1]$ with a final softmax layer. For each of the four questions above, I adjusted these baseline parameters and hence obtained the following three models:

1. **No LSTM:** To study the impact of the LSTM on the classification performance, the LSTM unit was completely removed in this model. The embedding vector directly served as input for the fully connected linear layer.

2. **Double Embedding:** Baseline model with twice the embedding dimension EMBEDDING_DIM=32.

3. **Double LSTM:** Baseline model where the size of the hidden layer was doubled to HIDDEN_DIM=32.

4. **Multilayer:** Baseline model with LSTM_LAYERS=3 instead of only 1 layer. The output of the prior LSTM was sequentially fed into the next layer.

5. **Minimal:** Baseline model whose embedding and hidden state dimension were both shrinked to EMBEDDING_DIM=6 and HIDDEN_DIM=6 respectively.

2.1 Implementation Details

2.1.1 Model

From the phrases in the dataset, a vocabulary was created, where each word was uniquely associated with an integer. A sequence of words would hence be mapped to their sequence of corresponding integers before they were fed into the model. The model was implemented with the `torch.nn` library. At first, a

2.1.2 Training

One fifth of the training dataset was randomly selected as a cross validation dataset. The remaining four chunks were concatenated to the training dataset. Each model was trained with the Adam optimizer and 80 training epochs. In each epoch, the entire training dataset was split into 100 batches and one gradient descent step was performed on the cross entropy loss between the model's argmax predictions on this entire batch and the ground truth labels. As a learning rate, $\alpha = 0.0001$ was observed to yield adequate results. For each batch, the prediction accuracy was computed as the number of correctly predicted classes divided by the total number of classes. While this metric may only implicitly account for the uncertainty of the model's prediction, I used the average prediction probability of the ground truth for the testing dataset to finally take this uncertainty into account. The training batch accuracy was averaged over all batches in one epoch. After each epoch, the model's accuracy on the cross validation dataset was computed in the same way and both values along with the average cross entropy loss were logged to the `results.txt` file in this folder.

2.1.3 Testing

As aforementioned, the accuracy on the test dataset was computed as the average prediction probability of the ground truth. Moreover, the computation of the confusion matrix followed the same spirit, as each row $i, 0 \leq i \leq 3$, in the confusion matrix represents the average softmax vector that the model predicted for a digit i.

2.1.4 Reproducibility

To ensure the reproducibility of the experiment results, all random number generators from torch, numpy and the internal random package were fixed. Moreover, I tried to force torch and CUDA to use only deterministic algorithms by setting the environment variable `CUBLAS_WORKSPACE_CONFIG` according to the API Reference Guide of cuBLAS (1). However, I wasn't able to perform any operations on the GPU if I set this variable. For this reason, there was still a small amount of randomness in the execution when running the code on a GPU, which was however far smaller than during the last week's experiments.

3 Results

Because the accuracy seemed the most intuitive and understandable metric, I decided to plot the learning curves over this metric. In contrast to the experiment in the last week with the CNN, the training curves are much smoother and do not seem to ascend in discretized, sudden steps (Fig. 1). The baseline model achieves an accuracy of 96.71% on the training dataset, but only 90.91% on the test dataset (Fig. 2). All in

all, the other models can not raise this score significantly, i.e. the model without the LSTM unit achieves the best accuracy on the test dataset, which is however not even one percent better than the accuracy of the baseline model. On the other side, stacking multiple LSTMs on top of each other even damages the performance, as is reduces the test accuracy by almost 4% (Fig. 2). These results might be mainly due to the encoder unit, which receives the entire batch of news texts as an one-dimensional input along with the offsets of each word and transforms each sample in the batch to an embedding vector. During this process, valuable positional information which the LSTM could seize to improve its predictions on the sequence might get lost. The dominance of the model without the LSTM unit even suggests that the hidden state dimension of 16 acts as a bottleneck on the expressivity along the entire pipeline of the model and hence renders the LSTM superfluous. Furthermore, a brief look on the proximity of the cross validation and training accuracy curve reveals that the performance gap between the validation and training distribution diminishes for the simpler models (Fig. 1e, 1f), which might indicate that the stronger models overfit on the training data after having reached the 90% landmark for a few training epochs. Interestingly, the accuracy curve for the minimal model 1f exhibits a similar discrete pattern with sudden, steep inclines like in the last week's experiment. It might support the fact that a weight shift in a small model has a stronger influence on the entire model's decision making than a weight would have for a stronger model. Naturally, the weak model thence allows its parameters to shift the accuracy more drastically. All models except the minimal model steeply improve their accuracy in the first few training epochs and mostly reach the 90% landmark within 10 training epochs (Fig. 1b, 1c, 1d). On the other hand, the baseline and the LSTM-free model need around 20 epochs to reach that level (Fig. 1f). One final interesting observation was made regarding the choice of the optimization algorithm: when I chose stochastic gradient descent instead of the Adam optimizer, all solutions didn't seem to improve at all but kept stuck at a random prediction behaviour with 25% success rate. Although I didn't figure out why the stochastic gradient descent does not seem suitable for this problem setting, I got sensitized that the choice of the optimization algorithm is a crucial component of the entire model and can drastically affect its performance. A final glance on the confusion matrices reveals the common ambiguities the models struggle to tell apart. Most significantly, there seems to be a relatively large overlap between Business and Science & Technology news that all model's didn't manage to separate (Fig. 3). On the other side, news related to sports seem easily distinguishable for all models, as no model has a weaker false positive rate (FPR=$\frac{FP}{FP+TP}$) than four percent (Fig. 3d). Conversely, both the accuracy and the true positive rate (TPR) seem particularly bad for the news related to business. However, as the borders between these fields are quite coarse and flexible, the overall performance preservation with regard to the small amount of available weights is still impressive.

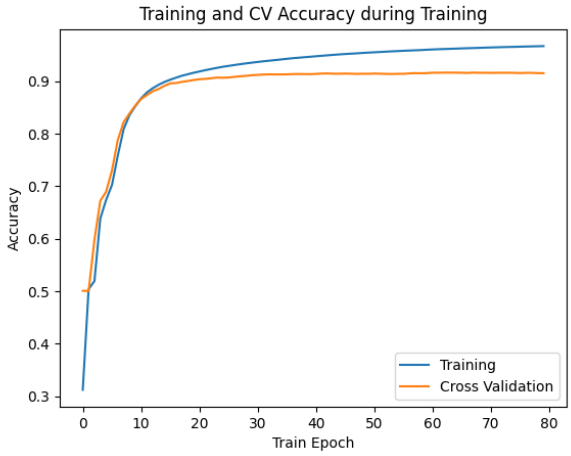
4 Conclusion

This experiment firstly confirms that the model's performance is quite sensitive to the initial choice of the optimization algorithm. Secondly, the efficacy of a LSTM largely relies on the preservation of positional relations in the embedding that is computed one layer before the LSTM. With an unsuitable encoding, the LSTM can not unleash its potential in contextual memory and reasoning and can even act as a performance bottleneck on the other components. Thirdly, stacking more layers of LSTMs upon each other won't naturally improve the classification performance and can even conversely exacerbate it.

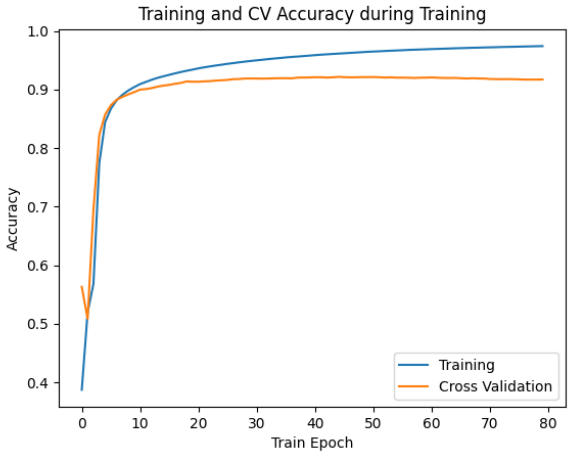
References

[1] NVIDIA Corporation & affiliates. cuBLAS API Reference Guide. https://docs.nvidia.com/cuda/cublas/index.html#cublasApi_reproducibility, 2023. Accessed: 2023-12-20.

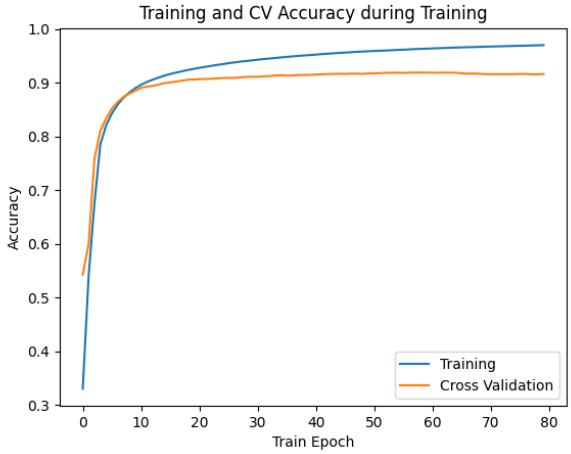
[2] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.



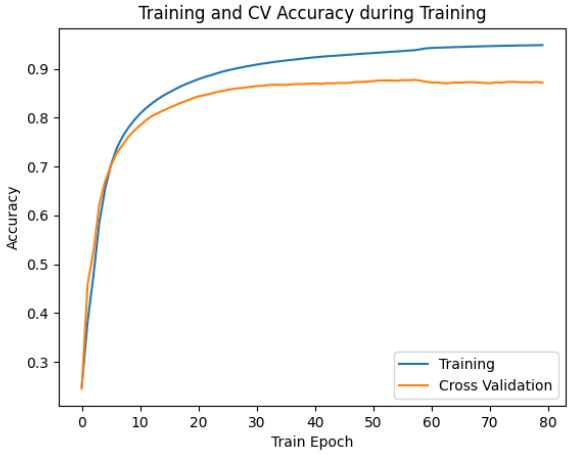
(a) Baseline



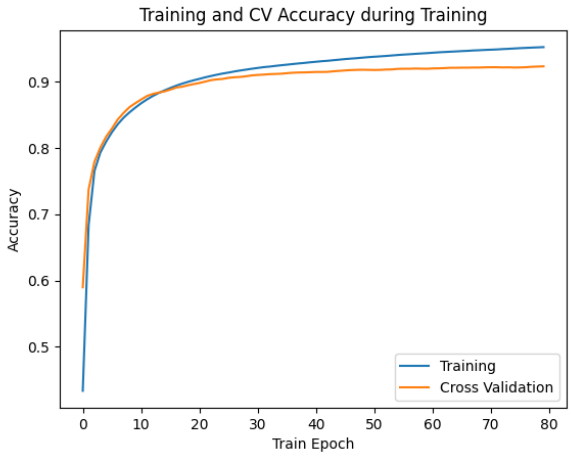
(b) Double Embedding



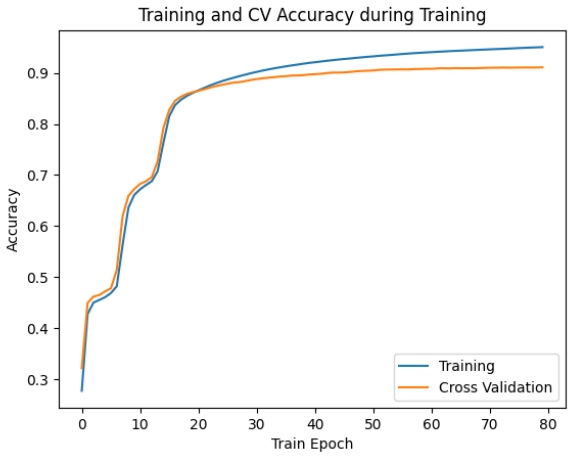
(c) Double Hidden State



(d) Multilayer



(e) No LSTM

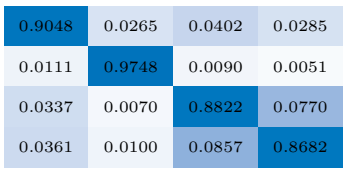


(f) Minimal

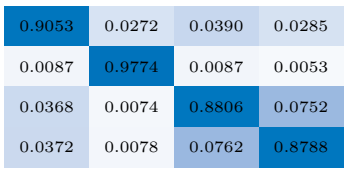
Figure 1: Learning curve during training that reports the classification accuracy for both the training and the cross validation dataset.

Model	Training Accuracy	CV Accuracy	Test Accuracy
Baseline:	0.9671	0.9155	0.9091
Double Embedding	0.9743	0.9173	0.9135
Double LSTM	0.9700	0.9162	0.9125
Multilayer	0.9486	0.8717	0.8699
No LSTM	0.9523	0.9233	0.9189
Minimal	0.9504	0.9107	0.9059

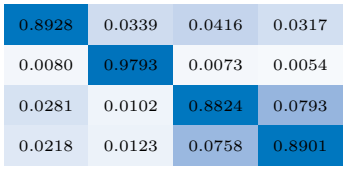
Figure 2: Accuracies of the six models on the training, cross validation (CV) and test dataset. This data was extracted from `results.txt`.



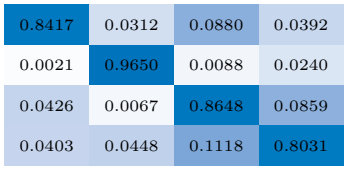
(a) Baseline



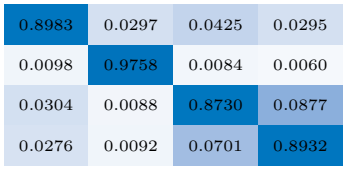
(b) Double Encoding



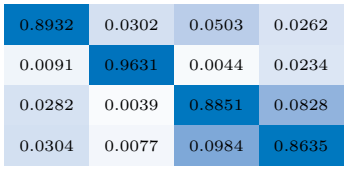
(c) Double LSTM



(d) Multilayer



(e) No LSTM



(f) Minimal

Figure 3: Comparing the confusion matrices of the six LSTM models. The index (starting from 0) of each row indicates the ground truth label, the column index represents the softmax vector prediction of the model. This data was extracted from `results.txt`.