Neural Networks Notes

Nhan Trong

July 6, 2016—July 10, 2016

> Pipey. *Looks like you want to compress a movie file, can I help? You know with Pied Piper's revolutionary neural network optimized sharded data distribution system, it's just six clicks away, follow meeee!*
>
> Silicon Valley

Contents

Part I
Different Types of Neurons and Learning

Question 1. *How many other neurons does a neuron talk to? Do they change neighbours?*

Note 2. "Goal of unsupervised learning: provides a compact, low-dimensional representation of the input," like Pied Piper's compression algorithm using neural networks!

Keywords

Fruit flies, MNIST, TIMIT, linear, binary threshold, rectified / linear threshold, logistic, stochastic binary neurons, supervised, unsupervised, reinforcement learning.

Part II
Neural Network Architectures

Keywords

Feed forward, recurrent, symmetrically connected neural network, perceptrons, convexity condition.

Part III
Perceptron Learning Algorithm

TODO 1. Proof of why perceptron learning works is very sketchy! Need more details.

1 Binary Threshold Neurons / McCulloch-Pitts

Used in Perceptrons.

Question 3. *Also called Linear Threshold Neurons?*

Definition 4. *First compute $z = w^T x$, then output*

$$y = \begin{cases} 1 \text{ if } z \geq 0 \\ 0 \text{ otherwise,} \end{cases}$$

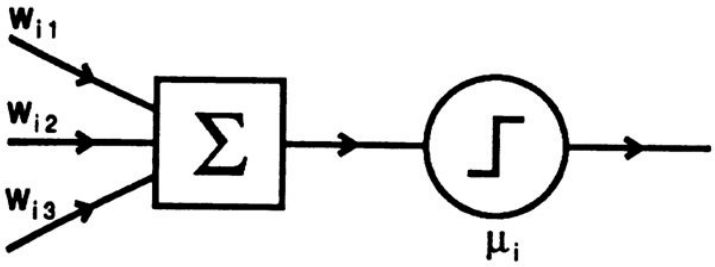
representing "all VS none" activation.

Note 5. Here we implicitly added the threshold as a bias unit: $w = (b, w_1, w_2, \ldots)$.

Remark 6. The function $y(z)$ is also called the Heaviside / unit step function.

McCulloch–Pitts "neuron" (1943)

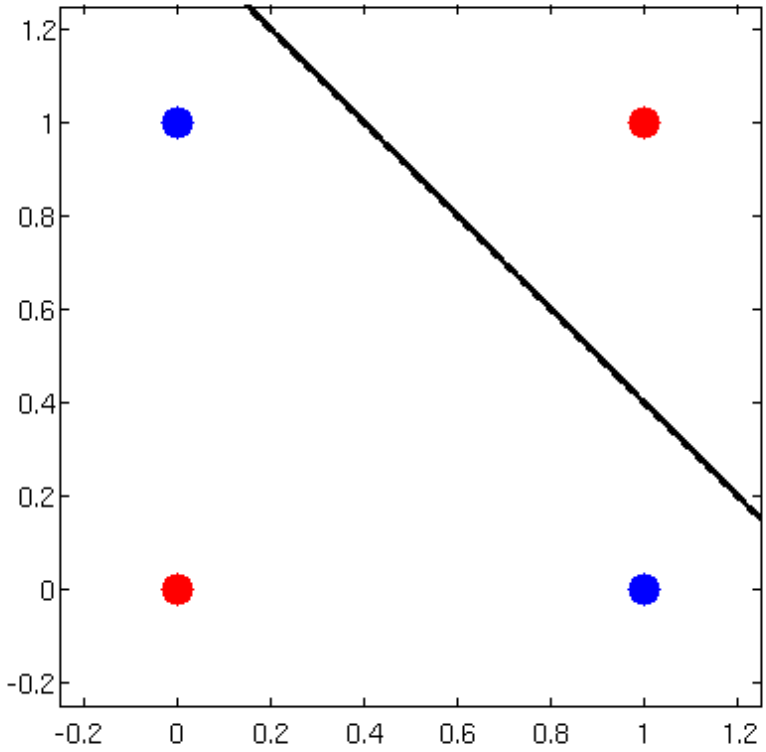
- Attributes of neuron
  - m binary inputs and 1 output (0 or 1)
  - Synaptic weights w_{ij}
  - Threshold μ_i



R. Rao, Week 6: Neural Networks 3

2 Limitations of the Binary Threshold Neuron

Proposition 7. *A single binary threshold neuron cannot learn the XOR function, because geometrically its truth table represented on a plane is not linearly separable.*



2.1 Group Invariance Theorem

Proposition 8. *Perceptrons can't learn patterns if they're subject to transformations that form a group, e.g. translations with wrap-around.*

Question 9. *Details?*

Keywords

Data space, weight space, Group Invariance Theorem.

Part IV
Linear Neuron Learning Algorithm

Definition 10. *Given a training case x_n and a weight vector w, the neuron's estimate y_n of the desired output is*

$$y_n = \sum_i w_i x_{ni} = w^T x_n.$$

Define the cost function E_n to be the squared difference error

$$E_n = \frac{1}{2}(t_n - y_n)^2,$$

where t_n is the target output, i.e. the "ground truth", and define the total error to be

$$E = \sum_n E_n.$$

Finally the goal of learning is to minimize E:

$$\min_w E.$$

3 Delta Rule: Learning by Gradient Descent

The error partials are

$$\frac{\partial E}{\partial w_i} = \sum_n \frac{dE_n}{dy_n}\frac{\partial y_n}{\partial w_i} = -\sum_n (t_n - y_n)x_{ni}.$$

The Delta Rule / Gradient Descent says that we should change w_i in the opposite direction as the change in error along w_i, give or take a learning rate α:

$$\Delta w_i = -\alpha \frac{\partial E}{\partial w_i} = \sum_n \alpha(t_n - y_n)x_{ni},$$

i.e. α tells us how much to change, and the negative sign tells us which direction to go, namely the opposite direction. E.g. if $\frac{\partial E}{\partial w_i} > 0$, that means the error goes up as w_i increases, so we want to decrease w_i to make it go down, and vice versa.

4 Error Surface of a Linear Neuron

Question 11. *IIRC feature normalization should help with slow learning due to unscaled data? What about pathological cases like this?*

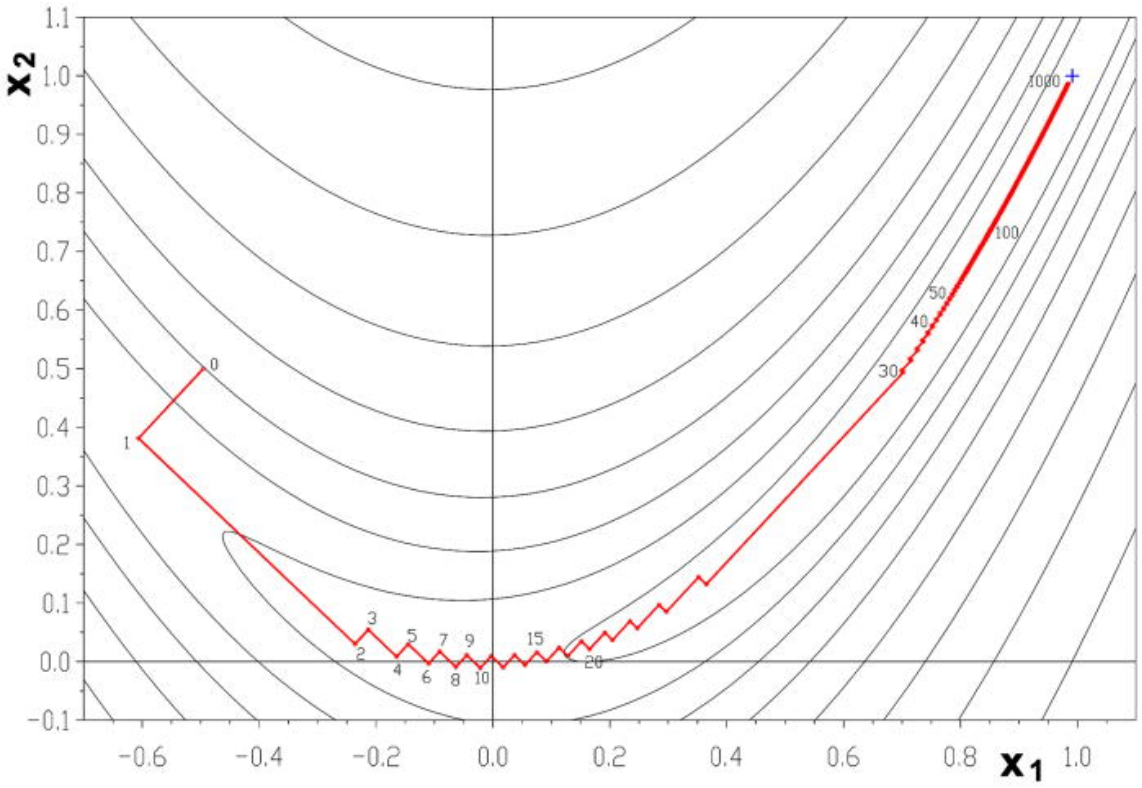


Figure 1: Rosenbrock Valley.

Keywords

Linear neurons / linear filters, iterative / computational VS analytic / mathematical approach, Delta Rule / Gradient Descent, batch VS online, error surface, extended weight space, Rosenbrock function.

Part V
Logistic Neurons

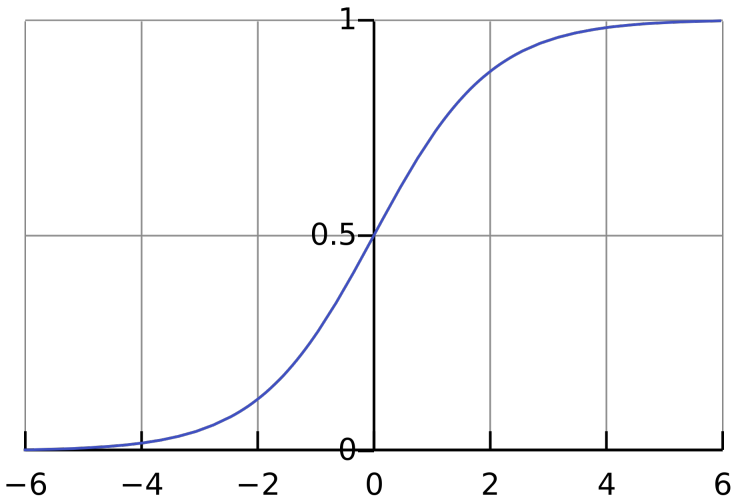
5 Learning Rule

Definition 12. *The estimator for a logistic neuron is given by*

$$y = \frac{1}{1 + e^{-z}}$$

where $z = w^T x$*. The function* $y(z)$ *is also known as a logistic / sigmoid function, and* z *is sometimes called the logit. As before, the error is the squared difference*

$$E = \frac{1}{2} \sum_n (t_n - y_n)^2.$$



Proposition 13. *The estimator derivatives are*

$$\frac{\partial y}{\partial w_i} = \frac{dy}{dz} \frac{\partial z}{\partial w_i} = y(1 - y) x_i,$$

and so the error derivatives are

$$\frac{\partial E}{\partial w_i} = \sum_n \frac{dE_n}{dy_n} \frac{\partial y_n}{\partial w_i} = -\sum_n (t_n - y_n)(1 - y_n) y_n x_{ni}.$$

6 Learning with Hidden Units

6.1 Simplifying Notations

Deliberately conflating a neuron y_j with layer j.

In a neural networking using logistic neurons with hidden layers, the neurons y_n are arranged in layers, with neurons in each layer receiving input from every neuron in the layer below, and outputting to every neuron in the layer above it. To simplify notations, y_i is used to denote any neuron in a fixed row i, and y_j to denote any neuron in the layer j above it.

The weights controlling how neurons in row i act on neurons in row j are w_{ij}, where i ranges over the neurons in row i and j ranges over neurons in row j. Furthermore, w_{ij} is also used to denote the weight of neuron y_i on neuron y_j, so that e.g. the logit for neuron y_j is

$$z_j = w_{-}y_{-} + \cdots + w_{ij}y_i + \cdots + w_{-}y_{-},$$

where each unnamed $w_{-}y_{-}$ is a weight and neuron in row i, and we're only interested in the weight and neuron $w_{ij}y_i$, so we give them names ij and i.

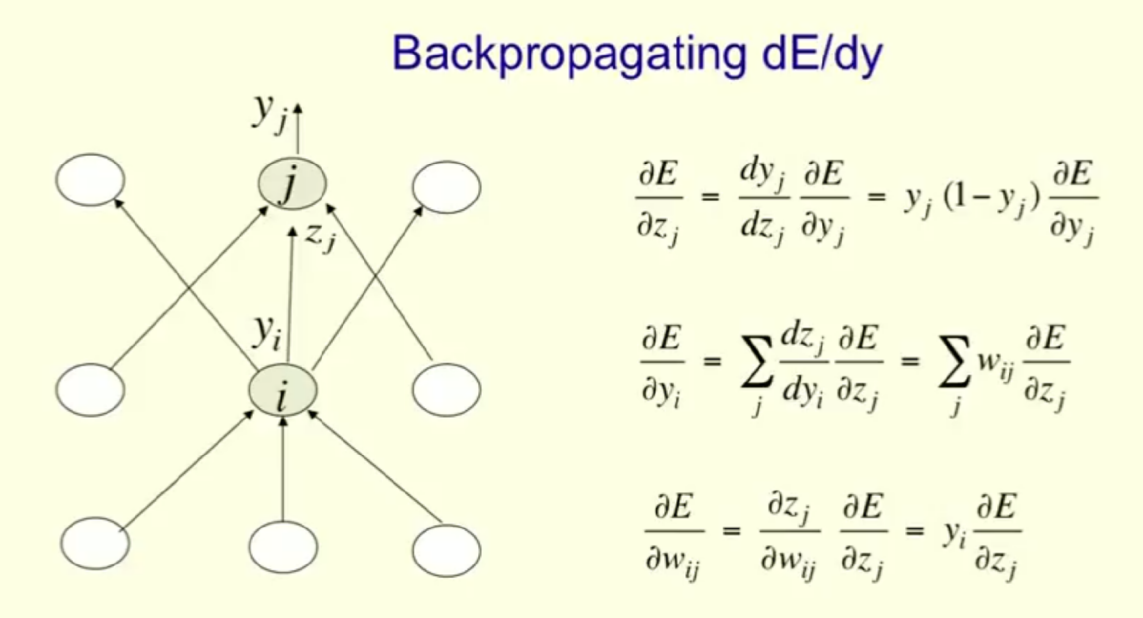


Figure 2: Model and main equations for Backpropagation.

7 Backpropagation Algorithm

Given an error E and a training case $x = y_1$, we want to compute the errors of E due to y_n for all neurons y_n, i.e. we want to compute $\frac{\partial E}{\partial w_{ij}}$ for all ij. With these partial derivatives calculated over all training cases, we can then apply Gradient Descent or some other optimization algorithm to *compute* the minimum weights W. [a]

[a] Keep in mind that there is a weight matrix w for each pair of layers i and j, where each entry w_{ij} corresponds to the connection between neuron y_i in layer i and neuron y_j in layer j. Therefore the weights of all connections together form a three-dimensional matrix W.

7.1 Analytic Solution vs Random Perturbation

Is this right?

An alternative to using Backpropagation is to randomly perturb the weights and see how each change affects the error, but that's a lot slower than solving for the optimal solution analytically, since you have to perturb the weights a large number of times just to see which change was the best.

7.2 Deriving the Error Derivatives

Definition 14. *For quick reference, once again the logit and estimator of neuron y_j in layer j are:*

$$z_j = w^T x = w\_y\_ + \cdots + w_{ij} y_i + \cdots + w\_y\_$$
$$y_j = \frac{1}{1 + e^{-z_j}}.$$

And the total error of the neural network is:

$$E = \frac{1}{2} \sum_n (t_n - y_n)^2.$$

It's like a recursive Chain Rule.

Proposition 15. *The error derivatives for a logistic neural network with hidden*

units are:

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{dy_j}{dz_j} = y_j(1-y_j)\frac{\partial E}{\partial y_j} \tag{1}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial y_i} = \sum_j w_{ij}\frac{\partial E}{\partial z_j} \tag{2}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}} = y_i\frac{\partial E}{\partial z_j}. \tag{3}$$

These equations are simple applications of the Chain Rule. But what could they possibly mean?!

7.3 Meaning of the Error Derivatives and How to use them

At layer i, consider E as a function of the z_j's in layer j. Since z_j depend on y_i, by the Chain Rule we have the second equation:

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial y_i} = \sum_j w_{ij}\frac{\partial E}{\partial z_j}.$$

To calculate $\frac{\partial E}{\partial z_j}$, we use equation (1):

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{dy_j}{dz_j} = y_j(1-y_j)\frac{\partial E}{\partial y_j},$$

which requires $\frac{\partial E}{\partial y_j}$, so we use equation (2) again to level up, and so on until the top output layer.

In practice we'd start at the top and calculate down, and along the way down we use the third equation to calculate $\frac{\partial E}{\partial w_{ij}}$. Once we have all of those we use the Delta Rule above to calculate the necessary weight changes:

$$\Delta w_{ij} = -\alpha\frac{\partial E}{\partial w_{ij}}.$$

Keywords

Sigmoid function, logit, Backpropagation Algorithm, weight perturbation, evolution.

References

[1] Geoffrey E. Hinton's Neural Networks video lectures.

[2] `http://www.cs.toronto.edu/~rgrosse/csc321/`

[3] Images from the Internet.