Analyzing protein sequences with Deep Neural Networks

Rasmus Porse Bjørneskov
Roald Frej Vitus Simonsen

June 6, 2019

Abstract

Her skriver vi på dansk.

Contents

1 Introduction **1**
1.1 Motivation 1
1.2 Proteins 1
1.2.1 What is 1
1.2.2 Secondary Structures 1
1.2.3 Solvent accessible surface area . 1
1.3 Neural networks 1
1.3.1 Basics 2
1.3.2 Training 2
1.3.3 Convolutional neural networks . 2
1.3.4 Multitask learning 2
1.4 Prior research in this field 2

2 Methods **3**
2.1 Dataset 3
2.1.1 Ophav 3
2.1.2 Features 3
2.1.3 Opdeling i træning, test og validering 3
2.2 Tools 3
2.2.1 Math 3
2.2.2 Tech 3
2.3 Predicting secondary structures alone . 3
2.3.1 Beregning af præcision 3
2.4 Implementing multitask learning 3
2.4.1 Generelt om Multi-task learning 3
2.4.2 Opbygningen af vores model . . 3
2.4.3 Træning 3
2.4.4 Beregning af præcision 3
2.5 Hyperparametre 3

3 Results **3**
3.1 Hyperparameter-optimering 3
3.1.1 Antal lag 3
3.1.2 Learning rate 3
3.1.3 Antal neuroner i hvert lag 3
3.1.4 Kernel size 3
3.2 Final predictive capabilities 4
3.3 Comparison 4

4 Conclusion **4**

5 Appendix **4**

1 Introduction

1.1 Motivation

Neurale netværk er rigtigt seje [1]

1.2 Proteins

1.2.1 What is

1.2.2 Secondary Structures

1.2.3 Solvent accessible surface area

1.3 Neural networks

The common perception is that humans and animals process information, i.e. transform perceptional stimuli and physiological conditions into behaviour, by using their brains. This is imprecise however, as the brain as such is only one functioning part of what is called the *nervous system*, which is in turn responsible for the internal workings of human and animal behaviour.

This nervous system is an abstaction over a number of *neurons* interconnected by synapses. Neurons in turn are so-called electrically exitable cells. In a gross over-simplification, this can be translated into the case that each neuron can have different internal states, depending on the internal states of the neurons it is connected to, thus forming a *neural network*.

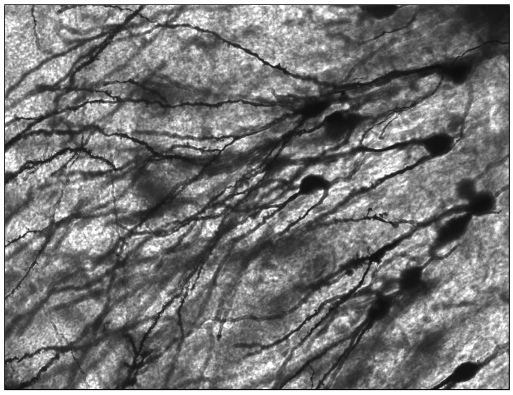

Figure 1: Neurons in the dentate gyrus of an epilepsy patient.

While obviously interesting within the fields of biology or psychology, this structure has shown to be enourmously interesting in the field of computation, as one can in fact model this very thing and use it to

make predictions based on prior observations, for example in regards to the aforementioned structures of proteins folding.

Analogously, or rather, digitally, we can use this model to construct an *artificial neural network*. These set themselves apart from biological neural networks in a couple of ways: most urgently in that while neurons in biological neural networks are connected to each other via synapses, so that each neuron has an either inhibitory or activating effect on whether or not a connected neuron 'fires', in artificial neural networks neurons are connected by *edges*, each with an associated *weight*, allowing the artificial networks to leave the discrete domain and enter the continuous.

1.3.1 Basics

An artificial neural network is a specific instantiation of a *multi-layer perceptron*. A multi-layer perceptron in turn is a system of nodes arranged into layers and each receiving their value from the value of the nodes in the layer before them (adjusted by some weight) in combination with a bias (a scalar) and an activation function of some sort.
In this way the first layer will be called the *input layer*, the last layer the *output layer* and all layers in between *hidden layers*.
In the case of neural networks, the nodes in the system are referred to as neurons. Thus, if one was to describe it more precisely, the value z_i of neuron layer i with d number of neurons in it, given the matrix of sets of weights w (the biases being the very first row) and the activation function $h()$ can be expressed as follows:

$$z_i = h\left(\sum_{j=0}^{d} w_{ij} \cdot z_{i-1} + w_{i0}\right)$$

Such a model is useful for calculating continuous values as well as for classification purposes. A common example of the latter is the MNIST data set of handwritten digits in 28x28 pixel greyscale images. In this case the light intensity of the 784 (28^2) pixels could fittingly be the input layer, while a layer consisting of 10 nodes (corresponding to digits 0 through 9) could be the output layer, such that the system could be used to predict which digit is written in a given image. A common implementation of a neural network to perform this task is to have a single hidden layer of 800 nodes, however the most successful implementations in regards to the MNIST data set have been made with *convolutional neural networks*, but we shall return to these later.

Figure 2: Handwritten digits in the MNIST data set.

1.3.2 Training

Simply having defined the system of the neural network as above (denoted f_{NN}) naturally does not give us a reliable method for classifying, as the accuracy of the prediction inherently depends on the correctness of the weights and biases of the system (denoted Θ), which must be initialized with random values.
Enter a rather nifty idea named *backpropagation*. Having a set of predicted values ($\hat{y}$) for a data set (x) along with the observed correct values (y), one can establish a *loss* by applying a loss function (that we shall elaborate on later), but which can be as straightforward as a root mean square error. We shall denote this loss function $l(\hat{y}, y)$, so that the total loss of the system when applied to data set x is $l(f_{NN}(\Theta, x), \hat{y})$.
Now backpropagation refers to the practice of essentially taking the derivative of the loss function in regards to the matrix of weights and biases, thereby calculating a *gradient* $\nabla l(f_{NN}(\Theta, x), \hat{y})$ of these values with respect to the accuracy of the system, and then adjusting the weights and biases by the gradient multiplied by a learning rate scalar.
In laymans terms, by doing this one gets a matrix of 'change your weights by this and that much in this and that direction to increase prediction accuracy'.

1.3.3 Convolutional neural networks

Before talking about convolutional neural networks, we have to specify what a convolution is. Again the MNIST data set serves as an obvious example.
One can understand each of the images in the data set as a matrix of 26×26 values in the range of 0 - 255. The formal definition of a convolution is *TODO*, however the more practical explanation is that it is the resulting matrix of the product of a matrix and a filter. For example if one has the matrix a and the filter f:

$$a = \begin{bmatrix} 2 & 3 & 4 \\ 1 & 2 & 3 \\ 0 & 1 & 2 \end{bmatrix} \qquad f = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

the result of convolving a over f would be:

1.3.4 Multitask learning

1.4 Prior research in this field

Skrive noget om den artikel vi tager udgangspunkt i.

2 Methods

2.1 Dataset

2.1.1 Ophav

Hvor har vi dette datasæt fra, og hvem har lavet det?

2.1.2 Features

Skrive noget om hvad vores datasæt er og hvordan det er sat op. Dvs. hvilke egenskaber er der, hvordan er de encoded (one-hot vs. binært (solvent)) og hvorvidt vi bruger dem til noget. Fortæl også om at strukturerne her er encoded som de 8 substrukturer og ikke de 3 grupperinger af strukturer.

2.1.3 Opdeling i træning, test og validering

Fortæl om at vi har rigtigt meget træningsdata og hhv. 255 og 236 værdier i test og validering. Vi træner og backpropagerer over træningssættet, holder øje med valideringssættet (og træffer vores beslutninger ud fra det) og udsætter kun modellen for testsættet til sidst.

2.2 Tools

2.2.1 Math

Gennemgå at vi bruger konvolutioner, Binary Cross Entropy loss, ReLU og Softmax (formler for de tre). ((overvej om det skal i et andet afsnit))

2.2.2 Tech

Skrive om pyTorch - herunder dets værktøjer til automatisk at skabe lag af neuroner samt backpropagation.

Derudover laver vi vores præcisionsberegninger i numpy.

Vores træning er lavet i jupyter notebooks på en Dell-computer med en i7 processor, 16GB RAM og en Nvidia GeForce GTX 1050 GPU med 4GB RAM.

2.3 Predicting secondary structures alone

Fortælle om hvordan vi byggede vores single-model (convolutionelle lag, ReLU, softmax, BCELoss, padding).

2.3.1 Beregning af præcision

Forklar at det vi tæller er antallet af korrekte forudsigelser over antal af faktiske aminosyrer. Gennemgå herunder matematikken i at vi kollapser fra one-hot til classification, og så kører vores mask-magi på det.

2.4 Implementing multitask learning

2.4.1 Generelt om Multi-task learning

Start med refleksioner over hvordan vi bruger shared-parameters i vores model, og forklar at vi stadig kører udelukkende konvolutionelle lag igennem modellen.

2.4.2 Opbygningen af vores model

Fortæl om opbygningen af vores model - dvs. at vi ReLU'er hele vejen igennem, undtagen sidste lag, og så hvordan vi splitter dataen ud for så at ReLU'e og softmax'e sekundærstrukturerne, medens at vi afrunder solvent-egenskaberne.

2.4.3 Træning

Fortæl hvordan vi udregner loss på vores model, dvs. at vi udregner tre losses; sekundærstruktur, relativ solvency og absolut solvency. Vi bruger BCE på dem alle sammen, så redegør hvorfor vi godt kan det på solvent også.

2.4.4 Beregning af præcision

Forklar hvordan vi til strukturerne bruger samme beregning som ovenfor, mens vi gør næsten det samme for solvent egenskaberne (men hver for sig).

2.5 Hyperparametre

Fortæl at vi i projektet vil prøve at iterere over forskellige hhv. lagstørrelser, antal lag, kernelsizes, og learning rate. Forklar at vi vil starte med at tage udgangspunkt i de værdier som Xi bruger, og så iterere lidt frem og tilbage over dem.

3 Results

3.1 Hyperparameter-optimering

Gentag at vi tog udgangspunkt i værdier fra Xi og andre.

3.1.1 Antal lag

Fortæl om lag, lav en tabel og en graf.

3.1.2 Learning rate

Fortæl om LR, lav en tabel og en graf.

3.1.3 Antal neuroner i hvert lag

Fortæl om det, lav en tabel og en graf.

3.1.4 Kernel size

Fortæl om kernel sizes og om hvordan vi først prøvede én størrelse på dem alle og siden at variere størrelsen (lav evt. en reference til nogen af dem der har trænet på MNIST og deres kernel sizes), lav en tabel og to grafer.

3.2 Final predictive capabilities

Forklar hvordan vi fandt frem til vores hyperparametre på multi-task modellen og så anvendte de samme på single-task. Skriv noget tekst om hvordan vi på test-sættet nåede op på so-and-so meget præcision på hhv. den ene og anden model og hhv. strukturer og solvent egenskaber, samt sammeligning af resultatet på test- og valideringssæt.

3.3 Comparison

Forklar hvordan de to modeller ender med næsten samme præcision, omend multi-task modellen tager langt længere tid om at komme derop. De convergerer (jeg tror ordet er 'predictive ceiling') begge omkring de 68.5%, men for single allerede omkring 10 epoker medens multi skal bruge 25 epoker.

4 Conclusion

Modellen blev ikke bedre, men blev nogenlunde god til både at forudsige struktur og solvent-egenskaber. Vi kan have nogle overvejelser omkring hvad vi kunne have gjort bedre, f.eks:

- Gaussisk filtrering over dataen
- Evt. soft parameter sharing
- Regularization hvis vi tør
- En anden opbygning af multi-modellen hvor der var et enkelt eller flere fully-connected layers til solvent-egenskaberne.

5 Appendix

References

[1] C. Bishop, *Pattern Recognition and Machine Learning.* Information Science and Statistics, Springer, 2006.