CSC311 Project Part B

Paul He, Wilson Zhang

March 2023

Contributions on Part B:

1. Paul He: Changing the Loss Function to CE and all the write up relating to it. Incorporating the question meta data and all the write up / code relating to it.

2. Wilson Zhang: Multilayer Neural Network and all the write up /code relating to it.

1. **Formal Description:** We will be extending the Neural Network algorithm by modifying the loss function. Currently, the algorithm uses Mean-Squared-Loss (MSE), we will add a way to use Cross Entropy Loss. We also want to further extend the algorithm by using the `question_meta.csv` data, which maps each question to a list of subjects that are involved.

   (a) **Background** The main objective we are trying to achieve is to Predict the correctness of students' answers to as yet unseen diagnostic question. Recall that the `is_correct` column of the data is a binary indicator whether the student's answer was correct (0 or 1).

   Currently, our objective is minimize $\sum_{\mathbf{v} \in \mathcal{S}} ||\mathbf{v} - f(\mathbf{v};\boldsymbol{\theta})||_2^2$ with respect to $\boldsymbol{\theta}$. f is defined as:

   $$f(\mathbf{v};\boldsymbol{\theta}) = h(\mathbf{W}^{(2)} g(\mathbf{W}^{(1)}\mathbf{v} + \mathbf{b}^{(\mathbf{1})}) + \mathbf{b}^{(\mathbf{2})}) \in \mathbb{R}^{N_{questions}}$$

   where $\mathbf{W}^{(1)} \in \mathbb{R}^{k \times N_{questions}}$ and $\mathbf{W}^{(2)} \in \mathbb{R}^{N_{questions} \times k}$ where $k \in \mathbb{N}$ is the latent dimension.

   For this project, I am appending each subject for each user at the end of the 1774 questions. So we will need to modify the input, $\mathbf{v} \in \mathbb{R}^{1 \times (1774+388)}$, since 1774 is the number of questions, and 388 is the number of subjects. (I will explain more on this later). The output of f will be in $\mathbb{R}^{1 \times (1774+388)}$

Since only the first 1774 index of $\mathbf{v}$ are the questions, We will compute the CE for each of first 1774 entries in $\mathbf{v}$. So, mathematically, our current objective is to minimize:

$$\sum_{\mathbf{v}\in\mathcal{S}}\sum_{i=0}^{N_{\text{q}}} -v^{(i)}\log f(v^{(i)};\theta^{(i)}) - (1-v^{(i)})\log(1-f(v^{(i)};\theta^{(i)}))$$

*Note: N_q is the number of questions, in this case, its 1774.

where the reconstruction of v is as stated from above.

I am also adding the option of using two extra layers j and i in the reconstruction of v as follows,

$$\begin{aligned} f'(\mathbf{v};\boldsymbol{\theta}) =& j(\mathbf{W}^{(4)}i(\mathbf{W}^{(3)}h(\mathbf{W}^{(2)}g(\mathbf{W}^{(1)} \\ & \mathbf{v}+\mathbf{b}^{(\mathbf{1})})+\mathbf{b}^{(\mathbf{2})})+\mathbf{b}^{(\mathbf{3})})+\mathbf{b}^{(\mathbf{4})}) \end{aligned}$$

*Note: W and b are the weight matrices and bias vectors for their respective layers.

However, the primary focus of our modifications is on the implementation of metadata using cross-entropy.

Also we will calculate accuracy as:

$$\text{Prediction Accuracy} = \frac{\text{The number of correct predictions}}{\text{The number of total predictions}}$$

(b) **Motivation** There are a few main reasons to why we want to make these changes.

i. It is a more natural fit for the question we are doing. Since we are constructing the value of a single question that is either 0 and 1, using MSE might not be optimal since MSE is designed for regression tasks where the target is a continuous variable.

ii. Cross entropy encourages incorrect values further away from the target to be punished more. CE loss can be adapted to handle class imbalance by adjusting the class weights. This is important in situations where the classes are not evenly distributed in the dataset.

iii. In real life, if a student does well on Math Questions, but not well on History questions, given a random unseen question, they should have a higher probability of answering it correctly if it was a math question compared to a history question. This is why we

believe it is important to include the subject information related to each question.

iv. Our baseline model contains 2 layers, and adding more layers allows the neural network to learn more complex and abstract representations of the input data. Each layer can learn to extract more specific features of the data, and the combination of these features in multiple layers can lead to better representations of the input data. Since we are adding a few extra layers, we expect the model to capture more complex functions neatly without potentially overfitting the data.

(c) **Algorithm**

Algorithm Box

i. In `utils.py`, modify it to read the question meta file. Then, for each student in the training data, we append a vector of size (1 x 388) to the current size (1 x 1774).

ii. We can now train the model by passing into our AutoEncoder (optionally use the extra layers).

iii. Compute the cross-entropy loss for the first 1774 indexes of the input.

iv. Repeat for all the students.

v. Tune hypeparemeters, evaluate the model.

(d) **Hypothesis**

Hypothesis

We suspect that the training accuracy and validation will both increases, but not by a significant amount. The cross entropy will be a better loss function for each index, extra layers may capture more complex patterns in our dataset, and the neural network has extra information about each student.

2. **Figure** See Figure 1 for the idea of data process and what the new input is. See Figure 6 at the end of the report, which shows the computation flow of our training model generated by torchviz.

3. **Comparison or Demonstration** After tuning hyperparameters, the base model results in the following data:
`Best k: 50, Best lr: 0.01, Best num epoch: 17`
`best lambda: 0.01, Valid acc: 0.6840248377081569, Test acc: 0.6768275472763196`

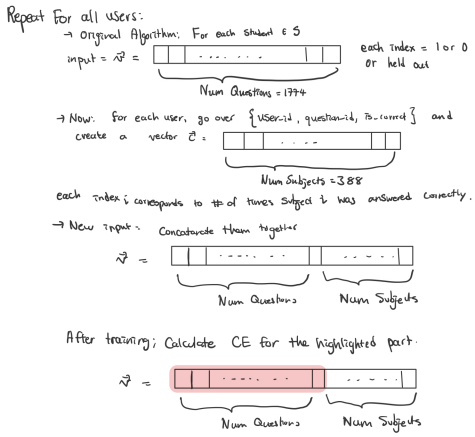

Figure 1: Figure of Data Processing

I will do it step by step. First, we change the Loss Function only. After tuning hyperparemeters, we get the result of:
`Final Validation Acc: 0.6913632514817951, Test acc: 0.6900931414055885`
This means that, there is a tiny increase in both validation and test accuracy.

Lets compare the graphs for accuracy and epoch for the final model (see figure 3 for base model, figure 4 for CE model). You may ask, it seems like the graph is still increasing, but this is the tuned version already, the final epoch is the maximum of the function (even if there are more epochs to come).
So the first thing that I see is that the CE model requires less epochs to reach the same accuracy (17 vs 47)
Lets take a look at the training cost for the first 3 iterations: This is for the base model:

```
Epoch: 0 Training Cost: 15440.621415 Valid Acc: 0.6107818233135761
Epoch: 1 Training Cost: 14282.521825 Valid Acc: 0.6212249506068304
Epoch: 2 Training Cost: 13814.365648 Valid Acc: 0.6253175275190517
```

And this is for the CE model:

```
Epoch: 0 Training Cost: 37904.692242 Valid Acc: 0.626728760937059
Epoch: 1 Training Cost: 35800.922309 Valid Acc: 0.628281117696867
Epoch: 2 Training Cost: 35013.940387 Valid Acc: 0.635196161445103
```

It seems like the training cost for the CE is higher, per epoch. So if we value accuracy and speed, then the CE model is better, but if we want to reduce the cost, MSE is better. However, the difference in both models are not significant, but our hypothesis was correct.

With multilayer:

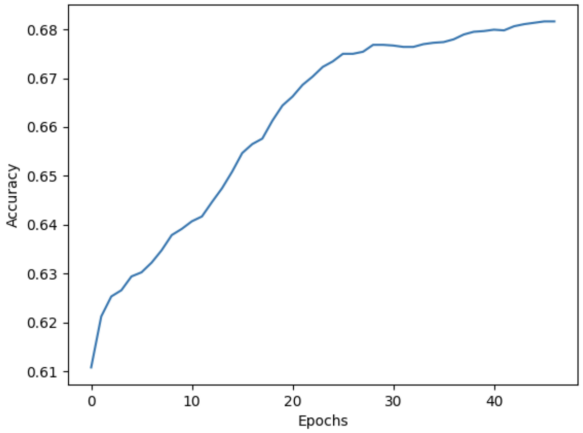

Figure 2: Base Model Test Accuracy for each epoch

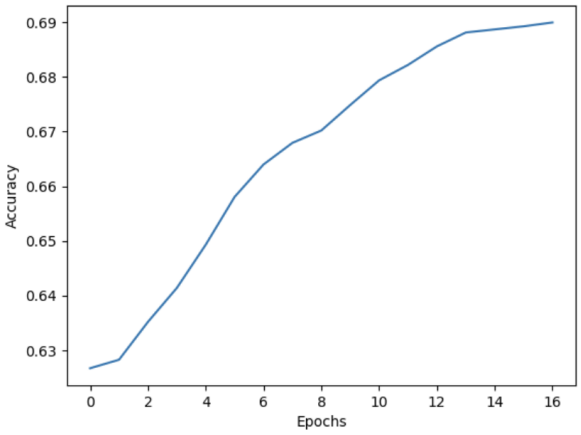

Figure 3: CE Model Test Accuracy for each epoch

`Best k: 50, Best lr: 0.01, Best num_epoch: 23, best_lambda: 0.00001, Valid acc: 0.6285633643804686, Test acc: 0.6255997742026531` Our accuracy decreased by a noticeable amount around (6-8%) However, there are some key things to note: the number of epochs needed to reach peak accuracy is around 23 (this is around the same time it took CE to get the same accuracy with 17 epochs), accuracy at early epochs is significantly poor (but spikes up quickly). See the figure 4 for more details
`Epoch: 0 Training Cost: 39055.869842 Valid Acc: 0.5517922664408693`

```
Epoch:  1 Training Cost:  37259.288455 Valid Acc:  0.6160033869602032
Epoch:  2 Training Cost:  36282.412107 Valid Acc:  0.6232006773920407
```

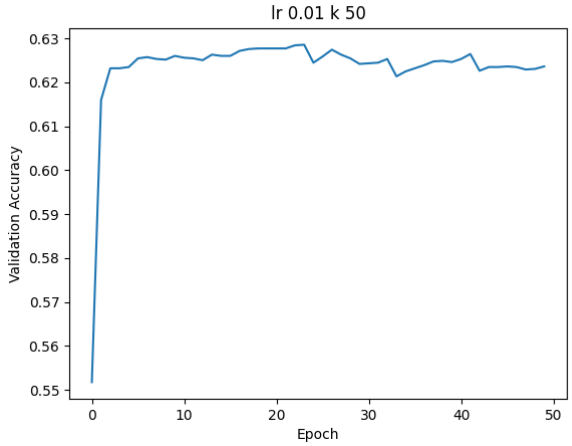

Figure 4: CE Model Test Accuracy for each epoch

With meta data:
`Best k:  10, Best lr:  0.1, Best num_epoch:  48,`
`best_lambda:  0.01, Valid acc:  0.6241885407846458, Test acc:  0.624611910810048`
It seems like our hypothesis were wrong, and the accuracy decreased a lot. Lets take a look at the graph (see figure 5):

It looks like the validation accuracy is quite consistent, (note the range displayed in this graph). The latent dimension and learning rate for this accuracy however, is very different compared to the other models. The model might be trying to overfit the data, hence the decrease in accuracy.

4. **Limitations**: Our hypothesis were correct for CE, but it was incorrect for meta data. I believe using the meta data should increase the accuracy, but we did not implement it correctly.

   (a) Limitations:

      i. The first limitation is the way I have concatenated the extra data, and then only updating the linear function g in the auto encoder to take in extra inputs. This means that the model is not properly learning and comparing based on the new data. (For the first 1774 indexes, we are not passing in the subject information that does this). This limitation can be replicated by using a small learning rate such and large latency dimension, and the accuracy was around 50% only. To be more specific: for each

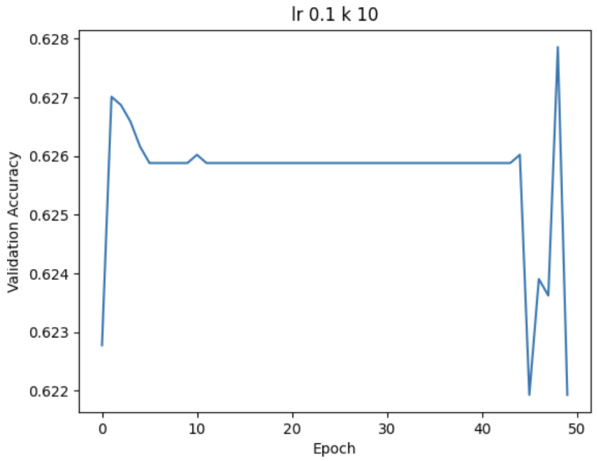

Figure 5: Base Model Test Accuracy for each epoch with modified input (meta data) (note: on the last minute, the y-axis should be test-accuracy

student, we did not find a way to compare the 1774 questions category to the student's strength, I could've created a $399 \times (1774 + 388)$ matrix, but that would take forever.

ii. The function `add_subjects` has a long running time. Everything is a dictionary, we only have around 500 users, if the number of users increase, it would take too long to process the data.

iii. The multilayer implementation was given a strict bound of 4 layers. This could imply that the model is overfitting the data, or is not perfectly fine-tuned

iv. There needs to be some sort of loss function for the 388 columns in the end of the input too. They're just kinda there right now.

v. The cost of this method is too high.

(b) Possible Extensions:

i. We create a multi-dimension representation of the input. Each question would have a vector assigned to it in the 388 dimension. Now, each question is directly correlated to the subjects. But the problem is this dimension is too high, so we can perform PCA by reducing the data to a lower dimension (since right now lots of subjects can be categorized together). This is definitely something I will try after final exams (due to time constraints, I couldn't implement this time).

ii. Use two different Loss functions, if we use the current model, we can use CE on the first 1774 indices and MSE on the 388 indices after.

iii. Use a language processing framework such as BERK and include the subject to actual english description to learn the model.

iv. We could allow for tuning based on a variable number of layers. This would allow us to fine-tune our parameters based on the layer chosen, which will likely result in a visible increase in accuracy.

5. **References:**

Grosse, R. (n.d.). Lecture 3, Part 2: Training a Classifier. Retrieved from https://www.cs.toronto.edu/ michael/teaching/csc311_w23/readings/TrainingAClassifer.pdf

Grosse, R. (n.d.). Lecture 5: Multilayer Perceptrons Retrieved from https://www.cs.toronto.edu/ michael/teaching/csc311_w23/readings/mlps.pdf

6. **Extra**

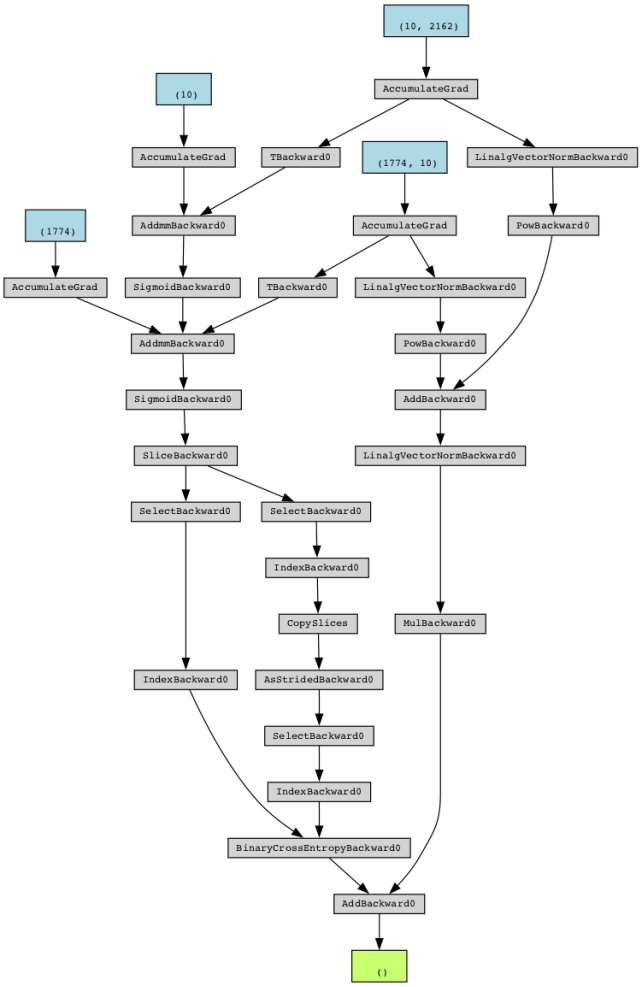

Figure 6: Computation Flow generated by torchviz