AI6127: Deep Neural Networks for Natural Language Processing

ASSIGNMENT 1: NAMED ENTITY RECOGNITION

Chen Yongquan | G2002341D

1c.

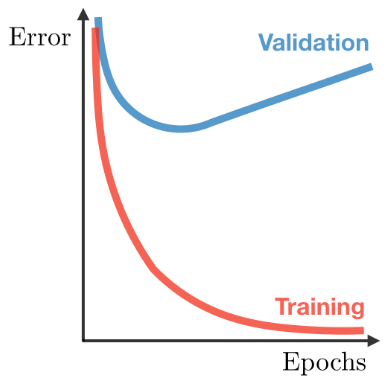
In order to optimize the hyperparameters of the Softmax classifier in terms of predictive performance based on its F-score, we remove the 'O' label from the set of labels to include for the weighted average score as we do not want it to overwhelm the scores of the more important classes and give an overly optimistic result.

Our baseline set of hyperparameters is:

1. Window size: 2
2. Embedding size: 25
3. Hidden layer size: 25
4. Number of hidden layers: 1
5. Learning rate: 10
6. Number of epochs: 500
7. Batch size: 150
8. Activation function: Hyperbolic tangent
9. Dropout layers: None
10. Scheduler: ReduceLROnPlateau(patience=5, factor=0.5, threshold=0.00001, min_lr=1.0)

We do not experiment with freezing word embeddings as we do not use pretrained word embeddings and every set of hyperparameters are tested using a fresh model, so freezing word embeddings would only give worse performance when they are untrained.

The learning rate may seem extremely large compared to the normal range of [0.0,1.0], however based on our testing, a learning rate of 10 is a good value offering faster convergence without impeding proper learning. We supposed this is because the losses are averaged not by batch size (sentences) but by total number of words per batch and there is a varying number of words per sentence in the dataset. Also, because the dataset is unbalanced, only a few words are from the harder to predict entity classes per batch, while the rest belong to the relatively easier to predict 'O' class, so once the model has learnt to predict the 'O' class, losses from the entity words will be overwhelmed by the large denominator of total words in the batch. Thus, a larger learning rate is required to balance this. We could alternatively use a weighted cross-entropy loss function to increase the loss from the entity classes based on their inverse frequency in the training dataset.



We train for 500 epochs and if overfitting occurs within the 500 epochs, we pick the epoch with the best validation score and a reasonably good training score. Generally, validation score should improve as training progresses until a point where overfitting occurs and starts to drop while training score continues to improve. In our experiments, there are cases during which validation score achieves a record high before training score saturates but we do not pick the model at that epoch as that can be attributed to randomness and the low training score can also hint at a low test score. It should also be noted that the traditional overfitting curve is just a simplification of the problem and training long enough with sufficiently large amounts of data can still allow deep networks to generalize well [1].

Batch size is set at 150 to reduce the time taken per epoch at the expense of lesser batches and gradient updates. We do an evaluation on the validation set only after every 10 epochs to reduce training time because much of the computational time is spent on data loading and collation on the CPU instead of the model running on the GPU.

Best training score on baseline was 0.835 while best validation score was 0.0262 on the 280th epoch. Further training up until epoch 800 results in overfitting.

We then experiment with different hyperparameters and architectures from the baseline to improve our F-score. Best training and validation scores are shown in the table below. Note that training scores are averaged scores across all training batches in that epoch, and weights are updated across each batch

Hyperparameters	Best Model F-score
Baseline Window size: 2 Embedding size: 25 Hidden layer size: 25 Number of hidden layers: 1 Learning rate: 10 Number of epochs: 500 Batch size: 150 Activation function: Hyperbolic tangent Dropout layers: None Scheduler: ReduceLROnPlateau	T: 0.8349229860563048 V: 0.02616703242978282
Batch size: 20	T: 0.823521546825552 V: 0.0472602291760523
Batch size: 20 Scheduler: None	T: 0.9727235787185505 V: 0.07254580068327819
Embedding size: 150 Hidden size: 150	T: 0.7606103228875668 V: 0.08964197405445547
Batch size: 20 Embedding size: 150 Hidden size: 150 Scheduler: CosineAnnealingWarmRestarts	T: 0.8352012877808912 V: 0.09856396605379791
Batch size: 20 Embedding size: 300 Hidden size: 300 Scheduler: CosineAnnealingWarmRestarts	T: 0.06043880819653723 V: 0.5022933473565581
Window size: 1	T: 0.8376800420414307 V: 0.048314686342538764
Window size: 3	T: 0.8497757130797894 V: 0.019741770632098671

Number of hidden layers: 2	T: 0.300679894400209 V: 0.064347968099135
Number of hidden layers: 3	T: 0.0014174502720477712 V: 0.0
Learning rate: 50 Scheduler: None	T: 0.0186125149879 12842 V: 0.0201033167849 05435
[Best Model/Hyperparameters] Batch size: 20 Number of hidden layers: 3 Activation function: Sigmoid Scheduler: None	T: 0.8656600635816363 V: 0.0805085229955497 Epoch 590: T: 0.6844320671568037 V: 0.11477646533562093
Batch size: 20 Activation function: Sigmoid Scheduler: None	T: 0.9914986090922294 V: 0.09983897187129542
Batch size: 20 Embedding size: 300 Hidden size: 300 Number of hidden layers: 3 Activation function: Sigmoid Scheduler: None Number of epochs: 1500	T: 0.986382575542124 V: 0.07606014841673074
Batch size: 20 Activation function: Sigmoid Scheduler: None Dropout layers: P(0.25) Stemming: NLTK.SnowballStemmer	T: 0.912880808055158 V: 0.05857835360621159
Batch size: 20 Activation function: Sigmoid Scheduler: None Stemming: NLTK.SnowballStemmer	T: 0.841581516327442 V: 0.05510925640515403
Batch size: 20 Activation function: Sigmoid Scheduler: CosineAnnealingWarmRestarts Stemming: NLTK.SnowballStemmer	T: 0.9766877846773181 V: 0.05165450548725808
Batch size: 20 Embedding size: 150 Hidden size: 150 Number of hidden layers: 3 Activation function: Sigmoid Scheduler: None Residual blocks: Yes	T: 0.9907915748862546 V: 0.05097768880769482

DISCUSSION

Reducing batch size resulted in faster convergence as there were more batches and parameter updates per epoch, but time taken to run one epoch is longer with much of it spent on CPU cycles in data loading and collation, particularly in the specified collate_fn() of the dataloaders.

Increasing embedding and hidden layer size by their widths resulted in a network that took much longer to train, and training score was not able to reach saturation point in 500 epochs.

Setting too high a learning rate causes learning to not progress smoothly and rubber-bands around high loss values.

Increasing model depth made the model harder to train and learning entirely stopped once depth reaches 3. Deep networks are harder to train not only because of vanishing/exploding gradients which can be addressed by input normalization and batch normalization layers. However, vector lengths hold meaning in natural language processing and can convey the frequency a word is encountered in the domain, so we didn't try to normalize any inputs or intermediate outputs. We experimented with residual learning [2] to see if the problem was due to the layers having difficulty in learning an identity mapping or a mapping close to it. The idea behind residual learning is to add the inputs to a layer to its outputs. This way instead of the layer learning the original mapping $F(x)$, it is forced to learn the mapping $F(x) - x$, then when adding the input to its output gives the original mapping output. This allows it to easily learn an identity mapping by setting its output to 0 (multiplying inputs by 0). In the presentation in CVPR2016, the authors of ResNet did note the applicability of ResNet in many domains other than computer vision, among which is natural language processing. However, the residual blocks in the original paper used ReLU activation functions and efficacy of residual blocks may be reduced when using other activation functions with limited range like sigmoid and hyperbolic tangent. Nonetheless, we tested with all 3 activations functions with and without residual learning, and it seems the problem was not related to residual learning.

Instead, changing the activation function to sigmoid not only resulted in much faster convergence (0.9 F-score in ~100 epochs) than the hyperbolic tangent function, but also allowed us to have proper learning at depth 3, though learning still slows to a halt as depth goes up to 5.

For learn rate scheduling, CosineAnnealingWarmRestarts is better than the baseline scheduler ReduceLROnPlateau in our experiments. The idea behind cosine annealing is to gradually lower the learning rate then restart the learning rate to a higher rate after a set number of epochs. This mimics the metal annealing process whereby letting the metal slowly cool can allow the molecules to settle into a better equilibrium that makes the metal harder. The F-score after each cosine annealing cycle ends should be better than the previous score until saturation. However, it might also be possible that cosine annealing was better than ReduceLROnPlateau because the hyperparameters for the latter scheduler was not well optimized (e.g. reducing the learning rate too early because patience was too low, or threshold was too high).

Finally, we tried various ways to improve the validation and test F-scores because generalization was extremely poor. We experimented with deeper layers coupled with residual blocks, adding dropout layers for regularization [3], and stemming the dataset. However, it seemed the best configuration followed Occam's razor: batch size 20, sigmoid activation, 3 linear layers, no learn rate scheduling.

Future work can be done to see if adding character level embeddings, POS tag features and concatenating them to the word embeddings can allow the model to better recognize Out-Of-Vocabulary words and perform better on the test and validation set, because the training set is probably too small to build a sizeable dictionary even with stemming, resulting in large portions of the test and validation data being tagged with <UNK> and fed into the network. There just weren't enough features for the Softmax classifier to process in order to compete with the CRF classifier which had POS tags to aid in prediction.

References

[1] M. Belkin, D. Hsu, S. Ma and S. Mandal, "Reconciling modern machine learning practice," 2019.

[2] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2015.

[3] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 2012.