predicting co-translational pausing with neural networks

bioc218 final project | winter 2013 | biafra ahanonu*

abstract

Neurodegenerative diseases are unified by the problem of protein misfolding and upset of proteostasis leading to neuronal death. Co-translational folding may play a role in proteostasis and codon optimality is a likely mechanism used to control folding rates. We give an overview of neural networks and their current implementation then propose a pipeline to use them in analyzing domain pausing in *S. cer* as a first step to see what mechanisms have evolved to counter misfolding in addition to the chaperone network. ROC analysis of the resulting neural network appears to indicate its validity, but the results are heavily skewed toward predicting non-pausing sites. Neural networks should be a valid method of analyzing the complex relationship between domain pausing and protein properties, but further refinement of the method proposed here is needed before a definitive conclusion can be made.

introduction

Huntington's, Parkinson's, Alzheimer's, Frontotemporal dementia (FTD), and many other neurodegenerative diseases have a common thread: protein misfolding and neuronal death.[39] Currently several proteins are implicated in each disease: huntingtin[47], α-synuclein[31], A[24], and TDP-43[26], respectively. Some of these proteins are known to have cellular functions, such as huntingtin's role in vesicular axonal transport.[45] Further, they form inclusions that appear to recruit both protein quality control machinery and other proteins (Ron Kopito, personal communication, 2013). This might implicate them in upsetting proteostasis and thus leading to a generalized increase in misfolded proteins in the cell. Studying the mechanisms of protein folding might provide insight as to how the cell compensates for toxic species and possible routes of intervention to slow the progress of these diseases.

Folding can occur at various times throughout a protein's life: co-translational, re-folding, post-translational, and others. Each can be mediated by various chaperones–such as SSB, Hsp90, and TRIC, respectively. Of particular interest is co-translational folding[6] and how the cell might use both the nucleotide sequence, tRNA pool, and other properties to dynamically control folding as a protein forms. One idea is that downstream of each domain are non-optimal codons (due to low cognate tRNA levels or other factors) that cause the ribosome to pausing and allowing time for the domain to fold or interact with a chaperone.[28] In this way, the same amino acid sequence can have a different folding landscape depending on codon composition. Synonymous codon mutations have been shown to affect protein function and stability.[17,46] Further, it has also been seen that synonymous codons might not be fitness neutral and could be slightly selected against.[3,5,7,43] Lastly, ribosome profiling and other methods should provide a way to verify whether the ribosome is indeed pausing at these sites for any specific amount of time.[14,15]

Dept. of Biology, Stanford University, Stanford, California
Correspondence: bahanonu@stanford.edu

To help generate solid hypothesis and complement growing experimental evidence, we propose to use neural networks to validate predictions made based on analysis of translational efficiencies of domains in the *S. cer* genome. This should allow us to use subsets of domains predicted to use pausing and see if by using known biochemical (hydrophobicity, secondary structure, disorder, etc.) and cellular (expression levels, heat-shock response, etc.) properties we can predict the other subset. This would give us insight as to what pausing is used for and how it might be regulated. Further, we can then use this to generate hypothesis about whether neurons are able to alter tRNA pools, quality control machinery, or other factors to compensate for increased load caused by the toxic species mentioned above.

history

Before diving into the proposed approach, we will give an overview of neural networks, both their history and implementation. Neural networks have a fairly long history–marked by both popularity and abandonment–stretching back to the early days of computing.[2,19,33] In 1943, Warren McCulloch and Walter Pitts published a bulletin outlining a neural network that could be used to describe all arithmetic functions and show applications of this finding in 1947.[23,30] Donald Hebb formulated the Hebbian rule–neurons that fire together, wire together–that would help form the basis for addition of learning algorithms to train the weights of a neural network.[10] This idea was extended to weights that were incorporated into several machines of the era, most notably the Mark I perceptron created by Frank Rosenblatt, Charles Wightman and others at MIT in the late 1950s.[9] The perceptron was described in detail by Rosenblatt (Figure 2) and was an early precursor to current implementations of neural networks.[36]

At the same time, over at Stanford, Marcian Hoff and Bernard Widrow were developing ADALINE, which would later evolve into MADALINE, a commercial system to help reduce noise in telephone systems.[20] In a classic example of good engineering, much like Bell Lab's ESS No. 1A Processor.[35] The ADALINE system was slightly different than that proposed by McCulloch and Pitts, namely the learning algorithm involved a weighted sum of the inputs to a net (Equation 2, also has error function used) instead of the activation/transfer functions output being used to update the network. In addition, they introduced gradient descent or the delta rule to converge on a local minima for the weight vector (basically want $\frac{\delta E}{\delta w} = 0$ where E and w are the error and weights respectively). The novel idea, and why it improved on the perceptron, was that the weights could change in steps that corresponded in magnitude to the error (e.g. $\Delta w \sim |E|$) The output function was just a summed weight of the input (Equation 1) and is thus a linear system compared to later non-linear designs.

$$y = \sum_{j=1}^{n} x_j w_j + \theta \quad (1)$$

$$w \leftarrow w + \eta(d-o) * x,\ E = (d-o)^2,\ note \quad (2)$$

Then, in 1969, Marvin Minsky and Seymour Papert showed several flaws in the perceptron that implied the field itself was fundamentally flawed, no matter how much neural networks improved in complexity and organization.[25] This, along with the rise of artificial intelligence

w = weight, y = output, θ = constant, n = number inputs, x = input
d = desired output, o = actual output, ν = learning rate

and a general adoption or von Neumann computing (with its focus on serial over distributed, parallel processing), lead to a reduction in research in the field through the 1970s and early 1980s. There were other issues, including use of a flawed error function and prevalence of single-layered networks.

These problems were corrected in the early 1970s and new layers of sophistication added. Teuvo Kohnen and James Anderson independently developed the linear associator (Figure 4), an feedforward extension of both ADALINE and the perceptron.[1,18] A year later Chris Malsburg proposed a non-linear model, both more biologically accurate in terms of response of voltage gated sodium channels and other receptors while introducing the idea that would come to encompass the non-linear functions (hyperbolic tangent, logistic function, etc.) used in later models.[22] This also would make networks without inhibitor networks stable. Then in 1974 Paul Werbos proposed the idea of error backpropagation, which is the basis for most neural network learning algorithms used today.[41] Additionally, the use of neural networks for clustering as give a boost by the invention of adaptive resonance theory (ART) for neural networks by Stephen Grossberg and Gail Carpenter.

Several years later, John Hopfield introduced Hopfield networks that would help explain how a memory could be stored and retrieved a memory using recurrent neural networks, e.g. activation of only a (relevant) subset would cause the network to converge to a specific minima, e.g. an attractor pattern.[13] Several years later, a team–consisting of Rumelhart, Hintont, Williams–published an article in *Nature* detailing the backpropagation of error (Figure 6) that would come to be used in many later models.[37] This also helped extend the delta rule of Widrow and Hoff to neural networks with multiple layers, but also increased the computation time needed to train the network.

The field has settled down since then and implemented several algorithms to deal with unsupervised learning. For example, deep learning has been applied to neural networks and consist of creating several hidden layers that begin by learning salient, general features of the input followed by successive layers that narrow the focus.[12] This latter technique begins to solve one of the problems of neural networks: they can be over-trained and tend to need many examples.

hierarchical neural networks

An aside: though this is not implemented to discussed in great detail in this paper, it would be interesting to build a hierarchy of neural networks to allow more detailed processing. For example, suppose you needed to train a robot to navigate an environment. In the most general sense, the robot needs to know if it should turn left/right or go forward/back. Once these are known, the problem becomes more specific, e.g. should it make a slight right, does it need to make a right that involves movement around an obstacle of size $n*n*n$, or a variety of other sub-problems. Thus, a possible extension of neural networks would be to have an initial coarse-grained classifier network that decided what general type the input was then relayed that to more specific networks to deal with (example of model in Figure 7).

An example of this in the brain would be your response to a snake-like object on a trail: visual information is coarsely classified initially and the snake-like object is classified as a thread and information from the LGN is sent to the amygdala to induce a reflexive backing away. The

input is also send to be further classified by the visual cortex and downstream dorsal/ventral pathways to decide whether the threat is a snake specifically or just a rope.

My proposed system, a hierarchical neural networks, would mimic this somewhat and might provide a way to combine very generalizable neural networks with nearly overtrained networks to help better classify objects. See pseudo-code block at end of paper for example implementation of this method in the R statistical language (Code 1)–error handling and background of creation of some parts of the neural networks are left out for sake of clarity. A similar method (Figure 9) has bee implemented to predict β-turns in a sequence.[29]

classic neural networks

Construction of a neural network follows several stages: choosing input nodes, specifying network topology and weights, deciding on the various functions used to transform neuronal input to output, and making a set of output nodes that contain classifications. In addition, the learning algorithm used to train the algorithm needs to be chosen. We will look at each step sequentially–along the way describing the mathematics and how it fits into the entire model. First we will outline the multilayered perceptron (Figure 8)then briefly go over radial basis function networks.

In general, you want the number of input nodes to exceed the number of neurons in the hidden layer by some multiplier to avoid immediate overfitting of the data (fitting n inputs to m parameters given $m \gg n$ will guarantee a network which exactly models each datapoint). Akaike information criterion can be used can be used to discourage overfitting.[27] In general, inputs should be determined first based on the type of problem.[33]

Next comes determination of the parameters that define each neuron in the hidden layer (Figure 10). The hidden layer is where weights are adjusted to help determine a matrix of weights, **W**, that satisfies a particular minimization procedure, which involves presenting input with a known, target output (in the case of supervised learning). There are three basic properties of a hidden layer neuron: propagation, activation, and output.

A propagation function is how the inputs to a neuron are combined and fed to the activation function. Normally propagation functions are just weighted summations of the input from the previous layer i to current layer j. In this case we have y_i inputs from the previous layer multiplied by $w_{i,j}$ weights to get the a_j activation of this neuron (see Figure 10):

$$a_j = \sum_{i=0}^{I} y_i * w_{i,j} \tag{3}$$

The output of the propagation function feeds into an activation function which determines how the neuron will respond to its inputs. Various types of activation functions have been used and, as mentioned previously, the most powerful are the non-linear ones. The most common are the logistic function and hyperbolic tangent, as seen in Equation 4 and Equation 5.[40] The advantage of these over a step function or other non-differentiable (over the entire function) functions is that it allows the network to have more flexibility.

$$g(a_j) = \frac{1}{1 + e^{-a_j}} \quad (4) \qquad g(a_j) = \frac{1 - e^{-2a_j}}{1 + e^{-2a_j}} \quad (5)$$

Once the activation function has been chosen, the output function needs to be set. Normally the output function is chosen to be the identity function (Equation 6). This allows the activation of the neuron to just represent the output and reduces complexity. While this might not be biologically precise (start of action potential propagation at the axon hillock depends on a variety of other factors besides just the input at dendrites and depolarization of the soma), it still allows us to model nearly any function and thus find one that captures the trend of the input data.

$$f_{out}(g(a_j)) = y_j = g(a_j) \quad (6)$$

Once a network architecture, neuronal model, and weights have been specified, we need to train the network. This can be done in a variety of ways, but the most common is to take advantage of least square minimization of difference between expected and actual outcomes by altering the weights of the network, encapsulated in the idea of backpropagation. A simple error function to minimize can be:

$$E = \frac{1}{2} \sum_{q=1}^{n} \sum_{k=1}^{K} [y_k(x^q, w) - t_k^q]^2 \quad (7)$$

Where y is some output over q different output patterns and k inputs (samples, with K being the last one) compared to a t target vector. The idea is to sum the errors between the output of all samples against the expected output for each classifier. For example, if we have an output vector with possible classifications of $\vec{y_1} = \{tree, cat, dog, human\}$, then we would expect if we give it an image of a cat, the output would look like $\vec{y_1} = \{.02, .82, .10, .06\}$ with the target vector being $\vec{y_1} = \{0, 1, 0, 0\}$ leading to a $E_1 = 0.0464$. The same procedure would be repeated over all samples. With this in mind, we can derive error backpropagation[2] by

$$e_j = g(a_j) \sum_{k=1}^{K} w_{k,j} e_k \quad (8)$$

Where $g(a)$ is some activation function, $w_{k,j}$ are weights from neuron j to output k, and e is the error signal at a given node. Thus, we see a backpropagation of the error signal, error signals at the output affect the error signal in the layer. This continues until you reach the input layer. With this in mind, we can then use the gradient descent method mentioned previously to update each weight according to a learning rate η.

$$\Delta w_{j,i}^{\tau} = -\eta \frac{\delta E}{\delta w_{i,j}} \Big|_{w^{\tau}} \quad (9)$$

This basically tells us that we change weight $w_{i,j}$ (ith input to j neuron) at time-step τ based on some constant (or as we'll see variable) learning rate η. Optimizing what value η should take is crucial and values that are too high might lead to instability due to the weight adjustments being too large and continually missing the true minima or hopping to a different valley in the error space. There are extensions of that general formula that take into account various problems that arise when searching the error space, such as when the curvature between the error space of one pattern is steeper compared to others.

The neural network we just outlined is considered a multilayered perceptron, there are other neural network methods of capturing the trends in data. The radial basis function networks take advantage of the mathematics involved in *exact* interpolation of data and apply it to neural networks. The idea with radial basis functions is that we can use a Gaussian or other mean-centered distribution ϕ_i (Equation 11 & Figure 8) for each input from the input layer and then sum these distributions to get a curve that exactly matches our data (Equation 10).

$$y_j(x) = \sum_{i=0}^{n} w_{i,j}\phi(x)_i \qquad (10)$$

$$\phi(x)_i = exp(-\frac{x - {\mu_i}^2}{2\omega_i^2}) \qquad (11)$$

To help determine the parameters, maximum likelihood or K-means have been used.[2] With K-means our goal is to find a set of radial basis functions that are centered on the mean of a cluster of input patterns. Although radial basis functions are another method of constructing neural networks, we will focus on derivatives of the multi-layered perceptron.

current implementations

Several groups have extended the classic neural network paradigm outlined above to various experiments. Application of neural network to predicting secondary structure has been done many times.[32] For example, Punta and Rost provide a fairly detailed guide on how to both use sparse encoding to represent amino acid sequences to be input into a network as well as providing advice on picking the correct performance parameter (e.g. ROC vs. Q measures). They also identify (as we will show) that population biases can reduce how generalizable a network is as well as making the performance parameter appear like the network is performing better than it actually is.[33]

Neural networks have also been used to detect a other protein properties. For example, in higher vertebrates, MHCI class receptors present antigen fragments to allow the immune system to detect if a cell has been infected. The antigen itself needs to be cleaved and determining where these cleavages by the proteasome occur could be informative. Neural networks were used to classify proteasome cleavage motifs between *in vitro* and *in vivo* experimental results.[16] The network performed well and they further showed that neural networks could help show if there was an underlying difference between two different experimental datasets of the same system. Of note, they used the coefficient of correlation to evaluate network performance and used experimental evidence to evaluate network performance.

P_x is the number of true positives (experimentally verified cleavage sites which are also predicted as cleavage sites), N_x the number of true negatives (experimentally verified non-cleavage sites, predicted as non-cleavage sites), P_{fx} the number of false positives (experimentally verified non-cleavage sites, predicted as cleavage sites) and N_{fx} the number of false negatives (experimentally verified cleavage sites, predicted as non-cleavage sites)

$$C = \frac{P_x N_x - N_{fx} P_{fx}}{\sqrt{(N_x + N_{fx})(N_x + P_x)(P_x + N_{fx})(P_x + P_f x)}} \tag{12}$$

Besides these papers and many others, a variety of more general tools have been developed to analyze different aspects of protein sequence structure, function, and other properties using neural networks. Of particular interest is the Center for Biological Sequence Analysis at the Technical University of Denmark. They have many servers for classification of proteasome cleavage sites, transmembrane helices, solvent accessibility, and others. For example, they have NetMHC to help predict binding of epitopes to MHC class 1 receptors.[21] That paper also provides a method to input short sequences into our neural net using sparse encoding, similar to the method discussed by Punta and Rost.

A recent paper[34] looks to extend neural networks to analyze multiple protein properties at the same time (Figure 12). By training the network to recognize the different properties jointly, they are able to obtain more accurate results with the prediction. This is likely due to the interdependencies of different properties, such as secondary structure and solvent accessibility, that allows the model to create a more robust picture of a protein and therefore allow for better classification.

neural networks and domain pausing

As mentioned at the outset, we are attempting to develop a neural network that can help predict and validate sites of domain pausing, specifically in *S. cer.* This will allow us to determine whether there is some higher-dimensional relationship between the domains identified to use pausing. While principal component analysis (PCA), K-means, and other clustering algorithms can tell us if they cluster in higher-dimensional space, they are not necessarily predictive and in the case of PCA, if the first two or three components don't capture the majority of the variance, it is hard to interpret anything from the clusters observed. On the other hand, we hope to use a high-quality, small dataset of domains with pausing to possible predict others that might use pausing, but for which our current method is unable to identify.

Our current implementation combines a whole genome analysis stage to extract sites we believe are using significant pausing (Figure 1) with a neural network stage. This allows us to use a supervised learning method, we can give the network domains we think are true targets and train it on those. A problem with this approach, and most of supervised neural network approaches, is that they require the targets to be valid. If there is a degree of uncertainty, this is not directly modeled in the learning algorithm. I would thus propose an extension (Equation 13) to the current error analysis functions used, in which a confidence vector s_k for each k input is multiplied by the error to help the network give larger weight to higher quality data.

$$E = \frac{1}{2} \sum_{q=1}^{n} \sum_{k=1}^{K} s_k [y_k(x^q, w) - t_k^q]^2 \tag{13}$$

This is of particular importance given the variable quality in various databases whether it be

PDB structures, secondary structure annotation of particular proteins (especially those with many regions predicted to have disorder), differences in kinetic measurements (half-life, etc.) that bias results toward certain classes of proteins, and other difficulties. The confidence vector would be pre-computed based on either metrics from the source data identifying quality or a new metric would have to be devised.

With this out of the way, we propose the following pipeline for analyzing domain pausing with neural networks. First, the translation efficiency of a variable length window (between 4-7 residues) around 45 residues downstream of the end of identified domains (from SCOP classifications) will be given as part of the input pattern. In addition, the size of the window (our procedure finds a windows length between 4-7 residues with the highest composition of non-optimal codons); the average translational efficiency of the codons in the window; the length of the domain (longer domains might need more pausing to help facilitate folding); the domain hydrophobicity and disorder; translational efficiency of the domain; what type of function it has in protein interaction networks (hub, bottleneck, etc., this can be determined by graph analysis of BioGRID or other databases and is illustrated in Figure 11); and half-life, transcriptional frequency, and mRNA expression levels of the protein will be fed into the neural network. These parameters should initially help capture the various biochemical properties of the domain along with cellular features. Cellular localization of the protein is available (via Gene Ontology and other databases), but we will perform the analysis initially by segmenting the proteins into groups based on localization, e.g. ER, mitocondrial, nuclear and cytoplasmic proteins will all be separately analyzed.

The initial dataset will then be sent to a cleaning routine to remove proteins with NaNs at any data-points (these cause problems and setting them to zero or the mean will still affects results unlike in PCA analysis). In addition, we will identify homologs (using ProDom, PSI-BLAST, or other methods) and give each family a unique ID to enable random filtering later on. This is done to prevent training a neural network that just predicts protein families rather than domain pausing. As this is a separate sub-routine, we can add other cleaning functions later if needed.

The cleaned dataset is returned to the main function and then a random subset of the homologs is chosen to be used in the creation of the neural network. Those not chosen are retained for use in the validation set after cross-training and other procedures are finished. This will allow us to present our network with sequences that are similar, but for which it has had no previous exposure. Next we send the dataset to a bias removal routine. This crawl through SCOP superfamilies to identify if our dataset is biased toward helical, strand, WD40 or other proteins and initially attempt to flatten this bias by picking random subsets from over-represented families equal in size to the smaller families. We wanted to remove secondary structure and disorder biases, but as these would likely be important parameters in determining domain pausing, this has been removed from the final proposal.

Lastly, we will choose a subset of domains without pausing equal to the size of the domains with pausing. Initial analysis shows that we only identified 1079 domains with pausing compared to 8932 domains analyzed, or just 12% of domains contain pausing. Neural networks are influenced by the magnitude of inputs given, e.g. if we give it 8x more no-pausing domains, it will become a no-pausing domain predictor rather than a classifier of domain paus-

ing. Thus, the bias routine will also choose a random subset of no-pausing domains equal to the number of pausing domains, keeping the neural network more generalizable.

After the dataset has been removed of biases and made sequence unique (remove homologs), it is split into *N* sets. Each set will function as a training set and a cross-training set. A neural net will be trained with each n set and then the patterns from cross-training (consisting of domain and protein properties) will be run through the trained network and the classifications compared to the actual classifications. Because the output of the neural network are probabilities of being in a particular class, we can use various thresholds from 0 to 1 and create a ROC curve of the true-positive rate vs. false-positive rate to give us an idea of how the network is performing. This will be done over all training sets.

An additional advantage of this procedure is that we can identify input patterns that actual influence the weights. For example, the neuralnets package allows us to view the generalized weights from the neural net and observe whether they differ from zero for each input parameter given.[8] This is likely a useful method of determining what properties don't seem to have a influence in grouping pausing and non-pausing domains and thus those that can be thrown out to reduce the input noise to the network.

After training of the network, the validation set can be compared to the network to determine how generalizable it is. While it may seem that we have covered the entire genome with our analysis, the main thrust behind this proposed analysis is to identify whether we can get a good, generalizable classifier for proteins with unannotated domains. We have 15121 domains in our current database, of these only 8932 are usable for extraction of domain pausing and only 1079 actually have pausing, or about 7% of annotated domains. As there are likely many more domains in the yeast genome and this may be a general mechanism used across species, obtaining a good neural network will allow us to get a first pass estimate of which proteins might be using pausing and we can investigate those in more detail experimentally. After mutation experiments like those done with actin have been performed with other proteins, we can then also increase the number of classifications, from just pausing and no-pausing to pausing with/without folding, function, cellular, and other effects post-synonymous codon mutation. This will allow a higher level of analysis that can guide future experiments.

Draft code and implementation were performed in R. Our neural network (Figure 14) appears at first glance to be very good as the ROC curve almost has an area of 1 (Figure 13). But further inspection shows that this is mostly due to correct identification of non-pausing domains but misclassification of many pausing domains. However, this initial run doesn't completely remove the pausing/non-pausing bias. Thus, we are seeing an example of the number of samples in each class biasing analysis. Further tests need to be done showing (code in progress but final results not shown in this paper).

A modification of this procedure would be to input the entire sequences translational efficiency profile along with other properties and see if the neural network find an input pattern. This would be unsupervised learning. This is not discussed further, but is something to keep in mind should the supervised learning prove difficult. It might allow us to identify pausing domains in the entire sequence that our other analysis doesn't address should the proposed

method fail to produce accurate networks.

extracting biochemical parameters

Once our algorithm has picked a set of domains that are most representative of the entire dataset (i.e. that subset has the highest ROC area), we need to extract some biological information from it. We propose several methods of doing this. Given we have the biological parameters, we can use principal component analysis to cluster domains that produce networks that accurately segment the data and thus represent those that contain the most information in terms of generalized properties needed to evolve domain pausing. From this, we can inspect disorder, hydrophobicity, secondary structure, and other variables that are likely to be involved. K-means clustering can be another method of observing whether those domains with high information content also seem to have obvious biochemical similarities. Further, we have heat-shock and other data, which will allow us to try and observe whether this mechanism is more used to facilitate continued function under stress rather than purely biophysical in nature.

neural networks with small sample size

A crucial problem in our implementation is the small sample size for the gold standard data. Because there are only a few papers that have actually shown synonymous codon mutations lead to functional changes,[17,46] if we are to use only these and a manually curated subset of the above data, we will inevitably have small set sizes. One method to get around this is to divide the data into N subsets and then train the network against each subset and use the other subsets as cross-training. However, the inherent problem with this method, as it has been proposed by others[33], is that it likely will not represent the total parameter space and that our small set is inherently biased toward either more disordered domains that we have crystal structures for (from the PDB) or that are more experimentally tractable. It might be worthwhile to include dirtier data, but as proposed above, adding a confidence factor to the error function could allow us to make the network more generalizable while also preventing the less reliable information from throwing off the network. This is feasible, for example, the protein structure and domain predictions database we initially used contains scores for less reliable domain classifications.[4] A metric could be made to convert these into usable input to the neural network error function.

conclusions

We presented a history of neural networks, from the original work of McCulloch and Pitts to the backpropagation work of Werbos. This gave us a background needed to present both a novel model for analyzing hierarchical datasets and then to look in detail at a classical model of a neural network. Using this as a spring-board, we briefly looked at some current implementation of neural networks and used some of these ideas to propose a pipeline for analysis of domain pausing in *S. cer*. Although the skeleton code is complete, full cleaning of the dataset and removal of biases is not and thus our current neural network is biased toward classifying most domains as non-pausing. While this might be biologically relevant, it likely points to a population bias in the dataset. We propose to then use the results of a improved neural net to analyze biochemical parameters using clustering algorithms. The small sample size problem should be solved by dividing the main dataset into N subsets and

training the model on these. Thus, should the proposed system work correctly, we will be able to use this to detect novel domains that utilize pausing and then check to see whether they can be experimentally disrupted. Should this hold true, we can bring the analysis to human cells and patient samples to see whether synonymous, otherwise silent mutations in various populations is actually what is inducing aggregation and identify proteins that might be at risk for aggregation.

code

Proposed pseudo-code for hierarchical neural networks.

Code 1: hierarchical neural networks in R

```
# Load neural net library
library(nnet)
library(neuralnet)

# Load trained neural nets
source("model.training.NN.hierarchical.v1")
trainedNeuralNets = NNtrained()

# Load data
source("data.run.NN.hierarchical")
this.data = NNdata()

# define the number of loops before an error has occured
error.loop.value = 3

neuralNet <- function(neuralNetFxn, this.loop.input){
       # this function runs specific level of a hierarchal neural network then
           recursively calls the next layer
       # base case is defined when a parent neural net has no children

       # run the initial neural net on the input
       # classifications is a tuple of probabilities for being in given class
       list(classifications,subfunctions) := neuralNetFxn(this.loop.input)


       # set the threshold
       threshold = this.loop.threshold.value

       # get index of next neural net function to be called
       subidx = max(find.col(classifications>threshold))

       # each neural net function outputs a list of functions that can further be
           called
       subfunctions = [...
       fearNN(),
       runNN(),
       thinkNN(),
       ...]
       #
       next.NN.fxn = subfunctions[[subidx]]

       # base case, if no more children, exit loop, else
       # recursively call the next neural network
       if(next.NN.fxn(children=TRUE)){
              return(classifications,next.NN.fxn)
```

```
        }else if(){
                neuralNet(next.NN.fxn,this.loop.input)
        }
}

main <- function(inputData,trainedNeuralNets,run.count){
        # get root neural net by passing no argument to trained neural nets
        root.NN = trainedNeuralNets()

        # Run the neural net hierarchical function until reach end node
        list(classification, id.NN) := neuralNet(root.NN,inputData)

        # Check that we have reached a neural net without children, else call
            main() again
        if(id.NN(children=TRUE)&run.count<error.loop.value){
                main(inputData,trainedNeuralNets,run.count+1)
        }else if(){
                return(NULL)
        }

        # Let user know choice
        print(classification)

        # Return classification, used later if integrated into system
        return(list(classification,id.NN))
}

# start the program
main(this.data,trainedNeuralNets,run.count=0)
```

Figures

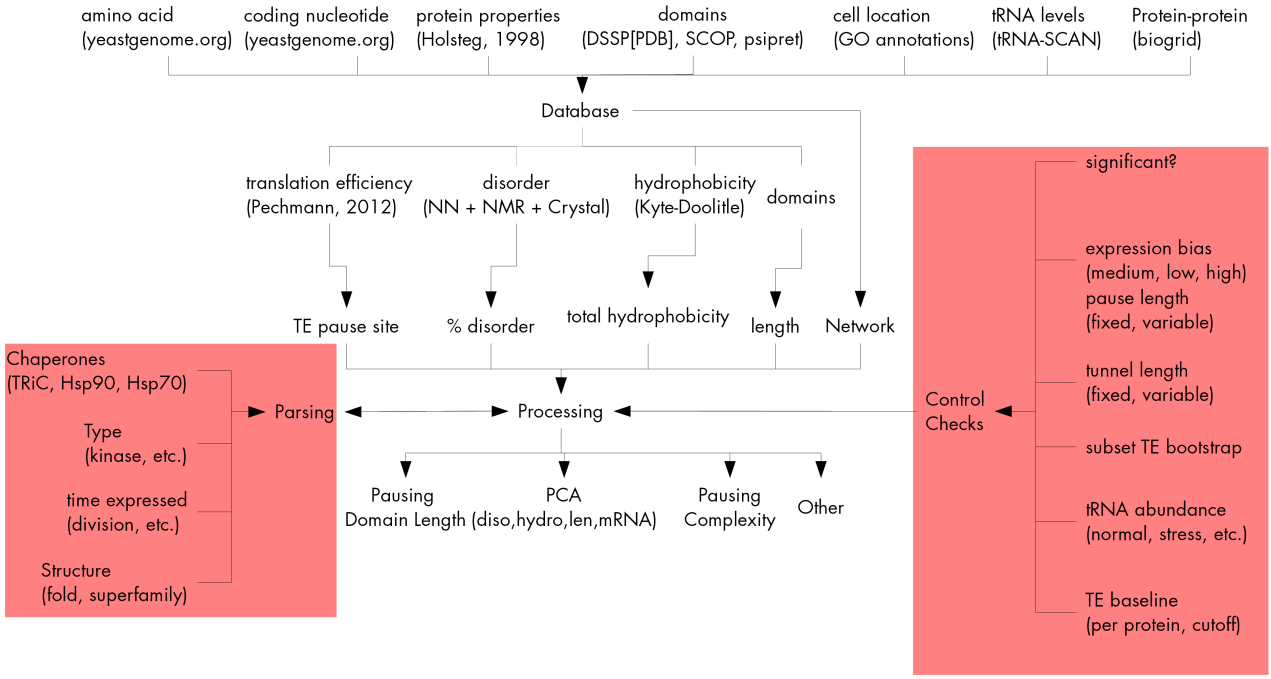

Figure 1 | Translation Efficiency Workflow
This diagram shows the primary sources used to gather each type of data used to analyze protein properties. Protein properties include half-life, mRNA expression level, and others.[11]

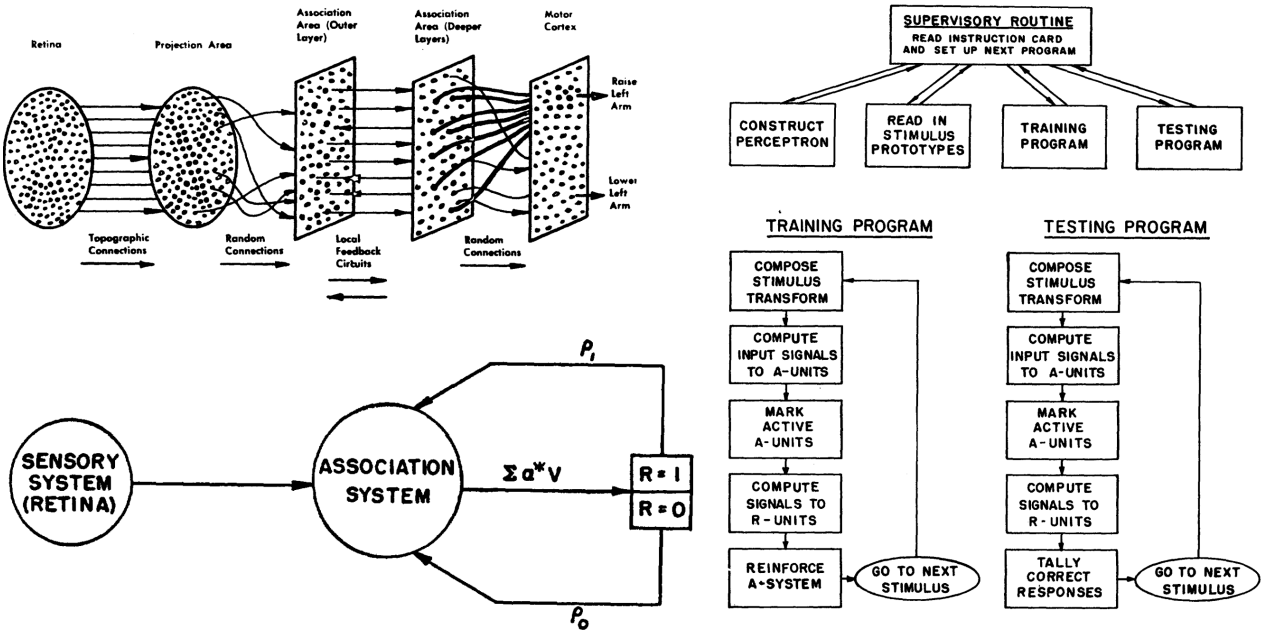

Figure 2 | Early perceptron
Rosenblatt proposed an early system that included feedback to help train and test the perceptron.[36]

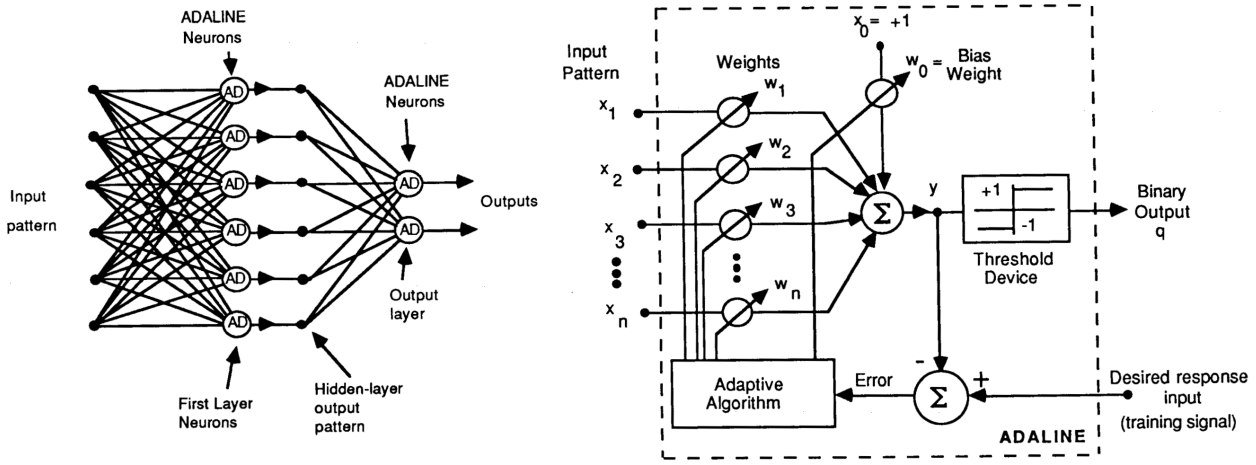

Figure 3 | ADALINE Implementation
Widrow and Hoff developed ADALINE to help predict and filter binary input, this was later implemented as MADALINE for commerical telephone operation to filter out noise.[42]

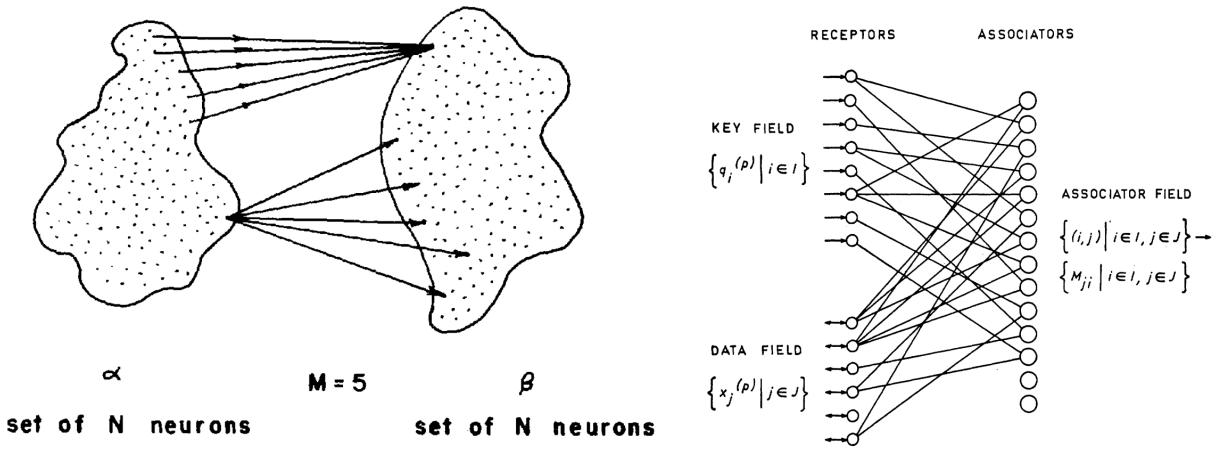

Figure 4 | Linear associator
Anderson (left) and Kohonen (right) introduced the concept a linear associator.[1,18]

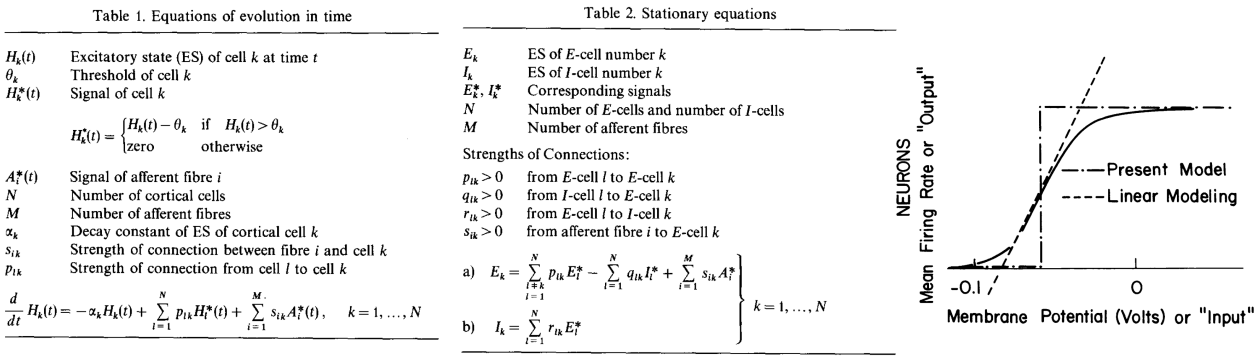

Table 1. Equations of evolution in time

$H_k(t)$ Excitatory state (ES) of cell k at time t
θ_k Threshold of cell k
$H_k^*(t)$ Signal of cell k

$$H_k^*(t) = \begin{cases} H_k(t) - \theta_k & \text{if } H_k(t) > \theta_k \\ \text{zero} & \text{otherwise} \end{cases}$$

$A_i^*(t)$ Signal of afferent fibre i
N Number of cortical cells
M Number of afferent fibres
α_k Decay constant of ES of cortical cell k
s_{ik} Strength of connection between fibre i and cell k
p_{ik} Strength of connection from cell l to cell k

$$\frac{d}{dt} H_k(t) = -\alpha_k H_k(t) + \sum_{l=1}^{N} p_{lk} H_l^*(t) + \sum_{i=1}^{M} s_{ik} A_i^*(t), \qquad k = 1, \ldots, N$$

Table 2. Stationary equations

E_k ES of E-cell number k
I_k ES of I-cell number k
E_k^*, I_k^* Corresponding signals
N Number of E-cells and number of I-cells
M Number of afferent fibres

Strengths of Connections:

$p_{ik} > 0$ from E-cell l to E-cell k
$q_{ik} > 0$ from I-cell l to E-cell k
$r_{ik} > 0$ from E-cell l to I-cell k
$s_{ik} > 0$ from afferent fibre i to E-cell k

$$\left.\begin{aligned} \text{a)} \quad E_k &= \sum_{\substack{l=1 \\ l \neq k}}^{N} p_{lk} E_l^* - \sum_{l=1}^{N} q_{lk} I_l^* + \sum_{i=1}^{M} s_{ik} A_i^* \\ \text{b)} \quad I_k &= \sum_{l=1}^{N} r_{lk} E_l^* \end{aligned}\right\} \quad k = 1, \ldots, N$$

Figure 5 | Non-linear neuron
Malsburg proposed a non-linear activation function based on the observation that a linear activation function leads to network instability, which can only be solved by inhibitory inputs.[22] An example of Hopfield's implementation of the non-linear IO relationship is seen at right.[13]

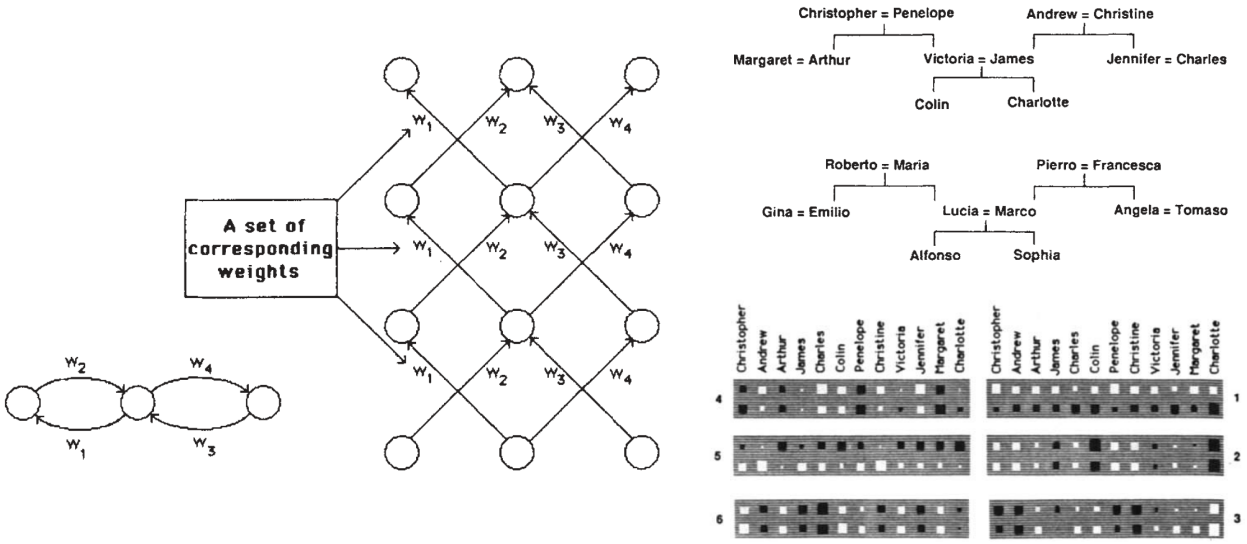

Figure 6 | Backpropagation of error
Rumelhart, Hintont, Williams publish their backpropagation of error model. On the right is seen the application of the net to a family tree.[37]

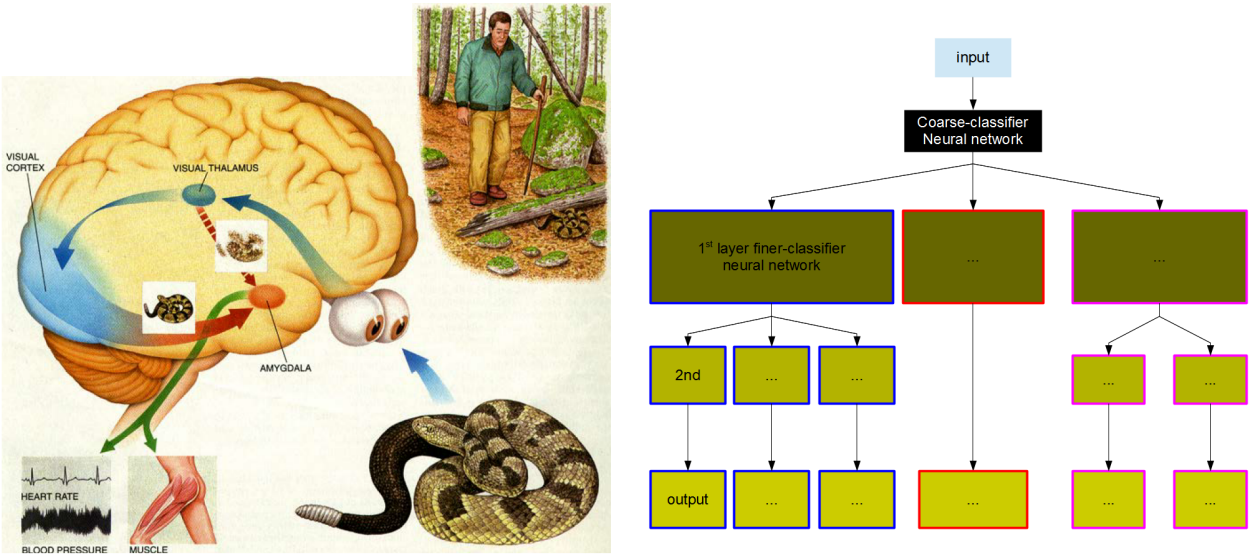

Figure 7 | Tiered neural networks
My proposed system to deal with network over-training and dealing with input that has multiple levels of representation.[38]

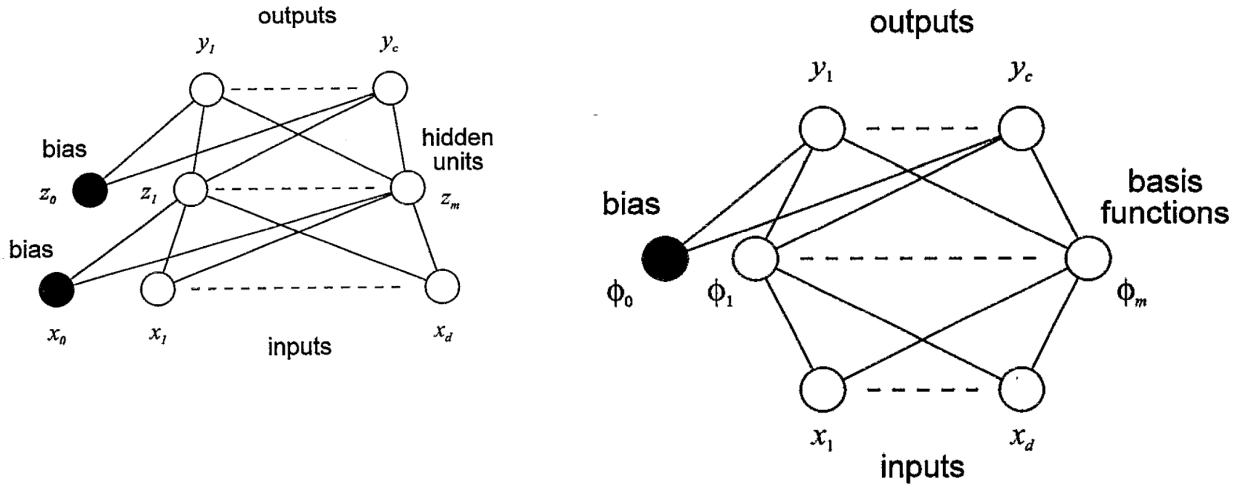

Figure 8 | Multi-layered perceptron and radial basis networks
Examples of neural network topology for classic perceptrons (left) and radial basis function (right) networks.[2]

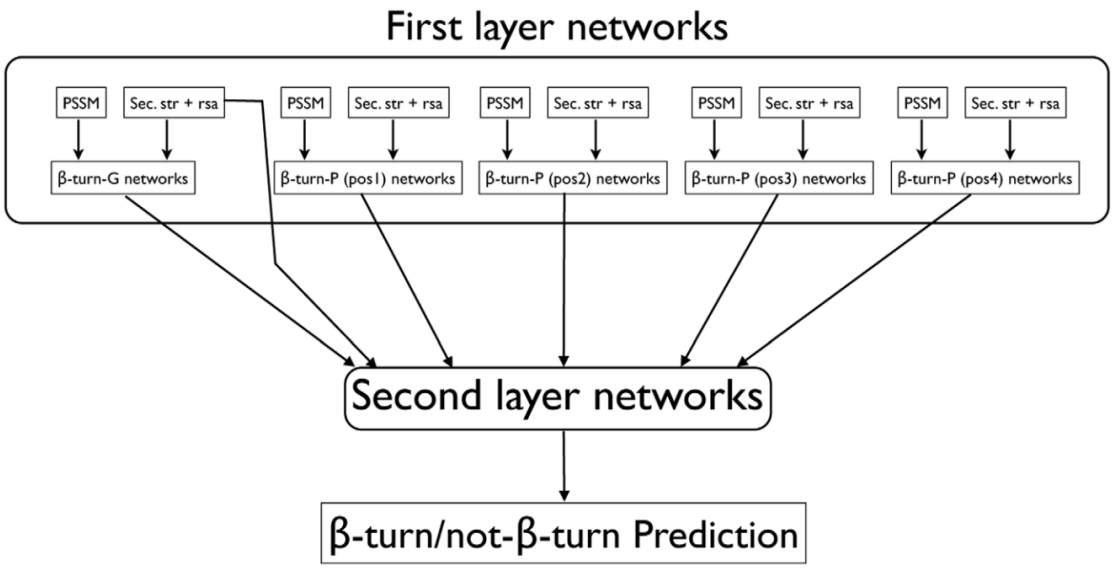

Figure 9 | hierarchical neural network
An implemented hierarchical neural network for prediction of beta-turns.[29]

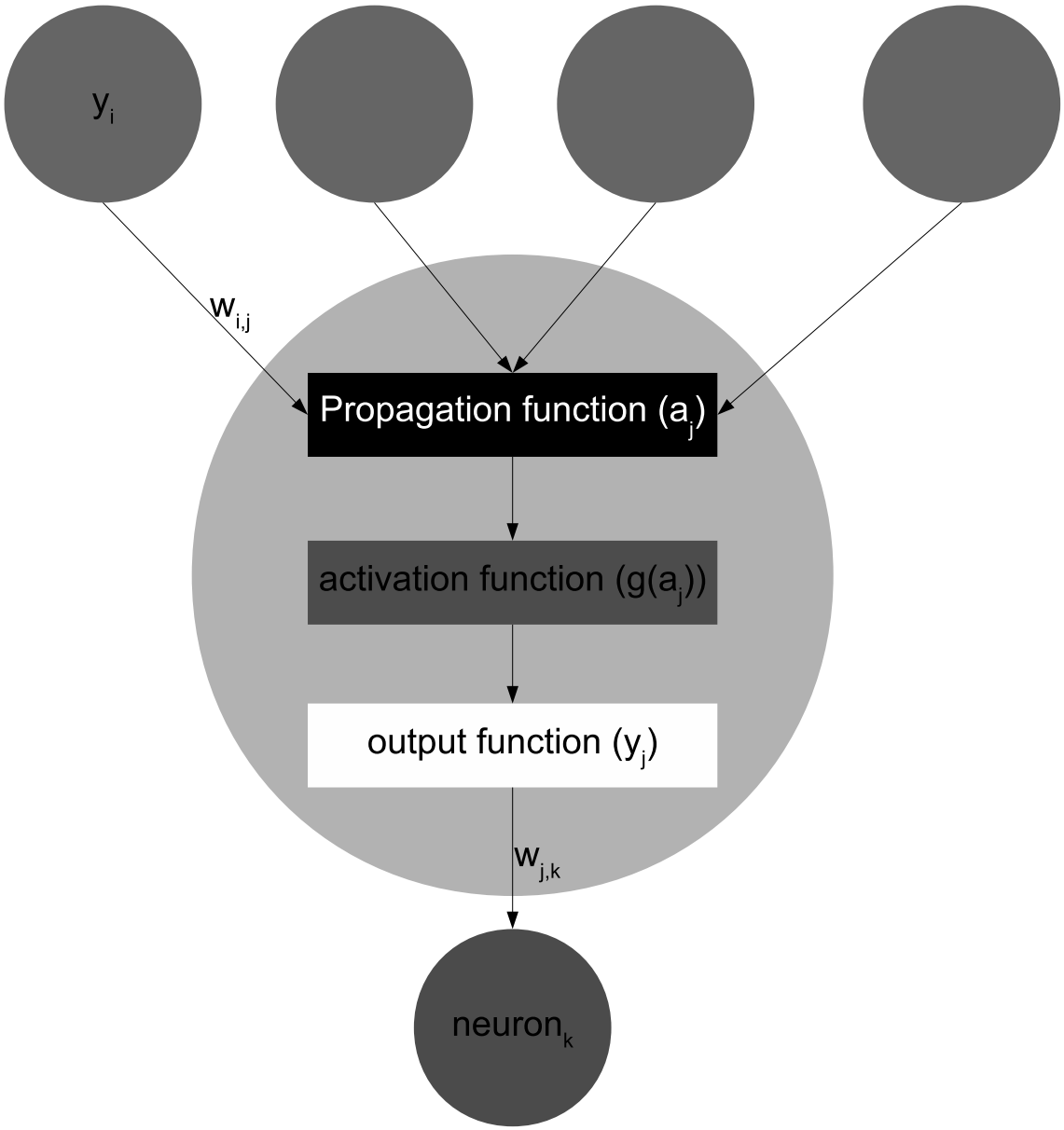

Figure 10 | Model of neuron in network
Neuron receives input from previous layer. This is run through propagation, activation, and output functions and send to the next layer.

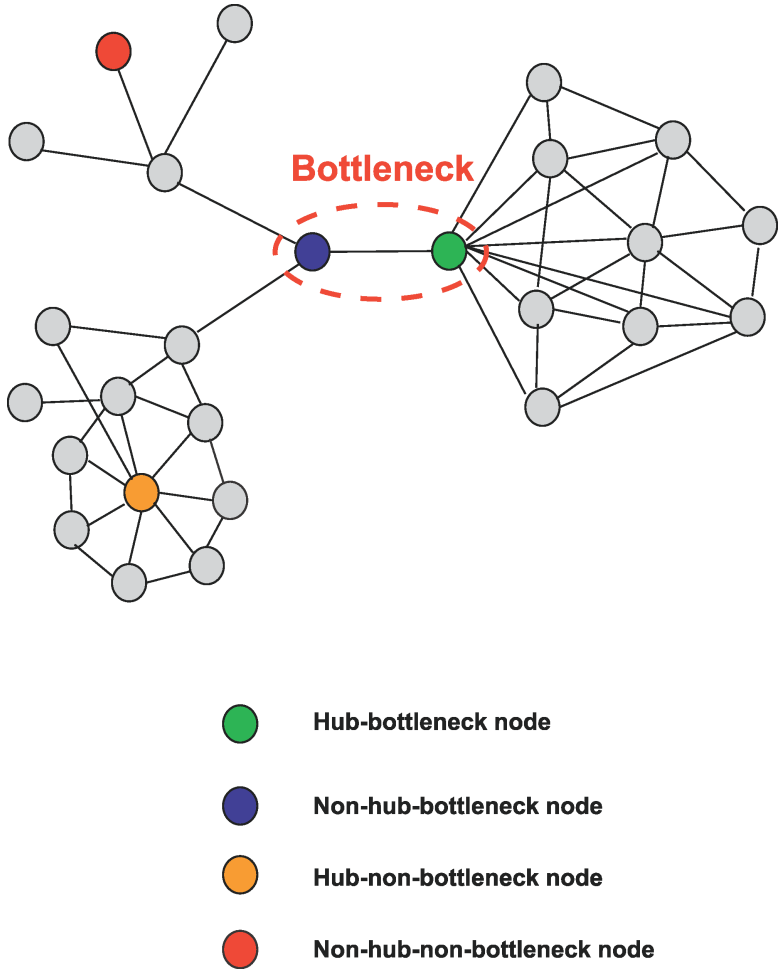

Figure 11 | Networks and hubs
It is possible that proteins with domain pausing form different subsets of network hubs and that along with other properties (hydrophobicity, etc.) help determine whether they need pausing as a back-up mechanism to ensure they fold properly and thus keep the network functioning.[44]

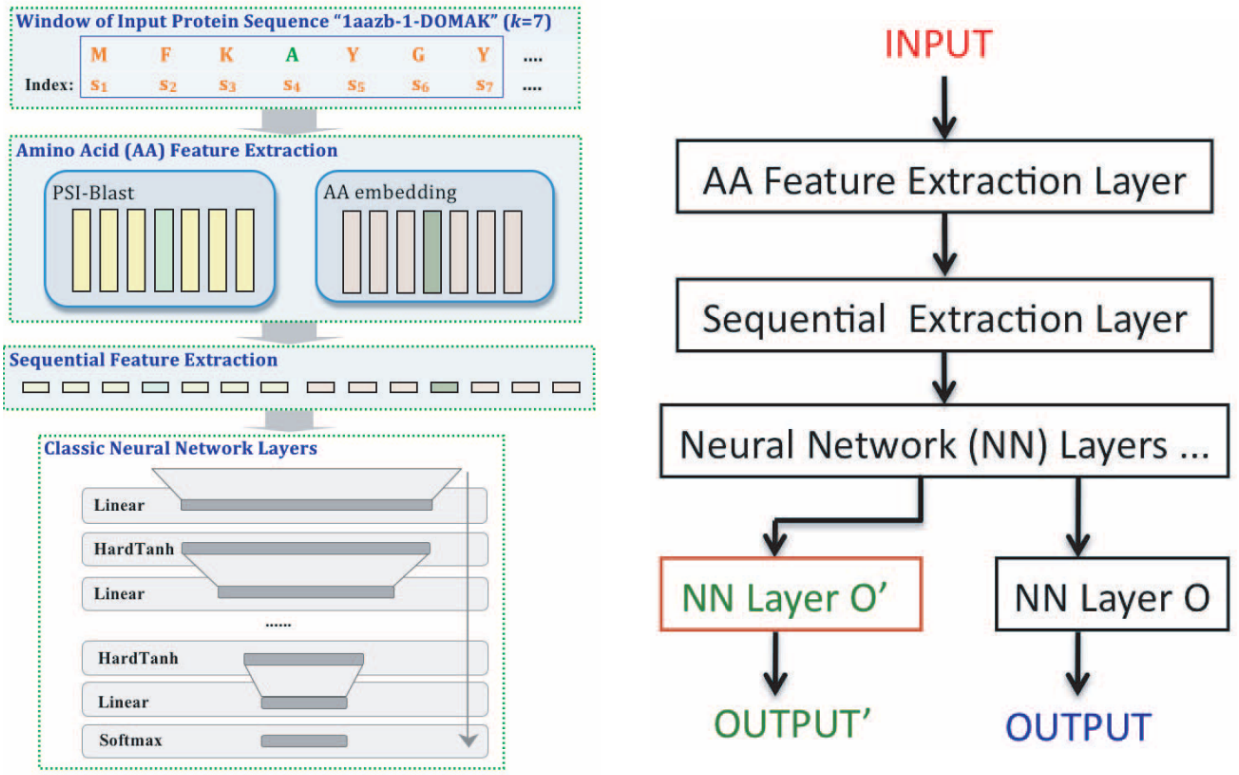

Figure 12 | Multi-task neural networks
This paper attempted to identify multiple protein properties at once using an integrated neural network procedure.[34]

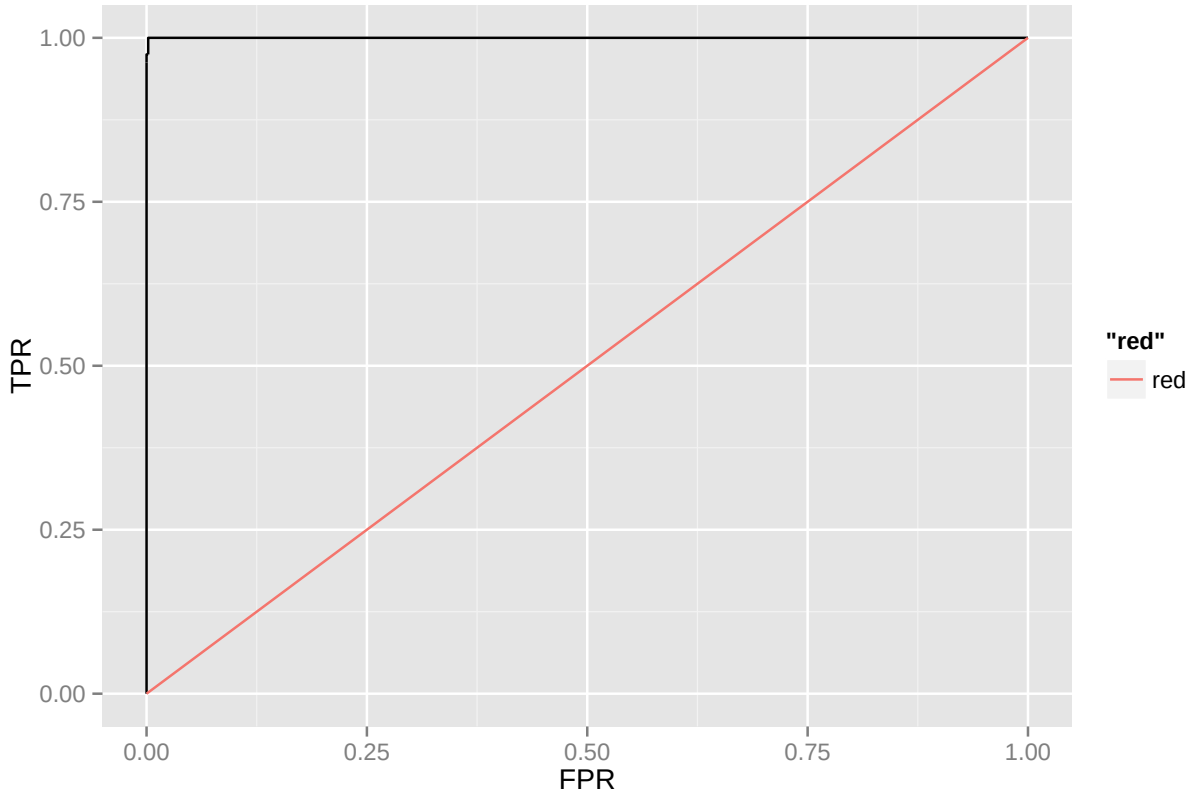

Figure 13 | Initial run ROC
This is the ROC curve running our initial dataset using the procedure outlined above. No homologs or various other cleaning was done, it will be added in later implementations.

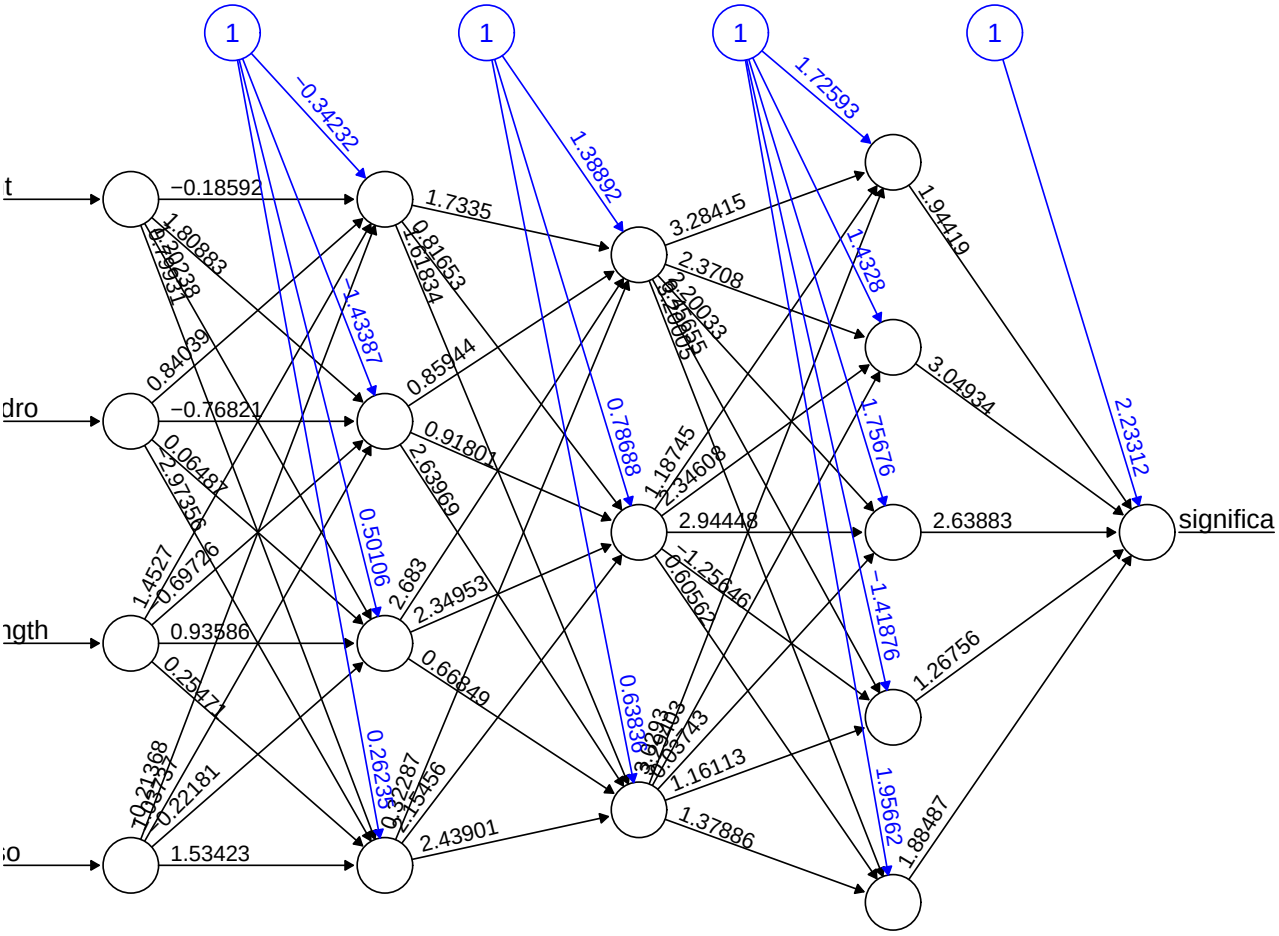

Figure 14 | Domain pausing neural network
An example neural network of our model implemented in the *neuralnet* package in R.

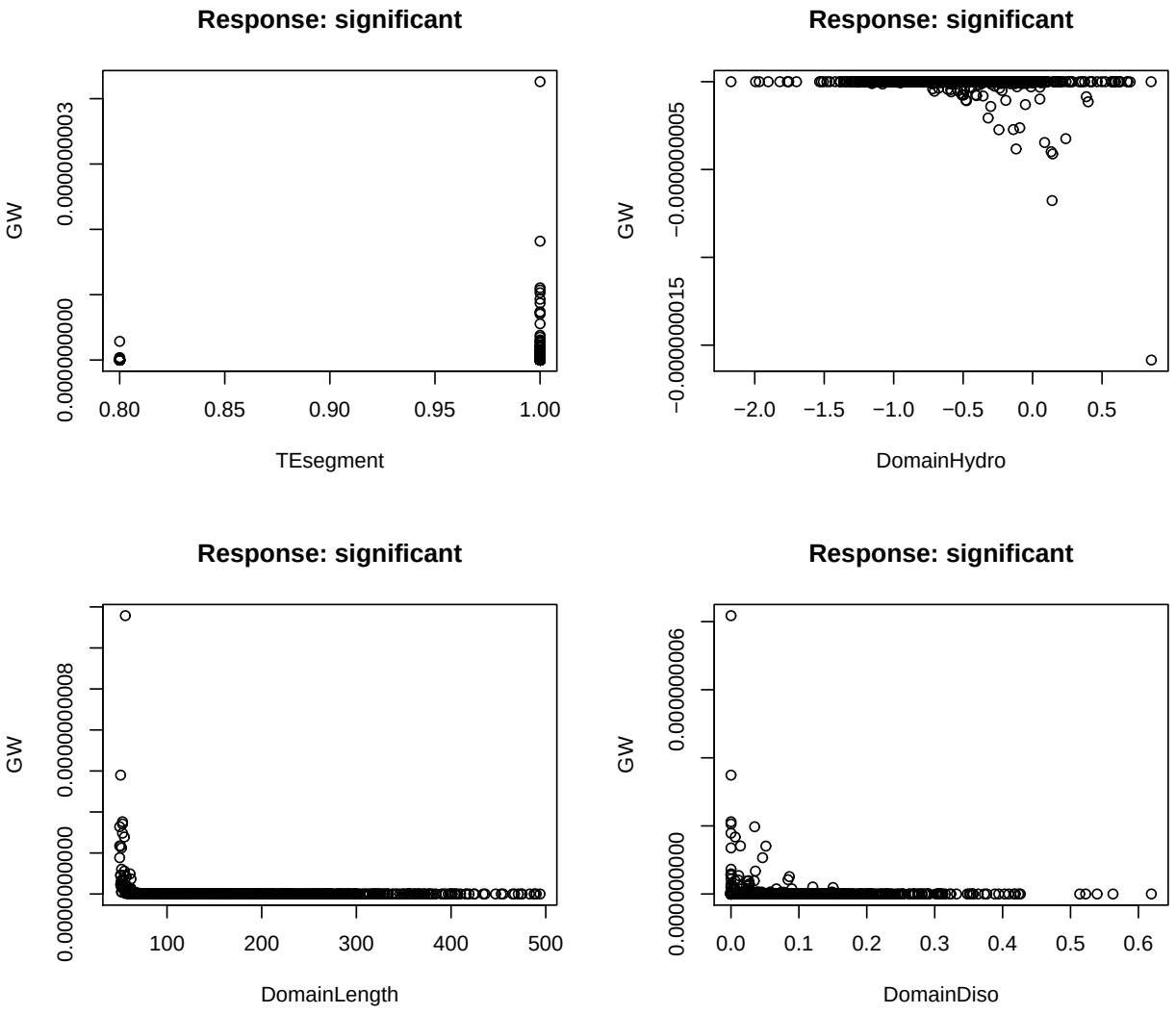

Figure 15 | Domain pausing neural network generalized weights
The generalized weights show that less disordered and hydrophobic along with shorter domains helped determine weights in the network.

References

[1] James A Anderson. A simple neural network generating an interactive memory. *Mathematical Biosciences*, 14(3):197–220, 1972.

[2] Chris M Bishop. Neural networks and their applications. *Review of scientific instruments*, 65(6):1803–1832, 1994.

[3] JV Chamary, Joanna L Parmley, and Laurence D Hurst. Hearing silence: non-neutral evolution at synonymous sites in mammals. *Nature Reviews Genetics*, 7(2):98–108, 2006.

[4] Kevin Drew, Patrick Winters, Glenn L Butterfoss, Viktors Berstis, Keith Uplinger, Jonathan Armstrong, Michael Riffle, Erik Schweighofer, Bill Bovermann, David R Goodlett, et al. The proteome folding project: Proteome-scale prediction of structure and function. *Genome research*, 21(11):1981–1994, 2011.

[5] D Allan Drummond and Claus O Wilke. Mistranslation-induced protein misfolding as a dominant constraint on coding-sequence evolution. *Cell*, 134(2):341–352, 2008.

[6] Daria V Fedyukina and Silvia Cavagnero. Protein folding at the exit tunnel. *Annual Review of Biophysics*, 40:337–359, 2011.

[7] King Leung Fung and Michael M Gottesman. A synonymous polymorphism in a common mdr1 (abcb1) haplotype shapes protein function. *Biochimica et Biophysica Acta (BBA)-Proteins & Proteomics*, 1794(5):860–871, 2009.

[8] Frauke Günther and Stefan Fritsch. neuralnet: Training of neural networks. *The R Journal*, 2(1):30–38, 2010.

[9] John C Hay, BEN E LYNCH, and DAVID R SMITH. Mark i perceptron operators'manual. Technical report, DTIC Document, 1960.

[10] Donald O Hebb. The organization of behavior: A neuropsychological approach. *NewYork: John Wiley & Sons. Hinton, GE (1989). Deterministic Boltzmann learning performs steepest descent in weightspace. Neural Computation*, 1:143–150, 1949.

[11] Christoph J Hengartner and Michael R Green. Dissecting the regulatory circuitry of a eukaryotic genome. *Cell*, 95:717–728, 1998.

[12] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[13] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[14] Nicholas T Ingolia, Sina Ghaemmaghami, John RS Newman, and Jonathan S Weissman. Genome-wide analysis in vivo of translation with nucleotide resolution using ribosome profiling. *Science*, 324(5924):218--223, 2009.

[15] Nicholas T Ingolia, Liana F Lareau, and Jonathan S Weissman. Ribosome profiling of mouse embryonic stem cells reveals the complexity and dynamics of mammalian proteomes. *Cell*, 2011.

[16] Can Ke□mir, Alexander K Nussbaum, Hansjörg Schild, Vincent Detours, and Søren Brunak. Prediction of proteasome cleavage motifs by neural networks. *Protein engineering*, 15(4):287–296, 2002.

[17] Chava Kimchi-Sarfaty, Jung Mi Oh, In-Wha Kim, Zuben E Sauna, Anna Maria Calcagno, Suresh V Ambudkar, and Michael M Gottesman. A" silent" polymorphism in the mdr1 gene changes substrate specificity. *Science*, 315(5811):525–528, 2007.

[18] Teuvo Kohonen. Correlation matrix memories. *Computers, IEEE Transactions on*, 100(4):353–359, 1972.

[19] David Kriesel. A brief introduction to neural networks. *Retrieved August*, 15:2011, 2007.

[20] Stanford University. Stanford Electronics Laboratories, B. Widrow, E. Hoff, United States. Office of Naval Research, United States. Army Signal Corps, United States. Air Force, and United States. Navy. *Adaptive switching circuits*. 1960.

[21] Claus Lundegaard, Ole Lund, and Morten Nielsen. Prediction of epitopes using neural network based methods. *Journal of immunological methods*, 2010.

[22] Chr Malsburg. Self-organization of orientation sensitive cells in the striate cortex. *Biological Cybernetics*, 14(2):85–100, 1973.

[23] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of mathematical biology*, 5(4):115–133, 1943.

[24] Catriona A McLean, Robert A Cherny, Fiona W Fraser, Stephanie J Fuller, Margaret J Smith, Konrad Vbeyreuther, Ashley I Bush, and Colin L Masters. Soluble pool of aβ amyloid as a determinant of severity of neurodegeneration in alzheimer's disease. *Annals of neurology*, 46(6):860–866, 2001.

[25] Marvin Minsky and Seymour Papert. Perceptron (expanded edition), 1969.

[26] Manuela Neumann, Deepak M Sampathu, Linda K Kwong, Adam C Truax, Matthew C Micsenyi, Thomas T Chou, Jennifer Bruce, Theresa Schuck, Murray Grossman, Christopher M Clark, et al. Ubiquitinated tdp-43 in frontotemporal lobar degeneration and amyotrophic lateral sclerosis. *Science*, 314(5796):130–133, 2006.

[27] Gaurang Panchal, Amit Ganatra, YP Kosta, and Devyani Panchal. Searching most efficient neural network architecture using akaike's information criterion (aic). *International Journal of Computer Applications IJCA*, 1(5):54–57, 2010.

[28] Sebastian Pechmann and Judith Frydman. Evolutionary conservation of codon optimality reveals hidden signatures of cotranslational folding. *Nature structural & molecular biology*, 2012.

[29] Bent Petersen, Claus Lundegaard, and Thomas Nordahl Petersen. Netturnp–neural network prediction of beta-turns by use of evolutionary information and predicted protein sequence features. *PloS one*, 5(11):e15079, 2010.

[30] Walter Pitts and Warren S McCulloch. How we know universals the perception of auditory and visual forms. *Bulletin of Mathematical Biology*, 9(3):127–147, 1947.

[31] Mihael H Polymeropoulos, Christian Lavedan, Elisabeth Leroy, Susan E Ide, Anindya Dehejia, Amalia Dutra, Brian Pike, Holly Root, Jeffrey Rubenstein, Rebecca Boyer, et al. Mutation in the α-synuclein gene identified in families with parkinson's disease. *Science*, 276(5321):2045–2047, 1997.

[32] Dariusz Przybylski and Burkhard Rost. Predicting simplified features of protein structure, 2007.

[33] Marco Punta and Burkhard Rost. Neural networks predict protein structure and function. *Methods Mol Biol*, 458:203–30, 2008.

[34] Yanjun Qi, Merja Oja, Jason Weston, and William Stafford Noble. A unified multitask architecture for predicting local protein properties. *PloS one*, 7(3):e32235, 2012.

[35] Brian Randell, Pete Lee, and Philip C. Treleaven. Reliability issues in computing system design. *ACM Computing Surveys (CSUR)*, 10(2):123–165, 1978.

[36] Frank Rosenblatt. Perceptron simulation experiments. *Proceedings of the IRE*, 48(3):301–309, 1960.

[37] David E Rumelhart, Geoffrey E Hintont, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[38] D. Sadava, H.C. Heller, D.M. Hillis, and M. Berenbaum. *Life: The Science of Biology*. W. H. Freeman, 2009.

[39] Daniel M Skovronsky, Virginia M-Y Lee, and John Q Trojanowski. Neurodegenerative diseases: new concepts of pathogenesis and their therapeutic implications. *Annu. Rev. Pathol. Mech. Dis.*, 1:151–170, 2006.

[40] Adi L Tarca, Vincent J Carey, Xue-wen Chen, Roberto Romero, and Sorin Drăghici. Machine learning and its applications to biology. *PLoS computational biology*, 3(6):e116, 2007.

[41] Paul Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. 1974.

[42] Ron Winter and Bernard Widrow. Madaline rule ii: a training algorithm for neural networks. In *Neural Networks, 1988., IEEE International Conference on*, pages 401–408. IEEE, 1988.

[43] Ziheng Yang and Rasmus Nielsen. Mutation-selection models of codon substitution and their use to estimate selective strengths on codon usage. *Molecular biology and evolution*, 25(3):568–579, 2008.

[44] Haiyuan Yu, Philip M Kim, Emmett Sprecher, Valery Trifonov, and Mark Gerstein. The importance of bottlenecks in protein networks: correlation with gene essentiality and expression dynamics. *PLoS computational biology*, 3(4):e59, 2007.

[45] Diana Zala, Maria-Victoria Hinckelmann, Hua Yu, Marcel Menezes Lyra da Cunha, Géraldine Liot, Fabrice P Cordelières, Sergio Marco, and Frédéric Saudou. Vesicular glycolysis provides on-board energy for fast axonal transport. *Cell*, 152(3):479–491, 2013.

[46] Fangliang Zhang, Sougata Saha, Svetlana A Shabalina, and Anna Kashina. Differential arginylation of actin isoforms is regulated by coding sequence-dependent degradation. *Science Signalling*, 329(5998):1534, 2010.

[47] Zhiqiang Zheng and Marc I Diamond. Huntington disease and the huntingtin protein. *Progress in Molecular Biology and Translational Science*, 107:189, 2012.

Index