Leveraging User Embeddings and Text to Improve CTR Predictions With Deep Recommender Systems

Carlos Miguel Patiño*
carlos@factored.ai
Factored

Camilo Velásquez*
camilo.velasquez.a@factored.ai
Factored

Juan Manuel Muñoz*
juan.manuel.m@factored.ai
Factored

Juan Manuel Gutiérrez*
juan.manuel.g@factored.ai
Factored

David Ricardo Valencia*
david.ricardo.v@factored.ai
Factored

Cristian Bartolome Aramburu
cristian@factored.ai
Factored

ABSTRACT

Predicting user engagement, often framed as a CTR prediction problem, is important to maximize user satisfaction in social networks. The 2020 Recsys Challenge was sponsored by Twitter and set the goal of predicting four types of user engagement using a dataset with 160 million tweets. Our approach extracted information from the tweet's text tokens and built optimized user embeddings. We designed our model based on ideas from recommender systems and deep learning that had been successful in CTR prediction tasks. We show that our modifications to existing state-of-the-art architectures and feature engineering improved the model's ability to predict user engagement. Factored's team was called Los Trinadores and had the 6th best submission of the challenge with an overall score of 22. The code for our solution is available at https://github.com/factoredai/recsys20-challenge/.

CCS CONCEPTS

• Computer systems organization → Neural networks; **• Computing methodologies → Information extraction**.

KEYWORDS

Recommender Systems, Deep Learning, Attention, Factorization Machine, Embedding

ACM Reference Format:
Carlos Miguel Patiño, Camilo Velásquez, Juan Manuel Muñoz, Juan Manuel Gutiérrez, David Ricardo Valencia, and Cristian Bartolome Aramburu. 2020. Leveraging User Embeddings and Text to Improve CTR Predictions With Deep Recommender Systems. In *Proceedings of the Recommender Systems Challenge 2020 (RecSysChallenge '20), September 26, 2020, Virtual Event, Brazil.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3415959.3415995

1 INTRODUCTION

Systems that predict whether users are going to engage with content online are critical to optimize online advertising and maximize user satisfaction on social networks. Predicting user engagement is framed under click-through rate (CTR) prediction and usually involves overcoming the challenges of high-dimensional and sparse data, implicit feedback, and predictions that rely on high-order interactions between features [10, 13, 14]. Recent models used for CTR predictions [4, 6, 9] typically have two components: one to model low-order interactions with, for instance, generalized linear models or factorization machines (FM) and another one to detect high-order interactions with deep neural networks. Combining these two types of components has proven useful when building recommender systems for CTR prediction problems.

In this work, we describe Factored's approach to building a CTR prediction system for the RecSys Challenge 2020 [1], where the objective was to predict user engagement on Twitter, the sponsor of the challenge. We developed a model that predicts user engagement with components that calculate high and low order interactions using attention [15], deep neural networks, and a custom FM [12]. We designed algorithms to extract relevant topic information from tweets and learn quality user embeddings while optimizing computational resources. Our team for the challenge was called Los Trinadores, and we had the 6th best submission of the challenge with an overall score of 22. The source code for our solution is available at https://github.com/factoredai/recsys20-challenge/.

2 CHALLENGE DESCRIPTION

The objective of the 2020 RecSys Challenge was to predict whether a user will engage with a tweet with four possible types of engagement: reply, retweet, retweet with comment, and like. A user could engage in multiple ways with a tweet, so it was possible to find, for example, a user that liked and retweeted a tweet.

The dataset contained around 160 million tweets with possible engagements sampled over one week and 40 million tweets for the submission sets sampled the following week. The organizers made sure to only include information publicly available on Twitter by creating pseudo-negative features, scrubbing deleted content, and providing tokenized instead of raw text [3].

The metrics used to evaluate the models' performance were Relative Cross Entropy (RCE) and Area Under the Precision-Recall Curve (PR-AUC). RCE is calculated using Equation 1 and measures the improvement of the binary cross entropy from the model (BCE_{pred}) compared to the binary cross entropy of a naive prediction (BCE_{naive})—e.g., the average observed engagement in the training set.

*Authors contributed equally to this work.

Table 1: Numerical, categorical, and boolean features grouped by feature encoder.

Encoder	Feature
Engager	Logarithm of follower difference with author
	Logarithm of number of followers
	Logarithm of number of accounts followed
	Is the account verified?
	Quantile of account creation date
	Follows author?
	5 topics with most appearances for each user
	Number of appearances of 5 topics
Author	Logarithm of follower difference with engager
	Logarithm of number of followers
	Logarithm of number of accounts followed
	Is the account verified?
	Follows engager?
	Quantile of account creation date
Tweet	Type of tweet (e.g., retweet or reply)
	Day of the week the tweet was written
	Hour the tweet was written
	Language
	Topic
	Number of videos
	Number of GIFs
	Number of hashtags
	Number of domains
	Number of photos
	Number of links
Text	BERT encodings of text tokens

$$RCE = \frac{BCE_{naive} - BCE_{pred}}{BCE_{naive}} \tag{1}$$

3 DATA PROCESSING

The size of the dataset required a dedicated infrastructure to process the data and create new features. We built our processing pipeline using Spark [17] to process the dataset on an Elastic MapReduce cluster of multiple c5.2xlarge instances on Amazon Web Services (AWS). We additionally used a p2.8xlarge instance to create features from the text tokens using PyTorch [11]. Table 1 details the numerical, boolean, and categorical features created using this data processing infrastructure.

3.1 Topic Extraction

We considered that the tweet's text was one of the primary sources of information in the dataset. Information about the text was provided using an ordered list that corresponded to the text's BERT [5] tokenization. Our first approach was to feed the tokenized text to HuggingFace's BERT base multilingual cased model [16] to obtain trainable encodings of the text during training and inference. The size of the BERT model made this approach computationally expensive in training and inference time, so we decided to create features that encoded text information.

Our solution to balance the tradeoff between computational cost and information quality was to use BERT to create text encodings to use them as non-trainable input features in the model. We also extracted the main topic from each tweet using the BERT encodings and determined the topics that appeared the most for a specific user. We detail the process to determine the topic of a tweet in Algorithm 1, where the number of clusters in step 3 was determined using the elbow method. The main advantage of using Algorithm 1 and BERT encodings as non-trainable features was that we had information about the text while feeding only once the text tokens to the BERT model. This approach was computationally more efficient than updating BERT's parameters for each batch at training time.

Each tweet had an assigned topic after running Algorithm 1, but we also required a connection between the tweet topic and a user. On average, a user was exposed to 5 topics in the training data, so we chose the 5 topics—from the 150 topics list—that were seen the most by a particular user. The number of times a user was exposed to each of the five topics encoded information about the most relevant topics for each user.

Algorithm 1 Extract topic from BERT tokens.

```
1: Encode tokens using BERT model.
2: Reduce dimensionality of encodings keeping 95% of the variance using PCA.
3: Generate 150 clusters from the reduced encodings using k-means.
4: Assign the topic to each tweet from the corresponding cluster.
```

3.2 User Buckets

Building quality embeddings for the users in the dataset was important because each user may have a particular engagement behavior. We decided not to create an embedding for every user in the dataset because the model would not be able to learn quality embeddings for users that appeared only a few times in the dataset.

We devised a process to learn relevant embeddings for users through buckets of users. The process, detailed in Algorithm 2, assigns a bucket to a single user if the user appears more than 71 times in the dataset. Users with fewer appearances are clustered into groups of similar users and split into buckets with multiple users from each cluster. The user clusters were built using variables related to users, such as the number of followers and whether the user's account is verified. We tuned the number of buckets created from a cluster and found that 400 was the optimal value. The threshold of 71 appearances was defined because more than 200k users (around 70% of the dataset) have more than 71 appearances.

Each bucket was assigned a trainable embedding, so more than 200k users had a dedicated embedding. In contrast, the remaining users had an embedding learned for all the users in the bucket. The buckets allowed us to, on the one hand, reduce the number of embeddings that the model had to learn during training. On the other hand, we had embeddings with better quality for users that appeared less than 71 times because we could use the data from similar users to learn a single embedding.

Without using the user buckets, we would have had to define a single embedding for each of the 29M users in the dataset. We reduced the user embeddings to around 220k and allowed the model to learn quality embeddings using user buckets instead of single-user embeddings.

Algorithm 2 Assign users to user buckets

```
1: Create 60 clusters with users that appear less than 71 times in
   the dataset using k-means clustering.
2: if User appearances > 71 then
3:     Assign entire bucket to user
4: else
5:     Assign user to one of the 60 clusters
6: end if
7: Group clusters with less than 450 users into single cluster
8: for cluster in clusters do
9:     Divide into 400 buckets using feature hashing.
10: end for
```

3.3 Data Split

We took special care to design our data splits with two objectives: optimizing time spent validating the model and generalizing the validation score to the submission score. Figure 1 shows a diagram of how we performed the data split. We optimized the time spent on validation by creating a validation set size of around 500k samples. This set size was small enough to run validation multiple times per epoch. Since the validation size was less than 1% of the training dataset and 20 times smaller than the submission set, we had to design our out-of-time and out-of-space criteria to guarantee that the validation dataset measured how the model would perform on a submission set.

We avoided data leakage through time dependence using a time split where the validation samples were taken from the last 16 hours from the training set. This time split was necessary because, since all the interactions in the submission set occurred in the week after the training set, the model had to generalize well to out-of-time samples.

Our out-of-space criteria were tailored to the characteristics of this dataset. The first step was to replicate the ratio of cold-start vs. warm-start users in the submission set. We determined that 27% of users in the submission set were not present in the training set, so we replicated this cold-start distribution in the validation and training sets. For the validation set, we took 27% of the users selected for validation and deleted any previous appearances they had in the training set. Deleting the data allowed us to evaluate the model with the right ratio of cold-start users. This type of user is User 2 in Figure 1.

We also wanted the model to learn how to handle cold-start users during training, so we imputed features related to previous behavior—such as the number of appearances of each topic—for 7% of the users in the training set as represented by User 3 in Figure 1. We defined the imputation of the historical features to resemble a cold-start user, so, for example, we replaced the topic clusters with random cluster values and set the number of appearances for each topic to zero.

Our focus was to perform well on the submission set, so we did not include in the validation set users that appeared in the submission set, as shown by User 4 in Figure 1. The purpose of this restriction was to allow the model to learn as much as possible about the users in the submission set during training.

4 MODEL

We approached the design of our architecture based on existing models—DeepFM [6], xDeepFM [9], and AutoInt [14]—that performed well on CTR predictions. Figure 2 summarizes our architecture by showing how the three interaction components receive encoded features and output vectors used by additional layers to calculate the probability for each type of engagement.

4.1 Encoders

The model's encoders consist of 3 dense layers with ReLU activations and dropout rates of 0.3, based on the number of layers and dropout rates of the deep module of the DeepFM architecture [6]. Each encoder receives the numerical and boolean features and embeddings that are relevant to the encoder. For example, the author encoder receives numerical and boolean features and embeddings for categorical variables and user bucket that correspond to the author of the tweet. We included a batch normalization layer [8] between each dense layer in the encoders to make our model robust to the data shifts caused by the rapid changes in the conversation on Twitter [3]. The input features for each encoder are detailed in Table 1.

Figure 2 shows that we defined a skip connection that connects the author and engager embeddings directly to the interaction components. We used this skip connection to ensure that the learning process of the embeddings was not affected negatively by the additional layers in the author and engager encoders. This part of the architecture was inspired by the success of residual connections in deeper networks that improve the model's performance [7, 13].

4.2 Interaction Components

We included a deep learning component in our model based on the success of DeepFM and xDeepFM in calculating high-order interactions with this type of approach. The deep learning component is the DL module in Figure 2 and includes three dense layers with batch normalization, ReLU activation functions, and tuned dropout rates of 0.25. We based the depth of the DL module on previous results [6, 9] that achieved optimal performance with three dense layers.

Our model includes an attention mechanism based on the success of the AutoInt models on recommendation tasks for CTR. This attention mechanism is the Transformer component in Figure 2 and includes two transformer encoder layers [15], each with four attention heads.

One difference between our architecture and CTR models is that CTR models usually do not include additional layers between the interaction modules and the output probability. These additional layers motivated using a component with a FM that had a vector and not a scalar as an output. The first choice for a FM-based component with a vector output was the Compressed Interaction Network (CIN) [9] of the xDeepFM architecture. However, the known shortcoming of the CIN's time complexity made it more efficient to rely on the

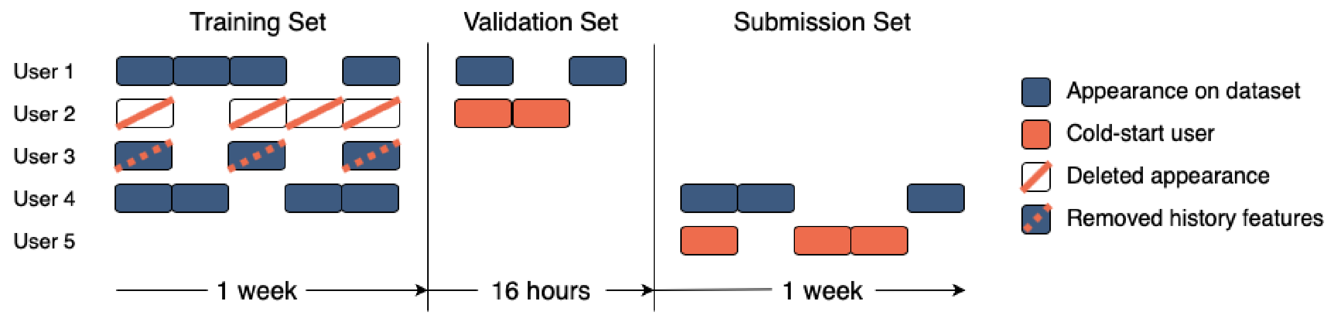

Figure 1: Diagram showing our out-of-space and out-of-time splits: User 1 simulates a warm-start user for validation; User 2 simulates a cold-start user for validation; User 3 simulates a cold-start user for training; User 4 shows how we did not include in the validation users that appeared in the submission set; User 5 shows cold-start users that only appeared in the submission set.

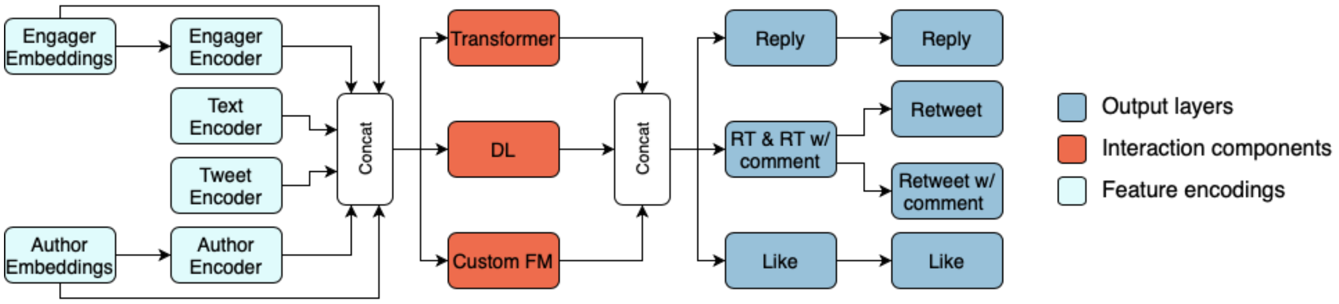

Figure 2: Summarized architecture of our model. All features are grouped through dense layers into encodings of the relevant entities of the problem and are passed to the three interaction components. The output from the three modules is passed to the final layers to output a probability for each of the engagements.

other interaction modules for high-order interactions and develop a modified FM with the dot product of the classical FM [12].

The modified FM is similar to the classical FM, but we do not sum over the dot products between the encodings. Instead, the output of the modified FM is a vector built from each of the dot products as shown in Equation 2, where m is the total number of encodings and $y_{FM}^{(i,j)}$ is the dot product between the latent vectors of encodings i and j. Having a vector output from the FM component allowed us to keep information about the interactions in vector form that was useful for dense networks in the output layers.

$$y_{FM} = \left[y_{FM}^{(1,2)}, y_{FM}^{(1,3)}, \ldots, y_{FM}^{(m,m-1)} \right] \tag{2}$$

4.3 Output Layers

Our data exploration showed that the correlation between *retweet* and *retweet with comment* engagements was higher than all the correlations between engagement types. We defined three modules before the model's output layer to leverage the correlation between *retweet* and *retweet with comment* such that the *RT & RT w/ comment* module shares the weights of the dense layers before calculating the output probability for each engagement.

5 EXPERIMENTS

We built our model using TensorFlow [2] and ran training and inference on AWS using a c5d.4xlarge EC2 instance—an instance with 16 vCPU cores and without a GPU. We chose multiple CPUs over a GPU because the bottleneck was loading the data to feed the model, not the forward or backpropagation steps. Using the c5d.4xlarge allowed us to solve the data loading bottleneck and optimize the cost of training the model.

Table 2 summarizes the impact of the main components in the architecture by measuring the percentage change in binary cross entropy (BCE) when we removed a component of the model. We used BCE to compare performance because, as shown in Equation 1, a minimum in BCE is a maximum in RCE for any naive prediction used as a reference. The results show that it was useful to use batch normalization to make the model robust to data shifts and to keep the output of the FM in vector form. In contrast, our experiments show that the Transformer module was not relevant to improving the model's performance.

We also ran an experiment where we trained the model without the topic features described in Section 3.1, and the best BCE we obtained was 0.8694: a percentage change of 2.07% compared to the model trained with the topic features. The BCE we obtained without topics shows that, compared to the results in Table 2, using

Table 2: Impact of components in the architecture. We removed a single component of the architecture and compared the BCE of the model without the component with the BCE of the model with all the components (Full Model).

Model	BCE	ΔBCE from Full Model
Full Model	0.8523	–
Without Batch Normalization	0.8763	2.88%
Without DL module	0.8588	0.82%
Without custom FM	0.8565	0.55%
Without weight sharing	0.8547	0.34%
Without transformer	0.8512	-0.07%

the information in the text tokens was one of the most important contributions to improve the model's performance.

6 CONCLUSIONS

In this article, we presented Factored's approach to building a CTR prediction system for the 2020 RecSys Challenge. We extracted topics from text tokens and defined user buckets to optimize training time while encoding relevant information. We designed our architecture based on ideas of deep learning and recommender systems that had been successful in CTR prediction tasks. We showed that our feature engineering and additions to state-of-the-art architectures—especially topic features and making the model robust to data shifts—improved the model's performance and led us to have the 6th best submission of the challenge.

ACKNOWLEDGMENTS

We want to thank Factored's management team for their continuous support throughout the challenge. Juan Jaramillo proposed and arranged our participation in the challenge, contributed to the data processing pipeline, and coordinated progress among the team members. Tomás Olarte helped us setup AWS infrastructure to speed up the submission to the challenge's website and data processing.

REFERENCES

[1] [n.d.]. RecSys Challenge 2020. http://www.recsyschallenge.com/2020/

[2] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283. https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[3] Luca Belli, Sofia Ira Ktena, Alykhan Tejani, Alexandre Lung-Yut-Fon, Frank Portman, Xiao Zhu, Yuanpu Xie, Akshay Gupta, Michael Bronstein, Amra Delić, Gabriele Sottocornola, Walter Anelli, Nazareno Andrade, Jessie Smith, and Wenzhe Shi. 2020. Privacy-Preserving Recommender Systems Challenge on Twitter's Home Timeline. *arXiv:2004.13715 [cs, stat]* (April 2020). http://arxiv.org/abs/2004.13715 arXiv: 2004.13715.

[4] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. *CoRR* abs/1606.07792 (2016). arXiv:1606.07792 http://arxiv.org/abs/1606.07792

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]* (May 2019). http://arxiv.org/abs/1810.04805 arXiv: 1810.04805.

[6] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. *arXiv:1703.04247 [cs]* (March 2017). http://arxiv.org/abs/1703.04247 arXiv: 1703.04247.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv:1512.03385 http://arxiv.org/abs/1512.03385

[8] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]* (March 2015). http://arxiv.org/abs/1502.03167 arXiv: 1502.03167.

[9] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (July 2018), 1754–1763. https://doi.org/10.1145/3219819.3220023 arXiv: 1803.05170.

[10] H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. 2013. Ad Click Prediction: a View from the Trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*.

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*. 8026–8037.

[12] S. Rendle. 2010. Factorization Machines. In *2010 IEEE International Conference on Data Mining*. 995–1000.

[13] Ying Shan, T. Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. 2016. Deep Crossing: Web-Scale Modeling without Manually Crafted Combinatorial Features. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 255–262. https://doi.org/10.1145/2939672.2939704

[14] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. 2019. AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (Nov. 2019), 1161–1170. https://doi.org/10.1145/3357384.3357925 arXiv: 1810.11921.

[15] 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:1706.03762 [cs]* (Dec. 2017). http://arxiv.org/abs/1706.03762 arXiv: 1706.03762.

[16] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *ArXiv* abs/1910.03771 (2019).

[17] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. 2016. Apache Spark: A Unified Engine for Big Data Processing. *Commun. ACM* 59, 11 (Oct. 2016), 56–65. https://doi.org/10.1145/2934664