An Analysis of Capsule Networks for Part of Speech Tagging in High- and Low-resource Scenarios*

Andrew Zupon*, Faiz Rafique†, and Mihai Surdeanu†
*Department of Linguistics, †Department of Computer Science
University of Arizona
{zupon, faizr, msurdeanu}@email.arizona.edu

Abstract

Neural networks are a common tool in NLP, but it is not always clear which architecture to use for a given task. Different tasks, different languages, and different training conditions can all affect how a neural network will perform. Capsule Networks (CapsNets) are a relatively new architecture in NLP. Due to their novelty, CapsNets are being used more and more in NLP tasks. However, their usefulness is still mostly untested. In this paper, we compare three neural network architectures—LSTM, CNN, and CapsNet—on a part of speech tagging task. We compare these architectures in both high- and low-resource training conditions and find that no architecture consistently performs the best. Our analysis shows that our CapsNet performs nearly as well as a more complex LSTM under certain training conditions, but not others, and that our CapsNet almost always outperforms our CNN. We also find that our CapsNet implementation shows faster prediction times than the LSTM for Scottish Gaelic but not for Spanish, highlighting the effect that the choice of languages can have on the models.

1 Introduction

Neural networks have become a common tool in natural language processing (NLP) for many tasks, but are different architectures better suited for different tasks, languages, and/or resources? To try to answer this question, we examine the performance of two common neural network architectures, long short-term memory networks (LSTM) (Greff et al., 2017) and convolutional neural networks (CNN) (LeCun et al., 1989), against the newer capsule networks (CapsNets), another neural network architecture based on CNNs (Hinton et al., 2011). While LSTMs and CNNs are common in NLP, capsule networks are relatively new to the field. Due to their recency, it's not always clear if or when they are better than other widely used sequence models. This paper investigates the CapsNet architecture in comparison with LSTMs and CNNs. For our analysis, we apply these three architectures to a part of speech (POS) tagging task, on two languages, and using both low- and high-resource scenarios.

Much of the focus of NLP research is on resource-rich languages like English. However, the performance of different models can depend on the linguistic properties of the language under study (Bender, 2009) and the amount of training data available. To compare the performance of these architectures under different training conditions, we look at Spanish—another resource-rich language—and Scottish Gaelic—a low-resource language using different amounts of training data. This comparison is a step in the right direction, but it does have the limitations of comparing neural network architectures implemented in different frameworks and only comparing two languages.

The main contribution of this paper is comparing the LSTM, CNN, and CapsNet architectures across different training conditions. Our analysis finds that none of the architectures consistently performs best across training conditions. This illustrates how different languages and training conditions can inform which architecture is best suited for a given NLP task, and that there is no obviously correct answer.

2 Related Work

CapsNets are a relatively new type of neural network. Hinton et al. (2011) introduces the architecture, with modifications by Sabour et al. (2017) (dynamic routing) and Hinton et al. (2018) (EM routing). A CapsNet is essentially a modified ver-

*The code and data for this paper can be found at https://github.com/clulab/releases/tree/master/emnlp2020-capsnet.

Proceedings of the First Workshop on Insights from Negative Results in NLP, pages 66–70

ISBN 978-1-952148-66-8

sion of a CNN that trades max pooling for a more data-retentive process called routing by agreement. Instead of the prediction with the highest score getting chosen, the weighted sum of all predictions are considered for classification. Essentially, a CapsNet uses convolution to create first round predictions for objects—primary capsules—and then utilizes routing by agreement to predict the presence of higher level objects—secondary capsules.

Many implementations of CapsNets are designed for image recognition (Hinton et al., 2011; Sabour et al., 2017; Hinton et al., 2018). However, the CapsNet architecture is being applied more and more to NLP tasks, including Chinese word segmentation (Li et al., 2018), and multi-label text classification and question answering (Zhao et al., 2019). This paper continues this path by investigating how CapsNets compare to other neural network architectures for the task of part of speech tagging.

3 Data

Our comparison considers two languages: Spanish[1] and Scottish Gaelic[2]. Spanish is a resource-rich language, being the second most spoken language by number native speakers, fourth most spoken language by total number of speakers, and the third or fourth most widely used language on the internet[3]. Scottish Gaelic is a low-resource language, with 57,375 fluent speakers in Scotland per the 2011 census[4]. The Spanish data come from the UD Spanish AnCora treebank[5]. The Scottish Gaelic data come from the UD ARCOSG treebank[6]. Both corpora use 17 part of speech tag classes.

To study how different low-resource conditions affect training, we artificially create training partitions of different sizes. From the original training data (`train100`), we create partitions consisting of 50% (`train50`), 10% (`train10`), and 1% (`train1`) of the training sentences. The amount of data for each partition is shown in Table 1 for Spanish and Table 2 for Scottish Gaelic. We use FastText word embeddings (Grave et al., 2018) for both Spanish (2,000,000 words) and Scottish Gaelic (14,318 words). The embedding dimension is 300.

[1]Indo-European, Romance

[2]Indo-European, Celtic

[3]Third by number internet users by language, fourth by number of websites by language

[4]Only 1.1% of Scotland's population over 3 years old

[5]UD Ancora

[6]UD Scottish Gaelic ARCOSG

Partition	Sentences	Tokens	Avg. Sent. Length
train100	14,305	446,144	31.2
train50	7,152	255,213	35.7
train10	1,430	43,480	30.4
train1	143	5,912	41.3
dev	1,654	52,511	31.7
test	1,721	52,801	30.7

Table 1: Number of sentences, tokens, and average sentence length for each partition of Spanish. The *n* in the train partitions corresponds to the amount (percent) of the original data used for training.

Partition	Sentences	Tokens	Avg. Sent. Length
train100	1,015	22,963	22.6
train50	507	10,870	21.4
train10	101	1,543	15.3
train1	10	67	6.7
dev	642	9,949	15.5
test	536	9,946	18.6

Table 2: Number of sentences, tokens, and average sentence length for each partition of Scottish Gaelic. The *n* in the train partitions corresponds to the amount (percent) of the original data used for training.

4 Approach

In this section, we describe the implementation details of our CapsNet, CNN, and LSTM methods. Our CapsNet and CNN implementations build on top of Yeung et al.'s implementation[7], which was kept as close as possible to the architectures described by Sabour et al. (2017). Importantly, we tried to keep all three models as close to each other as possible in order to make our comparison as faithful as possible. However, certain differences persist for this project—for example, the CapsNet and CNN are implemented in Python using Tensorflow[8] and Keras[9], whereas the LSTM is implemented in Scala using DyNet.[10] The hyperparameters for our CapsNet and CNN implementations were chosen to be as close as possible to the original implementation. The hyperparameters of the LSTM were chosen to be a reasonable approximation to the CapsNet and CNN models. It is important to note that our comparison does not attempt to compare the best of the best of each architecture.

[7]https://github.com/Chucooleg/CapsNet_for_NER

[8]https://www.tensorflow.org/

[9]https://keras.io/

[10]We used the implementation from the processors library (https://github.com/clulab/processors), which relies on DyNet (https://dynet.readthedocs.

Model	P	R	F1	Train t	Predict t
100% of train, caps, no learn	93.85 (0.35)	94.47 (0.24)	94.16 (0.23)	9,032 s	218 s
100% of train, cnn, no learn	93.76 (0.40)	94.20 (0.11)	93.98 (0.22)	4,802 s	200 s
100% of train, lstm, no learn	**98.54** (0.03)	**98.54** (0.03)	**98.54** (0.03)	3,222 s	165 s
50% of train, caps, no learn	92.78 (0.45)	93.58 (0.20)	93.18 (0.31)	5,386 s	223
50% of train, cnn, no learn	92.36 (0.59)	93.54 (0.12)	92.95 (0.29)	3,392 s	206 s
50% of train, lstm, no learn	**98.31** (0.03)	**98.31** (0.03)	**98.31** (0.03)	1,566 s	175 s
10% of train, caps, no learn	88.37 (0.73)	89.48 (0.67)	88.92 (0.56)	3,186 s	208 s
10% of train, cnn, no learn	88.23 (0.60)	89.17 (0.45)	88.70 (0.33)	3,373 s	191 s
10% of train, lstm, no learn	**96.89** (0.14)	**96.89** (0.14)	**96.89** (0.14)	613 s	170 s
1% of train, caps, no learn	76.63 (1.11)	80.62 (0.74)	78.56 (0.30)	3,370 s	205 s
1% of train, cnn, no learn	73.96 (2.66)	74.78 (0.68)	74.34 (1.44)	2,898 s	187 s
1% of train, lstm, no learn	**91.79** (0.24)	**91.79** (0.24)	**91.79** (0.24)	375 s	162 s
100% of train, caps, learn	96.30 (0.35)	95.61 (0.19)	96.00 (0.08)	14,223 s	219 s
100% of train, cnn, learn	96.01 (0.34)	95.43 (0.15)	95.72 (0.16)	12,794 s	211 s
100% of train, lstm, learn	**98.43** (0.04)	**98.43** (0.04)	**98.43** (0.04)	4,280 s	172 s
50% of train, caps, learn	95.59 (0.27)	94.68 (0.16)	95.13 (0.09)	11,571 s	225 s
50% of train, cnn, learn	95.36 (0.10)	94.61 (0.08)	94.98 (0.06)	11,318 s	206 s
50% of train, lstm, learn	**98.17** (0.06)	**98.17** (0.06)	**98.17** (0.06)	1,333 s	171 s
10% of train, caps, learn	92.45 (0.26)	90.18 (0.32)	91.30 (0.08)	3,767 s	209 s
10% of train, cnn, learn	91.41 (0.42)	89.49 (0.25)	90.44 (0.20)	3,832 s	191 s
10% of train, lstm, learn	**96.84** (0.07)	**96.84** (0.07)	**96.84** (0.07)	1,157 s	172 s
1% of train, caps, learn	84.12 (0.66)	82.65 (0.46)	83.38 (0.24)	3,452 s	205 s
1% of train, cnn, learn	79.71 (0.91)	75.87 (1.01)	77.73 (0.35)	2,979 s	188 s
1% of train, lstm, learn	**91.80** (0.21)	**91.80** (0.21)	**91.80** (0.21)	440 s	178 s

Table 3: Spanish Precision, Recall, and F1 scores. The scores are an average of 5 different random seeds and their standard deviation, along with the average training/prediction times of each model.

4.1 CapsNet Implementation

Our CapsNet model has two 1D convolutional layers, two routing by agreement capsule layers and one fully connected layer. Both convolutional layers have 256 channels, a kernel size of 3, and a stride of 1. The primary capsule layer has 160 capsules with 8 dimensions, a kernel size of 3 and stride of 1. There are 17 secondary capsules with dimensions of 16 and 3 dynamic routing passes.

4.2 CNN Implementation

Our CNN model has three 1D convolutional layers, a max pooling layer, and two fully connected layers. The first two convolutional layers are identical to the first two layers of the CapsNet. The third convolutional layer has 128 channels, size of 3 and stride 1. The two feed-forward layers have a size fo 328 and 192. These settings were chosen to make the CNN implementation as comparable as possible to the CapsNet implementation.

io).

4.3 LSTM Implementation

The LSTM code we used is a reimplementation of the LSTM-CRF approach of Lample et al. (2016). To make this implementation as similar as possible with the previous two approaches, we: (a) removed the CRF layer,[11] and (b) removed the character-level biLSTM encoder from the word embeddings.

Thus, the actual LSTM architecture used consists of three layers: (i) an input layer with 300-dimensional FastText word embeddings; (ii) one biLSTM intermediate layer, where each LSTM has a hidden state of dimension 128 neurons, and (iii) a linear output layer coupled with a softmax function to output the POS tags.

5 Results

In addition to our four training data conditions per language, we evaluate the use of learning the word embeddings during training for all models ("learn" vs. "no learn"), yielding 24 training conditions per language. We trained all models five times with

[11] In initial experiments we observed that the CRF layer had a major contribution to other sequence models such as named entity recognition, but no impact on POS tagging.

Model	P	R	F1	Train t	Predict t
100% of train, caps, no learn	**82.34** (0.82)	80.58 (0.22)	81.45 (0.32)	1,020 s	19 s
100% of train, cnn, no learn	79.40 (1.12)	77.19 (0.96)	78.27 (0.53)	651 s	16 s
100% of train, lstm, no learn	81.86 (0.30)	**81.86** (0.30)	**81.86** (0.30)	256 s	38 s
50% of train, caps, no learn	**75.90** (1.02)	73.98 (0.47)	74.92 (0.54)	809 s	18 s
50% of train, cnn, no learn	71.97 (1.62)	69.33 (0.80)	70.61 (0.47)	616 s	15 s
50% of train, lstm, no learn	75.36 (0.27)	**75.36** (0.27)	**75.36** (0.27)	119 s	37 s
10% of train, caps, no learn	49.24 (2.78)	37.36 (1.43)	42.44 (1.35)	641 s	19 s
10% of train, cnn, no learn	**54.64** (2.16)	50.04 (1.69)	52.18 (0.64)	555 s	15 s
10% of train, lstm, no learn	53.31 (8.11)	**53.31** (8.11)	**53.31** (8.11)	86 s	40 s
1% of train, caps, no learn	7.94 (1.40)	7.82 (1.28)	7.86 (1.26)	486 s	18 s
1% of train, cnn, no learn	16.87 (3.86)	8.42 (3.52)	10.58 (3.09)	546 s	16 s
1% of train, lstm, no learn	**21.37** (3.17)	**21.37** (3.17)	**21.37** (3.17)	44 s	35 s
100% of train, caps, learn	**90.81** (0.35)	87.91 (0.25)	89.34 (0.25)	1,907 s	19 s
100% of train, cnn, learn	88.82 (0.80)	85.57 (0.50)	87.17 (0.32)	1,742 s	16 s
100% of train, lstm, learn	89.84 (0.15)	**89.84** (0.15)	**89.84** (0.15)	317 s	39 s
50% of train, caps, learn	**85.38** (0.62)	81.63 (0.22)	83.46 (0.32)	1,285 s	18 s
50% of train, cnn, learn	82.26 (0.63)	77.95 (0.48)	80.05 (0.54)	1,075 s	16 s
50% of train, lstm, learn	83.66 (0.57)	**83.66** (0.57)	**83.66** (0.57)	155 s	39 s
10% of train, caps, learn	55.16 (1.55)	46.21 (2.04)	50.26 (1.45)	717 s	19 s
10% of train, cnn, learn	57.84 (3.16)	55.13 (1.53)	56.37 (1.26)	629 s	15 s
10% of train, lstm, learn	**65.49** (0.67)	**65.49** (0.67)	**65.49** (0.67)	101 s	40 s
1% of train, caps, learn	8.31 (1.99)	8.55 (2.25)	8.42 (2.10)	509 s	19 s
1% of train, cnn, learn	18.23 (3.47)	11.63 (5.57)	13.63 (5.31)	544 s	16 s
1% of train, lstm, learn	**21.37** (3.17)	**21.37** (3.17)	**21.37** (3.17)	52 s	36 s

Table 4: Scottish Gaelic Precision, Recall, and F1 scores. The scores are an average of 5 different random seeds and their standard deviation, along with the average training/prediction times of each model.

LSTM Hidden State Size	Spanish-100	Spanish-1	Scottish Gaelic-100	Scottish Gaelic-1
64	98.40	91.76	89.46	18.55
128	**98.43**	**91.80**	**89.84**	**21.37**
256	98.42	91.44	89.25	15.02
Capsule Layer Kernel Size	**Spanish-100**	**Spanish-1**	**Scottish Gaelic-100**	**Scottish Gaelic-1**
3	**96.00**	**83.38**	**89.34**	8.42
5	**96.00**	82.69	89.21	**9.92**
7	95.99	82.59	88.62	6.06

Table 5: F1 scores for different hyperparameter choices for LSTM hidden state size and CapsNet capsule layer kernel size on the Spanish and Scottish Gaelic 100% and 1% learned embeddings training conditions. The hyperparameter values in italics (hidden state size 128 and kernel size 3) are the values chosen for our bigger comparison.

a different random seed and averaged the results. Each condition trained for 10 epochs, with early stopping after 2 epochs if the loss did not improve.

The results are given in Table 3 (Spanish) and Table 4 (Scottish Gaelic). We report Precision, Recall, F1, training time, and prediction time. These results show a few trends:

1. The LSTM always outperforms the CapsNet and CNN for Spanish, but the CapsNet and CNN occasionally outperform the LSTM for Scottish Gaelic, whose training dataset is an order of magnitude smaller than the Spanish one.

2. The difference in F1 on the no learn train condition between the Spanish 10% and Scottish Gaelic 100% partitions, which have a comparable number of sentences, is greater for the LSTM (down 9.93%) than the Capsnet (down 2.98%) or CNN (up 3.93%). This suggests that properties of the language, not just the amount of data, play a role in performance.

3. The LSTM benefits only slightly from using learned embeddings, while both the CapsNet and CNN get a much larger performance boost. We see this in the Spanish 1% condition, where the LSTM

F1 improves by 0.01%, but the CapsNet and CNN models improve by 4.82% and 3.39%, respectively.

4. Another obvious difference is in the model training and prediction times. The training time for the CapsNet and CNN is much slower than the LSTM. However, for the Scottish Gaelic case CapsNets are much faster than the LSTM at prediction time. This is an encouraging result, considering that our CapsNet implementation is in Python, whereas the LSTM is implemented in a faster Scala framework.

Overall, the LSTM performs best in most conditions, but the CapsNet often comes close. The CapsNet also usually outperforms the CNN. These performance differences are potentially offset by faster prediction time, depending on the language. The balance between predictive accuracy, training time, and prediction time can be delicate, especially when looking at low-resource languages. These results suggest that depending on the use case, a CapsNet architecture may be preferable to an LSTM, despite the fact that when more resources are available, the LSTM tends to perform the best under the common hyperparameters investigated here.

We also compared different hyperparameters for the LSTM and CapsNet, which is shown in Table 5. The values we chose for the LSTM hidden state size and the CapsNet capsule layer kernel size perform the best in nearly all conditions.

6 Conclusion

In this paper, we compare the performance of three neural network architectures—LSTM, CNN, and CapsNet—on part of speech tagging and find that LSTMs are not always better under the common hyperparameters investigated. We examine how the best performing model changes under different high- and low-resource training conditions using Spanish and Scottish Gaelic. We show that the relatively new CapsNet architecture performs nearly as well as the more complex LSTM under certain conditions and outperforms the CNN under most conditions we examined. These results suggest that there is no one obviously clear choice for a model architecture, and that the properties of a language and the amount of training data can affect which architecture performs best. Future work should address the limitations of this paper. Specifically, future effort should consider more training conditions, including other languages; the consistency of these results within groups of similar languages; and making the implementation of these architectures closer, to guarantee the performance differences are due to the architecture and not an artifact of how they were implemented.

References

Emily M. Bender. 2009. Linguistically naïve != language independent: Why NLP needs linguistic typology. In *Proceedings of the EACL 2009 Workshop on the Interaction between Linguistics and Computational Linguistics: Virtuous, Vicious or Vacuous?*, pages 26–32, Athens, Greece. Association for Computational Linguistics.

Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. 2018. Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

Klaus Greff, Rupesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. 2017. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232.

Geoffrey E. Hinton, Alex Krizhevsky, and Sida D. Wang. 2011. Transforming auto-encoders. In *Artificial Neural Networks and Machine Learning*, pages 44–51. Springer.

Geoffrey E. Hinton, Sara Sabour, and Nicholas Frosst. 2018. Matrix capsules with EM routing. In *International Conference on Learning Representations*.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.

Si Li, Mingzheng Li, Yajing Xu, Zuyi Bao, Lu Fu, and Yan Zhu. 2018. Capsules based chinese word segmentation for ancient chinese medical books. *IEEE Access*, 6:70874–70883.

Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. 2017. Dynamic routing between capsules. In Isabelle Guyon, Ulrike. Von Luxburg, Samy Bengio, Hanna Wallach, Rob Fergus, S.V.N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3856–3866. Curran Associates, Inc.

Wei Zhao, Haiyun Peng, Steffen Eger, Erik Cambria, and Min Yang. 2019. Towards scalable and reliable capsule networks for challenging nlp applications. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.