UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING
ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

Applying Saliency Map Analysis to CNNs on Protein Secondary Structure Prediction

by Guillermo Romero Moreno

The relatively new field of deep learning is slowly being transferred to the biology field and pushing its state-of-the-art achievements. However, improvements in performance come with the drawback of opaqueness, as what deep learning machines learn cannot be fully understood. Although a few authors have already started applying deep network interpretability techniques on biological problems to overcome this issue, none of them has been applied yet to the problem of protein secondary-structure prediction.

The aim of this work is to develop interpretability techniques for state-of-the-art deep networks that have been trained to solve the secondary-structure prediction problem. For doing so, a way to apply and aggregate saliency maps has been construed and applied to a near-state-of-the-art convolutional network, showing some further insights of the relationship between the inputs and the outputs. These results could be of double value: on one side, it may help biologists to get a better understanding on the underlying structural protein processes; on the other, machine learning researchers can understand better their machines and spot their flaws more easily.

Chapter 1

Theory

1.1 Deep Learning

The Deep Learning term was created in contrast to "shallow" Machine Learning. Typical machine learning algorithms involve processing the information from the input into a different *feature space* in which the data is represented in a more structured way. With deep learning, this process is repeated several times allowing for feature spaces with higher-level abstractions that have more power to learn complex relations in the input. These bigger networks were really hard to train at the beginning because of an increased number of parameters that significantly extend the computation training time, a higher risk to over-fit due to their increased power, and *vanishing gradients*, meaning that lower-layer parameters receive very weak signals from the error and could barely learn. Such problems have been recently overcome thanks to the use of Graphical Processing Units (GPUs) that boost computing speed, techniques such as *dropout* (**?**) and *batch normalization* (**?**) that add extra regularization, and the inclusion of Rectifying Linear Units (ReLUs) as non-linear functions. Since then, its use has been growing and it has been translated to many other research fields, with particularly high performance in image and speech recognition, language translation and natural language processing.

Multi-Layer Perceptrons (MLPs), Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) can be found among the most popular deep networks. MLPs are composed by consecutive layers of fully-connected units that pass the information in one direction, and hence are called *feedforward.* RNNs differ from MLPs in that units are also allowed to have connections to themselves. Instead of having an input vector that produces an output in a *one-go* fashion, RNNs store information from previous inputs and are particularly well-suited for inputs that are sequentially correlated. CNNs slide a local filter throughout the input, allowing obtaining location-invariant information. A deeper look into this kind of network is shown in the following sub-section.

1.1.1 Convolutional Neural Networks

”Convolutional Neural Networks (CNN) are one of the most popular deep learning architectures nowadays. They provide especially good results over other methods in image recognition and natural language processing. Among their strengths, they are able to detect features independently of their location in the input vector, and the reduced number of weights per layer favours the possibility of making the architecture remarkably deep (even more than 100 layers He et al. (2015)).

Instead of having fully-connected layers of neurons, CNNs perform the transformation of the data by sliding a filter (convolutional operator) over the output of the previous layer. Several convolutions may be performed in parallel on the same layer, and different layers are usually interleaved with pooling operations (average or max) that reduce the dimensionality of the data and help smoothing out local deformations. In the field of bio-informatics, CNNs are useful for image processing. Another interesting application is to sequence data (proteins, DNA), where they are able to recognise specific motifs with independence of their location Jurtz et al. (2017).” [My lit review]

”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 neuron in the next layer each time it is moved as illustrated in Figure 1B. The filter will thereby identify features in input irrespectively of where they appear. This concept is visualized in Figure 1C using the example of a convolutional filter detecting a motif in an amino acid sequence. Pooling such as mean pooling (averaging of nearby positions) enables the network to become invariant to small local deformations in the input. Convolutional neural networks often consist of many convolutional filters and many convolutional and pooling layers to enable the network to integrate the information from the different filters and various levels of abstraction (LeCun et al.,2015). When applied to biological problems, convolutional neural networks are ideally suited to locate motifs, for example in a protein sequence, independent of their position within the sequence.” Jurtz et al. (2017)

”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.” Jurtz et al. (2017)

"Each convolution layer consists of four consecutive operations: 1) The convolution operation with certain kernel size. 2) The batch normalization (Ioffe and Szegedy, 2015) operation is applied to help speed up the training process and acts as a regularizer. 3) The activation operation, ReLU, (Radford et al., 2015) was used as an activation function. 4) The dropout (Srivastava et al., 2014) operation to prevent the neural network from overfitting randomly drops neurons during the deep network training process such that the network can avoid too much co-adapting." Fang et al. (2017b)

"The proposed Deep3I network (see Fig. 2) differs from the previous net- work (Li and Yu, 2016; Busia and Jaitly, 2017) in that the latter ones used residual blocks and multi-scale layer containing CNN layers with a convolution window size of 3, 7, and 9 to discover protein local and global context. Deep3I consists of stacked CNN layers, whose convolution window size is only 3. When stacked deep convolution blocks are put together, they can perform both local and global context extraction. Applying convolution on top of convolution, the sliding window will cover a wide range of protein sequences by using this hierarchical convolutional operation." Fang et al. (2017b)

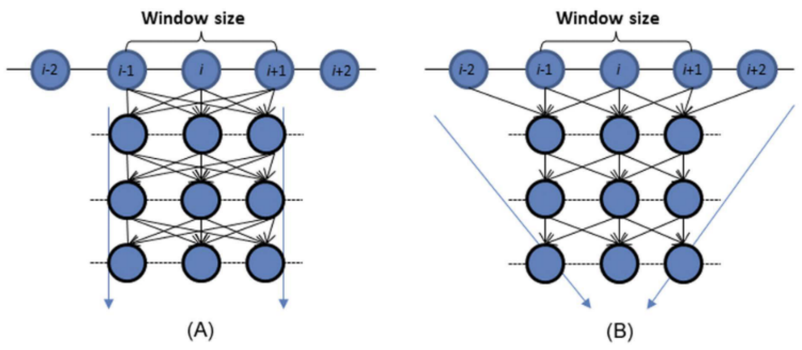


Figure 1. A typical deep neural network (**A**) vs. a convolutional deep neural network (**B**). A convolutional deep neural network can capture longer-range sequence information than a typical deep neural network when both use the same window size.

"Although a fully-connected layerwhere every input of interest is connected to every neuron in the layer by a distinct weightis usually considered the simplest type of layer in a neural network, in some senses a convolutional layer can be thought of as a further simplification. Whereas fully-connected layers can be understood as an attempt to capture information from the inputs simultaneously by defining a large number of neurons with distinct weights, convolutional layers consist of filterssmaller groups of neurons which look at segments of the input sequence at a time. Thus, a convolutional filter can be interpreted as sliding along the input sequence, reusing the same few weights on each local patch of the input. [...] The convolutional filters, CF and CF, are defined by their widththe number of inputs examined at a timeand depththe number of neurons in the filter. [...] one of the major benefits of convolutions compared to a the previous fixed-window, fully-connected approach: in situations where local properties of the data are critical, stacking small filterseach of which examines a small local input patch at a timeintroduces the ability to learn and maintain information about sequence dependencies at different scales. [...] filters in lower layers focus on extracting information from small local contextsin this case, of three residueswhile filters in higher layers cover correlations which are more spatially spread-out in the input sequence." Busia and Jaitly (2017)

"Convolutional neural networks (CNN) [LeCun et al., 1998], a specific type of deep neural networks using translation-invariant convolutional kernels, can be applied to extracting local con- textual features and have proven to be effective for many natural language processing (NLP) tasks [Yih et al., 2011; Zhang et al., 2015]. Inspired by their success in text classification, in this paper, CNNs with various kernel sizes are used to extract multiscale local contexts from a protein sequence." Li and Yu (2016)

1.1.2 Drawbacks of deep-learning

Despite the benefits of improved performance, deep learning still finds some resistance in the broader public due to varied reasons. A First, its higher complexity imposes a steeper learning curve. Second, the lack of mature set of tools and samples due to its novelty makes its access difficult, although there has been an increasing, active community that is swiftly covering these needs. Third, the high computational times are only partially relieved by GPUs. Last, deep networks are hard to interpret and obtaining information from the middle layers renders futile. Their *black box* behaviour hinders developers in both understanding where networks fail and why certain outputs are produced. This drawback has been of big concern lately, and the research community has switched the focus on developing *interpretability techniques* to overcome it.

1.2 Interpretability techniques

"There is a growing sense that neural networks need to be interpretable to humans. The field of neural network interpretability has formed in response to these concerns. As it matures, two major threads of research have begun to coalesce: feature visualization and attribution.

Feature visualization answers questions about what a networkor parts of a networkare looking for by generating examples.

Attribution (or saliency maps) studies what part of an example is responsible for the network activating a particular way." Olah et al. (2017)

"One of the challenges of neural networks is understanding what exactly goes on at each layer. We know that after training, each layer progressively extracts higher and higher-level features of the image, until the final layer essentially makes a decision on

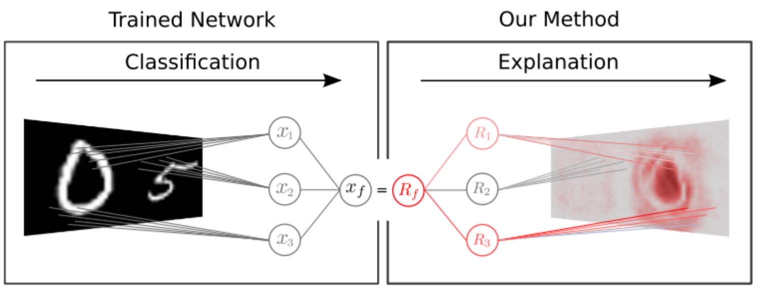


Fig. 1. Overview of our method for explaining a nonlinear classification decision. The method produces a pixel-wise heatmap explaining *why* a neural network classifier has come up with a particular decision (here, detecting the digit "0" in an input image composed of two digits). The heatmap is the result of a *deep* Taylor decomposition of the neural network function. Note that for the purpose of the visualization, the left and right side of the figure are mirrored.

what the image shows. For example, the first layer maybe looks for edges or corners. Intermediate layers interpret the basic features to look for overall shapes or components, like a door or a leaf. The final few layers assemble those into complete interpretations—these neurons activate in response to very complex things such as entire buildings or trees." Mordvintsev et al. (2015)

"Deep neural networks are highly expressive models that have recently achieved state of the art performance on speech and visual recognition tasks. While their expressiveness is the reason they succeed, it also causes them to learn uninterpretable solutions that could have counter-intuitive properties." Szegedy et al. (2013)

"Nonlinear methods such as Deep Neural Networks (DNNs) are the gold standard for various challenging machine learning problems, e.g., image classification, natural language processing or human action recognition. Although these methods perform impressively well, they have a significant disadvantage, the lack of transparency, limiting the interpretability of the solution and thus the scope of application in practice. Especially DNNs act as black boxes due to their multilayer nonlinear structure. [...] An interpretable classifier explains its nonlinear classification decision in terms of the inputs. For instance, in image classification problems, the classifier should not only indicate whether an image of interest belongs to a certain category or not, but also explain what structures (e.g. pixels in the image) were the basis for its decision (cf. Figure 1). This additional information helps to better assess the quality of a particular prediction, or to verify the overall reasoning ability of the trained classifier. Also, information about which pixels are relevant in a particular image, could be used for determining which region of the image should be the object of further anal- ysis. Linear models readily provide explanations in terms of input variables (see for example [19], [20]). However, because of the limited expressive power of these models, they perform poorly on complex tasks such as image recognition. Extending linear analysis techniques to more realistic nonlinear models such as deep neural networks, is therefore of high practicala relevance." Montavon et al. (2017)

”Networks with Rectified Linear Units (ReLUs) create nonlinearities that must be addressed. Several variants exist for handling this [448,454]. Backpropagation-based methods are a highly active area of research. Researchers are still actively identifying weaknesses [455], and new methods are being developed to address them [219,456,457]. Lundberg and Lee [458] noted that several importance scoring methods including integrated gradients and LIME could all be considered approximations to Shapely values [459], which have a long history in game theory for assigning contributions to players in cooperative games.” Ching et al. (2017)

”The purported black box nature of neural networks is a barrier to adoption in applications where interpretability is essential.” Shrikumar et al. (2017)

1.2.1 Feature visualization

”While quantitative analyses and comparisons of such models exist, and visualizations of the first layer representations are common in the literature, one area where more work needs to be done is the qualitative analysis of representations learned beyond the first level. [...] Our aim was to explore ways of visualizing what a unit computes in an arbitrary layer of a deep network. The goal was to have this visualization in the input space (of images), to have an efficient way of computing it, and to make it as general as possible (in the sense of it being applicable to a large class of neural-network-like models).” Erhan et al. (2009)

1.2.1.1 Linear combination of filters

”Lee et al. (2008) showed one way of visualizing what the units in the second hidden layer of a network are responding to. They made the assumption that a unit can be characterized by the filters of the previous layer to which it is most strongly connected (i.e. whose weight to the upper unit is large in magnitude). By taking a weighted linear combination of the previous layer filters [...]. Lee et al. (2009) used an extended version of this method for visualizing units of the third layer [...]. Such a technique is simple and efficient. One disadvantage is that it is not clear how to automatically choose the appropriate number of filters to keep at each layer. [...], this method also bypasses the nonlinearities between layers” Erhan et al. (2009)

It is also bad because there is a unbalanced overlapping that is stronger on the middle than the sides.

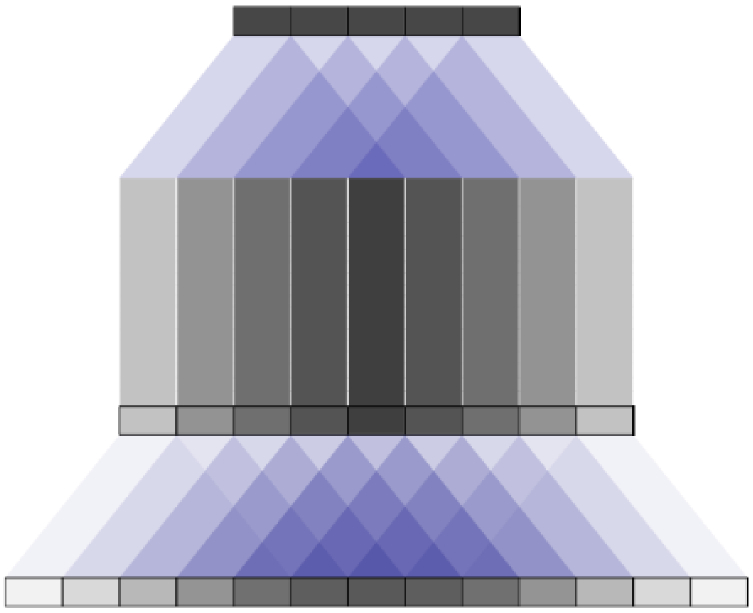


FIGURE 1.1:

1.2.1.2 Activation maximization

”we look for input patterns of bounded norm which maximize the activation of a given hidden unit (The total sum of the input to the unit from the previous layer plus its bias.); since the activation function of a unit in the first layer is a linear function of the input, in the case of the first layer, this input pattern is proportional to the filter itself. [...] One simple way of doing this is to find, for a given unit, the input sample(s) (from either the training or the test set) that give rise to the highest activation of the unit. Unfortunately, this still leaves us with the problem of choosing how many samples to keep for each unit and the problem of how to combine these samples. Ideally, we would like to find out what these samples have in common. Furthermore, it may be that only some subsets of the input vector contribute to the high activation, and it is not easy to determine which by inspection. [...] maximizing the activation of a unit as an optimization problem.

$$x* = arg \max_{x \ s.t.||x||=\rho} h_{ij}(\theta, x) \tag{1.1}$$

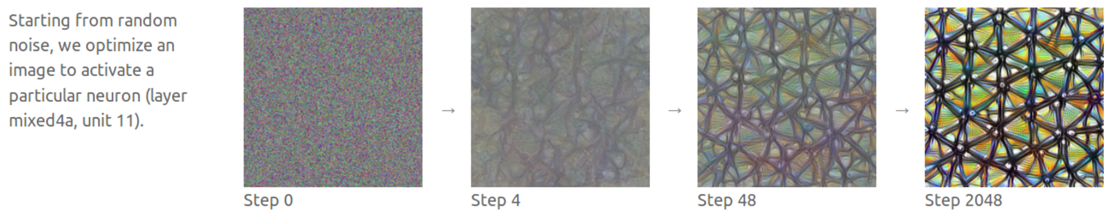
This is, in general, a non-convex optimization problem. But it is a problem for which we can at least try to find a local minimum. This can be done most easily by performing simple gradient ascent in the input space [...] the unit can then be characterized by the minimum or set of minima found. In the latter case, one can either average the results, or choose the one which maximizes the activation, or display all the local minima obtained to characterize that unit. This optimization technique (we will call it activation maximization) does involve a choice of hyperparameters: the learning rate and a stopping criterion” Erhan et al. (2009)

"the response of an internal unit to input images, as a function in image space, appears to be unimodal, or at least that the maximum is found reliably and consistently for all the random initializations tested. This is interesting because finding this dominant mode is relatively easy, and displaying it then provides a good characterization of what the unit does." Erhan et al. (2009)

"We tested the activation maximization procedure on image patches of 20 20 pixels (instead of 1212) and found that the optimization does not converge to a single global minimum. Moreover, the input distribution that is sampled with the units clamped to 1 has many different modes and its expectation is not meaningful or interpretable any-more. [...] It is perhaps unrealistic to expect that as we scale the datasets to larger and larger images, one could still find a simple representation of a higher layer unit. We should note, however, that there is a recent trend of developing convolutional versions of deep architectures (Kavukcuoglu et al., 2009; Lee et al., 2009; Desjardins & Bengio, 2008): it is likely that one will be able to apply the same techniques in that scenario and still be able to recover good visualizations, even with large inputs." Erhan et al. (2009)

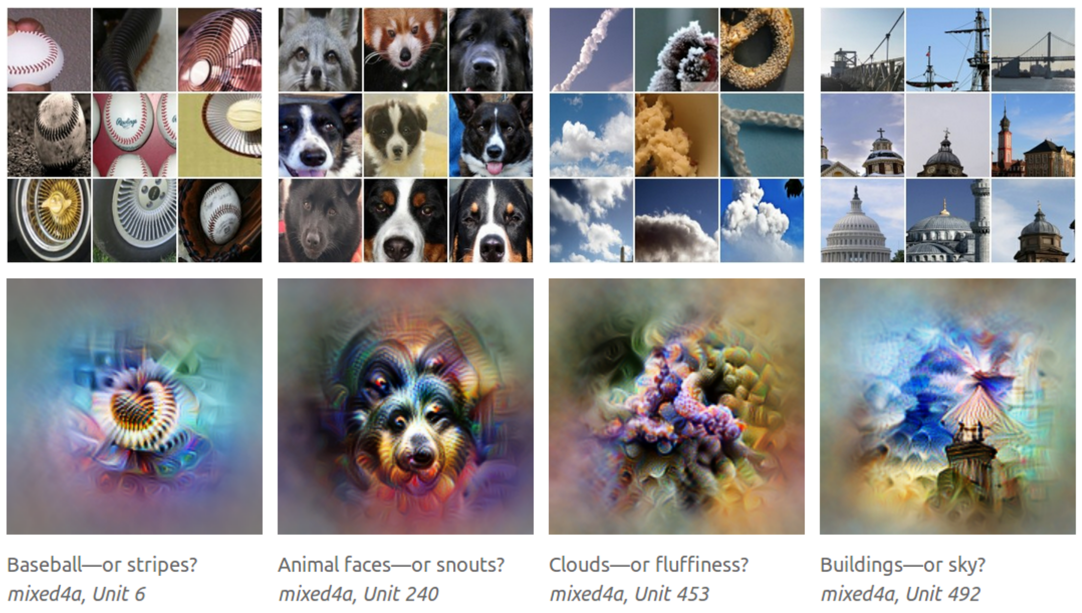
"One way to visualize what goes on is to turn the network upside down and ask it to enhance an input image in such a way as to elicit a particular interpretation. Say you want to know what sort of image would result in Banana. Start with an image full of random noise, then gradually tweak the image towards what the neural net considers a banana (see related work in [1], [2], [3], [4]). By itself, that doesnt work very well, but it does if we impose a prior constraint that the image should have similar statistics to natural images, such as neighboring pixels needing to be correlated." Mordvintsev et al. (2015)

"Neural networks are, generally speaking, differentiable with respect to their inputs. If we want to find out what kind of input would cause a certain behavior—whether that's an internal neuron firing or the final output behavior—we can use derivatives to iteratively tweak the input towards that goal [3]." Olah et al. (2017)


Starting from random noise, we optimize an image to activate a particular neuron (layer mixed4a, unit 11).
Step 0
Step 4
Step 48
Step 2048

" Optimization can give us an example input that causes the desired behavior—but why bother with that? Couldn't we just look through the dataset for examples that cause the desired behavior?

It turns out that optimization approach can be a powerful way to understand what a model is really looking for, because it separates the things causing behavior from things that merely correlate with the causes. [...] A neuron may not be detecting what you initially thought. Optimization also has the advantage of exibility. For example, if we want to study how neurons jointly represent information, we can easily ask how a particular example would need to be different for an additional neuron to activate. This exibility can also be helpful in visualizing how features evolve as the network trains. If we were limited to understanding the model on the xed examples in our dataset, topics like these ones would be much harder to explore. " Olah et al. (2017)



1.2.2 Saliency maps

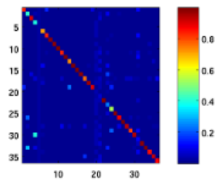
"Bach et al. [22] have introduced the concept of pixel-wise decomposition of a classification decision, and how such decomposition can be achieved either by Taylor decomposition, or by a relevance propagation algorithm. Specifically, the authors distinguish between (1) functional approaches that view the neural network as a function and disregard its topology, and (2) message passing approaches, where the decomposition stems from a simple propagation rule applied uniformly to all neurons of the deep network. [...] the methods proposed in [27], [28] do not explain the decision of a classifier but rather perform sensitivity analysis by computing the gradient of the decision function. This results in an analysis of variations of that function, without however seeking to provide a full explanation why a certain data point has been predicted in a certain way. Specifically, the gradient of a function does not contain information on the saliency of a feature in the data to which the function is applied. Simonyan et al. [24] incorporate saliency information by multiplying the gradient by the actual data point. The method proposed by Zeiler and Fergus [21] was designed to visualize and understand the features of a convolutional neural network with max-pooling and rectified linear units. The method performs a backpropagation pass on the network, where a set of rules is applied uniformly to all layers of the network, resulting in an assignment of values onto pixels. The method however does not aim to attribute a defined meaning to the assigned pixel values, except for the fact that they should form a visually interpretable pat- tern. [22] proposed a layer-wise propagation method where the backpropagated signal is interpreted as relevance, and obeys a conservation property. The proposed propagation rules were designed according to this property, and were shown quantitatively to better support the classification decision [25]. However, the practical choice of propagation rules among all possible ones was mainly heuristic and lacked a strong theoretical justification" Montavon et al. (2017)

"Shrikumar et al. and Kindermans et al. (Shrikumar et al., 2016; Kindermans et al., 2016) showed that absent modifications to deal with numerical stability, the original LRP rules were equivalent within a scaling factor to an elementwise product between the saliency maps of Simonyan et al. and the input (in other words, gradient input)." Shrikumar et al. (2017)

"When we assign blame to a certain cause we implicitly consider the absence of the cause as a base-line for comparing outcomes. In a deep network, we model the absence using a single baseline input. For most deep networks, a natural baseline exists in the input space where the prediction is neutral. For instance, in object recognition networks, it is the black image. The need for a baseline has also been pointed out by prior work on attribution (Shrikumar et al., 2016; Binder et al., 2016)." Sundararajan et al. (2017)

1.2.2.1 Perturbation-based forward propagation approaches (sensitivity analysis)

"To further test the robustness of the activation maximization method, we perform a sensitivity analysis in order to test whether the units are selective to these patterns found by the optimization routine, and whether these patterns strongly activate other units as well. The figure on the right shows the post-sigmoidal activation of unit j (columns) when the input to the network is the optimal pattern i (rows), found by our gradient procedure for unit i, normalized across columns in order to eliminate the effect of units that are activated for very many patterns in general. The strong values on the diagonal suggest that the results of the optimization have uncovered patterns that are mostly specific to a particular unit." Erhan et al. (2009)



"Simonyan et al. [24], however, demonstrated that DeConvNets can be interpreted as a sensitivity analysis of the network input/output relation." Mahendran and Vedaldi (2015)

"the methods proposed in [27], [28] do not explain the decision of a classifier but rather perform sensitivity analysis by computing the gradient of the decision function. This results in an analysis of variations of that function, without however seeking to provide a full explanation why a certain data point has been predicted in a certain way. Specifically, the gradient of a function does not contain information on the saliency of a feature in the data to which the function is applied. Simonyan et al. [24] incorporate saliency information by multiplying the gradient by the actual data point." Montavon et al. (2017)

"Zeiler and Fergus [36], who backtrack the network computations to identify which image patches are responsible for certain neural activations" Mahendran and Vedaldi (2015)

"These approaches make perturbations to individual inputs or neurons and observe the impact on later neurons in the network. Zeiler & Fergus (Zeiler & Fergus, 2013) occluded different segments of an input image and visualized the change in the activations of later layers. In-silico muta- genesis (Zhou & Troyanskaya, 2015) introduced virtual mutations at individual positions in a genomic sequence and quantified the their impact on the output. Zintgraf et al. (Zintgraf et al., 2017) proposed a clever strategy for analyzing the difference in a prediction after marginalizing over each input patch. However, such methods can be com- putationally inefficient as each perturbation requires a sep- arate forward propagation through the network. They may also underestimate the importance of features that have sat- urated their contribution to the output." Shrikumar et al. (2017)

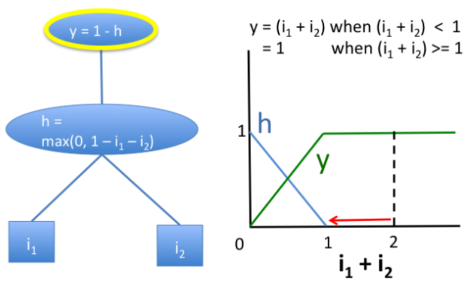


Figure 1. **Perturbation-based approaches and gradient-based approaches fail to model saturation**. Illustrated is a simple network exhibiting saturation in the signal from its inputs. At the point where $i_1 = 1$ and $i_2 = 1$, perturbing either i_1 or i_2 to 0 will not produce a change in the output. Note that the gradient of the output w.r.t the inputs is also zero when $i_1 + i_2 > 1$.

1.2.2.2 Backpropagation-based approaches

”In contrast to perturbation methods, backpropagation approaches are computationally efficient as they propagate an importance signal from the output neuron backwards through the layers towards the input in a single pass. DeepLIFT belongs to this family of approaches. [...]

GRADIENTS, DECONVOLUTIONAL NETWORKS AND GUIDED BACK-PROPAGATION Simonyan et al. Simonyan et al. (2014) proposed using the gradient of the output w.r.t. pixels of an input image to compute a saliency map of the image in the context of image classification tasks. The authors showed that this was similar to deconvolutional networks Zeiler and Fergus (2014) except for the handling of the nonlinearity at rectified linear units (ReLUs). When backpropagating importance using gradients, the gradient coming into a ReLU during the backward pass is zerod out if the input to the ReLU during the forward pass is negative. By contrast, when back-propagating an importance signal in deconvolutional networks, the importance signal coming into a ReLU during the backward pass is zerod out if and only if it is neg-ative, with no regard to sign of the input to the ReLU during the forward pass. Springenberg et al., (Springenberg et al., 2014) combined these two approaches into Guided Backpropagation, which zeros out the importance signal at a ReLU if either the input to the ReLU during the for-ward pass is negative or the importance signal during the backward pass is negative. Guided Backpropagation can be thought of as equivalent to computing gradients, with the caveat that any gradients that become negative during the backward pass are discarded at ReLUs. Due to the zeroing out of negative gradients, both guided backpropagation and deconvolutional networks can fail to highlight inputs that contribute negatively to the output. Additionally, none of the three approaches would address the saturation problem illustrated in Fig. 1, as the gradient of y w.r.t. h is negative (causing Guided Backprop and deconvolutional networks to assign zero importance), and the gradient of h w.r.t both i1 and i2 is zero when i1 + i2 ¿ 1 (causing both gradients and Guided Backprop to be zero). Discontinuities in the gradients can also cause undesirable artifacts.” Shrikumar et al. (2017)

”Gradients violate Sensitivity(a): For a concrete example, consider a one variable, one ReLU network, f(x) = 1 ReLU(1x). Suppose the baseline is x = 0 and the input is x = 2. The function changes from 0 to 1, but because f be- comes flat at x = 1, the gradient method gives attribution of 0 to x. Intuitively, gradients break Sensitivity because the prediction function may flatten at the input and thus have zero gradient despite the function value at the input being different from that at the baseline. This phenomenon has been reported in previous work (Shrikumar et al., 2016).” Sundararajan et al. (2017)

”A second set of approaches involve back-propagating the final prediction score through each layer of the network down to the individual features. These include DeepLift, Layer-wise relevance propagation (LRP), Deconvolutional networks (DeConvNets), and Guided

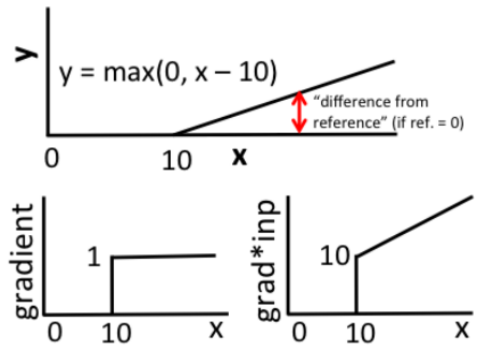


Figure 2. **Discontinuous gradients can produce misleading importance scores.** Response of a single rectified linear unit with a bias of -10. Both gradient and gradient$\times$input have a discontinuity at $x = 10$; at $x = 10 + \epsilon$, gradient$\times$input assigns a contribution of $10 + \epsilon$ to x and -10 to the bias term (ϵ is a small positive number). When $x < 10$, contributions on x and the bias term are both 0. By contrast, the difference-from-reference (red arrow, top figure) gives a continuous increase in the contribution score.

back-propagation. These methods differ in the specific backpropagation logic for various activation functions (e.g., ReLU, MaxPool, etc.). [...] Unfortunately, Deconvolution networks (DeConvNets), and Guided back-propagation violate Sensitivity(a). This is because these methods back-propogate through a ReLU node only if the ReLU is turned on at the input. This makes the method similar to gradients, in that, the attribution is zero for features with zero gradient at the input despite a non-zero gradient at the baseline." Sundararajan et al. (2017)

"Two networks are functionally equivalent if their outputs are equal for all inputs, despite having very different implementations. Attribution methods should satisfy Implementation Invariance, i.e., the attributions are always identical for two functionally equivalent networks. [...] gradients are invariant to implementation. In fact, the chain-rule for gradients f/g = f h h g is essentially about implementation invariance." Sundararajan et al. (2017)

LAYERWISE RELEVANCE PROPAGATION AND GRADIENT INPUT

"Bach et al. (Bach et al., 2015) proposed an approach for propagating importance scores called Layerwise Relevance Propagation (LRP).

Shrikumar et al. and Kindermans et al. (Shrikumar et al., 2016; Kindermans et al., 2016) showed that absent modifications to deal with numerical stability, the original LRP rules were equivalent within a scaling factor to an elementwise product between the saliency maps of Simonyan et al. and the input (in other words, gradient input). [...] While gradient input is often preferable to gradients alone as it leverages the sign and strength of the input, it still does not address the saturation problem in Fig. 1 or the thresholding artifact in Fig. 2." Shrikumar et al. (2017)

"Simonyan et al. [24] incorporate saliency information by multiplying the gradient by the actual data point." Montavon et al. (2017)

"Gradients. For linear models, ML practitioners regularly inspect the products of the model coefficients and the feature values in order to debug predictions. Gradients (of the output with respect to the input) is a natural analog of the model coefficients for a deep network, and therefore the product of the gradient and feature values is a reasonable starting point for an attribution method (Baehrens et al., 2010; Simonyan et al., 2013); see the third column of Figure 2 for examples. The problem with gradients is that they break sensitivity, a property that all attribution methods should satisfy. [...] Practically, the lack of sensitivity causes gradients to focus on irrelevant features" Sundararajan et al. (2017)

INTEGRATED GRADIENTS "Instead of computing the gradients at only the current value of the input, one can integrate the gradients as the inputs are scaled up from some starting value (eg: all zeros) to their current value (Sundararajan et al., 2016). This addressess the saturation and thresholding problems of Fig. 1 and Fig. 2, but numerically obtaining high-quality integrals adds computational overhead. Further, this approach can still give highly misleading results (see Section 3.4.3)." Shrikumar et al. (2017)

DeepLIFT "Our approach is unique in two regards: first, it frames the question of importance in terms of differences from a reference state, where the reference is chosen by the user according to what is appropriate for the problem at hand. In contrast to most gradient-based methods, using a difference-from-reference allows DeepLIFT to propagate an importance signal even in situations where the gradient is zero and avoids artifacts caused by discontinuities in the gradient. Second, by optionally giving separate consideration to the effects of positive and negative contributions at nonlinearities, DeepLIFT can reveal dependencies missed by other approaches. As DeepLIFT scores are com- puted using a backpropagation-like algorithm, they can be obtained efficiently in a single backward pass through the network after a prediction has been made." Shrikumar et al. (2017)

"We now discuss two axioms (desirable characteristics) for attribution methods. We find that other feature attribution methods in literature break at least one of the two axioms. These methods include DeepLift (Shrikumar et al., 2016; 2017), Layer-wise relevance propagation (LRP) (Binder et al., 2016), Deconvolutional networks Zeiler and Fergus (2014), and Guided back-propagation (Springenberg et al., 2014)." Sundararajan et al. (2017)

"Methods like DeepLift and LRP tackle the Sensitivity issue by employing a baseline, and in some sense try to compute discrete gradients instead of (instantaeneous) gradients at the input. (The two methods differ in the specifics of how they compute the discrete gradient). But the idea is that a large, discrete step will avoid flat regions, avoiding a break- age of sensitivity. [...] Methods like LRP and DeepLift replace gradients with discrete gradients and still use a modified form of backpropagation to compose discrete gradients into attributions. Unfortunately, the chain rule does not hold for discrete gradients in general. Formally f(x1)f(x0) / g(x1)g(x0) != f(x1)f(x0) / h(x1)h(x0) h(x1)h(x0) / g(x1)g(x0) , and therefore these methods fail to satisfy implementation invariance. If an attribution method fails to satisfy Implementation Invariance, the attributions are potentially sensitive to unimportant aspects of the models." Sundararajan et al. (2017)

Deep Taylor Decomposition "The main goal of this paper is to reconcile the functional and rule-based approaches for obtaining these decompositions, in a similar way to the error backpropagation algorithm [23] that also has a functional and a message passing interpretation. We call the resulting framework deep Taylor decomposition. This new technique seeks to replace the analytically intractable standard Taylor decomposition problem by a multitude of simpler analytically tractable Taylor decompositionsone per neuron. [...] Our method is based on deep Taylor decomposition and efficiently utilizes the structure of the network by backpropagating the explanations from the output to the input layer. [...] In this paper, we focus instead on the interpretation of the prediction of individual data points, for which portions of the trained model may either be relevant or not relevant. [...] The classification decision is first decomposed in terms of contributions R1, R2, R3 of respective hidden neurons x1, x2, x3, and then, the contribution of each hidden neuron is independently redistributed onto the pixels, leading to a relevance map (or heatmap) in the pixel space, that explains the classification 0. A main result of this work is the observation that application of deep Taylor decomposition to neural networks used for image classification, yields rules that are similar to those proposed by [22] (the -rule and the ?-rule), but with specific instantiations of their hyperparameters, previously set heuristically." Montavon et al. (2017)

1.3 Deep Learning in biology

"The field of biology has gone through major changes in the past decades thanks to new technologies that allow measuring and harvesting increasing amounts of data. Despite its obvious potential for pushing forward our knowledge, such data comes with intrinsic difficulties for its analysis; namely (i) **extremely bulky datasets** that require highly efficient algorithms and machines, and long processing times; (ii) **high-dimensionality**, that hinders biologists from getting direct insights from the data distributions, and brings the so-called *curse of dimensionality* to stage; (iii) **complicated interactions** and data dependencies, due to the high complexity of processes such as gene regulatory networks or protein folding and binding; (iv) **heterogeneous and multi-platform resources**, imposing difficulties at integrating data from different sources in the same mathematical model.

In order to carry analysis on the data, biologists traditionally utilised machine learning algorithms and statistical models, performing data simplification (e.g. clustering, dimensionality reduction), prediction, classification, or modelling. They not only offer the intrinsic benefits of their outcome, but also great insight on the underlying bio-chemical processes by inspection of the learned parameters that the models produce. Despite their (sometimes moderate) success, these methods usually depend on key prior information for achieving their goals, usually coming from online records of annotations related to the data at task, which are not always exhaustive enough. They also bring the drawback of requiring expertise in the field for the careful selection of the right features **?**.

This scenario is seeing a major improvement with the recent explosion of the set of techniques under the label *deep learning*. Such algorithms differ from *shallow* machine learning methods in the sense that they include more layers of non-linearities, providing stronger modelling power for complex and hierarchical relations in the data. Such property allows them to cope with some of the afore-mentioned problems, as (i) they are able to learn *intermediate key features* from the data on their own, freeing biologists from the tedious process of hand-crafting them; (ii) they are also good at dealing with high-dimensionality; and (iii) their higher generalization properties make them more suitable for integrating heterogeneous sources." [My lit review]

"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" Jurtz et al. (2017)

"

1.3.0.1 Genomics

The DNA encodes the information about all the proteins that the cells need, alongside with purely regulatory sections, or parts that appear to have no purpose. New techniques in genome sequencing, such as *high-throughput sequencing (HTS)*, have provided unprecedented quantities of genomic data at ever-lower costs. The availability of huge amounts of data makes deep learning an appropriate tool for helping in the task.

Due to the sequential nature of the data, RNNs are especially suited for structural annotations of the DNA chain. CNNs can also be of help at detecting specific motifs. Successful applications of deep learning in the field of genomics include the detection of non-coding regulatory fragments Zhou and Troyanskaya (2015), or protein binding site prediction **?**. Deep learning is also useful for *metagenomics*, i.e. the analysis of microbial genome from specific environments, where Ditzler *et al.* showed that in spite of having similar accuracy than shallow MLPs, it provided useful hierarchical representations of the data **?**.

1.3.0.2 Transcriptomics

Transciptomics refers to the study of the processes through which specific DNA sequences are copied into different types of RNA strands: *messenger RNA* (mRNA), *long non-coding RNA* (lncRNA), and *microRNA* (miRNA). Deep learning techniques have been applied for predicting splicing sites in mRNA (5% AUC improvement **?**), monitoring gene expression on images (5% AUC improvement with CNNs **?**), classifying lncRNA (3,4% higher accuracy **?**), or identifying *expression quantitative trait loci (eQTL)*(with few points improvement in AUC over baseline methods **?**).

1.3.0.3 Proteomics

Proteomics is the large-scale study of proteins, usually looking at the scope of the entire collection of proteins that an organism produces. Although the data from this field is still not enough for consistent use of deep learning methods, some unsupervised techniques have helped building hierarchical representations of the interaction of protein networks **?**.

1.3.0.4 Structural biology

Structural biology applications include protein folding and drug design. Proteins are the basic building unit of cells. They are formed by chains of amino-acids that fold into different shapes, which will determine the function that the protein performs. Modifications on the structure can alter their dynamics and influence the appearing of the diseases. Techniques for 3D measuring of their shapes are hard and costly, so it has only been done in very small sample sets. Deep learning can be used as an alternative for predicting the structure purely from the protein sequence. An intermediate typical phase consists on the prediction of the *secondary structure*, which is whether the specific sections of the protein assemble into an α-helix or a β-sheet (a few points improvement over previous methods Jurtz et al. (2017)). This annotation task is particularly well suited for RNNs, as explained before.

Other problem addressed by deep-learning include the classification of intrinsic disordered proteins (IDPs) **?**, or predicting binding behaviour for RNA-binding proteins (few points AUC improvement over baseline methods **?**).

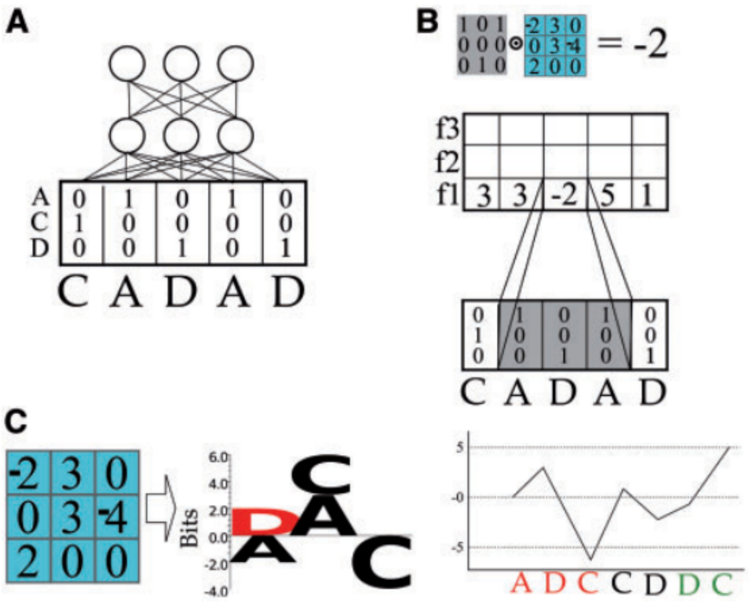
1.3.0.5 Multiomics

Multiomics refers to the combination of data and problems for the different omics fields, which can lead to more powerful models as they include interactions between the different processes of regulatory networks. Deep learning is especially well suited for performing this sort of study thanks to its capability of assimilating heterogeneous data and learning highly complex interactions. One particular example of this use can be found in epigenomics, which are the processes by which gene expression is changing without any modification of the DNA (5% to 10% improvement in AUC over previous methods Zhou and Troyanskaya (2015)). This requires the inclusion of data from genes, RNA, methylation, and histone modifications into the same model. Another example is the detection and clustering of cancer diseases from multiple source of data **?**. " [My lit review]

1.3.1 CNNs on biological sequence data

DeepCNF uses 2D convolutions with filters of size window times feature vector size.

"Amino Acids C, A, D, A, D are encoded as one-hot vectors with a 1 at the position corresponding to the amino acid type (A, C or D), and zero otherwise. A filter (blue) is slid over the input sequence. The filter here has a length of three amino acids. At each position the filter has a preference for different amino acid types. The filter output is calculated by taking the sum of the element-wise product of the input and the filter position-specific weights. Each time the filter is moved, it feeds into a different hidden neuron in the hidden layer, here visualized in 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." Jurtz et al. (2017)



"Convolutional Neural Networks (CNNs) have been used to overcome the limitations of windowing by applying layers of shifting convolutional kernels across the input sequence

(Wang et al., 2016). The addition of each subsequent convolutional layer includes a window of information from the layer prior. This has the effect of growing the size of the window being applied to the input data, each time a layer is added. However, even though the effective window size increases at each layer, the total window is still finite and limited. DeepCNF for example uses five CNN layers, each with a window size of 11 (five on either side), this is only able to capture information from residues plus or minus 25 sequence positions away." Heffernan et al. (2017)

1.3.2 Interpretability techniques on biological problems

"In recent years, there has been an explosion of deep learning models which have lead to groundbreaking results in many fields such as computer vision[13], natural language processing[25], and computational biology [2, 19, 27, 11, 14, 22]. However, although these models have proven to be very accurate, they have widely been viewed as black boxes due to their complexity, making them hard to understand. This is particularly unfavorable in the biomedical domain, where understanding a models predictions is extremely important for doctors and researchers trying to use the model. [...] While making accurate predictions is important in biomedical tasks, it is equally important to understand why models make their predictions. Accurate, but uninterpretable models are often very slow to emerge in practice due to the inability to understand their predictions, making biomedical domain experts reluctant to use them. Consequently, we aim to obtain a better understanding of why certain models work better than others, and investigate how they make their predictions by introducing several visualization techniques. [...] The main difference between our work and previous works on images and natural language is that instead of trying to understand the DNNs given human understanding of such human perception tasks, we attempt to uncover critical signals in DNA sequences given our understanding of DNNs." Lanchantin et al. (2016)

1.3.2.1 Feature visualization

"The networks whose filters were saved and plotted only used one-hot encoding so as to be able to represent all features as PSSM-logos. The visualization of the convolutional filters was based on the methodology used in [37]. The convolutional filters are represented as a matrix of filter_width columns and aa_encoding rows. So as to visualize the relative importance of each of the positions in the filter, each of them is rescaled in a way that the height of the highest column is 1. After this transformation, each filter can be visualized as a PSSM logo where the position importance is proportional to the height of the column and the height of each letter is proportional to the importance of the amino acid in that position. Seq2logo [42] is used in order to generate the PSSM-logo plots. The Seq2Logo default amino acid colour coding is used: negatively charged (DE) residues are red, polar uncharged residues (NQSGTY) are green, positively charged (RKH) residues are blue and the remaining are black.” Fontal (2017)

”For TFBS prediction, Alipanahi et al.[2] was the first to implement a visualization method on a DNN model. They visualize their CNN model by extracting motifs based on the input subsequence corresponding to the strongest activation location for each convolutional filter (which we call convolution activation). Since they only have one convolutional layer, it is trivial to map the activations back, but this method does not work as well with deeper models. We attempted this technique on our models and found that our approach using saliency maps outperforms it in finding motif patterns (details in section 4). Quang and Xie [19] use the same visualization method on their convolutional-recurrent model for noncoding variant prediction.” Lanchantin et al. (2016)

”In the case of the NN trained in this project, where the inputs are protein sequences encoded as one-hot vectors, the optimized inputs lose the one-hot vector encoding but can be forced to behave as a Position Probability Matrix (PPM) which can then be represented as a motif. Formally, the following equation, where Si is the score (defined as the unscaled values for class Ci pre Softmax transformation) and X is the input sequence, is optimized: arg max Si(X) + ? X ?2 2. is the regularisation parameter. The L2-regularization term is introduced in order to minimize the number of significant amino acids per position in the optimized input sequence. The score used is the value of the pre-Softmax layer because the post-Softmax value can be maximized by just minimizing the scores of the other classes, and the objective is to get an ideal input for the desired class, not an input modified to score low in the remaining classes.” Fontal (2017)

Fontal didn’t add any prior, thus risking having unnatural samples.

”Now we introduce an approach to extract a class-specific visualization for a DNN model, where we attempt to find the best sequence which maximizes the probability of a positive TFBS, which we call class optimization. Formally, we optimize the following equation where S+(X) is the probability (or score) of an input sequence X (matrix in our case) being a positive TFBS computed by the softmax equation of our trained DNN model for a specific TF: arg maxS+(X) + ?X?2 2 where is the regularization parameter. We find a locally optimal X through stochastic gradient descent, where the optimization is with respect to the input sequence. In this optimization, the model weights remain unchanged. This is similar to the methods used in Simonyan et al.[21] to optimize toward a specific image class. This visualization method depicts the notion of a positive TFBS class for a particular TF and is not specific to any test sequence.” Lanchantin et al. (2016)

”In order to focus on higher probable regions of the input space, the 2-norm regularizer can be replaced by a data density model p(x) which is called expert by Montavon et al.(2017). This leads to the following optimization problem: max x log p(c—x) + log p(x). Here, the prototype is encouraged to simultaneously produce strong class response and to resemble the data. By application of Bayes rule, the newly defined objective can be identified, up to modeling errors and a constant term, as the class-conditioned data density p(x—c). The learned prototype thus corresponds to the most likely input x for the class c. A possible choice for the expert is the Gaussian Restricted Boltzmann Machine (RBM). The RBM is a two-layer, bipartite, undirected graphical model with a set of binary hidden units p(h), a set of (binary or real-valued) visible units p(v), with symmetric connections between the two layers represented by a weight matrix W. The probabilistic semantics for an RBM is defined by its energy function" Montavon et al. (2018)

1.3.2.2 Saliency maps

Signal masking approach "This method was developed along the course of the project in order to evaluate the effect of removing the signal from certain regions of the input sequences. It is a way of assessing, for a trained model, which parts of the input sequence have a higher importance to determine the final classification." Fontal (2017)

Gradient-based approach "Our first visualization method is finding a test sequences saliency map which uses first-order derivatives to describe the importance of each nucleotide in making the final prediction. [...] we seek to visualize the influence of each position (i.e. nucleotide) on the prediction. Our approach is similar to the methods used on images by Simonyan et al. (2014) and Baehrens et al.[4]. Given a sequence X0 of length —X0—, and class c C, a DNN model provides a score function Sc(X0). We rank the nucleotides of X0 based on their influence on the score Sc(X0). Since Sc(X) is a highly non-linear function of X with deep neural nets, it is hard to directly see the influence of each nucleotide of X on Sc. Mathematically, around the point X0, Sc(X) can be approximated by a linear function by computing the first-order Taylor expansion: Sc(X) wTX + b = ?—X— wixi + b where w is the derivative of Sc with respect to the sequence variable X at the point X0: w =ScX—X0 = saliency map This derivative is simply one step of backpropagation in the DNN model, and is therefore easy to compute. We do a pointwise multiplication of the saliency map with the one-hot encoded sequence to get the derivative values for the actual nucleotide characters of the sequence (A,T,C, or G) so we can see the influence of the character at each position on the output score. Finally, we take the element-wise magnitude of the resulting derivative vector to visualize how important each character is regardless of derivative direction. We call the resulting vector a saliency map[21] because it tells us which nucleotides need to be changed the least in order to affect the class score the most. As we can see from equation 5, the saliency map is simply a weighted sum of the input nucleotides, where the each weight, wi, indicates the influence of that nucleotide position on the output score. [...] From each positive test sequence (thus, 500 total for each TF dataset) we extract a motif from the saliency map by selecting the contiguous length-9 subsequence that achieves the highest sum of contiguous length-9 saliency map values." Lanchantin et al. (2016) He uses gradient * input.

They prove that the method is working right by comparing the motifs with motif database of known samples.

1.4 Secondary Structure Prediction

Soon after they have been formed, proteins fold into 3D structures that primarily determine their function in the cell. Protein functions are of great value for the medical field, as their knowledge can help drug design, disease treatment, or early diagnosis. However, due to the molecular scale, these structures cannot be easily measured, and any attempt to do so remains costly. A more feasible alternative is utilizing computational tools to predict protein structures from amino-acid sequences alone, as these can be easily obtained from DNA sequencing and they are widely available on heavily annotated public databases. Hattori et al. (2017) points out that from the nearly 85 millions proteins whose sequences are known, we only know the structure of about 130 thousands.

(Dill and MacCallum, 2012) "it is a problem which has remained unsolved since its inception half a century ago (Gibson and Scheraga, 1967)" Heffernan et al. (2017) "site-specific mutation experiments (Drozdetskiy et al., 2015)." Fang et al. (2017b) "Accurately and reliably predicting structures, especially 3D structures, from protein sequences is one of the most challenging tasks in computational biology, and has been of great interest in bioinformatics [Ashraf and Yaohang, 2014]."Li and Yu (2016) (Yaseen and Li, 2014)

3D protein structure prediction from the sequence is a rather complex task, and thus researchers have focused firstly on the prediction of the so-called secondary structure, which consists on detecting local structures that can be used later for an easier modelling of the tertiary (3D) structure. Broadly, there are three kinds of local structures: α-helix, β-sheet, and coil, which includes everything else. A finer classification schema developed by DSSP comprehends eight classes, three helices, two sheets, and three coils.

(Kabsch and Sander, 1983) "Secondary structure prediction can be dated back to 1951 when Pauling and Corey proposed helical and sheet conformations of protein backbone based on hydrogen bonding patterns (Pauling et al.,1951)." Heffernan et al. (2017) "This hypothesis has driven a decades-long effort, spanning multiple disciplines, to deduce how protein sequence determines a proteins structural and functional properties (Dill et al., 2008; Anfinsen, 1977)." Busia and Jaitly (2017)

"After translation, proteins fold into a 3-dimensional structure, known as the tertiary structure (Fig. 3A)."Jurtz et al. (2017) "The 3D structure of a protein is determined largely by its amino acid sequence1." Wang et al. (2016) "Knowledge of a proteins 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)." Jurtz et al. (2017) "However it is extremely challenging to predict protein structure from sequence alone2." Wang et al. (2016) "An important step in the prediction of tertiary protein structure is the prediction of the secondary structure, the local conformations of the peptide back-bone." Jurtz et al. (2017) "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)." Jurtz et al. (2017) "Since protein structure is critical to analysis of its function and many applications like drug and/or enzyme design35, understanding the complex sequence-structure relationship is one of the greatest challenges in computational biology68. Accurate protein structure and function prediction relies, in part, on the accuracy of secondary structure prediction912. [...] Overall, protein secondary structure can be regarded as a bridge that links the primary sequence and tertiary structure and thus, is used by many structure and functional analysis tools1518" Wang et al. (2016) "Protein tertiary structure prediction from amino acid sequence is a very challenging problem in computational biology (Yaseen and Li, 2014; Dill and MacCallum, 2012). However, if a protein secondary structure can be predicted accurately, it can provide useful constraints for 3D protein structure prediction. Protein secondary structure can also help identify the protein function domains and may guide the rational design of site-specific mutation experiments (Drozdetskiy et al., 2015)." Fang et al. (2017b) "The three-dimensional structure is also known as native conformation and it is a function of its secondary structures. A number of diseases, including cancer, Alzheimers disease, cystic fibrosis, Huntingtons disease and diabetes are linked to the result of the aggregation of ill-formed proteins [1], [2]. Notwithstanding, despite a large number of proteins that have been discovered recently (around 84.8 million records in the UniProtKB/TrEMBL repository as in may/2017), only a few of them have their structure known (129,745 in the Protein Data Bank PDB as in may/2017). Therefore, acquiring knowledge about the structure of proteins is an important issue, since such knowledge can lead to important medical and biochemical advancements and even to the development of new drugs with specific functionality [3], [2]. [...] A possible way to infer the full structure of an unknown protein is to determine secondary structures in it. However, the pattern formation rules of secondary structures of proteins are still not known precisely [3]." Hattori et al. (2017) "Proteins perform a wide variety of molecular functions, and most of those functions are determined by the proteins three-dimensional structures. The problem of predicting the three-dimensional structure of a protein, given only its linear sequence of residues, is crucially important due to the high cost of experimental protein structure determination and the low cost of DNA sequencing to obtain protein sequences. However, even with this importance it is a problem which has remained unsolved since its inception half a century ago (Gibson and Scheraga, 1967; Dill and MacCallum, 2012; Zhou et al., 2011). Due to the challenge of predicting three-dimensional protein structure, the problem is commonly divided into smaller, more achievable problems, in the hopes that their solutions will ultimately lead to the solution for three-dimensional structure prediction. One type of these sub-problems is the prediction of one-dimensional structural properties of proteins, which can be represented as a one-dimensional vector along the protein sequence. The most commonly predicted one-dimensional structure of proteins is secondary structure. Secondary structure prediction can be dated back to 1951 when Pauling and Corey proposed helical and sheet conformations of protein backbone based on hydrogen bonding patterns (Pauling et al.,1951). Secondary structure is a course-grained descriptor of the local structure of the polypeptide backbone." Heffernan et al. (2017) "Protein-based interactions are responsible for controlling a variety of vital functions: they are critical in driving the immune system, regulating breathing and oxygenation, controlling aging and energy usage, and determining drug response, with a proteins particular functional role determined by its structure (Breda et al., 2006; Guo, Ellrott, & Xu, 2008). Over time, scientists have reached a consensus that a proteins structure primarily depends on its amino acid sequencelocal and long-range interactions between amino acid residues and their side-chains are both a cause and consequence of protein secondary and tertiary structure (Dill et al., 2008). This hypothesis has driven a decades-long effort, spanning multiple disciplines, to deduce how protein sequence determines a proteins structural and functional properties (Dill et al., 2008; Anfinsen, 1977). As the number of proteins with known sequence continues to outpace the number of experimentally determined secondary and tertiary structures (Huang, Liu, & Weinberger, 2016), computational approaches to protein structure prediction become increasingly desirable. Computational tools that can handle large amounts of data while making sufficiently accurate predictions of secondary structures can potentially serve to mitigate the cost and time burden in the experimental determination of protein structures." Busia and Jaitly (2017)

"There is a natural upper-bound on the maximum possible Q3 accuracy from inconsistencies in secondary structure assignment likely due to the coarseness and inherent arbitrariness of three-class labels (Kihara, 2005). This is likely also the case for the eight-class instantiation of the problem. A better approach might be to predict the sequence of back-bone angles for the amino acids, since these are experimentally observed values." Busia and Jaitly (2017)

"Accurately and reliably predicting structures, especially 3D structures, from protein sequences is one of the most challenging tasks in computational biology, and has been of great interest in bioinformatics [Ashraf and Yaohang, 2014]. Structural understanding is not only critical for protein analysis, but also meaningful for practical applications including drug design [Noble et al., 2004]. Understanding protein secondary structure is a vital intermediate step for protein structure prediction as the secondary structure of a protein reflects the types of local structures (such as 310helix and bridge) present in the protein. Thus, an accurate secondary structure prediction significantly reduces the degree of freedom in the tertiary structure, and can give rise to a more precise and high resolution protein structure prediction [Ashraf and Yaohang, 2014; Zhou and Troyanskaya, 2014; Wang et al., 2011]." Li and Yu (2016)

1.4.1 Features

"The input features we used are the same as (Busia and Jaitly, 2017). The amino acid at position i is represented as one-hot vector. There are 20 different types of amino acids. Some special amino acid cases were handled as follows: Amino Acid X was treated as amino acid A. B was treated as amino acid N. Z was treated as amino acid Q. Since the input size was fixed at 700, if the input protein sequences were less than 700 amino acids; the remaining amino acid positions were padded with the NoSeq label, which means no amino acid was there. Current implementation can handle any protein sequence length less or equal to 700 amino acids. The protein sequences will be split into smaller segments if they have more than 700 amino acids. [...] The profile information is also represented as 700-by-21 array. Hence, the input dimension is 700-by-42 in total." Fang et al. (2017b)

"each amino acid (ai) of the sequence (with n amino acids) has 42 features: 22 profile features are used for representing scores obtained from a Position-Specific Scoring Matrix (PSSM) [21], which are normalized using the logistic Sigmoid function. The last 20 features are used for representing the amino acid, using the one-hot encoding. In this encoding, only one bit is set and represents the name category of the amino acid." Hattori et al. (2017)

"amino acid was encoded using one-hot encoding and additionally a sequence profile." Jurtz et al. (2017)

"the first set of predictors take seven representative physio-chemical properties (PP) of amino acids (Fauche‘re et al., 1988), 20-dimensional Position Specific Substitution Matrices (PSSM) from PSI-BLAST (Altschul et al., 1997), and 30-dimensional hidden Markov Model sequence profiles from HHBlits (Remmert et al., 2012) per residue as input [...] These features were selected previously in the development of SPIDER for secondary structure and contact prediction (Lyons et al., 2014; Heffernan et al., 2016, 2015). [...] The outputs from iteration 1 are then added to the PSSM, PP and HMM profile features, as inputs to a second set of predictors (iteration 2). This process is repeated two more times (iterations 3 and 4), for a total of four sets of networks as the result converges. [...] The TR4950 set is split into the same 10 folds for training every iteration of the network." Heffernan et al. (2017)

"Each protein is represented as a sequence of amino acids, padded if necessary by no-sequence tokens to a sequence length of 700 residues. In turn, each amino acid is encoded as a 42-dimensional vector: the first twenty-one dimensions represent the one-hot encoding of the residues identity (or the no-sequence token), while the remaining dimensions contain Position-Specific Scoring Matrices (PSSM) generated with PSI-BLAST (see Zhou&Troyanskaya (2014) for more details) which we normalize via mean-centering and scaling by the standard deviation." Busia and Jaitly (2017)

"Overtime, scientists have reached a consensus that a proteins structure primarily depends on its amino acid sequence and concluded that the local and long-range interaction are a cause of protein second and tertiary structure. Based on this hypothesis, we can deduce that proteins with similar amino acid sequence tend to have similar secondary structure sequence. Therefore, the common sequence information contained by PSSM can contribute to the secondary structure prediction. For a protein with length L, PSSM is usually represented as a matrix with L 21 dimensions where 21 denotes the 20 standard types of residues and one extra residue type which represents all non-standard residue types. Before PSSMs are used in- puts for CNNH_PSS, they need to be transformed to 01 range by the sigmoid function. By concatenating sequence features and evolutional information, each residue in protein sequences can be encoded by a feature vector with dimension of 42." Zhou et al. (2018)

1.4.1.1 Beyond one-hot

"the input feature sequence X is decomposed into two parts, one is a sequence of 21-dimensional feature vectors encoding the types of the amino acids in the protein and the other is a sequence of 21-dimensional profile features obtained from the PSI-BLAST [Altschul et al., 1997] log file and rescaled by a logistic function [Jones, 1999]. Note that each feature vector in the first sequence is a sparse one-hot vector, i.e., only one of its 21 elements is none-zero, while a profile feature vector has a dense representation. In order to avoid the in- consistency of feature representations, we adopt an embed- ding operation from natural language processing to transform sparse sequence features to a denser representation [Mesnil et al., 2015]. This embedding operation is implemented as a feedforward neural network layer with an embedding matrix, Wemb R21Demb, that maps a sparse 21-dimensional vector into a denser Demb-dimensional vector. In this paper, we empirically set Demb = 50, and initialize the embedding matrix with random numbers. The embedded sequence feature vector is concatenated with the profile feature vector before being fed into multiscale CNN layers." Li and Yu (2016)

"As the dimension of amino acid representation is already low, we only calculate a real value for every dimension by embedding technique and dont decrease the dimension. The residue embedding in this paper is implemented by a feedforward neural net- work layer before multi-scale CNN in CNNH_PSS [42]." Zhou et al. (2018)

”The protein sequence input is one-hot vector, which is a sparse matrix. In the future work, ProtVec (Asgari and Mofrad, 2015) will be explored to represent the sequence so that it will become a much denser and picture-like matrix, which may improve the Q3 and Q8 accuracy.” Fang et al. (2017b)

1.4.1.2 PSI-BLAST profiles

”PSI-BLAST generates a PSSM of size T 20 for a T lengthed sequence, where a higher score represents a higher likelihood of the ith amino acid replacing the current one in other species. Generally, two amino acids that are interchangeable in the PSSM indicates that they are also interchangeable in the protein without sig- nificantly modifying the functionality of the protein.” Lin et al. (2016)

”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 run- ning 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).” Jurtz et al. (2017)

”DeepCNF has no significant advantage over the others when a protein under prediction has very sparse sequence profile (i.e., Neff 2). That is, it is still challenging to predict SS structure from primary sequence instead of sequence profile.” Wang et al. (2016)

”We used the input features described in36. In particular, for each protein sequence, we ran PSI-BLAST41 with E-value threshold 0.001 and 3 iterations to search UniRef9079 to generate the position specific scoring matrix (PSSM). We then transform PSSM by the sigmoid function 1/(1 + exp(x)) where x is a PSSM entry. We also use a binary vector of 21 elements to indicate the amino acid type at position i. We use 21 since there might be some unknown amino acids in a protein sequence. In total there are 42 input features for each residue, 21 from PSSM and the other 21 from the primary sequence. Note that besides using PSSM generated by 3 iterations, we could also add the PSSM generated by 5 iterations into the input features.” Wang et al. (2016)

”A protein sequence is represented as a 700-by-21 array in the system. Next, protein profiles were generated using PSI- BLAST (Altschul et al., 1997), as performed in some previous work (Wang et al., 2016; McGuffin et al., 2000). The PSI-BLAST tool parameters were set as follows: evalue: 0.001, num_iterations: 3, db, and UniRef50 to generate a position-specific scoring matrix (PSSM). PSSM is then transformed by the sigmoid function so that the value is in range (0,1).” Fang et al. (2017b) "profile search time is relatively shorter than current available tools because we used a filtered version of the UniRef50 database. [...] The PSI-BLAST can generate the protein sequence profile in a short time if the search database is small, while a larger search database takes a longer time to get the profile and the prediction accuracy may not always increase. [...] Four different databases were downloaded from UniProt (http://www.uniprot.org/downloads). They are Swiss_Prot (0.08GB), UniRef50 (4.3GB), UniRef90 (12GB) and UniRef100 (25GB). Take the UniRef50 database as an example: This database was used to perform a PSI-BLAST search with the parameter setup of evalue 0.001 and num_iterations 3 on all protein sequences in the JPRED data set. Then, the train-ing set was used to train the Deep3I network, and the test set was used to report Q3, as shown in Table 1. The UniRef50 yielded the best results as a larger database was not needed to yield a better performance." Fang et al. (2017b)

"The PSI-BLAST search database is UniRef50 and evalue is set to 0.001. Four experiments were performed with different num_iteration of PSI-BLAST ranging from 2 to 5. Table 2 shows that too few or too many PSI-BLAST iterations do not yield good profile. The number of iterations of PSI-BLAST should be set to 3." Fang et al. (2017b)

"Based on the information presented thus far, one could assume that the smaller the database, the better the Q3. One might ask: What will happen if an even smaller database is used for a PSI-BLAST search? To find out, we filtered a UniRef50 data set and kept those protein sequences whose length fell between 70 and 3000 thereby forming the shrunk database, UniRef50_shrunk. Some other smaller databases were built, as shown in Table 3. Table 4 shows that the UniRef50_smaller was sufficient for PSI- BLAST to use as a database and saves computing time. But when the database became even smaller, the prediction accuracy started to drop or fail. Another option would be to apply CD-Hit to shrink the database with a lower sequence similarity threshold" Fang et al. (2017b)

"A PSSM, or Position-Specific Scoring Matrix, is a type of scoring matrix used in protein BLAST searches in which amino acid substitution scores are given separately for each position in a protein multiple sequence alignment. Thus, a Tyr-Trp substitution at position A of an alignment may receive a very different score than the same substitution at position B. This is in contrast to position-independent matrices such as the PAM and BLOSUM matrices, in which the Tyr-Trp substitution receives the same score no matter at what position it occurs.

PSSM scores are generally shown as positive or negative integers. Positive scores indicate that the given amino acid substitution occurs more frequently in the alignment than expected by chance, while negative scores indicate that the substitution occurs less frequently than expected. Large positive scores often indicate critical functional residues, which may be active site residues or residues required for other intermolecular interactions.

PSSMs can be created using PSI-BLAST, which finds similar protein sequences to a query sequence, and then constructs a PSSM from the resulting alignment. Alternatively, PSSMs can be retrieved from the NCBI CDD database, since each CD is represented by a PSSM that encodes the observed substitutions in the seed alignments. These CD records can be found either by text searching in Entrez Conserved Domains or by using RPS-BLAST (Reverse Position-Specific BLAST), also known as CD-Search, to locate these domains on an input protein sequence." [https://www.ncbi.nlm.nih.gov/Class/Structure/pssm/pssm$_{vie}$

1.4.2 Targets

"Pauling et al. (1951) proposed the earliest concept of protein secondary structure determining that the poly-peptide backbone contains regular hydrogen-bonded geometry, forming -helices and -sheets." Fang et al. (2017b)

"Protein secondary structure (SS) refers to the local conformation of the polypeptide backbone of proteins. There are two regular SS states: alpha-helix (H) and beta-strand (E), as suggested by Pauling13 more than 60 years ago, and one irregular SS type: coil region (C). [...] Overall, protein secondary structure can be regarded as a bridge that links the primary sequence and tertiary structure and thus, is used by many structure and functional analysis tools1518" Wang et al. (2016)

1.4.2.1 Q3

"(ssp) A collapsed version of the 8 class prediction task, since many protein secondary structure prediction algorithms use a 3 class approach instead of the 8-class approach given in dssp. H, G H =Helix, B, E B =Beta sheet, and I, S, T, L C =Coil" Lin et al. (2016)

"The secondary structures (SS) of a protein represent the local conformations of a three-dimensional structure. There are three main secondary structures: -helices [8], -sheets [9] and turns [10]. For instance, proteins 1DG2 and 1KFP represent an helix and an sheet, respectively." Hattori et al. (2017)

"approaching the theoretical limit in the range of 8890% (Rost, 2001)." Heffernan et al. (2017)

"we predict three states by converting the DSSP assigned states G, H, and I to H; B and E to E; and S, T, and C to C." Heffernan et al. (2017)

1.4.2.2 Q8

"Cooperative secondary structure is recognized as repeats of the elementary hydrogen-bonding patterns turn and bridge. Repeating turns are helices, repeating bridges are ladders, connected ladders are sheets. [...] Curved pieces are defined as bends. [...] Pieces of 3- or 5-helix, reduced to less than minimal size due to overlaps, are labeled T. [...] H bonds with n = +3,+4,+5 are reported as turns or helices. [...] 3-Helices are more frequent than previously believed, although they are entirely pure: numerous H bonds in them are bifurcated, i.e., (i,i + 4) and usually short and have mediocre hydrogen bonds. a-Helices are rarely (i,i + 3) or sometimes (i,i + 5). The ends of a-helices often are overwound, ending in a 3-turn or 3-helix, or underwound, ending in a 5-turn. Some of these cases were already noted and generalized by Schellman21 and Richardson.8 We even find a few 5-helices (π-helices)-see Fig. 3. [...] Structure summary: H = 4-helix (α-helix) B = residue in isolated β-bridge E - extended strand, participates in β-ladder G = 3-helix (3_{10}-helix) I = 5-helix (π-helix) T = H-bonded turn S = bend "Kabsch and Sander (1983)

"The class labels are H = alpha helix, B = residue in isolated beta bridge, E = extended strand, G = 3-helix, I = 5-helix, T = hydrogen bonded turn, S = bend, L = loop." Lin et al. (2016)

"The Q8 accuracy is another evaluation metric to evaluate the accuracy of eight-class classification: 310-helix (G), ?-helix (H), ?-helix (I), ?-strand (E), ?-bridge (B), ?-turn (T), bend (S) and loop or irregular (L) (Yaseen and Li, 2014; Zhou and Troyanskaya, 2014)." Fang et al. (2017b)

"Sander14 developed a DSSP algorithm to classify SS into 8 fine-grained states. In particular, DSSP assigns 3 types for helix (G for 310 helix, H for alpha-helix, and I for pi-helix), 2 types for strand (E for beta-strand and B for beta-bridge), and 3 types for coil (T for beta-turn, S for high curvature loop, and L for irregular)." Wang et al. (2016)

"The DSSP program (Touw et al., 2015; Kabsch and Sander, 1983) was used to get the secondary structure label from the PDB files." Fang et al. (2017b)

"Instead of using three classes, [11] group the secondary structures into 8 classes, including others, such as 310-helix, -heliX and -bridge." Hattori et al. (2017)

"The secondary structure of the proteins, which should be predicted, is given as eight different DSSP classes (Q8) (Kabsch and Sander, 1983). The dataset was originally downloaded from PDB and annotated with the DSSP program (Kabsch and Sander, 1983)." Jurtz et al. (2017)

"During training both of these networks mask out the loss contributed by any undefined labels, such as residues with no secondary structure assignment according to the program

Dictionary of Secondary Structure of Proteins (DSSP) (Kabsch and Sander 1983)." Heffernan et al. (2017)

"DSSP (Kabsch and Sander, 1983) specifies eight secondary structure states. Three helix shapes: 310-helix G, alpha-helix H, and pi-helix I; two strand types: beta- bridge B and beta-strand E; and three coil types: high curvature loop S, beta-turn T, and coil C. [...] These secondary structure labels are calculated from their experimentally determined structures." Heffernan et al. (2017)

1.4.2.3 Accuracy

"The Q3 (Q8) accuracy is defined as the percentage of residues for which the predicted secondary structures are correct32." Wang et al. (2016)

"Q3 and Q8 were used as a performance metric, as commonly used in Zhou and Troyanskaya (2014), Wang et al. (2016), Li and Yu (2016), and Busia and Jaitly (2017). The Q3 or Q8 accuracy measured the percentage of residues being correctly predicted among three-state or eight-state protein secondary structure classes." Fang et al. (2017b)

1.4.3 State of the art

1.4.3.1 Initial research

"Protein SS prediction has been extensively studied1012,1935. Many computational methods have been developed to predict both 3-state SS and a few to predict 8-state SS. Meanwhile, 8-state prediction may provide more detailed local structure information33,34,36. Holley & Karplus19 and Qian & Sejnowski20 may be the first that used neural networks (NN) to predict SS, which have been followed by a few others19,21,23,24,37. The most significant improvement in SS prediction was achieved by Rost & Sander23 and Zvelebil et al.35 by making use of sequence profile derived from multiple sequence alignment3840. Jones et al.24 developed a 2-stage neural network method PSIPRED, which takes PSI-BLAST sequence profile41 as input and obtains 80% accuracy for 3-state SS predic- tion. Other machine learning methods include bidirectional recurrent neural networks26,34,37 (which can capture spatial dependency), probabilistic graphical models25,29,42, support vector machines27,28,30,43 and hidden Markov models22,31." Wang et al. (2016)

"In the 1980s, the Q3 accuracy was below 60% due to the lack of input features. In the 1990s, the Q3 accuracy reached above 70% because of using the protein evolutionary information in the form of position-specific score matrices. Since then, the Q3 accuracy has gradually improved to above 80%." Fang et al. (2017b) "there is no consensus about the classification task which can be done using different properties of proteins. For instance, early approaches were based on stereochemical principles [12], statistics [13] and Position-specific Scoring Matrices (PSSM) [14]. More recently, [3] used the Kyte and Doolittle (K & D) hydrophobicity scale in order to represent the aminoacids of proteins. From the approach point of view, [3] present the application and comparison of Machine Learning and Evolutionary Computation methods to define suitable classifiers for predicting the secondary structure of proteins, such as Gene expression programming (GEP), Sequential Minimal Optimization algorithm (SMO) and RandomForest (collection of tree-structured classifiers). [15]" Hattori et al. (2017)

"The accuracy of predicting protein local and global structural properties such as secondary structure and solvent accessible surface area has been stagnant for many years because of the challenge of accounting for non-local interactions between amino acid residues that are close in three-dimensional structural space but far from each other in their sequence positions. All existing machine-learning techniques relied on a sliding window of 1020 amino acid residues to capture some short to intermediate non-local interactions. [...] Commonly used simple Artificial Neural Networks (ANNs) (Faraggi et al., 2012), and Deep Neural Networks (DNNs) (Lyons et al., 2014; Heffernan et al., 2016, 2015), relied on their input windows of neighbouring residue information, which means that they are unable to fully learn the relation- ship between the residues in the whole sequence." Heffernan et al. (2017)

"Applications of machine learning to the protein secondary structure problem have a rich history: the use of neural networks for secondary structure prediction was pioneered by Qian and Sejnowski in 1988," Busia and Jaitly (2017)

"The study of protein secondary structure prediction dates back to 1970s. In the 1970s, statistical models were frequently used to analyze the probability of specific amino acids appearing in different secondary structure elements [Chou and Fasman, 1974]. The Q3 accuracy, i.e., the accuracy of three-category classification: helix (H), strand (E) and coil (C), of these models was lower than 60% due to inadequate features. In the 1990s, significant improvements were achieved by exploiting the evolutionary information of proteins from the same structural family [Rost and Sander, 1993] and position-specific scoring matrices [Jones, 1999]. During this period, the Q3 accuracy exceeded 70% by taking advantage of these features. However, progress stalled when it came to the more challenging 8-category classification problem, which needs to distinguish among the following 8 categories of secondary structure elements: 310helix (G), helix (H), helix (I), strand (E), bridge (B), turn (T), bend (S) and loop or irregular (L) [Zhou and Troyanskaya, 2014; Yaseen and Li, 2014]." Li and Yu (2016)

1.4.3.2 Recently

”Zhou & Troyanskaya36 reported another deep learning approach to 8-state SS prediction using a supervised generative stochastic network, which to our best knowledge may be the best 8-state predictor. However, neither Cheng nor Zhou reported a better than 80% accuracy for 3-state prediction. SS prediction is usually evaluated by Q3 or Q8 accuracy, which measures the percent of residues for which 3-state or 8-state secondary structure is correctly predicted44. So far the best Q3 accuracy for ab initio prediction (i.e., templates are not allowed) is 80% obtained by PSIPRED and a few other state-of-the-art approaches such as JPRED47,48” Wang et al. (2016)

”In the 21-st century, various machine learning methods, especially artificial neural networks, have been utilized to improve the performance, e.g., SVMs [Sujun and Zhirong, 2001], recurrent neural networks (RNNs) [Pollastri et al., 2002], probabilistic graphical models such as conditional neural fields combining CRFs with neural networks [Wang et al., 2011], generative stochastic networks [Zhou and Troyanskaya, 2014].” Li and Yu (2016)

1.4.3.3 Hall of fame

Names	**Q8**	**Architecture**	**Date**
[Wang et al., 2011]	64.9%	CNF	2011
Zhou and Troyanskaya (2014)	66.4%	**CGSN**	2014
Magnan and Baldi (2014)	66.5%	SSPro (BRNN)	2014
Sønderby and Winther (2014)	67.45%	LSTM	2014
Wang et al. (2016)	68.3%	**DeepCNF**	Jan 2016
Li and Yu (2016)	69.7%	**DCRNN**	Apr 2016
Lin et al. (2016)	68.4%	**MUST-CNN**	May 2016
Busia and Jaitly (2017)	71.4%	**Deep3I+conditioning**	Feb 2017
Hattori et al. (2017)	68.0%	DBLSTM	May 2017
Jurtz et al. (2017)	70.2%	**LSTM+CNN**	Aug 2017
Johansen et al. (2017)	70.9%	BGRU-CRF	Aug 2017
Fang et al. (2017a)	71.1%	**DeepNRN**	Nov 2017
Zhou et al. (2018)	70.3%	**CNN+highway**	Jan 2018
Fang et al. (2017b)	70.63%	**MUFold-SS (Deep3I)**	Feb 2018
Mine	67.7%	3-layers ResNet	Aug 2018

A good part of the winning architectures are actually using CNNs, either alone, or in combinations with other methods.

Chapter 2

Methods

Most of the computation work is performed on the *IRIDIS High Performance Computing Facility*, with help of associated support services at the university. They make use of NVIDIA® GeForce GTX 1080 Ti Graphical Processing Units (*GPU*s). The use of the GPUs has been possible thanks to the CUDA 9.0 toolkit[1].

2.1 Dataset

For the purpose of this project, the database produced and made public by Zhou and Troyanskaya (2014) is used[2]. This database has been taken as the flagship benchmark since its release and making use of it allows a fair comparisons with most state-of-the-art algorithms[3]. This dataset includes two sub-sets (training and test) of proteins that come from different sources in order to ensure that the test set is composed by completely new samples. For this same reason, they filtered the training set, stripping out every sequence that held 25% or more similarity with any protein from the test set. The training set was obtained from PISCES server (Wang and Dunbrack (2003))[4] from a date before January 2014, which was in time culled from the Protein Data Bank (PDB) (Berman et al. (2003)). It has an original size of 6133 proteins and 5534 after the filtering. The test set comes from the CB513 dataset Cuff and Barton (1999) and includes 514 sequences[5].

Protein sequences can be up to 700 amino-acids long (with an average of about 214) and have already been pre-processed by Zhou and Troyanskaya (2014), with one-hot encoded inputs for the 21 one types of amino-acid along with a 21-long vector of the *pssm* values

[1] `https://developer.nvidia.com/cuda-toolkit`

[2] It can be accessed at `https://www.princeton.edu/~7Ejzthree/datasets/ICML2014/`

[3] See section 1.4.3.3

[4] `http://dunbrack.fccc.edu/Guoli/PISCES_OptionPage.php`

[5] Originally 513, but the last one was split in two since it was larger than the 700 amino-acids limit.

and their one-hot encoded secondary structure Q8 classes. The appearance frequencies of the eight classes in both datasets is shown in Figure 2.1.

A sub-set of 256 proteins were taken out of the training set and used for validation, leaving 5278 proteins for the training. This splitting is common in previous papers (Zhou and Troyanskaya (2014); Sønderby and Winther (2014); Busia and Jaitly (2017); Jurtz et al. (2017); Hattori et al. (2017)).

FREQUENCY DISTRIBUTION

Freq. Train (%)	Freq. Test (%)	Class	
34.535	30.85	H	(α-helix)
21.781	21.25	E	(β-strand)
19.185	21.14	L	(loop)
11.284	11.81	T	(β-turn)
8.258	9.81	S	(bend)
3.911	3.69	G	(3_{10}-helix)
1.029	1.39	B	(β-bridge)
0,018	0.03	I	(π-helix)

TABLE 2.1: Target frequencies on the CB6133 (left) and the CB513 (right). Table reproduced from Hattori et al. (2017).

2.2 Network of study

The interpretability methods could be applied to any state-of-the-art network, since these would provide the most refined information about the secondary structure problem. However, in order to reach peak accuracies, these models include incredibly huge models that are not handy to work on (especially when it comes to computational time). For this reason, a simpler network is produced and employed for this analysis, with the hope that it will not lose generality over more complex networks.

The network that is used in this study has been built and trained using the open-source code developed by Jurtz et al. (2017)[6] on the *Lasagne* framework (Dieleman et al. (2015)). While most training configurations are preserved (cross-entropy with L2 regularization as error function, batch normalization, uniform glorot initialization, mini-batches of size 64, RMSprop rule updates, gradient clipping), the network architecture itself is completely rebuilt. The weights for the final model are the ones from the epoch that has the best performance on the validation set.

The network is composed by three successive convolutional networks and a dense layer on top. Each of the convolutional layers contains three sets of filters of size 3, 5 and 7, respectively, with 16 filters per size. There are skip connections that by-pass every convolutional network in the same fashion as *ResNet*(He et al. (2015)), so the dense layer gets the concatenation of the raw input along with the outputs from the first,

[6]Accesible at `https://github.com/vanessajurtz/lasagne4bio`.

second and third layer altogether. The dense layer has 200 neurons and is connected to a *soft-max* layer that gives the output (secondary structure prediction).

The total window size for this network is 19, meaning that for making a single secondary structure classification the network obtains information from 9 adjacent positions at each side. Although some authors claim that the network should capture long interactions between amino-acids (Li and Yu (2016); Lin et al. (2016); Hattori et al. (2017); Heffernan et al. (2017)), it has been proven that most relevant information comes from the local environment (Busia and Jaitly (2017)), so the limited window size should not impinge the model. This fact also supports the idea that the skip connections are beneficial since they avoid the most local information to be "washed out" in upper layers (Busia and Jaitly (2017)).

2.3 Feature visualization

First order filters. They only show the first layer of feature extraction. Optimization maximization. Bound to give unreal data. Not credible priors.

2.4 Saliency maps

Saliency maps have been calculated by the common technique of computing the gradient of the output with respect to the inputs and multiplying it by the value of the input itself (Shrikumar et al. (2016)). A major difference between this work and most previous papers that make use of saliency maps is that they perform many-to-one classification (one output class per sequence / image), here the classification task is many-to-many (each position of the sequence is assigned a class), producing many saliency maps for a single sequence. More specifically, each single position produces a saliency map that contains the influence of the 42-size input (21 amino-acids plus 21 *pssm* scores) onto the 8-size soft-max output. This information covers the 19 positions surrounding the one being predicted, ending up with a saliency map with total dimension 8x42x19. They are computed using Theano framework (The Theano Development Team et al. (2016)) since it allows automatic differentiation of symbolical expressions and the use of GPUs.

2.4.1 Extracting information from saliency maps

The presence of overlapping saliency maps allows for different ways of aggregating them and extracting meaningful information. In order to obtain information for a specific sequence, the overlapping saliency maps of such sequence could just be added up, resulting in a single, long *sequence-specific* saliency map of size 8x42xl.

By changing the focus to general information of the network behaviour, the sheer addition of all saliency maps produces a single 8x42x19 map with an average behaviour. From this map we could extract information about the general behaviour around a certain class (*class-specific* saliency map) or a certain amino-acid (*pssm-specific* saliency map). If a more fine-grained inspection is desired, clustering techniques can be used and every cluster create their independent 8x42x19 representative map by addition of all their components.

2.5 Open-source

As it is the common practice in both the bioinformatics and machine learning research communities, all the code from this project has been released as open-source in the web platform GitHub and can be accessed through the url `https://github.com/Juillermo/lasagne4bio`, with the hope that the tools here developed can be of use for future research.

Chapter 3

Results & Discussion

I have trained the network described in section 2.2 for 400 epochs with the same parameters as the ones used by Jurtz et al. (2017); i.e., gradient clipping at 20, regularization term $\lambda = 10^{-4}$, learning rate varying from $\mu = 0.025$ down to 10^{-4} throughout the epochs. The resulting network reaches an accuracy of 67.7% on the test set, which is not far from the 71% of the state-of-the-art (see section 1.4.3.3). I believe that the techniques of analysis here presented and the conclusions withdrawn from them can be transferred to current state-of-the-art methods without losing validity.

3.1 Outlier analysis

In order to analyse the performance space a bit better, the average accuracy per sequence has been calculated and it has been plotted in Figure 3.1 with respect to the sequence length. The distribution exhibits the typical funnel shape that one could expect from processes with random variables forming groups of different sizes: the bigger the groups, the smaller the variance. The funnel ceases to shrink at length about 400, so it would be particularly interesting to understand why the network is classifying worse (60% and below) some of the sequences above that length.

If we observe the colour scheme of the figure, we can understand right away that sequences rich in α-helix are generally better predicted than β-sheets and coils. An explanation could be that while α-helix sizes are up to five positions away, which is inside the window size, β-sheets interact with amino-aids further away in the sequence, which is not possible to be captured with the window of the network, of lateral size of nine.

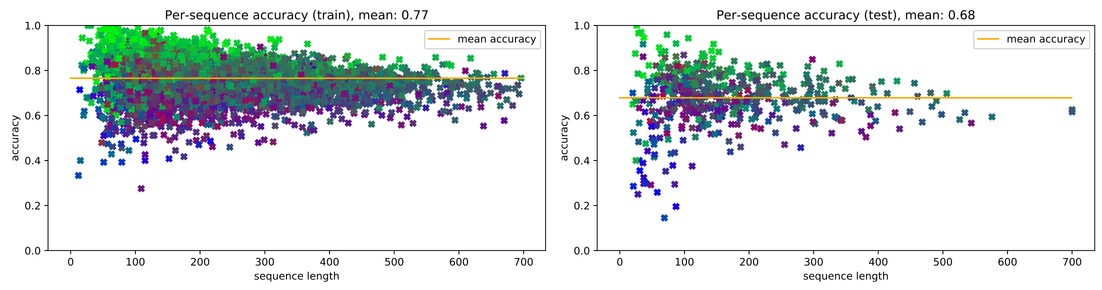


FIGURE 3.1: *The mean accuracy per sequence by sequence length.* The 5504 sequences of the training set are shown on the left and the 514 of the test set on the right. Each point represents a single protein, and its colour corresponds to the amount of β-sheets (red), α-helices (green), and coils (blue) it has. A purple point, for instance, would predominantly have β-sheets and coils.

3.2 Feature visualization

3.2.1 First layer filters

Saliency maps on layers?

3.3 Saliency maps on inputs

Before going through the analysis, it is worth recalling that the saliency map outputs has dimensions 8x42x19, for each position at each sequence, corresponding to the 8 classes (outputs), the 42 inputs and and the total window size of 19 (as explained in Section 2.4).

Analysis on amino-acids and *pssm*

When looking at typical secondary-structure prediction algorithm, there is one point that may raise some suspicion: the inclusion of half of the inputs as one-hot encoded (amino-acids) and the other half as dense vectors (*pssm*). One could thing that this discrepancy may strongly favour the information coming from the dense part, since the weights associated to it will learn much faster in a typical gradient descent learning schema.

Saliency maps can be used to prove whether this hypothesis is right by inspecting which of the input groups is being most decisive in the classification process. For doing so, each saliency map is divided into two groups of $8x21x19$, one corresponding to the values for one-hot amino-acids and the other for the *pssm* inputs, and all the values inside each group are added up in absolute value to a single **saliency score**. Thus, each position of each sequence will have two scores, one for the amino-acids and one for the *pssm*, and

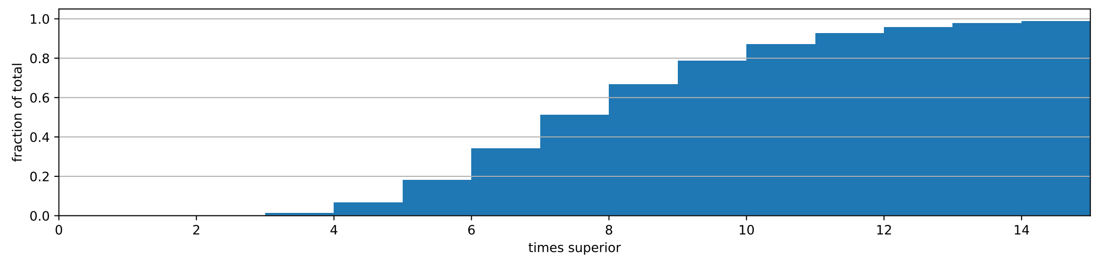


FIGURE 3.2: *Pssm saliency superiority.* This figure shows a cumulative histogram of the number of positions whose *pssm* saliency score was higher than the one-hot score. The x axis represent the superiority ratio, i.e. how many times bigger was the score.

its comparison will give us which part of the inputs is more decisive and quantifies this difference. Only a single case out of the close to 1.3 millions positions showed a higher one-hot than *pssm* score, and the different only accounted for 1% of their value. The rest of the scores were in favour of the *pssm* saliency scores, with the distribution shown Figure 3.2. From the figure we can know that almost the totality of the *pssm* scores were at least four times bigger than the one-hot ones, and therefore we can solidly confirm that the influence of the one-hot data on the input is minor and its omission would not cause much loss in the performance of the network. Consequently, further results focus only on the effect of the *pssm* on classification.

3.3.1 Sequence-specific saliency maps

This section will present the sort of analysis that can be done for a specific sequence. This can be specially useful for analysing the sequences whose accuracy was remarkably lower (outliers). Each sequence has a number of saliency maps equal to its length, l, and they can either be inspected one by one or be overlapped through the sequence, obtaining a single 8x42xl **sequence map**.

Figure 3.3 shows consecutive saliency maps belonging to a single class. The patterns they present are generally preserved (note the differences in scales), just being shifted by one position on the window axis. This suggests that the algorithm is not differentiating that much between specific amino-acid positions, but rather looking for them to be in the vicinity. For this reason, we can consider that overlapping them in a single sequence map does preserve most of the information, as it is shown in Figure 3.4. This sort of map reveals which amino-acids of the vicinity are mostly responsible for a prediction in a particular position. For instance, the predictions of class E on the left side of the shown fragment have to do with the presence of amino-acids A, V and L, while the failure to predict class H on position 109 is likely related to the presence of amino-acid D at position 110. Note that these saliency values come from the aggregation of all the 42 inputs and not only from the one-hot encoded side, which explains why positions 105 and 106 have different values in spite of having the same amino-acid (D). It is specially

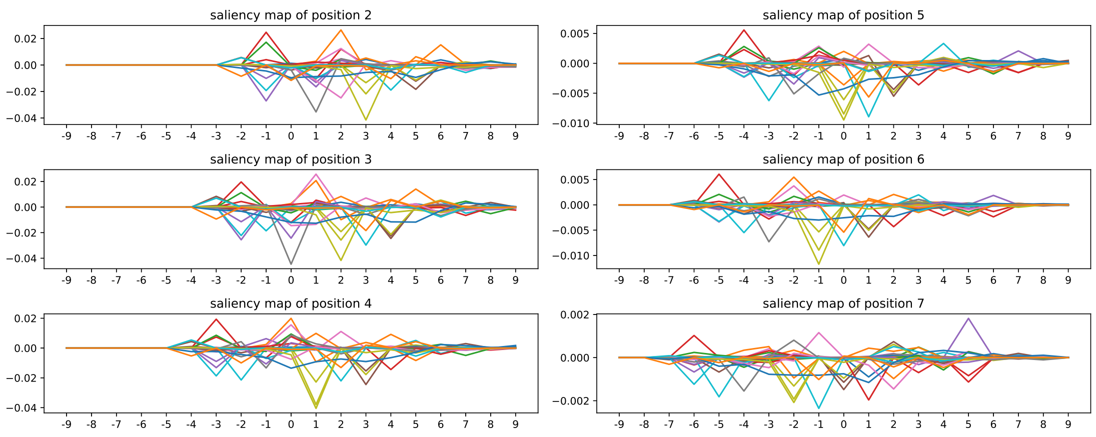


FIGURE 3.3: *Saliency maps for consecutive positions in a sequence.* In every sub-figure there are 42 lines of the 42 inputs, corresponding to their saliency values for class H.

important to take this into account since we have discussed before that one-hot amino-acid information is not as decisive for making the predictions.

Another issue worth noting is a non-realistic inconsistency among predictions: the algorithm assigns labels individually per-position and does not take into account the labels it is providing at the sides. This leads to situations such as having individual Gs, when these only come at least in groups of three. Different approaches in the state-of-the-art for dealing with this are adding a *struct2struct* machine on top of the predictions Rost and Sander (1993); Fang et al. (2017b), iterative processes that learn from previous step's prediction Heffernan et al. (2017), or probabilistic methods that convey information about adjacent positions, like *Conditional random fields* Wang et al. (2016) or *next-step conditioning* Busia and Jaitly (2017).

For a deeper look into the whole *pssm* spectrum, we would need to narrow the scope down to a single class, as it is done in Figure 3.5. This figure reveals that is indeed common that the real amino-acid in the sequence is not among the most influential ones from the *pssm* for making the decision. Note that the pssm values of the amino-acids not always need to have contributions of the same sign (such as P or D in the figure), revealing that the network is capturing something more than pure presence: location and combinations of amino-acids are also important.

3.3.2 Class-specific and pssm-specific saliency maps

Instead of looking at the saliency information along single sequences, aggregating the information from all the 1.3 millions saliency maps could give us some broader information of what the network has learned. Again, the high dimensionality of the data to explore requires us to seek for meaningful ways to aggregate and represent the data.

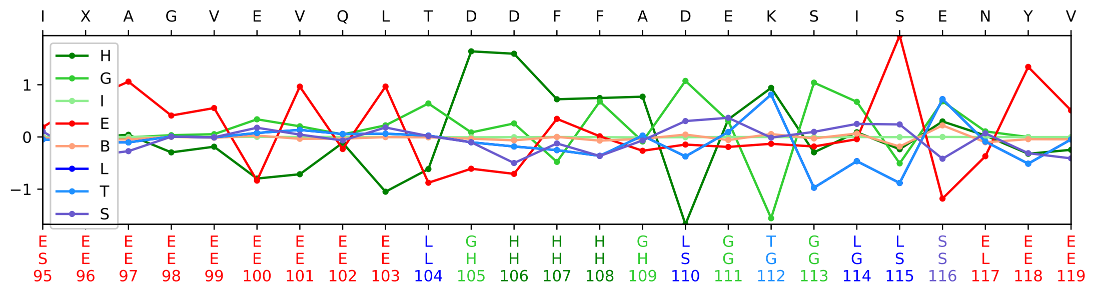


Figure 3.4: *Fragment of a sequence map aggregated by input.* Each line corresponds to the aggregated saliency values of one of the eight output classes. The upper x axis displays the amino-acids in the sequence. The labels at lower x axis contains three sets of labels in this order: the predictions, the true values, and the positions in the sequence. The code of colours of the labels is the same as the lines and is set according to the class of the predictions.

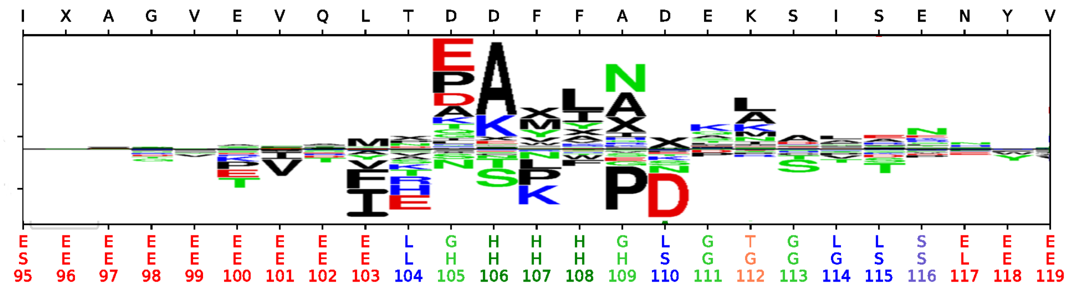


Figure 3.5: *Fragment of sequence map for class H in SeqLogo form.* The saliency of the *pssm* values is shown here. The frame and labels of the figure is the same as in Figure 3.4. The image has been generated by *SeqLogo* Thomsen and Nielsen (2012).

Sheer addition

A simple way to aggregate the saliency maps would be adding them up with element-wise addition, thus composing a single saliency map of size 8x42x19. This mega-saliency map should convey rough information of the general features of the network. Since there is no a sense of “left” or “right” in the sequences, the aggregation should be *side-invariant* and therefore an extra *mirroring* operation is applied to the aggregation in order to make the aggregated saliency map symmetric along the centre of the window.

To start with, Figure 3.6 shows the aggregated class-specific saliency maps for the three main classes and the 21 *pssm* values. There are a few details to notice here. First, higher saliency values concentrate around the centre of the window, meaning that most relevant information is located around 5 positions from the predicted one. Second, most *pssm* values preserve the same sign along the window of a class, although a few exceptions also occur. Third, as it could be expected, class L generally presents inverse patterns with respect to classes H and E. This makes sense since coils appear wherever there are no α-helices or β-strands, they are complementary. Lastly, each class has a “feature” *pssm* value that is their best indicator: A for class H; V for E; and P for L.

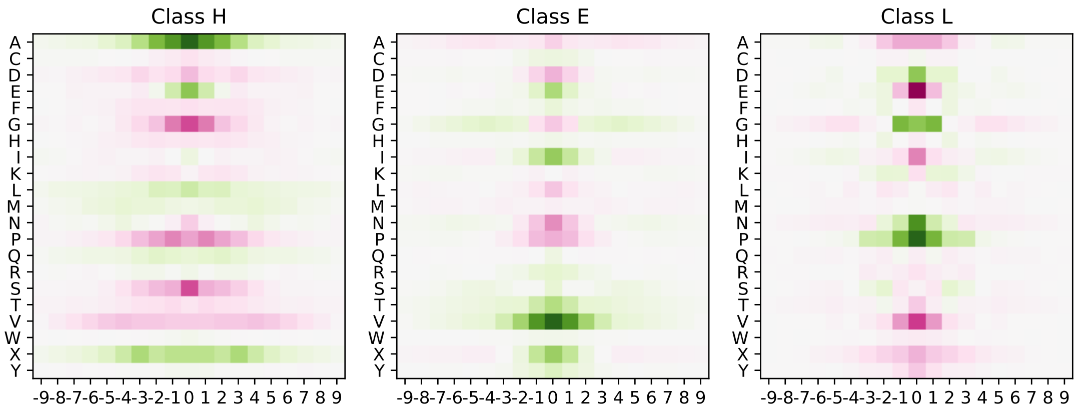


FIGURE 3.6: *Per-class aggregated saliency maps of the main three classes along the window.* Only the saliency values of *pssm* are shown here. Green means positive saliency values and purple means negative.

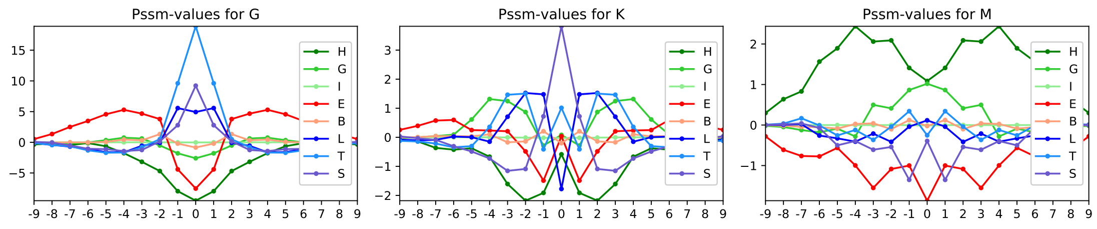


FIGURE 3.7: *Per-class aggregated saliency map of amino-acids G, K, and M, in this order.* Y axis is in 1000s. Note the change in scale, particularly for the left-most plot (amino-acid G).

Figure 3.7 shows the same information but from a different perspective: as it is *pssm-specific*, it focuses on how a single *pssm*'s position affects the classification on the eight classes. These plots show important effects of the position, as all of them make significant shifts at the centre of the window. *Pssm* value G strongly favours coils at its spot, but foresees the presence of β-strands a few positions away. *Pssm* value K has a somehow similar behaviour, although with a more intricate ladder of classes as you move further away from the centre. *Pssm* value M is generally favouring α-helices, although 3_{10}-helices gain equal importance at the very centre. Note the difference in scales, which indicates that *pssm*-values of G are in any case more influential than the other two.

We can use the aggregated saliency maps to investigate on the span of the class influence, i.e. how much of the further away positions the network uses to predict the class. For performing this analysis the addition of the saliency maps must be done in absolute value and then aggregated by the *pssm* dimension as well. Figure 3.8 reveals the results for this operation. α-helices (H) and β-bridges (B) seem to have the highest side influence, although β-strand's E influence gets more weight for further positions. Most classes show a quasi-linear descend on influence from one to five-positions away and some tail afterwards. Coils are the most centred classes.

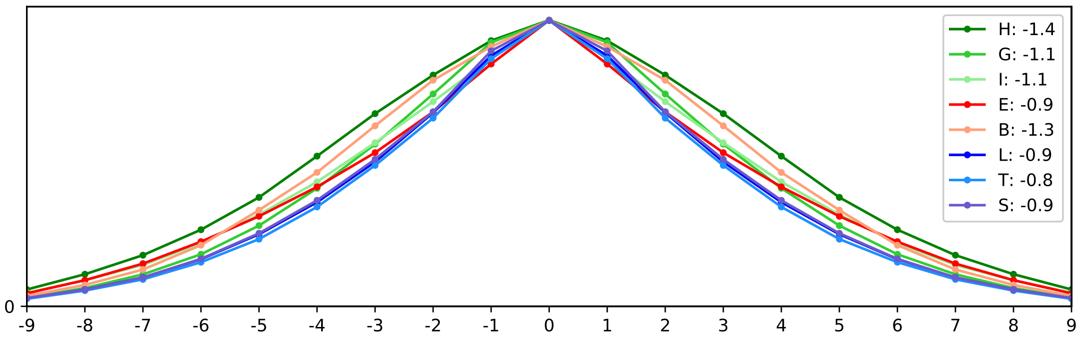


FIGURE 3.8: *Classes overall influence along the window size.* All lines are normalized to have the same hight , as the analysis focuses on the shapes rather than the values. Labels include a measurement of the *kurtosis* of the distributions.

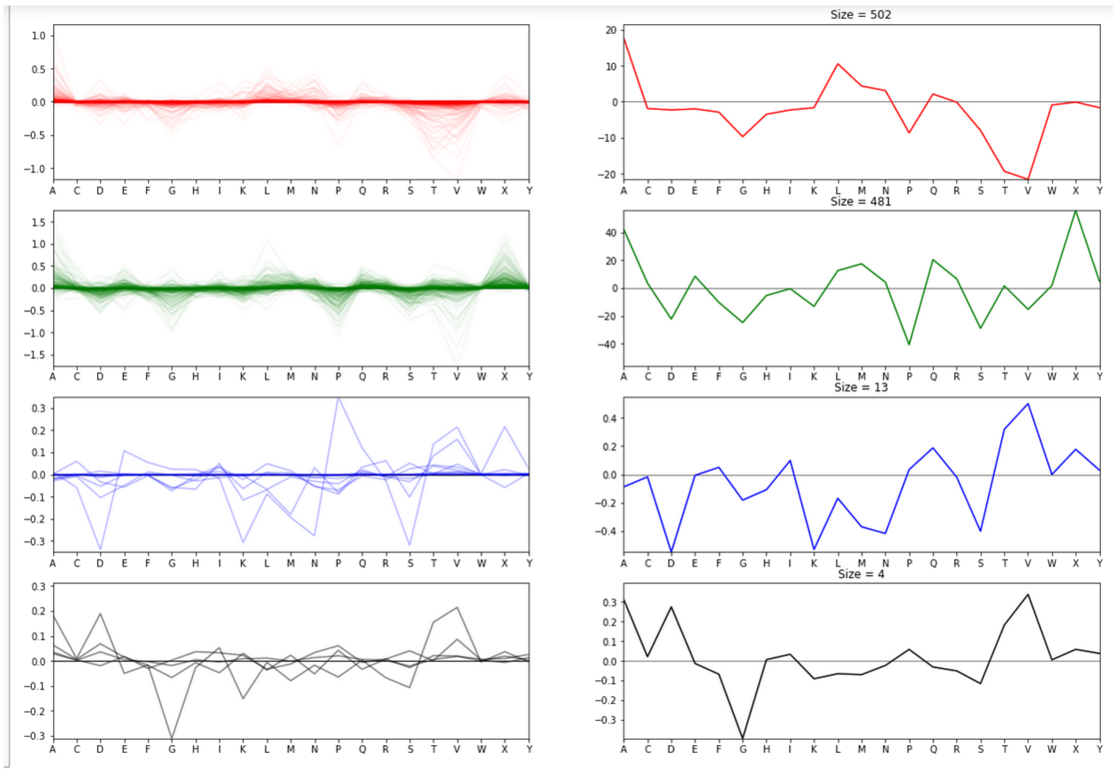


FIGURE 3.9:

Clustering techniques

aggregate all individual saliency maps

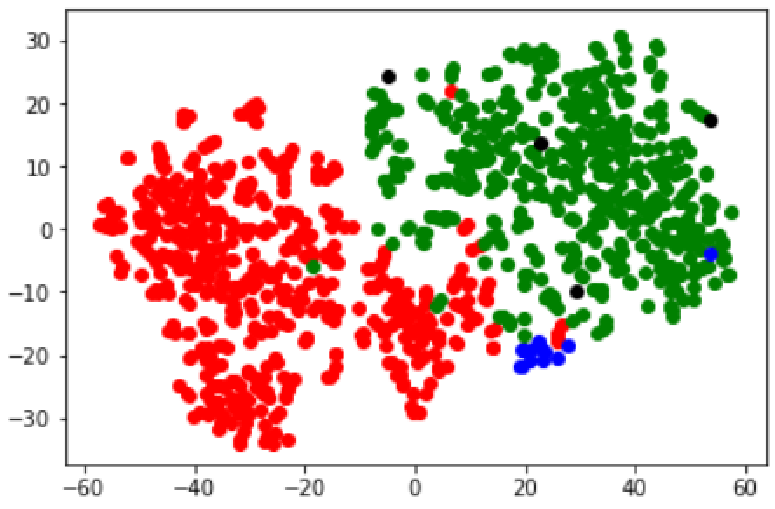


FIGURE 3.10:

Bibliography

Helen Berman, Kim Henrick, and Haruki Nakamura. Announcing the worldwide Protein Data Bank, 2003. ISSN 10728368.

Akosua Busia and Navdeep Jaitly. Next-Step Conditioned Deep Convolutional Neural Networks Improve Protein Secondary Structure Prediction. *arXiv:1702.03865v1*, 2017.

Travers Ching, Daniel S. Himmelstein, Brett K. Beaulieu-Jones, Alexandr A. Kalinin, Brian T. Do, Gregory P. Way, Enrico Ferrero, Paul-Michael Agapow, Wei Xie, Gail L. Rosen, Benjamin J. Lengerich, Johnny Israeli, Jack Lanchantin, Stephen Woloszynek, Anne E. Carpenter, Avanti Shrikumar, Jinbo Xu, Evan M. Cofer, David J. Harris, Dave DeCaprio, Yanjun Qi, Anshul Kundaje, Yifan Peng, Laura K. Wiley, Marwin H. S. Segler, Anthony Gitter, and Casey S. Greene. *Opportunities And Obstacles For Deep Learning In Biology And Medicine.* 2017. ISBN 0000000305396.

J a Cuff and G J Barton. Evaluation and improvement of multiple sequence methods for protein secondary structure prediction. *Proteins*, 1999. ISSN 0887-3585.

Sander Dieleman, Jan Schlüter, Colin Raffel, Eben Olson, Søren Kaae Sønderby, 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., 2015.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *Bernoulli*, 2009.

C. Fang, Y. Shang, and D. Xu. A new deep neighbor residual network for protein secondary structure prediction. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 66–71, Nov 2017a.

Chao Fang, Yi Shang, and Dong Xu. MUFold-SS: Protein Secondary Structure Prediction Using Deep Inception-Inside-Inception Networks. sep 2017b.

Alejandro Fontal. *Neural Networks for Subcellular Localization Prediction.* PhD thesis, Wageningen University & Research, 2017.

Leandro Takeshi Hattori, Cesar Manuel Vargas Benitez, and Heitor Silverio Lopes. A deep bidirectional long short-term memory approach applied to the protein secondary structure prediction problem. In *2017 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, pages 1–6, 2017. ISBN 978-1-5386-3734-0.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. ResNet. *arXiv preprint arXiv:1512.03385v1*, 2015. ISSN 1664-1078.

Rhys Heffernan, Yuedong Yang, Kuldip Paliwal, and Yaoqi Zhou. Capturing non-local interactions by long short-term memory bidirectional recurrent neural networks for improving prediction of protein secondary structure, backbone angles, contact numbers and solvent accessibility. *Bioinformatics*, 33(18):2842–2849, 2017. ISSN 14602059.

Alexander Rosenberg Johansen, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. Deep Recurrent Conditional Random Field Network for Protein Secondary Prediction. In *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology,and Health Informatics - ACM-BCB '17*, 2017. ISBN 9781450347228.

Vanessa Isabell Jurtz, Alexander Rosenberg Johansen, Morten Nielsen, Jose Juan Almagro Armenteros, Henrik Nielsen, Casper Kaae Sønderby, Ole Winther, and Søren Kaae Sønderby. An introduction to deep learning on biological sequence data: Examples and solutions. *Bioinformatics*, 33(22):3685–3690, 2017. ISSN 14602059.

Wolfgang Kabsch and Christian Sander. Dictionary of protein secondary structure: Pattern recognition of hydrogenbonded and geometrical features. *Biopolymers*, 22 (12):2577–2637, 1983. ISSN 10970282.

Jack Lanchantin, Ritambhara Singh, Beilun Wang, and Yanjun Qi. Deep Motif Dashboard: Visualizing and Understanding Genomic Sequences Using Deep Neural Networks. 2016. ISSN 23356936.

Zhen Li and Yizhou Yu. Protein Secondary Structure Prediction Using Cascaded Convolutional and Recurrent Neural Networks. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 2016. ISBN 978-1-57735-770-4.

Zeming Lin, Jack Lanchantin, and Yanjun Qi. MUST-CNN: A Multilayer Shift-and-Stitch Deep Convolutional Architecture for Sequence-based Protein Structure Prediction. may 2016.

Christophe N. Magnan and Pierre Baldi. SSpro/ACCpro 5: Almost perfect prediction of protein secondary structure and relative solvent accessibility using profiles, machine learning and structural similarity. *Bioinformatics*, 2014. ISSN 14602059.

Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015. ISBN 9781467369640.

Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus Robert Müller. Explaining nonlinear classification decisions with deep Taylor decomposition. *Pattern Recognition*, 2017. ISSN 00313203.

Grégoire Montavon, Wojciech Samek, and Klaus Robert Müller. Methods for interpreting and understanding deep neural networks, 2018. ISSN 10512004.

Alexander Mordvintsev, Michael Tyka, and Christopher Olah. Inceptionism: Going deeper into neural networks, google research blog, 2015.

Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature Visualization. *Distill*, 2017. ISSN 2476-0757.

Burkhard Rost and Chris Sander. Prediction of protein secondary structure at better than 70% accuracy. *Journal of Molecular Biology*, 1993. ISSN 00222836.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning Important Features Through Propagating Activation Differences. apr 2017.

Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. Not Just a Black Box: Learning Important Features Through Propagating Activation Differences. *arXiv:1605.01713 [cs]*, 2016. ISSN 1938-7228.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *arXiv.org*, 2014.

Søren Kaae Sønderby and Ole Winther. Protein Secondary Structure Prediction with Long Short Term Memory Networks. *arXiv:1412.7828*, 2014.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic Attribution for Deep Networks. 2017. ISSN 1938-7228.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. dec 2013.

The Theano Development Team, Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, Yoshua Bengio, Arnaud Bergeron, James Bergstra, Valentin Bisson, Josh Bleecher Snyder, Nicolas Bouchard, Nicolas Boulanger-Lewandowski, Xavier Bouthillier, Alexandre de Brébisson, Olivier Breuleux, Pierre-Luc Carrier, Kyunghyun Cho, Jan Chorowski, Paul Christiano, Tim Cooijmans, Marc-Alexandre Côté, Myriam Côté, Aaron Courville, Yann N. Dauphin, Olivier Delalleau, Julien Demouth, Guillaume Desjardins, Sander Dieleman, Laurent Dinh, Mélanie Ducoffe, Vincent Dumoulin, Samira Ebrahimi Kahou, Dumitru Erhan, Ziye Fan, Orhan Firat, Mathieu Germain, Xavier Glorot, Ian Goodfellow, Matt Graham, Caglar Gulcehre, Philippe Hamel, Iban Harlouchet, Jean-Philippe Heng, Balázs

Hidasi, Sina Honari, Arjun Jain, Sébastien Jean, Kai Jia, Mikhail Korobov, Vivek Kulkarni, Alex Lamb, Pascal Lamblin, Eric Larsen, César Laurent, Sean Lee, Simon Lefrancois, Simon Lemieux, Nicholas Léonard, Zhouhan Lin, Jesse A. Livezey, Cory Lorenz, Jeremiah Lowin, Qianli Ma, Pierre-Antoine Manzagol, Olivier Mastropietro, Robert T. McGibbon, Roland Memisevic, Bart van Merriënboer, Vincent Michalski, Mehdi Mirza, Alberto Orlandi, Christopher Pal, Razvan Pascanu, Mohammad Pezeshki, Colin Raffel, Daniel Renshaw, Matthew Rocklin, Adriana Romero, Markus Roth, Peter Sadowski, John Salvatier, François Savard, Jan Schlüter, John Schulman, Gabriel Schwartz, Iulian Vlad Serban, Dmitriy Serdyuk, Samira Shabanian, Étienne Simon, Sigurd Spieckermann, S. Ramana Subramanyam, Jakub Sygnowski, Jérémie Tanguay, Gijs van Tulder, Joseph Turian, Sebastian Urban, Pascal Vincent, Francesco Visin, Harm de Vries, David Warde-Farley, Dustin J. Webb, Matthew Willson, Kelvin Xu, Lijun Xue, Li Yao, Saizheng Zhang, and Ying Zhang. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, may 2016.

Martin Christen Frolund Thomsen and Morten Nielsen. Seq2Logo: A method for construction and visualization of amino acid binding motifs and sequence profiles including sequence weighting, pseudo counts and two-sided representation of amino acid enrichment and depletion. *Nucleic Acids Research*, 40(W1), 2012. ISSN 03051048.

Guoli Wang and Roland L. Dunbrack. PISCES: A protein sequence culling server. *Bioinformatics*, 2003. ISSN 13674803.

Sheng Wang, Jian Peng, Jianzhu Ma, and Jinbo Xu. Protein Secondary Structure Prediction Using Deep Convolutional Neural Fields. *Scientific Reports*, 6, 2016. ISSN 20452322.

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014. ISBN 9783319105895.

Jian Zhou and Olga G. Troyanskaya. Deep Supervised and Convolutional Generative Stochastic Network for Protein Secondary Structure Prediction. 2014.

Jian Zhou and Olga G. Troyanskaya. Predicting effects of noncoding variants with deep learning-based sequence model. *Nature Methods*, 12(10):931–934, 2015. ISSN 15487105.

Jiyun Zhou, Hongpeng Wang, Zhishan Zhao, Ruifeng Xu, and Qin Lu. CNNH_PSS: Protein 8-class secondary structure prediction by convolutional neural network with highway. *BMC Bioinformatics*, 2018. ISSN 14712105.