Protein's Solvent Accessibility Prediction using Convolutional Neural Network 1D

Fabrizio Di Guardo

University of Florence
Machine Learning — Paolo Frasconi

UNIVERSITÀ
DEGLI STUDI
FIRENZE

fabrizio.diguardo@stud.unifi.it

12 June 2018

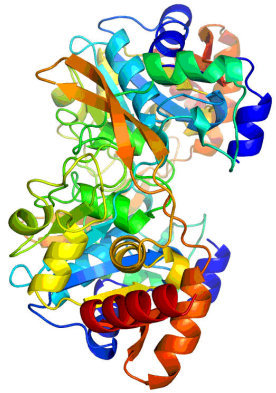
Summary

Importance of protein structures prediction

- In *bioinformatics* the **3D structure** of proteins can be determined by their **one-dimensional** sequences of amino acid residues.

A challenging problem

Prediction 3D structures from 1D sequences is difficult because it demands an efficient technique to search in a very large conformational space.



- We need to divide the prediction into many **smaller problems**.
- Common predicted proprieties are *secondary structure* and *solvent accessibility*.

In this project: focus on the prediction of protein's *relative Solvent Accessibility* from the Amino acid residues and Secondary Structure.

State of art

Common methods for prediction of structural properties of proteins are:

- **Support Vector Machine**[1]
- Bidirectional **Long Short Term Memory**[2] recurrent neural networks
- **Convolutional Neural Network** in conjunction with a LSTM[3]

In this project

In this study is used a **Convolutional Neural Network 1D** in conjunction with some *optimization* techniques.
The result is then compared with the network built in [3].

---

[1] Yuan, Zheng, Kevin Burrage, and John S. Mattick. *Prediction of protein solvent accessibility using support vector machines*, **2002**.

[2] S.K.Snderby, O. Winther. *Protein secondary structure prediction with long short term memory networks*. arXiv preprint arXiv:1412.7828, **2014**.

[3] A. R. Johansen, S.K.Snderby, O. Winther, *Protein and secondary structure prediction with convolutions and vertical-bi-directional rnns*, DTU, **2016**

Preparing Dataset 1/2

Features extraction

The starting dataset (*PISCES CullPDB*) is a set of 8000 **PDB** files from which the features of interest (*protein residues*, *secondary structure* and *solvent accessibility*) have been extracted using the **DSSP** algorithm.
This process is done with a *Python* script developed in Linux environment.

- **PDB**: the Protein Data Bank is the worldwide archive of structural data of biological macromolecules, such as proteins and nucleic acids.
- **DSSP**: standard *algorithm* for assigning secondary structure and accessibility to the amino acids of a protein, given the atomic-resolution coordinates of the protein.

Preparing Dataset 2/2

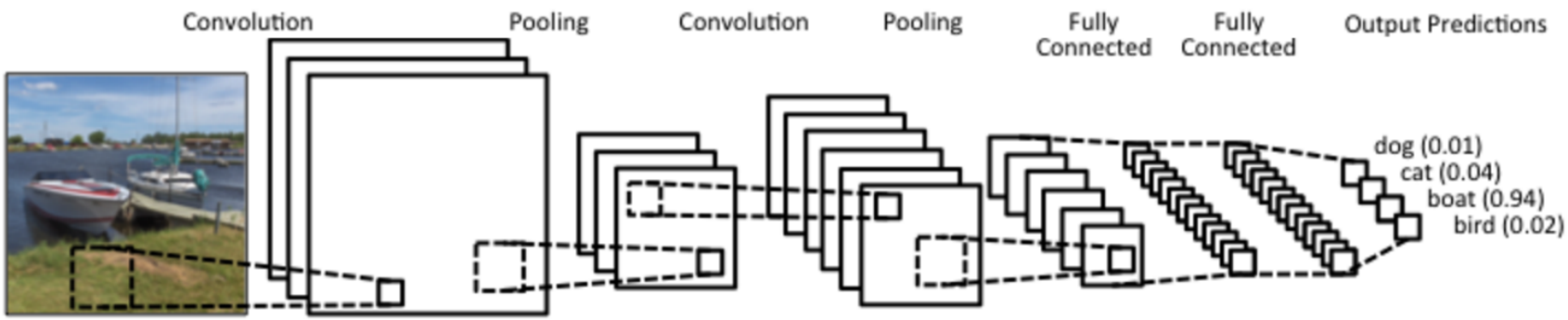
Dataset built

Proteins with less than **50** and more than **700** amino acids were discarded. The final dataset consist in **6397 samples** (proteins) for a 3D array of shape: [6397, 700, 31]. Features are distributed in this way:
22 for *amino acid residues*, 8 for *secondary structure* and 1 for *accessibility*.

Padding

Keras, the framework used for classification, does not allow an array of features of a different shape in input into a CNN. So it was necessary to insert **padding** in case of less than 700 amino acids, with a consequent change in the calculation of the loss and **accuracy**.

Convolutional Neural Network



How it works

A CNN is a **feed-forward** artificial neural network in witch each layer emulates the response of an individual neuron to visual stimuli.

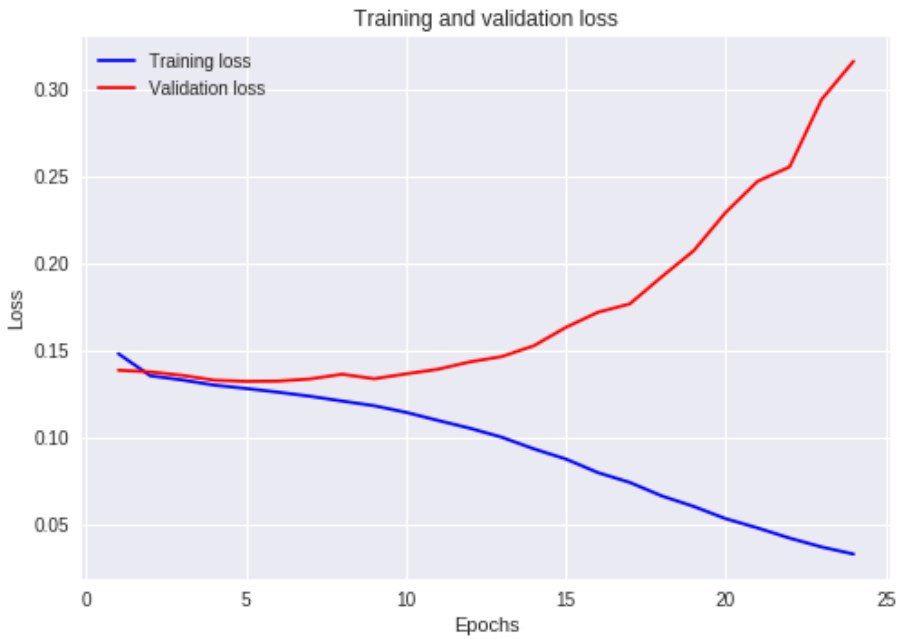
- **1D** or **2D** data is disposed in grid
- Each input connected to some neurons of the next **hidden layer**
- The idea is look for the **same feature** in all the given input
- **Shared weights**: neurons looking for the same feature have to learn to calculate the same function, so they have the same weights.
- Typically after a convolutional layer there is a **pooling** layer.

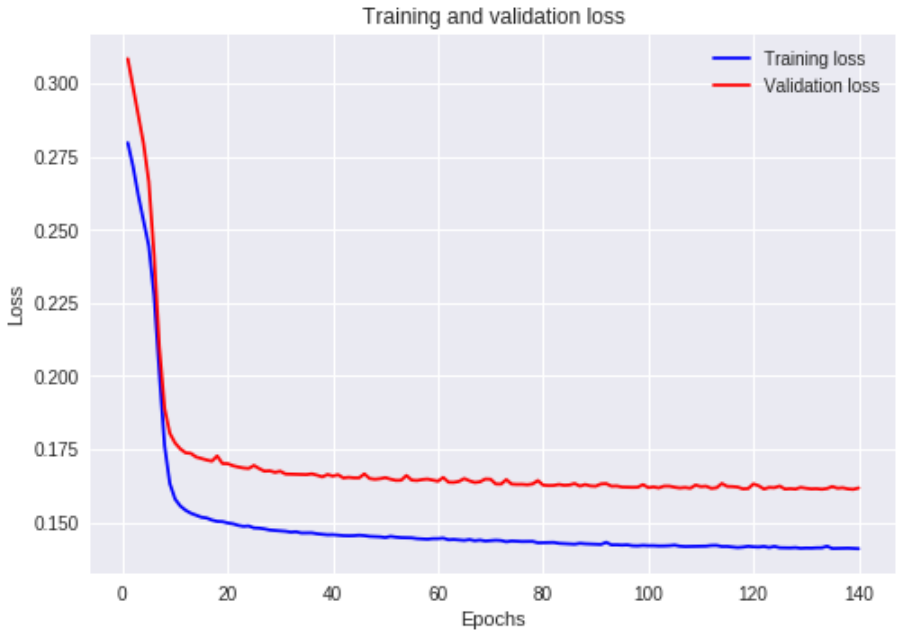
Model generalization problems

Fitting problems

During the network training have occurred two "fitting" problems:

- **Overfitting**: good with the trainset, bad with other data.
- **Underfitting**: bad with the trainset, bad with other data.





To solve these problems classic techniques have been used:

- **Adjustment of size** of hidden layers, **Max pooling** and **Dropout**

Generalizing the model

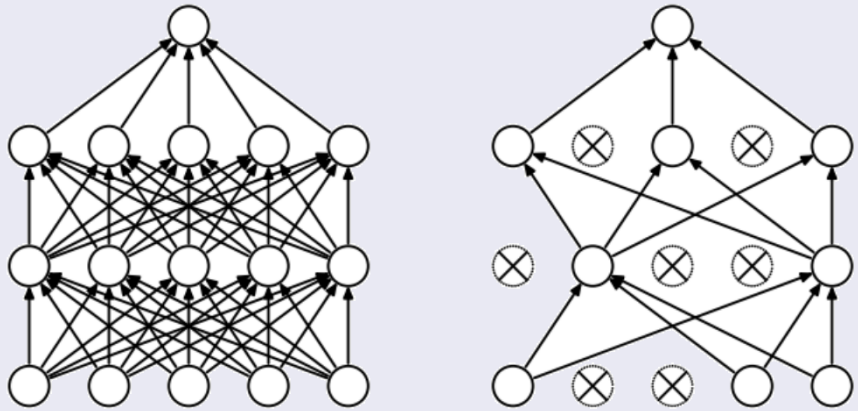
Max pooling 1D

Non-linear **downsampling** of an input tensor X partitioned into 1D *subtensors* along a dimension and transformed to the output tensor Y by replacing each subtensor with its **maximum element**.

Dropout

Regularization technique where randomly selected neurons are ignored during training, **removing their contribution to the activation**.

$$a_k = \sum_l w_{kl}\phi_l\delta_l, \ \ \delta = Bernulli(p)$$



(a) Standard Neural Net

(b) After applying dropout.

The Loss function: Cross Entropy

The **loss function** is a function that gives the quantity that will be **minimized** during training and represents a measure of a "cost".

Cross Entropy

In this projects is used the **Cross Entropy** loss function:

$$-\sum_{i=1} y_i \cdot \log \widetilde{y}_i$$

where y is the tensor of true targets and $\widetilde{y}$ is the tensor of predicted targets (probabilities).

The values $\widetilde{y}_i$ are obtained with a **softmax** activation at the last layer.

Softmax issue in Keras

The implementation of softmax includes the calculation of $exp(x)$, but it may be too high to be interpretable from Python, which may return *nan*. So a **stabler version** of softmax is required using a scalar C like $max(x)$:

$$\sigma_i(x) = \frac{e^{x_i}}{\sum_{k=1}^{N} e^{x_k}} \Rightarrow \frac{Ce^{x_i}}{C\sum_{k=1}^{N} e^{x_k}} \Rightarrow \frac{e^{x_i+log(C)}}{\sum_{k=1}^{N} e^{x_k+log(C)}}$$

Metric and Optimizer used

Masked Accuracy

Cause the padding introduced for handling *varied length* of amino-acids sequences, the accuracy function needs to consider only the predictions on the not padded elements.

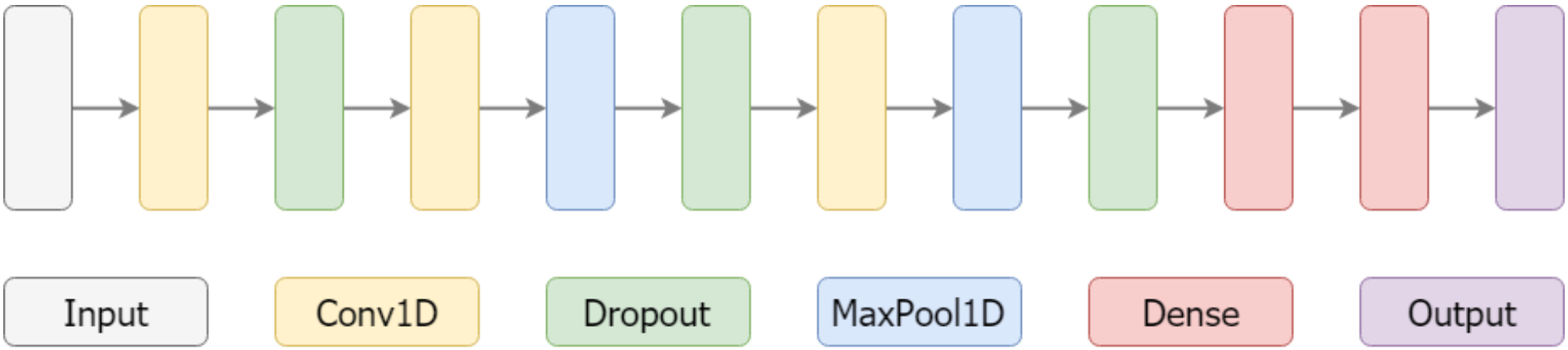
$$Accuracy = \frac{\sum_n (\arg\max(y_n) == \arg\max(\widetilde{y}_n)) \times sum(y_n)}{\sum_n y_n}$$

where $\widetilde{y}$ is the predicted label, and y is the true label.

Adam Optimizator

Adam (2014) is different from *sgd*: don't maintains a single learning rate for all weight updates, **adapting the lr** based on the gradients. It combines the advantages of two other extensions of sgd: *AdaGrad* and *RMSProp*.
They have been tried other optimizers (SGD, RMSProp, AdaDelta, etc), but Adam gives the best performance in terms of *loss minimization*.

Network structure



- Three *convolutional layers* followed by two *fully connected* layers.
- *Dropout* and *Max-pooling* are applied after the Conv1D.
- After the first convolutional layer there is no max-pooling.
- There are three Conv1D because empirically has been observed that:
  - with two layers the network doesn't learn
  - with more layers than three the loss doesn't increase much, but execution time increases
- The hyper-parameters of the layers have been selected after a *grid search*.

Grid search for hyperparameters selection

Next step is a phase of grid search for **hyperparameters optimization** to find the **best configuration** that would lead to a loss as low as possible. The networks were trained for **120** epochs.

Hyperparameter	Values tested
Dropout's rate	[0, 0.1, 0.25, 0.5]
Activation functions	['relu','sigmoid','tanh']
Conv layers filters	[16, 32, 64, 128]
Conv layers kernel size	[3,5,7]
Max pooling	['True','False']
Dense units	[16, 32, 64, 128]

for a total of **1152** models, using *Google Colab* platform (so using a **GPU**), in an average execution time of about **430 seconds** (per model). It was used a **Early-Stopping** condition (*patience* 5) for loss control.

The **6397** samples have been divided in:

- **5697** for *trainset*, with **697** for *validationset* and **700** for *test set*

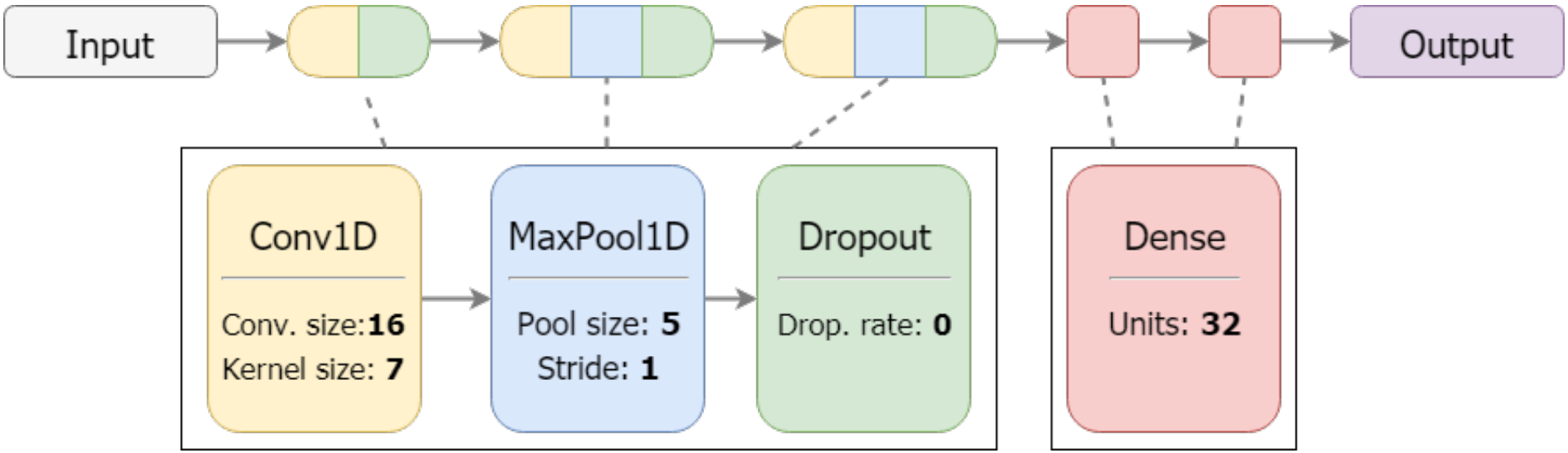
Grid search: best models

After the the best hyperparameters search, have been selected the five *best models*, based on loss and accuracy:

	Loss	**Accuracy**	**Model**
1	**0.0938**	0.8532	tanh, filters 16, kernel: 7, drop: 0, dense:32
2	0.0945	**0.8546**	relu , filters 16, kernel: 7, drop: 0, dense:32
3	0.0959	0.8478	tanh, filters 32, kernel: 5, drop: 0, dense:16
4	0.0960	0.8485	tanh, filters 16, kernel: 5, drop: 0, dense:16
5	0.9701	0.8515	tanh, filters 32, kernel: 7, drop: 0, dense:32

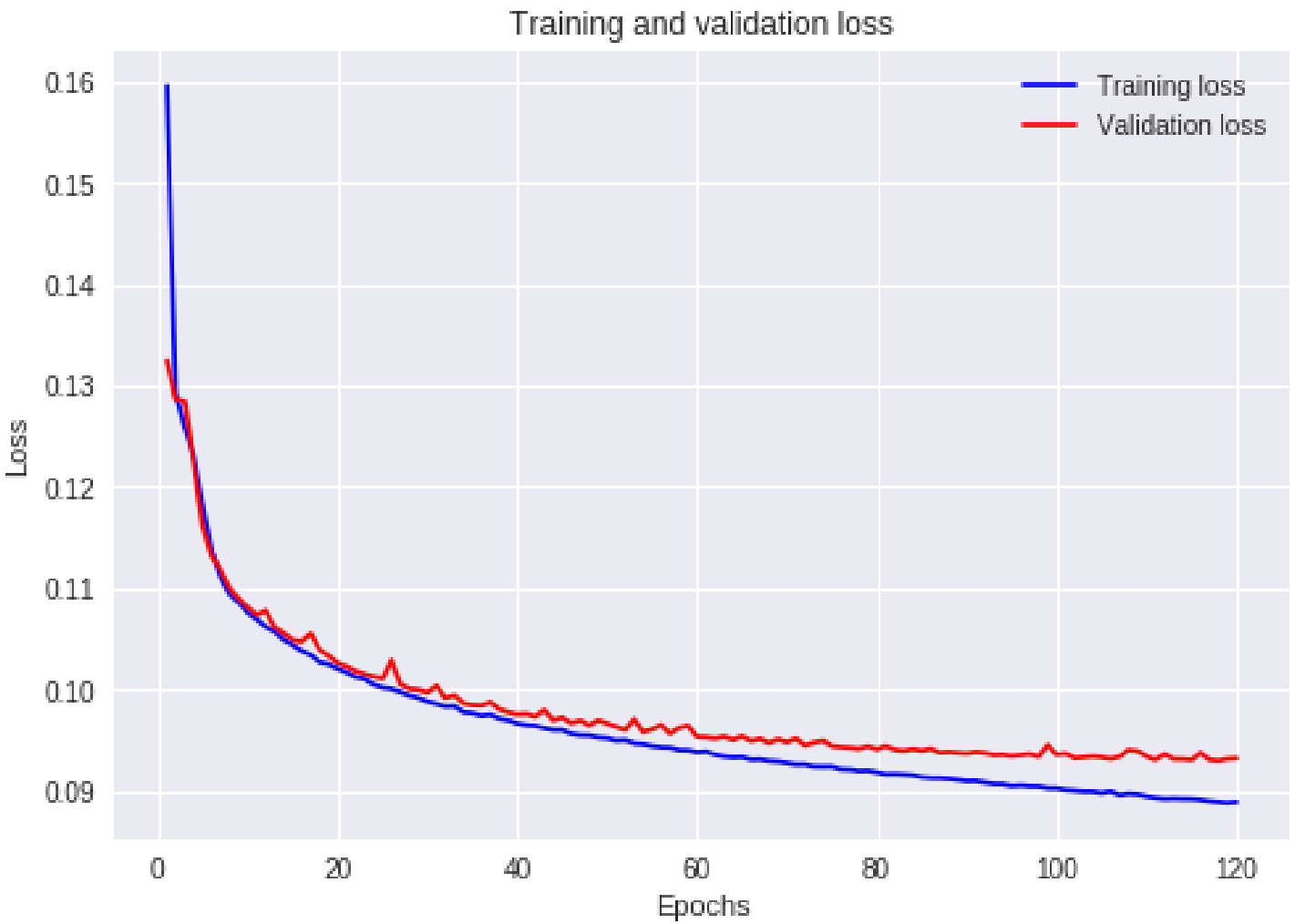
So for *120* epochs the best loss reached is 0.0938 and the best accuracy about 85%. *Tanh* results as the best activation.
Note that all the best models were with Max Pooling.

The final model



The final model, trained in **120** epochs, gives an accuracy of **85.2%** and a loss of **0.0913** evaluated with the *test set*.

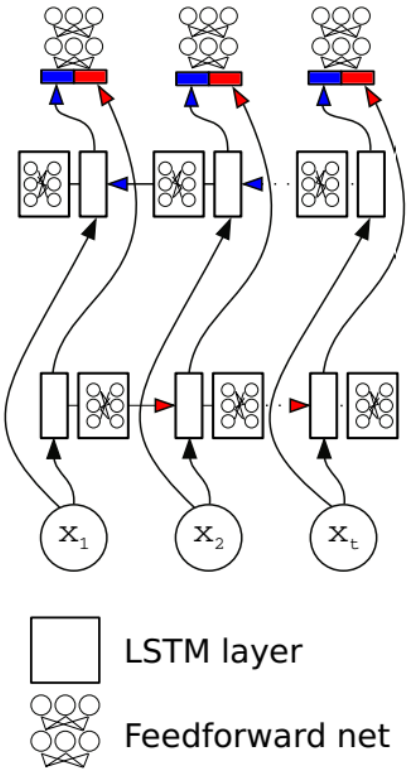
The final model - Loss Trend



Analysing the loss trend of the final model, we can say that **80 epochs** is a good compromise between *performance* and *execution time*.

Comparison with LSTM

Since at state of art the problem of *protein structure prediction* is dealt principally with a **Recurrent Neural Network**, as a final step of this project the network built in [1] was reproduced to compare the results.

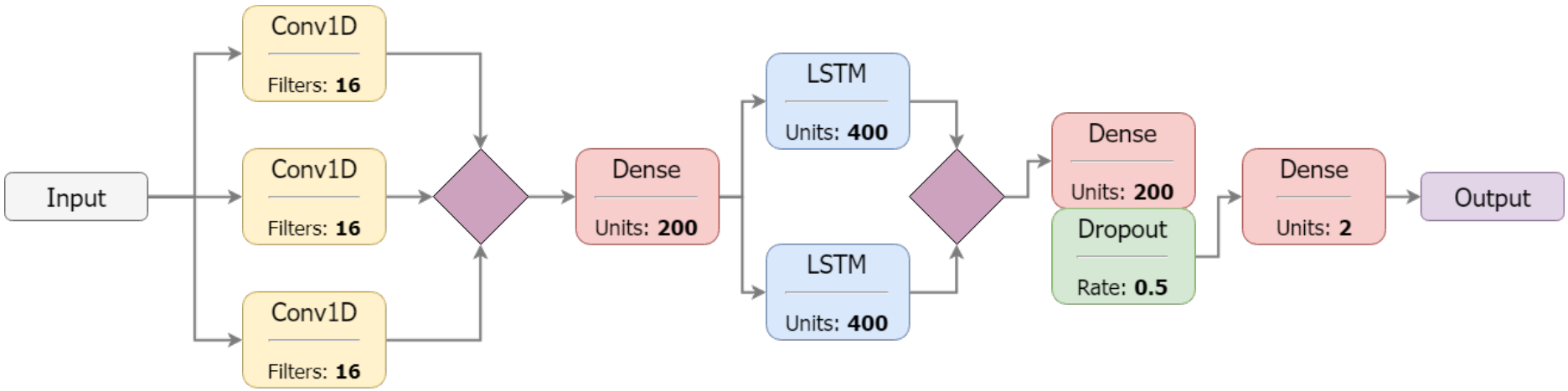


A **Bidirectional LSTM** network can be used to learn longterm dependencies. The forward LSTM (red) starts at time 1 and the backwards LSTM (blue) starts at time n, then they go **forwards** and **backwards** respectively and errors are combined using a feedforward net.

[1] A. R. Johansen, S.K.Snderby, O. Winther, *Protein and secondary structure prediction with convolutions and vertical-bi-directional rnns*, DTU, **2016**

CNN and Bidirectional LSTM Network

In the mentioned article was built a model with the combination of three concatenated **Convolutional** Layers and two **LSTM** Layers: one *forward* an another *backward*. After each concatenation block there is a **Dense** Layer and in the second one is applied a **Dropout** with a rate of **0.5**. It used **AdaDelta** as optimizer and *cross entropy* as loss function.



In the article this model was used for secondary structure prediction from proteins residues and profiles. In this project has been used for predicting *solvent accessibility* from the amino acids and secondary structure of proteins. It has been used the **same dataset** of the CNN.

Results

The CNN+LSTM Model was trained for **120 epochs**, as the only CNN version, and evaluated in testset. From the loss trend analysis, it may be deduced that the CNN+LSTM model is *yet improvable*, but yet after 120 epochs it gives a very better accuracy and loss.

	Convolution + LSTM	**Only Convolution**
Loss	**0.0793**	0.0938
Accuracy	**0.897**	0.854
Time	$\sim$ 6 hours	$\sim$ **5 min.**
# parameters	2100850	7602

However the Bidirectional LSTM requires a lot of time to be trained, reaching about 6 hours for 120 epochs vs 5:34 minutes of the only CNN model.

Conclusions

- It has been studied the proteins **solvent accessibility prediction**
- The dataset has required a **preprocessing** phase for extract the necessary features
- It has been created a Convolutional Neural Network 1D
  - Configured with a **grid search** of hyper-parameters
  - Regularized with **Dropout** and **Max Pooling** techniques
- The best model reach an accuracy of about **85.5%**
- The result has been compared with a **combing of CNN and LSTM**, for having the advantages of the RNN
- It has been found that with a RNN **the performance is better** than the only CNN in terms of loss and accuracy, even if it required **a lot of time** for the training step.