Neural Networks and Applications in Bioinformatics

Yuzhen Ye

School of Informatics and Computing, Indiana University

Contents

- Biological problem: promoter modeling
- Basics of neural networks
- Perceptrons
  - The perceptron training rule
  - Gradient descent
  - Stochastic gradient descent
- Multilayer perceptron (MLP)
  - Backpropagation algorithm
- More applications
- References

Promoter modeling

- Promoter sequences are DNA sequences that define where **transcription of a gene by RNA polymerase begins**. Promoter sequences are typically located directly upstream or at the 5' end of the transcription initiation site. **RNA polymerase and the necessary transcription factors bind to the promoter sequence and initiate transcription**. Promoter sequences define the direction of transcription and indicate which DNA strand will be transcribed; this strand is known as the **sense strand.**
- The promoter contains specific DNA sequences that are recognized by proteins known as transcription factors.

Ref: http://www.nature.com/scitable/definition/promoter-259

Promoter elements

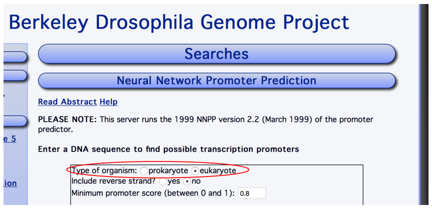
- **Core promoter -** the minimal portion of the promoter required to properly initiate transcription
  - Transcription Start Site (TSS)
  - Approximately -34
  - A binding site for RNA polymerase
  - General transcription factor binding sites
- **Proximal promoter -** the proximal sequence upstream of the gene that tends to contain primary regulatory elements
  - Approximately -250
  - Specific transcription factor binding sites

Ref: http://www.scfbio-iitd.res.in/tutorial/promoter.html

Eukaryotic vs prokaryotic promoters

- **Prokaryotic promoters**
  - In prokaryotes, the promoter consists of two short sequences at -10 and -35 positions upstream from the transcription start site.
- **Eukaryotic promoters**
  - Eukaryotic promoters are extremely diverse and are difficult to characterize.
  - They typically lie upstream of the gene and can have regulatory elements several kilobases away from the transcriptional start site.
  - Many eukaryotic promoters, contain a TATA box (sequence **TATAAA**), which in turn binds a TATA binding protein which assists in the formation of the RNA polymerase transcriptional complex. The TATA box typically lies very close to the transcriptional start site (often within 50 bases).

BDGP



Berkeley Drosophila Genome Project

PLEASE NOTE: This server runs the 1999 NNPP version 2.2 (March 1999) of the promoter predictor.

Enter a DNA sequence to find possible transcription promoters

Type of organism: ○ prokaryote ● eukaryote

Include reverse strand? ○ yes ● no

Minimum promoter score (between 0 and 1): 0.8

Time-delayed neural network (TDNN)

- TDNN was first introduced by Waibel et al. (1989)
- This architecture was originally designed for **processing speech sequence** patterns in time series with local time shifts.
- The usual way of transforming sequence patterns into input activity patterns is the extraction of a subsequence using a fixed window.
- This window is shifted over all positions of the sequence and the **subsequences are translated into input activities**.
- The network produces an output activity or score for each input subsequence.
- TDNN was applied to promoter modeling in 2000

TDNN for promoter modeling

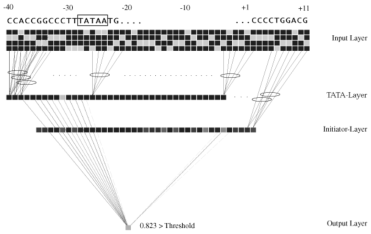


Image from: http://www.fruitfly.org/~martinr/doc/PhDThesisReese2000.pdf

ANN: Basics

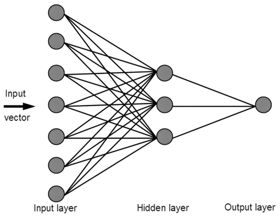
- Artificial neural networks (ANN) provide a robust approach to approximating real-valued, discrete-valued, and vector-valued target functions. For certain types of problems, such as learning to interpret complex real-world sensor data, ANN are among the most effective learning methods currently known. (Michell)
- Applications: handwritten characters, spoken words, face recognition, computer vision
- Bioinformatics applications:

ANN: Basics

- ANNs are inspired by the way in which the human brain learns and processes information, with the ability to handle complex (non-linear) features within data in order to generalize and predict well for future cases. Their concept simulates the behavior of a biological neural network; in humans, learning involves minor adjustments to the synaptic connections between neurons, in ANNs, the learning process is based on the interconnections between the processing elements that constitute the network topology.
- McCulloch and Pitts first described the concept of the artificial neuron in 1943 as a mathematical function derived from simulating the basic functions of biological neurons
- The majority of ANNs have a similar topology consisting of an input layer, one or more hidden layers and an output layer. The number of hidden layers and the number of neurons in each layer is dependent on the complexity of the problem, i.e. the number of input neurons.

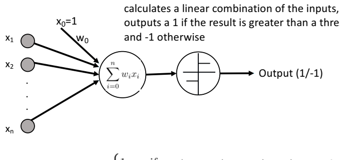
Brief Bioinform (2009) 10 (3): 315-329.

Typical form of an ANN



Perceptron

A perceptron takes a vector of real-valued inputs, calculates a linear combination of the inputs, then outputs a 1 if the result is greater than a threshold and -1 otherwise



$$O(x_1, \ldots, x_n) = \begin{cases} 1, & \text{if } w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n > 0 \\ -1, & \text{otherwise} \end{cases}$$

$$O(\vec{x}) = Sgn(\sum_{i=0}^{n} w_i x_i)$$

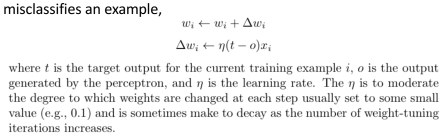
where $Sgn(y) = 1$ if $y > 0$, -1 otherwise

What perceptron can do

- A perceptron can be used to represent many Boolean function including AND, OR, NAND, and NOR, but not XOR
  - AND, two inputs, w_0=0.8, w_1=w_2=0.5
  - OR, two inputs, w_0=-0.3, w_1=w_2=0.5

Perceptron training rule

- The learning problem is to determine the weights so that the perception produces the correct output (1 or -1) for each of the training examples.
- Perceptron training rule
  - Begin with random weights, then iteratively apply the perceptron to each training example, modifying the weights whenever the perceptron misclassifies an example,



$$w_i \leftarrow w_i + \Delta w_i$$

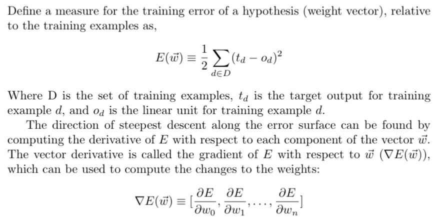
$$\Delta w_i \leftarrow \eta(t-o)x_i$$

where t is the target output for the current training example i, o is the output generated by the perceptron, and η is the learning rate. The η is to moderate the degree to which weights are changed at each step usually set to some small value (e.g., 0.1) and is sometimes make to decay as the number of weight-tuning iterations increases.

Gradient descent search & delta rule

- Perceptron rule finds a successful vector when the training examples are linearly separable, but fails to converge if the examples are not linearly separable.
- If the training examples are not linearly separable, use the delta rule, which converges toward a best-fit approximation to the target concept
- The key idea behind the delta rule is to use gradient descent to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.

Gradient descent rule



Define a measure for the training error of a hypothesis (weight vector), relative to the training examples as,

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$

Where D is the set of training examples, t_d is the target output for training example d, and o_d is the linear unit for training example d.

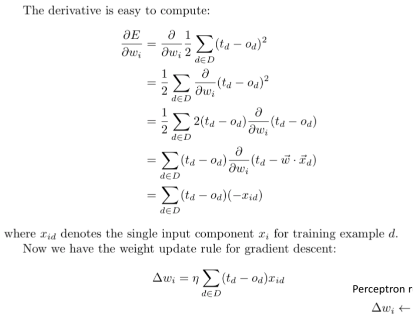
The direction of steepest descent along the error surface can be found by computing the derivative of E with respect to each component of the vector $\vec{w}$. The vector derivative is called the gradient of E with respect to $\vec{w}$ ($\nabla E(\vec{w})$), which can be used to compute the changes to the weights:

$$\nabla E(\vec{w}) \equiv [\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}]$$

The training rule for gradient descent

$$\vec{w} \leftarrow \vec{w} + \Delta \vec{w} \qquad w_i \leftarrow w_i + \Delta w_i$$

$$\Delta \vec{w} = -\eta \nabla E(\vec{w}) \qquad \Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$



The derivative is easy to compute:

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d \in D}(t_d - o_d)^2 \\ &= \frac{1}{2}\sum_{d \in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2 \\ &= \frac{1}{2}\sum_{d \in D}2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\ &= \sum_{d \in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w} \cdot \vec{x}_d) \\ &= \sum_{d \in D}(t_d - o_d)(-x_{id})\end{aligned}$$

where x_{id} denotes the single input component x_i for training example d.

Now we have the weight update rule for gradient descent:

$$\Delta w_i = \eta \sum_{d \in D}(t_d - o_d)x_{id}$$

Perceptron rule for comparison:

$$\Delta w_i \leftarrow \eta(t-o)x_i$$

Gradient descent & stochastic gradient descent

Gradient descent: computes weight updates after summing over all the training examples

$$\Delta w_i = \eta \sum_{d \in D}(t_d - o_d)x_{id}$$

Stochastic gradient descent: computes weight updates for each individual example.

$$\Delta w_i \leftarrow \eta(t-o)x_i$$

Delta rule

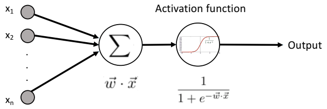
Both are commonly used in practice.

The perceptron training rule and delta rule appear to be identical! But the rules are different. In the delta rule, *o* refers to the linear unit output, but in the perceptron rule, *o* refers to the thresholded output sgn(net).

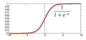
Multilayer networks

- Single perceptrons can only express linear decision surfaces
- Multilayer networks learned by the backpropagation algorithm are capable of expressing a rich variety of nonlinear decision surfaces.
- Multiple layers of cascaded linear units still produce only linear functions
- Need a unit whose output is a nonlinear function of its inputs (e.g., sigmoid function), and whose output is a differentiable function of its inputs.

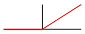
Differentiable threshold unit



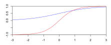
$sigmoid(x) = \dfrac{1}{1+e^{-x}}$



ReLU: $f(x)$=max(0,x)



tanh



(blue: sigmoid; red: tanh)

ReLU (Rectified linear function): typically applied for solving regression problems
Sigmoid (logistic function): for classification
Tanh: rescaled logistic sigmoid function

Backpropagation algorithm

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

Error function for single layer network

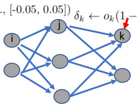
$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd} - o_{td})^2$$

Error function for multiple layer network

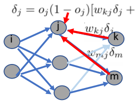
Backpropagation algorithm

```
1. Create a feed-forward network with $n_{in}$ inputs, $n_{hidden}$ hidden units, and
   $n_{out}$ output units.
2. Initialize all network weights to small random numbers (e.g., [-0.05, 0.05])
3. Repeat until the termination condition is met
   For each $(\vec{x}, \vec{t})$ in training examples:
    (a) Propagate the input forward through the network
    (b) Propagate the errors backward through the network:
         i. For each output unit $k$, calculate the error term,
            $\delta_k \leftarrow o_k(1 - o_k)/(t_k - o_k)$
        ii. For each hidden unit $j$, calculate the error term,
            $\delta_j \leftarrow o_j(1 - o_j)\sum_{k\in outputs} w_{kj}\delta_k$
       iii. Update each network weight $w_{ji}$
            $w_{ji} \leftarrow= w_{ji} + \Delta w_{ji}$, where $\Delta w_{ji} = \eta\delta_j w_{ji}$.
```

$\delta_k \leftarrow o_k(1 - o_k)/(t_k - o_k)$



$\delta_j = o_j(1 - o_j)[w_{kj}\delta_j + w_{mj}\delta_m]$



[the error terms shown here are based on sigmoid function; see Mitchell pp101-103 for derivation of the rules]

Adding momentum

One common variation to the backprogagation weight-update rule is to make the weight update on the nth ieration depend partially on the update that occured during the $(n-1)$th iteration as follows,

$$\Delta w_{ji}(n) = \eta\delta_j x_{ji} + \alpha\Delta w_{ji}(n-1)$$

Pros and Cons

- Pros
  - ANN are fault tolerant, i.e. they have the ability of handling noisy or fuzzy information, whilst also being able to endure data which is incomplete or contains missing values.
  - In addition to this (like other machine learning methods), they are capable of generalization, so they can interpret information which is different to that of the training data, thus representing a 'real-world' solution to a given problem by their ability to predict future cases or trends based on what they have previously seen.
  - There are several techniques that can be used to extract knowledge from trained ANNs, and the importance of **individual variables** can be easily recovered using various methods such as the analysis of interconnecting network **weights**, sensitivity analysis and rule extraction.
- Cons
  - Training might be time consuming
  - Over-fitting

ANN: Regularization

- In ANNs the risk of low generalization is mainly attributed to over-training of the model, leading to over-fitting and subsequently poor predictive performance during independent validation.
- Regularization is to add a *regularization term* to the cost/error function to prevent the coefficients to fit training data so perfectly
- Approaches
  - Weight decay
  - Resampling and early stopping
  - Bayesian regularization
  - Cross validation

L1 and L2 regularization

- L2 is the sum of the square of the weights
- L1 is just the sum of the weights

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{td})^2$$

$$\Big\downarrow \text{Add L2}$$

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{td})^2 + \gamma \sum_{i,j} w_{ji}^2$$

which yields a weight update rule identical to the backpropagation rule, except that each weight is multiplied by the constant $(1 - 2\gamma\eta)$ upon each iteration. γ is the regularization rate.

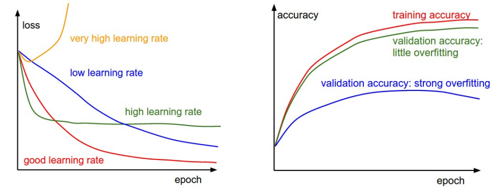
Machine Learning: Tom Mitchell

Convergence and local minima

- Common heuristics to alleviate the problem of local minima
  - Add a momentum term to the weight-update rule
  - Use stochastic gradient descent rather than true gradient descent: the different error surfaces typically will have different local minima, making it less likely that the process gets stuck in any one of them.
  - Initialize each network with different random weights. Select the one with the best performance over a separate validate data set; or retain all networks and treat them as a "committee" of networks, whose output is the average of the individual network outputs.

Tracing the performance of ANN

Quantities to monitor during training of a neural network: loss, accuracy, etc



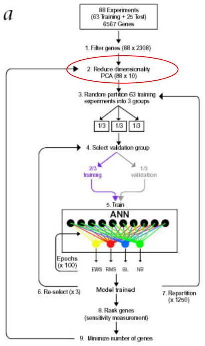
Epoch measures how many times every example has been seen during training in expectation (e.g. one epoch means that every example has been seen once)

Ref: http://cs231n.github.io/neural-networks-3/

Application of ANN to cancer prediction (genomics)

- Ref: Nat Med. 2001 Jun;7(6):673-9.
- It reports a method of **classifying cancers to specific diagnostic categories based on their gene expression signatures** using artificial neural networks (ANNs).
- ANNs were trained using the small, round blue-cell tumors (SRBCTs) as a model. These cancers belong **to four distinct diagnostic categories** and often present diagnostic dilemmas in clinical practice. The ANNs correctly classified all samples and identified the genes most relevant to the classification.

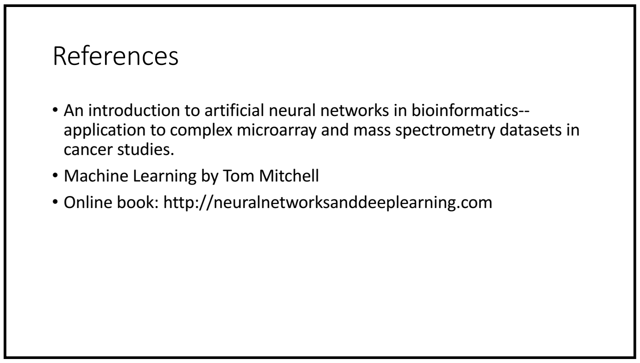
"The random shuffling was redone 1250 times and for each shuffling we analyzed 3 ANN models. Thus, in total, each sample belonged to a validation set 1250 times, and 3750 ANN models were calibrated. For each diagnostic category (EWS, RMS, NB or BL), each ANN model gave an output between 0 (not this category) and 1 (this category). The 1250 outputs for each validation sample were used as a committee as follows. We calculated the average of all the predicted outputs (a committee vote) and then a sample is classified as a particular cancer if it receives the highest committee vote for that cancer."



a
88 Experiments (63 Training + 25 Test) 6567 Genes
1. Filter genes (88 x 2308)
2. Reduce dimensionality PCA (88 x 10)
3. Random partition 63 training experiments into 3 groups
1/3
1/3
1/3
4. Select validation group
2/3 training
1/3 validation
5. Train
ANN
Epochs (x 100)
EWS
RMS
BL
NB
6. Re-select (x 3)
Model trained
7. Repartition (x 1250)
8. Rank genes (sensitivity measurement)
9. Minimize number of genes

Application of ANN in proteomics

- Ref: Bioinformatics. 2002 Mar;18(3):395-404.
- "Using a multi-layer perceptron Artificial Neural Network (ANN) (Neuroshell 2) with a back propagation algorithm we have developed a prototype approach that uses a model system (comprising five low and seven high-grade human astrocytomas) to identify **mass spectral peaks** whose relative intensity values correlate strongly to tumour grade."



References

- An introduction to artificial neural networks in bioinformatics--application to complex microarray and mass spectrometry datasets in cancer studies.
- Machine Learning by Tom Mitchell
- Online book: http://neuralnetworksanddeeplearning.com