Homework 3 Write-up

Federico Cimini, Kyle Sullivan

1 Deep Averaging Network

1.1 Architecture

We tried various learning rates, dropouts, non-linear functions, and model depths. Our final DAN model consisted of a 0.0001 learning rate, 0 dropout, 1e-5 weight decay, a 128 batch size, and 3 epochs. The best-performing model contained a single-depth linear + ReLU model with a final linear (outputting to binary) layer, followed by softmax. Unlike the LSTM, there were two candidates for the best model, as our choice had the best dev accuracy (87.3307), but only the second best validation loss (0.4377) – a depth of 2 TanH model edged it out with a loss of 0.4367. We prioritized the bottom line accuracy and appreciated that the single-layer ReLU had a more impressive accuracy jump (+0.0989) compared to the slight increase in loss (+0.001).

1.2 Results

See Table 2 in the appendix for the results.

1.3 Discussion

We tried many hyper-parameter combinations, iteratively adjusting them based on the other current best settings. Unlike with the LSTM, dropout never helped the best dev accuracy, likely because our best-performing DAN model was much simpler, with only one linear + ReLU layer feeding into a final linear (to binary) softmax pairing. With fewer layers and inputs from the average of the sentence embeddings (no hidden state input), dropout wasn't as useful.

Larger learning rates also hurt update stability, considering our somewhat small 128 batch size. Thus, a tiny 0.0001 learning rate worked best. Of the different non-linear functions we tried, TanH worked best (still a bit worse than ReLU). Adding a second linear + non-linear activation layer failed to improve the results. Our smaller 300-dimension input didn't benefit from this added complexity. In general most of the results on the table were very similar, with differences often being within the margin of error.

2 Recurrent Neural Network

2.1 Architecture

Our final RNN model consisted of an LSTM with 300 dimension hidden layers, a 0.0001 learning rate, 0.3 dropout, 20 layers, single-directional, 0.001 weight decay, 5 epochs, and a 256 batch size. The data process began with putting the embedded arrays through a pack-padded sequence before being fed into the LSTM. The resulting flattened H_n (the rightmost/vertical hidden layer, coming from the last value in the sequence) was then passed through 4 linear + ReLu pairings, followed by a last linear (outputting to binary) layer and, finally, softmax. Regarding alternative feature extraction methods, we tried taking the last (rightmost) hidden state from the deepest (top) stack by sequencing the final indexed value in the H_n. Additionally, we tried averaging and maxing over the H_t values in the deepest (horizontal) stack's hidden states. The full H_n layer performed best according to the validation loss and best accuracy.

We tried various epochs, batch sizes, learning rates, hidden layer dimensions, dropouts, number of layers, feature extractions, and model architectures. We opted for more stability, pushing the batch size up 256 with a small learning rate (0.0001) and a slightly larger 5 epochs. Additionally, we also tried a GRU and a standard RNN with these same best LSTM parameters. Ultimately, the LSTM worked best, as it produced the highest best dev accuracy (87.8633) and best validation loss (0.4302).

2.2 Results

See Table 3 in the appendix for the results. We experimented with many hyper-parameters, including the dimension of the hidden layers, the learning rate, dropout, number of layers, feature extraction, and model architectures. We conducted many tuning iterations to accommodate better hyper-parameter values as they were found, thus requiring repeat testing of previous hyper-parameters. Ultimately, we found that the 4th row's (in bold) settings were best. Individually altering each hyper-parameter from this optimal baseline always produced a lower best dev accuracy and higher dev validation loss. Only the feature extraction looked at a slightly different L=10 setup, which under-performed compared to the L=10 tweak of the optimal hyper-parameter settings.

2.3 Discussion

We tried many hyper-parameter combinations, iteratively adjusting them based on the other current best settings. Our best accuracy came when using a hidden layer dimensionality of 300. Using fewer dimensions produced slightly lower accuracies. Similarly, we tried a 10x reduction and a doubling of the learning rate, both of which performed worse. A dropout rate near 0.3 achieved the best results. Using no dropout or a larger value (e.g., 0.5) either caused overfitting (lower dev accuracy) or too much regularization, respectively. Both ultimately hurt the final dev accuracy. Twenty stacked LSTMs proved to be the sweet spot, but there was a limited amount of accuracy variation associated with this hyper-parameter. Unsurprisingly, adding bidirectionality didn't improve the results, considering that, unlike sequence labeling or text generation problems, we weren't performing a task requiring outputs/predictions at each step in the sequence. The number of fully connected linear + ReLU layers following the LSTM had a minor impact. ReLU was our top-performing non-linearity function, followed by TanH and LeakyReLU. Switching from an LSTM to a GRU produced similar results, whereas a standard RNN performed much worse - particularly, when compared to the LSTM.

Interestingly, using just the deepest stack's last sequential hidden state (top right) – rather than an average or max over this entire deepest (top horizontal) hidden state worked better than these aggregate alternatives. This result makes sense, considering the top right value is the final culmination of the stacked layers and sequential hidden state updates. Taking the mean provided the worst performing model, as the reduced input size (compared to H_n concat) focused on all sequential layers equally (unlike max, which found the most positive extreme values), despite latter sequential states containing more refined values, having been adjusted over all the prior states.

3 Qualitative Analysis

3.1 Importance of the GLOVE initialization

In class we discussed that word embeddings offer a good initialization for features of sentences. In this section, we will analyze how true this claim is.

3.1.1 Training with alternative word embedding initialization

Experiments on word embeddings can be seen on Table 1.

Architecture	Embeddings	Update during learning	Validation Accuracy
Single linear layer	GLoVE	YES	86.9046
Single linear layer	GLoVE	NO	82.3467
Single linear layer	Randomized	YES	77.7507
Single linear layer	Randomized	NO	67.7522
Linear + ReLu + Linear	Randomized	NO	78.0475

Table 1: Experimentation on different word embedding initializations

3.1.2 Discussion

According to these results, using GLoVE embedding is important for classification performance. We can see that when we use GLoVE, validation accuracy goes up 10 percentage points compared to when starting with random embeddings. this means that bringing in pre-trained embeddings with information from other data is important to get a boost in performance.

Updating these embeddings is also important to the overall result. The model is still able to perform decently with random compared to GLoVE embeddings, but it does need to update these embeddings while training because if not the performance is very poor. We can say that updating embeddings away from their initialization is crucial to getting a solid performance. This is likely because the updated embeddings give us more information on how each word relates to its context.

We also see that increasing the depth of the DAN increases performance significantly, which would indicate that with a deep, robust DAN architecture you could get a decent prediction without needing to pre-train embeddings. To further be sure about this, we could perform more experiments using a deeper DAN architecture, or a different RNN altogether.

3.2 Model behavior as a function of sentence length

By grouping input sentences in buckets of length of 5, we can see how the sentence length affects accuracy in both models. We performed this analysis on the validation set. In Figure 1 we can see input sentences grouped by buckets of 5 (0 means length between 0 and 4, 5 means sentences between 5 and 9, etc) and the accuracy of both models for those buckets.

First, we can see that both models perform well on short sentences between 0 and 4 tokens long, with LSTM outperforming DAN. After that, we see that LSTM performs better in general the longer the input sentences are. This is likely because longer sentences have more complexity in terms of sentiment, and a model like LSTM can understand this complexity better. For example 3-word long sentences like "this is good" are more straight to the point and have less convoluted meaning than longer phrases like "while the quality of character development is acceptable, the plot was not the greatest".

4 Extra Credit

(up to 10 points) Manually construct failure cases (inputs) for classifier (eg: negation and metaphor). Consider both false positive and false negative cases. Extra credit will be awarded if you can describe a pattern for the new failure cases you have discovered, and report at least 5 instances not from the dataset that demonstrate the pattern and that your model fails to classify correctly. We will award credit based on how interest and coherent of errors you find.

5 Appendix

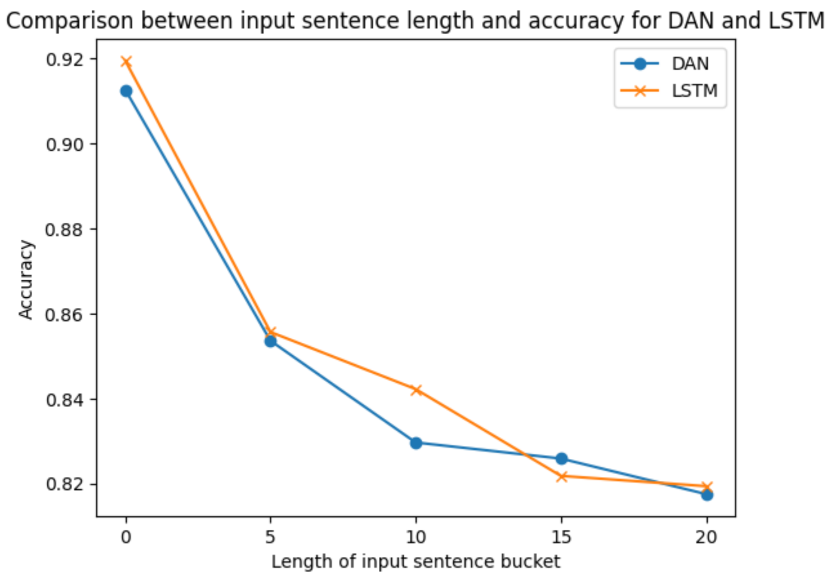

Figure 1: Comparison between input sentence length and accuracy for DAN and LSTM

LR	Drop	Model	Val. Loss	Best Acc.
0.01	0	Linear+ReLu+Linear+Softmax	0.4786	83.0619
0.001	0	Linear+ReLu+Linear+Softmax	0.4421	86.4100
0.0001	**0**	**Linear+ReLu+Linear+Softmax**	**0.4377**	**87.3307**
0.0001	0.1	Linear+ReLu+Linear+Softmax	0.4412	87.2089
0.0001	0.25	Linear+ReLu+Linear+Softmax	0.4402	86.9122
0.0001	0.5	Linear+ReLu+Linear+Softmax	0.4384	86.9578
0.0001	0	Linear+Sigmoid+Linear+Softmax	0.4487	86.9655
0.0001	0	Linear+TanH+Linear+Softmax	0.4396	87.4068
0.0001	0	Linear+LeakyReLU+Linear+Softmax	0.4393	86.9198
0.0001	0	2x(Linear+ReLu)+Linear+Softmax	0.4378	87.0263
0.0001	0	2x(Linear+Sigmoid)+Linear+Softmax	0.4387	87.1937
0.0001	0	2x(Linear+TanH)+Linear+Softmax	0.4367	87.2318
0.0001	0	2x(Linear+LeakyReLU)+Linear+Softmax	0.4393	86.8741

Table 2: DAN Variants

H. Dim.	LR	Drop	L	Bidir.	Model	Feat. Extract	Val. Loss	Best Acc.
50	0.0001	0.3	20	No	LSTM+4x(Linear+ReLu)+Linear+Softmax	H_n concat	0.4387	87.1557
100	0.0001	0.3	20	No	LSTM+4x(Linear+ReLu)+Linear+Softmax	H_n concat	0.4313	87.8557
150	0.0001	0.3	20	No	LSTM+4x(Linear+ReLu)+Linear+Softmax	H_n concat	0.4319	87.6427
200	0.0001	0.3	20	No	LSTM+4x(Linear+ReLu)+Linear+Softmax	H_n concat	0.4335	87.4068
300	**0.0001**	**0.3**	**20**	**No**	**LSTM+4x(Linear+ReLu)+Linear+Softmax**	H_n **concat**	**0.4302**	**87.8633**
300	0.001	0.3	20	No	LSTM+4x(Linear+ReLu)+Linear+Softmax	H_n concat	0.4334	87.6351
300	0.00005	0.3	20	No	LSTM+4x(Linear+ReLu)+Linear+Softmax	H_n concat	0.4346	87.3079
300	0.00001	0.3	20	No	LSTM+4x(Linear+ReLu)+Linear+Softmax	H_n concat	0.4440	86.4328
300	0.0001	0	20	No	LSTM+4x(Linear+ReLu)+Linear+Softmax	H_n concat	0.4334	87.6351
300	0.0001	0.1	20	No	LSTM+4x(Linear+ReLu)+Linear+Softmax	H_n concat	0.4314	87.8177
300	0.0001	0.25	20	No	LSTM+4x(Linear+ReLu)+Linear+Softmax	H_n concat	0.4313	87.7416
300	0.0001	0.5	20	No	LSTM+4x(Linear+ReLu)+Linear+Softmax	H_n concat	0.4323	87.6427
300	0.0001	0.3	1	No	LSTM+4x(Linear+ReLu)+Linear+Softmax	H_n concat	0.4362	87.3916
300	0.0001	0.3	5	No	LSTM+4x(Linear+ReLu)+Linear+Softmax	H_n concat	0.4329	87.5894
300	0.0001	0.3	10	No	LSTM+4x(Linear+ReLu)+Linear+Softmax	H_n concat	0.4365	87.3687
300	0.0001	0.3	30	No	LSTM+4x(Linear+ReLu)+Linear+Softmax	H_n concat	0.4374	87.2166
300	0.0001	0.3	20	Yes	LSTM+4x(Linear+ReLu)+Linear+Softmax	H_n concat	0.4341	87.6122
300	0.0001	0.3	20	No	LSTM+(Linear+ReLu)+Softmax	H_n concat	0.4329	87.8025
300	0.0001	0.3	20	No	LSTM+3x(Linear+ReLu)+Softmax	H_n concat	0.4390	87.2394
300	0.0001	0.3	20	No	LSTM+5x(Linear+ReLu)+Softmax	H_n concat	0.4315	87.6275
300	0.0001	0.3	10	No	LSTM+4x(Linear+ReLu)+Linear+Softmax	Deepest H_n	0.4366	87.3231
300	0.0001	0.3	10	No	LSTM+4x(Linear+ReLu)+Linear+Softmax	Mean of Deepest H_t	0.4525	85.6567
300	0.0001	0.3	10	No	LSTM+4x(Linear+ReLu)+Linear+Softmax	Max of Deepest H_t	0.4396	87.0872
300	0.0001	0.3	20	No	LSTM+4x(Linear+TanH)+Linear+Softmax	H_n concat	0.4345	87.6046
300	0.0001	0.3	20	No	LSTM+4x(Linear+LeakyReLU)+Linear+Softmax	H_n concat	0.4363	87.3003
300	0.0001	0.3	20	No	GRU+4x(Linear+ReLu)+Linear+Softmax	H_n concat	0.4345	87.2698
300	0.0001	0.3	20	No	RNN+4x(Linear+ReLu)+Linear+Softmax	H_n concat	0.4404	86.9122

Table 3: LSTM Variants