Fine Grained Gating on SQuAD

Stanford CS224N Default Project

Aaron Kaufer
Department of Computer Science
Stanford University
akaufer@stanford.edu

Abstract

The purpose of this project is to implement an embedding mechanism on top of the BiDaf model that serves as a compromise between word-level embeddings and character-based embeddings that can compete with a simple concatenation of word and character level embeddings. In particular, the mechanism is what is called a fine-grained gating method (explained in [1]), in which, given a character level embedding $c \in \mathbb{R}^h$ and a word-level embedding $d \in \mathbb{R}^h$ a parameter $g \in \mathbb{R}^h$ is learned such that final embedding of a given word is $g \odot c + (1 - g) \odot w$, where $\odot$ represents termwise multiplication. After various experiments varying the methods by which the parameter g is learned, results ultimately show that none of the fine-tuned gating methods perform better than mere concatenation of the word and character embeddings.

1 Key Information to include

- Mentor: Zihan Wang
- External Collaborators (if you have any): N/A
- Sharing project: N/A

2 Introduction

When it comes to question-answering tasks, the first step in most models is to calcualte some sort of embeddings. Embeddings need to be calcualted for both the contexts and the queries before being fed into any sort of sequential models like an RNN. The way that these embeddings are calculated is very important, and there are two general themes: word-level embeddings and character-level embeddings. Word-level embeddings are good when it comes to learning the semantics of words, especially in relation to other words, whereas character-level embeddings are good at modeling individual sub-word morphologies. For example, a word-level embedding would need lots training data to learn that "work" and "worked" are related, by seeing them used in similar contexts, whereas a character level embedding could spot the similarity of stems.

Character-level embeddings and word-level embeddings both offer their own strengths and weaknesses, so a common tactic to strengthen a question-answering model is to find some way to combine the character-level and word-level embeddings of a given word. The most straigthforward way to do this would be to simply concatenate the two different embeddings and use that as the final embedding of the word. This allows a model to see information relating to both the character and word level embeddings of a word when feeding it through an RNN. Another possible approach would be to average the character and word level embeddings of a word. This requires less parameters, but also loses more information, and can be not very useful in situations where the general scale and variance of the character-level and word-level embeddings differ.

Following the work of [1], this paper seeks to implement a compromise between the two methods of averaging the two different embeddings and concatenating them. In particular, it forms a component-

wise weighted average of the two different embeddings, where the weights themselves are parameters learned by some function of the data itself. This is known as a fine-grained gating mechanism.

3 Related Work

This paper is based almost entirely on the work of [1]. In the paper, the author proposes a hybrid word-character modeling method that it calls Fine-Grained Gating. As some setup, suppose that **w** is a word represented as a one-hot vector and **C** be the sequence of characters of the word represented as a matrix whose rows are one-hot vectors representing each character. Then, given a word embedding matrix **E**, the product **Ew** is the word-level embedding. Additionally, they apply an RNN to **C** and take the hidden state of the final timestep **c** and call that the character-level embedding. The paper then computes a feature vector **v** consisting of "the concatenation of named entity tags, partof-speech tags, binned document frequency vectors, and the word-level representations". It then computes the fine grained gate **g** by applying a single dense neural layer: $\mathbf{g} = \sigma(\mathbf{W}_g\mathbf{v} + \mathbf{b}_g)$ where $\mathbf{W}_g$ and $\mathbf{b}_g$ are model parameters.

Finally, the embedding **h** of the given word is computed as $\mathbf{h} = \mathbf{g} \odot \mathbf{c} + (1 - \mathbf{g}) \odot (\mathbf{Ew})$ where $\odot$ is elementwise multiplication. Inutitively, in each component, **h** is a weighted average of the the character-wise embedding and the word embedding, where the weights are given by the gate **g**, which in turn gets its information from a variety of features related to the word. The motivation for this idea is that for some words, the word-level embeddings are more important, and for some the character-level embeddings are more important. This model, then, is able to learn a parameter **g** which can tell when a word should put more weight on its word-level embedding and when a word should put more weight on its character-level embedding.

4 Approach

The base structure of the question-answering model created in this project is entirely identical to the given Bidaf model, with the one exception of the embedding layer. The embedding layer is changed from a straightforward word-embedding to a variety of different combinations of word and character embedding (with the goal of analyzing comparatively how a fine-tuned grating method works). The word-embedding given to us by the original model shall be named **Ew**, so as to align with notation from the previous section.

Everything in the remainder of this section was implemented and coded by me.

The first step in computing the new embeddings is of course to compute character level embeddings. This was accomplished by first taking the character embeddings for every letter in a word for a given sequence and passing it though a single layer of a convolutional neural network, where the filter only convolves along the axis representing the different characters in a given word so as to capture positional similariy, and then max-pools along that same axis so as to create one vector embedding per word in the sequence. The number of output channels of the convolutional neural network (which is equal to the number of dimensions of the resulting character level embeddings) was chosen to be 300 so as to match the dimensions of the word-level embeddings. This was done so that later arithmetic (namely, the weighted average) can be computed between the word and character level embeddings. The filter size for the convolutional neural network was chosen to be 5. The resulting character level embedding of the word shall be named **c**, so as to align with the notation from the previous section.

From here, a baseline word-level model is computed by just simply using **Ew** as the embedding of the word and feeding it into the RNN. Next, a baseline word-character-concatenation model is computed by concatenating **Ew** and **c** and feeding it into the RNN.

To start computing the fine-grained gating models, the first step is to decide what the feature vector **v** is going to be. In the original paper, this feature vector consisted of "the concatenation of named entity tags, partof-speech tags, binned document frequency vectors, and the word-level representations", but since this project forbids the inclusion of any outside data, the feature vectors must come from the character and word level embeddings themselves. One model attempted in this project was to just use the word-level embedding **Ew** as the feature vector **v**, and another model attempted was to use the

concatenation of the word-level embedding **Ew** and the character-level embedding **c** as the feature vector **v**.

Once this feature vector is chosen, the fine-grained gate **g** was computed by passing **v** through a single dense neural layer with a sigmoid activation function $\mathbf{g} = \sigma(\mathbf{W}_g\mathbf{v} + \mathbf{b}_g)$, and then finally the fine-grained gated embedding **h** was calculated as the **g**-weighted average of **Ew** and **c** via $\mathbf{h} = \mathbf{g} \odot \mathbf{c} + (1 - \mathbf{g}) \odot (\mathbf{Ew})$ where $\odot$ is elementwise multiplication.

From here, we can either feed the resulting embedding **h** into the RNN for question-answering, or we can supercharge the information fed into the RNN by declaring our final embedding vector to be the concatenation of **Ew**, **c**, and **h**, hence giving the RNN the ability to glimpse information from the character and word level models directly as well as glimpse more fine tuned information weighted according to which style of embedding is learned to be more important for a given word.

5 Experiments

5.1 Data

The dataset for this project is the SQuAD 2.0 dataset, as given directly from the project assignment.

5.2 Evaluation method

The two metrics for evaluating the different models are:

1. The EM score, which is the fraction of queries that the model gets exactly correct, with no room for error.
2. The F1 score, which represents the harmonic mean of the precision and the recall for a given query, averaged across all queries.

5.3 Experimental details

For every experiment, a constant learning rate of 0.5 was used, and the model was trained for 30 epochs, which generally consisted in about 3.85 million steps. Training generally took around 15 hours to complete for each model.

The following different model configurations were trained and tested:

1. **Word-Level Baseline**: This model is simly the default implemented Bidaf model, using just word-level embeddings.
2. **Word-Character-Concatenation Baseline**: This model is the same as above, except the embedding for a given word consisted of the concatenation of both the word-level embeddings and the character-level embeddings. It is a baseline in the sense that it is used as a point of comparison for future fine-tuned gated models, but it is not necessarily presupposed to perform worse.
3. **Word-Based Gating**: In this model, the embedding of a word was the fine-grained embedding **h** obtained by using the word-level embedding as the feature vector **v**.
4. **Word-Character-Concatenation-Based Gating**: In this model, the embedding of a word was the fine-grained embedding **h** obtained by using the concatenation of word-level embedding and the character-level embedding of a word as the feature vector **v**.
5. **Word-Character-Concatenation-Based Gating With Post-Concatenation**: For this model, the fine-grained embedding **h** is computed in the same way as in the previous model, but the final embedding used for a word is the concatenation of the word-based embedding **Ew**, the character-based embedding **c**, and the fine-grained embedding **h**.

5.4 Results

1. **Word-Level Baseline**: This model achieved a maximum F1 score of 60.6 and an EM score of 57.1 on the dev set.

2. **Word-Character-Concatenation Baseline**: This model achieved a maximum F1 score of 65.19 and an EM score of 61.80 on the dev set, showing significant improvement from the Word-Level Baseline.
3. **Word-Based Gating**: This model achieved a maximum F1 of 61.33 and a maximum EM of 57.64, which is a slight improvement over the word-level baseline, but falls far short of the word-character-concatenation baseline.
4. **Word-Character-Concatenation-Based Gating**: This model achieved a maximum F1 of 62.96 and a maximum EM of 59.47, showing notable improvement from the word-based gating model, but still falling short of the word-character-concatenation baseline.
5. **Word-Character-Concatenation-Based Gating With Post-Concatenation**: This model achieved a maximum F1 of 63.97, and EM of 60.39, again showing notable improvement from the word-character-concatenation-based gating model, but falling short again of the word-character-concatenation baseline.

The results are summarized in the following table:

Model	Maximum F1	Maximum EM
Word-Level Baseline	60.6	57.1
Word-Based Gating	61.33	57.64
Word-Character-Concatenation-Based Gating	62.96	59.74
Word-Character-Concatenation-Based Gating With Post-Concatenation	63.97	60.39
Word-Character-Concatenation Baseline	65.19	61.80

The best of these models is the word-character-concatenation baseline model, so this model was submitted to the test leaderboard and recieved an F1 score of 63.56 and an EM score of 59.87.

Figure 1 and Figure 2 show how these models compared to each other over time with respect to the F1 and the EM scores on the dev set.

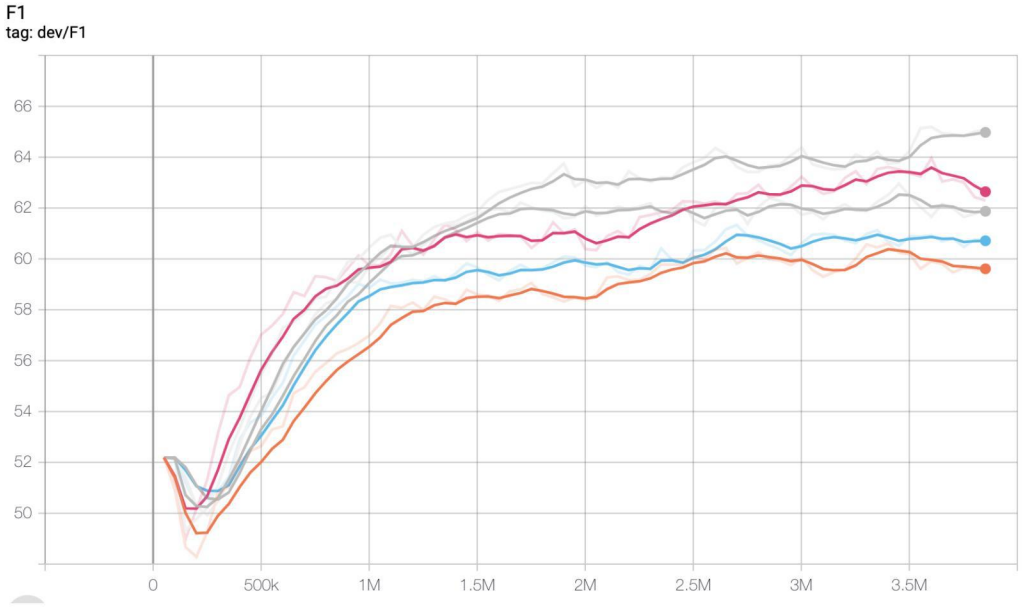

Figure 1: A graph of the F1 score of the five different models (y axis) plotted against the step number (x axis). The upper gray is the word-character-concatenation baseline (2), the pink is the word-character-concatenation-based-gating with post-concatenation (5), the lower gray is the the word-character-concatenation-based gating (4), the blue is the word-based gating (3), and the orange is the word-level baseline (1).

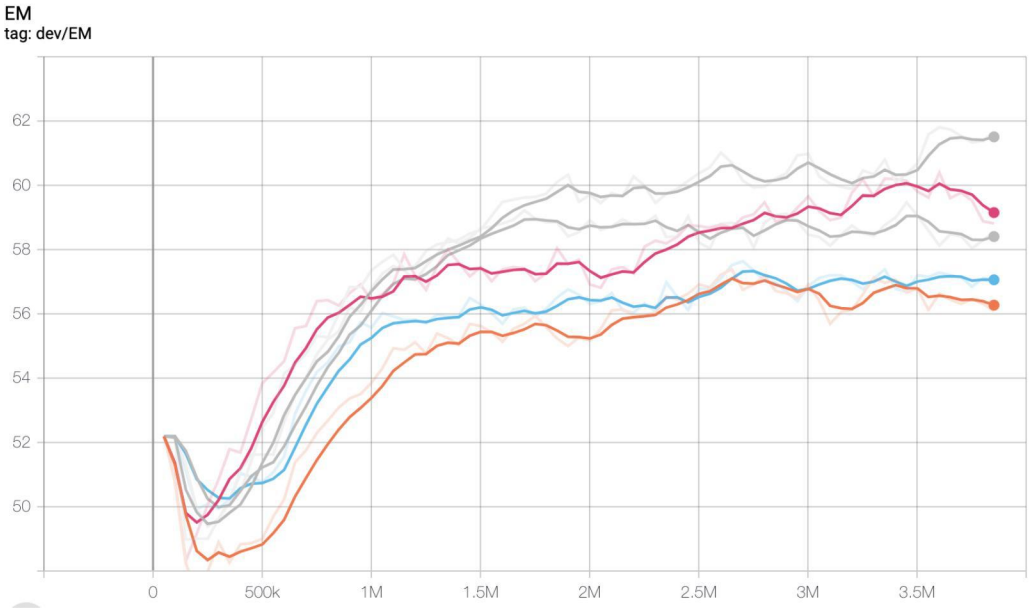

Figure 2: A graph of the EM score of the five different models (y axis) plotted against the step number (x axis). The upper gray is the word-character-concatenation baseline (2), the pink is the word-character-concatenation-based-gating with post-concatenation (5), the lower gray is the the word-character-concatenation-based gating (4), the blue is the word-based gating (3), and the orange is the word-level baseline (1).

It is not surprising that all of the gating models performed better than the word-level baseline model. It is, however, surprising that none of the gating models performed better than simply concatenating the word and character level embeddings. This is not particularly surprisng for models (3) and (4), but it is surprising for model (5) in which the final word embedding was the concatenation of the word-level, character-level, and fine-grained embeddings. This model's RNN has access to all of the information that the concatenation baseline's model has access to, but it additionally gives it access to more information, namely the gated embedding. Thus, theoretically, with access to more information, model (5) should have been able to create a more robust question answering system than model (2). This, however, did not turn out to be the case. A possible explanation for this phenomenon is simply the fact that model (5) has a signfiicantly higher number of parameters than model (2) in its embedding layer (basically, all of the parameters that come with the gating mechanism, together with the fact that the final embedding is of dimension 900 instead of 600 for model (2)). Thus, with more parameters it could take longer to properly learn. One way to verify this would be to let model (5) run for many more epochs and seeing if it ever overtakes the performance of model (2).

6 Analysis

One particularly interesting observation regarding Figures 1 and 2 is that in both instances, the word-concatenation-based gating model (lower gray) initially seems to be performing better than the word-concatenation-based gating with post-concatenation (pink), but the latter overtakes the former around step 2.5 million. This is more generally attributed to the fact that the former model's F1 and EM scores tend to roughly flatline after approximately step 1.75 million, suggesting that the model has essentially become saturated with knowledge and is unable to learn much more. The latter model, however, has more parameters, and though it also appears to flatline rather early, it switches to an increasing trend around step 2 million, suggesting that the model began learning to balance in a more effective way the word-level, character-level, and fine-grated embeddings that make up the final word embeddings.

7 Conclusion

The main finding of this paper is that fine-grained gating, when using feature vectors that just consist of the combinations of the original word-level and character-level embeddings, is not more effective than simply concatenating the word-level and character-level embeddings. Nonetheless, in general fine-grained gating models always performed better than the standard word-level baseline, since they were able to use the gate to gleam information about the character level embeddings that previously wasn't there. Additionally, when the feature vector for the fine-grained gate consisted of both the word-level embedding and the character-level embedding, the resulting fine-grained gate model gives notable improvement over a baseline word-level embedding model.

The discrepency between the results of this paper and those of [1] (in which the fine-grained gating model outperformed the concatenation baseline) can primary be explained by the choice of feature vector. In [1], the feature vector is packed with information not otherwise known to the Bidaf model, for example named entity tags, part-of-speech tags, and binned document frequency vectors. These values can help provide critical insight into whether or not a character-level embedding or a word-level embedding is more important (which is the purpose of the gate itself). In this project, however, outside data was not allowed to be incorporated, so these values could not be used, and instead the gate had to learn its values from the word and character level embeddings themselves.

References

[1] Zhilin Yang, Bhuwan Dhingra, Ye Yuan, Junjie Hu, William W. Cohen, and Ruslan Salakhutdinov. Words or characters? fine-grained gating for reading comprehension, 2017.