Introduction to Neural Network

Dr. Mahmoud N Mahmoud
mnmahmoud@ncat.edu

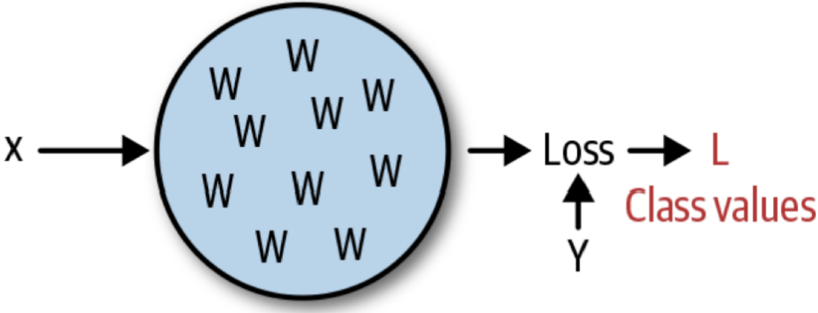
North Carolina A & T State University

November 26, 2022

Outline

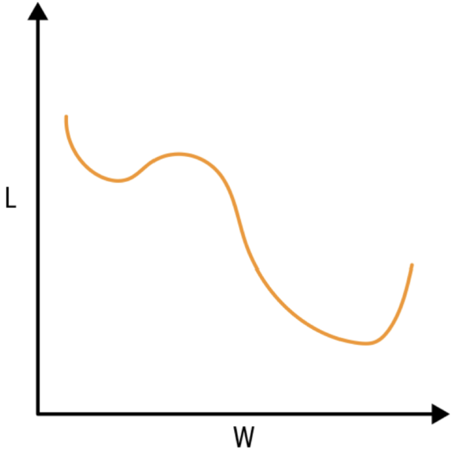
1. Introduction
2. Neural Networks for Classification
3. The Softmax Cross Entropy Loss Function
   - The Softmax Function
   - The Cross Entropy Loss
4. Activation Functions
5. Learning Strategies
6. Weight Intitialization

Some Intuition About Neural Networks

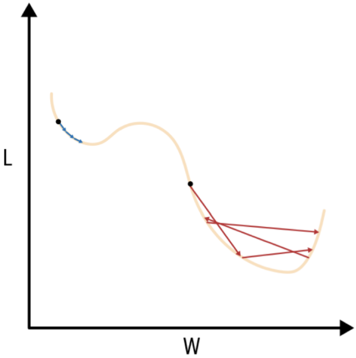


Some Intuition About Neural Networks

In reality, each individual weight has some complex, nonlinear relationship with the loss L.



Some Intuition About Neural Networks



- The blue arrows represent repeatedly applying the gradient update rule with a smaller learning rate than the red arrows on the right.
- The goal of training a deep learning model is to move each weight to the "global" value for which the loss is minimized

Some Intuition About Neural Networks

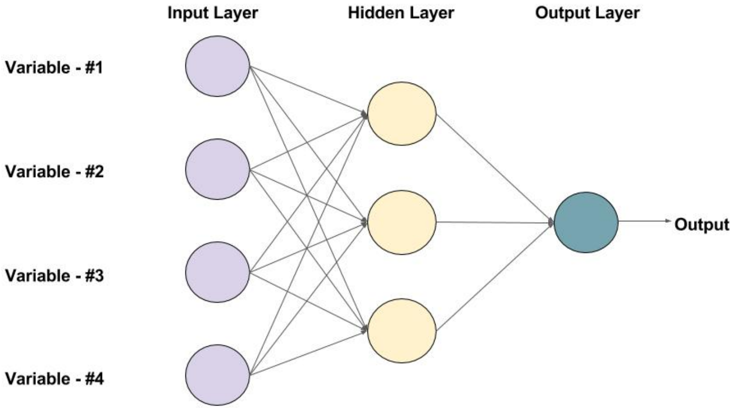
In reality, the picture is far more complicated than this.

- we are searching for a global minimum in a space that has thousands or millions of dimensions (weights).
- Moreover, since we update the weights on each iteration as well as passing in a different X and y, the curve we are trying to find the minimum of is constantly changing!

Outline

1. Introduction
2. Neural Networks for Classification
3. The Softmax Cross Entropy Loss Function
   - The Softmax Function
   - The Cross Entropy Loss
4. Activation Functions
5. Learning Strategies
6. Weight Intitialization

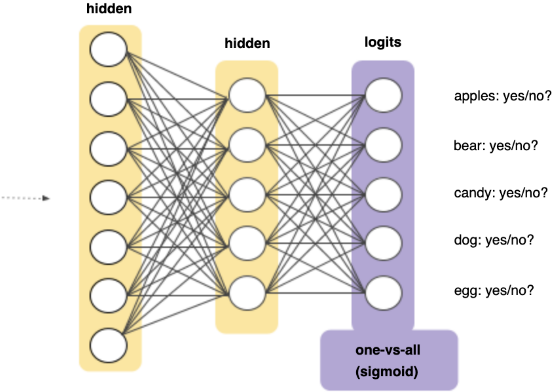
Binary Classification



An example of a Feed-forward Neural Network with one hidden layer (with 3 neurons)

(One vs All) Multi-class Classification

Given a classification problem with N possible solutions, a one-vs.-all solution consists of N separate binary classifiers



Note. The output vector has to be encoded using one hot encoding

Outline

1. Introduction

2. Neural Networks for Classification

3. The Softmax Cross Entropy Loss Function
   - The Softmax Function
   - The Cross Entropy Loss

4. Activation Functions

5. Learning Strategies

6. Weight Intitialization

More Accurate Multi-class Classification

Previously, we used sigmoid with mean squared error (MSE) as loss function

- Pros
  - Convex (i.e., steeper gradient as you are away from the correct output)
  - Acceptable performance
- Cons
  - Outputs can not be interpreted as probability.
  - Does not work with all problem types

More Accurate Multi-class Classification

Previously, we used sigmoid with mean squared error (MSE) as loss function

- Pros
  - Convex (i.e., steeper gradient as you are away from the correct output)
  - Acceptable performance
- Cons
  - Outputs can not be interpreted as probability.
  - Does not work with all problem types

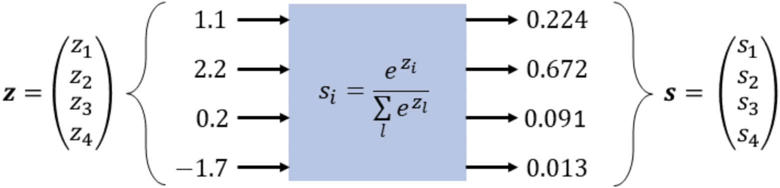
In practice, we use Softmax + cross Entropy loss

Outline

1. Introduction
2. Neural Networks for Classification
3. The Softmax Cross Entropy Loss Function
   - The Softmax Function
   - The Cross Entropy Loss
4. Activation Functions
5. Learning Strategies
6. Weight Intitialization

Softmax Fuction

- Softmax extends binary logistic regression idea (probability adds upto 1) into a multi-class world.
- It helps training converge more quickly than it otherwise would.


$\mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix}$
1.1
2.2
0.2
-1.7
$s_i = \frac{e^{z_i}}{\sum_l e^{z_l}}$
0.224
0.672
0.091
0.013
$\mathbf{s} = \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix}$

How it works?

Suppose we have an input vector from the previous layer

$$[5, 3, 2]$$

How it works?

Suppose we have an input vector from the previous layer

$$[5, 3, 2]$$

One way to transform these values into a vector of probabilities

$$\text{Normalize}\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = \begin{bmatrix} \frac{x_1}{x_1+x_2+x_3} \\ \frac{x_2}{x_1+x_2+x_3} \\ \frac{x_3}{x_1+x_2+x_3} \end{bmatrix}$$

How it works?

Suppose we have an input vector from the previous layer

$$[5, 3, 2]$$

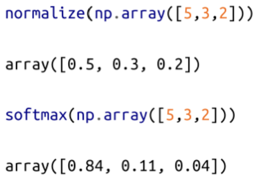
One way to transform these values into a vector of probabilities

$$\text{Normalize}\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = \begin{bmatrix} \frac{x_1}{x_1+x_2+x_3} \\ \frac{x_2}{x_1+x_2+x_3} \\ \frac{x_3}{x_1+x_2+x_3} \end{bmatrix}$$

A better way

$$\text{Softmax}\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = \begin{bmatrix} \frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}} \\ \frac{e^{x_2}}{e^{x_1}+e^{x_2}+e^{x_3}} \\ \frac{e^{x_3}}{e^{x_1}+e^{x_2}+e^{x_3}} \end{bmatrix}$$

Intuition



```
normalize(np.array([5,3,2]))

array([0.5, 0.3, 0.2])

softmax(np.array([5,3,2]))

array([0.84, 0.11, 0.04])
```

- Softmax is more strongly amplifies the maximum value relative to the other values.
- Softmax is partway between normalizing the values and actually applying the max function!

Outline

1. Introduction
2. Neural Networks for Classification
3. The Softmax Cross Entropy Loss Function
   - The Softmax Function
   - The Cross Entropy Loss
4. Activation Functions
5. Learning Strategies
6. Weight Intitialization

Cross Entropy

Recall that any loss function will take in a vector of probabilities $\begin{bmatrix} p_1 \\ \vdots \\ p_n \end{bmatrix}$

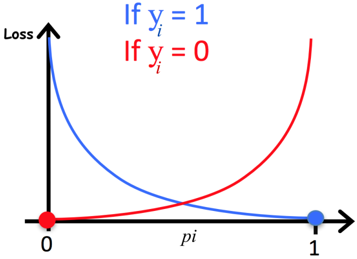
and a vector of actual values $\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$

The cross entropy loss function, for each index i in these vectors, is:

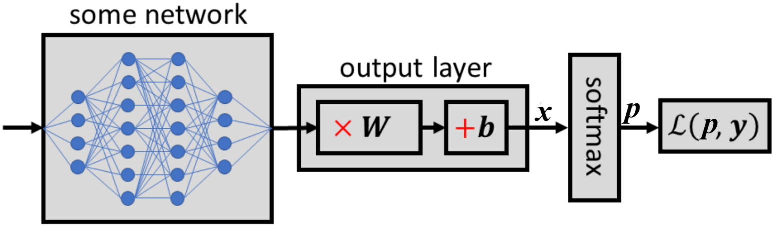
$$\ell = -y_i log(p_i) - (1 - y_i) log(1 - p_i)$$

Intuition

- Since p_i is a probability between 0 and 1.
- Our loss can be defined as follows.
  - if yi = 1 : Loss = -log(pi)
  - if yi = 0 : Loss = -log(1 - pi)



Our Neural Network



Gradient Computation

The real magic happens when we combine this loss with the softmax function

$$SCE_1 = -y_1 \times log\left(\frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}}\right) - (1-y_1) \times log\left(1-\frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}}\right)$$

Gradient Computation

The real magic happens when we combine this loss with the softmax function

$$SCE_1 = -y_1 \times log\left(\frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}}\right) - (1-y_1) \times log\left(1 - \frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}}\right)$$

Based on this expression, the gradient would seem to be a bit trickier for this loss. (Proof Is Required)

$$\frac{\partial SCE_1}{\partial x_1} = \frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}} - y_1$$

Gradient Computation

The real magic happens when we combine this loss with the softmax function

$$SCE_1 = -y_1 \times log\left(\frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}}\right) - (1-y_1) \times log\left(1 - \frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}}\right)$$

Based on this expression, the gradient would seem to be a bit trickier for this loss. (Proof Is Required)

$$\frac{\partial SCE_1}{\partial x_1} = \frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}} - y_1$$

That means that the total gradient to the softmax cross entropy is

$$\mathrm{softmax}\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) - \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Outline

1. Introduction
2. Neural Networks for Classification
3. The Softmax Cross Entropy Loss Function
   - The Softmax Function
   - The Cross Entropy Loss
4. Activation Functions
5. Learning Strategies
6. Weight Intitialization

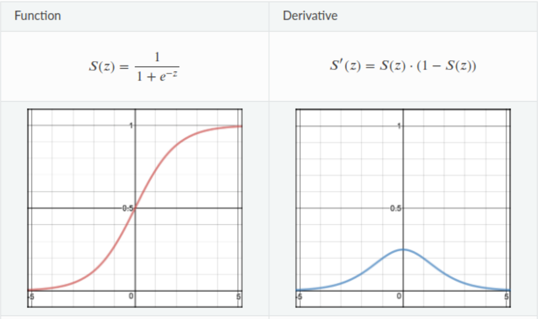
Activation functions

Activation functions typically have the following properties:

1. **Non-linear:** To model complex relationships
2. **Continuously Differentiable:** To improve our model with gradient descent.
3. **Fixed Range Activation** functions typically squash the input data into a narrow range that makes training the model more stable and efficient.

Sigmoid / Logistic

- The stacking is possible.
- Smooth gradient.
- Maximum gradient is 0.25 proof? (Successive sigmoid vanish backward gradients)
- Computationally expensive.

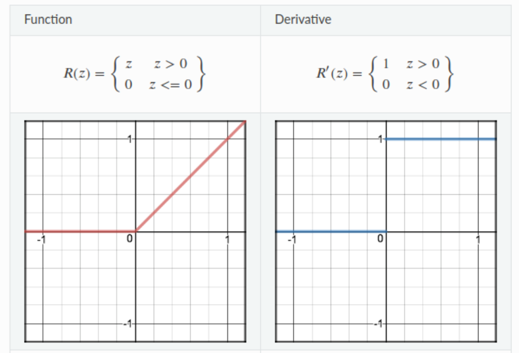


Function	Derivative
$S(z) = \frac{1}{1+e^{-z}}$	$S'(z) = S(z) \cdot (1 - S(z))$

1
0.5
-5
0
5
1
0.5
-5
0
5

ReLU (Rectified Linear Unit) - Other Extreme

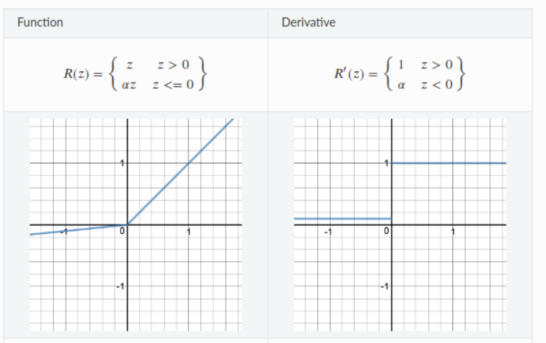
- Computationally cheap.
- Larger gradients
- Suffer from the Dying ReLU problem—when inputs approach zero, or are negative.
- Gradient not smooth.
- It can blow up the activation.



Function	Derivative
$R(z) = \begin{Bmatrix} z & z > 0 \\ 0 & z <= 0 \end{Bmatrix}$	$R'(z) = \begin{Bmatrix} 1 & z > 0 \\ 0 & z < 0 \end{Bmatrix}$

Leaky ReLU

- Computationally cheap.
- No Dying ReLU problem.
- Sometimes results are not consistent.
- Gradient not smooth.
- It can blow up the activation.

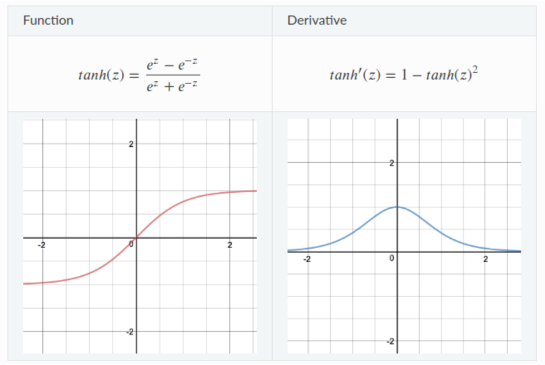


Function	Derivative
$R(z) = \left\{ \begin{array}{ll} z & z > 0 \\ \alpha z & z <= 0 \end{array} \right\}$	$R'(z) = \left\{ \begin{array}{ll} 1 & z > 0 \\ \alpha & z < 0 \end{array} \right\}$

Leaky ReLU

TanH / Hyperbolic Tangent - Happy Medium

- The stacking is possible.
- Smooth gradient.
- Zero centered output
- Gradinet is steeper than the sigmoid.
- Computationally expensive.



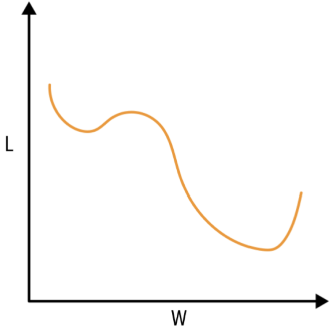
Function	Derivative
$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$tanh'(z) = 1 - tanh(z)^2$

Outline

1. Introduction
2. Neural Networks for Classification
3. The Softmax Cross Entropy Loss Function
    - The Softmax Function
    - The Cross Entropy Loss
4. Activation Functions
5. Learning Strategies
6. Weight Intitialization

Momentum

- In the real world objects rolling down a hill don't just depend on the hill slope to obtain their velocities (updates)
- However,their velocity at a given instant is a function not just of the forces acting on them in that instant but also of their accumulated past velocities, with more recent velocities weighted more heavily



Momentum

The parameter update at each time step will be a weighted average of the parameter updates at past time steps, with the weights decayed exponentially

$$\mathrm{update} = \square_t + \mu \times \square_{t-1} + \mu^2 \times \square_{t-2} + \ldots$$

Momentum

The parameter update at each time step will be a weighted average of the parameter updates at past time steps, with the weights decayed exponentially

$$\mathbf{update} = \square_t + \boldsymbol{\mu} \times \square_{t-1} + \boldsymbol{\mu}^2 \times \square_{t-2} + \dots$$

If our momentum parameter μ was 0.9, we would multiply the gradient from one time step ago by 0.9, the one from two time steps ago by $0.9 = 0.81$, the one from three time steps ago by $0.9 = 0.729$

How do we implement this?

Do we have to keep track the entire update history for each parameter?

How do we implement this?

Do we have to keep track the entire update history for each parameter?

We need only one variable to track the update (lets say "velocity"). Then we follow the following steps.

1. Multiply it by the momentum parameter.
2. Add the gradient

How do we implement this?

Do we have to keep track the entire update history for each parameter?

We need only one variable to track the update (lets say "velocity"). Then we follow the following steps.

1. Multiply it by the momentum parameter.
2. Add the gradient

This results in the velocity taking on the following values at each time step, starting at t = 1:

1. $\square_1$
2. $\square_2 + \mu \times \square_1$
3. $\square_3 + \mu \times (\square_2 + \mu \times \square_1) = \mu \times \square_2 + \mu^2 \times \square_1)$

Learning Rate Decay

- Learning rate decay can give us more fine-grained control over the training process.
  - In the beginning of the training we take large steps
  - While approaching the minimum we decrease the learning rate (prevent overshooting the minimum)
- Two types
  - Linear decay.
  - Exponential decay.

Linear Decay

The learning rate declines linearly from its initial value to some terminal value.

$$\alpha_t = \alpha_{start} - (\alpha_{start} - \alpha_{end}) \times \frac{t}{N}$$

where N is the total number of epochs.

Exponential Decay

The learning rate declines exponentially by a constant proportion each epoch

$$\alpha_t = \alpha_{start} \times \delta^t$$

where:

$$\delta = \frac{\alpha_{end}}{\alpha_{start}}^{\frac{1}{N}}$$

N is the total number of epochs.

Outline

1 Introduction

2 Neural Networks for Classification

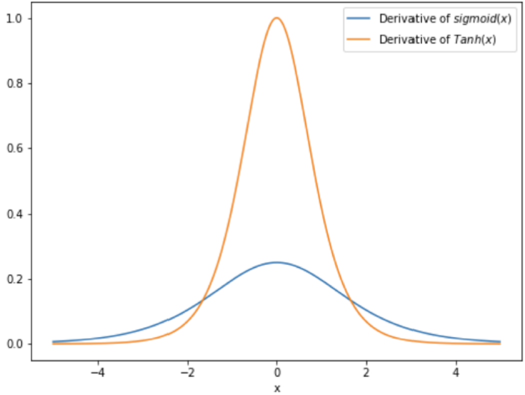
3 The Softmax Cross Entropy Loss Function
- The Softmax Function
- The Cross Entropy Loss

4 Activation Functions

5 Learning Strategies

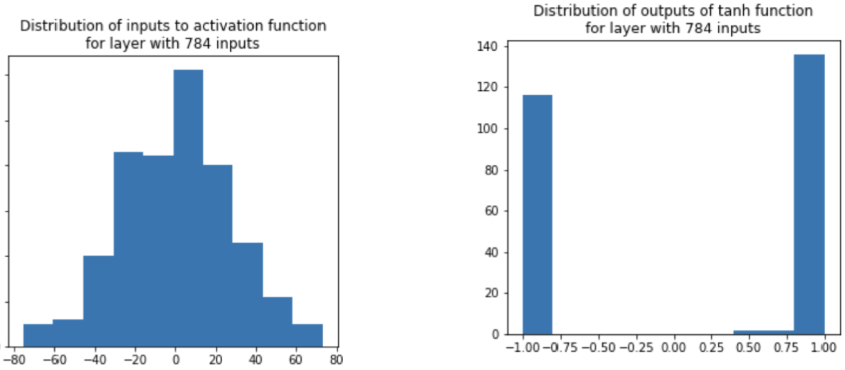
6 Weight Intitialization

Weight Intitialization (Intiuation)



Sigmoid and Tanh, have their steepest gradients when their inputs are 0, with the functions quickly flattening out as the inputs move away from 0

Weight Intitialization (Intiuation)



The hidden layer in the MNIST with 784 inputs.

Weight Intitialization (Intiuation)

For a given neuron

$$f_n = w_{1,n} \times x_1 + ... + w_{784,n} \times x_{784} + b_n$$

Assume we are using guassian weight initialization with $\mu = 0$ and $\sigma = 1$ then

$$\mathrm{Var}(f_n) = 785$$

Or the standard deviation is around $\sqrt{785} \approx 28$, which justify the previous figure

Weight Intitialization (Solution)

- Idea is to reduce the variance of each weight variable instead of 1 to be some fraction
- Glorot initialization [1] is proposed to address this problem
- Simply use the following variance for the weight matrix between two layers with n_{in} neurons and n_{out} neurons

$$\frac{2}{n_{in}+n_{out}}$$

[1] Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, 2010.

Questions ⸮