Bioinformatics: Neural Networks

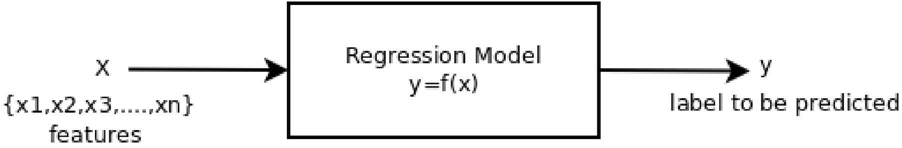
Swakkhar Shatabda

Department of Computer Science and Engineering
BRAC University

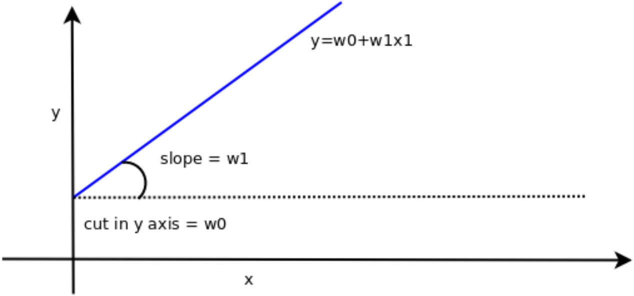
References

Inspiring Excellence

1. In regression problems, we are given data as X and labels as y.
2. Here, we are upto learn a model where, y will be predicted as a function of X.
3. In regression, the label y is numeric and continuous in value.
4. For example, suppose you are given many features of a protein, like toxicity, inflammation, gram-positive/negative, localization, structure and you have to predict its lowest energy state value. This problem can be formulated as a regression problem.



Simple Linear Regression



- Here, w_0 is the value of y, when $x_1 = 0$, this could be either positive or negative
- Also note, w_1 is the rate of change of y, w.r.t. x_1
- We have to predict the relationship between x_1 and y, and we assume it to be linear.
- Here are problem is to estimate the correct values of w_0 and w_1

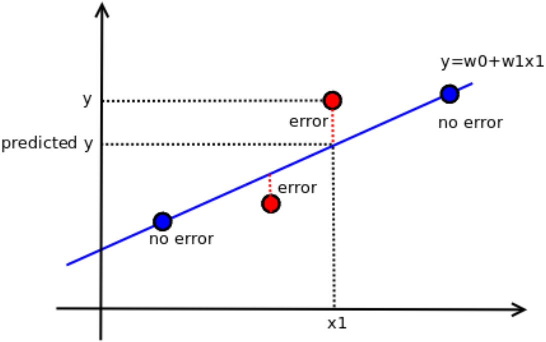
Error in Prediction

- We are going to formulate this estimation as an optimization problem, we will define an error and our goal will be to find such w_0, w_1 so that the error is minimized.
- The error function,

$$e = \frac{1}{2}\sum_{i=1}^{m}(\hat{y}(i) - y(i))^2$$

- Here, $\hat{y}(i)$ is the predicted label and $y(i)$ is the real label for a given instance or data i.
- We square it to negate the sign and put a half before for a mathematical convenience.

Explanation of Error



- Here the blue ones are correctly predicted by the line and thus have no error and the red ones are with errors.
- So error is the difference between real y and predicted $\hat{y} = w_0 + w_1 x_1$
- We can write,

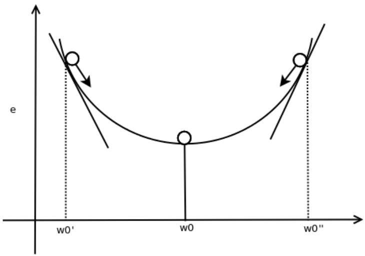
$$e = \frac{1}{2}\sum_{i=1}^{m}((w_0 + w_1 x_1(i)) - y(i))^2$$

- Our task is to find w_0, w_1 so that the error e is minimized.

Finding w_0, w_1

- We call these co-efficients or weights.
- We are going to use gradient descent algorithm to find these values.
- The function is minimized at a point where the slope is zero.
- We randomly start from any value of w_0 or w_1 and eventually reach the minimum.
- Lets see how that is done!

Intution for Gradient Descent



- We could either start at w_0' or at w_0'' but we wish to reach w_0
- From w_0', we have to increase the value and move right and here at this point slope of the tangent is negative.
- From w_0'', we have to decrease the value and move left and here at this point slope of the tangent is positive.
- Its interesting to note that, more the distance from the point to the minimum is the value is slope is larger. We thus can change the weights proportionate to the slope.

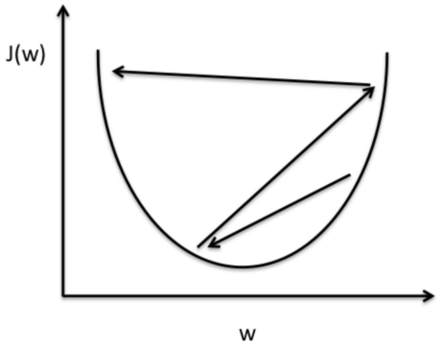
Intution for Gradient Descent

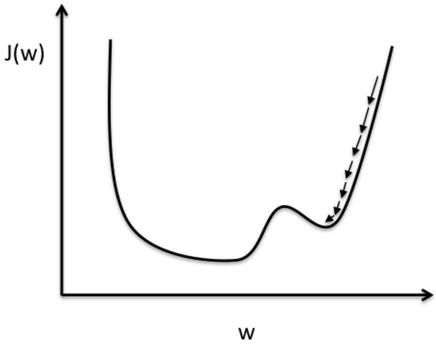
- However, a large value of slope might drastically change the value. We can minimize that effect by using a learning constant, α.
- Task of α is to control the changes of values of weights.
- The smaller the value of α is, slower the movement/change is. Again too high value will cause divergence.
- The increase or decrease in the values will be decided by the sign of the slope
- In general, we can apply the following in iterations:

$$w_0(\text{new value}) = w_0(\text{old value}) - \alpha \frac{\delta e}{\delta w_0}$$

$$w_1(\text{new value}) = w_1(\text{old value}) - \alpha \frac{\delta e}{\delta w_1}$$

Learning Rate





Large learning rate: Overshooting.

Small learning rate: Many iterations until convergence and trapping in local minima.

Figure Soruce: `https://sebastianraschka.com/images/blog/2015/singlelayer_neural_networks_files/perceptron_learning_rate.png`

Finding Slopes

- To find slope we need to differentiate the following equation:

$$e = \frac{1}{2}\sum_{i=1}^{m}((w_0 + w_1 x_1(i)) - y(i))^2$$

- With respect to w_0,

$$\frac{\delta e}{\delta w_0} = \sum_{i=1}^{m}((w_0 + w_1 x_1(i)) - y(i)).1$$

- With respect to w_1,

$$\frac{\delta e}{\delta w_1} = \sum_{i=1}^{m}((w_0 + w_1 x_1(i)) - y(i)).x_1(i)$$

- Now, we will try to extend this for multi-variable linear regression:

$$\hat{y}(i) = w_0 + w_1 x_1(i) + w_2 x_2(i) + \cdots + w_n x_n(i)$$

- And, now we write a general equation for slope for a weight w_i:

$$\frac{\delta e}{\delta w_j} = \sum_{i=1}^{m}(\hat{y}(i) - y(i)).x_j(i)$$

- Note, each time the error in prediction is multiplied by 1 (for w_0) or multiplied by the feature x_j (for w_j)

BRAC UNIVERSITY
Inspiring Excellence

Gradient Descent

GradientDescent$(X, y, alpha, maxIter)$

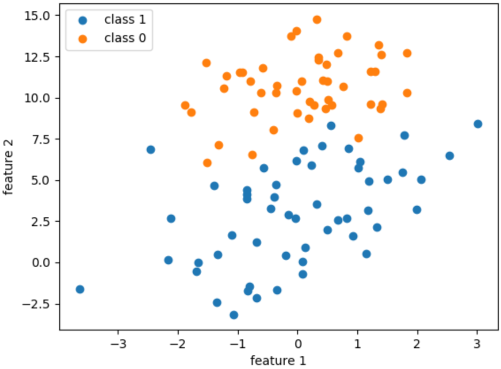
$W = [w_0, w_1, \cdots, w_n]$ initialized randomly
$iter = 0$
while $iter++ \leq maxIter$
    $\hat{Y} = X_E \times W$
    $J = \hat{Y} - Y$
    $Slope = \frac{\delta J}{\delta W}$
    $W = W - \alpha \times Slope$
return $w_0, w_1, \cdots, w_n$

Classification

1. In classification problems, we are given data as X and labels as y.
2. Here, we are upto learn a model where, y will be predicted as a function of X.
3. In classification, the label y is categorical or discrete in value.
4. For example, suppose you are given many features of a protein, like toxicity, inflammation, gram-positive/negative, localization, structure and you have to predict whether this protein is anti-viral or not. This problem can be formulated as a classification problem.

Example

We will first try to predict the class of the dataset based on two features, x_1 and x_2.



Logistic Regression

At first, we are going to try a linear classifier called logistic regression. We can apply logistic regression when the data is linearly separable.

- The relationship will be predicted as:

$$y = w_0 + w_1 x_1$$

- This is again an equation of a straight line
- We need the best line that separates blue from the orange
- learn $w_0, w_1, \cdots$
- Can we use gradient descent here? A little trick required!

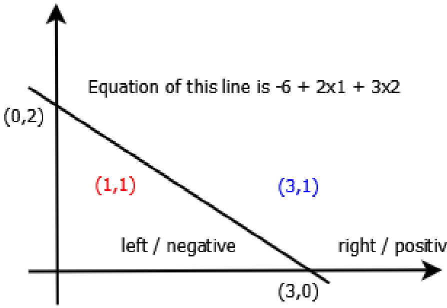
Gradient Descent for Logistic Regression

- The cost function / loss function of gradient descent

$$e = \frac{1}{2}\sum_{i=1}^{m}(\hat{y}(i) - y(i))^2$$

- This time too predicted label $\hat{y}$ is a function of $\vec{x}$ and w
- The labels are discrete, for this binary classification two labels 0 (no or negative) and 1 (yes or positive)
- Now, we try to define $\hat{y}$ with help of the weights or coefficients of the line.

Linear Classification



- This linear classifier divides instances based on the local wrt the line, on the right positive, negative on the left
- Any point on the line satisfies the equation. Any point on the right (3,1) yields positive result and any point on the left (1,1) yields negative result.
- Based on this we can define a linear classifier

Linear Classification

This following function will help us in making decision:

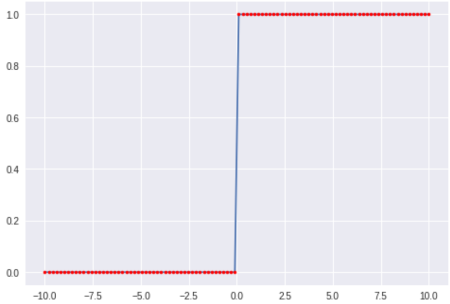
$$f(\vec{x}) = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

LinearClassifier

if $f(\vec{x}) > 0$ or $w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n > 0$
    return 1
else return 0

This simple classifier just checks whether a point is on the left or right.

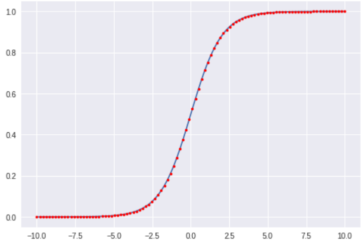
A step function!



Alas! This is not a continuous function and thus not differentiable. We can't calculate gradients! We need to find an alternate!

A sigmoid function!

$\sigma(\vec{x}) = \frac{1}{1+exp(-\vec{x})}$

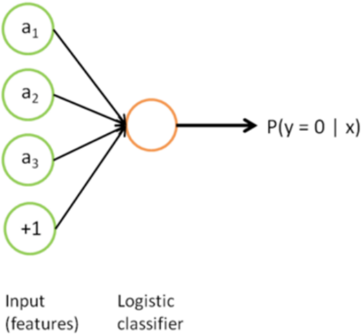


Good things about sigmoid!

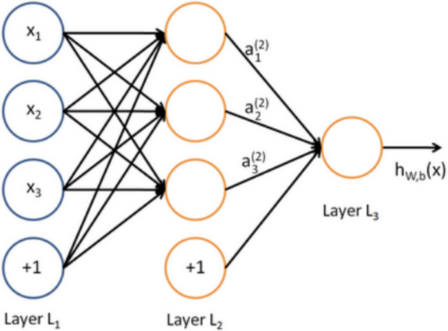
1. Its continuous and differentiable.
2. $\sigma'(\vec{x}) = \sigma(\vec{x})(1 - \sigma(\vec{x}))$

BRAC UNIVERSITY
Inspiring Excellence

Towards Neural Network

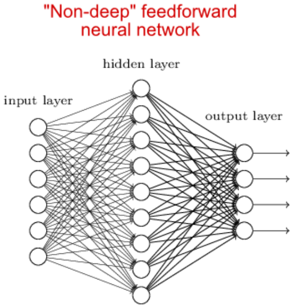


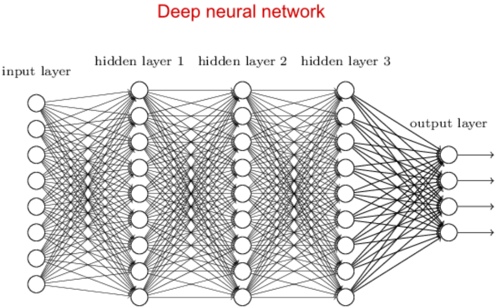
Logistic Regression



Neural Network

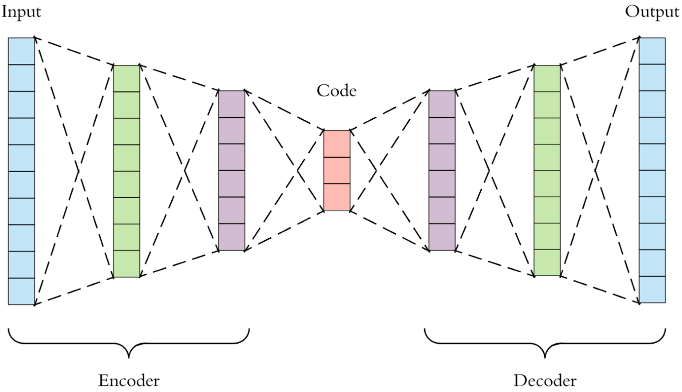
Wide vs Deep Networks





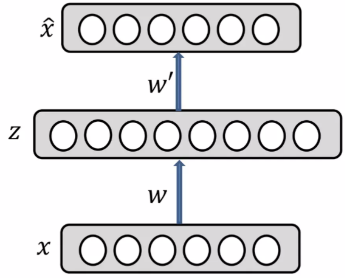
Auto-encoders

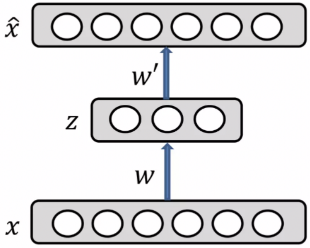
- An autoencoder is a type of artificial neural network used to learn efficient codings of unlabeled data (unsupervised learning).
- An autoencoder learns two functions: an **encoding function** that transforms the input data, and a **decoding function** that recreates the input data from the encoded representation.



Undercomplete Autoencoders

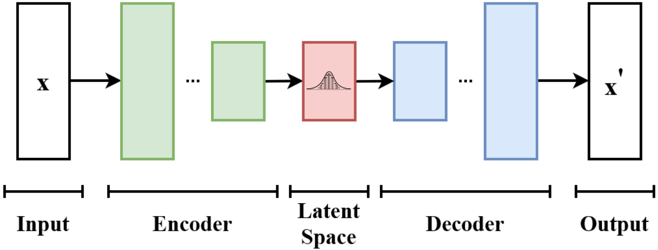
- Undercomplete Autoencoders intentionally restrict the size of the hidden layer to be smaller than the input layer.
- This bottleneck forces the model to compress the data helps in learning only the most significant features and discarding redundant information.
- The model is trained by minimizing the reconstruction error while ensuring the latent space remains compact.





Variational Autoencoder

- Variational Autoencoder (VAEs) extend traditional autoencoders by learning probabilistic latent distributions instead of fixed representations.
  - Reconstruction loss to ensure accurate data reconstruction.
  - KL Divergence to regularize the latent space towards a standard Gaussian helps in preventing overfitting and smooth latent structure.



BRAC UNIVERSITY
Inspiring Excellence

Denoising Autoencoder

- Denoising Autoencoders are designed to handle corrupted or noisy inputs by learning to reconstruct the clean, original data.