Learning when to skim and when to read

Alexander R. Johansen and **Richard Socher**
Salesforce Research, Palo Alto, CA, USA
{ajohansen,rsocher}@salesforce.com

Abstract

Many recent advances in deep learning for natural language processing have come at increasing computational cost, but the power of these state-of-the-art models is not needed for every example in a dataset. We demonstrate two approaches to reducing unnecessary computation in cases where a fast but weak baseline classier and a stronger, slower model are both available. Applying an AUC-based metric to the task of sentiment classification, we find significant efficiency gains with both a probability-threshold method for reducing computational cost and one that uses a secondary decision network. [1]

1 Introduction

Deep learning models are getting bigger, better and more computationally expensive in the quest to match or exceed human performance (Wu et al., 2016; He et al., 2015; Amodei et al., 2015; Silver et al., 2016). With advances like the sparsely-gated mixture of experts (Shazeer et al., 2017), pointer sentinel (Merity et al., 2016), or attention mechanisms (Bahdanau et al., 2015), models for natural language processing are growing more complex in order to solve harder linguistic problems. Many of the problems these new models are designed to solve appear infrequently in real-world datasets, yet the complex model architectures motivated by such problems are employed for every example. For example, fig. 1 illustrates how a computationally cheap model (continuous bag-of-words) represents and clusters sentences.

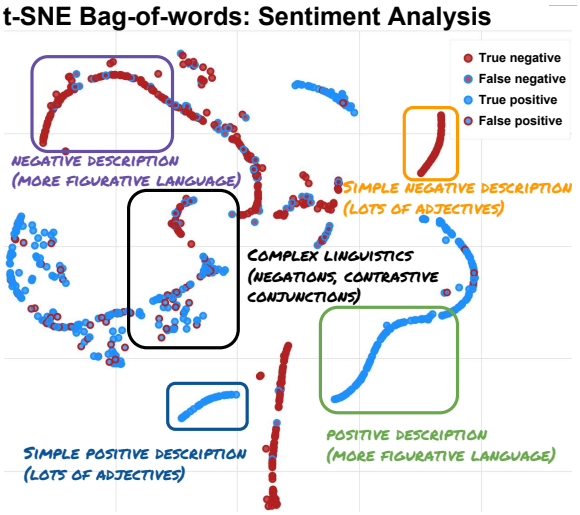


Figure 1: Illustration with t-SNE (van der Maaten and Hinton, 2008) of the activations of the last hidden layer in a computationally cheap bag-of-words (BoW) model on the binary Stanford Sentiment Treebank (SST) dataset (Socher et al., 2013). Each data point is one sentence, while the plot has been annotated with qualitative descriptions.

Clusters with simple syntax and semantics ("simple linguistic content") tend to be classified correctly more often than clusters with difficult linguistic content. In particular, the BoW model is agnostic to word order and fails to accurately classify sentences with contrastive conjunctions.

This paper starts with the intuition that exclusively using a complex model leads to inefficient use of resources when dealing with more straightforward input examples. To remedy this, we propose two strategies for reducing inefficiency based on learning to classify the difficulty of a sentence. In both strategies, if we can determine that a sentence is easy, we use a computationally cheap bag-of-words ("skimming"). If we cannot, we default

[1] Blog post with interactive plots is available at https://metamind.io/research/learning-when-to-skim-and-when-to-read

Proceedings of the 2nd Workshop on Representation Learning for NLP, pages 257–264,
Vancouver, Canada, August 3, 2017.

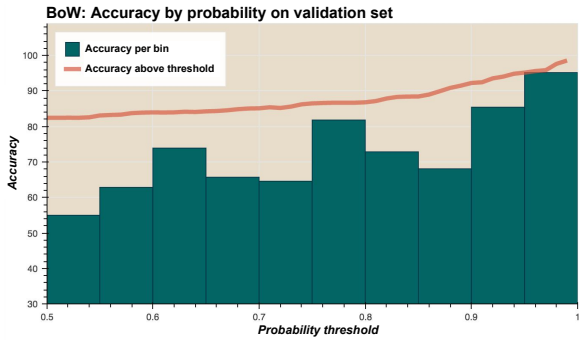


Figure 2: Accuracy for thresholding on binary SST. Probability thresholds are the maximum of the probability for the two classes. The green bars corresponds to accuracy for each probability bucket (examples within a given probability span, e.g. $0.5 < \Pr(Y|X, \theta_{\text{BoW}}) < 0.55$), while the orange curve corresponds the accuracy of all examples with a probability above a given threshold (e.g. $\Pr(Y|X, \theta_{\text{BoW}}) < 0.7$).

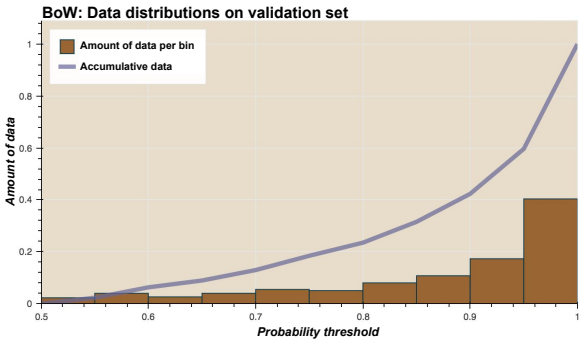


Figure 3: Histogram showing frequency of each BoW probability bin on the SST validation set. The line represents the cumulative frequency, or the fraction of data for which the expensive LSTM is triggered given a probability threshold.

SST Valid		BoW 82%	
		True	**False**
LSTM	**True**	76%	12%
88%	**False**	6%	6%

Table 1: Confusion matrix for the BoW and the LSTM, where **True** means that the given model classifies correctly.

to an LSTM (reading). The first strategy uses the probability output of the BoW system as a confidence measure. The second strategy employs a decision network to learn the relationship between the BoW and the LSTM. Both strategies increase efficiency as measured by an area-under-the-curve (AUC) metric.

2 When to skim: strategies

To keep total computation time down, we investigate cheap strategies based on the BoW encoder. Where the probability thresholding strategy is a cost-free byproduct of BoW classification and the decision network strategy adding a small additional network.

2.1 Probability strategy

Since we use a softmax cross entropy loss function, our BoW model is penalized more for confident but wrong predictions. We should thus expect that confident answers are more likely correct. Figure 2 investigates the accuracy by thresholding probabilities empirically on the SST based on the BoW outputs, strengthening such hypothesis. The probability strategy uses a threshold τ to determine which model to use, such that:

$$\hat{Y}_{\Pr(Y|X,\theta_{\text{BoW}})>\tau} = \arg\max_Y \Pr(Y|X, \theta_{\text{BoW}})$$
$$\hat{Y}_{\Pr(Y|X,\theta_{\text{BoW}})<\tau} = \arg\max_Y \Pr(Y|X, \theta_{\text{LSTM}})$$

where Y is the prediction, X the input, θ_{BoW} the BoW model and θ_{LSTM} the LSTM model. The LSTM is used only when the probability of the BoW is below the threshold. Figure 3 illustrates how much data is funneled to the LSTM when increasing the probability threshold, τ.

2.2 Decision network

In the probability strategy we make our decision based on the expectation that the more powerful LSTM will do a better job when the bag-of-words system is in doubt, which is not necessarily the case. Section 2.1 illustrates the confusion matrix between the BoW and the LSTM. It turns out that the LSTM is only strictly better 12% of the time, whereas 6% of the sentences neither the BoW or the LSTM is correct. In such case, there is no reason to run the LSTM and we might as well save time by only using the BoW.

2.2.1 Learning to skim, the setup

We propose a trainable decision network that is based on a secondary supervised classification task. We use the confusion matrix between the BoW and the LSTM from Section 2.1 as labels. We consider the case where the LSTM is correct and the BoW is wrong as the LSTM class and all

Model	Cost per sample
Bag-of-words (BoW)	0.16 ms
LSTM	1.36 ms

Table 2: Computation time per sample for each model, with batch size 64.

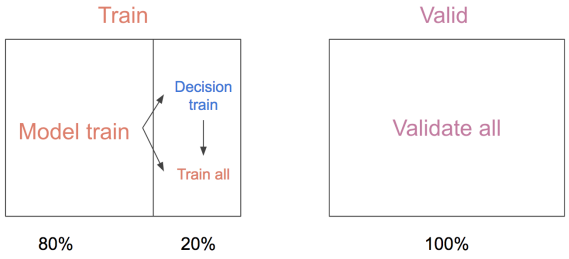


Figure 4: We train both models (BoW and LSTM) on "model train", then generate labels and train the decision network on "decision train" and lastly fine tune the models on the full train set. The full validation set is used for validation.

other combinations as the BoW class.

However, the confusion matrix on the train set is biased due to the models overfitting—which is why cannot co-train the decision network and our models (BoW, LSTM) on the same data. Instead we create a new held-out split for training the decision network in a way that will generalize to the test set. We split the training set into a model training set (80% of training data) and a decision training set (remaining 20% of training data). We first train the BoW and the LSTM models on the model training set, generate labels for the decision training set and train the decision network on the decision training set, and lastly fine-tune the BoW and the LSTM on the original full training set while holding the decision network fixed. We find that the decision network will still generalize to models that are fine-tuned on the full training set. The entire pipeline is illustrated in 4.

3 Related Work

The idea of penalizing computational cost is not new. Adaptive computation time (ACT) (Graves, 2016) employs a cost function to penalize additional computation and thereby complexity. Concurrently with our work, two similar methods have been developed to choose between computationally cheap and expensive models. Odena et al. (2017) propose the composer model, which chooses between computationally inexpensive and

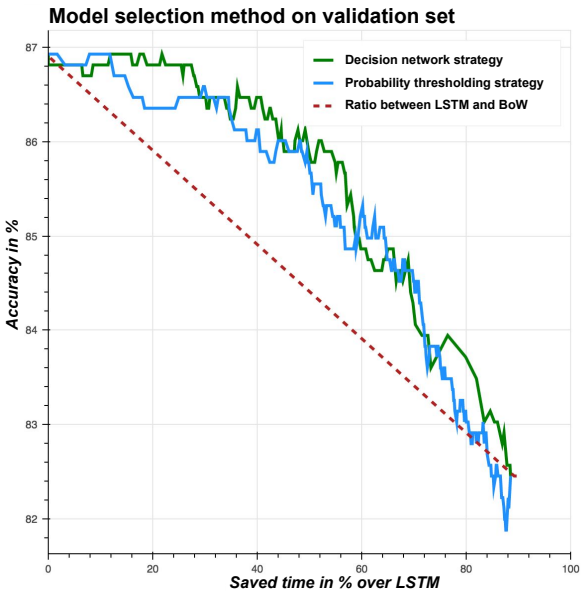


Figure 5: Model usage strategies plotted with thresholds for the probability and decision network strategies chosen to save a given fraction of computation time. The curve stops at around 90% savings as this represents using only the BoW model. The dashed red line represents the naïve approach of using the BoW and LSTM models at random with a fixed ratio.

expensive layers. To model the compute versus accuracy tradeoff they use a test-time modifiable loss function that resembles our probability strategy. The composer model, similar to our decision network, is not restrained to the binary choice of two models. Further, their model, similar to our decision network, does not have the drawbacks of probability thresholding, which requires every model of interest to be sequentially evaluated. Instead, it can in theory support a multi-class setting; choosing from several networks Bolukbasi et al. (2017) similarly use probability output to choose between increasingly expensive models. They show results on ImageNet (Deng et al., 2009) and provides encouraging time-savings with minimal drop in performance. This further suggest that the probability thresholding strategy is a viable alternative to exclusively using SoTA models.

4 Results

4.1 Model setup

The architecture and training details of all models are all available in section 5. In table 2 is an overview of the computational cost of our models. Our dataset is the binary version of the Stan-

Strategy	Validation AUC	Test AUC
Naïve ratio	84.84	83.77
Probability thresholding	$\mu = 86.03, \sigma = 0.3$	$\mu = \mathbf{85.49}, \sigma = 0.3$
Decision network	$\mu = \mathbf{86.13}, \sigma = 0.3$	$\mu = \mathbf{85.49}, \sigma = 0.3$

Table 3: Results for each decision strategy. The AUC is the mean value of the curve from 5. Each model is trained ten times with different initialization, and results are reported as mean and standard deviation over the ten runs.

ford Sentiment Treebank (SST), where "very positive" is combined with "positive", "very negative" is combined with "negative" and all "neutral" examples are removed.

4.2 Benchmark model

To compare the two decision strategies we evaluate the trade-off between speed and accuracy, shown in fig. 5. Speedup is gained by using the BoW more frequently. We vary the probability threshold in both strategies and compute the fraction of samples dispatched to each model to calculate average computation time. We measure the average value of the speed-accuracy curve, a form of the area-under-the-curve (AUC) metric.

To construct a baseline we consider a naïve ratio between the two models, i.e. let Y_{ratio} be the random variable to represent the average accuracy on an unseen sample. Then Y_{ratio} has the following properties:

$$\begin{cases} \Pr(Y_{\text{ratio}} = A_{\text{BoW}}) = \alpha \\ \Pr(Y_{\text{ratio}} = A_{\text{LSTM}}) = 1 - \alpha \end{cases} \tag{1}$$

Where A is the accuracy and $\alpha \in [0, 1]$ is the proportion of data used for BoW. According to the definition of the expectation of the random variable, we have the expected accuracy be:

$$\begin{aligned} \mathrm{E}(Y_{\text{ratio}}) &= \sum \Pr(Y_{\text{ratio}} = y) \times y && (2) \\ &= \alpha \times A_{\text{BoW}} + (1 - \alpha) \times A_{\text{LSTM}} && (3) \end{aligned}$$

We calculate the cost of our strategy and benchmark ratio in the following manner.

$$\begin{aligned} C_{\text{strategy}} &= C_{\text{BoW}} + (1 - \alpha) \times C_{\text{LSTM}} && (4) \\ C_{\text{ratio}} &= \alpha \times C_{\text{BoW}} + (1 - \alpha) \times C_{\text{LSTM}} && (5) \end{aligned}$$

Where C is the cost. Notice that the decision network is not a byproduct of BoW classification and requires running a second MLP model, but for simplicity we consider the cost equivalent to the probability strategy.

4.3 Quantitative results

In fig. 5 and Table 3 we find that using either guided strategy outperforms the naïve ratio benchmark by **1.72 AUC**.

4.4 Qualitative results

One might ask why the decision network is performing equivalently to the computationally simple probability thresholding technique. In 5 we have provided illustrations for qualitative analysis of why that might be the case. For example, A1 provides a t-SNE visualization of the last hidden layer in the BoW (used by both policies), from which we can assess that the probability strategy and the decision network follow similar predictive patterns. There are a few samples where the probabilities assigned by both strategies differ significantly; it would be interesting to inspect whether or not these have been clustered differently in the extra neural layers of the decision network. Towards that end, A2 is a t-SNE plot of the last hidden layer of the decision network. What we hope to see is that it learns to cluster when the LSTM is correct and the BoW is incorrect. However, from the visualization it does not seem to learn the tendencies of the LSTM. As we base our decision network on the last hidden state of the BoW, which is needed to reach a good solution, the decision network might not be able to discriminate where the BoW could not or it might have found the local minimum of imitating BoW probabilities too compelling. Furthermore, learning the reasoning of the LSTM solely by observing its correctness on a slim dataset could be too weak of a signal. Co-training the models in similar fashion to (Odena et al., 2017) might have yielded better results.

5 Conclusion

We have investigated if a cheap bag-of-words model can decide when samples, in binary sentiment classification, are easy or difficult. We found that a guided strategy, based on a bag-of-words

neural network, can make informed decisions on the difficulty of samples and when to run an expensive classifier. This allow us to save computational time by only running complex classifiers on difficult sentences. In our attempts to build a more general decision network, we found that it is difficult to use a weaker network to learn the behavior of a stronger one by just observing its correctness.

References

D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. Engel, L. Fan, C. Fougner, T. Han, A. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang, C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu. 2015. Deep speech 2: End-to-end speech recognition in english and mandarin. *CoRR* abs/1512.02595. http://arxiv.org/abs/1512.02595.

D. Bahdanau, K. Cho, and Y. Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *ICLR*.

T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama. 2017. Adaptive neural networks for fast test-time prediction. *CoRR* abs/1702.07811. http://arxiv.org/abs/1702.07811.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*.

F. Gers, J. Schmidhuber, and F. Cummins. 2000. Learning to forget: Continual prediction with lstm. *Neural Comput.* 12(10):2451–2471.

A. Graves. 2012. Neural networks. In *Supervised Sequence Labelling with Recurrent Neural Networks*, Springer Berlin Heidelberg, pages 15–35.

A. Graves. 2016. Adaptive computation time for recurrent neural networks. *CoRR* abs/1603.08983. http://arxiv.org/abs/1603.08983.

K. He, X. Zhang, S. Ren, and J. Sun. 2015. Deep residual learning for image recognition. *CoRR* abs/1512.03385. http://arxiv.org/abs/1512.03385.

S. Hochreiter and J. Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9(8):1735–1780.

D. Kingma and J. Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

S. Merity, C. Xiong, J. Bradbury, and R. Socher. 2016. Pointer sentinel mixture models. *CoRR* abs/1609.07843. http://arxiv.org/abs/1609.07843.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *ICLR (workshop)*.

A. Odena, D. Lawson, and C. Olah. 2017. Changing model behavior at test-time using reinforcement learning. *arXiv preprint arXiv:1702.07780* .

J. Pennington, R. Socher, and C. D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.

D. Ruck, S. Rogers, M. Kabrisky, M. Oxley, and B. Suter. 1990. The multilayer perceptron as an approximation to a bayes optimal discriminant function. *Neural Networks, IEEE Transactions on* 1(4):296–298.

M. Schuster and K. Paliwal. 1997. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions* .

N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *CoRR* abs/1701.06538.

D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7587):484–489. https://doi.org/10.1038/nature16961.

R. Socher, A. Perelygin, J. Wu, J. Chuang, C. Manning, A. Ng, and C. Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15(1):1929–1958.

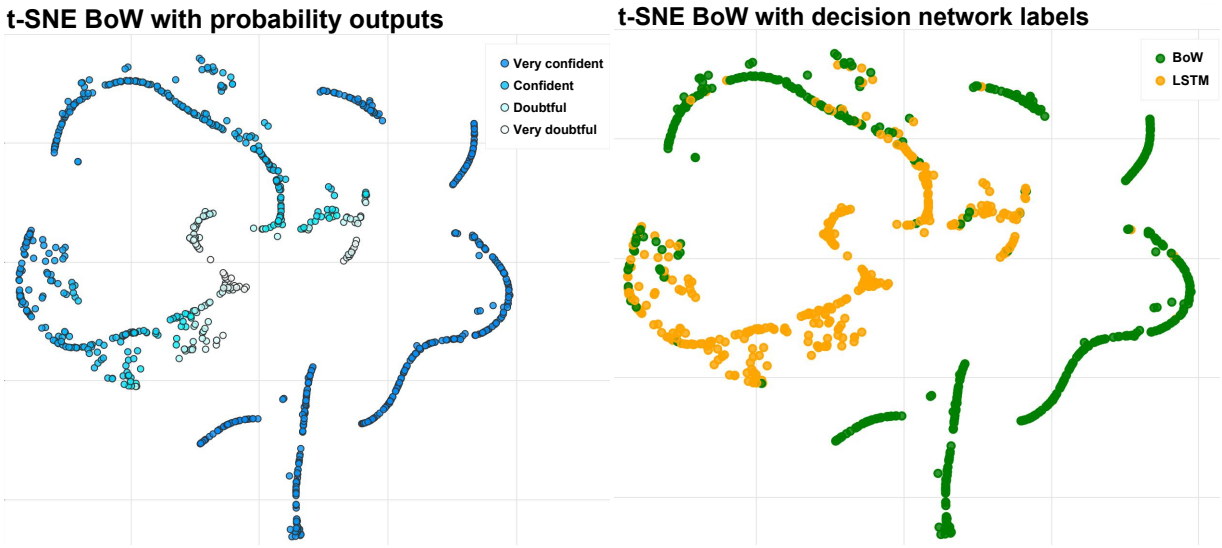
L. van der Maaten and G. Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research* 9(Nov):2579–2605.

Y. Wu, M. Schuster, Z. Chen, Q. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR* abs/1609.08144. http://arxiv.org/abs/1609.08144.

Supplementary material

Visualizations

t-SNE plots for the qualitative analysis section.



(a) Probability (b) Decision Network

Figure A1: t-SNE plot of the last hidden layer of the BoW model. The probabilities are colored by the confidence of the probability strategy. The colors in the decision network plot are the predictions of which model should be used at a given threshold of the decision network.

Model training

All models are optimized with Adam (Kingma and Ba, 2014) with a learning rate of 5×10^{-4}. We train our models with early stopping based on maximizing accuracy of all models, except the decision network where we maximize the AUC as described in section 4.2. We use SST subtrees in both the model and decision train splits and for training both models.

Model illustration: The bag of words (BoW)

As shown in fig. A3, the BoW model's embeddings are initialized with pretrained GloVe (Pennington et al., 2014) vectors, then updated during training. The embeddings are followed by an average-pooling layer (Mikolov et al., 2013) and a two layer MLP (Ruck et al., 1990) with dropout of $p = 0.5$ (Srivastava et al., 2014). The network is first trained on the model train dataset (80% of training data, as shown in fig. 4) until convergence (early stopping, at max 50 epochs) and afterwards on the full train dataset (100% of training data) until convergence (early stopping, at max 50 epochs).

Model illustration: The LSTM

The LSTM is visualized in fig. A3. The LSTM's word embeddings are initialized with GloVe (Pennington et al., 2014). Instead of updating the embeddings, as is done in the BoW, we apply a trainable projection layer. We find that this reduces overfitting. After the projection layer a bi-directional (Schuster and Paliwal, 1997) recurrent neural network (Graves, 2012) with long short-term memory cells (Hochreiter and Schmidhuber, 1997; Gers et al., 2000) is applied, followed by concatenated mean- and max-pooling of the hidden states across time. We then employ a two layer MLP (Ruck et al., 1990) with dropout of $p = 0.5$ (Srivastava et al., 2014). The network is first trained on the model train dataset (80% of training data) until convergence (early stopping, max 50 epochs) and afterwards on the full train dataset (100% of training data) until convergence (early stopping, max 50 epochs).

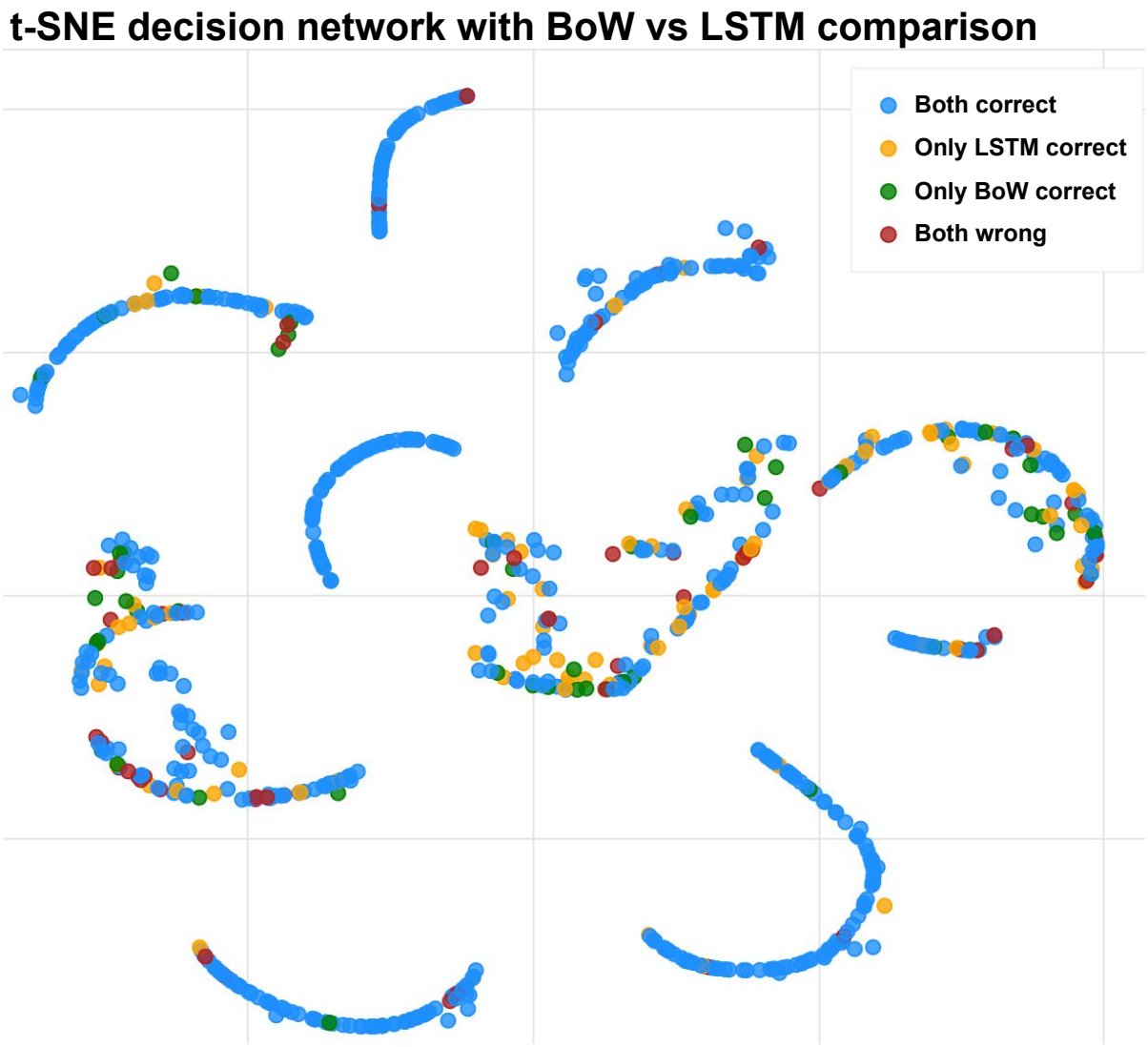
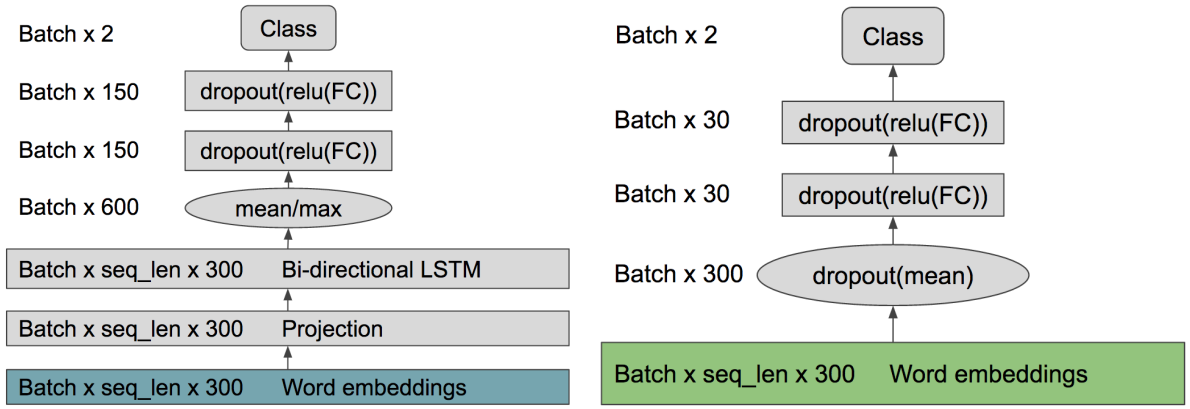


Figure A2: We compare the predictions of the BoW and the LSTM to assess when one might be more correct than the other. We train the decision network to separate the yellows (only LSTM correct) from the rest. This plot enables us to evaluate if the model is able to learn the relationship between the correctness of the two models, even though it only has access to the BoW model.

Model illustration: The Decision Network

The decision network is pictured in fig. A4, it inherits all but the output layer of the BoW model trained on the model train dataset, without dropouts. The layers originating from the BoW are not updated during training. We find that it overfits if we allow such. From the last hidden layer of the BoW model, a two layer MLP (Ruck et al., 1990) with dropout of $p = 0.5$ (Srivastava et al., 2014) is applied on top.

The network is trained on the decision train portion of the dataset (20% of training data) until convergence. We use early stopping by measuring the AUC metric between the BoW and LSTM trained only on the model train dataset.



(a) LSTM (b) Bag of words

Figure A3: Visualization of the BoW and LSTM model. Green refers to initialization by GloVe and updating during training. Grey is randomly initialized and updated during training. Turquoise means fixed GloVe vectors.

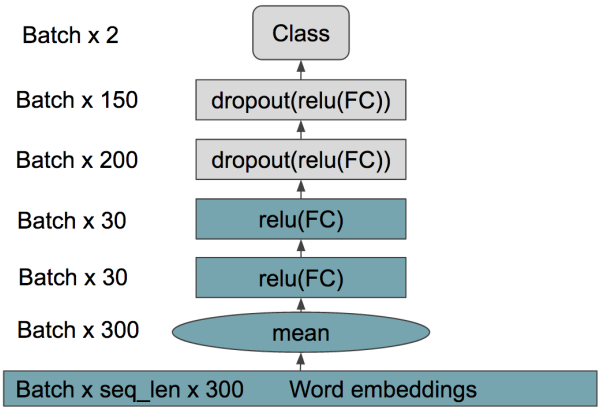


Figure A4: Architecture of the decision network. Turquoise boxes represent layers shared with the BoW model trained on the model train dataset and not updated during decision model training.