PROTEIN AND SECONDARY STRUCTURE PREDICTION WITH CONVOLUTIONS AND VERTICAL-BIDIRECTIONAL LSTMS

Alexander Rosenberg Johansen[a], *John*[b], *Helene* [c], *Esben*[a], *Gurli* [b]

[a] Department for Applied Mathematics and Computer Science, Technical University of Denmark (DTU), 2800 Lyngby, Denmark

[b] Center for Biological Sequence Analysis, Technical University of Denmark (DTU), 2800 Lyngby, Denmark

[c] Bioinformatics Centre, Department of Biology, University of Copenhagen, Copenhagen, Denmark

ABSTRACT

We trained a bidirectional long-short term memory neural network with convolutional filters and hidden states from previous long-short term memory passes as input for mapping the amino acid and sequence profiles from the CB513 data set onto secondary protein structures. On the test data we achieved an accuracy of 68.5%, which is an improvement on state-of-the-art. The neural network, which has 2.1 million parameters, consists of three convolutional layers, followed by a dense layer, then bidirectional long-short term memory layers where the hidden state from the forward pass is used as input in the backwards pass, then feed into a dense layer with dropout before the final 8-way softmax layer. The convolutional and dense layers used non-saturating neurons. The network was regularized by dropout and the L_2 norm. Further gradients where normalised and output logit clipped to stabilize training.

***Index Terms*—** biology, deep learning, long-short term neural network, convolutional neural network, GPU

1. INTRODUCTION

Current approaches to annotating secondary protein structures from amino acid profiles and sequence profiles makes uses of machine learning methods[1]. Recently deep learning methods for modelling sequential data has emerged and received state of the art performance[2].

The approaches has risen as the challenge of annotating secondary protein structures often is predictable from structures along the sequence of the amino acid and sequence profiles[**?**]. A Long-Short Term Memory (LSTM) network has the property of parametizing 'memory cells', which can inform the model about earlier data in a sequence, thus utilizing the sequential structure[3]. Further the local sequence typology has a high importance for predicting secondary protein structures[**?**]. A Convolutional Neural Network (CNN) is an approach to parametrize filters for detecting defined spaces of data [4], thus convolutions propose a method for utilizing the local structures along the sequence of amino acid and sequence profiles.

As the problem of predicting secondary structures given a sequence of amino acid and sequence profiles is not direction or time dependant, as the whole sequence is given at once, Bidirectional LSTM can be utilized to analyse the sequence from both directions[5].

The contribution of this paper is testing convolutional filters on the input sequence and feed the filters into a Bidirectional LSTM. We further test using the hidden state of the forward pass as input for the backwards pass, which to our knowledge has not been tested in LSTM literature. As well as L^2 regularization.

2. MATERIALS AND METHODS

2.1. The Dataset

The test set which we performed predictions on is the CB513 test set on secondary protein structures. To train our classifier we used the filtered version of cb513 data set with no further preprocessing applied[1]. For every point in the sequence there is a recording of the amino acid and sequence profiles coded in one-hot encoding. The purpose of the data set is to predict the secondary structure of every sequence step. For validation purposes we extracted 256 sequences in the training set to optimize hyper parameters in our model.

2.2. The Model

The architecture of the network is defined by three different types of learn-able layers, the fully connected, the convolutional layer and the LSTM. Below, we describe how the different layer types used in our network in more detail.

2.2.1. Neural Network

The fully connected layer, also known as the Multi Layer Perceptron(MLP)[6] is a collection of layers which performs non-linear transformations on the previous layer. The first

[1] http://www.princeton.edu/ jzthree/datasets/ICML2014/

layer, $l = 0$, is considered the input. In our case a one-hot encoding of the amino acid and sequence profiles. The standard MLP is defined in the following linear algebraic operation:

$$z_{l+1} = h_l \theta_{l+1} \tag{1}$$

$$h_{l+1} = a(z_{l+1}) \tag{2}$$

Where h_l is the current layer and θ is the weight matrix used to compute the linear combination of the input; z_{l+1}. A non-linear activation function, $a(z_{l+1})$ is applied to the linear combination of the input which results in the next layer; h_{l+1}.

Most commonly used functions for the activation function $a(z)$ includes the Logistic Sigmoid $a(z) = 1/(1 + e^{-z})$ and the Hyperbolic Tangent $a(z) = (e^z - e^{-z})/(e^z + e^{-z})$. Non-saturating functions such as the Rectifier Linear Unit (ReLU) $a(z) = max(0, z)$ has become popular as their gradients do not vanish and it is computationally easier, which makes optimizing the network faster using stochastic gradient descent [4]. Unfortunately the ReLU suffers from "dead" gradients (large gradients could cause the weights to update in such a way that the neuron will never activate), which can make the network unable to learn. To avoid this problem we are using the Leaky ReLU $a(z) = max(\alpha z, z)$ with $\alpha = 0.3$[7], where "Leaky" refers to the added slope when $a(z) < 0$.

As regularization technique, to avoid overfitting, we add Bernoulli dropouts[8] as a layer, it works as defined:

$$h_{l+1} = h_l \odot p \tag{3}$$

where $p_i \epsilon [0, 1]$ is a random sampled number with a hyper parameter determining bias towards 0 or 1 and $\odot$ is element-wise multiplication.

2.2.2. Convolutional Neural Network

The 1D convolutional layers are implemented as follows:

$$z^{id}_{\ell+1} = x^i_\ell \Theta^d_{\ell+1},$$

$$h^{id}_{\ell+1} = a(z^{id}_{\ell+1}), \tag{3}$$

Where the input $x^i_\ell \in R^{kc \times 1}$ represents a co-located vector of length k in c channels and i refers to the specific co-located vector. The weight matrix $\Theta^d_\ell \in R^{1 \times kc}$ corresponds to a spatial weight filter with kc connections, in the d^{th} output channel, between the input layer and each neuron in the output layer. As a result, convolutional layers are configurable to their filter size k and number of filters d [4], leading to many possible configurations of the CNN architecture. It has been found that multiple convolutional layers on top of one another allows to reduce parameter space and model nonlinear filters[9]. Also using different filter sizes on the same input can increase performance[10]. In our solution we found inputs of different sizes to have the superior validation accuracy.

2.2.3. Bidirectional Long-Short Term Memory with Vertical Links

A Recurrent Neural Network(RNN) is a type of neural network layer that works along the sequence of the data and utilizes computation on previous input in the sequence[3]. The LSTM is a special version of the RNN that uses memory cells to increase focus on important part of the sequence and improve convergence speed. It is defined as described in Graves, 2012[3] without peepholes.

$$i_t = \sigma(x_t W_{xi} + h_{t-1} W_{hi} + b_i) \tag{4}$$

$$f_t = \sigma(x_t W_{xf} + h_{t-1} W_{hf} + b_f) \tag{5}$$

$$o_t = \sigma(x_t W_{xo} + h_{t-1} W_{ho} + b_o) \tag{6}$$

$$g_t = tanh(x_t W_{xg} + h_{t-1} W_{hg} + b_g) \tag{7}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{8}$$

$$h_t = o_t \odot tanh(c_t) \tag{9}$$

Where $\sigma(z)$ is the sigmoid function $1/(1 + e^{-z})$ and x_t is the input from the previous layer: h^{l-1}_t. The LSTM only works in one direction. To utilize information from both directions two LSTM's working from each direction of the sequence are applied are applied. Their hidden states, h_t for $t = [1, 2, ..., T]$ where T is sequence length, Are stacked such as suggested by Schuster & Paliwal, 1997 [5].

Further to incorporate sharing of hidden states, we propose the "Vertical Links" which is, illustrated in figure 1, stacking the input, x_t, to the next pass with the hidden state, h_t, of the previous pass. This operation allows the backwards pass to have a hidden state from both directions while computing. The concept could be extended to stacking several passes on top of one another with "Vertical Links", however, we leave this for further research.

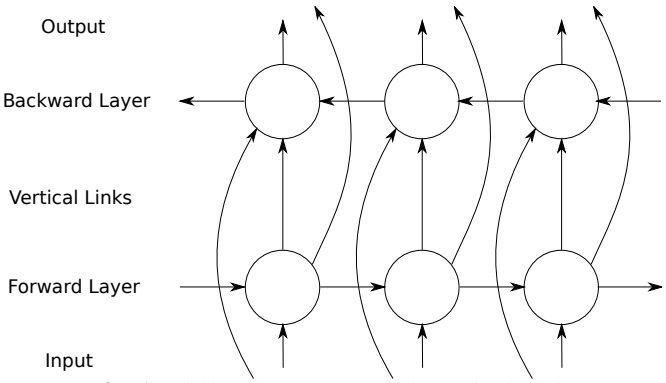

Fig. 1: Bidirectional RNN with Vertical Links

2.2.4. Overall architecture

As depicted in figure **??** the network contains nine layers, the first three are convolutional layers with different filter sizes on the input, these are then concatenated with the input and feed

to a fully connected network representing the fourth layer, the fifth is a LSTM working in the forward direction which has its hidden states concatenated with the fourth layer, this is then feed into the sixth layer being an LSTM working in the backwards direction. The LSTMs(fifths and sixth layers) hidden states are concatenated and feed to a seventh layer being a dropout layer, the eight layer is a dense layer. The ninth layer is an 8-way softmax function(to normalize the output) which provides us with a probability of the secondary protein structures.

To optimize, also known as training, our network we minimize the multi class cross-entrophy objective on the basis of the probability output from the neural network:

$$L(x, y) = -\sum_{c} y_c \ln(f_c(x)) \tag{10}$$

Where x is the input vector, y the true label, $f(x)$ the neural networks probability prediction and c the class. The probability prediction is clipped to be between $1e-5$ and $1-1e-5$ to stabilize training and avoid exploding gradients.

We further apply the L_2 norm such that.

$$regterm(\lambda, \theta) = \lambda\left(\sum_{n=1}^{N} \theta_n^2\right) \tag{11}$$

Where λ is a tuneable hyperparameter, N is the number of non-bias weights and θ is the weights in the neural network. This is then added to the cost function.

$$L_{reg}(x, y) = L(y, f(x)) + regterm(\lambda, \theta) \tag{12}$$

For further details on overall architecture see appendix **??**.

2.2.5. Details of learning

We trained our model with the first order method: stochastic gradient descent(SGD). SGD works by utilizing chain ruling to take the partial derivative of the loss function with respect to each weight vector in the network, such that.

$$g = \frac{\partial L(x, y)}{\partial \theta} = \frac{\partial L(x, y)}{\partial f(x)} \frac{\partial f(x)}{\partial \theta} \tag{13}$$

Where the partial derivativ for each weight, g, is used to update the weights, such that:

$$\theta_{b+1} = \theta_b - \alpha g_b \tag{14}$$

Where α is a tunable parameter determining the size of gradients updated with and b is the current training batch. We use a version of SGD called RMSProp[11] which uses historic information to adapt the learning rate for every parameter doing training, such that:

$$\theta_{b+1} = \theta_b - \alpha G_b^{-1/2} \odot g_b \tag{15}$$

Where G is a moving average og squared gradients, such that:

$$G_{b+1} = \rho\, G_b + (1-\rho) g_b^2 \tag{16}$$

Where ρ is the gradient moving average decay factor.

To avoid exploiting gradients we normalized the gradient, g, if its norm exceeded a threshold of 20. Given such case, all of the gradients would be scaled by:

$$norm_2 = \| \frac{g}{batch\_size} \|_2 \tag{17}$$

Hyper parameters used in the models trained are in Appendix **??**

We trained the network until we minimized the validation error. Occasionally we would run well performing models on the test set and found a high correlation between validation and the test performance.

The training was performed on a Nvidia GeForce GTX Titan X GPU, we used the python built Theano library[12][13] to compile to CUDA (a GPU interpretable language). On top of Theano we applied the neural network specific Lasagne library to built and configure our models[14].

The models took on average 24 hours to train on the Nvidia GPU.

3. RESULTS AND APPLICATIONS

Our results on the CB513 dataset is summerized in table 1. Our 10 model network average achieves an accuracy of 42.0%. The best known previous result is at 67.4% with an approach of using a Bi-Directional LSTM[2], which we also manage to achieve in our initial model build.

Models	AUC
Best know result, Sønderby et al. 2014 [2]	0.674
Bi-LSTM	0.674
Bi-LSTM w. L^2	0.677
Vertical Bi-LSTM w. L^2	0.680
CNN Bi-LSTM w. L^2	0.683
CNN Vertical Bi-LSTM w. L^2	0.685
20 model avg. CNN Vertical Bi-LSTM w. L^2	**0.42**

Table 1: Benchmark experiment amongst various neural networks models.

It is visable from the

4. DISCUSSION

Our results show that convolutional filters, vertical links and L^2 regularization are all able to improve performance and their improvement are not exhaustive of one another. During the training we further found that vertical links with no convolutional filters did not have any significant improvement

when using L^2 regularization. However, when convolutional filters where applied L^2 regularization became critical, as the network otherwise would overfit. So adequate testing of hyperparameters are essential for constructing an optimal fit between convolutional filters, vertical links and L^2 regularization. Additionally we tested batch normalization[15] which interestingly enough converged four times as fast, but overfitted and did not manage to obtain the same validation performance. Testing batch normalization might increase the performance of the network and could be the interest of further research.

5. REFERENCES

[1] Jian Zhou and Olga G Troyanskaya, "Deep supervised and convolutional generative stochastic network for protein secondary structure prediction," *arXiv preprint arXiv:1403.1347*, 2014.

[2] Søren Kaae Sønderby and Ole Winther, "Protein secondary structure prediction with long short term memory networks," *arXiv preprint arXiv:1412.7828*, 2014.

[3] A. Graves, "Supervised sequence labelling with recurrent neural networks," *Springer*, 2012.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, Eds., pp. 1097–1105. Curran Associates, Inc., 2012.

[5] Schuster & Paliwal, "Bidirectional recurrent neural networks," *Signal Processing, 45*, 1997.

[6] Dennis W Ruck, Steven K Rogers, Matthew Kabrisky, Mark E Oxley, and Bruce W Suter, "The multilayer perceptron as an approximation to a bayes optimal discriminant function," *Neural Networks, IEEE Transactions on*, vol. 1, no. 4, pp. 296–298, 1990.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015.

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[10] Søren Kaae Sønderby, Casper Kaae Sønderby, Henrik Nielsen, and Ole Winther, "Convolutional lstm networks for subcellular localization of proteins," *arXiv preprint arXiv:1503.01919*, 2015.

[11] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude," COURSERA: Neural Networks for Machine Learning, 2012.

[12] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010, Oral Presentation.

[13] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio, "Theano: new features and speed improvements," Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[14] Sander Dieleman, Jan Schlter, Colin Raffel, Eben Olson, Sren Kaae Snderby, Daniel Nouri, Daniel Maturana, Martin Thoma, Eric Battenberg, Jack Kelly, Jeffrey De Fauw, Michael Heilman, diogo149, Brian McFee, Hendrik Weideman, takacsg84, peterderivaz, Jon, instagibbs, Dr. Kashif Rasul, CongLiu, Britefury, and Jonas Degrave, "Lasagne: First release.," Aug. 2015.

[15] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.