3

Forward and Back Propagation

x1, w1, x2, w2, b → $z = x_1 * w_1 + x_2 * w_2 + b$ → $\hat{y} = a = \sigma(z)$ → $\ell(a, y)$

$$\frac{\partial \ell}{\partial z} = \frac{\partial \ell}{\partial a} \cdot \frac{\partial a}{\partial z}$$
$$= \{ -y / a + (1 - y) / (1 - a) \} * a * (1-a)$$
$$= a - y$$

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z) \cdot (1 - \sigma(z))$$
$$\frac{\partial a}{\partial z} = a \cdot (1 - a)$$

$$\frac{\partial \ell}{\partial a} = \frac{\partial [-\{ y * \log(a) + (1-y) * \log(1-a) \}]}{\partial a}$$
$$\frac{\partial \ell}{\partial a} = -\{ y/a + (1-y) / (1-a) * (-1) \}$$
$$\frac{\partial \ell}{\partial a} = -y/a + (1-y) / (1-a)$$

$z = X * W + b$

$\hat{y} = a = \sigma(z)$

$\sigma(z) = \frac{1}{1+e^{-z}}$

$\ell(a, y) = -[y * \log(a) + (1-y) * \log(1-a)]$

For binary classification:

$\ell(a, y) = -y * \log(a)$

$\Rightarrow$

$$\frac{\partial \ell}{\partial w1} = x_1 \cdot \frac{\partial \ell}{\partial z} = x_1 \cdot (a-y)$$
$$\frac{\partial \ell}{\partial w2} = x_2 \cdot \frac{\partial \ell}{\partial z} = x_2 \cdot (a-y)$$
$$\frac{\partial \ell}{\partial b} = \frac{\partial \ell}{\partial z} = (a-y)$$

$\Rightarrow$

$$w_1 = w_1 - \alpha * \frac{\partial \ell}{\partial w_1} = w_1 - \alpha * x_1 * (a-y)$$
$$w_2 = w_2 - \alpha * \frac{\partial \ell}{\partial w_2} = w_2 - \alpha * x_2 * (a-y)$$
$$b = b - \alpha * \frac{\partial \ell}{\partial b} = b - \alpha * (a-y)$$

Where α is learning rate. The cost function is

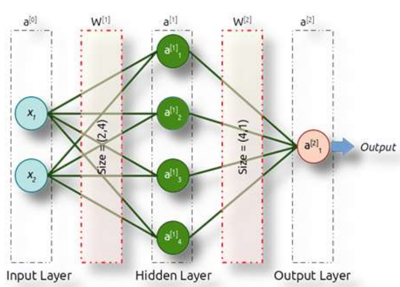
$$J(W, b) = \frac{1}{m} * (\Sigma\, \ell(a, y))$$

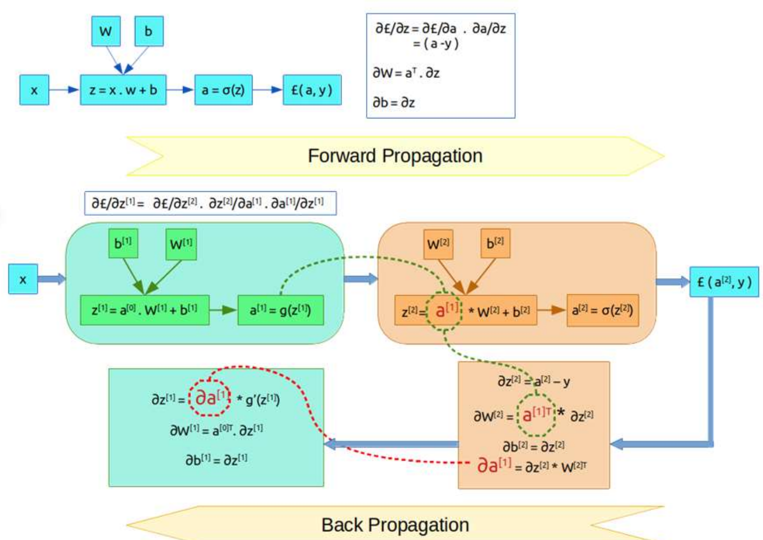
Hence $\frac{\partial J}{\partial w_1} = \frac{1}{m} * (\Sigma \frac{\partial \ell(a, y)}{\partial w_1})$, etc...

pra-sâmi

4

Neural Network





pra-sâmi

5

Activation Function

pra-sâmi

6

Overview

- The choice of activation functions in Deep Neural Networks has a significant impact on the training dynamics a formance.
- Till very t successful and widely-used activation function was t ed inear Unit (ReLU)
- Alth arious alternatives to ReLU been proposed, none have managed to to inco gains

pra-sâmi

7

Activation Functions

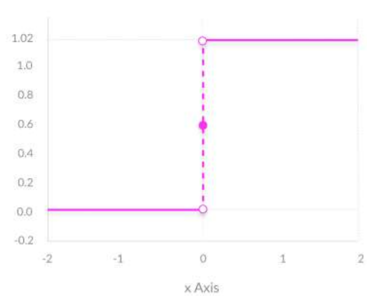
- Activation functions is a function attached to each neuron in the network
  - It determines whether it should be activated ("fired") or not, based on whether each neuron's input is relevant for the model's prediction
- Activation functions also help normalize the output of each neuron to a range:
  - Between 1 and 0 or
  - Between -1 and 1
  - Or other desired ranges
- Need to be computationally lightweight
  - It is calculated for each neuron for every data instance (row)
- It's a mathematical gate that turns a neuron on or off

pra-sâmi

8

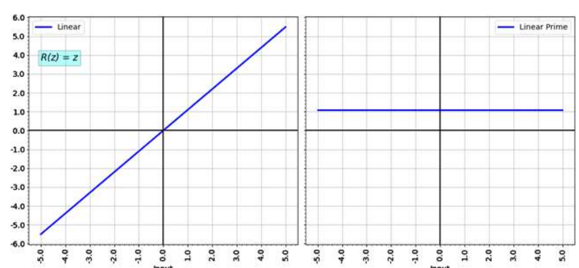
Activation Functions

- Binary Step function



- We already seen this in previous session!!!!

- Linear Activation Function



- That will be simple linear regression!
  - There are some use cases…

pra-sâmi

9

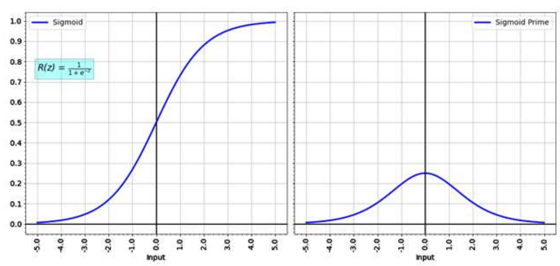
Non-Linear Activation Functions

- There are many popular activation functions
  - Sigmoid / Logistic
  - Softmax
  - Tanh (Hyperbolic Tangent)
  - ReLU (Rectified Linear Unit)
  - Leaky ReLU
  - Parametric ReLU
  - Swish
  - Lisht
  - Mish
- Stay tuned… it’s an active research area…

pra-sâmi

10

Sigmoid



- Takes a real value as input and outputs another value between 0 and 1 i.e. [0,1]
- It’s easy to work with; Most suitable as activation functions
- Non-linear, continuously differentiable, monotonic, and has a fixed output range
- Good for binary classification tasks

pra-sâmi

14

Sigmoid – drawbacks

- Towards either end, becomes sluggish
  - Problem of “vanishing gradients”
  - The network refuses to learn further or is drastically slow
  - Another reason why we need to scale values
- Its output isn’t zero centered. It makes the gradient updates go too far in different directions.
  - 0 < output < 1, and it makes optimization harder
- Sigmoid saturates and kills gradients

11/24/2023 pra-sâmi

15

Softmax Function

- In physics and statistical mechanics, it is known as the **Boltzmann** distribution or the **Gibbs** distribution.
- Formulated by the Austrian physicist and philosopher **Ludwig Boltzmann** in **1868**.
- In 1959, **Robert Duncan Luce** proposed the use of the Softmax function for reinforcement learning in his book “Individual Choice Behavior: A Theoretical Analysis”.
- Take vector of N values and convert into vector of N values with sum = 1
- Input values are natural numbers (Positive, Negative).
- Output is always numbers between 0 and 1 i.e. $A = \{ a \mid real(a) \wedge 0 \leq a \leq 1 \}$

11/24/2023 pra-sâmi

16

Softmax Function

- Softmax is multi-class logistic regression,
  - Takes vector of N values and converts into vector of N values with sum = 1
  - Input values are natural numbers (Positive, Negative).
  - Output is always numbers between 0 and 1 i.e. A = { a | real(a) ∧ 0 ≤ a ≤ 1 }
  - It is differentiable everywhere.

$$S(\vec{Z}) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_i}}$$

- Its helps in representing values as probabilities
  - Smaller the value, smaller the probability and vice versa

Like Sigmoid Activation function, Vanishing Gradient is still a problem!

- Its formula is very similar to Sigmoid function,
  - Sigmoid function is one special case of Softmax
- Softmax is very useful because it converts the scores to a normalized probability distribution
  - Invariably, multi-layer neural networks end in a penultimate layer which outputs real-valued scores,
  - It is non-linear in nature. So, it introduces non-linearity in the network enabling it to learn better.

11/24/2023

pra-sâmi

17

Softmax vs. Sigmoid

- Softmax

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- Sigmoid

$$S(\vec{Z}) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_i}}$$

- For single class value will be [0, x], Softmax

$$S(\vec{Z}) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_i}}$$

$$S(\vec{Z}) = \frac{e^{z_1}}{e^{z_1} + e^{z_2}}$$

$$S(Z) = \frac{e^{z}}{e^{0} + e^{z}}$$

$$S(Z) = \frac{e^{z}}{1 + e^{z}}$$

$$S(Z) = \frac{1}{1 + e^{-z}}$$

11/24/2023

pra-sâmi

18

Softmax vs. Argmax

- Both work the same way, Softmax is expected to be a differentiable alternative to argmax
- Argmax returns index of highest value and no idea about other values.
- It is common to train using the Softmax

3 classes

$Out_1 = [0.01, 0.97, 0.02]$

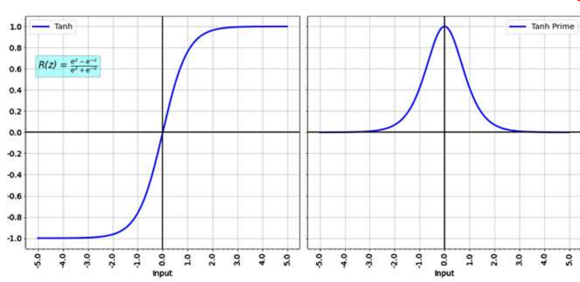
$Out_2 = [0.33, 0.34, 0.33]$

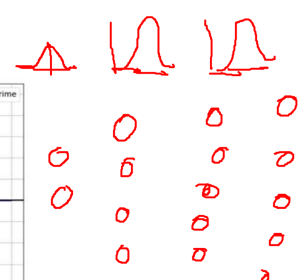
11/24/2023

pra-sâmi

20

Tanh





- Mathematically shifted version of the sigmoid function with
- Non-linear, but zero-centered
  - Very useful in hidden layers
  - Helps in centering the data around zero (bring mean closer to zero). Learning next layer becomes easier.
- The gradient is stronger than sigmoid
  - Derivatives are steeper
- Other problems are similar to sigmoid

11/24/2023

pra-sâmi

21

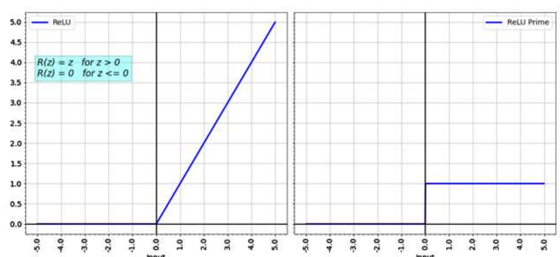
Tanh

- Advantage:
  - The negative inputs will be mapped negative and the zero inputs will be mapped near zero
  - The function is differentiable.
  - The function is monotonic while its derivative is not monotonic.
  - Faster convergence for two reason:
    - Steeper than Sigmoid function
    - Zero centric output
- Disadvantage:
  - Vanishing gradient have not gone away yet!
- Different research papers different views as to why it is better or even it is not always better!
- And the debate will continue…
- Early stages of design, Tanh in intermediate layer is a good starting point

pra-sâmi

22

Rectified Linear Units (ReLU)



- Non-linear function (almost)
- Better performance than Sigmoid or Tan in almost all models
- It avoids and rectifies vanishing gradient problem.
- ReLU is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations.
- Suitable for Hidden layers only.

pra-sâmi

23

Rectified Linear Units (ReLU)

- Some gradients can be fragile during training and can die.

- Could result in Dead Neurons.

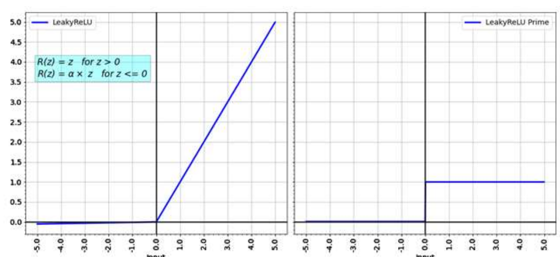
- For activations in the region (x<0) of ReLu , gradient will be zero
  - Weights will not get adjusted during descent
  - Neurons which go into that state will stop responding to variations in error/ input
  - Dying ReLu problem

- The range of ReLu is [0, ∞]
  - Can blow up the activation

pra-sâmi

24

Leaky ReLU



- Attempt to fix the “dying ReLU” problem by having a small negative slope (of 0.01, or so).

- LeakyRelu is a variant of ReLU ; allows a small, non-zero negative values
  - R(z_i) = $\begin{vmatrix} z_i & if\, z_i \geq 0 \\ a_i \, . z_i & if\ z_i < 0 \end{vmatrix}$
  - Work–under–progress : benefits across different architectures and domains still being investigated

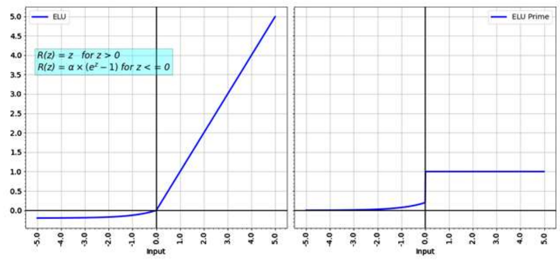
- As it possess linearity, it can’t be used for the complex Classification.

- Lags behind the Sigmoid and Tanh for some of the use cases.

pra-sâmi

25

Exponential Linear Unit (ELU)



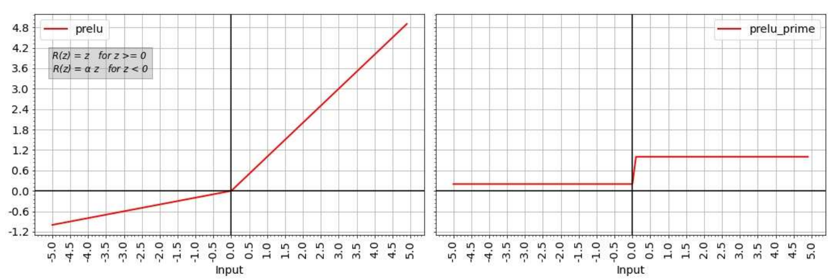
- Converges faster ; Has alpha constant which should be positive number
- ELU is a strong alternative to ReLU.
- Unlike to ReLU, ELU can produce negative outputs.
- For x > 0, it can blow up the activation with the output range of [0, ∞].

11/24/2023

pra-sâmi

26

Parameterized ReLU



- A Parametric Rectified Linear Unit, or PReLU, is an activation function that generalizes the traditional rectified unit with a slope for negative values.
- The intuition is that different layers may require different types of nonlinearity.

11/24/2023

pra-sâmi

27

Parameterized ReLU

$$F(z_i) = \begin{vmatrix} z_i & if\, z_i \geq 0 \\ a_i \, . \, z_i & if\ z_i < \ 0 \end{vmatrix}$$

- Pick your own parameter
- In experiments with convolutional neural networks, PReLus for the initial layer have more positive slopes, i.e. closer to linear.
  - Since the filters of the upper layers are edge or texture detectors,
  - This shows a circumstance where positive and negative responses of filters are respected.
- In contrast, deeper layers have smaller coefficients
  - Model becomes more discriminative at later layers
  - While it wants to retain more information at earlier layers.

pra-sâmi

28

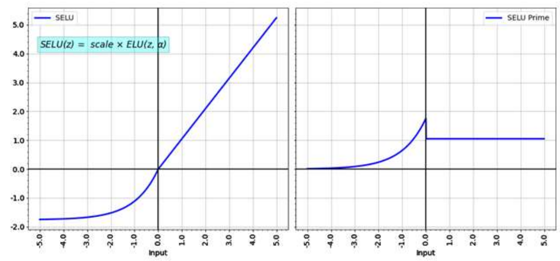
Challenges with ReLU

- The consistent problem is that its derivative is 0 for half of the values of the input x in the Function, i.e. f(x)=max(0,x)
- As parameter update algorithm, could used Stochastic Gradient Descent and other optimizers
  - If the parameter itself is 0, then that parameter will never be updated as it just assigns the parameter back to itself
  - Leading close to 40% Dead Neurons in the Neural network environment where z is negative
  - Various substitutes like Leaky ReLU Prameterized ReLU have unsuccessfully tried to devoid it of this issue.

pra-sâmi

29

Scaled ELU (SELU)



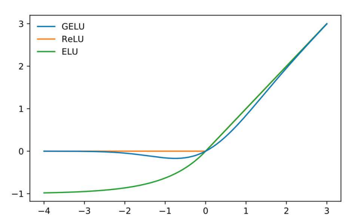
- Activation was introduced in a 2017 paper by Klambauer et al
- Properly initialization, the networks will self-normalize
  - Each layer's output will roughly be zero-centered with standard deviation equal to one
- Helps prevent the vanishing or exploding gradients problems

11/24/2023

pra-sâmi

30

Gaussian Error Linear Unit (GELU)



The GELU ($\mu = 0, \sigma = 1$), ReLU, and ELU($\alpha = 1$)

- Contrary to the ReLU, GELU weights its inputs by their value instead of thresholding them by their sign
- Defines as The GELU activation function is x∗ Φ(x),
  - where Φ(x) : the standard Gaussian cumulative distribution function refer scipy's norm.cdf(x)

$$\mathrm{GELU}(x) = xP(X \leq x) = x\Phi(x)$$

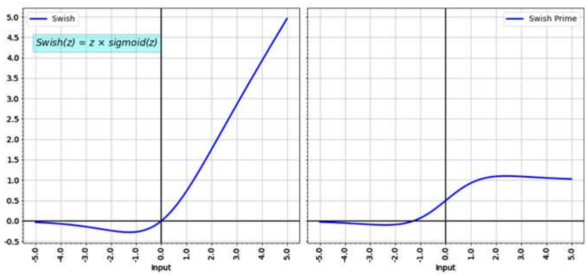
  - ≈ $0.5x(1 + \tanh[\sqrt{2/\pi}(x + 0.044715x^3)])$
  - or $x\sigma(1.702x)$,

11/24/2023

pra-sâmi

31

Swish



- Google Brain Team proposed a new activation function:
  - f(x) = x · sigmoid(x)
- Experiments show that Swish tends to work better than ReLU on deeper models across a number of challenging data sets
  - Simply replacing ReLUs with Swish units improves top-1 classification accuracy on ImageNet by 0.9% for Mobile NASNetA and 0.6% for Inception-ResNet-v2
- The simplicity of Swish and its similarity to ReLU make it easy for practitioners to replace ReLUs with Swish units in any neural network.
- Swish is a smooth, non-monotonic function that consistently matches or outperforms ReLU on deep networks

pra-sâmi

32

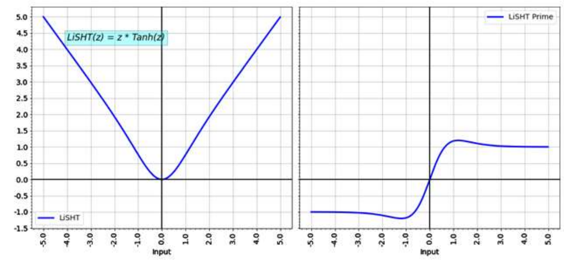
Swish

- Unbounded above and bounded below
  - Non-monotonic attribute that actually creates the difference
- We can train deeper Swish networks than ReLU networks when using BatchNorm (Ioffe & Szegedy, 2015) despite having gradient squishing property
- With MNIST data set, when Swish and ReLU are compared, both activation functions achieve similar performances up to 40 layers.
- Swish outperforms ReLU by a large margin in the range between 40 and 50 layers
  - For less than 40 layers, performance is comparable
- In very deep networks, Swish achieves higher test accuracy than ReLU.
- Swish outperforms ReLU on every batch size, suggesting that the performance difference between the two activation functions remains even when varying the batch size.
- Gradient descent problem was still there may be to a lesser degree!

pra-sâmi

33

LiSHT Activation Function

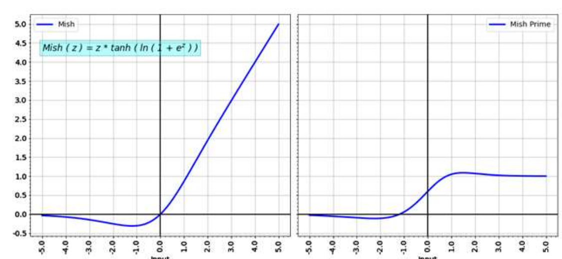


- The function scale the non-linear Hyperbolic Tangent (Tanh) function by a linear function
    - Help tackle the dying gradient problem

- According to paper it has outperformed Swish on a number of problems

34

Mish



f(z) = z * tanh (softplus (z))
= z * tanh (ln (1 + e^z))

- Inspired by Swish and has been shown to outperform it in a variety of computer vision tasks

- Mish was “found by systematic analysis and experimentation over the characteristics that made Swish so effective”.

- Mish seems to be the best activation in stock,
    - But jury is still out

35

Next Session…

- Loss / Cost Optimization
- Stochastic Gradient Descent (SGD) and others
- Momentum Learning Rates
- Adaptive Learning Rates

11/24/2023

pra-sâmi

36

11/24/2023

pra-sâmi