PREDICTION OF RNA SUBCELLULAR LOCALIZATION WITH FAST-TRAINING QUASI-RECURRENT NEURAL NETWORKS

Alessandro Montemurro [a], Léa Riera [a], Niels Mølgaard Knudsen [b]

[a] DTU Compute, Technical University of Denmark
[b] DTU Bioengineering, Technical University of Denmark

ABSTRACT

The subcellular localization of produced RNA molecules is often of interest in biological research, as misplaced mRNA can, for example, lead to neurological disease in humans. This study presents a deep learning based approach for predicting the localization of an RNA molecule, based only on the RNA sequence and known localization.

Due to the high length of RNA sequences, common deep learning approaches for sequence modelling such as LSTMs are unfit, as the computations at each timestep cannot occur in parallel, leading to slow training. A novel network type, quasi-recurrent neural networks (QRNNs) are used in this study to combat this issue. QRNNs embrace the benefits of both convolutional and recurrent neural networks alike.

A basic QRNN was benchmarked against a similar LSTM in a simple two-class problem, reaching a higher accuracy and training more than 10 times faster at the same batch size. In the full six-class problem, a complex QRNN model reached an accuracy of 44.5%, beating the largest-class baseline of 37.6%

***Index Terms*—** RNA, Subcellular Localization, Quasi-Recurrent Neural Networks, Sequence Modeling

1. INTRODUCTION

It is becoming increasingly apparent that the subcellular localization of RNA molecules is of huge importance. Localization of mRNA allows for regulation by spatially restricting production of proteins. Extra-cellular microRNA (miRNA) have recently seen a surge in interest due to them potentially being cancer biomarkers [1]. In mammals, the localization of long non-coding RNAs (lncRNAs) has been shown to be an important regulator of neural cell growth and size [2]. This also means that various neurological diseases, such as epilepsy and some forms of muscular dystrophy, are caused by errors in RNA localization [3, 4].

As a high-throughput way of determining subcellular localization of RNA is not feasible yet, and given that RNA localization plays a much larger role in human health than previously thought, it remains attractive to use the RNA sequence itself for prediction of its subcellular localization, as the sequence is often easier to determine. There are many steps involved in localization including e.g. the initial binding of RNA-binding proteins, translational repression during the transport and anchoring of the RNA at the destination [1]. This means that the exact motifs determining the localization of a particular set of sequences can often be very convoluted, and as such advanced sequence analysis methods are necessary.

Deep learning techniques are well suited for modelling these kinds of complex black-box situations. In particular, convolutional neural networks (CNNs) and recurrent neural networks (RNNs) have seen much use in sequence analysis due to their ability to understand local motifs and sequentiality of data, respectively [5]. In this paper, we use a novel mix of the two networks to predict subcellular localization of RNA, embracing the useful features from each network type. Different QRNN variants exist - in this paper, the variant using *fo*-pool is described and used.

2. RELATED WORK

Recurrent neural networks have become one of the most commonly used type of neural network for natural language processing tasks (NLP), partly due to their ability to capture sequential information in data. The performance of RNNs has been demonstrated in a range of applications going from sentiment analysis [6] to question answering [7] and language modeling [8]. However the inability of RNNs to properly handle very long sequences restricts their usage. Gated variants including the long short-term memory (LSTM) [9] have not been able to fully deal with this limitation either[10]. Biological polymers of DNA, RNA and amino acids typically have sequence lengths in the thousands, making them somewhat unfit for RNNs and LSTMs for this reason.

Convolutional neural networks [11] are very popular for tasks including image data but has also been used for NLP problems. For example, CNNs has been applied with success in document classification [12] and language modelling [13]. Recent studies even suggest that they should be considered as

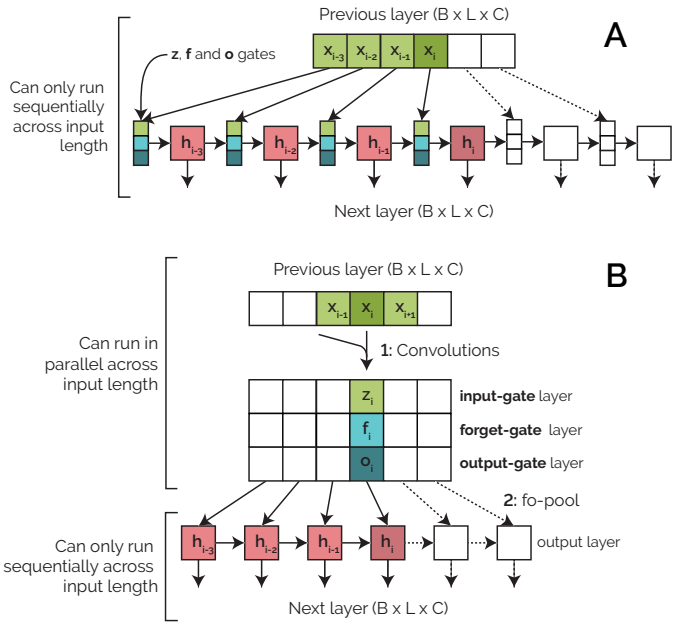

Fig. 1. Comparison between a LSTM (**A**) layer and a QRNN layer (**B**). In **A** computation of the gates in each cell requires the gates and states of the previous cell being computed first. In **B**, gates are computed with convolutions and can thus be computed in parallel. The cell states and hidden states are then computed sequentially with a fast pooling function.

a starting point for language modelling tasks [14]. CNNs possessed several advantages over RNNs, including easier parallellization and better capacity to handle long sequences.

Hybrid architectures were developed to combine advantages of RNNs and CNN but their ability to fully use the large-scale sequence order information stays limited s [15]. Quasi-recurrent neural networks (QRNNs) is a recent type of network that alternate convectional layers with a minimal recurrent pooling function [10]. QRNNs seems promising for processing of long sequences, and has not, to the best of our knowledge, been applied biological data.

3. METHODS

3.1. Model overview

First we present an overview of the classification model. Consider an RNA sequence $\mathbf{S}$ that is L nucleotides (equivalent to characters) long. Before it can be fed into a deep learning model, it must be converted into a numeric format. We decided on using word embeddings for representing single nucleotides.

The five characters used in the RNA sequences in the data set are {A, T, C, G, N}, N being any nucleotide. Each sequence is mapped into a $5 \times L$ matrix by one-hot encoding. Afterwards, the sequences are embedded: each sequence is mapped to a $L \times D$ dimensional space where D is the embedding dimension.

Let $\mathbf{E} \in \mathbb{R}^{D \times 5}$ be the embedding matrix that maps each nucleotide into a D-dimensional vector. Then the embedded sequence X will be

$$X = \mathbf{E} \cdot S \qquad X \in \mathbb{R}^{L \times D}.$$

A vector of predicted class probabilities $\mathbf{y} \in \mathbb{R}^6$ is generated by giving a neural network $\mathbf{NN}$ with parameters θ the embedded sequence X as input, and applying the softmax function σ.

$$\mathbf{y} = \sigma(\mathbf{NN}(E, \theta))$$

The predicted class y is then given by $y = \max(\mathbf{y})$.

Creating a good model for predicting subcellular localization given the sequence is thus equivalent to solving the following optimization problem:

$$\max \mathbb{P}(y = y_c|X),$$

where $X \in \mathbb{R}^{L \times D}$ is the embedded sequence, $y \in \mathbb{R}$ is the predicted class, and $y_c \in \mathbb{R}$ is the true class.

3.2. Quasi-Recurrent Neural Networks

When working with sequential data of variable length, one natural choice of neural network architecture are RNNs and variants such as LSTMs. Each hidden unit is activated on every element of the sequence. The activation $h_{i,j}$ of a hidden unit i at a given position j uses *both* the activation of the previous position, $h_{i,j-1}$, and the current position from the previous layer, x_i, as input. This allows information to persist across the length of the sequence in addition to the local information from the previous layer. In particular, well-tuned LSTM networks can achieve good performance by learning long-term dependencies through using its four gates. However, calculation of these gates can only occur sequentially, which poses a problem when training with very long sequences. On the other hand, convolutions in CNNs can be run in parallel across the entire sequence, but the information hidden in the sequence-like structure of the data is not utilized fully in these computations.

Quasi-Recurrent Neural Networks imitate LSTMs, but merge the benefits from recurrent and convolutional networks. In QRNNs computation of the gates is accomplished using convolutions, thus enabling these heavy calculations to run in parallel. Sequential information is still retained by pooling the gate values in sequence afterwards, similar to RNNs. As a result, QRNNs perform like LSTMs but the computation time dramatically decreases. Figure 1 illustrates the QRNN concepts.

3.3. The mechanism of a QRNN layer

Each layer of a QRNN is composed by two elements: a convolutional layer followed by a pooling operation. Assume

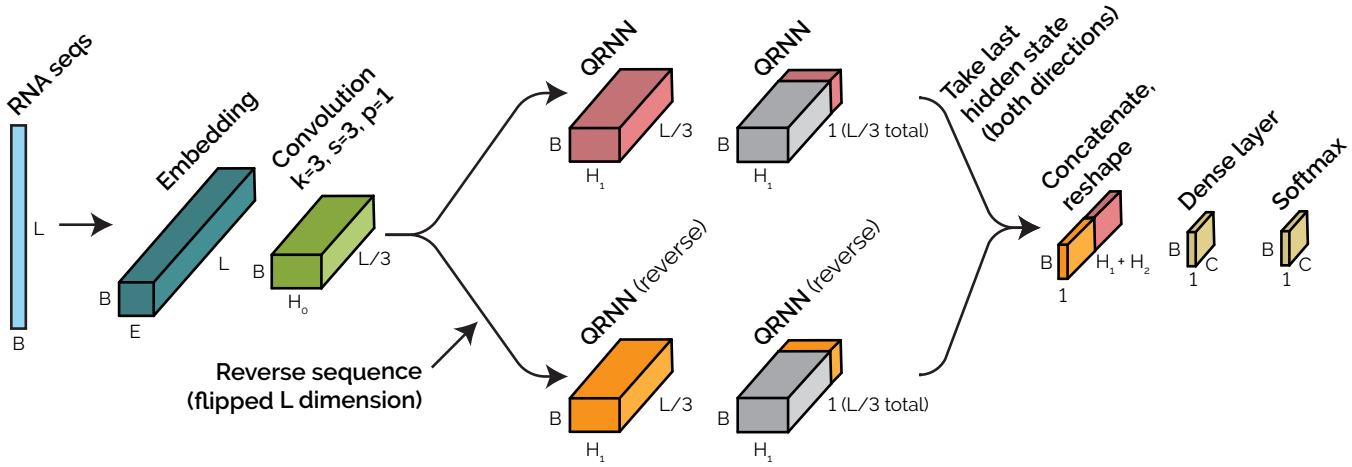

Fig. 2. Architecture of the complex QRNN model uses for modelling the full data set (six classes). For the results presented in this report, the batch size used is $B = 4$, we use $E = 8$ embedding dimensions and each QRNN-layer contains $H = 512$ channels.

a single input sequence X with positions from $1..L$, and D channels at each position, giving $X \in \mathbb{R}^{L \times D}$.

The convolutional component of the QRNN performs three different convolutions at each position t across the L dimension, producing H channels for each, giving $X \in \mathbb{R}^{L \times H \times 3}$. The first convolution produces a value at each position t of a channel, in total $\mathbf{Z} \in \mathbb{R}^{L \times H}$ consisting of vectors $\mathbf{z}_t \in \mathbb{R}^H$.

When looking at a single channel, z-values are similar to the input gate of the LSTM, deciding what to add from the current input x_t to the current cell state c_t. The second convolution produce the forget-gate f_t at each time step, deciding *how much* to remember from the previous cell state c_{t-1} and what to keep of the current input z_t. $f_t = 1$ results in no new input to the cell state.

The third convolution produces the output gate o_t, deciding which parts of the current cell state c_t to pass on to the hidden state h_t, which becomes the output at position t, for use in the next layer.

$$\mathbf{Z} = \tanh(\mathbf{W}_z * X)$$
$$\mathbf{F} = \sigma(\mathbf{W}_f * X)$$
$$\mathbf{O} = \sigma(\mathbf{W}_o * X)$$

Where $\mathbf{W}_z$, $\mathbf{W}_f$, $\mathbf{W}_o$ are the filters of the input, forget and output gate respectively, and $*$ denotes the convolution operation.

Now that the gates have been calculated using convolutions, the hidden states can be calculated using a simple, parameterless pooling function. One option for the pooling function, as presented in [16], is the *fo*-pool (forget/output-pool) that uses forget and output gate:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \odot \mathbf{z}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \mathbf{c}_t.$$

In the above equations, $\odot$ represents the Hadamard (element-wise) product. The choice of such pooling function makes QRNNs much faster than LSTM: a slow hidden-to-hidden matrix multiplication is not performed at each time-step for the QRNN, they rely on a fast element-wise operator.

To sum up, the essence of QRNNs is to do the heavy computations in parallel in the convolutional layer, while keeping the lightweight sequential processing in the pooling layer, where $\mathbf{z}_t$, $\mathbf{f}_t$ and $\mathbf{o}_t$ don't depend on their previous values. In addition, as the pooling component has no trainable parameters, training is not hindered significantly.

3.4. QRNN regularization

In addition to dropout, a QRNN layer can use *zoneout*, which is a feature borrowed from RNNs. As opposed to dropout, where the same set of channels is zeroed across all timesteps, zoneout instead freezes the cell state at a single time step t, not allowing the input to affect it. The channels affected change at each time step. This hides some of the sequential information in a given channel, ideally letting the channel generalize better to new data [17].

4. EXPERIMENTAL SETUP

All models were implemented in Python v.3.6.5 using the deep learning framework PyTorch v.0.4.1 [18]. All experiments were performed using a Google Cloud [1] virtual machine running Debian with a NVIDIA® Tesla ® K80. The code and details of the setup are available on GitHub[2].

[1] https://cloud.google.com/
[2] https://github.com/jcd12/biomind/

4.1. Dataset

The data set is a subset of the RNALocate database [19], consisting of RNA sequences and their curated subcellular localization. The data is divided into two sets, training (90%) and validation (10%), maintaining the same distribution of sequence lengths in each. Copies (same sequence and localization), as well as a few ambiguous sequences (appears twice with different localization) were removed. In figure 3, a histogram showing the distribution of sequence lengths can be found, along with a bar plot of the categories (that is possible destinations). From figure 3, it's clear that the data set was unbalanced with respect to the classes frequencies as well as sequences lengths. To deal with classes imbalance, we used either upsampling, where sequences from the low-frequency classes were replicated, or a weighted cross-entropy loss function, where wrongly predicting low-frequency classes is punished more harshly.

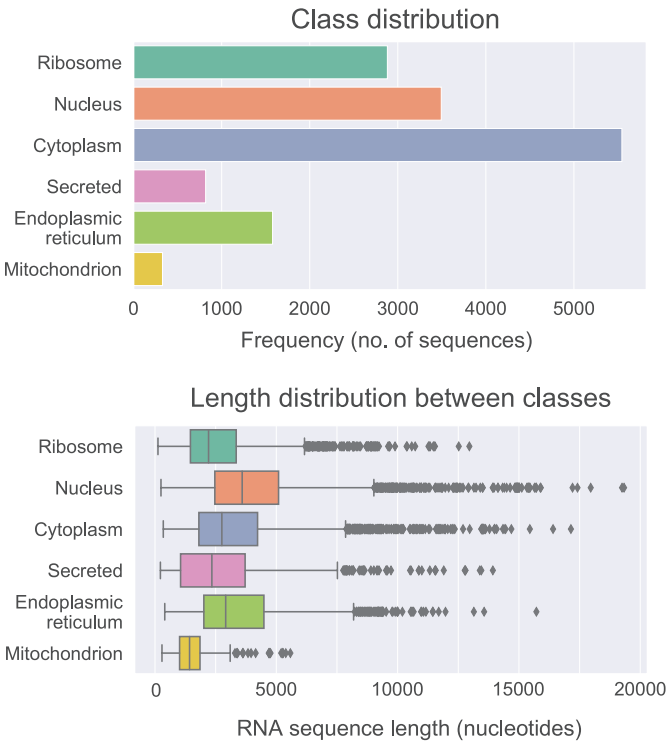

Fig. 3. **Upper**: Bar plots showing the class distribution in the data set. **Lower:** Box plots showing the distribution of sequence lengths in the entire data set. Notice how most sequences are around 3000-7000 nucleotides, while very few are above 30000.

Moreover, due to variable sequence lengths, when training with minibatches, the sequences have to be padded to match the size of the longest sequence in the batch, so e.g. vectorized convolution operations can work across the minibatch. However, all the computations are done also on the padded part of the sequence, resulting in wasted resources with no gain of information. Due to the large variation in sequence lengths, having e.g. a 3k length sequence paired with a 20k length sequence can be a problem. Dynamic batching is used to address this problem: all sequences are first sorted according to their length, then mini-batches are extracted. In this way, sequences with similar lengths are grouped in the same minibatch and computation time spent on the empty padding of the sequences is minimal.

4.2. Experiments

Two experiments were performed. First, a toy experiment using only two of the classes (cytoplasm and nucleus) to test out the feasibility of a QRNN, and to compare the predictive performance and training time to a regular LSTM. Second, after the success of the two-class network, and after ironing out some memory issues in PyTorch, we used all six classes and set up a more complex QRNN to tackle this.

4.3. Models

The architecture of the models used in the two-class toy example can be found in appendix A, as the final, more complex model yields better results on the six-class problem and is thus the primary point of interest.

Figure 2 shows the QRNN architecture for the complex model. The RNA input sequence is embedded and fed into a convolution layer with filter size $k = 3$ and stride $s = 3$. This operation reduces the length of the input of a factor 3. Two bidirectional QRNN layers are then applied, using a filter size of 1 with zoneout $p = 0.1$. The last hidden states from both directions are concatenated and fed into a fully-connected layer. Lastly, the output of the dense layer is passed through a *softmax* activation function, computing the class probabilities.

During the training of the complex model, the network was optimized by minimizing a weighted cross-entropy loss function. The QRNN model was trained using Adam optimizer [20] using a batch-size of 4 and learning rate $\epsilon = 0.0001$. The network was trained for 30 epochs.

5. RESULTS

The complex QRNN model described in 4.3 was trained on the entire data set, while for the simpler LSTM/QRNN models that were generated for the purpose of comparison, only two classes (nucleus and ribosome) were in the data set. In both cases, accuracy is used as performance measure. The performance of the classifiers is compared to the accuracy of a majority classifier that merely assigns each sequence to the largest class. In the two-classes case, the accuracy of a majority classifier is 55%; in the six-classes case, the majority classifier accuracy is 37.6%.

5.1. Two-class toy problem

For the two-class problem, a basic QRNN was benchmarked against a LSTM of similar complexity. Training and validation results over 25 epochs can be seen in figure 4. The QRNN reaches a validation accuracy of approx. 68%, while the LSTM reaches around 64%, both beating random guessing by a fair margin. Despite the LSTM only using the first and last 500 nucleotides of the sequences due to performance issues, a single epoch at a batch size of two takes much longer: around 127 seconds for the QRNN, compared to the LSTM which didn't finish an epoch in the 20 minutes we tried.

Fig. 4. Model accuracies throughout training for the two-class toy problem.

5.2. Six-class problem

For the six-class problem, the complex QRNN model seen in figure 2 was trained for 40 epochs, leading to a final accuracy of 44.5% in the validation data set, beating the baseline of 37.7%. Each epoch took around 660 seconds. The confusion matrix for validation data can be found in figure 5, along with the training and validation accuracy for the first thirty epochs.

Destination	Sensitivity (TPR)	Specificity (TNR)
Cytoplasm	0.48	0.88
Nucleus	0.52	0.66
Ribosome	0.12	0.98
End. Reticulum	0.42	0.76
Secreted	0.21	0.96
Mitochondria	0.95	0.99

Table 1. Recall (true positive rate) and selectivity (true negative rate) values for the complex QRNN model. Based on the predictions from the confusion matrix in figure 5.

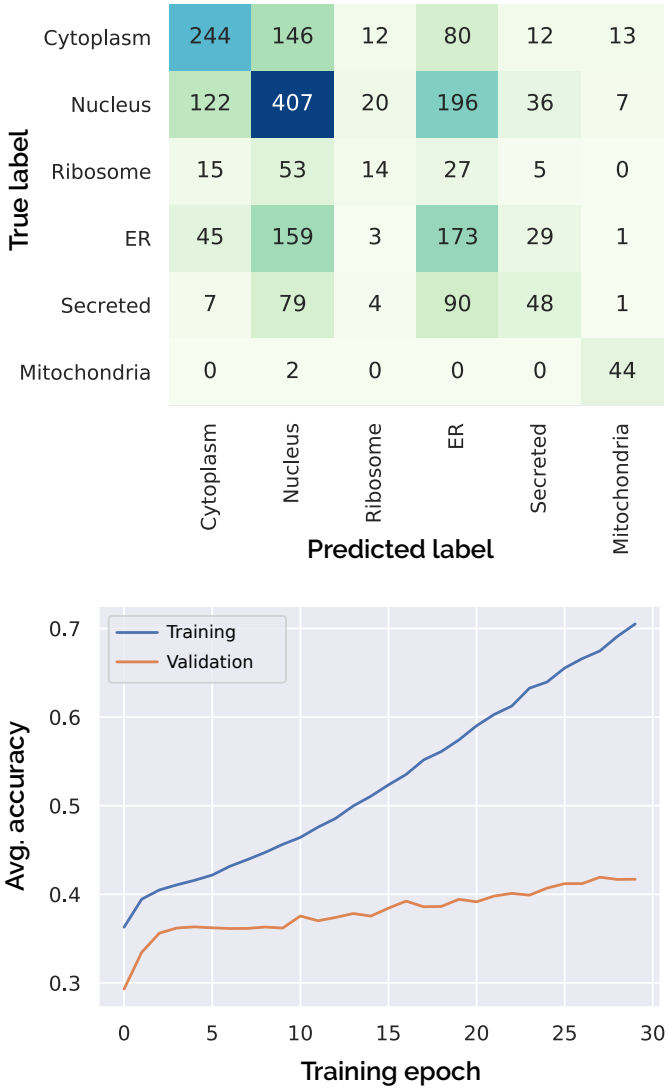

Fig. 5. **Upper:** QRNN confusion matrix for validation set class predictions after 40 epochs. **Lower:** Model accuracies throughout training for the six-class classification problem. Notice that only 30 out of 40 epochs are shown.

6. DISCUSSION

The discussion section will focus on the six-class model, as this classification problem is the focus of the paper.

6.1. Interpretation of results

In this paper, we have presented a deep learning model based mainly on the novel QRNN architecture. While an abundance of software exists for the prediction of **protein** subcellular localization software [21, 22], the authors were not able to find any general RNA localization prediction tools, only a couple focused on non-coding RNA localization [23, 24], despite an

extensive literature search.

Unfortunately, time and computational constraints prevented us from making our own benchmark model that we would feel was fair, though is an obvious next step.

While describing the implemented model as state-of-the-art may be overly ambitious, the model certainly shows that there is *some* extractable pattern hidden in some RNA sequences that determines their subcellular localization. This is especially clear in light of the mitochondrial RNA sequence predictions, where the recall (true positive rate) and specificity (true negative rate) are at 95% and 99%, respectively. This means that the network is not carelessly assigning a lot of sequences to the mitochondria incorrectly, and neither is it being too careful with assigning sequences to this class in fear of the assignment being wrong. The network has a clear idea of when an RNA sequence *should* be sent to the mitochondria, and when it *should not*, based on the sequence alone.

This can not be said as clearly for the other subcellular localizations. In particular some of the sequences from the secreted and ribosome classes seem to be hard to predict correctly. They instead get assigned mostly to the nucleus and ER classes. This indicates that the signals responsible for the secretion and ribosome transport are either:

- very weak/convoluted signals that require a more complex, fine-tuned network to separate correctly from nucleus/ER signals.
- signals located outside of the RNA sequence, requiring other types of data to be fed to the model before it can make the correct prediction

For example, we hypothesize that different species may have different RNA signalling patterns. Feeding the organism class to the model in addition to the RNA sequence, as stated in the second point, could possibly improve the performance.

We can also see that the signals responsible for nucleus and ER may be fairly similar, as the network seems to confuse those two classes more often than between other classes, as also seen by their low specificity. The cytoplasm signals seem less ambiguous, as the specificity here is relatively high despite the recall being in the same range as those two classes.

6.2. Evaluation of model

Looking at the training and validation accuracies during training in 5, it shows that the model fits to the training data very quickly, whereas the validation accuracy shows a rather slow performance increase. A difference this large indicates that the network is overfitting too much to the training data, and partly fails to generalize as it learns. This can likely be solved by introducing more regularization to the model. While 10% zoneout (described in section 3.4 was used in the QRNN layers, dropout was not tested due to time constraints. Some papers report RNN models improving by using up to 50% dropout, so adding dropout may be allow the network to generalize better [25].

7. CONCLUSION

TThrough different experiments, we have shown that QRNNs are a powerful tool for relatively accurate, sequence-based prediction of RNA subcellular localization, being the first non-specialized RNA localization prediction tool that we know of. With a validation accuracy of 44.5%, QRNNs beats a simple majority classifier. The second aspect that renders QRNNs attractive is fast training speed compared to other deep learning approaches, e.g. LSTMs, which is made possible by doing most of the heavy computational work in parallel instead of sequentially.

To improve the performance of the classifier, we propose several strategies. First, increasing regularization by adding dropout to the network, as the large differences in training and validation error indicates that the model is not generalizing enough. Second, we propose feeding additional types of information to the network, for example the species responsible for the particular RNA molecule, as different species likely have different ways of signalling. Third, an ensemble model could be made by combining several classifiers, allowing the individual classifiers to compensate for the weaknesses of other classifiers.

To improve the performance of the classifier, cross-validation could be used where the accuracy of the overall classifier is given but the mean accuracy of the ensemble of classifiers. Cross-validation would allow more complex models to be trained since hard problems are split in simple ones and then an ensemble is taken.

8. ACKNOWLEDGEMENTS

We would like to thank Ole Winther and (especially) our two teaching assistants Alexander R. Johansen and Jose Juan Almagro Armenteros for provided valuable feedback during the project, and putting up with our weird questions.

9. REFERENCES

[1] "Genomic analysis of RNA localization," *RNA Biology*, vol. 11, no. 8, pp. 1040–1050, 2014.

[2] "Nucleolin-Mediated RNA Localization Regulates Neuron Growth and Cycling Cell Size," *Cell Reports*, vol. 16, no. 6, pp. 1664–1676, 2016.

[3] "Dysregulation of mRNA Localization and Translation in Genetic Disease," *The Journal of Neuroscience*, vol. 36, no. 45, pp. 11418–11426, 2016.

[4] Doo Young Lee, Jangsup Moon, Soon Tae Lee, Keun Hwa Jung, Dong Kyu Park, Jung Seok Yoo,

Jun Sang Sunwoo, Jung Ick Byun, Jung Ah Lim, Tae Joon Kim, Ki Young Jung, Manho Kim, Daejong Jeon, Kon Chu, and Sang Kun Lee, "Dysregulation of long non-coding RNAs in mouse models of localization-related epilepsy," *Biochemical and Biophysical Research Communications*, vol. 462, no. 4, pp. 433–440, 2015.

[5] Vanessa Isabell Jurtz, Morten Nielsen, Ole Winther, Søren Kaae Sønderby, Casper Kaae Sønderby, and Alexander Rosenberg Johansen, "An introduction to Deep learning on biological sequence data Examples and solutions," *Bioinformatics*, vol. 31, pp. 31–37, 2016.

[6] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631–1642.

[7] Shayne Longpre, Sabeek Pradhan, Caiming Xiong, and Richard Socher, "A way out of the odyssey: Analyzing and combining recent insights for lstms," *arXiv preprint arXiv:1611.05104*, 2016.

[8] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[9] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[10] James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher, "Quasi-recurrent neural networks," *arXiv preprint arXiv:1611.01576*, 2016.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[12] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun, "Very deep convolutional networks for text classification," *arXiv preprint arXiv:1606.01781*, 2016.

[13] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier, "Language modeling with gated convolutional networks," *arXiv preprint arXiv:1612.08083*, 2016.

[14] Shaojie Bai, J Zico Kolter, and Vladlen Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.

[15] Jason Lee, Kyunghyun Cho, and Thomas Hofmann, "Fully character-level neural machine translation without explicit segmentation," *arXiv preprint arXiv:1610.03017*, 2016.

[16] David Balduzzi and Muhammad Ghifary, "Strongly-typed recurrent neural networks," *arXiv preprint arXiv:1602.02218*, 2016.

[17] R Andomly, Yoshua Bengio, Aaron Courville, and Christopher Pal, "ZONEOUT: REGULARIZING RNNS BY RANDOMLY PRESERVING HIDDEN ACTIVATIONS," pp. 1–11, 2017.

[18] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[19] "RNALocate: A resource for RNA subcellular localizations," *Nucleic Acids Research*, vol. 45, no. D1, pp. D135–D138, 2017.

[20] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[21] Juan Almagro Armenteros, Casper Kaae Sønderby, Søren Kaae Sønderby, Henrik Nielsen, and Ole Winther, "DeepLoc : prediction of protein subcellular localization using deep learning," vol. 33, no. July, pp. 3387–3395, 2017.

[22] Castrense Savojardo, Pier Luigi Martelli, Piero Fariselli, Giuseppe Profiti, and Rita Casadio, "BUSCA : an integrative web server to predict subcellular localization of proteins," vol. 46, no. April, pp. 459–466, 2018.

[23] Zhen Cao, Xiaoyong Pan, Yang Yang, Yan Huang, and Hong-Bin Shen, "The lnclocator: a subcellular localization predictor for long non-coding rnas based on a stacked ensemble classifier," *Bioinformatics*, vol. 34, no. 13, pp. 2185–2194, 2018.

[24] Minakshi Gandhi, Maiwen Caudron-herger, and Sven Diederichs, "RNA motifs and combinatorial prediction of interactions, stability and localization of noncoding RNAs," *Nature Structural & Molecular Biology*, vol. 25, no. December, 2018.

[25] Vu Pham, "Dropout improves Recurrent Neural Networks for Handwriting Recognition," *arXiv preprint arXiv: ...*, 2014.

A. APPENDIX: TWO-CLASS EXAMPLE MODELS

See figure 6 for an overview of the models. Notice that the QRNN uses the entire sequence, while the LSTM uses the first and last 500 nucleotides, as it could not handle entire sequences. The following parameter values were used:

$$B = 2$$
$$E = 20$$
$$H_0 = 96$$
$$H_1 = 96$$

Upsampling of low-frequency classes was used for countering class imbalance during training, instead of a weighted cross-entropy function. The Adam optimizer was used with a learning rate was 0.001, using cross-entropy as a loss function.

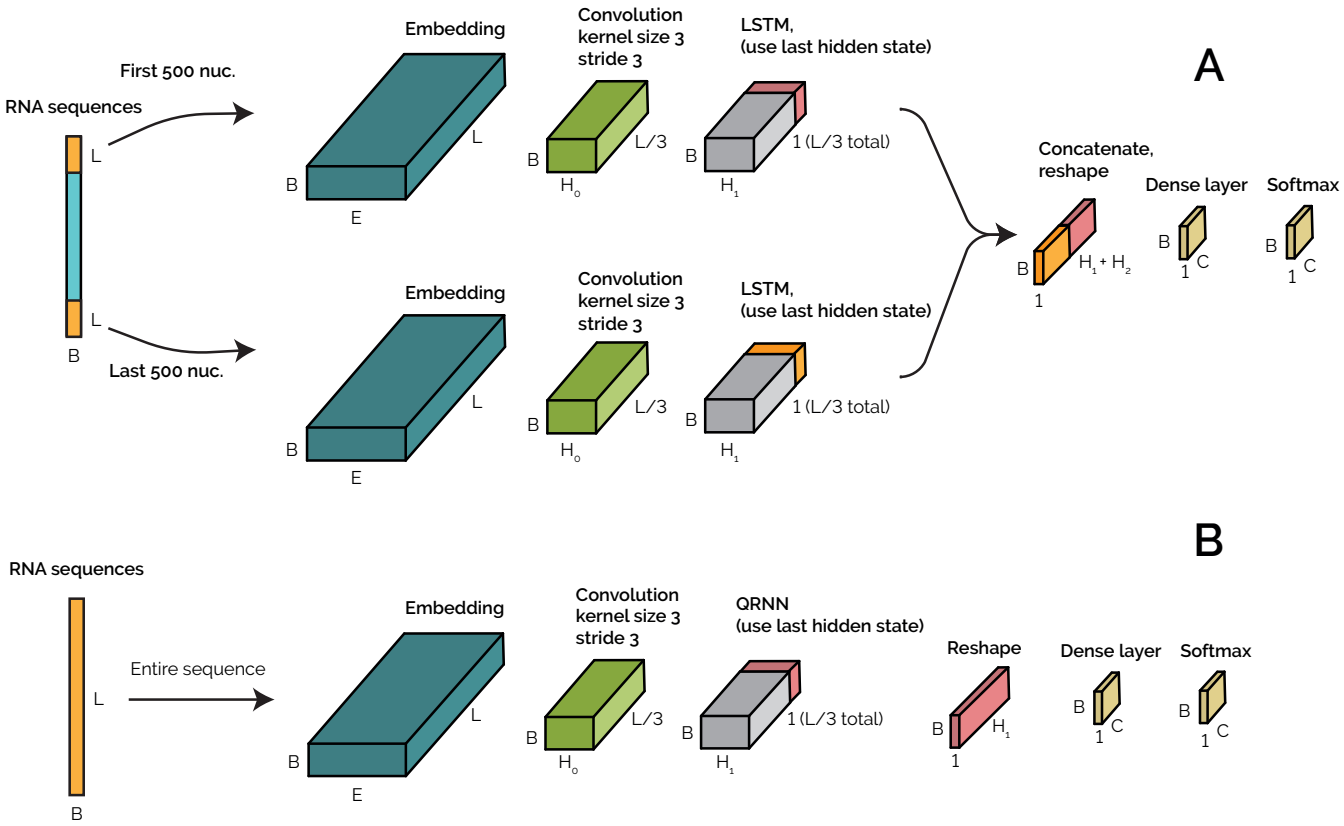

Fig. 6. Architecture of the LSTM (A) and QRNN (B) applied in the toy two-class problem.