QANet without Backtranslation on SQuAD 2.0

Stanford CS224N Default (IID) Project

Jacky Huang
Department of Computer Science
Stanford University
jackyh@stanford.edu

Abstract

This paper investigates two different approaches to the question answering problem on the SQuAD 2.0 dataset [1]. We explore a baseline model based on the BiDaF architecture [2], and improve its performance through the implementation of character embeddings and hyperparameter tuning. Further, we implement variations on the convolution and self-attention based QANet architecture [3], and explore an alternative technique for data augmentation. Finally, we form an ensemble model based on our different experiments which achieves an F1 score of 70.340 and an EM score of 67.354 on the test set.

1 Introduction

The question answering task is a very important subfield of Natural Language Processing as it has many applications in different contexts, from phone assistants like Siri to Google Search. In this task, the goal for the model is to process and understand text and to return accurate answers to queries about the text. The rise in popularity of deep learning over the past decade has contributed to renewed interest in the reading comprehension field, leading to an outpour of research papers and rapid development in this area. Two examples of (at the time) state of the art performance on the Stanford Question Answering Dataset (SQuAD) [4], a question answering dataset based on Wikipedia articles, include the Bidirection Attention Flow (BiDaF) architecture proposed by Seo et al [2], and the QANet architecture proposed by Yu et al [3].

In this paper, we explored and improved upon both architectures on the SQuAD 2.0 [1] dataset, which is an extension of the original SQuAD dataset with around 50% more queries that have no answer in the given context. First, we developed a respectable baseline by implementing character embeddings in the starter BiDaF code and performing hyperparameter tuning. We then implemented the QANet architecture and explored different architectural changes, such as changing the hidden size and the number of convolutional layers within an encoder block, as well as modifying the formula for stochastic depth dropout as defined by Huang et al [5] from a linear decay based rule to an exponential decay based rule. We also introduced a simple method for data augmentation which significantly improved the F1 and EM scores. Finally, we implemented an ensemble model by applying the plurality voting algorithm to the predictions from our different models to generate our ensemble prediction, ending up with F1/EM scores of 72.647/69.820 on the validation set and 69.041/65.883 on the test set.

2 Related Work

In recent years, Recurrent Neural Network (RNN) based architectures have been popular in the field of Natural Language Processing, and in particular the task of question answering (QA). A major reason for its popularity is its success in modelling sequential inputs like text. Usually coupled with some kind of attention mechanism to capture long term dependencies, many researchers have developed models based on RNNs to QA tasks such as the SQuAD/SQuAD 2.0 dataset [4, 1]. One such model is the Bidirectional Attention Flow (BiDaF) model [2], which uses bidirectional LSTMs

within its encoder layers, and attention layers from both context to hidden and hidden to context directions. We implement the BiDaF model as our baseline for this paper.

One drawback to using RNNs for these tasks is that it can be slow for both training and inference, as we are unable to parallelize our computations due to the sequential nature of the network. This bottleneck presents many problems to the overall machine learning pipeline. For example, if inference is slow, then the solution does not scale well with the size of the test samples. This would mean that it would be difficult to deploy this model in a real time setting.

Researchers understood the problems of using recurrent networks in QA settings as discussed above, and were motivated to design a new architecture that is fast in both training and inference, leading to the QANet architecture [3] by Yu et al. They explore a novel approach to the question answering task with a different architecture based on convolutional layers and self-attention. By avoiding recurrency in the architecture and instead relying on the easily parallelizable convolutional layers, they are up to 13x faster in training their model and up to 9x faster in inference compared to similar RNN based architectures. Compared to the BiDaF model in [2], the authors demonstrate that their model achieves the same level of accuracy in just 3 hours compared to 15 hours for the BiDaF.

They are able to take advantage of this speed increase by implementing an interesting data augmentation technique known as backtranslation, where the training data is first translated to a foreign language (say French) with a machine translation system, and then translated back to English. This backtranslated text is then appended to the training data. At the time of publication, the QANet architecture was state of the art in the context of the QA task on the SQuAD dataset.

However, the dependency on the neural machine translation (NMT) system in the data augmentation phase presents certain problems. Firstly, the quality of the NMT system has a direct effect on the effectiveness of the model. It is very easy for semantics to be lost in the backtranslation, leading to incorrectly labelled training data. Further, there is a lot of overhead in the training process as the time taken to perform the backtranslation is non trivial and should not be dismissed.

Thus, a natural next step in building upon the work of Yu et al [3] is to remove the dependency on NMTs for data augmentation. This paper explores alternative methods of data augmentation that do not have any external dependencies.

3 Approach

All new ideas and code, except for the provided BiDaF baseline and the implementation of the depthwise separable convolutions, was original.

3.1 BiDaF baseline

Our first goal was to improve upon the existing baseline model given in the project handout. More information about this baseline can be found in the handout and the original BiDaF paper [2] by Seo et al. Our first improvement was to implement character level embeddings [6] as in the original BiDaF architecture, as the baseline did not have this feature. These character level embeddings have certain advantages over word embeddings, such as being able to handle Out of Vocabulary (OOV) words much better.

For each word in both the context and the query, we first obtained a matrix of character embeddings for the word by looking up each character in a pre-trained character embedding. We then applied a 1D convolutional neural network to that matrix. Finally, max pooling was used in the word length dimension to make the dimension of the embedding fixed. These character embeddings were then concatenated with the word level embeddings before being passed to the highway encoder. This proved to be an effective technique for improving our scores.

3.2 QANet

3.2.1 Architecture

Implementation of the base QANet model followed. The network architecture of the QANet is similar in style to the BiDaF baseline and many other QA models. As seen in Figure 1, it consists of five

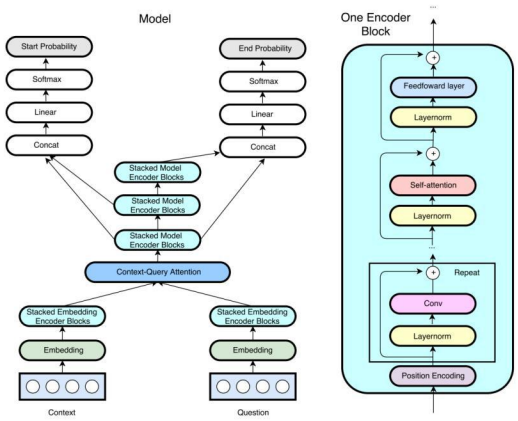

Figure 1: QANet architecture diagram [3].

layers; the embedding layer, the embedding encoder layer, then context-query attention layer, the model encoder layer, and the output layer.

Input Embedding Layer. This layer is very similar to its BiDaF counterpart. Both the context and question texts are embedded with the same embedding layer (i.e. the weights are shared). The words are embedded into pre-trained 300 dimensional GloVe embeddings [7], and the characters into a trainable 200 dimension embedding using a 1D convolutional neural network. For each word, the embeddings are then concatenated into a 500 dimensional vector. This vector is then passed through a two layer highway network as described in [2], which is the output of the layer.

One of the variations that we explored was to replace the input embedding layer with the BiDaF version, namely replacing the trainable character embeddings with a fixed 64 dimensional embedding provided as part of the starter baseline code. This turned out to provide a small boost in the F1 and EM scores.

Embedding Encoder Layer. The core insight of the QANet architecture comes from the individual encoder blocks seen on the right of Figure 1. QANet does away with the LSTM based encoder blocks and replaces it with depthwise separable convolutions [1] [8], multi-head self-attentions [9], and feed forward networks, which as mentioned before is much more parallelizable. Each of these layernorm + convolution/attention/feedforward layers are wrapped in a Residual Block which helps with gradient flow due to the increased depth of the network. Finally, as there is no recurrency in our network, we add positional encoding as in [9] to the input of the encoder block in order to encode information about the relative positioning of the words.

The motivation behind this approach is that the convolutions would capture the local signals in the text, while the self attention layer would learn the global interactions between words. Each of the cyan coloured encoder blocks on the left of Figure 1 has the same structure as the right block, except for tunable hyperparameters such as the number of convolutional layers per residual block, the number of encoder blocks used, and the kernel size. For the embedding encoder layer, these numbers are 4, 1 and 7 respectively. The weights are shared between the Context/Question encoders, and between the output encoders.

Furthermore, stochastic depth dropout [5] is used as a regularization technique by setting the *survival probability* of a ResBlock ℓ to be $p_\ell = 1 - \frac{\ell}{L}(1 - p_L)$, where ℓ is the current sublayer, L is the total number of layers in the whole network, and $p_L = 0.9$. Thus, each ResBlock has a $1 - p_\ell$ chance of being dropped with each training batch. We explore an alternative exponential decay based survival probability function $p^*_\ell = \exp(-p^*_L \cdot \frac{\ell}{L})$, where p^*_L is a hyperparameter, with the idea that the earlier layers are more important and thus should be preserved much more often than the later layers, as seen

[1] We got the idea for how to implement the depthwise separable convolutions from https://discuss.pytorch.org/t/how-to-modify-a-conv2d-to-depthwise-separable-convolution/15843/7.

in Figure 3. Thus as the expected number of active layers in each batch is decreased, this allowed us to explore much deeper networks by quadrupling the number of convolution layers in each encoder block and doubling the hidden size.

Context-Query Attention Layer. For our experiments, this layer was unchanged from the provided baseline BiDaF model so we refer the reader to the handout and the original BiDaF paper by Seo et al [2].

Model Encoder Layer. These are similar to the Embedding Encoder Layer except for small variations in the hyperparameters. In this layer, the number of convolutional layers per residual block is 2, the total number of encoder blocks is 7, and the kernel size is 5. The weights are shared between each of the three model encoder layers. Each of the three layers outputs a matrix M_j to be used in the span prediction.

Output Layer. This layer is responsible for the prediction of the model. Like the BiDaF architecture, QANet also predicts the probabilities of each position being the start and end of an answer span. However, the way it computes the probabilities is different. The probabilities are computed as

$$p^1 = softmax(W_1[M_0; M_1]), p^2 = softmax(W_2[M_0; M_2]),$$

where the W_i are trainable matrices and the M_j are the outputs of the model encoder layers. The predicted span is then the pair of indices such that the product of the probabilities is maximized. In the case of SQuAD 2.0, we append an OOV token to the start of the text so that we can predict no answer in the case the maximal product is between $p^1[0]$ and $p^2[0]$.

3.2.2 Data Augmentation

Data augmentation via backtranslation plays an important role in the success of the QANet architecture. However, the dependency on machine translation models has its downsides as mentioned previously. Thus, we investigate an alternative augmentation algorithm which is lightweight and can scale.

We run through the entire training set once, and for each query with an answer, we either 1) convert it to an unanswerable query by firstly removing the answer span from the context entirely and then dropping 10% of the remaining words at random, or 2) creating a new answerable query by concatenating another random context to the end of the current context. We pick each option with equal probability.

The motivation behind option 1 was that it was an easy method of generating unanswerable queries with a high probability of success. However, we cannot guarantee that every new sample generated is correct, as for example the context may contain the answer elsewhere. Option 2 was designed to train the model to better handle noisier/less curated English, because for example the super-context might be disjointed in the sense that it could start off talking about baseball and then randomly end up talking about volcanos. This would help the model to pick up the specific subtext which is relevant to the query.

4 Experiments

4.1 Data

The dataset used is the SQuAD 2.0 dataset [1], which is a collection of passages (context) from Wikipedia along with a crowdsourced question (query) and answer on each of these passages. In the original SQuAD dataset [4], answers to the query can be expressed as a *span* from the passage; that is, a contiguous collection of words with a well defined start and end point. Thus, it is enough for models working on the SQuAD dataset to predict the start and end points of the span which it thinks contains the answer to the query.

The main difference between SQuAD 2.0 and the original SQuAD is the addition of questions which cannot be answered by their corresponding context. Thus, models now have to decide whether or not the question is answerable before predicting the span (if applicable). We note that for the purposes of this project, the dev and test splits are different to the publicly available SQuAD 2.0 dataset for reasons mentioned in the handout.

4.2 Evaluation method

Our evaluation metrics are the F1 Score and Exact Match (EM). As described in the handout, the EM metric is a boolean metric that gives a score of 1 if and only if the predicted span matches the given answer exactly, and 0 otherwise. The F1 score is the harmonic mean of precision and recall, and as such will give partial credit to the model if the predicted span has some overlap with the ground truth.

We note that as the answers to the queries can be subjective, the SQuAD dataset provides three ground truths for each query in the dev and test sets. For each prediction, the metrics are computed against all three ground truths and the maximum is taken to be the score for that sample. The overall score for the model is the average of all the scores for samples in the dataset.

For this project, the evaluation metrics will be obtained by uploading the predictions into the GradeScope leaderboard, which will then compute the scores.

4.3 Experimental details

4.3.1 BiDaF

The provided baseline was trained and evaluated with no changes at all to its configuration or hyperparameters. Some code changes were needed to implement the BiDaF with character level embedding architecture. We decided to go with a 1D CNN where the number of in channels was the dimension of the character embeddings (64), the number of out channels was the hidden size (100) and the kernel size was 3. The learning rate was fixed at 0.5 for both experiments along with the AdaDelta optimizer. We trained for a total of 30 epochs, taking around 3 hours for both models. We started with a dropout of 0.2 but after observing a lot of overfitting, we did some hyperparameter tuning with respect to dropout and L^2 weight decay, and found that a dropout of 0.25 with no weight decay worked best. The batch size was 64.

4.3.2 QANet Baselines

Our first QANet baseline has the same configuration as the original architecture proposed by Yu et al [3], with the exception that there is no learning rate warm up; the learning rate is fixed at 0.001. We use a hidden size of 128 and 0.1 dropout between layers, and 0.1/0.05 dropout for word and character embeddings respectively. Adam optimizer is used with $\beta_1 = 0.8, \beta_2 = 0.999, \epsilon = 10^{-7}$, with L^2 weight decay of 3×10^{-7} and an exponential moving average decay of 0.999. The number of heads in the self attention layer was 8, and the number of convolution layers, kernel size and number of encoder blocks as described above. The batch size was 16 as opposed to 32 due to GPU memory limitations.

We then modified our baseline to use the same embedding layer as the BiDaF models, using the pretrained 64 dimensional character embeddings. Everything else was constant. Training time for both models took around 11 hours on an NVIDIA 2080TI GPU.

4.3.3 Deeper QANet

We were interested to see if making the model deeper would improve the predictions. In particular, whether increasing the hidden size of the network as well as the number of convolutional layers in each residual block would improve the generalizability of the model. As mentioned previously, we explored the effects of doubling the number of hidden units in the network, as well as quadrupling the number of convolutional layers. In order to reduce overfitting, we explored different dropout values as well as implemented the exponential decay idea for stochastic depth dropout [5]. The idea for this is to have relatively small networks during training which help to reduce training time as well as overfitting, but have the power of a full sized network for inference. To account for the memory usage, batch size had to be dropped to 8.

4.3.4 QANet Data Augmentation

Our algorithm for data augmentation was defined in a previous section. However, we explored different thresholds for triggering which kind of augmentation technique. In the end, we found that keeping it at a 50/50 split for the answerable questions provided a nice balance in the dataset, and the models were able to train nicely. We also explored the effects of augmenting more than once for

each example, however this lead to memory issues. On the other hand, we also tried out augmenting each answerable question with only 50% probability. This ended up doing quite well but due to the smaller amount of training data also started plateauing earlier.

4.3.5 Ensembling

Our final idea was to combine our previous models together to form an ensemble. For each query in the test set, we look up the predicted answer spans for each of our models in the ensemble and output the span with the most votes. We break ties by picking the span which was produced by the model with the highest F1 score.

4.4 Results

			Dev		Test	
Model	Hidden	Dropout	EM	F1	EM	F1
BiDaF Baseline	100	0.2	58.259	61.586	-	-
BiDaF Baseline + Char Emb	100	0.2	60.645	64.134	-	-
BiDaF Baseline + Char Emb	100	0.25	**61.452**	**65.032**	-	-
QANet	128	0.1	64.383	67.821	61.183	64.957
QANet + Pretrained Char Emb	128	0.1	66.207	69.447	-	-
QANet + PCE + 4x conv + 0.3 Exp Decay	128	0.1	65.485	69.084	-	-
QANet + PCE + 2x conv + 0.6 Exp Decay	256	0.1	65.854	69.240	-	-
QANet + PCE + Augment	128	0.1	**67.669**	**71.054**	**63.804**	**67.427**
Ensemble	-	-	**69.820**	**72.647**	**67.354**	**70.340**

Table 1: Results from the leaderboard.

Table 1 presents a selection of models we trained. Our best model is the Ensemble model, achieving test scores of 70.340 and 67.354 for F1 and EM respectively, which at the time of writing puts us in second place on the IID leaderboard.

As expected, the provided BiDaF baseline was by far the worst model. After implementing the char embeddings as the first step, we were able to improve the dev F1 score by over 2.5 points, highlighting the importance of character embeddings. Some effort was spent in the context of hyperparameter tuning the BiDaF model for the milestone, focusing mainly on regularization in order to avoid overfitting. We tried many different configurations changing both the dropout values as well as the L^2 weight decay, and it was surprisingly sensitive to get a good set of hyperparameters. Our best attempt netted us a 0.9 increase in the dev F1 score.

The QANet immediately performed well right from the baseline, having a significantly higher dev F1 score from the moderately tuned BiDaF model. This is likely due to the efficiency of the non recurrent nature of the model, allowing us to stack deep layers into the network. The running time for the QANet was slower than the BiDaF, however it comes with the benefit of having a much more complex network that is capable of understanding deeper patterns, as shown by the results.

Our first improvement to the QANet model was from changing the trainable 200 dimension character embeddings in favour of the 64 dimensional pre trained embeddings as used in the given BiDaF model. This was quite a surprising result, as we expected that a higher dimensional embedding would be more useful for the model. On the other hand, we would be training something from scratch whereas the pre trained ones are ready to go out of the box and probably fine tuned to be very effective.

The two exponential decay based stochastic dropout models performed slightly worse compared to the pretrained character embedding QANet model. Although disappointing, this was not much of a surprise as we were already seeing issues with overfitting in the latter model, so increasing the complexity of the model was unlikely to improve the result. Unfortunately we ran out of time to run an experiment with only the stochastic dropout.

On the other hand, the data augmentation algorithm did much better than we anticipated, improving the pretrained char embedding baseline by 1.6 dev F1 points. We expected it to improve the F1 score by much less. In part, this was designed to handle the overfitting issues we saw in previous iterations

of the model. Furthermore, we wanted to remove the dependency on machine translation systems by proposing a quicker and simpler method of augmentation.

Finally, the ensemble model performed as we expected. By combining the various models together, we are able to leverage all of the training done to aid in our predictions. It was not a surprise that we were able to increase both the dev and the test scores by a significant margin as compared to the best single model we had.

5 Analysis

5.1 Question Type Analysis

We will firstly consider the different types of questions that are asked in the dataset, as seen in Table 2. Comparing the scores between the best ensemble model and the best QANet model, we can see that the scores match up fairly well. Both models do well on "When" type questions, and both do poorly on "Why" type questions. It makes sense that "When" type questions score high, since they are likely quantitative ("When was Shakespeare born") and the solution is unlikely to be subjective. However, "Why" type questions may require deep understanding of the text and the ability to relate different words to each other, making it a somewhat harder task. On the other hand, it is somewhat surprising that "Where" type questions do fairly poorly; one would think that it is in the same category as "When" type questions with a fairly well defined answer.

		Ensemble			**QANet**		
Type	**Count**	**F1**	**EM**	**AvNA**	**F1**	**EM**	**AvNA**
Total	6078	0.72	0.69	0.77	0.71	0.67	0.76
What	3726	0.72	0.69	0.77	0.70	0.66	0.76
Who	714	0.73	0.71	0.76	0.72	0.70	0.77
How	605	0.69	0.65	0.74	0.70	0.65	0.75
When	561	0.78	0.77	0.81	0.77	0.75	0.81
Where	275	0.67	0.63	0.74	0.66	0.61	0.72
Which	274	0.75	0.72	0.81	0.72	0.68	0.80
Why	82	0.69	0.61	0.76	0.65	0.56	0.73
Other	59	0.55	0.49	0.67	0.58	0.52	0.71

Table 2: Breakdown of scores per question type.

It also makes sense that the performance between the Ensemble and the QANet model is so similar throughout the matrix. This is likely due to the tie breaking mechanism in our ensembling code, where spans with the same amount of votes would be decided by comparing the largest F1 scores of the models that predicted that span. Indeed, we enforce the tie breaker logic on around 12% of the validation dataset.

5.1.1 Analysis by category

We will now investigate how the model does on certain categories, as seen in Figure 2. It is immediately clear that the categories are not even. For example, European Union Law has much higher scores than Packet switching. By looking at the ratio of answerable to unanswerable questions as in Figure 3, we can see that Packet Switching has significantly more unanswerable questions compared to European Union Law. We will investigate what exactly is tripping up the model.

Consider the following passage:

> The National Science Foundation Network (NSFNET) was a program of coordinated, evolving projects sponsored by the National Science Foundation (NSF) beginning in 1985 to promote advanced research and education networking in the United States. NSFNET was also the name given to several nationwide backbone networks operating at speeds of 56 kbit/s, 1.5 Mbit/s (T1), and 45 Mbit/s (T3) that were constructed to support NSF's networking initiatives from 1985-1995. Initially created to link researchers to the nation's NSF-funded supercomputing centers, through further public funding and private industry partnerships it developed into a major part of the Internet backbone.

	Packet Switching	European Union Law
Answerable	104	231
Unanswerable	269	190
Total	373	421

Table 3: Answerable vs Unanswerable questions for Packet Switching and European Union Law

The query on this context is "NSF began in 1985 to promote what?". This is quite a tricky question. Firstly, the acronyms "NSF" and "NSFNET" are really similar and are closely related. Furthermore, the query itself seems to lead the reader to believe that NSF actually did promote something in 1985, when really it was NSFNET. We found quite a lot of similar adversarial examples in the Packet Switching topic, which could possibly indicate why our model did quite poorly. We can improve our model to be more robust by constructing more tricky questions in our training set.

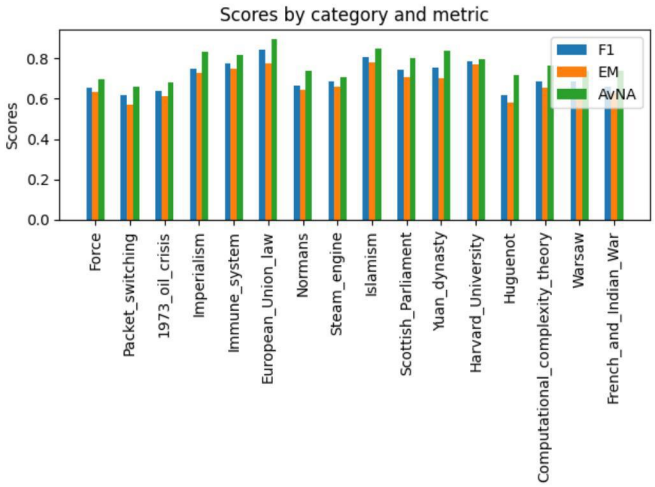

Figure 2: Evaluation Metric scores on different categories with QANet model.

6 Conclusion

This paper explores different deep learning approaches to the question answering task with the SQuAD 2.0 dataset. We implemented a baseline BiDaF model with character level embeddings, and performed hyperparameter tuning on the dropout and L^2 weight decay parameters in order to reduce overfitting.

Furthermore, we implemented the QANet architecture and experimented with some variations on the architecture. We also introduced a simple but effective data augmentation algorithm that does not have a dependency on machine translation systems. Finally, we created an ensemble model based on the above and achieved an F1 score of 70.340 and an EM score of 67.354 on the test dataset.

Limitations in the paper include the lack of resources, including time and computational power. Having more of both would've allowed for more experimentation and the ability to test more ideas, in particular the exponential decay based stochastic dropout method which did not work as well as we hoped. Further work in this regard, assuming the resources are available, include defining even deeper models and more aggressive data augmentation to see if that can help improve performance.

7 Appendix

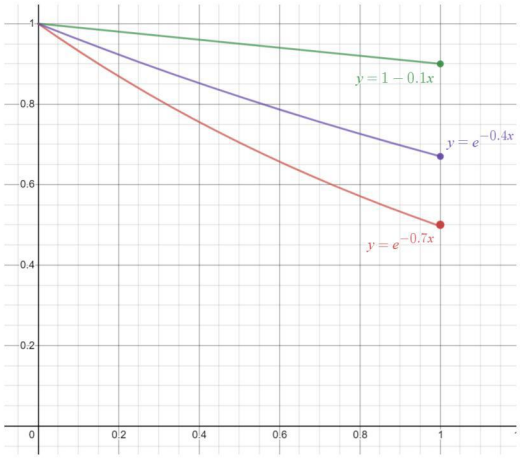

Figure 3: Survival probability functions plotted on https://www.desmos.com/. Green corresponds to the default stochastic dropout layer, and purple/red corresponds to the exponential decay based formula with different hyperparameters.

References

[1] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.

[2] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2018.

[3] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension, 2018.

[4] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics (ACL).

[5] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep networks with stochastic depth, 2016.

[6] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification, 2016.

[7] 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.

[8] François Chollet. Xception: Deep learning with depthwise separable convolutions, 2017.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.