Prediction of Hotel Reservations Using Neural Networks

Adonis García Anés, Mario García Berenguer & Eder Tarifa Fernández

Introduction

This study aims to predict hotel reservation cancellations using neural networks. The goal is to predict a binary variable that takes two values: canceled or not canceled. The initial dataset contains 19 variables and 36,275 observations. After processing the data, we end up with 16 variables and 36,270 observations. The model will attempt to predict whether a reservation will be canceled or not using a neural network. We will explore which is the optimal neural network, and the code can be accessed through a link to a Google Colaboratory notebook. A more detailed description of this dataset can be found in the article by Nuno, Almeida, and Nunes (2019).

Process Design

First, we define the training, development, and test sets. The training set contains 70% of the observations, which is 25,389. The development set (also called validation) contains 5,440 observations. Finally, the test set, with which we evaluate the model, contains 5,441 observations. In the initial models, we focus on reducing the bias so that the network fits the training set as much as possible. Once this is achieved, we will focus on reducing the variance to allow the model to generalize adequately from the training set.

Step One: Reducing Bias

For the first model, we start with the following hyperparameter configuration:

- Number of hidden layers: 3
- Number of neurons per layer: 50, 150, 300, 150, and 80
- Mini-batch size: 512. We will maintain this batch size throughout the study as it is a power of two, which is computationally desirable. Additionally, it distributes the training data into 50 batches, which seems appropriate to maintain a balance between too many batches and too few.
- Learning rate: 0.001. This seems like an appropriate value, but we will test other learning rates.
- Number of epochs: 750. We will observe that after this epoch, or even before, the models stop improving.
- Weight initialization method: He, in Keras referred to as HeNormal. This assigns values randomly from a normal distribution centered around zero, with a standard deviation that considers the number of input units. We use seed 1243 to make the initializer deterministic, so that when the initializer is called, the weight values remain consistent across multiple tests (Keras, n.d.).
- Activation function for hidden layers: ReLU, and two variants: ELU and SELU (Ponce, 2021). All of these are activation functions that do not saturate, avoiding the problem of gradient descent.
- Activation function for the output layer: sigmoid. This function is commonly used in binary classification problems.
- Optimization method: Adam. We use this optimizer because we believe it is more comprehensive than others like Gradient Descent, Momentum, or RMSprop (Molina, 2024).
- Error functions: the chosen loss function is binary cross-entropy, which is specific to binary classification problems (Rubiales, 2021; Keras, n.d.).
- Regularization method: none.

With the command `model = Sequential()`, we declare the model. Next, we add the input, hidden, and output layers to the model using the function `model.add`. The metric used to evaluate the model's performance is `binary_accuracy`.

We employ batch normalization, which can be understood as an intermediate layer between each neuron layer that normalizes the output of each layer for the next one. Each "batch" normalization layer learns two parameters related to how the data should be scaled and shifted. This allows for further tuning the normalization to the output data. In addition to normalizing, this is a regularization strategy in itself (Durán, 2019).

For the activation function of the hidden layers, refer to Figure 1 for a graphical overview of the results obtained with ReLU, ELU, and SELU. In all models, overfitting occurs as we

achieve 98-99% accuracy in the training phase but only 86% in the development phase. This results in a 12-point difference. Techniques will be applied later to reduce this variance.

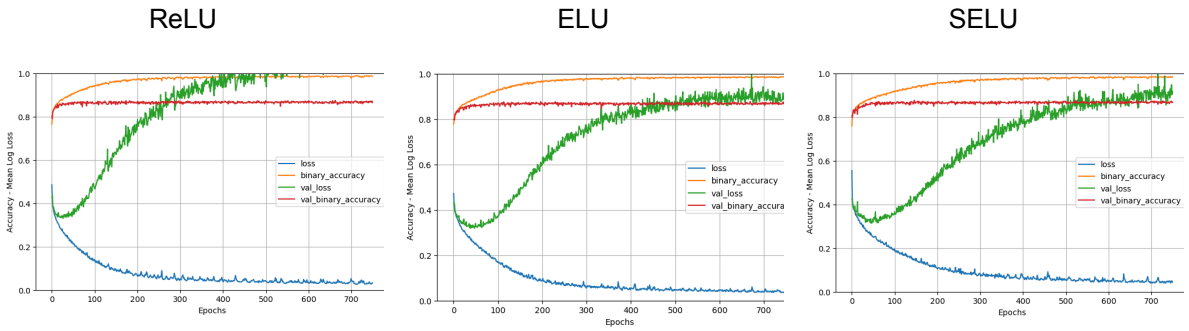

Figure 1. Graph of accuracy and loss in the training and development sets for the neural networks with ReLU, ELU, and SELU activation functions in the hidden layers.

We consider the best model among the three explored to be the one using the ELU activation function due to less overfitting. Therefore, in the following models, we use the ELU activation function for the hidden layers. Although the differences with the other models are very small (see Table 1).

	train accuracy	*dev accuracy*	*train loss*	*dev loss*
ReLU	0.9874	0.8564	0.0277	1.2380
ELU	0.9860	0.8597	0.0358	0.9824
SELU	0.9839	0.8674	0.0440	0.9167

Table 1. Results for the models with different activation functions for the hidden layers. "Train" refers to the training subset, "dev" refers to the development subset, and "loss" is the binary cross-entropy. This notation will be used for future metric presentations.

With this result, our goal for the first phase, which was to reduce the difference between the model's error and the human prediction error, is achieved.

Second Step: Reduce Variance

In this step, our goal is to correct overfitting. To do this, we will try two regularizers: L2 and dropout. We will test with different regularization strengths. The hyperparameters chosen remain the same as in the previous section. We decided not to increase the number of epochs, as we observed that around epoch 600, the values remain fairly stable.

For models using the L2 regularizer, we test with different values for the regularization factor. The variation of this factor influences the penalty applied to the weights. The values tested were: 0.001, 0.0001, 0.0005, 0.00025, 0.00035, 0.0004, 0.00045. The regularization value that achieved the lowest variance, error, and highest accuracy in both training and development was 0.00035 (see Table 2 for all results). The results are presented in the same chronological order in which they were executed. Results show that a factor that is too

high prevented the network from learning, while a factor that was too low did not correct enough. Thus, we found the optimal factor for our case.

factor	*train accuracy*	*dev accuracy*	*train loss*	*dev loss*
0.001	0.8311	0.8161	0.4160	0.4335
0.0001	0.9738	0.8661	0.1718	0.7246
0.0005	0.8627	0.8507	0.3791	0.4028
0.00025	0.9209	0.8757	0.2953	0.8757
0.00035	0.8822	0.8595	0.3527	0.8595
0.0004	0.8756	0.8569	0.3623	0.4085

Table 2. Performance metrics for models with different values for the L2 regularization factor. Highlighted in yellow is the regularization factor 0.00035 considered the best.

We consider that the results obtained with L2 are improvable, so we move on to use the other method mentioned. Regarding the dropout regularization technique, we test different elimination percentages: 2%, 5%, and 7%. The optimal percentage turned out to be 5%, as 2% regularized too little and still showed much overfitting, while 7% regularized too much, and although the variance was minimal, accuracy was affected. Therefore, we prefer to have a slight variance but better accuracy (see Table 3 for the results).

%	*train accuracy*	*dev accuracy*	*train loss*	*dev loss*
2	0.9422	0.8720	0.1442	0.4106
5	0.8967	0.8717	0.2389	0.3106
7	0.8721	0.8466	0.2857	0.3566

Table 3. Performance metrics for models with different dropout values. Highlighted in yellow is the model considered the best.

Below, we can see graphically the difference between the two best models using the L2 regularizer and dropout.

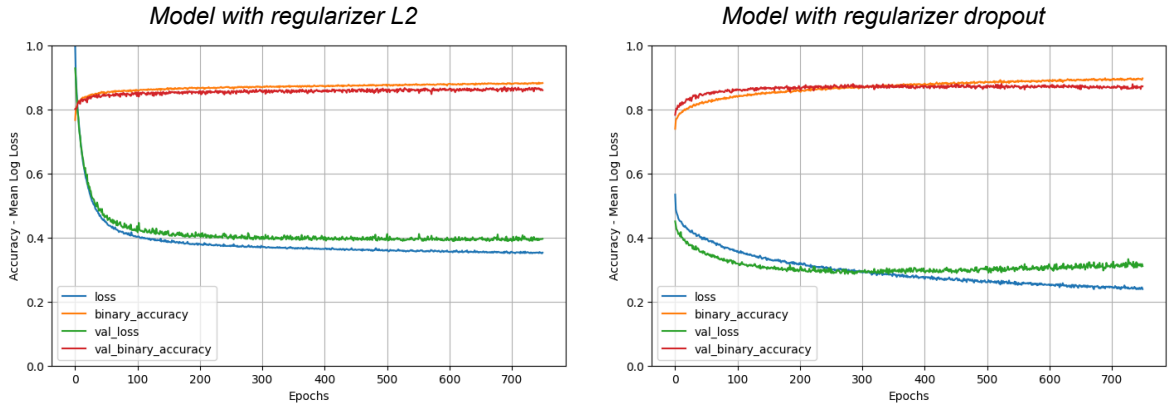

Figure 2: On the left, the model with L2 regularizer with a regularization factor of 0.00035. On the right, the model with dropout regularizer with a 5% neuron drop rate.

Comparing both, we choose the 5% dropout. However, we observe that around epoch 350, the loss starts to increase, so it would be ideal to stop training right there since the loss

begins to rise afterward. We apply Keras' EarlyStop, selecting binary cross-entropy (referred to as val_loss in Keras) as the monitored metric for the development set and stopping training when the function stops decreasing (parameter 'mode'). The function should start to grow within 50 epochs (parameter 'patience'). We chose 50 epochs because, since the function oscillates slightly, we don't want the training to stop due to a false plateau. With fewer epochs for the stop condition, it has been observed that training stops too early when there is still room to improve accuracy and the error has barely increased or stopped decreasing.

As shown in the comparison in Figure 4, we succeed in preventing the error from rising, and the accuracy results are only slightly lower. Additionally, the overfitting issue has been addressed. The model stops training at epoch 314 and achieves an accuracy value for the development subset with just a 0.05 difference compared to training for 750 epochs (see Table 3).

	train accuracy	***dev accuracy***	***train loss***	***dev loss***
without early stop	0.8967	0.8717	0.2389	0.3106
with early stop	0.8690	0.8694	0.2919	0.3030

Table 4: Performance metrics for the models with 5% dropout, comparing with and without early stopping when the loss in the training subset begins to increase.

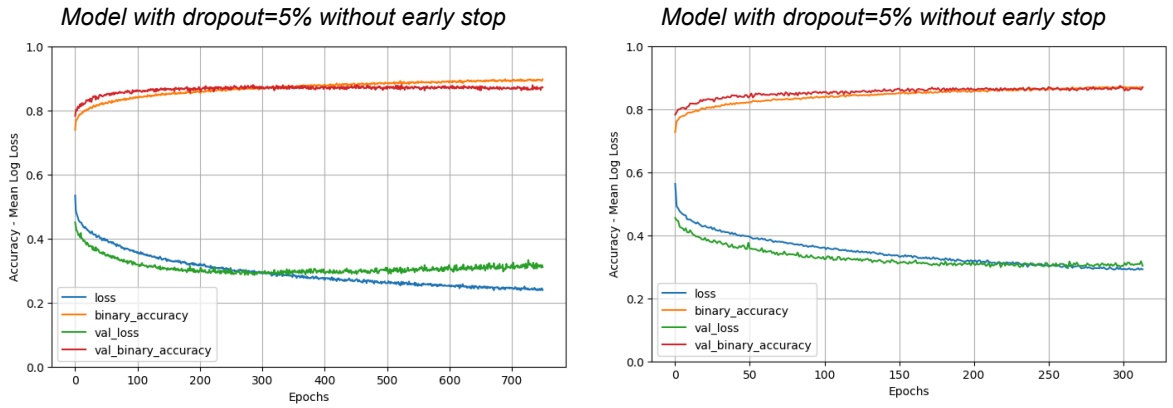

Figure 3: Both models use 5% dropout. On the left, without early stop; on the right, with early stop.

We also tested the effect of combining both regularizers. We tried dropout with 5% and L2 with values of 0.0001 and 0.00001. The combination of dropout with 5% and L2 with 0.00001 results quite similar to dropout with 5% and early stop. Although it shows slight overfitting, we continue to prefer the model with 5% dropout and early stop (see Table 4 for the results).

	train accuracy	***dev accuracy***	***train loss***	***dev loss***
dropout = 5% L2 = 0.0001	0.8508	0.8654	0.3859	0.3694
dropout = 7% L2 = 0.00001	0.8791	0.8621	0.3215	0.3566

Table 5: Performance metrics for models combining dropout and L2 regularizers.

Based on the findings, we consider the best model to be the one with 5% dropout without the stop condition, as, despite showing slight overfitting, it has great learning capacity and nearly matches the Bayesian error. This model stands out compared to the other models we tested.

Step 3: Increase predictive capacity

Step 3: Increase predictive capacity

After reducing variance as much as possible, we now aim to see if we can increase the model's predictive capacity, as, while reducing variance, the model's bias has increased. To do this, we consider increasing the number of neurons and testing different learning rates.

On one hand, reducing the learning rate to 0.0001 (instead of 0.001) yields similar results. On the other hand, increasing the learning rate to 0.1 (instead of 0.001) gives slightly worse results. Thus, we decide to stick with the learning rate initially chosen, 0.001.

Additionally, when adding more neurons, we consider that regularization may need to be different. But first, we will test the regularization that gave the best results (5% dropout).

The number of neurons in the layers was [50, 150, 300, 150, 80]. Now, the architecture is [250, 500, 700, 700, 500, 350], meaning we added one more hidden layer and increased the number of neurons per layer.

In this model, variance increases significantly, resulting in much overfitting, achieving 0.94 accuracy in training compared to 0.86 in development. Thus, we try increasing the number of neurons randomly dropped in the dropout to 7%. However, overfitting is still not fully corrected, with 0.92 accuracy in training compared to 0.85 in development. See Figure 4 for a comparison of these two models.

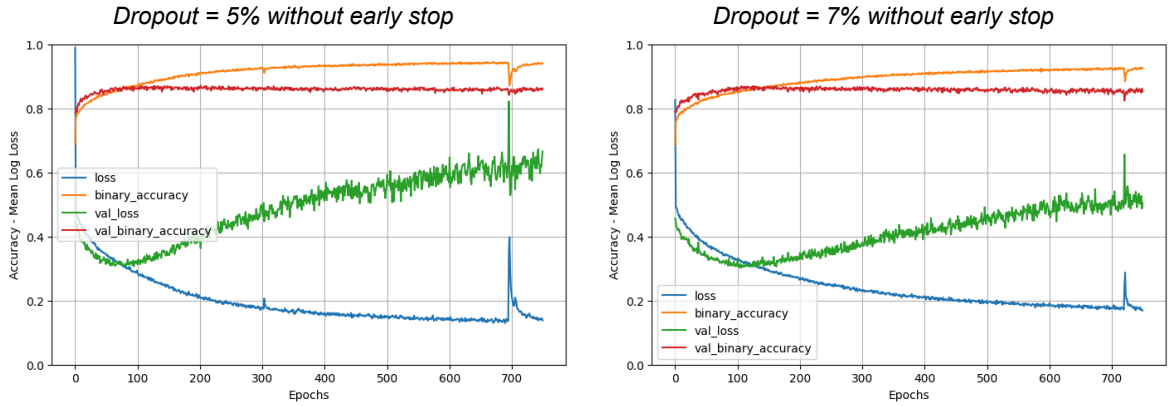

Figure 4: Graphs resulting from models with more neurons.

Since this result doesn't seem satisfactory, we try applying two regularizers: dropout = 5% and L2 = 0.0001 (another of the best models from earlier). Although variance is reduced, it remains present, with 0.87 accuracy in training compared to 0.86 in development. We apply early stop to see if we can maintain accuracy while reducing overfitting, but we do not

succeed—it drops by a tenth and variance stays the same (0.86 accuracy in training vs. 0.85 in development).

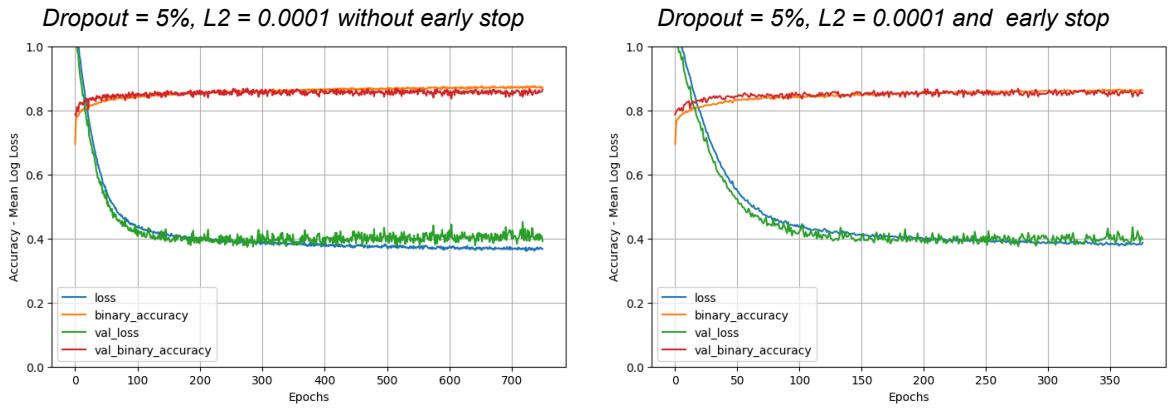

Figure 5: Graphs from models with more neurons.

In conclusion, these results do not outperform the first models we proposed, so we will stick with the initial ones—specifically, the model with 5% dropout and no early stop.

Final Results

We consider the best model to be the one using 5% dropout with this layer architecture: [50, 150, 300, 150, 80]. The other model that closely matches these results uses 5% dropout and L2 = 0.0001 with the architecture [250, 500, 700, 700, 500, 350]. However, due to the higher computational cost of the more complex architecture, we consider it less optimal overall. Also, it does not achieve the accuracy levels seen in our final model.

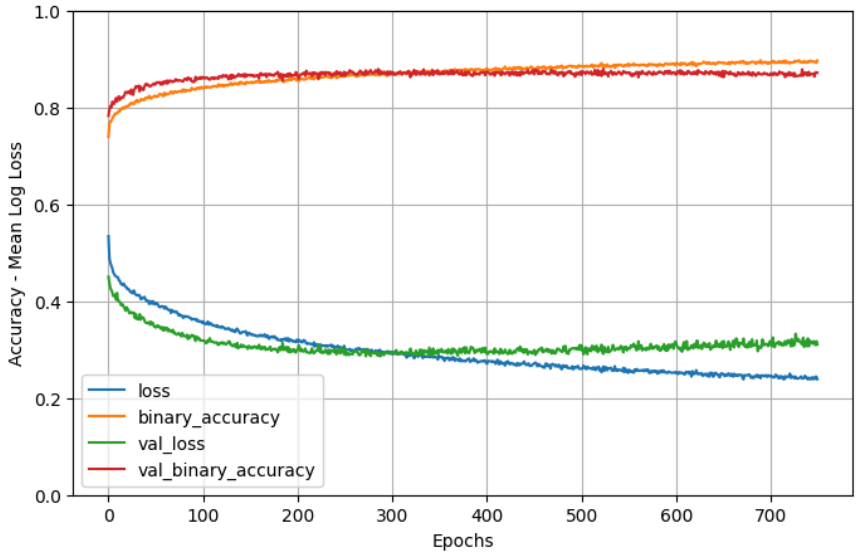

Figure 6: *Neural model with dropout regularizer (5%)*

The reason for our choice is that, although this model presents slight overfitting, the values for the loss function and accuracy are very good and surpass the other models we have

proposed. Therefore, we find it fair to accept slight overfitting to achieve a good performance.

	train accuracy	*dev accuracy*	*train loss*	*dev loss*
Dropout=5%	0.8980	0.8711	0.2364	0.3153
Dropout=5% earlystop	0.8690	0.8694	0.2919	0.3030

Table 6: Metrics for the best performing models.

Confusion matrix for the final test set:

```
                precision  recall    f1-score  support
   Cancelado=0   0.81       0.79      0.80      1785
No cancelado=1   0.90       0.91      0.90      3656
      accuracy                        0.87      5441
     macro avg   0.85       0.85      0.85      5441
  weighted avg   0.87       0.87      0.87      5441
```

We see that the "cancelled" class is predicted with only 80% f1-score, while the "not cancelled" class is predicted much better, achieving 90%. An important observation is that the classes are imbalanced. The cancellation rate is 32.7%, while the "not cancelled" rate is 67.23%. The number of observations in one class is almost double that of the other. Thus, it makes sense that the performance metrics are better for the class with more observations, as the neural network has had more examples to train on and can generalize better. Similarly, we believe that simply increasing the number of samples could considerably improve the predictive capacity of our network.

Conclusions

The different models used showed accuracy values for the training set between 0.85 and 0.96, while for the development set, the accuracy ranged from 0.83 and never exceeded the 0.875 mark. The challenge of this work was to increase the model's predictive capacity without overfitting. Regularization has been a key aspect in achieving the goal of variance reduction. To reduce bias, we tried increasing the number of neurons, but this approach was not satisfactory. Ultimately, we identified two models that were satisfactory, with the only difference being that one applied early stopping when the loss in the test set started increasing more than the loss in the training set, without a significant increase in accuracy. While early stopping eliminates overfitting, it does not allow the accuracy to reach as high a value. Finally, a table with the statistics of the models used, along with the chosen hyperparameters, can be found in the annex.

References

Durán, J. (8 de octubre de 2019). *Técnicas de Regularización Básicas para Redes Neuronales.* Medium. https://medium.com/metadatos/t%C3%A9cnicas-de-regularizaci%C3%B3n-b%C3%A1

sicas-para-redes-neuronales-b48f396924d4#:~:text=Normalizaci%C3%B3n%20por%20lotes%20(Batch%20normalization)&text=La%20normalizaci%C3%B3n%20en%20lotes%20consiste,normalizar%20las%20activaciones%20de%20salida

Keras. (s.f). *EarlyStopping*. https://keras.io/api/callbacks/early_stopping/

Keras. (s.f.). *Layer weight initializers*. https://keras.io/api/layers/initializers/

Keras. (s.f). *Probabilistic losses*. https://keras.io/api/losses/probabilistic_losses/#binaryfocalcrossentropy-class

Manrique, D. (2024). *Softmax regression* [Diapositivas]. Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid.

Molina, D. (2024). *Optimization Algorithms* [Diapositivas]. Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid.

Nuno, A., De Almeida, A., & Nunes, L. (2018). Data Article Hotelbooking demand datasets. *Data in Brief*, *22*, 41-29. https://doi.org/10.1016/j.dib.2018.11.126

Ponce, J. (10 de julio de 2021). *Conoce qué son las funciones de activación y cómo puedes crear tu función de activación usando Python, R y Tensorflow*. Jahaziel Ponce. https://jahazielponce.com/funciones-de-activacion-y-como-puedes-crear-la-tuya-usando-python-r-y-tensorflow/

Sai, M. (10 de septiembre de 2022). *Keras EarlyStopping Callback to train the Neural Networks Perfectly*. Medium. https://pub.towardsai.net/keras-earlystopping-callback-to-train-the-neural-networks-perfectly-2a3f865148f7

Rubiales, A. (4 de octubre de 2021). *Funciones de error con Entropía: Cross Entropy y Binary Cross Entropy. Medium.* https://rubialesalberto.medium.com/funciones-de-error-con-entropia-cross-entropy-y-binary-cross-entropy-8df8442cdf35

Anexo

Hiper parámetros comunes a todas los modelos excepto que se especifique lo contrario:

- Tamaño de mini-batch: 512.
- Método para inicializar los pesos de las capas: He.
- Función de activación para la capa de salida: función sigmoide.
- Método de optimización: Adam.
- Funciones de error: entropía cruzada binaria
- Tasa de aprendizaje (α): 0.001.
- Función de activación: ELU

Arquitectura capas de neuronas modelos clase A: [50, 150, 300, 150, 80]

Arquitectura capas de neuronas modelos clase B: [250, 500, 700, 700, 500, 350]

C	*Hiperparámetros que cambian*	*train accuracy*	*dev accuracy*	*train loss*	*dev loss*
A	ReLU	0.9874	0.8564	0.0277	1.2380
	ELU	0.9860	0.8597	0.0358	0.9824
	SELU	0.9839	0.8674	0.0440	0.9167
	L2=0.001	0.8311	0.8161	0.4160	0.4335
	L2=0.0001	0.9738	0.8661	0.1718	0.7246
	L2=0.0005	0.8627	0.8507	0.3791	0.4028
	L2=0.00025	0.9209	0.8757	0.2953	0.8757
	L2=0.00035	0.8822	0.8595	0.3527	0.8595
	L2=0.0004	0.8756	0.8569	0.3623	0.4085
	Dropout=2%	0.9422	0.8720	0.1442	0.4106
	Dropout=5%	0.8980	0.8711	0.2364	0.3153
	Dropout=7%	0.8721	0.8466	0.2857	0.3566
	Dropout=5% earlystop	0.8690	0.8694	0.2919	0.3030
	Dropout = 5% L2 = 0.0001	0.8508	0.8654	0.3859	0.3694
	Dropout = 7% L2 = 0.00001	0.8791	0.8621	0.3215	0.3566
	Dropout=5% earlystop α=0.01	0.8630	0.8623	0.3153	0.3097
	Dropout=5% earlystop α=0.0001	0.8661	0.8693	0.3019	0.3020
B	Dropout=5%	0.9410	0.8619	0.1378	0.6653
	Dropout=5%	0.9249	0.8525	0.1687	0.5040
	Dropout=5% L2=0.0001	0.8701	0.8647	0.3670	0.3915
	Dropout=5% L2=0.0001 earlystop	0.8625	0.8542	0.3869	0.3978