Advanced Machine Learning, Deep Learning, NLP & Graphical Models

Ryan Vaz

Hunter College

Abstract. This PDF is a collection of lecture notes from the Statistics 72556 class at Hunter College with some brief simplifications and details regarding how to approach Machine Learning from a Statistical POV. A firm grasp of Python Programming, Statistics, and Matrix/Linear Algebra is assumed. I'll try to minimize the usage of Calculus however some understanding will help certain topics.

Table of Contents

1 Lecture 1 - ANN

1.1 Machine Learning Problems

Unsupervised Learning A problem where only the features are observed.

Supervised Learning A problem in which both the features and the target are observed. We can further classify supervised learning into two categories...

Regression A case of supervised learning where the variable we are studying is continuous/numerical

Classification A case of supervised learning where the variable we are studying is categorical

Table 1. Examples of different machine learning problems and classifications

Problem Type	Example
Unsupervised	Market segmentation where we divide customers into groups based on their characteristics
Regression	Predicting the value of the DOW in 6 months Predicting the value of a given house based on various inputs
Classification	Will the DOW be up (T) or down (F) in 6 months? Is this email a spam(T) or not(F)?

1.2 Basic Neural Networks

Artificial Neural Networks(ANNs) A neural network is a computing system that is inspired by biological neural networks that make up the brain. They consist of multiple connected units called artificial neurons or linear threshold units.

Linear Threshold Unit(LTU) A single artificial neuron that calculates the net output from a series of inputs
It uses inputs(x) and weights(w) to calculate a weighted sum
That value is then applied to a binary step function which turns the result into a Boolean value.

Theorem 1 (Weighted Sum).
$z = w_1x_1 + w_2x_2 + ... + w_kx_k = \boldsymbol{x}^T\boldsymbol{w}$

Theorem 2 (Step Function).
$step(z) = \begin{cases} 0 & if\ z < 0 \\ 1 & if\ z \geq 0 \end{cases}$

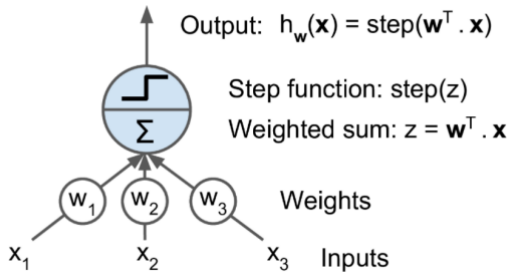

Fig. 1. Example of a LTU.

To summarize a single LTU: It takes in data from input layer and calculates output by multiplying it with the corresponding weight then summing it together and sending it through a step function(Binary Step Function in this case).

Perceptron A neural network formed from a single layer of LTUs
Each LTU is connected to all the input neurons along with an extra input neuron that is always 1(bias neuron). This bonds the Input Layer to the Output Layer.
These Perceptrons can output several values(Multioutput Classifier).

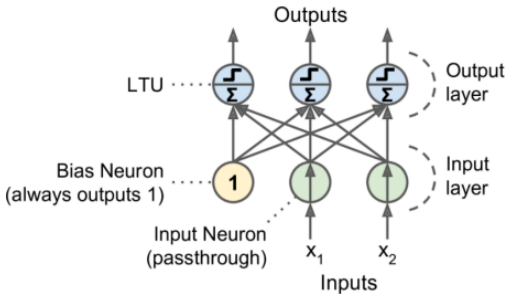

Fig. 2. Example of a Dense Perceptron.

If all the neurons in a layer are connected to every neuron in the previous layer, its a dense layer

Theorem 3 (Output of a Dense Layer).
$h_{W,b}(X) = \phi(XW + b)$

ϕ - activation function
X - inputs, W - weights, b - bias vector

1.3 Deep Neural Networks

Perceptron Training It's commonly believed that when biological neurons trigger others, it reinforces the connection between these two, the same logic is what's used for perceptrons.
To train perceptrons, they're given a training instance and it makes a prediction.
When it makes a WRONG prediction, this reinforces the connection between the inputs that would have contributed to a correct prediction.
This can be expressed mathematically as follows:

Theorem 4 (Weights Update).
$w_{i,j}^{next\ step} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i$

$w_{i,j}$ is the connection weight between the i-th input neuron and the j-th output neuron
x_i is the i-th input value of the current training instance
$\hat{y}_j$ is the output of the j-th output neuron for the current training instance
y_j is the target output of the j-th output neuron for the current training instance
η is the learning rate

Deep Neural Networks(DNNs) When a perceptron has multiple layers of LTUs(or hidden layers) that are fully connected to the next layer with a bias on each level, we call that a Multilayer Perceptron(MLP) or Deep Neural Network(DNN).

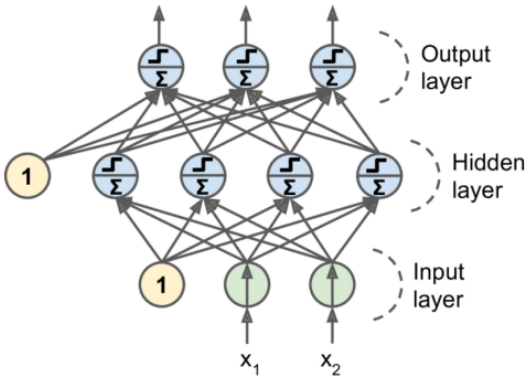

Fig. 3. Example of a DNN.

1.4 Error Backpropagation Learning Algorithm

The next set of topics is going to be significantly harder, so if you want to get the spark notes and skip to the coding section, they go as follows: in order to train DNN's a backpropagation training algorithm is used, which is basically gradient descent using reverse-mode autodifferentiation.

Note: This section is going to be different in that we will simulate each step of the algorithm

Initialization The weights of the input layer are initialized with random small values.

Forward pass The input layer is provided and the neural network begins processing the data. The difference between expected and actual value is provided directly to the backwards pass step.

Reverse Pass This step computes how much each neuron in the last hidden layer contributed to each output neuron's error and it tweaks the weights according to a gradient descent(coming soon!).

It then repeats the initialization, forward pass and backwards pass which is why this whole process is called backpropagation, or backprop for short as it propagates the error back to the weights of the neurons that contributed to the error.

Now the question remains, how do we minimize this error function?

1.5 Gradient Descent(GD)

Yes, Gradient Descent gets it's whole own section, mad? Stay mad.

Definition GD is a optimization algorithm that starts with a random value on a graph and will try tweaking the parameters in order to minimize a cost/error function
By using GD, we can find the minimum amount of error to best use with our DNN.

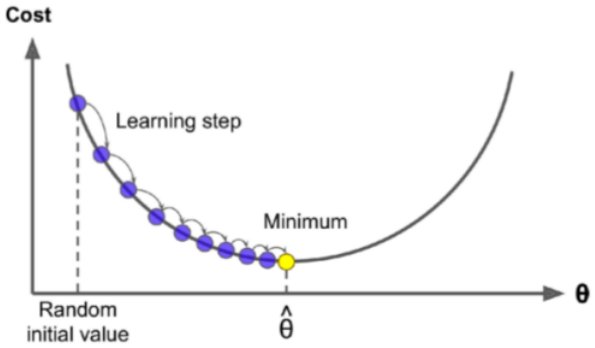

Fig. 4. Example of Gradient Descent.

Learning Rate(η) Learning rate will determine the size of the steps. If it's too small, it won't be able to get a solution in a reasonable amount of iterations. If it's too big, it may skip the final result entirely and diverge!
Learning rate is so important can make or break our code, thus it is a hyper-parameter. Our code has other hyper-parameters that we will address later on.

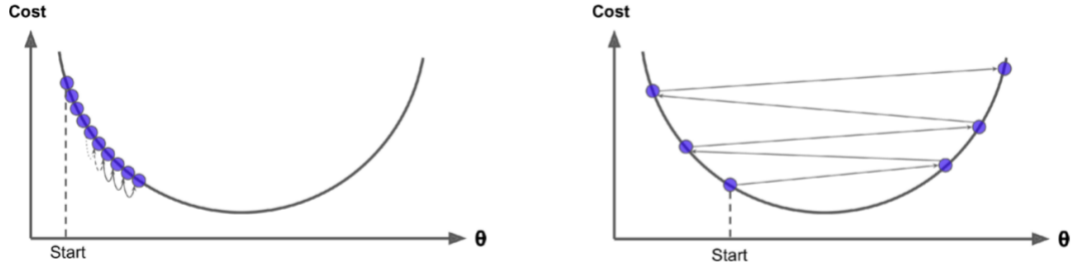

Fig. 5. Example of bad learning rates.

Math Problem: Minimize f(w) - weight/error function
iterative solution: $w_{k+1} = w_k - \gamma_k \nabla f(w_k)$
w_{k+1} = next iteration

w_k = current iteration
γ_k = step size
$\nabla f(w_k)$ = gradient of f
One such example is our linear regression problem, let's switch w with θ
$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_k = h_\theta(x) = \theta^T \mathbf{w}$
We need to find $\hat{\theta}$ which is the minimizes the criterion(loss)
$f(\lambda_k) = MSE(X, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}(\theta^T x^i - y^i)^2, \nabla f = \frac{2}{m} X^T(X\theta - y)$
Gradient Descent Step Equation
$\theta^{nextstep} = \theta - \eta \nabla_\theta MSE(\theta)$
where

$$\nabla_\theta MSE(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} MSE(\theta) \\ \frac{\partial}{\partial \theta_1} MSE(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} MSE(\theta) \end{pmatrix} = \frac{2}{m} X^T(X\theta - y)$$

Implementation in code:

```
eta = 0.1 #learning rate
n_iterations = 1000
m = 100

theta = np.random.randn(2,1)

for iter in range(n_iterations):
    gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
    theta -= (eta * gradients)
```

2 Lecture 2 - MLP Architecture

2.1 Math

MLP as a Composite Function Imagine each layer of the Neural Network is a function, then we can say that function n maps to function k with the invertable linear map $f : R^n \to R^k$ where f is expressed as the following

Theorem 5. $f(x) = g \circ f_N \circ f_{N-1} ... \circ f_2 \circ f_1(x), \quad f_i : R^{i-1} \to R^i$
where $f_i(x) = \sigma(w_i x + b_i), i = 1, 2, ...N$
$\implies f(x) = g(\sigma(...(\sigma(w_2 \sigma(w_1 x + b_1) + b_2) + ... + b_N)$

Where σ is a non-linear activation function and g(x) is the linear decision function.

Autodiff s.p.z. we have a function that maps n dimensional values to a scalar output, and we have $x_1, x_2, ..., x_n$ as inputs, and $x_{n+1}, ..., x_{N-1}$ as intermediate values, and x_N to be the final value.
Example problem: Compute $f(x), L(f(x), y)$, and dL/dw_i

1. Input $\mathbf{x}$
2. Set $v_{1,..n} \leftarrow x$
3. For i = n+1, n+1, ..., N:
$$v_i \leftarrow \sigma_i(w_i \cdot v_{Pa(i)})$$
$$v_i' \leftarrow \sigma_i'(w_i \cdot v_{Pa(i)})$$
4. Set $L \leftarrow L(f, y)$
5. Compute $\frac{dL}{df}$
6. Initialize $\frac{dL}{dv_i} \leftarrow 0$
7. For m = 1, 2, ..., M:
$$\frac{dL}{dv_{N-M+m}} \leftarrow \frac{dL}{df_m}$$
8. For i = N, N-1, ..., n+1:
$$\frac{dL}{dw_i} \leftarrow \frac{dL}{dv_i} v' v_{Pa(i)}$$
$$\frac{dL}{dv_{Pa(i)}} \leftarrow \frac{dL}{dv_{Pa(i)}} + \frac{dL}{dv_i} v' w_i$$

2.2 Hyperparameters

Hyperparameter Examples

1. # of hidden layers
2. # of neurons per hidden layer
3. Learning rate(η)
4. Optimizer
5. Batch size
6. Activation functions

Activation Functions

1. ReLU (most used)
2. Hyperbolic tangent function (tanh)
3. Logit function
4. Sigmoid (not as good as ReLU, but biologically more accurate)

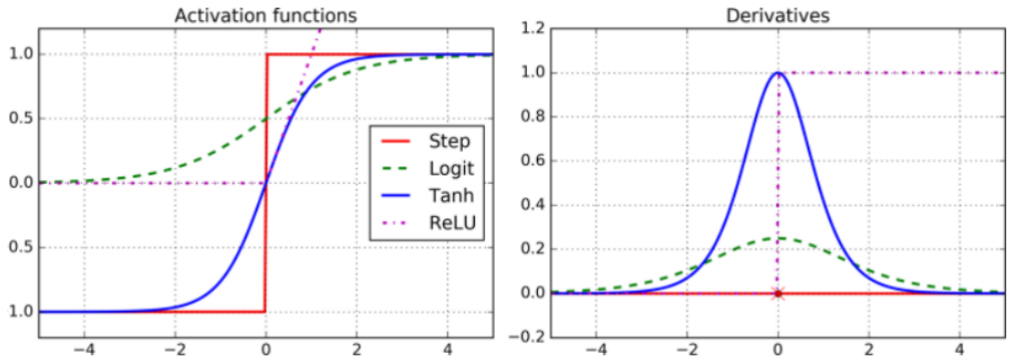

Fig. 6. Comparison of activation functions.

Regression MLP Architecture When Using Regression, we do not need an activation function, and furthermore the loss function is normally Mean Square Error(MSE). Due to no activation function, it's easy to think of Regression MLP's as a MLP with the identity activation function.

Table 2. Examples of standard values for Hyperparameters

Hyperparameter	Typical Value
# input neurons	One per input feature
Regression	Predicting the value of the DOW in 6 months Predicting the value of a given house based on various inputs
Classification	Will the DOW be up (T) or down (F) in 6 months? Is this email a spam(T) or not(F)?

Classification MLP Architecture Classification is best used when outputs can be grouped catagorically, there are 3 different classifications that we will focus on.

Table 3. Types of Classification MLP

Classification Model	Use Case
# Binary	A probability of it being the assumed output class
Multilabel Binary	Similar to binary, but using a 2 output classes
Multiclass	Also similar to binary, but using $>$ 2 output classes

Softmax