Interspeech 2018
2-6 September 2018, Hyderabad

i-Vectors in Language Modeling: An Efficient Way of Domain Adaptation for Feed-Forward Models

Karel Beneš[1], *Santosh Kesiraju*[1,2] *and Lukáš Burget*[1]

[1]Brno University of Technology, Speech@FIT and IT4I Center of Excellence, Czechia
[2]IIIT - Hyderabad, India

{ibenes,kesiraju,burget}@fit.vutbr.cz

Abstract

We show an effective way of adding context information to shallow neural language models. We propose to use Subspace Multinomial Model (SMM) for context modeling and we add the extracted i-vectors in a computationally efficient way. By adding this information, we shrink the gap between shallow feed-forward network and an LSTM from 65 to 31 points of perplexity on the Wikitext-2 corpus (in the case of neural 5-gram model). Furthermore, we show that SMM i-vectors are suitable for domain adaptation and a very small amount of adaptation data (e.g. endmost 5 % of a Wikipedia article) brings a substantial improvement. Our proposed changes are compatible with most optimization techniques used for shallow feed-forward LMs.

Index Terms: language modeling, feed-forward models, subspace multinomial model, domain adaptation

1. Introduction

Statistical language modeling is dominated by recurrent neural networks since around 2010 [1]. RNN LMs achieve the best test perplexities on most benchmarks and research community is eager to come up with more complex architectures, e.g. deep LSTMs [2, 3, 4] and Recurrent Highway Networks [5]. However, this comes at a cost of increased computational complexity: Aside from ever-increasing size of the models, the recurrence itself effectively prohibits parallelization over time.

On the other hand, feed-forward neural models (FN-LM) still outperform traditional n-grams counting models and are trivial to parallelize over the length of the sentence. Recently, up to 100x speed-ups were reported for the standard simple shallow architecture, achieved by combination of several tricks [6]. This brings FN-LM on par with n-gram look-up in terms of speed, making it a convenient choice for practical applications.

Yet, the difference in performance between an FN-LM and RNN LMs is vast, e.g. on Penn Treebank, single recurrent model achieves down to 50 PPL [4], while neural 5-grams have been reported to obtain 140 PPL [7]. This difference obviously comes from the ability of RNN to extract useful information from longer context.

There have been some efforts in enhancing RNN LM input by contextual vectors in previous works. Some of these approaches are oriented at collecting word-level context features [8, 9]. An influential paper from Chen et al. [10] provides an overview of incorporating "show-vectors"[1] derived from different topic models into RNN LMs. However, the discussed topic models consider the topic to be a categorical variable, which is unsuitable for further neural processing.

To address these issues, we propose to enhance FN-LMs by a document summary i-vector estimated from a variant of Subspace Multinomial Model (SMM) [11]. This way, we allow the model to exploit a long context. Vectors obtained from SMM contain topic-specific information, but represent it as a Gaussian-distributed deviation from certain mean statistics.

In order to give a fair picture of the performance of our proposed model, we not only show the improvement from a baseline shallow network, but we also display the performance gap to an appropriately big LSTM.

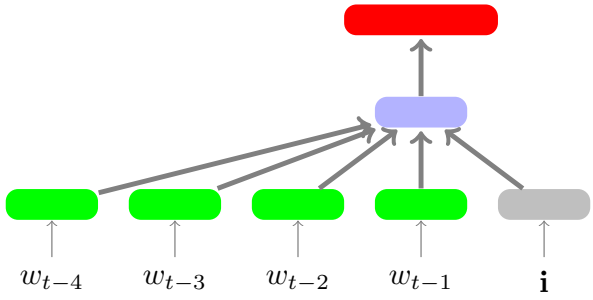

Figure 1: *Neural 5-gram model enhanced by an SMM document i-vector (iFN-LM). Green nodes represent replacing word index by word embedding (trained together with rest of the model, no external word2vec model is used). Gray node provides the i-vector to the model. A separate weight matrix is associated with every thick line in the schema.*

2. Subspace Multinomial Model

Let there be D documents consisting of words from a vocabulary V. In the following definitions, d indexes a specific document and j indexes a specific word w_j from V. Then, $\boldsymbol{c}_d$ is a $|V|$-dimensional vector of counts of individual words in a given document d.

Subspace Multinomial Model (SMM) [12], originally proposed for modeling discrete prosodic features, can be used as a probabilistic model of word densities in such documents. An SMM treats words in the document as independent events, effectively modeling unigram probabilities:

This work was supported by the U.S. DARPA LORELEI contract No. HR0011-15-C-0115. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. The work was also supported by Technology Agency of the Czech Republic project No. TJ01000208 "NOSICI" and by Czech Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project "IT4Innovations excellence in science - LQ1602".

[1]Work was done in context of BBC shows.

10.21437/Interspeech.2018-1070

(a) *Oracle i-vector* (b) *Partial i-vector* (c) *Domain-adaptation i-vector*

Figure 2: *Different schemes of i-vector extraction used in this work. Blue color represents part of the document from which the i-vector is extracted. Red arrow shows what part of the document will an LM process with this specific i-vector. Oracle i-vectors (a) are used for all language model training in this work. Partial i-vectors (b) are used in Section 6 for comparison with other language models. Domain-adaptation i-vectors (c) are used in Section 7 to simulate a situation, where additional topic-related data is available. Note that (a) and (c) only compute one i-vector per document. Partial i-vectors are computed after every block of words (typically 10–30).*

$$P_d(w_j) = \text{softmax}(\boldsymbol{\eta}_d)_j \quad (1)$$

Here, $\boldsymbol{\eta}_d$ are (unnormalized) unigram log-probabilities. The key idea of SMM is to model them in a low-dimensional subspace:

$$\boldsymbol{\eta}_d = \boldsymbol{m} + \boldsymbol{T}\boldsymbol{i}_d \quad (2)$$

Here, $\boldsymbol{m} \in \mathcal{R}^{|V|}$ is a global mean vector, typically close to the global unigram log-probabilities. Matrix $\boldsymbol{T} \in \mathcal{R}^{|V| \times K}$ defines a K-dimensional subspace, where the variablity will be modeled. Document specific vector $\boldsymbol{i}_d \in \mathcal{R}^K$ — from now on called document *i-vector*, thanks to an analogy with a concept used in speaker recognition — captures the deviation of the document from the global mean, in the subspace spanned by columns of $\boldsymbol{T}$.

An SMM is trained to maximize the log-probability of training documents:

$$\sum_{d=1}^{D} \log P(\boldsymbol{c}_d | \boldsymbol{T}, \boldsymbol{m}, \boldsymbol{i}_d) = \sum_{d=1}^{D} \sum_{j=1}^{|V|} c_{dj} \log P(w_{dj}) \quad (3)$$

During inference, only a new i-vector is found, that explains the document vector $\hat{\boldsymbol{c}}$ the best; $\boldsymbol{T}$ and $\boldsymbol{m}$ are left untouched.

In this work, we use the l_1-SMM variant [11], which adds regularization term to the objective function (3): An l_1 regularization on the entries of matrix $\boldsymbol{T}$ and l_2 regularization on the i-vectors themselves.

For details of the training procedure, see the respective paper [11]. We use a publicly available implementation[2] of l_1-SMM to obtain i-vectors.

3. Combining SMM i-Vectors with a Feed-Forward Neural Language Model

Next, we seek to combine i-vectors with an actual language model. As we foresee that an LSTM-based language model already captures a lot of topic-related information in its hidden state, we focus on feed-forward networks.

As the baseline, we use a shallow feed-forward neural network (FN-LM) [13]. We refer to this architecture as *neural n-grams*. Its operation is given as follows:

$$\boldsymbol{y}_t = \text{softmax}(\boldsymbol{O}\boldsymbol{h}_t + \boldsymbol{b}_o) \quad (4)$$

$$\boldsymbol{h}_t = \tanh(\boldsymbol{W}_1 \boldsymbol{e}_{t-1} + \cdots + \boldsymbol{W}_{n-1} \boldsymbol{e}_{t-n+1} + \boldsymbol{b}_h) \quad (5)$$

Here, $\boldsymbol{e}_t \in \mathcal{R}^E$ is an embedding of the input word at timestep t. Matrices $\boldsymbol{W}_{1,\ldots,n-1} \in \mathcal{R}^{H \times E}$ project the respective word embeddings to the hidden layer.

To incorporate SMM i-vectors, we replace (5) by:

$$\boldsymbol{h}_t = \tanh(\boldsymbol{W}_1 \boldsymbol{e}_{t-1} + \cdots + \boldsymbol{W}_{n-1} \boldsymbol{e}_{t-n+1} + \boldsymbol{W}_i \boldsymbol{i} + \boldsymbol{b}_h) \quad (6)$$

[2] https://github.com/skesiraju/smm

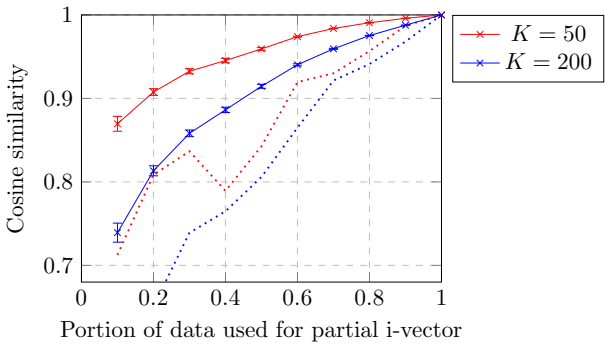

Figure 3: *Cosine similarity of two i-vectors: One summarizing the complete document, the other summarizing only a prefix of an increasing length. The solid line represents an average over validation documents; error bars represent 2 standard deviations. The dotted line represents the worst case for that particular length of prefix.*

Here, $\boldsymbol{i} \in \mathcal{R}^K$ is the document i-vector. Matrix $\boldsymbol{W}_i \in \mathcal{R}^{H \times K}$ is used to project the i-vector to the same hidden space where the input words are aggregated. We call the resulting model *i-vector feed-forward neural language model* (iFN-LM, Figure 1).

As $K \ll |V|$, the increase in number of parameters is negligible.

Furthermore, by inspecting (6), we see that the i-vector is treated as an additional word embedding. Therefore, optimization based on caching projected word embeddings [6] are straightforward to apply. Further optimizations based on changing activation function of the hidden layer or replacing the softmax with NCE are completely independent of this change.

4. Using the WikiText-2 Corpus

We perform all our experiments on the WikiText-2 (WT2) corpus introduced by Merity et al. [14]. This dataset contains high quality articles from English Wikipedia. The training set consists of 2 million tokens. The vocabulary is reduced to 33k unique words by replacing infrequent words by an `<unk>` symbol. It is less pre-processed than other datasets: the vocabulary contains numbers, punctuation, certain mark-up tags, and capitalization has been kept.

In order to extract i-vectors from sensible chunks of text, we split the corpus back into individual articles. This way, we work with 600 training documents and 60 validation ones. The test set also contains 60 documents.

However, we use this new information only to (1) compute i-vectors from respective chunks of data and (2) shuffle the documents between epochs. When training any recurrent

Table 1: *Performance of different unigram language models on the WT2 validation set. The first two rows and the last one show performance of regular unigram models.*

Source of probabilities	test PPL
Train data	911
Validation data	715
Partial i-vector, k=50	684
Partial i-vector, k=200	783
Oracle i-vector, k=50	667
Oracle i-vector, k=200	635
Specific document	252

networks, we pass the hidden state over document boundaries and the models are therefore comparable. Anyway, we found that zeroing the hidden vectors on beginning of documents has negligible impact on final perplexity.

While we work with the original vocabulary for all perplexity evaluations, we use a different setup for training the SMM and subsequently estimating i-vectors: We use the `CountVectorizer` component from Scikit-Learn [15] to lowercase all data, ignore punctuation and remove non-ASCII symbols. This way, the SMM vocabulary is reduced to 29k unique words.

5. Behaviour of Partial i-Vectors

Since we want to compare iFN-LM with other LMs in terms of perplexity, we cannot extract i-vectors from whole documents in test time, as that would mean that iFN-LM could access future words. This leads to extracting i-vectors from the data the LM has already processed. However, it would be costly to recompute the i-vectors at each timestep. Also, we do not want the iFN-LM to respond to local variations in i-vectors, as those are generally not informative about the topic.

Therefore, we first explore how *partial* i-vectors (Figure 2b) differ from the *oracle* i-vector (Figure 2a) as the former represent an increasing portion of a document. We do this in two experiments on the validation data: First, we compute cosine similarity between the partial and the oracle i-vector in the K-dimensional latent subspace. Second, we construct a simple language model conditioned on the i-vectors and compare the perplexity with other unigram models:

$$p(w_j|\boldsymbol{i}) = \text{softmax}(\boldsymbol{O}\boldsymbol{i} + \boldsymbol{b})_j \tag{7}$$

We have performed these experiment with two different i-vector extractors, with latent representations of $K = 50$ and $K = 200$ dimensions. Both regularization coefficients (l_1 on $\boldsymbol{T}$ and l_2 on $\boldsymbol{i}_d$) were set to 10^{-4}. We have trained the models using orthant-wise learning [11] until convergence.

Figure 3 shows how partial i-vectors evolve in the latent space. We can see that both SMM models produce partial i-vectors that converge consistently to the oracle i-vector, in term of cosine similarity. Additionaly, the standard deviation of cosine similarity is rather small. And as expected, the 50-dimensional SMM is more robust in extracting partial i-vectors.

Table 1 compares i-vector LMs with different unigram models. We can see that all i-vector systems perform better than the unigram LM estimated on the training data. Further, we see that in case of $K = 50$, LM based on partial i-vectors is also better than the unigram LM estimated on the validation data. For $K = 200$, the noise in partial i-vectors becomes too large and the resulting system is worse than the validation unigrams.

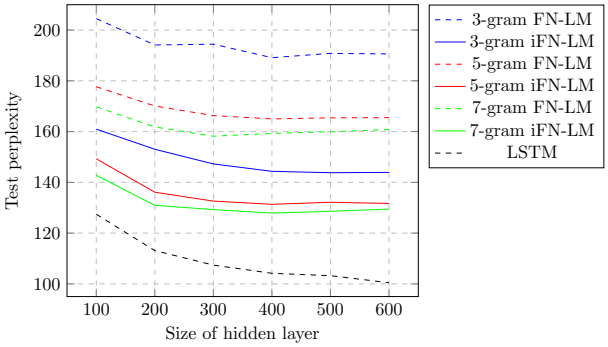

Figure 4: *Performance of neural n-gram models as a function of their size. We display the best model by validation perplexity, selecting from different dropout values and random initializations.*

Given the above results, we use the 50-dimensional SMM for further experiments; we train with oracle i-vectors and we evaluate as appropriate.

6. Evaluating Perplexity Improvements

While the main goal is to show how iFN-LM can be used to achieve improvements in domain adaptation setup, it is instructive to first study its impact on test perplexity, just as other LM works report. To do so, we process the validation and the test data using the partial i-vectors scheme (Figure 2b).

6.1. Training details

We implemented the model using PyTorch [16][3]. We optimized the models using Stochastic Gradient Descent without momentum. We started with learning rate 1 and halved it if the validation perplexity measured at the end of an epoch did not improve. We were training on 20 documents in parallel, processing 30 successive words in parallel (like typical RNN LM with batch size 20 and BPTT 30). We were training the models until convergence, which occurred typically in less than 30 epochs. We experimented with different values of dropout in range $\langle 0, 0.5 \rangle$, applied to the word embeddings and the hidden layer. When experimenting with the size of the model, we kept the embeddings as long as the hidden layer, i.e. $E = H$.

The LSTM models we use as contrastive systems were trained using the cuDNN blackbox implementation, as exposed by PyTorch. These models have 2 layers of LSTM. In this case, we set the initial learning rate to 20 and we clipped gradient norms at 0.25.

We did not use Monte-Carlo dropout [17] in test-time.

6.2. Results

We evaluate the model in terms of perplexity on the test portion of WT2. We explore the performance of the model as a function of the size of the hidden layer, see Figure 4.

We can see that the gain from adding i-vectors into the system is significant and consistent, for all considered model sizes and history lengths. As expected, the biggest improvement is

[3] https://www.github.com/ibenes/BrnoLM

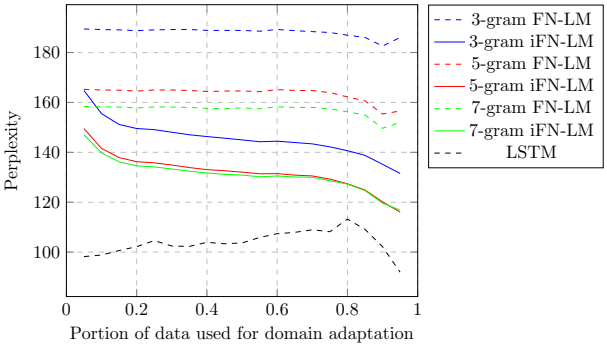

Figure 5: *Performance of different models in the case of domain adaptation. The x-axis goes over amount of data excluded from evaluation and used for domain adaptation, see Figure 2c for illustration. Thus, on the left end, the i-vectors are estimated from the least amount of data (as little as endmost 5 % of every document), while in the right, only a very short prefix of documents is used for perplexity computation.*

achieved in case of the weakest — 3-gram — system. However, the 5-gram and 7-gram systems benefit a lot as well.

It is interesting that 5-gram and 7-gram i-vector systems are already close to each other in performance, approximately twice closer than their i-vector-free counterparts. From this, we hypothesise that the specific word-order only matters in a short window before the current timestep and older history can be quite effectively modeled by its unigram statistics. Similar effect is exploited by classical n-gram cache models [18].

7. Domain Adaptation Scenario

Finally, we seek to evaluate iFN-LM in a simulated domain-adaptation setup. We do this by dividing every document in the test set into two parts (Figure 2c): A prefix of the data is processed by the LM — this is the scored part of the document. The remaining suffix is used for estimate the corresponding i-vector. Note, that this way we never use the introductory paragraph for i-vector estimation.

We pick the best performing models by validation perplexity from Section 6, both with and without i-vectors. Then, we evaluate their test perplexity in the domain-adaptive setup, with the size of domain section ranging from 0.05 to 0.95. Results are presented in Figure 5 as a function of the relative size of the domain section.

The results imply that systems employing i-vectors are rather robust to the amount of data used for i-vector estimation: Even when using as little as 5 % of the data for i-vector estimation, a 3-gram iFN-LM is slightly better than a 5-gram FN-LM, and a 5-gram iFN-LM outperforms a 7-gram FN-LM by a large margin. Further, while it always helps to add more domain data for i-vector estimation, the improvements are modest once we use at least 20 % of the documents.

The LSTM model shows an interesting behaviour, which is convenient to analyze from right to left. It works well for very short prefixes of the documents. This is in line with the performance of FN-LMs, which suggest that the initial parts of the documents are actually easy data. Then, as the processed portion of every document grows, the performance gets worse and the effect is much more pronounced than in case of feed-forward models. Once at least 20 % of every document has been seen (point 0.8 in the graph), LSTM gives better average performance as more and more of the document is processed. We hypothesise that this gradual improvement actually corresponds to the LSTM building a better topic representation in its hidden state. The degradation of performance in the right end, before the breaking point at 0.8, remains unexplained.

Table 2: *Performance of best models with different length of considered history. In case of domain adaptation, initial 75 % of every document is used for scoring, i-vector is extracted from the remaining 25 %.*

	LM scoring		Domain adaptation	
Model	FN-LM	iFN-LM	FN-LM	iFN-LM
3-gram	189.13	143.80	189.09	149.09
5-gram	165.00	131.35	164.99	135.78
7-gram	158.18	127.90	158.20	134.16
LSTM	100.45		104.61	

8. Summary of the Results

To summarize our experiments, we compare the best achieved results in every category. Table 2 captures the perplexities on the test portion of WT2.

We see that iFN-LM always outperforms a simple FN-LM by a significant margin. In the case of regular LM scoring, the addition of i-vectors effectively halves the distance between the feed-forward model and an LSTM. This approximately holds also for domain adaptation, despite the noisy estimate of the i-vectors.

We have also experimented with addition of i-vectors into the LSTM, but achieved improvements were negligible, around 1 point of PPL.

9. Conclusions

In this work, we have enhanced a shallow feed-forward neural language model with document summary i-vector extracted using a Subspace Multinomial Model, abbreviating the resulting model as iFN-LM. This contextual information is introduced in such a way, that is has a negligible impact on the total number of parameters. Also, as the i-vector can be formally understood as an additional word embedding, iFN-LM is open to all optimizations applied to FN-LM.

We have first explored behaviour of SMM i-vectors when they are computed from different amounts of data. Based on our findings, we proposed an effective way of training, when a single i-vector represents the whole document during the training of any language model. Then, we have trained iFN-LMs of different sizes on the WikiText-2 dataset. These models always outperformed their i-vector-free counterparts by a significant margin, on average halving the gap to a two-layer LSTM having the same number of hidden units per layer.

Finally, we have shown that iFN-LM can be successfully exploited in domain adaptation setup. We have observed that SMM i-vectors introduce significant performance improvements already when extracted only from 5 % of the respective documents.

Our future work in this direction will be oriented towards a detailed analysis of topic representation inside the hidden layers of LSTM LMs.

10. References

[1] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur, "Recurrent neural network based language model," in *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, 2010, pp. 1045–1048.

[2] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent Neural Network Regularization," *CoRR*, vol. abs/1409.2329, 2014. [Online]. Available: http://arxiv.org/abs/1409.2329

[3] Y. Gal, "A theoretically grounded application of dropout in recurrent neural networks," *arXiv:1512.05287*, 2015.

[4] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing LSTM language models," *CoRR*, vol. abs/1708.02182, 2017. [Online]. Available: http://arxiv.org/abs/1708.02182

[5] J. G. Zilly, R. K. Srivastava, J. Koutník, and J. Schmidhuber, "Recurrent highway networks," *CoRR*, vol. abs/1607.03474, 2016. [Online]. Available: http://arxiv.org/abs/1607.03474

[6] Y. Huang, A. Sethy, and B. Ramabhadran, "Fast neural network language model lookups at n-gram speeds," in *Interspeech 2017, 18th Annual Conference of the International Speech Communication Association, Stockholm, Sweden, August 20-24, 2017*, 2017, pp. 274–278.

[7] T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. Cernocký, "Empirical Evaluation and Combination of Advanced Language Modeling Techniques." in *INTERSPEECH 2011, 12th Annual Conference of the International Speech Communication Association, Florence, Italy, August 27-31, 2011*. ISCA, 2011, pp. 605–608.

[8] T. Mikolov and G. Zweig, "Context dependent recurrent neural network language model," in *2012 IEEE Spoken Language Technology Workshop (SLT)*, Dec 2012, pp. 234–239.

[9] J. Zhang, D. Qu, and Z. Li, "An improved recurrent neural network language model with context vector features," in *2014 IEEE 5th International Conference on Software Engineering and Service Science*, June 2014, pp. 828–831.

[10] X. Chen, T. Tan, X. Liu, P. Lanchantin, M. Wan, M. J. F. Gales, and P. C. Woodland, "Recurrent neural network language model adaptation for multi-genre broadcast speech recognition," in *INTERSPEECH*, 2015.

[11] S. Kesiraju, L. Burget, I. Szőke, and J. Černocký, "Learning document representations using subspace multinomial model," in *Proceedings of Interspeech 2016*. International Speech Communication Association, 2016, pp. 700–704.

[12] M. Kockmann, L. Burget, O. Glembek, L. Ferrer, and J. Cernocký, "Prosodic speaker verification using subspace multinomial models with intersession compensation." in *INTERSPEECH*, T. Kobayashi, K. Hirose, and S. Nakamura, Eds. ISCA, 2010, pp. 1061–1064.

[13] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A Neural Probabilistic Language Model," *J. Mach. Learn. Res.*, vol. 3, pp. 1137–1155, Mar. 2003.

[14] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer Sentinel Mixture Models," *CoRR*, vol. abs/1609.07843, 2016. [Online]. Available: http://arxiv.org/abs/1609.07843

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[16] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[17] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, pp. 1050–1059.

[18] R. Kuhn and R. D. Mori, "A cache-based natural language model for speech recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 6, pp. 570–583, Jun 1990.