# Modelling Second Language Learning

**Julia Henkel** **Konstantinos Saitas - Zarkias** **Clara Tump**

## Abstract

Modelling the learning process of a person is a difficult task, especially with a concept like language which can be very abstract and non-definite. In our project, inspired by the 2018 SLAM [1] competition (1) by Duolingo, we modelled users learning a second language. Achieving such a goal can be highly beneficial for the learning process. By monitoring the progress of a user and tracking in which exercises she is struggling or finding it too easy, the system could automatically adapt to provide a more personalised experience. Our model for this task is a Long Short-Term Memory (2) (LSTM) Neural Network trained with Adam (3). The data is provided by Duolingo and consists a total of 2.8 million words with meta-data information on the word and the user. Due to the great size of the dataset, training a model was particularly challenging. To overcome the different issues that arose when dealing with this high amount of data, we down-scaled our problem to a much smaller dataset. Using this subset of the data, we carefully selected a set of effective features based on the results reported by Duolingo and used word embeddings and binary encodings for dimensionality reduction. Subsequently, the LSTM parameters were tuned and the influence of using class weights tackling the unbalanced dataset were tested. By comparing binary token encoding to word embeddings it was found that word embeddings have a slightly positive effect on the models performance. Additionally, it was found that slightly undercompensating the class imbalance by the class weights was optimal.

## 1 Introduction

The last few years have seen an enormous rise in the amount of digital data in education. The availability of digital data creates possibilities of personalised education systems, which track the performance of a learner and give her next exercises and explanations that are personalised just for her. It has been shown that such personalisation results in more efficient learning and higher motivation of students (4). Some personalised education systems have been created in exact fields in mathematics, but introducing these in more abstract learning concepts such as language learning is more difficult since it requires knowledge of syntax, semantics, and other language related fields. However, there is a lot of data available already and even though it is difficult, it has a lot of potential.

Duolingo [2] is a game-like language learning application for web and mobile and it is used by over 200 million second language learners. Users receive different exercises in which a series of translation tasks are presented to them. An example can be seen in figure 1, where a user is trying to learn English, with native language Spanish. Duolingo available 2.8 million of such exercises for their Duolingo's 2018 SLAM competition (5). The aim of this competition was to explore how well one could model the second language learning of Duolingo users. The goal of the given task is to predict mistakes of a user when translating words during exercises. Such a prediction system could be used for improving the platform of Duolingo, to upgrade the exercises and to provide a more personalised experience for the user, since the system could adapt to specific strengths and weaknesses.

[1] Second Language Acquisition Modelling

[2] `www.duolingo.com`

| | PRON | VERB | PRON | NOUN | CONJ | PRON | VERB | PRON | NOUN |
|---|---|---|---|---|---|---|---|---|---|
| Correct | She | is | my | mother | and | he | is | my | father |
| Student | she | is | | mader | and | he | is | | fhader |
| Label | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ |

Figure 1: Duolingo translation exercise for a user learning English. The top line shows the correct words, below that the student answer is given. Each answer is labelled correct or incorrect.

In this project we used a Recurrent Neural Network with LSTM units for our prediction system. Before starting the actual training of the model, a series of pre-processing methods had to be done including for efficient feature selection and dimensionality reduction. The initial data provided by Duolingo consists of multiple features for each sample. Each feature has a value which is a string for the categorical features and a number for the numerical features. The first task was to find a memory efficient vector encoding for the different features. We tried different encoding and dimensionality reduction techniques on the data. Our next challenge was dealing with the huge amount of data samples. After trying multiple ways of loading and training on batches, we ended up using and building a much smaller dataset to experiment on. Finally, a series of experiments were performed to gain knowledge about different deep learning methods for building successfully a prediction model.

## 2 Related work

Second Language Acquisition modelling (SLAM) is a task that is related to multiple different fields. A plethora of approaches have been used for modelling student learning and predicting performance on future exercises. In one study (6), the authors proposed a method called Item Response Theory (IRT), which is often used in Intelligent Tutoring Systems. IRT is a model which takes into account both the difficulty of an exercise and the individual user model. Another method is Bayesian Knowledge Tracing (7) which is comparable to a Hidden Markov Model (HMM) and models at which point a user has learned a concept. One of the newer approaches is Deep Knowledge Tracing (8), which models learning over time using RNNs.

The Duolingo SLAM 2018 competition received 11 submissions (1) (review paper by Duolingo). The most successful learning algorithm choices were RNNs and Gradient Boosted Decision Trees (GDBTs). One team used 4 RNN encoders, each representing a different feature type: token context, linguistic information, user data, and exercise format. Another team used an ensemble of two RNNs, one to learn possible mistakes for a given token, and the other one to create a user model. Another high ranking team used an ensemble of GDBT's and creation of features based on cognition theory, for example a feature representing the user's motivation based on usage patterns. The winning team, SanaLabs, used an ensemble of an RNN with a Gradient Boosted Decision Tree (GBDT). The reasoning behind this choice was that RNNs have been shown to produce good results on sequential data, while GBDT's are still state of the art on tabular data (9). The model was a bidirectional LSTM, with multiple stacked ordinary and recurrent layers. Unfortunately, this team did not provide their code, so further specifics of the model are not known.

Additionally, many teams created extra features. However, as pointed out by the final review paper of Duolingo on the different approaches the teams had, the feature selection had a smaller impact than the choice of learning algorithm (1). The features with a positive impact on the performance are the following: $token$ (surface form), $token$ (word embedding form), $userID$, $country$, $session$, $days$ (the number of days since the user started learning this language), $time$ (the response time for this exercise), and the $userID$. However, of these only the influence of $days$ and $time$ were statistically significant.

## 3 Data

For this competition a large amount of data was provided by Duolingo which contained results of users using the app to learn a second language (5). The dataset we worked with contains a list

Table 1: Features of a data sample

| *Feature* | *Type* | *Description* |
|---|---|---|
| token | string | The actual word |
| user | string | The user taking this exercise |
| country | string | The country the user is from |
| session | string | lesson, practice or test |
| client | string | The OS the user is using Duolingo on |
| time | int | seconds taken to submit the answer |
| days | float | time since starting the current course |

of completed exercises from English to Spanish. Each exercise contains multiple samples, each representing a word. Each sample has a label, 0 or 1, representing if the user correctly translated that word or not respectively. Additionally, each word also comes with multiple other meta-information features. These features are given in Table 1.

The train dataset consisted of 824.012 exercises with a total of 2.622.957 samples (words with meta-data), where 10% of that was used for validation, and the test dataset consisted of 115.770 exercises of 387.374 samples. Due to hardware memory complications, analysed in 4.3, we ended up running our experiments on a smaller dataset of 12.000 samples consisting 9.681 training, 1.076 validation and 1.201 testing samples.

# 4 Methods

## 4.1 Learning model

We worked with a Long-Short Term memory Neural Network (LSTM), since the data is of sequential nature. This approach was also used by the Sanalabs team (9). The LSTM is trained with Back-Propagation Through Time (BPTT). The data is inputted as a feature matrix at each timestep, containing the feature vector of the current word, and the feature vector of the $t$ previous words, where $t$ is set to 100 by default. By default, it is trained using the Adam optimizer (3), and 100 timesteps. Dropout and batch normalisation were also used. It has one hidden layer with 128 nodes with ReLU activation. The last layer is a fully connected layer, with one sigmoid output node, which makes a binary prediction of the correctness of the current word.

## 4.2 Feature encoding

In order to use a Recurrent Neural Network with more then 50 timesteps we had to find solutions to reduce the feature space from more then 4000 to less then 100. The features with the highest amount of different values were the user, the country and the token. We decided to encode the user and the country as binary instead of as one-hot which reduced the size of the representation of these features exponentially. Instead of encoding the feature "token", which had more then 2000 values as one-hot we decided to use word embeddings.

Word embeddings are based on the assumption that words that occur in a similar context have a similar meaning. Each word is mapped to a low dimensional numerical vector which depends on the context in which the word occurs. Words that occur in a similar context have a similar representation and are close together in the feature space.

If we apply word embeddings to our learing problem we assume that words that occur in a similar context have a similar probability to be incorrect. This makes sense if we think about an example where a user learns different country names. If the user is just about to learn these names it is reasonable that she either translates everything wrong or everything right. On the other hand there might also exist synonymous words with one being very similar to the word in the users native language and the other one not. One of them would be learned faster then the other one. Here we could see a potential problem of using word embeddings.

Training our own word embeddings on our dataset would have been very time consuming and we would have needed an even larger dataset then ours and a lot of computational power. Instead we

decided to use the pre-trained GloVe 6b word embeddings. All the tokens from our dataset had a word embedding representation of length 50 in the GloVe dataset.

As we can not capture all the information with word embeddings as mentioned before we decided to compare them with a binary encoding. A binary encoding reduces the size of a one-hot encoding exponentially. As some categorical variables share the same weights we also lose information here. The results of using different methods are reported in section 5.

### 4.3 Dealing with a large dataset

The results of running the baseline model provided by Duolingo with part of the data and the whole dataset, indicated that the use of the whole dataset might be essential to have a full working model and thus we were motivated to spend a good amount of time to try and use all of this data.

After processing the data and creating a new dataset with the new representations of the features and with an added history for each word, the final data file turned out very memory heavy (120 GB train file, 20 GB test file). However, the Google Cloud can only run around 15 GB per run. This meant we had to find solutions on how to work with such a big amount of data.

As a solution we built a method to load smaller percentages of the data each time, do any preprocessing needed and save that batch to a file, thus ending up with multiple train and test batch files. When it came to training, we would load one of the files at a time, train the model with that batch, save its state (the weights but also the state of the optimiser, such as the learning rate in that timestep) and then go other to the next batch to continue training as mentioned.

Unexpectedly we still had memory issues and tried to find out the reasons for that. At the beginning of each experiment our program was fine with the amount of RAM it had. It would load each data package and train with it. But although we deleted all the data after the each training and used a garbage collector the memory usage still continuously increased for each data package we trained with. After a lot of research we found out that this could be a typical python issue from using *numpy* arrays. We could finally solve the memory problem by training the LSTM in a subprocess that starts with loading the data package and the model, trains the model on the data package, saves the new trained model to a file and ends. As we have a new process for each data package and the process ends with releasing all the memory it used, we managed to solve the memory problems.

### 4.4 Unbalanced classes

Another issue of this dataset is that around 85.7% of the users' translations are correct, causing a large imbalance between the two classes. This would mean for example that a classifier that only predicted zeroes would end up with 85.7% accuracy. In order to get better insights of the models' performance we implemented another metric called Matthews Correlation Coefficient (MCC) (10) which is specialised on imbalanced binary classification problems. This metric indicates the performance of a classifier based on the results of the confusion matrix and the values return are between -1 and 1, where -1 is a classifier that predicted all data samples incorrectly, 0 is equal to random and 1 equal to an all correct classifier. Using this metric and looking up our confusion matrix results we can a more intuitive feeling and make a clearer comparison between different models.

To combat this issue while training, we chose to use function provided by Keras called *class weights*. When enabled, samples of each class are fed into the network with small weight factor values when calculating the loss function during training.

## 5 Experiments

Multiple experiments were done to investigate the influence of feature choices and model choices. Firstly, we investigated the effect of using different features on our model and we look at the influence of using word embeddings versus binary encoding for the words. Next, experiments are done to find the optimal values for the models hyperparameters. Lastly, an experiment is performed to find the optimal class weight balance for tackling the unbalanced dataset. The metrics used are the following:

- AUC: The area under the ROC curve.
- F1: An overview metric of the confusion matrix.

Table 2: Results of the baseline model

| Data used | Accuracy | F1 | AUC |
|---|---|---|---|
| 20% | 85.7 | 0.0 | 0.611 |
| 100% | 88 | 0.190 | 0.774 |

- MCC: Another overview metric of the confusion matrix with more intuitive values of the classifier's performance. MCC is often used for datasets with unbalanced classes.
- Accuracy: the percentage of samples which were correctly predicted
- Correctly predicted as 0:True Negatives (TN)
- Correctly predicted as 1: True Positives (TP)
- Incorrectly predicted as 0: False Negatives (FN)
- Incorrectly predicted as 1: False Positives (FP)

### 5.1 Baseline

Duolingo provided code for a baseline model which is based on Logistic Regression. A very important note, however, is that when using a smaller amount of the data (20%) to train this baseline model, the results were same as random (table 2). Only when the whole dataset was used, was the model able to learn the data and it was able to perform better than random.

### 5.2 Feature Selection

Each word comes with meta-information (Table 1). We run three experiments to compare the effect and importance of feeding the network all this information. As shown in table 3, the differences between the models is not significant. This is expected as the meta-analysis of all the competition (1) mentioned that feature selection was not very important. This was still an interesting experiment as we can see that even seemingly irrelevant features such as the client, can still have a positive effect. This led us in our decision to include all features in our next experiments.

Table 3: Comparing the effect of using different features

| Features used | TN | TP | FN | FP | MCC | F1 |
|---|---|---|---|---|---|---|
| user, countries, session, token, days, time | 928 | 122 | 83 | 353 | 0.23 | 0.35 |
| user, countries, session, token, days, time, format | 965 | 135 | 70 | 316 | 0.30 | 0.41 |
| user, countries, session, token, days, time, format, client | 981 | 122 | 83 | 300 | 0.27 | 0.38 |

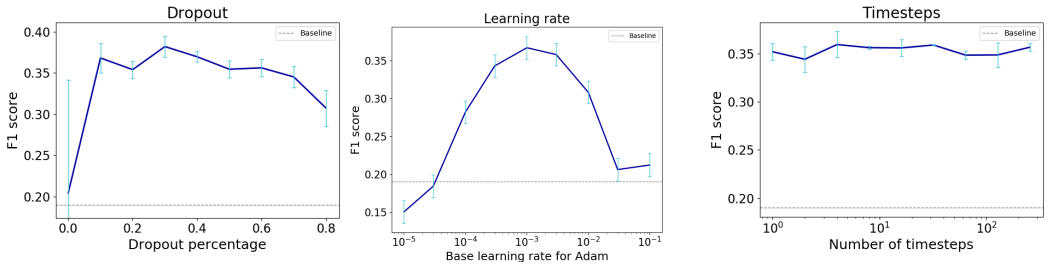
### 5.3 Word Embeddings

We ran 5 experiments to compare the results of using a binary token encoding to using word embeddings. In Figure 3 you can see that the model that we trained with word embeddings performed better. We think that this slight improvement is due to the fact described in section 4.2, where we described that words that appear in similar context tend to have a similar probability for error.

### 5.4 Hyperparameter Optimization

The parameters to be optimised are the learning rate, dropout rate and number of timesteps (words) to look back. For each parameter, an experiment is created which consists of the model being trained for 20 epochs with different values of the parameter. These experiments are repeated 5 times. The results are given in Figure 2. When compared with the logistic regression baseline (see section 5.1), the results are all well above the baseline F1 score. This shows that even with such a small subset of the data it is able to reach significantly better results than a logistic regression model trained on the full data set.

Figure 2a shows the experiment for the dropout rate. It can be seen that the values do not differ much, but the dropout rate of 30% is a clear peak in F1 score. This value is thus chose as optimal dropout rate. In Figure 2b one can see the experiments for the learning rate. It is clear that the best learning




(a) Dropout rate (b) Learning rate (c) Timesteps

Figure 2: Experiments for optimisation of the hypoerparameters of the LSTM. The data points of the graph are the mean of 5 repetitions, while the errorbars show the standard deviation. The F1 score of the baseline logistic regression model (0.192) is illustrated by the dotted gray line.

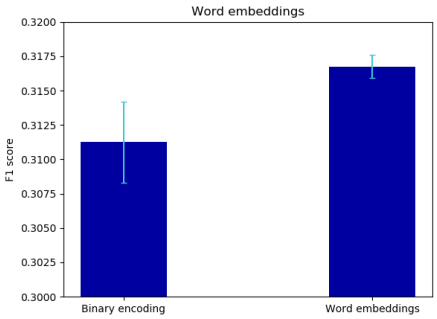



Figure 3: Token binary encoding and word embeddings compared

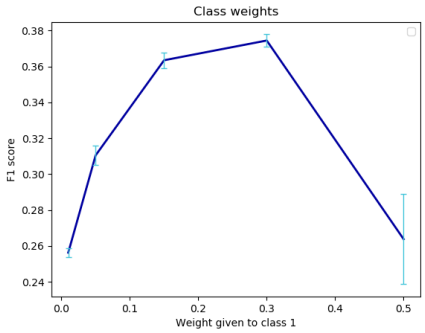



Figure 4: Experiments for optimizing the class weights

rate is in the range of $10^{-4}$ to $10^{-2}$. The peak value of $10^{-3}$ is chosen as the optimal parameter. Naturally, the optimal learning rate depends on the number of epochs. With more epochs, a smaller learning rate might give better results. However, due to time constraints, we did not test this. Lastly, figure 2c shows the results for varying the number of timesteps. Surprisingly, this indicates that a greater number of timesteps has no positive effect on the performance. Apparently, the current model is only using information about the word itself and the user and it is not using the information from the previous words in an exercise. A reason for this might be that the current model is based on such a small subset of the data, with not enough training data to build a complicated sequential model.

### 5.5 Class weight optimisation

Another experiment was done to find the optimal class weight balance. To be able to do statistical evaluation, the experiment was again repeated 5 times. The two present classes of our model are 0, meaning "correct" and 1 meaning "incorrect". Giving a weight of 0.2 to class 0 results in giving a weight of 0.8 to class 1. Using class weights in Keras means weighting the loss function, so that its given more attention to the high weighted class. This can be used to deal with unbalanced classes, as ours where 85.7 percent of the data belongs to class 0. Not using class weights results in a model that is not learning along the features but just predicting the class that has a higher representation in the dataset as this would be correct with a very high probability. As shown in figure 4 the best score was achieved by having a balance of 0.3 for class 0 and 0.7 for class 1. At first this was unexpected for us. We expected that the optimal balance would be 0.15 for class 0 and 0.85 for class 1 as this would results in completely balanced classes. We ascribe the high score for the weight 0.3 to the fact that we are just training with a low amount of data. This results in a model that benefits from having information about which class occurs more often.

## 6 Conclusion

The experiments show that even though the model is not trained on the whole data, it is able to learn significantly better than the logistic regression baseline model on the full data. Moreover, the experiments indicate that encoding the tokens as word embeddings have a positive effect on the performance. Additionally, introducing class weights are effective in tackling the class unbalance, and they are most effective if they slightly undercompensate the unbalance. However, these findings were on a subset of the dataset so it is not said that it is possible that these also hold for a model which is trained on the whole dataset.

It would be beneficial if future work focuses on repeating these experiments on the whole dataset. In addition, the results by SanaLabs (9) show that better performance is reached if the LSTM is combined with a method called Gradient Boosting Decision Trees. Another possibility for future work would be to focus on creating both a general model and a model for each individual user. This would make it possible to capture the specific struggles of each user better, and thus better facilitate a personalized learning experience.

## References


[1] B. Settles, C. Brust, E. Gustafson, M. Hagiwara, and N. Madnani, "Second language acquisition modeling," in *Proceedings of the NAACL-HLT Workshop on Innovative Use of NLP for Building Educational Applications (BEA)*. ACL, 2018. [Online]. Available: https://doi.org/10.7910/DVN/8SWHNO

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[4] K. Bauman and A. Tuzhilin, "Recommending learning materials to students by identifying their knowledge gaps." in *RecSys Posters*, 2014.

[5] "Duolingo 2018 slam competition." [Online]. Available: http://sharedtask.duolingo.com/

[6] F. M. Lord, "A theory of test scores (psychometric monograph no. 7)," *Iowa City, IA: Psychometric Society*, vol. 35, 1952.

[7] Y. B. David, A. Segal, and Y. K. Gal, "Sequencing educational content in classrooms using bayesian knowledge tracing," in *Proceedings of the sixth international conference on Learning Analytics & Knowledge*. ACM, 2016, pp. 354–363.

[8] C. Piech, J. Bassen, J. Huang, S. Ganguli, M. Sahami, L. J. Guibas, and J. Sohl-Dickstein, "Deep knowledge tracing," in *Advances in neural information processing systems*, 2015, pp. 505–513.

[9] A. Osika, S. Nilsson, A. Sydorchuk, F. Sahin, and A. Huss, "Second language acquisition modeling: An ensemble approach," *arXiv preprint arXiv:1806.04525*, 2018.

[10] B. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme," *Biochimica et Biophysica Acta (BBA) - Protein Structure*, vol. 405, no. 2, pp. 442 – 451, 1975. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0005279575901099

# 7 Appendix: Learning outcomes

## 7.1 Julia

As this project has been the first bigger machine learning project I did the most relevant part was to learn about the **general procedure of performing deep learning project**. This included first of all processing the data to a useful vector representation, then implementing an easy model with **Keras** and making sure that it is bug free by overfitting the model on the training data and finally doing a broad hyperparameter search by running every experiment with certain parameter settings several times. In particular I learned a lot about how to **process the data** so that it can be used in an LSTM. One issue that got a lot of my attention was exploring **dimensionality reduction techniques**. This included using binary encoding instead of one hot encoding and introducing word embeddings. Finally I learned how to implement a **framework for experiments** that is makes it easy to run a new experiment with different parameters and that builds clear representations of the results and the parameters used.

## 7.2 Konstantinos

Firstly, one of the most important things that I learned, through a lot of trial and error, is how to structure a framework for experiments to debug possible potential problems when trying to build a full RNN model with considerable memory limitations. Another big part of the project was to understand current research around Natural Language Processing and how RNN with LSTM nodes input and process sequences of words successfully. Moreover, I had the chance to learn more about dropout, how it works and can be used in an RNN and how effective it can be. Finally, it was really exciting to work in a group on such a project of a real-life scenario application and the possible obstacles this might bring.

## 7.3 Clara

The three deep learning skills I acquired during this project are described below.

- **Long Short-Term Recurrent Neural Networks (LSTMs).** Our model was an LSTM. LSTMs are recurrent neural networks (RNNs). Just as RNNs they are used for sequential data. However, LSTMs have an advantage over vanilla RNNs since LSTMs have the ability to model long term dependencies. This is due to them having nodes with gates instead of normal recurrent nodes.
- **Word embeddings.** We encoded the words in our dataset as word embeddings, and we showed that this has a positive effect on the performance compared to binary encodings. Word embeddings are word encodings based on the words' context. This results in the fact that word embeddings which are close together in the vector space are similar in meaning.
- **Dealing with unbalanced classes.** To tackle the class imbalance of our dataset, we did a lot of research into methods to deal with unbalanced classes. One should always look at the ratio of their classes, and if there is an unbalance, accuracy is often not the best metric to optimise. There are multiple ways to deal with unbalanced classes. One group of methods is performance metrics, e.g. MCC (Matthew correlation coefficient, or F1 score. A second option is weighting your classes. A third option is creating synthetic samples for the minority class.