The art of deep learning (applied to NLP)

Pascal Pompey
papompey@stanford.edu

In many regards, tuning deep-learning networks is still more an art than it is a technique. Choosing the correct hyper-parameters and getting a complex network to learn properly can be daunting to people not well versed in that art. Taking the example of entailment classification, this work will analyze different methods and tools that can be used in order to diagnose learning problems. It will aim at providing an deep understanding of some key hyper-parameters, sketch out their effects and explain how to identify recurrent problems when tuning them.

1 Introduction

Getting a deep neural network to converge, let alone selecting the optimal hyper-parameters, can be a very difficult. And this difficulty grows quickly with the complexity of a neural network. Is the learning rate right? The regularizer too strong? The dropout keep rate too low? Does my model have sufficient leanring capacity? All these questions can be difficult to answer. The aim of this project is to outline some methods and tools that can be useful to diagnose learning problems when tuning deep-learning networks. Each hyper parameter is described along with a discussion of its impact and an outline on how to identify common symptoms indicating problems coming from that particular hyper-parameter.

Each method will be illustrated on a model trained to solve the use case of entailment classification. Understanding entailment and causality relationships in text is a central task in artificial intelligence. For the sake of this study, entailment classification presents the advantage of (1) requiring complex and deep network structures, (2) having a large standardized data-set with (3) score of prior literature to compare with, (4) still being simpler to train to convergence than machine translation models and (5) requiring very careful hyperparameter tuning.

We will first aim at reproducing the baselines from the original paper [1] presenting the SNLI data-set using the following sentence vector models. Based on these example we will analyze how different poor choice of hyper-parameters can be diagnosed, with a particular focus on the learning-rate , the dropout rate and the L_2 regularization rate. Three different classes of sentence models were used: sum of words, GRU and LSTM and the hyper-parameter diagnostic method were found to be invariant across all three.

1.1 The SNLI data-set

The Stanford Natural Language Inference (SNLI) Corpus aims to serve as a reference for comparing methods recognizing causality relationships in text. It can be seen as a large triple store containing triples of the type (premise, hypothesis, judgment). The aim is to develop an algorithm that, given a premise and an hypothesis sentence, correctly classifies the type of judgment linking the two. Three labels of judgments are covered in the SNLI data-set: entailment, contradiction, neutral.

By construction, the SNLI data-set is well balanced in its label distribution as, for each text, three hypothesis are always given, one for each label (entailment, contradiction and neutral). While not necessarily true for every single text, this structure is sufficiently well kept to consider the SNLI data-set to be perfectly balanced and makes the use of the simple accuracy measure (as the percentage of correct prediction) meaningful.

The SNLI data-set comes already pre-separated into training, testing and development data-sets. The training, testing and development data-set contain respectively 550152, 10000 and 10000 samples. While this does not follow the usual $(80, 10, 10)$, this predefined separation will be kept in order to compare results with prior-art. It turns out that entailment is a difficult problem so a large data-set is indeed needed to work on that problem.

2 Background / Related work

Since its publication in 2015, the SNLI data-set has attracted a lot of attention. Different network architectures have been published using, amongst others, LSTM [1], GRU [6] or CNN [7]. The current benchmark [2] considers both the premise and hypothesis sentence jointly in order to leverage word alignment information.

The cheer number of hyper-parameter requiring tuning and complexity of the neural network architecture involved makes successfully reproducing the results reported in the above studies very hard. Hyper-parameters mainly come from three sources: (1) the network dimensionality, (2) the optimization routine and (3) regularization / over-fitting mitigation techniques.

Concerning the optimization methods, the Adagrad [3] and Adam optimizers [5] have yielded better learning convergence. Adagrad enables to dynamically fine tune the learning-rate of a gradient descent method for each parameter. The Adam optimizer further adapts the Adagrad technique to stochastic gradient descent (SGD) by using an exponential distribution to model the decay of the learning rate for each parameters. By tracking the first and second moments of the gradient, the parameters of that exponential distribution can be tracked during learning. Both methods enable to automatically tune the learning-rate during training. However both method also require to be given a starting learning rate as hyper-parameter, whose value is critical to the convergence of the algorithm.

Amongst the methods to prevent over-fitting are the well known L_2 regularization and the more recent dropout method [4, 9]. L_2 regularization acts as a prior over the parameter space that penalizes parameter values that are too big and, therefore, obviously wrong. L_2 regularization relies on an hyper-parameter (noted l_2) indicating the strength of the regularization prior. The dropout mechanism works by randomly obfuscating parts of the input information to some layers in a deep-network. For the dropout method to work well, a correct keep-rate parameter needs to be set.

3 Approach

3.1 Network architecture

We used the same overall network structure as that proposed in the original SNLI paper [1]. This architecture separately learns two sentence models, then feed these two sentence models to another network learning the inference relationship. Figure 1 describes this architecture (borrowed from the original SNLI paper) in detail: two sentence models are fed to a fully connected network classifying inference based on three $tanh$ layers. Dropout applications are indicated in red, L_2 regularization in yellow and the respective layers' dimensions in green. Note that recent studies changed that structure and achieved higher accuracy by jointly leanring the the premise and hypothesis sentence word by word. This work will mianly focus on reproducing the results of the original SNLI paper.

For the **embedding layer**, the Glove [8] word embeddings were used. The rationale behind leveraging pretrained word vectors is that Glove was trained on much more data than the actual SNLI data-set contains and, as a result, was able to indentify much finer relationships between words. In our experiments, we found that over-fitting could be mitigated without impacting prediction accuracy by keeping the Glove vectors constants.

For the **sentence representation layer**, three methods were used. The sum of all the words in the sentence was used as baseline, and LSTM and GRU, both well known for their suitability for language modeling, were subsequently tested.

Finally the **classification layer** is a standard densely connected layer made of three $tanh()$ layers in series.

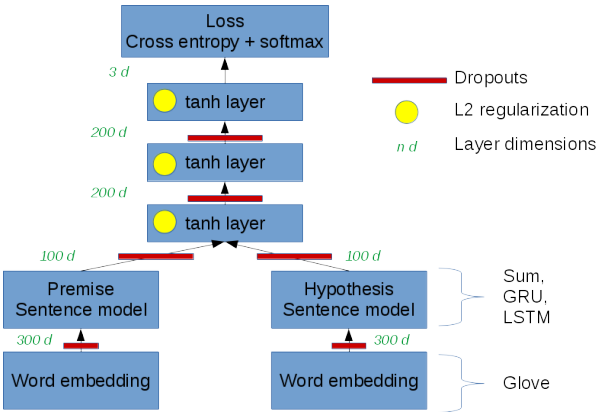

Figure 1: Network architecture

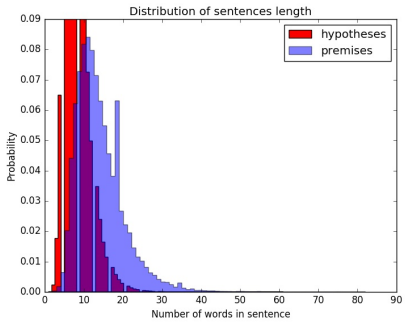

Figure 2: Sentence length distribution

This study is solely focused in the learning rate, dropout and regularization hyper-parameters. Hence other hyper-parameters like the size of each respective layer will be not be covered in this work.

3.2 Preprocessing

Unknown words and typos: as the main glove matrix is 5Gb big, a preprocessing step was required to select only the words that were actually present in the SNLI data-set. Doing this showed that there were a number of sentences in the SNLI data-set with words not present in the Glove data. Further inspection revealed that these words were typos and they were subsequently mapped to a special $< unk >$ token. The Glove matrix was extended with one further word vector and a new dimension for the $< unk >$ token. The $< unk >$ token word vector had value all zero except 1 for the newly added dimension. All the other words had value 0 for the added dimension. In short, the dimension added to the Glove matrix acted as a boolean encoding whether that particular word is unknown.

Sentence size and padding: tensorflow doesn't handle tensor with variable length. We used padding to enforce a uniform size across all the sentences. Padding consists in adding $< pad >$ tokens at the end of a short sentence to bring it to the expected length. As for the $< unk >$ token, the Glove matrix was extended with one entry and one dimension to encode whether a token was a $< pad >$filler. It remained to decide what the maximal sentence length shall be and this required analysis of the sentences' length distribution. As visible on figure 2, the hypothesis sentences have a fairly sharp distribution while the premise sentences have an heavy tail distribution with few sentences being ridiculously long (90 words). We curtailed the sentence size to 20 words (resp. 30) for the hypothesis sentences (resp. premise) which amounted to 1% (resp. 3%) of the samples being trimed to a smaller size.

4 Experiments

4.1 GRU vs LSTM vs sum of words

In our case, the optimal models were found for a learning rate of 10^{-3}, a dropout keep rate of 85% and a regularization strength of 5.10^{-6}, except for the sum of words model for which the best model was reached without any regularization (i.e. $l_2 = 0$). This study only focuses on these three parameters. The network structure and dimensions were therefore kept constant at exactly the values indicated in figure 1. We were not able to reproduce the results from the original SLNI paper and the sum of words baseline proved very hard to beat. GRU and LSTM proved very sensitive to the values of the dropout, learning rate and regularization parameters, which motivated the following study.

Accuracy	Training	Testing
Sum	**0.8864**	**0.7613**
GRU	0.7855	0.7604
LSTM	0.7806	0.7456

Table 1: Accuracy (% of correct predictions) achieved on the training and testing sets depending on the sentence model

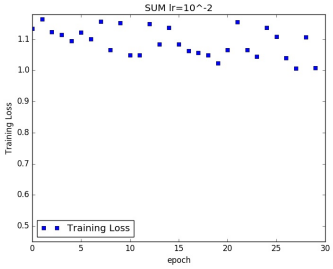

Figure 3: lr too big

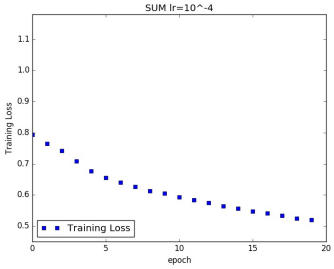

Figure 4: lr just right

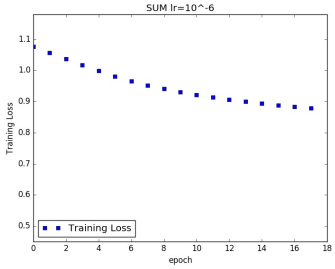

Figure 5: lr too small

Figure 6: Convergence through epoch depending on the learning rate hyper-parameter

4.2 Learning rates

Description: the learning rate determines the magnitude of change of the model's parameters for each batch. State of the art method such as Adagrad and Adam already optimize that parameter during training but still require a starting learning rate.

Intuition: a start learning rate that is too high will cause the optimization method to pass the length where a gradient direction is a descending direction. In this case, the optimization routine is, in effect, just randomly sampling the parameter space, leading to an heratic learning. At the other extreme, learning rate values that are too small willlead to poor and slow convergence.

Diagnosing learning rate issues can be done by tracking the evolution of the training loss across epochs. A learning rate that is too high will lead to the behavior of figure 3 where the training loss randomly oscillate at high values. On the other hand, a learning rate that is too low will lead to the behavior of figure 5 presenting a slow convergence stuck at a high loss score. Ideally, one shall observe a curve similar to that of figure 4, where each epoch leads to a visible monotonic decrease in the target loss.

4.3 Dropout rates:

Description: Dropouts was introduced in [4] and further discussed in [9]. Dropout is based on the keep ratio (noted do) which is the probability for any network unit subject to dropout to be retained. It intuitively prevents over-fitting by randomly obfuscating information in the network. This random obfuscation prevents the network units from co-adapting and to over-fit small details of the training set that are likely to be simply noise. Therefore dropouts force the network to only account for patterns in the data-set that occur often enought so that the dropout obfuscation cannot mask them.

Hyper-parameter intuition: it is quite easy to get a good intuition of the effect the dropout keeping rate parameter can have on learning. On one extreme, if the keep rate is 0, then the higher network layers gets no inputs, and consequently can't learn. On the other extreme, if the keep rate is 1, every unit in the network sees all the training data and therefore these unit can jointly align to match the training set almost perfectly, including its noise, hence leading to over-fitting.

Diagnosing dropouts problems can be done by tracking the training and validation accuracy across different epoch. Using the above intuition, if the dropout keep rate is too low, the network won't see enough of the input data-set to learn efficiently and therefore both training and validation accuracy

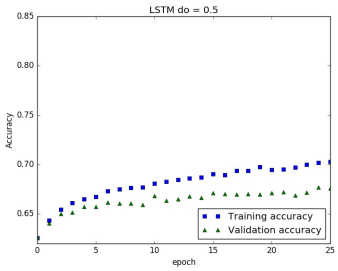

Figure 7: do 0.5 (too small)

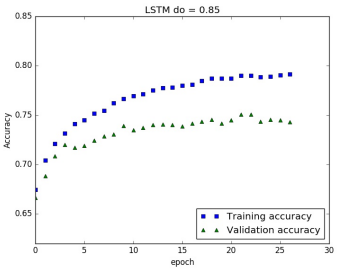

Figure 8: do=0.85 (right)

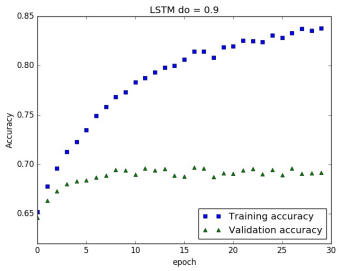

Figure 9: do=0.9 (too high)

Figure 10: Convergence through epoch depending on the dropout keep rate hyper-parameter when training LSTM sentence models on the SNLI data-set

will remain now. This is visible in figure 7 for example where the keep rate of 0.5 (which is too small for NLP problems) prevents the network from learning, leading to poor accuracy convergence.

A keep rate that is too high, however, leads to overfitting, which, as visible in figure 9, can be identified by seeing the training accuracy improve quickly while the validation set accuracy remains poor and stagnant. Figure 8 shows what could be expected with correct a dropout parameter: both the training and validation accuracy improve consistently throughout the epoch, the network can see enough of the input data-set through the dropout to learn yet, not enough to over-fit to every noisy details of the training set.

4.4 L_2 regularization rates

Description: L_2 regularization works by inducing a prior over the parameter set which imposes that parameters have small values (closer to zero). Thes enables to penalize big parameter values, wich usually result from numerical instabilities and not usefull learning. Regularization is most often expressed using half the L_2 norm of the parameter vector as this function is trivially differentiable.

Hyper-parameter intuition: the bayesian interpretation of L_2 regularization is useful here. With θ being the parameter vector, we have $P(\theta, data) = P(data|\theta)P(\theta)$ where $P(data|\theta)$ is the likelyhood of the parameter vector given the data and $P(\theta)$ is the prior over the parameters space. If the regularization parameter l_2 is too big, then the prior over-shadows $P(data|\theta)$ and forces the parameters to zero. If l_2 is too small, θ is free to fit the data perfectly, potentially leading to over-fitting.

Diagnosing regularization rates problems can be done by tracking both the training accuracy and the training loss across different epoch. Why? Because accuracy metric only tracks how θ fits the data and therefore only tracks $P(data|\theta)$. The loss metric, on the other hand, tracks the combined constraints of fitting the data and respecting the prior and therefore tracks the full posterior distribution $P(\theta, data) = P(data|\theta)P(\theta)$. If the prior is too strong, then the posterior distribution $P(\theta, data)$ will be dominated by $P(\theta)$ and the accuracy score will diverge from the loss score. Interestingly, the experimental results (reported in figure 14)did not comfirm this analysis. This is understandable for the lack of over-fittingvisible in figure on figure 11 but figure 13 with a high l_2 parameter is puzzling.

5 Conclusion

In this study, we tried to reproduce the original results reported in the SNLI paper. Doing so required to implement a complex and quite deep network architecture. It also showed how critical correct hyper-parameters are to model learning. Drawing from our experience tuning the learning-rate, dropout and l_2 hyper-parameters, we reported some methods we used to diagnose poor learning convergence and identified some key symptoms that can be leveraged whe diong hyper-parameter tuning. Each of the parameters were analyzed in details and the different scenarios were illustrated based on experiments run on the SNLI data-set. The hyper-parameter tuning method described proved to be invariant to the type of sentence model chosen (be it GRU, LSTM or sum of words).

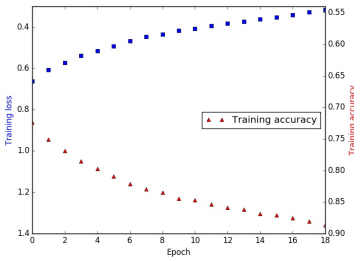

Figure 11: l2 = 0

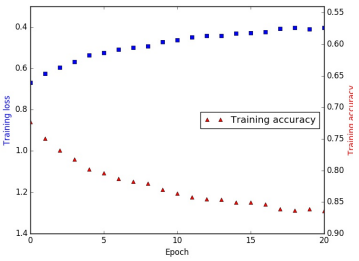

Figure 12: l2 =10^{-6}

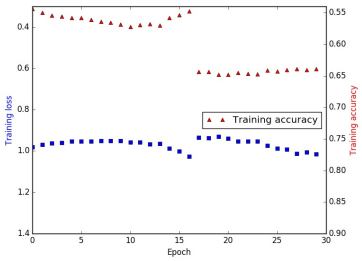

Figure 13: l2=0.02 (too high)

Figure 14: Convergence through epochs depending on the regularization hyper-parameter when training Sum of words sentence models on the SNLI data-set

6 References

References

[1] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2015. 1, 2, 3.1

[2] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *CoRR*, abs/1601.06733, 2016. 2

[3] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011. 2

[4] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. 2, 4.3

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 2

[6] Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Skip-thought vectors. *CoRR*, abs/1506.06726, 2015. 2

[7] Lili Mou, Rui Men, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. Recognizing entailment and contradiction by tree-based convolution. *CoRR*, abs/1512.08422, 2015. 2

[8] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. 3.1

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. 2, 4.3