Conditioning External and Internal Context Awareness through Attention Overload and Character-Level Embeddings

Stanford CS224N Default Project

Umar Patel
Department of Computer Science
Stanford University
umar2023@cs.stanford.edu

Abstract

Commonly used pre-trained token embeddings work well at encapsulating word meaning and semantic representation and act as great resources for numerous language modeling tasks. These models represent words such that similar words are mapped to similar regions knowns as embedding spaces. However, one major drawback that arises is that the same words can have different meanings within different contexts or positional orientations, and models generally have a difficult time picking up on these nuances. A single word's presence can dramatically affect the meaning of other words, and many pre-trained models have difficulty in tuning to these situations as sequences become longer. Furthermore, the internal structure of words is often lost when solely focusing on words as a whole and their external relationships with other words, which can prove difficult for accurately representing rarer words and phrases.

I aim to address these issues by constructing an NLP mechanism that will add this contextual awareness to word embeddings that would ultimately improve question and answering systems, which greatly depend on accurate word and phrase relationships within and between queries and their corresponding contexts. In this project, I implemented a sinusoidal and learned positional encoding scheme and a pre-model self-attention mechanism to transform the GLoVE input embeddings to the baseline model into more context-aware and positionally attuned inputs. I also developed a character-level embedding layer using CNN to condition this internal word structure. Overall, I found that while some increased level of context-awareness proves to enhance performance in our question and answering task, too many added layers of such context-building implementations can ultimately reverse the benefits and confuse the model.

1 Key Information to include

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

2 Introduction

Given the baseline BiDAF model for Question and Answering, I am to improve the model by implementing additional layers or altering the model architecture as a whole. Specifically, given a question and a context in which the answer to the question can be found, our goal is to improve

the baseline model that retrieves the correct answer from the context. Such a task can prove useful in a variety of disciplines from individuals conducting research in science and humanities, to helping patients and doctors diagnosing or treating illnesses in the medical field. Furthermore, many implementations withing the larger question-answering framework have applications in a variety of other language modeling tasks, such as machine translation, sentiment analysis, and text summarization. Thus, methods I implemented enhancing context awareness can potentially be applied to these other areas and improve their frameworks.

There are a variety of cutting-edge models which perform question and answering to a high degree, many of which I drew inspiration from for this project. The BiDAF model described in Seo et al., 2016 [3], which acts as the baseline for my project, makes use of bi-directional attention flow to encode contextual information between the query and context and an RNN modeling layer to obtain the interaction of the context words based on the query. Ultimately, the model's separation of the attention and modeling layers allows for enhanced learning and between query and context, and the attention mechanism avoids dependence on attended outputs at previous time steps which helps to avoid exacerbations on incorrect dependencies. One of the drawbacks of this approach is the slow rate of training caused by the recurrent networks. The QANet architecture described in Yu et al., 2018 [2], on the other hand, avoids these recurrent structures all together to speed up training and makes use of multiple encoder blocks involving convolution, attention, and feed forward layers to encode the context and query sequences.

Ultimately, I want to capture the novel mechanism of using convolutions and self-attention to enhance the BiDAF baseline. Specifically, I implemented three major components from scratch to the current model. The first is a learned positional encoding layer that adds information about relative word order, position, and distance into the context and the query vectors. Secondly, I implemented a scaled-dot product (SDP) attention layer to further refine contextual awareness among the context and the query. And finally, I implemented a character-level embedding layer using 2D-convolution and max pooling, which I concatenated to the attuned word embeddings prior to passing them along to the rest of the BiDAF model. Positional encoding allows for query and context sequences to encode information on the positions of words and their internal relationships. As I will explain later, both the sinusoidal approach (described in the original Transformer architecture, Vaswani et al., 2017 [1]) and the learned positional encoding scheme (this is the one I eventually used in my final model) allow for words that are closer together to have similar positional encodings which representing closeness and high association and words that are farther apart to have less positional similarity. The scaled dot product attention mechanism I implemented made use of learnable key, query, and value weights to compute attention scores prior to being forwarded to the rest of the model. Lastly, the character-level embedding layer made use of 2D convolution in order to hone in on the internal structure of words, enhancing embedding refinement for less common words or character sequences.

After concatenating the word and character embeddings, we pass on the result to the rest of the BiDAF model, as described in Seo et al., 2016 [3]. In short, it first enters a two-layer highway network to further refine the embeddings. Then, the model passes these embeddings through a bidirectional LSTM to model temporal interactions between words Afterward, it enters the bi-directional attention flow layer which models attention flowing from both the context to the query and from the query to the context. Finally, the model makes use of the query-conditioned context embeddings and performs a two-layer bi-directional LSTM which outputs a matrix passed onto the output to predict the answer.

Overall, I found that by adding implementations and layers that emphasize contextual and positional awareness to the context/query embeddings, as well as incorporating key information on internal word structures through character-level embeddings, I was able to significantly improve upon the baseline BiDAF model. Specifically, my implementation outscored the baseline by over 4 points in both the F1 and EM score categories (these metrics will be explained later). I also found that too much attention can be detrimental and confuse our model, as the SDP attention layer after the positional encoding implementation actually worsened overall performance, suggesting that the pre-encoder attention layer may confuse the model.

3 Related Work

The main inspiration of this project stems from the transformer-based approach described in Vaswani et al., 2017 [1], which uses multiple attention-based mechanisms and layers to replace the effective

but slow and complex nature of recurrent neural networks. Specifically, the transformer model makes use of multi-headed attention, which itself is a series of parallelizable scaled dot product attention schemes that I incorporated in my project. Another significant element of the transformer architecture is the positional encoding scheme it places before the encoder/decoder blocks to make use of the word order and to inject information about the position of a word within a sentence or paragraph. While the paper preferred the sinusoidal frequency-based approach, I will be experimenting with both this version as well as a learned positional encoding layer in my model. The transformer model is a major component in the BERT language model described in Devlin et al., 2019 [5], which makes use of the encoder block of a transformer as well as a pre-training and fine-tuning framework to perform question and answering, along with other common NLP tasks.

The use of convolution neural networks in NLP models was introduced by Kim, 2014 [6], which use computer vision techniques of applying convolving filters to local features. In my case, I will perform such operations on character sequences to learn information on the local and internal features of words which will then be used alongside the attuned word embeddings.

Ultimately, my work will show what would happen if you do not simply replace recurrent neural networks as a whole, but when you incorporate elements of these "replacements" (i.e., positional encodings and attention mechanisms) to current recurrent models like BiDAF. Many works like Kim's use of convolutional neural networks have shown the value in enhancing local and internal contextual awareness, and so enhancing this aspect of the baseline model will show how to implement such conditioning alongside recurrent networks.

4 Approach

The main idea behind positional encoding is that the position of a word ultimately matters, but regular word embeddings do not contain this positional information. Thus, we must inject some weight to the word embeddings (pre-determined or learned) in order to account for these positional relationships. Below is a trivial example of why position in a sentence is important, and also a simplified overview of how to positionally encode our embeddings.

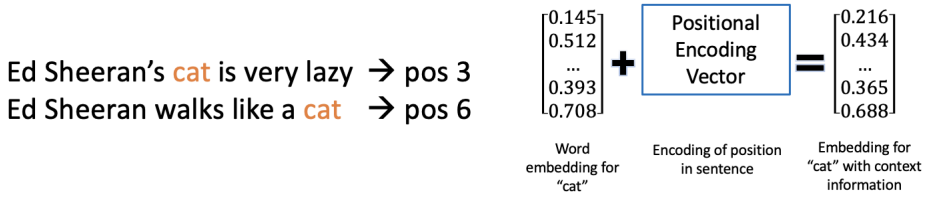

Figure 1: It is clear "cat" should not be treated the same in both these sentences.

I developed and tested two different approaches to positional encoding. While I implemented both from scratch, the first was described in the original Transformer paper, Vaswani et al., 2017 [1], while the second is a learned implementation I developed myself.

Prior to entering these layers, the current input, X, representing the query or context sequence, is of dimension (batch size, sequence length, word embedding size), where batch size is 64, sequence length is the length of the longest sequence in the batch, and the word embedding size is 100.

4.1 Positional Encoding: Sinusoidal (Coded Myself)

For this approach, I used the recommended sinusoidal frequency curves that were stated in Vaswani, et al., 2017,[1] which are shown below.

$$\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{2i/d_{\text{model}}}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{2i/d_{\text{model}}}) \end{aligned} \tag{1}$$

where pos is the current word position in the sequence (i.e., 0 < pos < total sequence length), dmodel is the embedding size (100), and i is the current embedding position. We apply the following equations at each word and embedding position and add the resulting matrix to the current word embeddings, essentially injecting the "positional" information into the current word embeddings. We also applied a dropout layer in order to prevent sequences from overfitting to themselves and to ultimately be more robust.

The key intuition stems from two concepts as shown in the sin and cosine graphs below. First, for different positions in the input, the height of the sine curve will vary with different positions, and so different positions will deviate different amounts along the y-axis, and within a fixed range, providing each position in the sequence a sense of which portion of the input it's dealing with without significant distorting the embedding. Secondly, the issue with the cyclical nature of sinusoidal representations and having later positions in the sequence re-map themselves to values associated with earlier positions is solved through the embedding indices being used to vary frequency of the sinusoidal representations. Positions that are truly close together will retain similar positional embedding values for lower frequencies, whereas positions that are farther apart but that may be mapped to the same sin or cosine value due to the cyclical nature of the curves will differ dramatically as frequency is increased minimally. Therefore, positional embedding encapsulates both these factors to accurately incorporate position information within the embeddings.

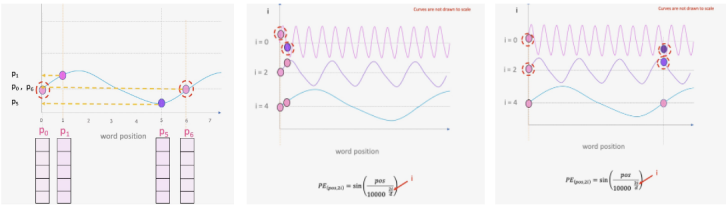

Figure 2: Visual Guide to Transformer Neural Networks - (Episode 1) Position Embeddings.

4.2 Positional Encoding: Learned (Original and Coded Myself)

I also wanted to extend this idea of positional information into a more trainable approach, since even the sinusoidal approach makes some underlying assumption regarding how position is distributed across a sequence. Specifically, I wanted to see if I could learn a positional matrix that could pick up on positional relationships not identified by the sinusoidal approach.

I initialized a trainable matrix to be the same dimensions of our input (i.e., sequence length x embedding size) via a linear layer. I then trained this weight on the input by adding the positional weight to the input batch and applying dropout in the same manner as the sinusoidal approach.

4.3 Self Attention (Coded Myself)

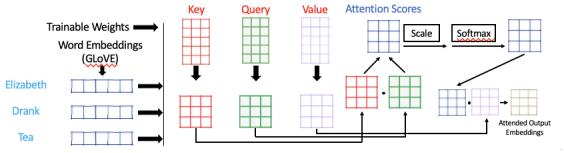

Figure 3: Key, Query, Value Self-Attention

This layer would take in the input X and perform scaled dot product attention described in Vaswani, et al., 2017 [1]. Specifically, this attention mechanism begins with initializing and training three weights, K, Q, and V, corresponding to the key, query, and value, respectively. To obtain the key, query, and value matrices, we multiply these trainable weights with our input embeddings. I also initialized a standard dropout to avoid overfitting and embedding co-adaptations. We then calculate attentions scores by multiplying the query vector by the keys, and then scaling the result by dividing by $\sqrt{d_k}$, the dimensionality of the key vector. We then apply the softmax function to normalize

the scores, implement dropout, and finally take the weighted sum of values using the value weight. Above is a model I created depicting the pathway of the input embeddings through the attention layer.

4.4 Character-Level Embeddings (Coded Myself, some original implementation)

The crux of our attention-based approach lies in the character-level embedding layer. To start, our character embeddings are slightly different than our word embeddings. That is, the character embeddings for each batch are of the shape (batch size, sequence length, maximum word length, character dimension), whereas the word embeddings for each batch are of shape (batch size, sequence length, word dimension size). Ultimately, we would want to concatenate the word embeddings with their respective character embeddings, and so to this we would need to project the character embeddings down to the shape (batch size, sequence length, character dimension size).

To implement this layer, we must first initialize a 2D convolution setting our in-channel to 1 and our out-channel to 100. The number of out channels represents how many different filters we would want to convolve over our character embeddings, and it ultimately represents the size of the character embedding (we will see why). It is typically a value between 100 and 1000, as explained in Kim, et al., 2015 [7]. We also initialize the size of a 2D kernel for convolution, which is of size (character embedding size, w), where character embedding size is 64 and w is the window over which we want to convolve over. We will discuss the different values I used for w in the experiments section, but ultimately, I decided with 7 as being the most ideal. Additional layers within this implementation include ReLU and dropout layers.

I first implemented the dropout upon the input, and then rearranged the dimensions of the input in order to prepare it for the convolution layer. After performing the 2D convolution, I then passed the input through the ReLU layer. Now, the values that are attained through the convolution are then compared in the maxpooling layer, where the highest convolution value would be the feature to represent the character embedding for that particular filter. The convolution does this for all 100 filters, and so we are left with a matrix of dimension (batch size, sequence length, number of filters). And as explained earlier, the number of filters we use in the 2D convolution is exactly our desired character embedding size.

Finally, we concatenate our final character embeddings to the end of the attuned word embeddings, which is then passed on to the highway encoder and the rest of the BiDAF model, where it will predict a start and end sequence within the context for the answer to the given query. Below is an illustration of how our word and character embeddings are passed through this implementation.

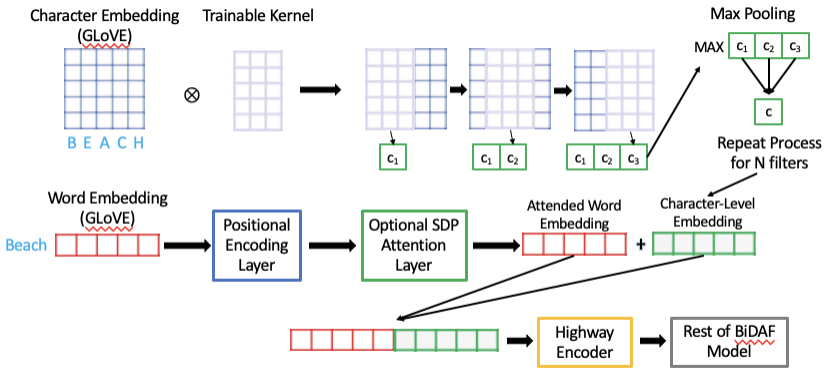

Figure 4: Character Embedding Layer Using 2D Convolution

5 Experiments

5.1 Data

The data I will be using to train and test my model will be the Official SQuAD 2.0 dataset. However, due to the official test set is hidden, my test set during the development phase of my project involved using roughly half of the official dev set as our "test" set. Specifically, the dataset involves a context and a query (there can be multiple queries per context), and for each query, several ground truth answers are given which can directly be found in the context. Below is an example.

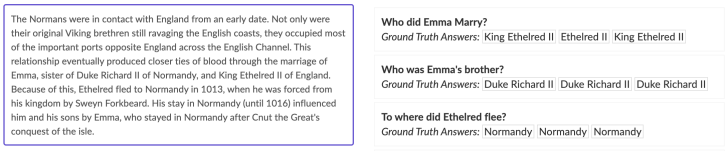

Figure 5: Sample Context and Query from the official SQuAD 2.0 Dataset

Basically, the output of our model is a start and end sequence within the context which corresponds to the answer for the given query. We would then evaluate our model's answers to the ground truth answers (examples shown above) via the methods described in the next section.

5.2 Evaluation method

I will use two metrics to evaluate my model's performance: the Exact Match (EM) score, which is a binary measure of whether the output matches the ground truth answer exactly, and the F1 score, a less strict measure that is the harmonic mean of precision and recall.

Essentially, for the EM score, the model's answer must match the ground truth answer exactly in order for the score for that example to be 1. If it is not an exact match, then the score for that example is 0.

For the F1 score, we first find the precision of the model's output answer (whether the answer is a subset of the ground truth answer) and the recall of the model's output answer (how many words from the ground truth answer did the model output divided by the total number of words in the ground truth answer) and calculate the harmonic mean of the two using the equation (2 • precision • recall) / (precision + recall).

5.3 Experimental details

I ran a total of 5 main experiments with different variations of my implementations in order to see what combination produces the best results. However, after finding the best structural implementation, I also experimented with various kernel sizes within my 2D convolution in the character embedding layer to find the optimal window size.

Unless otherwise mentioned, for all experiments, I trained until completion (i.e., the full training set, 30 epochs). Training was conducted using the Google Colab environment with the runtime set to GPU and High-RAM configurations.

The first experiment was simply training on the baseline BiDAF model without any of my implementations. Training took approximately 3.5 hours to train.

I then ran two separate tests to determine which positional encoding scheme produced the best results: the sinusoidal implementation or the learned implementation. Again, both took approximately 3.5 hours to train.

As I will explain in the next section, the learned positional encoding scheme fared much better than the sinusoidal approach, and so for the remaining experiments throughout this project I used the learned positional encoding layer rather than the sinusoidal approach.

After testing positional encoding, I experimented with a learned positional encoding layer coupled with the self-attention layer. Training time on this implementation also took roughly 3.5 hours.

For my final experiments, I tested the learned positional encoding scheme with the character level embedding via CNN implementation. However, for the character embedding layer, I ran several different experiments with varying kernel sizes (specifically, the second dimension for the kernel input for the 2D convolution). For my first experiment, my window size was 3. For my second experiment, my window size was 7. And for my third and final experiment, my window size was 9, although I did not run this third experiment until completion in training.

5.4 Results

I am on the IID SQuAD track

The results of the experiments run on the dev set are as follows:

	F1	EM
Baseline BiDAF model	60.55	57.10
Sinusoidal Positional Encoding Implementation Scores:	58.20	54.80
Learned Positional Encoding Scores:	60.91	57.92
Learned Positional Encoding + SDP Attention Scores:	52.19	52.19
Learned Positional Encoding + Best Character Level Embeddings:	**64.57**	**61.49**

Figure 6: Main Experiments, as described in 5.3. Notice that the final row is the best performance taken from the experiments in Figure 8.

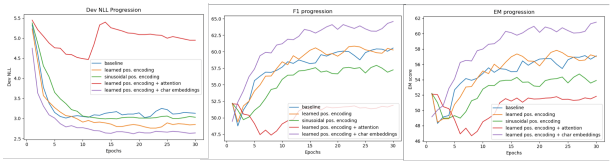

Figure 7: Visualization of training progressions for all 5 main tests

	F1	EM
Learned Positional Encoding + Character Level Embeddings with kernel size (64, 3)	63.42	60.31
Learned Positional Encoding + Character Level Embeddings with kernel size (64, 7)	64.57	61.49
Learned Positional Encoding + Character Level Embeddings with kernel size (64, 9)	N/A (Did not train to completion, clearly not performing as well as other kernel sizes)	N/A (Did not train to completion, clearly not performing as well as other kernel sizes)

Figure 8: Experiments on kernel size (window size) with the positional encoding and character level embedding implementation.

	F1	EM
Final Implementation on Test Set	62.532	59.560

Figure 9: Final test set performance.

It is clear that my final implementation of a learned positional encoding layer and a character-level encoding improved upon the baseline model, which was expected. However, I am quite surprised by some of the individual implementation performances. Specifically, I was initially surprised when the sinusoidal positional encoding implementation fared slightly worse than the baseline model, as it actually decreased performance. Perhaps the sinusoidal approach was too ideal for the complex Question and Answering system, as its initial use was for machine translation tasks in the transformer model. At the same time, I was not surprised that the learned positional encoding scheme fared better than the sinusoidal approach since it accounts for more nuances and positional patterns that occur in question answering and especially in context-query relationships which the model ultimately depends

on and can learn from. As expected, the positional encoding implementation improved upon the baseline.

I was not expecting that the scaled dot product attention layer would fare so poorly when included in the model, as it F1 and EM scores cause drops of 8 and 5 points from the baseline model, respectively. Furthermore, the training performance is somewhat of an outlier compared to all of the other implementation tests, as its F1 and EM scores decrease for far longer at the beginning of training before going up slightly. On the other hand, all the other implementations followed a trajectory of decreasing for several epochs but then rapidly increasing for the next 10 epochs, which is then followed by a gradual plateau of the scores.

Finally, the results from the kernel size experiments makes sense, as you would ideally want to perform convolution over enough nearby characters to get an accurate representation of the feature described by the current filter, while also not extending to characters that are farther away which may add undesired noise to the final embeddings.

6 Analysis

Overall, we find that the addition of positional encoding allows key positional information to be injected into our word embeddings, and the addition of character level embeddings on top of this layer adds important internal conditioning and awareness within words that can enhance language modeling tasks such as question and answering.

However, "attention overload" as the title of my paper suggests, is not as simple as including as many attention and context-based layers into a model. As we see from my experiments, the combination of a positional encoder, a self-attention layer for our word embeddings, and a bi-directional attention scheme can prove detrimental to our overall performance. It appears that too many attention-based layers can create noise and confuse our model to know exactly what to attend to. And beyond just the F1 and EM scores, the progressions seen in figure 7 show that this added attention layer causes the model to train abnormally when compared to the other implementations.

We also see that just as in computer vision, kernel size does can have a significant effect on the output. Too small of a kernel means you aren't adjusting for local features that are significant to the current feature of interest, and too large of a kernel can cause unnecessary noise and produce less useful information to supplement word embeddings. Ultimately, the character level embeddings are a key aspect of this internal conditioning, which ultimately enhance our model's awareness for the character-based structures within sequences.

7 Conclusion

From this project, we can conclude that implementing attention schemes such as learned positional encoding and adding character level embeddings prior to the highway encoder of the BiDAF model can significantly enhance the architecture's overall performance. Of course, I found that too much attention-based layers at the forefront of the model is not ideal, but it does not take away from the overall attention-filled nature of my new BiDAF model, one whose emphasis on internal and external conditioning can be seen through my own additions as well as the baseline model's bi-directional attention layer itself.

One of the main avenues I would like to explore is the diminishing returns that my self-attention layer produces, and the unorthodox training patterns that are associated with it. Although it makes sense why the self-attention layer could produce counterintuitive scores, the progression of its performance during training involves deeper research that would be essential to understanding why this specific attention approach does not work and how to change it so that it might.

Of course, the primary limitation of my work is the fact that I am new to the field. This was one of the most challenging projects I have ever confronted, but also one of the most rewarding. With more experience, I may have been able to find the time to implement structures and architectures that could enhance the model further, but unfortunately a lot of my time went into understanding some of the basic architectures and how the codebase worked. Despite this however, my implementation has found ways to apply additional attention-based mechanisms to recurrent models instead of simply replacing them.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv*: 1706.03762

[2] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. 2018. QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension. *arXiv preprint arXiv*: 1804.09541

[3] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv*: 1611.01603

[4] Caiming Xiong, Victor Zhong, and Richard Socher. 2016. Dynamic coattention networks for question answering. *arXiv preprint arXiv*: 1611.01604

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv*: 1810.04805

[6] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. *arXiv preprint arXiv*: 1408.5882

[7] Yoon Kim, Yacine Jernite, David Sontag, Alexander M. Rush. 2015. Character-Aware Neural Language Models

[8] CS224N Teaching Staff. CS 224N Default Final Project: Building a QA system (IID SQuAD track). 2022

[9] "Visual Guide to Transformer Neural Networks - (Episode 1) Position Embeddings." *YouTube*, YouTube, 8 Dec. 2020, https://www.youtube.com/watch?v=dichIcUZfOw. Accessed 26 Feb. 2022.