An introduction to Deep learning on biological sequence data - Examples and solutions

Jurtz, Vanessa Isabell; Johansen, Alexander Rosenberg; Nielsen, Morten; Almagro Armenteros, Jose Juan; Nielsen, Henrik; Kaæ Sønderby, Casper; Winther, Ole; Kaæ Sønderby, Søren

Published in:
Bioinformatics

Link to article, DOI:
10.1093/bioinformatics/btx531

Publication date:
2017

Document Version
Peer reviewed version

Link back to DTU Orbit

Citation (APA):
An introduction to Deep learning on biological sequence data – Examples and solutions

Vanessa Isabell Jurtz¹, Alexander Rosenberg Johansen², Morten Nielsen¹, Jose Juan Almagro Armenteros¹, Henrik Nielsen¹, Casper Kaae Sønderby³, Ole Winther²,³*, Søren Kaae Sønderby³*

¹Department of Bio and Health Informatics, Technical University of Denmark, ²Department of Applied Mathematics and Computer Science, Technical University of Denmark, ³Department of Biology, University of Copenhagen, and ⁴Instituto de Investigaciones Biotecnológicas, Universidad Nacional de San Martin, Buenos Aires, Argentina

*To whom correspondence should be addressed.

Abstract

Motivation: Deep neural network architectures such as convolutional and long short-term memory networks have become increasingly popular as machine learning tools during the recent years. The availability of greater computational resources, more data, new algorithms for training deep models and easy to use libraries for implementation and training of neural networks are the drivers of this development. The use of deep learning has been especially successful in image recognition; and the development of tools, applications and code examples are in most cases centered within this field rather than within biology.

Results: Here, we aim to further the development of deep learning methods within biology by providing application examples and ready to apply and adapt code templates. Given such examples, we illustrate how architectures consisting of convolutional and long short-term memory neural networks can relatively easily be designed and trained to state-of-the-art performance on three biological sequence problems: prediction of subcellular localization, protein secondary structure and the binding of peptides to MHC Class II molecules.

Availability: All implementations and datasets are available online to the scientific community at https://github.com/vanessajurtz/lasagne4bio.

Supplementary information: Supplementary data are available at Bioinformatics online.

1 Introduction

Machine learning is a specialization of computer science closely related to pattern recognition, data science, data mining and artificial intelligence (Machine Learning -Encyclopedia Britannica, 2016). Within the field of machine learning, artificial neural networks, inspired by biological neural networks, have in recent years regained popularity (Schmidhuber, 2015). Their most recent success began with the development of effective methods to train deep neural networks (networks with multiple hidden layers), and the coining of the term deep learning around 2006 (Hinton and Salakhutdinov, 2006; Schmidhuber, 2015). Since then improvements have been made in part enabled by the access to greater computational resources, especially graphics processing units (GPU), enabling training of deep neural networks containing many parameters in reasonable time. Given this, specialized neural network architectures like convolutional neural networks (CNN) and recurrent neural networks (RNN) with long short-term memory cells (LSTM) can now be trained efficiently and have been successfully applied to many problems including image recognition (Ciresan et al., 2011; Krizhevsky et al., 2012) and natural language processing tasks such as speech recognition (Geiger et al., 2014) and language translation (Sutskever et al., 2014).
The successes of neural networks have led to the development of various programming frameworks to build and train neural networks. Examples are PyTorch (http://pytorch.org/), Caffe (http://caffe.berkeleyvision.org) and TensorFlow (https://www.tensorflow.org). Our framework of choice here is Lasagne (Dieleman et al., 2015), a well-established easy to use and extremely flexible lightweight Python library built on top of the Theano numerical computation library (Bastien et al., 2012). While most other frameworks require the user to learn a dedicated programming language, Lasagne is Python-based and therefore relatively easy to use for bioinformaticians already programming in Python. Further Lasagne’s active community ensures that the latest neural network training algorithms and architectures are available to the user.

Within bioinformatics, examples of deep learning applications include prediction of splicing patterns (Leung et al., 2014), DNA and RNA targets of regulatory proteins (Alipanahi et al., 2015), protein secondary structure (Wang et al., 2016) and biomedical image analysis (Moeskops et al., 2016; Cha et al., 2016). However, the number of applications is still relatively small, and the application of deep learning methods within biology is in our view being held back due to a lack of examples and programming templates with a biological background facilitating a head start on the use of these libraries for non-experts.

Here, we seek to alter this by providing a non-expert introduction to the field of deep neural networks with application examples and ready to apply and adapt code templates illustrating how convolutional and LSTM neural networks can be successfully designed and trained on biological data to achieve state-of-the-art performance in prediction of 1) protein subcellular localization, 2) protein secondary structure, and 3) peptides binding to Major Histocompatibility Complex Class II molecules. Implementations of the methods are available online to be used by non-expert end-users as templates for developing models to describe a given problem of interest https://github.com/vanessajurtz/lasagne4bio. The code can run both on CPU and cuda-enabled GPU. GPU gives of the order of a 100fold speed-up.

2 Deep learning background

The traditional neural network architecture is the feed forward neural network with one hidden layer, in which each input neuron is connected to each neuron in the hidden layer and each neuron in the hidden layer is in turn connected to each neuron in the output layer. A non-linear so-called activation function, most often tanh or sigmoid, is applied to give the output of a neuron given its input. For classification problems the number of units in the output layer is equal to the number of classes. The softmax function is often used to make the outputs probabilistic, that is given an input the network assigns a probability to each class. For the prediction of a continuous output, a single linear output neuron is typically used. All connections in the network are directed from input to output, and it is possible to build deep networks by adding more hidden layers where each neuron will be connected to each neuron in the following layer. This type of architecture is visualized in Figure 1A. The hidden layer of a feed forward neural network is also referred to as a dense layer, because is fully connected to the previous layer.

In convolutional neural networks (CNNs) information also flows only from the input to the output, layer by layer. They are however not fully connected, but instead slide a filter (a set of weights) over the input that feeds into a different hidden neuron in the hidden layer, here visualized on the f1 row. Multiple filters will give multiple inputs to the next layer (f1, f2, f3, ...). A filter can be visualized as a sequence motif. This helps to understand which amino acids the filter prefers at each sequence position. When the filter is slid over the input sequence, it functions as motif detector and becomes activated when the input matches its preference. For example, this filter has negative output for sub-sequences ADC and positive for DCD.

Recurrent neural networks (RNNs) are neural network models for sequential data. In addition to the feed forward connections they have time-delayed connections between the neurons of a hidden layer (Graves, 2012). RNNs process the sequence one element at a time, in biological sequence context one residue after the other. The information therefore flows both from input to output and along the sequence. In this way memory is generated and the neural network gains the ability to store and integrate information from past inputs. Long short-term memory (LSTM) neural networks are a special type of RNNs in which the scalar-valued hidden neuron is replaced with the LSTM memory block. The LSTM memory block is inspired by a computer memory cell where context-dependent input, output and forget gates control what is written to, read from and kept in the cell in each time-step. In this way, it is easier for the network to store a given input over many time steps.

LSTMs (and in general RNNs) can by construction handle input sequences of varying length. This makes RNNs highly flexible for different types of tasks where either the input (many-to-one), the output (one-to-many) or both are sequences (many-to-many). Many-to-one is a model to process an entire sequence one amino acid at a time and then
predict whether or not it has a certain biological property after having seen the entire sequence (here exemplified in the prediction of peptide binding to MHCII). The many-to-many approach, also known as tagging or sequence labeling, can for example be used to predict the secondary structure classification of each amino acid in a protein. Performance can be improved by changing to the many-to-many bidirectional approach (biLSTM) where the network processes the input sequence forwards and backwards and bases each prediction not just on what came before the current position but also on what comes after. The same approach can be used in a many-to-one approach here exemplified in protein subcellular localization prediction.

For a more detailed explanation of the CNN and LSTM architectures, including the mathematical formulas to calculate the output and update weights, we refer the reader to textbook “Deep Learning” by Goodfellow et al. (2016).

2.1 Designing your model

In this section, we highlight some of the key issues to consider when constructing deep neural networks such as how to avoid overfitting when increasing network size and depth.

2.1.1 Sequence encoding

When working with biological sequence data the numeric representation of the data influences model performance. An obvious solution is to use the so-called sparse or one-hot encoding (see Figure 1A), where an amino acid is represented by a vector of the length 20, containing a single one and 19 zeros. The parameters connecting an amino acid input to the first hidden layer is often called the embedding for that amino acid. Another possibility is to use the BLOSUM matrix (Eddy, 2004) for encoding, representing each amino acid by its corresponding row in the BLOSUM matrix. The BLOSUM matrix captures information about which pairs of amino acids are easily interchangeable during evolution, but it does not capture information about the evolutionary constraints on a protein family (i.e. which amino acid positions are highly conserved and which are variable). This information can effectively be captured in a sequence profile. A sequence profile has the dimensions protein length times the number of amino acids and is conventionally generated by running PSI-BLAST (Altschul et al., 1997) against a reference database. Encoding protein sequences as such profiles has demonstrated very helpful for prediction of for instance secondary structure (Jones, 1999).

2.1.2 Regularization and training

Regularization techniques aim to prevent overfitting and are especially relevant for large models containing many parameters. Before applying regularization techniques it is important to partition the dataset carefully dealing with data redundancy and set up the neural network training using cross-validation (further detailed in the supplementary materials). Dropout (Hinton et al., 2012) reduces overfitting by introducing discrete noise during training. For every training input the output of each hidden unit is set to zero with a certain preselected probability. Batch normalization (Ioffe and Szegedy, 2015) re-parameterizes the hidden unit activation in order to increase convergence speed but also makes the output stochastic, creating a regularizing effect. Appropriate weight initialization (Glorot and Bengio, 2010) and gradient optimization techniques (e.g. Adam (Kingma and Ba, 2014)) also influence the overall performance. Combining ensembles of 5-10 models initialized with different random seeds usually leads to a substantial increase in performance.

2.1.3 Attention

In standard many-to-one RNNs, the last hidden state is most often used as input to a downstream network. However, when certain subparts of the input sequence contain most of the predictive information, this approach is suboptimal. Here, it will be beneficial to add attention weighting (Bahdanau et al., 2014; Jaderberg et al., 2015) (as illustrated in Figure 2B). Attention is implemented by learning a context dependent normalized weight for each hidden state of the input sequence. The input to the downstream network is then the weighted average over all hidden states. This normalized weight is computed in two steps: 1) a one hidden layer dense network takes each hidden state of the RNN as input and 2) passes these network outputs through a softmax function. In one of the examples shown later, we use such attention to visualize and identify parts of a protein sequence that are important for subcellular localization prediction.

2.1.4 Pooling

Max-pooling provides a way of reducing the input or hidden layer size by selecting only the maximally activated neuron from a number of neighboring neurons. Alternatively mean pooling can be performed where the mean activation is calculated among neighboring neurons. Pooling is often used in convolutional neural networks (LeCun et al., 2015). To make a convolutional neural network independent of the input sequence length, global max-pooling can be applied where only the maximally activated neuron of the input or hidden layer is selected for each of the convolutional filters. In this way, the number of hidden neurons generated by the convolutional filters is equal to the number of filters and not influenced by the input sequence length.

3 Example biological applications

To demonstrate the usefulness of the CNN and LSTM neural network architectures for machine learning on biological data, we have applied the framework to three important biological problems: subcellular localization, protein secondary structure and peptide binding to MHC Class II (MHCII) molecules. The details of the data, models and training and testing procedure are described in the supplementary materials. Note, however that not all problems are equally suited for analyses using deep neural network methods (the limiting factor in most cases being the amount of data available), and that definition of optimal network architectures (guided by biological insights) and training protocols is as important for a successful application as the choice of deep learning method. An in-depth coverage of these themes is however beyond the scope of this manuscript. However, a generally good starting architecture for machine learning on biological sequence data has an input-to-output structure as shown in Figure 2B: Encoded seq-to-CNN-to-biLSTM-to-Attention-to-Dense-to-Output. Dependent upon the problem at hand, one may omit architecture elements or introduce skip connections as shown in Figure 3B. In the online material, we have included Jupyter Notebook implementations of different models starting from a simple dense feed forward model up to a full model including CNN, LSTM and attention to illustrate the contribution of each architectural element. This is discussed in more detail below.

3.1 Subcellular localization
In eukaryotes, proteins are either secreted from the cell or sorted into numerous different cellular sub-compartments such as e.g. nucleus, cytoplasm and Golgi (see Figure 2A). For some locations, the proteins are known to contain specific sequence motifs, where the best understood signal is the N-terminal signal peptide in proteins belonging to the secretory pathway. However, other subcellular compartments are also known to contain weaker signals such as the nuclear localization signal in nuclear proteins, membrane crossing alpha-helices in membrane proteins or KDEL-type retention signals for endoplasmic reticulum proteins (Lodish et al., 2016).

![Image of subcellular localization classification and network architecture](image)

**Fig. 2.** A) Schematic illustration of subcellular localization classification. B) The neural network architecture used to predict the subcellular localization of proteins. C) Visualization of the positions within the protein amino acid sequence that have high importance for the prediction of subcellular localization. Sequence position importance is determined by an attention function and the middle part of the protein sequences have been cut out in order to align N- and C-terminus. The different subcellular localization classes are shown on the y-axis. D) Table of the A-LSTM performance compared to the state of the art sequence driven SVM prediction method MultiLoc. E) Visualization of convolutional filter. For this filter charged amino acids will suppress the output (blue, red) while hydrophobic amino acids will increase the output (black). C) and D) are adapted from (Sønderby et al., 2015).

The task of predicting protein sorting or subcellular localization has attracted large interest in the bioinformatics field (Emanuelsson et al., 2007) and can be generally stated as a sequence to multi-class prediction problem. The state of the art data driven approach for prediction of subcellular location of proteins is MultiLoc (Höglund et al., 2006), which is based on support vector machines (SVM) taking into account N-terminal targeting regions, amino acid composition and protein sequence motifs. The SherLoc2 method (Briesemeister et al., 2009) combines the sequence-based MultiLoc method with text mining approaches and the inclusion of Gene Ontology (GO) terms to improve performance.

We have implemented four models (see online Jupyter Notebooks) starting from a simple feed forward (FFN) dense model up to the full model including CNN, LSTM and attention architecture visualized in Figure 2B and called A-LSTM in the following. We apply these models to a reduced version of the subcellular localization dataset to illustrate the effect of hyper parameter choices and network architecture on predictive performance and running times. In this example, the model with the lowest performance was FFN, with a classification accuracy fraction (Acc) of 0.757. The CNN and CNN-LSTM models followed with an accuracy of 0.783 and 0.814, respectively. Finally, the model with the highest accuracy was the A-LSTM with Acc=0.830. If we evaluate the performance for the different prediction classes for the four models, we do not see large differences for the classes characterized with many proteins (data not shown). However, the A-LSTM demonstrates improved performance for classes characterized with very few examples, such as ER, Golgi apparatus, lysosome and vacuole. The main drawback of the models using LSTM layers compare to the FFN and CNN is the training time. On average, LSTM models take 10 to 12 seconds per epoch, whereas FFN takes 0.3 seconds and CNN takes 2.5 seconds. This could be a limiting factor with more data or a more complex architecture.

In the remainder of the section, we discuss the findings for A-LSTM trained on the entire MultiLoc dataset.

In the subcellular localization problem, the position of the sorting signal is often of interest to the biologists. LSTM neural networks combined with an attention function are ideally suited to solve this task. The model presented here is a modified and improved version of the one presented by Sonderby et al. (2015). The architecture of our A-LSTM neural network is visualized in Figure 2B and is based on a convolutional layer followed by a bi-directional LSTM. Additionally, we applied an attention function after the LSTM allowing us to assign an importance to each position in a protein sequence with regards to subcellular location. Here, we calculate the activations of the LSTM memory blocks (hidden state) for each position in the protein sequence, and apply the attention function to determine the importance of each hidden state (giving an indication of the importance of the underlying position in the input sequence). Next, we calculate the attention weighted sum of all hidden states that are used as input to a dense feed forward neural network with one hidden layer and softmax output to obtain the final prediction of subcellular localization. All models were trained on the same dataset as the MultiLoc method described above.

The reasoning behind this choice of architecture is based upon the following idealized flow of information from discriminative subsequences to classification: The CNN filters deal with the issue of sequence alignment, and provide a learned filter-bank sensitive to specific sequence motives. When such a motif is present somewhere in the sequence, it will lead to an increased absolute input to the LSTM at that position. The biLSTM layers will integrate this information locally both forward and backward in the sequence and the attention function uses its learned context awareness to pass the discriminatory information from these RNN hidden states on to the dense layer and then to the softmax classification.

A-LSTM achieved an accuracy of Acc=0.914, outperforming the MultiLoc method which had an Acc=0.767. SherLoc2 outperformed our models with an Acc=0.93. However, SherLoc2 is not purely sequence data driven but also includes text-mining approaches, and we hypothesize extending our A-LSTM ensemble in that way would improve performance further.

In Figure 2C, we illustrate where in the sequence the A-LSTM model assigns weight for proteins belonging to different subcellular compartments. Sequences from the compartments ER, extracellular, lysosomal, and vacuolar have clearly marked N-terminal signal peptides. Chloroplast and mitochondrial proteins also have N-terminal sorting signals. For the plasma membrane category, we observe that some proteins have signal peptides, while the model generally focuses on sections, presumably transmembrane helices, scattered across the rest of the protein sequence with some overabundance close to the C-terminus. Cytoplasmic
and nuclear proteins do not have N-terminal sorting signals, and we see that the attention is scattered over a broader region of the sequences. Figure 2D gives an example of a convolutional filter that prefers hydrophobic amino acids.

3.2 Secondary structure prediction

After translation, proteins fold into a 3-dimensional structure, known as the tertiary structure (Figure 3A). Knowledge of a protein’s structure can help to understand its function. Therefore de novo prediction of protein structure from sequence is a problem of great biological interest (Dill and MacCallum, 2012). An important step in the prediction of tertiary protein structure is the prediction of the secondary structure, the local conformations of the peptide backbone. There are three main classes of secondary structure: alpha-helix, beta-strand and coil. These can be further divided into 8 classes (Kabsch and Sander, 1983) (3 sub-classes of helix, 2 sub-classes of strands and 3 sub-classes of coil).

Secondary structure prediction is a many-to-many problem or sequence-tagging problem because we predict the secondary structure class for each amino acid in the protein sequence. The state of the art sequence-based method to solve this problem is DeepCNF (Wang et al., 2016) which is based on a combination of de novo convolutional neural networks and conditional neural fields.

The architecture of our neural network to predict secondary is shown in Figure 3B and is a combination of convolutional and LSTM neural networks (Sondorby and Winther, 2014). The attention step is omitted here because the information in input and output to a high degree is co-located. Using either a CNN layer and/or a skip layer makes no large difference for this application. Here, we report the CNN plus skip results. An ensemble of neural networks was trained on the same data as DeepCNF. The performance of our network ensemble is shown in Figure 3C and reaches a Q8 accuracy of 0.702, thus outperforming DeepCNF.

3.3 Predicting peptides binding to MHC Class II molecules

MHC class II molecules (MHCII) are an essential part of the adaptive immune system and involved in the detection of extracellular pathogens. They present peptides derived from proteins of the extracellular environment to T-helper cells (see Figure 4A). A peptide can only be recognized by a T cell, i.e. be a T cell epitope, if it is able to bind an MHCII molecule expressed by the host (Roche and Furuta, 2015). When interested in predicting an immune response to pathogens, the prediction of peptides binding to MHCII molecules is an important step, since only a small fraction of all peptides are able to bind to MHCII molecules (Castellino et al., 1997). Peptides binding to MHCII molecules can differ greatly in length greatly since the binding cleft of the MHCII molecules is open on both sides, allowing the peptide to extend beyond the binding cleft. Further, the binding core of a peptide, which has a length of approximately 9 amino acids, can be located anywhere within the peptide (Nielsen et al., 2010).

The state of the art method for predicting peptide binding to MHCII molecules is NetMHCIIpan-3.0 (Karosiene et al., 2013), which relies on an ensemble of feed forward neural networks trained by an algorithm tailored to the problem (Andreatta et al., 2011; Nielsen and Lund, 2009).

Here, we trained an ensemble of CNNs and LSTMs to predict peptide binding to MHCII molecules. The LSTM is uni-directional and attention is omitted because the peptide sequences are so short. The neural network architectures are visualized in Figure 4B. All models were trained on the data set and data set partitions used to develop the NetMHCIIpan method. An ensemble of CNNs and LSTMs achieved performance on par with NetMHCIIpan-3.0 on both testing and evaluation data. Further, combining feed forward nets trained in the NetMHCIIpan fashion with CNN and LSTM networks to form a consensus method significantly improved performance. Figure 4C shows the performance of the consensus method for individual MHCII molecules in the evaluation set.

4 Discussion

The regained interest in deep neural network architectures within machine learning fields such as image and speech recognition is only slowly spreading into biology and bioinformatics. One potential cause of this is the lack of examples or code templates tailored to bioinformatics problems combined with the notion that the implementation and training of deep learning methods is complicated and computationally challenging. Here, we have aimed to help overcome these barriers by demonstrating...
how simple code can easily and effectively be developed to train CNN and LSTM neural network models to predict properties of important biological problems. We have done this for three distinct biological problems: predictions of subcellular localization, secondary structure and peptide binding to MHCII molecules, and in all cases demonstrated state of the art performance.

We have performed all model development and evaluation using the Lasagne library. However, our findings are general and we expect comparable results could have been obtained using any of the currently available frameworks for deep learning.

The applications we have chosen are all described by protein data. It is however clear that the framework is equally suited to work on any type of sequence data, for example nucleotide sequences (Leung et al., 2014). Others have also applied similar models to images of biological or medical relevance (Cha et al., 2016; Moeskops et al., 2016).

Funding
This work has been supported by the the National Institute of Allergy and Infectious Diseases, National Institutes of Health, Department of Health and Human Services under Contracts HHSN272201200010C. We thank the NVIDIA Corporation for the donation of Titan X GPUs.

Conflict of Interest: none declared.

References