Assignment 3

CMSC 678 — Introduction to Machine Learning

Faisal Rasheed Khan

VB02734

vb02734@umbc.edu

Question 1----------

1.a

We want to use MSE loss if we have continuous data. MSE loss evaluates how closely is the data related, as it computes squared difference of predicted data from target data. MSE loss is typically used for Regression problems where data is continuous. Whereas we use per-component binary cross entropy loss if the data is categorical i.e., given data have labels with them. per-component binary cross entropy loss maximizes the likelihood of a particular label over the distribution. So for categorical data per-component binary cross entropy loss increases log-likelihood by minimizing the entropy loss.

1.b

$x=(1,0)^T$, W=(0.5,-0.5), U=((0.3),(-0.3))

$\mathcal{E}$: z = f(W x); f is sigmoid function, f = $\frac{1}{1+e^{-x}}$

z = f($(0.5 \quad -0.5)\begin{pmatrix}1\\0\end{pmatrix}$)

z = f(0.5-0)

z = f(0.5)

z = $\frac{1}{1+e^{-0.5}}$

z = 0.622

D : $\hat{x}$ = g(Uz); g is sigmoid function, g = $\frac{1}{1+e^{-x}}$

$\hat{x}$ = g($\begin{pmatrix}0.3\\-0.3\end{pmatrix}$ (0.622))

$\hat{x}$ = g($\begin{pmatrix}0.3*0.622\\-0.3*0.622\end{pmatrix}$)

$\hat{x}$ = g($\begin{pmatrix}0.1866\\-0.1866\end{pmatrix}$)

$\hat{x}$ = $\begin{pmatrix} 0.546 \\ 0.454 \end{pmatrix}$

$L_{xent}(x, \hat{x}) = \sum_d CE(x_d, \hat{x}_d)$ where CE is Cross Entropy

$L_{xent}(x, \hat{x}) = \sum_d \mathrm{CE}(x_d, \hat{x}_d)$

$L_{xent}(x, \hat{x}) = \sum_d -(x_d \log \hat{x}_d + (1- x_d) \log (1 - \hat{x}_d))$

$= -(1 \log 0.546 + (1- 1) \log (1 - 0.546)) - (0 \log 0.454 + (1- 0) \log (1 - 0.454))$

$= - \log 0.546 - \log 0.546$

$= 2 * 0.609$

$= 1.21$

1.c

The parameters to be learned are weights of the reduced dimensionality and the reconstructed dimensionality which are W, U. If biases are there for reduced representation and reconstructed representation, then they also should be learned.

1.d

D=100, E=50

ε : z = f(W x)

x = (100 x 1) shape

Then W must have columns as of the x dimension D and rows of W will have z dimension

W = (50 x 100) shape

z = (50 x 1) shape

U should have x dimension for rows in order to reconstruct back the input and columns will have dimensions of z

U = (100 x 50) shape

Total parameters = W + U

= (50 x 100) + (100 x 50)

= 10000

If bias terms are there then add E+D to the above parameters.

Total parameters with bias = 10000 + 100 + 50

= 10150

1.e

$L_{xent}(x, \hat{x}) = \sum_d CE(x_d, \hat{x}_d)$ where CE is Cross Entropy

$L_{xent}(x, \hat{x}) = \sum_d -(x_d \log \hat{x}_d + (1- x_d) \log (1 - \hat{x}_d))$

x_d

$\hat{x}_d = f(W x) = \frac{1}{1+e^{-W \hat{x}d}}$

$L_{xent}(x, \hat{x}) = \sum_d -(x_d \log \frac{1}{1+e^{-W \hat{x}d}} + (1- x_d) \log (1 - \frac{1}{1+e^{-W \hat{x}d}}))$

$\frac{\partial(\text{Lxent}(x,\hat{x}))}{\partial(W)} = \frac{\partial(\text{Lxent}(x,\hat{x}))}{\partial(xd)} \frac{\partial(\text{xd})}{\partial(\hat{x}d)} \frac{\partial(\hat{x}d)}{\partial(W)}$

$x_d = f(W x) = \frac{1}{1+e^{-W xd}}$

$\hat{x}_d = g(U z) = \frac{1}{1+e^{-U \hat{x}d}}$

$L_{xent}(x, \hat{x}) = \sum_d -(\frac{1}{1+e^{-W xd}} \log \frac{1}{1+e^{-U \hat{x}d}} + (1- \frac{1}{1+e^{-W \hat{x}d}}) \log (1 - \frac{1}{1+e^{-U \hat{x}d}}))$

$\frac{\partial(\text{Lxent}(x,\hat{x}))}{\partial(U)} = \frac{\partial(\text{Lxent}(x,\hat{x}))}{\partial(xd)} \frac{\partial(\text{xd})}{\partial(\hat{x}d)} \frac{\partial(\hat{x}d)}{\partial(W)} \frac{\partial(\text{W})}{\partial(U)}$

1.f

We can add bias term with minimal effort to the linear layers of the reduced representation and the reconstructed representation of the input before sending that linear layer to the activation function which we are using here is sigmoid.

i.e.

ε : $z = f(W x + b_0)$; f is sigmoid function

D : $\hat{x}$ = g(Uz); g is sigmoid function

Question 2----------

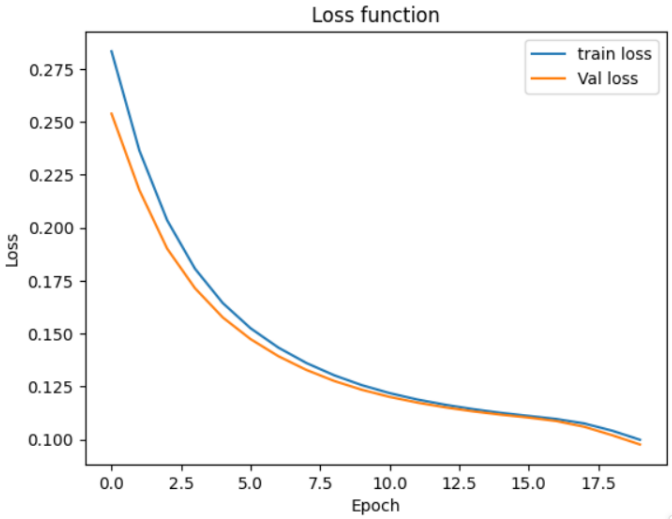
2.

Implementation: Auto encoder and decoder, encoder should reduce the large number of features to lower representation and decoder should restore back the lower features to the original features without any loss. A classifier is required to train the model. Based on various different configurations of model and applying it on dev data gives us many observations like what learning rate to choose, what activation function to choose and many more parameters. We will see our model accuracy on baseline classifier, mostly I consider logistic regression.

The autoencoder was trained to reduce the dimensionality of the input from its original size of 784 to a lower dimensionality of 64. The autoencoder was then trained on the reconstructed dimensionality of 784 features from the lower dimensionality of 64 features.

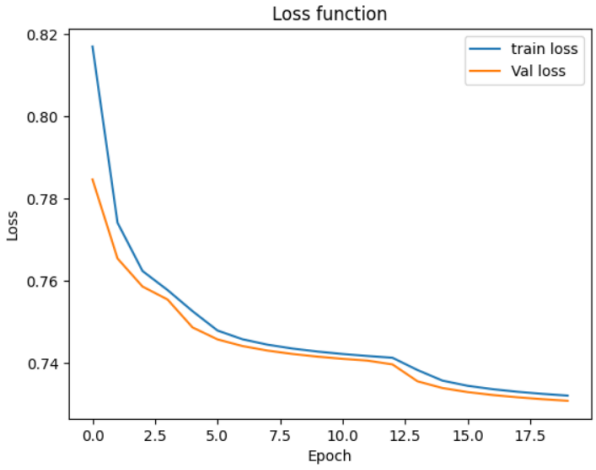
i. Sigmoid Activation function, epochs=20, batches=100, default learning rate of Adam Optimizer, loss function = Binary Cross Entropy, bias= False
Training Loss: 0.099
Validation Loss: 0.097
The model converged well below is the loss graph on training and validation

```
Epoch 19
Epoch loss: 0.099843
Val loss: 0.097598
```



ii. Softmax Activation function, learning rate = 0.01, and remaining same as above
Training Loss: 0.732, Validation Loss: 0.730
Perform worse compared to the above sigmoid activation function

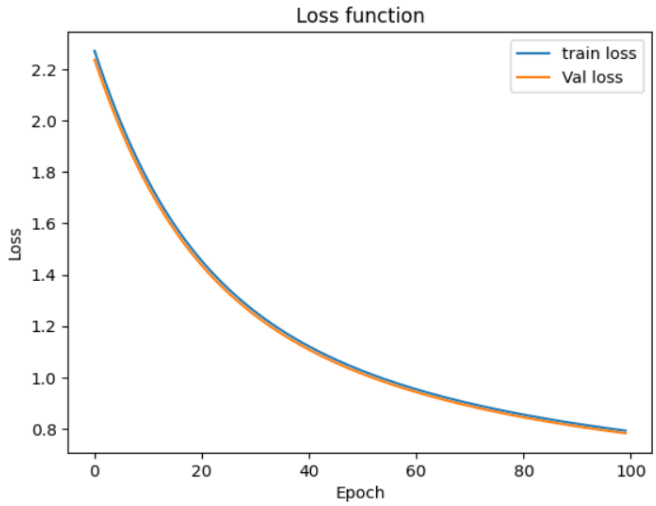
```
Epoch 19
Epoch loss: 0.732151
Val loss: 0.730894
```



Encode to 64 from 784 features and decode to 784 from 64 features

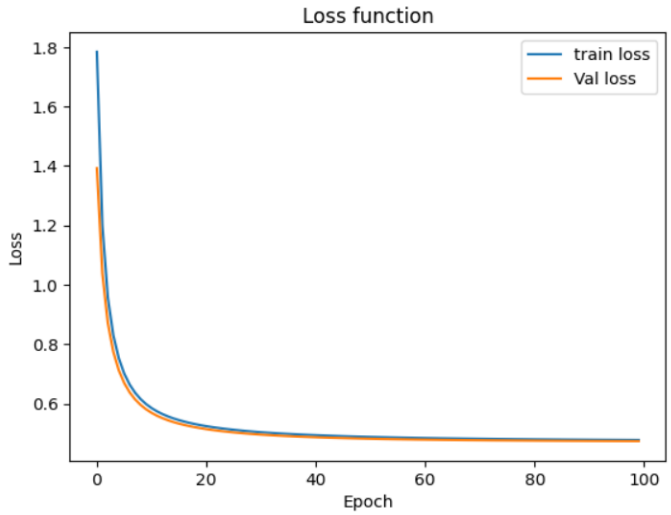
i. Tanh Activation function to encode and Sigmoid activation function to decode, epochs=100, batches=80, default learning rate of Adam Optimizer, loss function = Cross Entropy, bias= False
Training Loss: 0.793, Validation Loss: 0.78, f1_score = 0.789
Accuracy on test data: 0.8034
The model converged well below is the loss graph on training and validation

```
f1 score: 0.7987
Epoch loss: 0.793068
Val loss: 0.782949
<ipython-input-300-7e5eb1206d01>:30: UserWarning: Implicit dimension choice
  z = fn(h)
```



ii. ReLU Activation function to encode and Tanh activation function to decode, epochs=100, batches=80, default learning rate of Adam Optimizer, loss function = Cross Entropy, bias= False
Training Loss: 0.47, Validation Loss: 0.44, f1_score = 0.86, Accuracy on test data: 0.8599
The model converged better than the above setting.

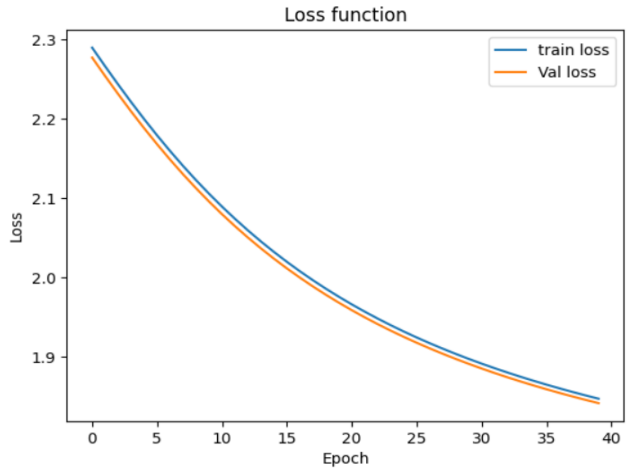
```
f1 score: 0.8607
Epoch loss: 0.477520
Val loss: 0.473631
<ipython-input-300-7e5eb1206d01>:30: UserWarning: Implicit dimension choice f
  z = fn(h)
```



iii. For both encode and decode sigmoid activation, epochs=40, batches=60, learning rate for adam optimizer=0.01 and remaining all same as above.

Training Loss:1.846, Validation Loss:1.841, f1_score:0.8123
Accuracy on test data: 0.8034
The losses are more compared to i. but still this has a very good f1_score. With less epochs than i. it still has good score for dev.

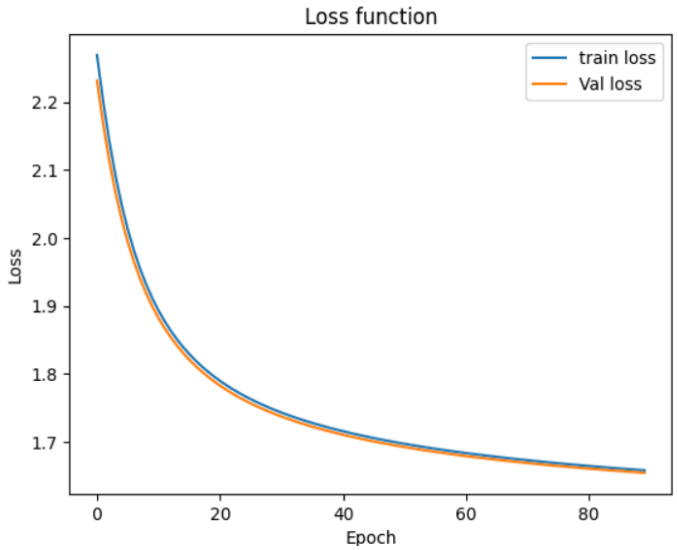
```
f1 score: 0.8123
Epoch loss: 1.846979
Val loss: 1.841356
<ipython-input-251-2d3f35457ae4>:44: UserWarning: Implicit dimension choice
  z = fn(h)
```



Encode to 256 from 784 features and decode to 784 from 256 features

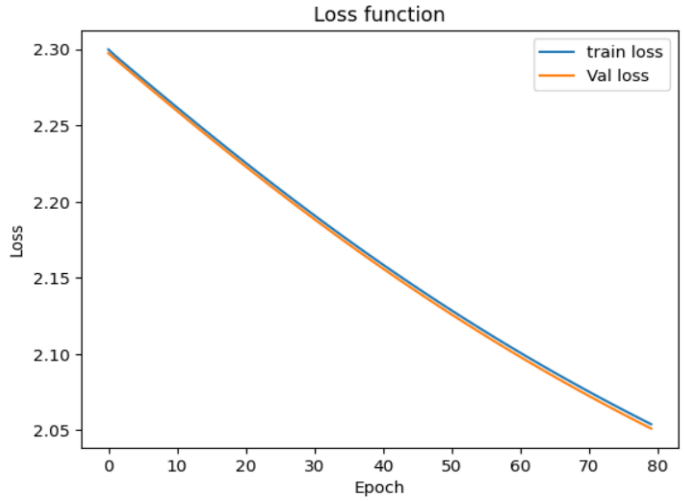
i. Sigmoid Activation function, epochs=90, batches=60, learning rate of Adam
Optimizer=0.001, loss function = Cross Entropy, bias= False
Training Loss: 1.658, Validation Loss: 1.654, f1_score=0.853
Accuracy on test data: 0.8879
Decent convergence rate but can do further convergence by modifying the parameters.

```
f1 score: 0.8573
Epoch loss: 1.658154
Val loss: 1.654482
<ipython-input-124-2d3f35457ae4>:44: UserWarning: Implicit dimension choice
  z = fn(h)
```



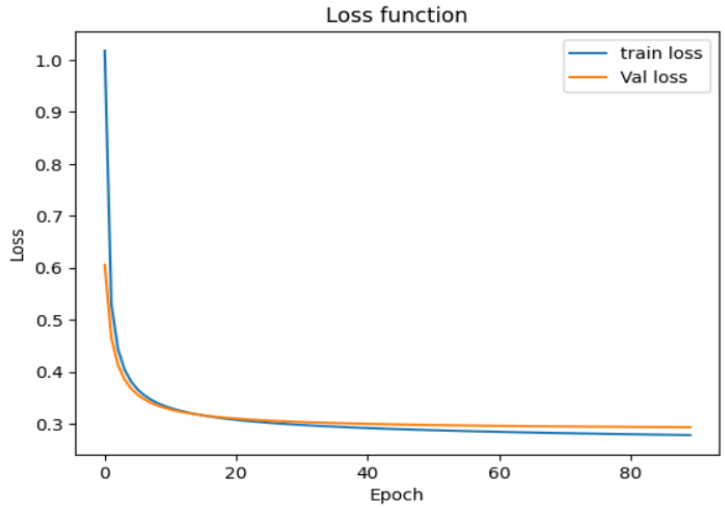
ii. Softmax Activation function, epochs=80, batches=25, learning rate of Adam
Optimizer=0.001, loss function = Cross Entropy, bias= False
Training Loss: 2.053, Validation Loss: 2.051, f1_score=0.763
Accuracy on test data: 0.7702
Worst convergence compared to the above sigmoid activation function

```
f1 score: 0.7637
Epoch loss: 2.053942
Val loss: 2.051100
<ipython-input-251-2d3f35457ae4>:44: UserWarning: Implicit dimension choice
  z = fn(h)
```



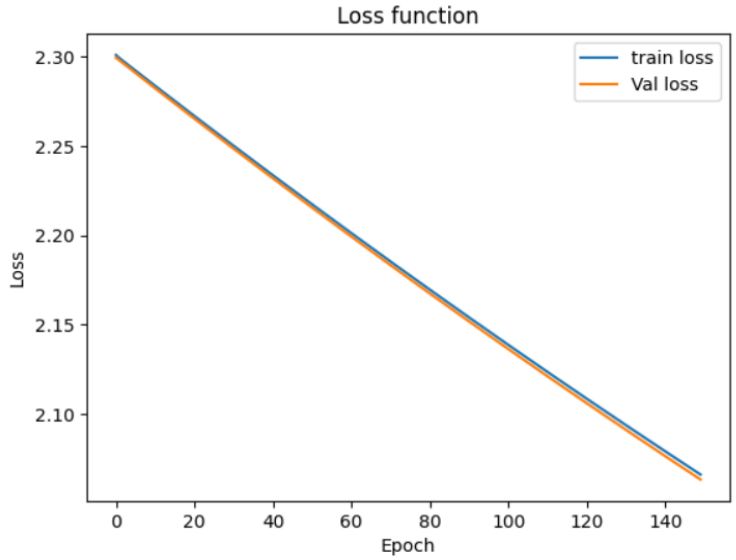
iii. Tanh Activation function, epochs=90, batches=60, learning rate of Adam Optimizer=0.001, loss function = Cross Entropy, bias= False
Training Loss: 0.27, Validation Loss: 0.29, f1_score=0.9186
This setting converged at a good rate.
Accuracy on test data: 0.917

```
f1 score: 0.9186
Epoch loss: 0.278600
Val loss: 0.293698
<ipython-input-300-7e5eb1206d01>:30: UserWarning: Implicit dimension choice f
  z = fn(h)
```



<u>The autoencoder model I configured was first reducing the original input(784) to 512 which is hidden layer and from 512 to 64 features.</u>
HiddenLayer-ReLU, OutputLayer-Softmax, default Adam Optimizer learning rate, loss function = Cross Entropy, epoch = 150, batch = 25, Tested on dev input setting
Training loss:2.067, Validation loss:2.064, f1_score:0.58.
Accuracy on test data: 0.6007
For good convergence should have increased the epoch iterations.

```
f1 score: 0.5896
Epoch loss: 2.066023
Val loss: 2.063235
<ipython-input-300-7e5eb1206d01>:30: UserWarning: Implicit dimension choice
  z = fn(h)
```



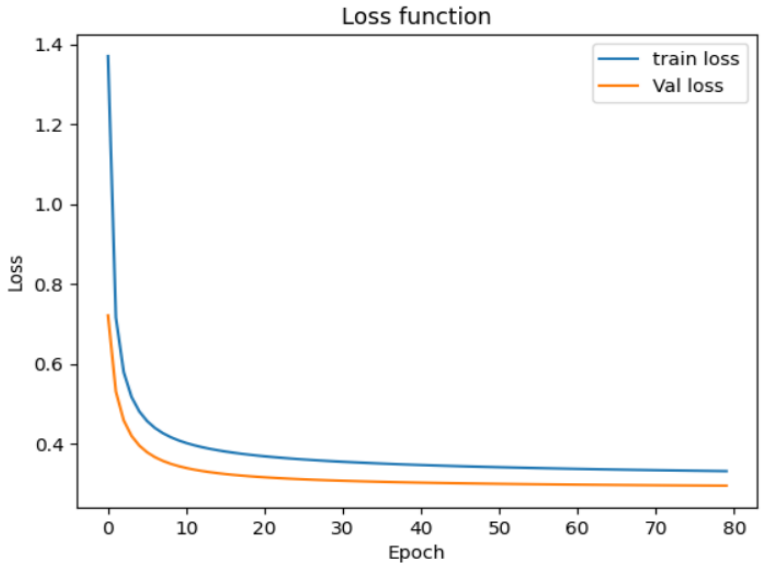
Each configuration above does well except 256 feature where activation function is softmax, its accuracy for test data 0.7702 performed worst than 64 feature encoded representation
The baseline model of Logistic Regression for the original data(784 features) performed with accuracy of 0.9296.
The tanh configuration of 256 representation test accuracy is almost same as the baseline model which is 0.917
For 64 feature representation tanh configuration scored accuracy of 0.859, remaining 64 feature configurations have accuracy around 0.80
In 256 feature representation sigmoid activate function has accuracy of 0.8879.
Overall, tanh activation outperforms sigmoid activation function.
Although 256 feature representation, it got accuracy almost same as baseline classifier

Question 3----------

3.

Extended the neural auto encoder decoder model for classification loss L_c and auto encoder decoder loss L_{autoec} i.e. joint loss of $L_c + L_{autoec}$
The loss functions used are L_c = Cross Entropy Loss and for L_{autoec} = Binary Cross Entropy Loss.
Although tried with MSE loss but need to convert it into one hot form because the data is not continuous.
Reduced Features = 256, tanh activation function, adam optimizer with learning rate = 0.001, epochs = 80, batch = 100, bias =False
Training Loss: 0.331, Validation Loss: 0.29, f1_score: 0.963
Compared to other models, this models joint loss minimized the loss further such that compared to other validation loss this joint loss minimizes up to a good extent. Validation loss performed good by converging.
Accuracy on test data: 0.919

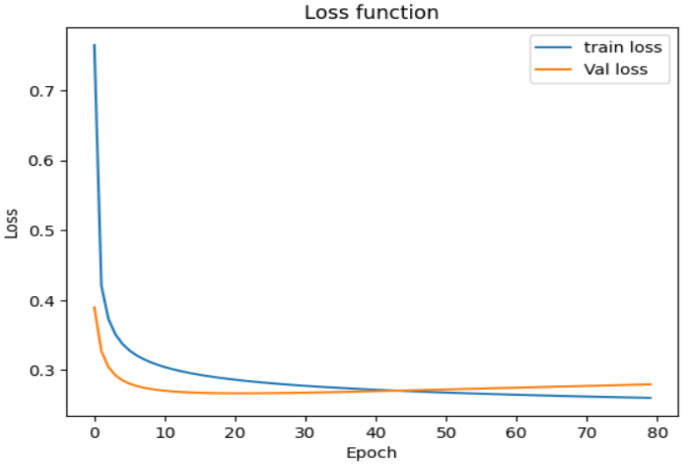
```
f1 score: 0.9163
Epoch loss: 0.331362
Val loss: 0.294911
<ipython-input-10-e44211bf9fb2>:44: UserWarning: Implicit dimension choice
  z = fn(h)
```



Checked how the original features representation performs i.e. 784 with almost same setting, it gave almost same as the 256 featured reduced representation. In this validation loss is just high compared to train loss.

Training Loss: 0.26, Validation Loss: 0.279, f1_score: 0.963,Accuracy on test data: 0.9264

```
f1 score: 0.9264
Epoch loss: 0.260071
Val loss: 0.279474
<ipython-input-10-e44211bf9fb2>:44: UserWarning: Implicit dimension choice
  z = fn(h)
```



4.

The paper (Zhang and Lapata 2017) Sentence Simplification with Deep Reinforcement Learning implements the Sentence Simplification using Reinforcement Learning. The main priority of this approach for Sentence Simplification is to reduce sentence complexity to very simple while maintaining certain constraints: Simplicity, preserving meaning and grammatical correctness. The authors have done the task of reducing the complex sentences to simpler sentences with the help of the model defined by them which is termed as Deep Reinforcement Sentence Simplification (DRESS). The authors used a neural encoder-decoder model, specifically Recurrent Neural Networks, as their initial approach to sentence simplification. This generative probabilistic model utilizes Long Short-Term Memory (LSTM; Hochreiter and Schmidhuber 1997) for both the encoder and decoder. They

then incorporated reinforcement learning by providing rewards to the agent, which improved its ability to satisfy constraints. The rewards which the reinforcement agent concentrates are Simplicity (which applies rewrite operations i.e. make changes for the sentence), Relevance (meaning of the target sentence is preserved), Fluency. The goal of Reinforcement Algorithm is to maximize the expected rewards. Further extending the DRESS model with Lexical Simplification (DRESS-LS) which substitutes complex words with simpler alternatives.
The Markov Decision Process (MDP) components of DRESS formulation are:
MDP: (States,Actions,Rewards,π,γ)
States: Source X which consists of complex sentences which needs to be simplified
Actions: performing rewrite operations such as copying, deletion, substitution and word ordering for complex
Rewards: The rewards in sentence simplification are Simplicity, Relevance, Fluency, for which the reinforcement agent aims to maximize
π: The distribution in sentence simplification is given by policy, it generates the distribution over the actions when the states are given. The transition of distribution to new state is done based on the Rewards.
γ: γ (discount factor) is set to 1 as it emphasis more on the future rewards
The loss function used in Neural Encoder-Decoder is minimizing the negative log-likelihood which is cross entropy loss, where as in DRESS the loss function has the cross entropy loss with the reward function I.e. expected reward from agent.
Datasets used are Newsela(Xu et al. (2015b)), WikiSmall (Zhu et al., 2010), WikiLarge. Comparison systems used are hybrid (Narayan and Gardent (2014)), PBMT-R (Narayan and Gardent, 2014), Reference. Evaluation of the model on metrics such as SARI (Xu et al., 2016), BLEU (Papineni et al., 2002), Flesch-Kincaid Grade Level index (FKGL; Kincaid et al. 1975). FKGL measures readability of output, lower the better. BLEU assess the degree to which generated simplifications differed from gold standard reference. SARI evaluates the quality of the output by comparing it against the source and reference simplifications. BLEU, SARI higher the better.
DRESS scores lower than EncDecA system in FKGL which means the DRESS is learning the reward and is better than the EncDecA for Newsela data. DRESS-LS performs the best for BLEU and slightly worse for FKGL and SAR for Newsela. For WikiSmall, DRESS performs the best for FKGL and Hybrid performs best for BLEU, SARI. For WikiLarge, EncDecA performs the best for BLEU.
Human evaluation ratings are given based on Fluency, Adequecy, Simplicity and All(combined ratings) for all the datasets and systems(not all) mentioned above. DRESS-LS has highest rating for Fluency and All on all data sets including best for simplicity on WikiSmall dataset. PBMT-R scores highest for all datasets. DRESS scores best for Simplicity for Newsela. DRESS and DRESS-LS are the best on Simplicity which was the main task of Sentence Simplification. Overall DRESS-LS performs better than other systems except DRESS which is almost similar.
The model conclusions based on rewrite operations which is done by Translation Error Rate (TER). Reinforcement Learning encourages to do more deletions. The task of the sentence to be simplified, Relevance and Fluency were there. Although, Reinforcement Models performed best at simplicity, it also performed best when combined all and it achieved the target of the Sentence Simplification.

References:

Dataset:
https://www.csee.umbc.edu/courses/graduate/678/spring23/materials/mnist_rowmajor.jsonl.gz

Welcome to PyTorch Tutorials — PyTorch Tutorials 1.13.1+cu117 documentation

Sentence Simplification with Deep Reinforcement Learning (aclanthology.org)

3.3. Metrics and scoring: quantifying the quality of predictions — scikit-learn 1.2.2 documentation

08-nn.pdf (umbc.edu)

09-cnn-rnn.pdf (umbc.edu)