**Martian Lawyers Club**

**Fixing the bug**

The reason why the network is not training is because the operation torch.argmax is not differentiable.

**Improvements made – model**

We make the following modifications to the model:

* We replace the layers with transformer layers which are state of the art for NLP. We want the highest quality model as possible, so I figured why not use transformer layers. Sorry if this goes outside what it was asked, I kind of got carried away and transformers are state of the art.
* We replace the loss with a more appropriate cross-entropy loss.

**Improvements made – main.py**

We make the following modifications:

* Increase number of iterations to 5000. We noticed that the model did not converge for 1000.
* Increase the batch size to 128. This along with the increase in batch size makes sure that we reach a sufficient amount of epochs. Input data is 40k lines with 70/30 split so 5000 iters \* 128 batch seemed sufficient. With a larger batch we also ensure more stability during training.
* Cross validate over number of layers, heads, embeddings and learning rates and save the results.

**Results**

lr: 0.0002, layers: 1, n\_embs: 24, n\_heads 1 ------- accuracy: 0.3488462462462462

lr: 0.0002, layers: 1, n\_embs: 24, n\_heads 3 ------- accuracy: 0.35227267267267265

lr: 0.0002, layers: 1, n\_embs: 24, n\_heads 6 ------- accuracy: 0.3524744744744745

lr: 0.0002, layers: 1, n\_embs: 48, n\_heads 1 ------- accuracy: 0.373439039039039

lr: 0.0002, layers: 1, n\_embs: 48, n\_heads 3 ------- accuracy: 0.38656216216216216

lr: 0.0002, layers: 1, n\_embs: 48, n\_heads 6 ------- accuracy: 0.38928168168168165

lr: 0.0002, layers: 1, n\_embs: 96, n\_heads 1 ------- accuracy: 0.40396606606606605

lr: 0.0002, layers: 1, n\_embs: 96, n\_heads 3 ------- accuracy: 0.41771861861861864

lr: 0.0002, layers: 1, n\_embs: 96, n\_heads 6 ------- accuracy: 0.4185279279279279

lr: 0.0002, layers: 3, n\_embs: 24, n\_heads 1 ------- accuracy: 0.3647282282282282

lr: 0.0002, layers: 3, n\_embs: 24, n\_heads 3 ------- accuracy: 0.3672051051051051

lr: 0.0002, layers: 3, n\_embs: 24, n\_heads 6 ------- accuracy: 0.3681603603603603

lr: 0.0002, layers: 3, n\_embs: 48, n\_heads 1 ------- accuracy: 0.39934864864864866

lr: 0.0002, layers: 3, n\_embs: 48, n\_heads 3 ------- accuracy: 0.40519069069069064

lr: 0.0002, layers: 3, n\_embs: 48, n\_heads 6 ------- accuracy: 0.4079282282282283

lr: 0.0002, layers: 3, n\_embs: 96, n\_heads 1 ------- accuracy: 0.4286444444444445

lr: 0.0002, layers: 3, n\_embs: 96, n\_heads 3 ------- accuracy: 0.43626306306306306

lr: 0.0002, layers: 3, n\_embs: 96, n\_heads 6 ------- accuracy: 0.4357342342342342

lr: 0.0002, layers: 4, n\_embs: 24, n\_heads 1 ------- accuracy: 0.36372672672672673

lr: 0.0002, layers: 4, n\_embs: 24, n\_heads 3 ------- accuracy: 0.37206276276276273

lr: 0.0002, layers: 4, n\_embs: 24, n\_heads 6 ------- accuracy: 0.37137117117117113

lr: 0.0002, layers: 4, n\_embs: 48, n\_heads 1 ------- accuracy: 0.4067135135135135

lr: 0.0002, layers: 4, n\_embs: 48, n\_heads 3 ------- accuracy: 0.4060264264264264

lr: 0.0002, layers: 4, n\_embs: 48, n\_heads 6 ------- accuracy: 0.4101393393393394

lr: 0.0002, layers: 4, n\_embs: 96, n\_heads 1 ------- accuracy: 0.4303144144144144

lr: 0.0002, layers: 4, n\_embs: 96, n\_heads 3 ------- accuracy: 0.4344825825825826

lr: 0.0002, layers: 4, n\_embs: 96, n\_heads 6 ------- accuracy: 0.4361861861861862

lr: 0.0002, layers: 6, n\_embs: 24, n\_heads 1 ------- accuracy: 0.36870480480480483

lr: 0.0002, layers: 6, n\_embs: 24, n\_heads 3 ------- accuracy: 0.37297267267267264

lr: 0.0002, layers: 6, n\_embs: 24, n\_heads 6 ------- accuracy: 0.3755531531531532

lr: 0.0002, layers: 6, n\_embs: 48, n\_heads 1 ------- accuracy: 0.40771501501501506

lr: 0.0002, layers: 6, n\_embs: 48, n\_heads 3 ------- accuracy: 0.41118138138138133

lr: 0.0002, layers: 6, n\_embs: 48, n\_heads 6 ------- accuracy: 0.41446516516516524

lr: 0.0002, layers: 6, n\_embs: 96, n\_heads 1 ------- accuracy: 0.43512462462462465

lr: 0.0002, layers: 6, n\_embs: 96, n\_heads 3 ------- accuracy: 0.4402054054054053

lr: 0.0002, layers: 6, n\_embs: 96, n\_heads 6 ------- accuracy: 0.44053933933933925

lr: 0.0003, layers: 1, n\_embs: 24, n\_heads 1 ------- accuracy: 0.3488462462462462

lr: 0.0003, layers: 1, n\_embs: 24, n\_heads 3 ------- accuracy: 0.35227267267267265

lr: 0.0003, layers: 1, n\_embs: 24, n\_heads 6 ------- accuracy: 0.3524744744744745

lr: 0.0003, layers: 1, n\_embs: 48, n\_heads 1 ------- accuracy: 0.373439039039039

lr: 0.0003, layers: 1, n\_embs: 48, n\_heads 3 ------- accuracy: 0.38656216216216216

lr: 0.0003, layers: 1, n\_embs: 48, n\_heads 6 ------- accuracy: 0.38928168168168165

lr: 0.0003, layers: 1, n\_embs: 96, n\_heads 1 ------- accuracy: 0.40396606606606605

lr: 0.0003, layers: 1, n\_embs: 96, n\_heads 3 ------- accuracy: 0.41771861861861864

lr: 0.0003, layers: 1, n\_embs: 96, n\_heads 6 ------- accuracy: 0.4185279279279279

lr: 0.0003, layers: 3, n\_embs: 24, n\_heads 1 ------- accuracy: 0.3647282282282282

lr: 0.0003, layers: 3, n\_embs: 24, n\_heads 3 ------- accuracy: 0.3672051051051051

lr: 0.0003, layers: 3, n\_embs: 24, n\_heads 6 ------- accuracy: 0.368160360360360d3

lr: 0.0003, layers: 3, n\_embs: 48, n\_heads 1 ------- accuracy: 0.39934864864864866

lr: 0.0003, layers: 3, n\_embs: 48, n\_heads 3 ------- accuracy: 0.40519069069069064

lr: 0.0003, layers: 3, n\_embs: 48, n\_heads 6 ------- accuracy: 0.4079282282282283

lr: 0.0003, layers: 3, n\_embs: 96, n\_heads 1 ------- accuracy: 0.4286444444444445

lr: 0.0003, layers: 3, n\_embs: 96, n\_heads 3 ------- accuracy: 0.43626306306306306

lr: 0.0003, layers: 3, n\_embs: 96, n\_heads 6 ------- accuracy: 0.4357342342342342

lr: 0.0003, layers: 4, n\_embs: 24, n\_heads 1 ------- accuracy: 0.36372672672672673

lr: 0.0003, layers: 4, n\_embs: 24, n\_heads 3 ------- accuracy: 0.37206276276276273

lr: 0.0003, layers: 4, n\_embs: 24, n\_heads 6 ------- accuracy: 0.37137117117117113

lr: 0.0003, layers: 4, n\_embs: 48, n\_heads 1 ------- accuracy: 0.4067135135135135

lr: 0.0003, layers: 4, n\_embs: 48, n\_heads 3 ------- accuracy: 0.4060264264264264

lr: 0.0003, layers: 4, n\_embs: 48, n\_heads 6 ------- accuracy: 0.4101393393393394

lr: 0.0003, layers: 4, n\_embs: 96, n\_heads 1 ------- accuracy: 0.4303144144144144

lr: 0.0003, layers: 4, n\_embs: 96, n\_heads 3 ------- accuracy: 0.4344825825825826

lr: 0.0003, layers: 4, n\_embs: 96, n\_heads 6 ------- accuracy: 0.4361861861861862

lr: 0.0003, layers: 6, n\_embs: 24, n\_heads 1 ------- accuracy: 0.36870480480480483

lr: 0.0003, layers: 6, n\_embs: 24, n\_heads 3 ------- accuracy: 0.37297267267267264

lr: 0.0003, layers: 6, n\_embs: 24, n\_heads 6 ------- accuracy: 0.3755531531531532

lr: 0.0003, layers: 6, n\_embs: 48, n\_heads 1 ------- accuracy: 0.40771501501501506

lr: 0.0003, layers: 6, n\_embs: 48, n\_heads 3 ------- accuracy: 0.41118138138138133

lr: 0.0003, layers: 6, n\_embs: 48, n\_heads 6 ------- accuracy: 0.41446516516516524

lr: 0.0003, layers: 6, n\_embs: 96, n\_heads 1 ------- accuracy: 0.43512462462462465

lr: 0.0003, layers: 6, n\_embs: 96, n\_heads 3 ------- accuracy: 0.4402054054054053

lr: 0.0003, layers: 6, n\_embs: 96, n\_heads 6 ------- accuracy: 0.44053933933933925

**Discussion and Limitations**

From the results above we see that learning rate doesn’t have much effect on validation accuracy most likely due to the adaptive optimizer. In terms of network architecture, number of heads has a small impact. This makes sense because the attention task is not complex, the context is small (only 10 characters) so we do not need a lot of heads. Embedding number however has a large impact which also makes sense because it has do with the size of the word embedding. More embeddings means more capacity and 24, 48 are very small embedding sizes.  
Number of layers has an impact however not nearly as much as embedding number. The highest performing model is:  
  
lr: 0.0003, layers: 6, n\_embs: 96, n\_heads 6 ------- accuracy: 0.44053933933933925

We might want to further increase the embedding size in future work as this is the most important parameter. Number of heads and layers are also important but not nearly as much. Still we can increase them as performance has not capped yet (the last model is the highest performing one). While the conclusions drawn above should hold, accuracy is not always a good validation metric.  
  
*For instance we noticed that for the larger networks (more layers) the quality of the output sentence was high and this is not reflected in the accuracy.* We might want to use the loss if we want to check convergence and performance in a general sense (although this is still character level), but as the purpose of this is to generate sentences, we might want to use a metric that reflects that (or human evaluation) at least for testing (see bonus below). In terms of architectural decision, we choose state of the art architecture (Attention, Gelu etc) same as for larger models. These might not be necessary but we decided to go with this regardless instead of sticking with the architecture provided. This is because we want to provide the highest quality model we can.

**Next Steps – Other improvements**

We could not modify trainer or dataset nor we could build a more robust training pipeline.  
Normally I would set up the dataset, the trainer and the evaluation of the model (especially during training) so that I have a very good understanding of what happens. In this case everything was set up for me and I was not supposed to touch the trainer.py or dataset.py

We suggest other modifications that could have been useful but outside the scope of this:

* Evaluate validation loss along with training loss during training to make sure the model is not overfitting/underfitting (maybe use Tensorboard).
* use a pretrained model and then finetune instead of training from scratch.
* Better evaluation (more metrics and maybe on a separate dataset) and stopping criterion.
* maybe look at augmentations techniques (use ChatGPT to augment the data?).
* be more careful in the dataset split, maybe try cross-validation.
* implements more complicated training tricks.

**Bonus Points**

1)Word level embedding has the drawback that there are many more words then characters so the input hidden dimensions would be much higher. In addition, if a word is not part of the training set, we cannot predict on that. The advantage is that we make the task easier for the network and also the size of the input sequence is smaller (one word is one input). Generally better if the task is complex.

2) To evaluate sentence level quality we could use ROUGE or METEOR. The drawback is that we need an appropriate dataset for this. We can also use something like BERTScore so we do not have to gather a dataset (I think this could be the best option). Otherwise we could use log likelihood/ perplexity but that has more to do with the convergence of the model and it is still character level. Still it might be better to use for validation as the purpose it to pick the overall best model. Then for testing we could use BERTScore.

3) We could implement a stopping criterion such that if the moving average of the loss is unchanged over a period of iterations then we stop training. I did not need to implement this as I found that increasing iterations to 5000 was enough to ensure convergence due to weight decay.