Department of Computing

Imperial College of Science, Technology and Medicine

Activation Functions

Lecture regarding Activation Layers and different types of normalization

Author: Anton Zhitomirsky

Contents

1 Activation Functions **3**
- 1.1 Linear . . . 3
- 1.2 Non-linear Activation Functions . . . 4
  - 1.2.1 Sigmoid . . . 4
  - 1.2.2 Tanh . . . 5
  - 1.2.3 ReLU . . . 6
  - 1.2.4 Leaky ReLU . . . 6
  - 1.2.5 PReLU . . . 7
  - 1.2.6 SoftPlus . . . 8
  - 1.2.7 Exponential Linear Unit . . . 8
  - 1.2.8 Continuously Differentiable Exponential Linear Unit . . . 9
  - 1.2.9 Scaled Exponential Linear Unit . . . 10
  - 1.2.10 Gaussian Error Linear Unit . . . 11
  - 1.2.11 ReLU6 . . . 11
  - 1.2.12 LogSigmoid . . . 12
  - 1.2.13 Softmin . . . 12
  - 1.2.14 Softmax . . . 13
  - 1.2.15 LogSoftmax . . . 13
- 1.3 Period activations . . . 14
  - 1.3.1 Sinusoidal Representation Network . . . 14
- 1.4 Summary . . . 15

2 Loss **15**
- 2.1 L2 Norm . . . 15
- 2.2 L1 Norm . . . 15
- 2.3 Smooth L1 . . . 16
- 2.4 Negative log likelihood loss . . . 16
- 2.5 Cross Entropy (CE) Loss . . . 16
- 2.6 Binary Cross Entropy (BCE) Loss . . . 17

2.7 Kullback-Leibler Divergence Loss 17
2.8 Margin Ranking Loss/Ranking Losses/Contrastive Loss 18
2.9 Triplet Margin Loss 18
2.10 Cosine Embedding Loss 19
2.11 Summary 19

1 Activation Functions

They determine how neurons in the network respond, or "activate", when they receive a set of inputs. Activation functions introduce non-linear properties into the system, allowing the network to learn from complex data.

Activation functions are mathematical formulas that dictate the output of a neuron given a certain input. In essence, they act as the "gatekeepers" of each node, deciding how much signal should pass through to the next layer. A key point to remember is that activation functions introduce non-linearity into the network. This non-linearity is crucial because it allows the neural network to learn from complex and varied data. Without non-linear activation functions, your neural network would essentially become a simple linear regression model, incapable of learning complex functions.

We aim to move beyond binary "activated" or "not activated" outputs, and instead seek a more nuanced, continuous range of outputs.

$$Input = \sum (w_i \cdot input) + b_i$$

A neuron applies weights and a bias to inputs it receives. This can vary between negative infinity and infinity.

We cannot simply truncate values because this isn't differentiable

Bounded Value Clipping

no theoretical mathematical benefit (except that unbound things don't map well to probabilities – In some cases, particularly in high-stakes scenarios, you might want your model not just to be accurate but also well-calibrated. This means the model's output probabilities should reflect true probabilities. Bounding the outputs can help in this calibration process.)

If the outputs of neurons are unbounded or extremely varied, it becomes difficult to interpret these outputs in a meaningful way, especially when combining outputs from many neurons.

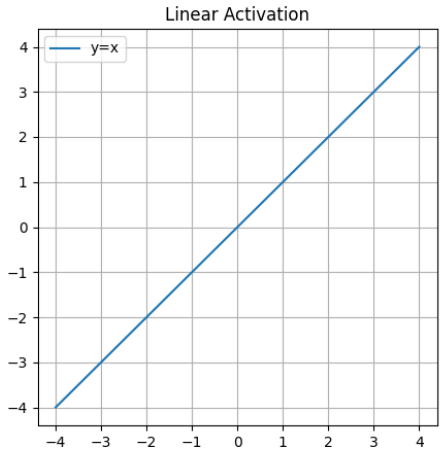
Unbounded outputs can lead to numerical instability in training. Extremely high values can cause issues with gradients and can lead to problems like exploding gradients.

A model that reacts too strongly to edge cases might overfit to those specific instances and fail to generalize well to other data. Limiting the output range can help in making the model more robust to unseen data.

1.1 Linear

Definition 1.1 (Linear Activation)**.**

$$f(x) = c \cdot x$$



- This produces a constant gradient, meaning that during backpropagation, the updates applied to weights are constant and independent of the change in input, denoted by Δx.
- If each layer in a multi-layered network employs a linear activation function, the output of one layer becomes the input to the next, perpetuating linearity throughout the network; no matter how many layers you have, the entire network behaves like a single-layer linear model. This means you could replace all N linear layers with just a single linear layer and achieve the same output. It renders the "depth" of the network irrelevant.

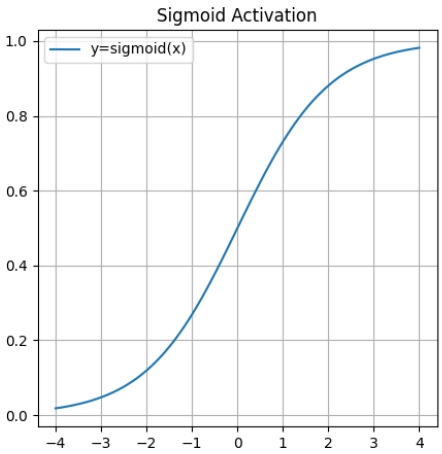
Therefore, while linear activation functions may have some use-cases, they aren't typically chosen for complex machine learning tasks that require the network to capture more complex, non-linear relationships in the data.

1.2 Non-linear Activation Functions

1.2.1 Sigmoid

Definition 1.2 (Sigmoid Activation)**.**

$$f(x) = \frac{1}{1 + e^{-x}}$$



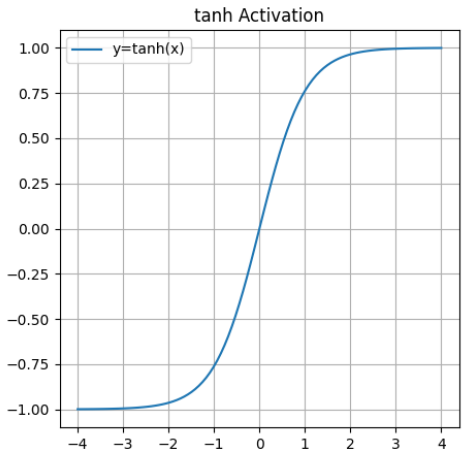
- This function is non-linear, allowing us to stack layers in a neural network, thereby facilitating the learning of more complex representations.

- gives a more analog or continuous output w.r.t binary step function.
- Smooth gradient which is crucial for gradient descent algorithms. One notable characteristic is that between the X values of -2 and 2, the curve is especially steep. This implies that small changes in the input within this region result in significant shifts in output, facilitating rapid learning during the training phase.
- Towards the tails of the function, the curve flattens out, and the output values become less sensitive to changes in input. This results in a vanishing gradient problem, where gradients become too small for the network to learn effectively, leading to slow or stalled training.
- Good for classifiers

1.2.2 Tanh

Definition 1.3 (Tanh Activation)**.**

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



- Scaled version of sigmoid but from -1 to 1 instead of 0 to 1
- non-linear, we can stack it
- since $(-1, 1)$ less concern about activations becoming too large and dominating the learning process
- One key benefit of tanh over sigmoid is that its gradient is stronger; that is, the derivatives are steeper. This can make it a better choice for certain problems where faster convergence is desired.
- its outputs are zero-centered, meaning the average output is close to zero. This is beneficial for the learning process of subsequent layers, as it tends to speed up convergence by allowing for a balanced distribution of outputs and gradients.

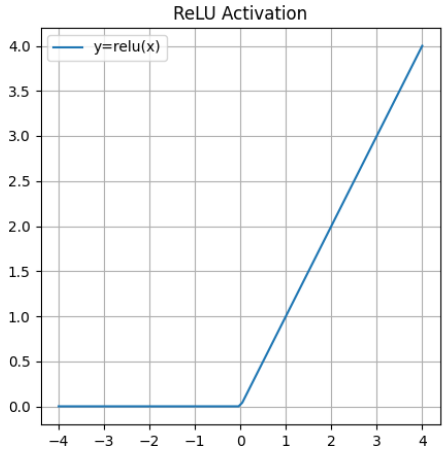
However, like the sigmoid function, tanh also suffers from the vanishing gradient problem when you stack many layers, which can slow down learning.

Careful normalization of the inputs is also essential when using tanh to ensure effective learning.

1.2.3 ReLU

Definition 1.4 (ReLU Activation)**.**

$$f(x) = max(0, x)$$

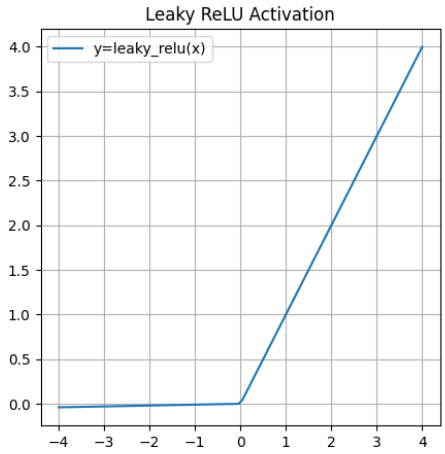


- piecewise linear that outputs the input directly if it is positive, otherwise, it outputs zero
- ReLU is inherently non-linear when considered as a whole, particularly due to the sharp corner at the origin.
- combinations of ReLU functions are also non-linear, enabling us to stack layers in neural networks effectively. Because it is a universal approximator.
- unbounded as $[0, \infty)$
- unboundedness can cause explosions of activations if not managed properly
- tends to produce sparse activations
- In a neural network with many neurons, using activation functions like sigmoid or tanh would cause almost all neurons to activate to some degree, leading to dense activations. ReLU, on the other hand, will often output zero, effectively ignoring some neurons, which can make the network more computationally efficient.
- Dying ReLU problem caused by often outputting zero (If a neuron's output is always zero (perhaps due to poor initialization), the gradient for that neuron will also be zero.) As a result, during backpropagation, the weights of that neuron remain unchanged, effectively "killing" the neuron. This can result in a portion of the neural network becoming inactive, thereby limiting its capacity to model complex functions.

1.2.4 Leaky ReLU

Definition 1.5 (Leaky ReLU Activation)**.**

$$f(x) = \begin{cases} x & for\ x \geq 0 \\ 0.01 \cdot x & for\ x < 0 \end{cases}$$



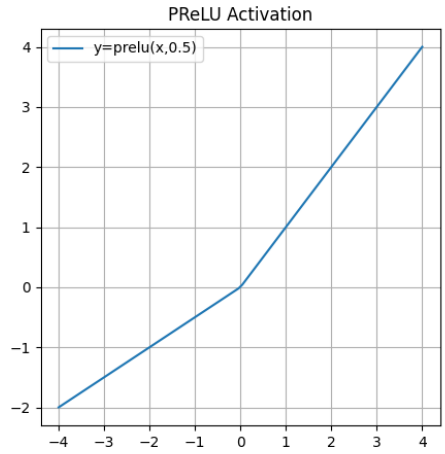
- addresses the "Dying ReLU problem"
- Leaky ReLU attempts to solve this by introducing a small slope for negative values, typically 0.01, to ensure the gradient is non-zero. This small slope allows "dead" neurons to reactivate during the course of training. In other words, it provides a pathway for gradients to flow, even when the neuron is not active.
- computationally efficient (simple math operations)
- maintains benefit of ReLU such as sparsity and ability to approximate a wide range of functions

1.2.5 PReLU

Definition 1.6 (Parametric ReLU Activation)**.**

$$f(x) = \begin{cases} x & for\ x \geq 0 \\ a \cdot x & for\ x < 0 \end{cases}$$

Where a is learnable.



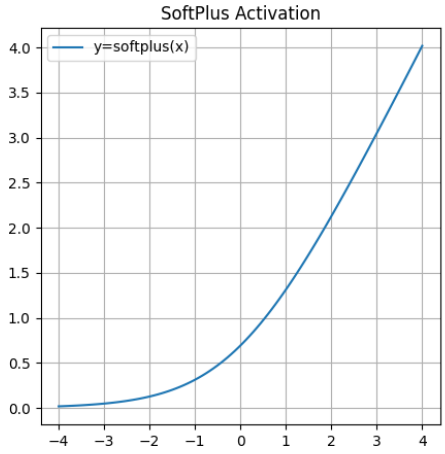
- negative slope becomes a learnable parameter

- flexibility allows to adapt during training, potentially leading to better performance than Leaky ReLU in some scenarios.
- scale invariant - if you multiply the input by a scalar, the shape of the output remains the same, just scaled. In the context of CNN architectures, where scale invariance can be valuable, PReLU and its variants can be especially useful

1.2.6 SoftPlus

Definition 1.7 (SoftPlus Activation)**.**

$$f(x) = \frac{1}{\beta} \cdot \log(1 + e^{\beta \cdot x})$$

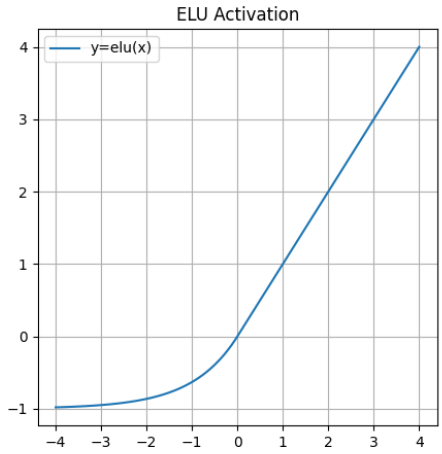


- smooth and (easier) differentiable approximation to ReLU
- parameterized by a scale factor β which controls how closely the function approximates the ReLU (highest meaning closest)
- outputs only positive values, suitable for layers where you specifically require positive activations
- numerical stability issues for large input values PyTorch implementation switches to a linear function when the condition $\beta \times x$ exceeds a predefined threshold
- non-linear across its entire domain
- SoftPlus is sensitive to the amplitude of the input signal, which means that it's non-linear regardless of the input size. That's beneficial for models where amplitude variation is a significant feature.

1.2.7 Exponential Linear Unit

Definition 1.8 (ELU Activation)**.**

$$f(x) = \max(0, x) + \min(0, \alpha \cdot (e^x - 1))$$

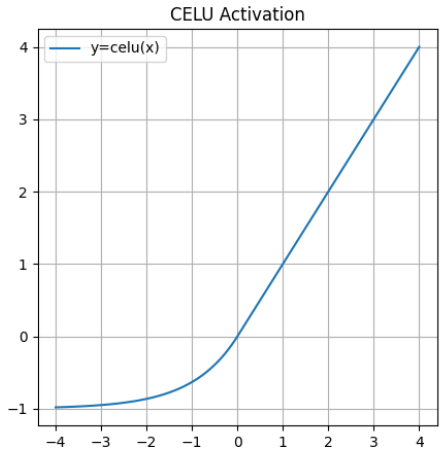


- extension to the ReLU, designed to be element-wise, operating on each element of the input independently.
- One thing that sets ELU apart is its ability to output negative values. Unlike ReLU, which only outputs positive values, ELU can go below zero.
- Being able to output negative values allows ELU to push the mean activation closer to zero. A zero-centered mean can help the network converge faster, a useful property in deep learning models.
- soft and smooth version of ReLU while still being positive and negative
- ELU is a strong candidate for scenarios where you want a balance of smoothness, differentiability, and the ability to have a mean activation around zero
- α can change everything

1.2.8 Continuously Differentiable Exponential Linear Unit

Definition 1.9 (CELU Activation)**.**

$$\max(0, x) + \min(0, \alpha \cdot (e^{\frac{x}{\alpha}} - 1))$$

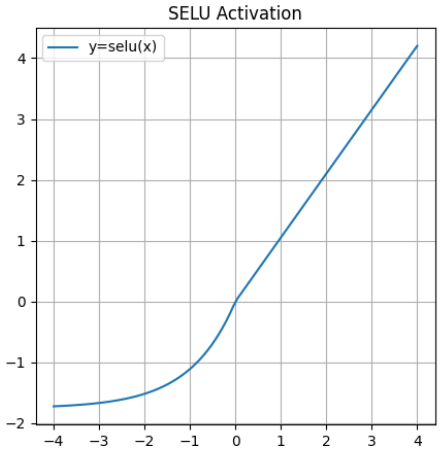


- also an element-wise function
- computationally efficient to apply across large tensors
- special emphasis on continuous differentiability, denoted as C1 continuity (desirable in optimization tasks as it ensures a smooth gradient, facilitating more efficient backpropagation)
- α doesn't just scale the exponential function for negative values, it also scales the input x inside the exponential term
- When $\alpha \neq 1$ CELU is continuously differentiable across its entire domain
- can produce negative values, which centers the mean activation towards zero, which could speed up the convergence of deep learning modules.

1.2.9 Scaled Exponential Linear Unit

Definition 1.10 (SELU Ativation)**.**

$$scale \cdot (\max(0, x) + \min(0, \alpha \cdot (e^x - 1)))$$



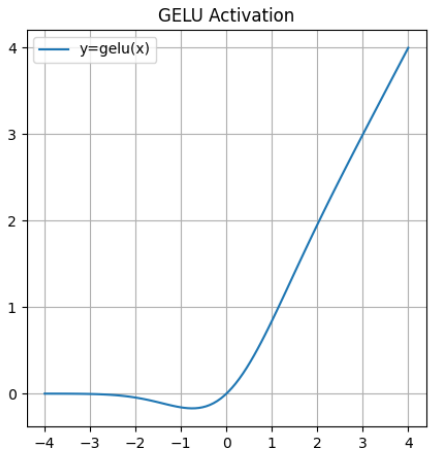
- pre-defined α and $scale$ parameters which are solutions to fixed-point equation to maintain a mean of 0 and variance of 1 across layers.
- performs internal normalization - mean and variance of the activations are preserved from one layer to the next
- To achieve this normalization, the function needs to produce both positive and negative outputs, which allows it to shift the mean towards zero.
- gradient close to 0 which caused "gradient vanishing" in other functions is beneficial in SELU for internal normalization.
- ELUs never die. Due to its design, SELU avoids the "dying unit" problem - robust!
- operates well without other normalization techniques; it's specifically designed to keep the internal statistics of your network stable, which can lead to faster and more reliable training.

1.2.10 Gaussian Error Linear Unit

Definition 1.11 (GELU Ativation)**.**

$$x \cdot \phi(x)$$

Where ϕ is the cumulative distribution function for gaussian distribution.

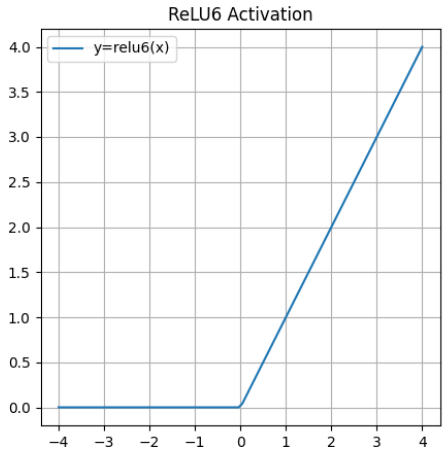


- introduces probabilistic flavor to the activation process
- implicates regularization of neural networks potentially providing a beneficial influence on network training and performance.

1.2.11 ReLU6

Definition 1.12 (ReLU6 Ativation)**.**

$$\min(\max(0, x), 6)$$

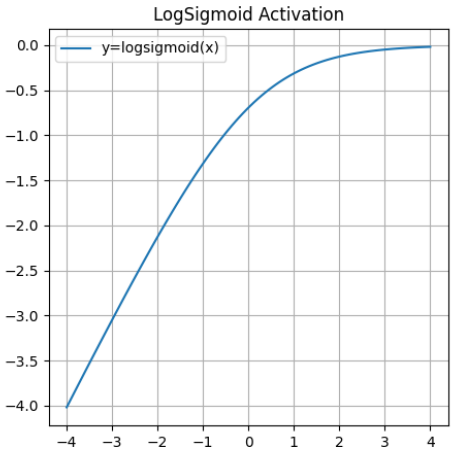


- can saturize activations - activations are limited to a certain range, which can be useful in preventing activations from growing excessively and causing numerical instability.
- 6 is a parameter than can be adjusted to achieve different levels of saturation
- not necessarily same behavior as the hard sigmoid or tanh

1.2.12 LogSigmoid

Definition 1.13 (LogSigmoid Ativation)**.**

$$\log(\frac{1}{1+e^{-x}})$$

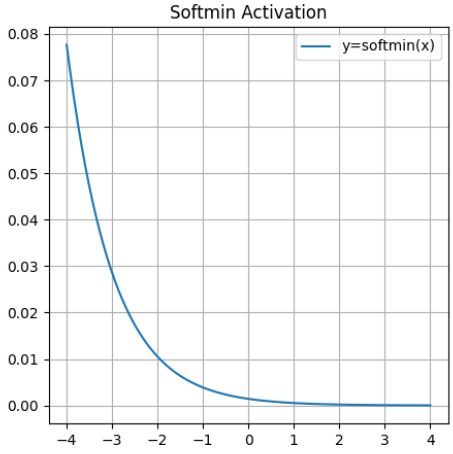


- element-wise, it is applied separately to each element of the input tensor
- used in the context of cost functions rather than as a primary activation function in neural network layers
- While LogSigmoid may not be a common choice for standard activation functions in neural network layers, its presence is significant in the realm of loss functions, where it contributes to the optimization process during training.

1.2.13 Softmin

Definition 1.14 (Softmin Ativation)**.**

$$\frac{e^{-x_i}}{\sum_j e^{-x_j}}$$

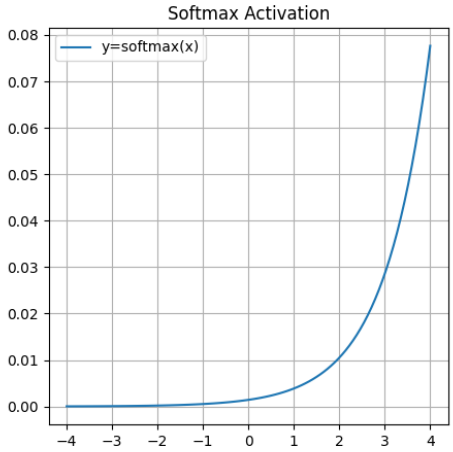


- applies softmin to an n-dimensional input tensor rescaling them so that the elements of the n-dimensional output tensor lie in the range $[0, 1]$ and sum to 1
- Softmin introduces multi-dimensional non-linearities to the neural network
- representation may resemble a probability distribution (valuable in scenarios where you want to assign relative weights or preferences among multiple alternatives)
- Softmin's role lies in rescaling and transforming input tensors into probability-like distributions, contributing to the multi-dimensional non-linearities of neural networks and enabling the modelling of various factors in the data

1.2.14 Softmax

Definition 1.15 (Softmax Ativation)**.**

$$\frac{e^{x_i}}{\sum_j e^{x_j}}$$

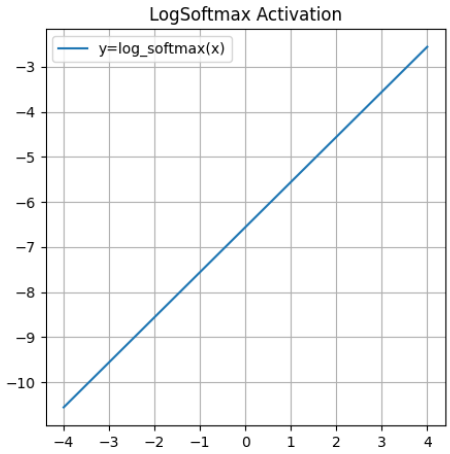


- applies softmax to an n-dimensional input tensor rescaling them so that the elements of the n-dimensional output tensor lie in the range $[0, 1]$ and sum to 1
- when dealing with image classification, the Softmax activation function is commonly employed to transform the network's logits into probabilities.

1.2.15 LogSoftmax

Definition 1.16 (LogSoftmax Ativation)**.**

$$\log(\frac{e^{x_i}}{\sum_j e^{x_j}})$$



- applies $\log(softmax)$ to an n-dimensional input tensor
- This logarithmic transformation can offer benefits in certain contexts, especially when dealing with probabilities and handling numerical stability
- logarithmic transformation provides a way to manipulate the Softmax probabilities to better align with the structure of certain loss functions
- possible to simplify mathematical calculations and potentially improve the training process
- LogSoftmax isn't typically used as an activation function in the output layers of neural networks, as its output values are not directly interpretable as class probabilities. Instead, it often plays a role in loss functions, helping to define the optimization process - logarithmic properties can be advantageous for numerical stability and simplifying mathematical operations.

1.3 Period activations

1.3.1 Sinusoidal Representation Network

Definition 1.17 (SIREN Ativation)**.**

$$\Phi(\boldsymbol{x}) = \boldsymbol{W}_n(\phi_{n-1} \circ \phi_{n-2} \circ \ldots \circ \phi_0)(\boldsymbol{x}) + \boldsymbol{b}_n, \quad \boldsymbol{x}_i \mapsto \phi_i(\boldsymbol{x}_i) = \sin(\boldsymbol{W}_i\boldsymbol{x}_i + \boldsymbol{b}_i)$$

- deals with implicit representations involving finding a continuous function that represents sparse input data, such as images.
- Unique architecture by leveraging sinusoidal activation functions
- well-suited for representing a wide range of signals, including images, wavefields, videos, and sounds, along with their derivatives
- an address challenging boundary value problems, such as Eikonal equations, the Poisson equation, and the Helmholtz and wave equations.
- require careful considerations due to their distinct convergence properties and architecture.

1.4 Summary

The choice of an activation function depends on the characteristics of the function you're aiming to approximate. If you have insights into the nature of the function, you can strategically select an activation function that aligns with those characteristics. This can significantly speed up the training process by allowing the network to approximate the desired function more efficiently.

- Choice of function depends on nature of targeted problem
- Most often ReLU is fine, but if network doesn't converge, use leakyReLU or PReLU
- Tanh is ok for regression and continuous reconstruction problems
- representative power of your training set will usually outweigh the contribution of a smartly chosen activation function.

2 Loss

These functions measure how well your network is doing, quantifying the difference between the predicted outputs and the actual ground truth. Backpropagation uses this error measurement to update the model parameters, aiming to minimize this error. It quantifies the gap between the network's predictions and the ground truth. The lower the value of the loss function, the closer the system's predictions are to the desired outcome.

2.1 L2 Norm

Definition 2.1 (L2 Norm, mean squared error)**.**

$$\ell(x, y) = \boldsymbol{\mathcal{L}} = \{l_1, \ldots, l_N\}^T, \quad l_n = (x_n - y_n)^2$$

With further reduction to a single value can be either $mean(\mathcal{L})$ or $sum(\mathcal{L})$

- Measured discrepancy between the predicted output and the actual ground truth
- in a mini-batch, perform squared difference calculation for each sample within the mini-batch, the individual squared errors can then be combined into a list.

2.2 L1 Norm

Definition 2.2 (L1 Norm)**.**

$$\ell(x, y) = \boldsymbol{\mathcal{L}} = \{l_1, \ldots, l_N\}^T, \quad l_n = |x_n - y_n|$$

With further reduction to a single value can be either $mean(\mathcal{L})$ or $sum(\mathcal{L})$

- Use for robust regression (noisy data); In robust regression, you want to give significant weight to small errors, but not as much weight to large errors, making it less sensitive to outliers in your dataset.
- it's not differentiable at exactly zero due to its pointy corners. This lack of differentiability can pose a problem during backpropagation in training neural networks. To address this, you can use a smoothed version of the L1 loss, such as the SmoothL1Loss

2.3 Smooth L1

Definition 2.3 (Smooth L1)**.**

$$loss(x, y) = \frac{1}{n}\sum_i z_i$$

$$z_i = \begin{cases} 0.5(x_i - y_i)^2, & if|x_i - y_i| < 1 \\ |x_i - y_i| - 0.5, & otherwise \end{cases}$$

With further reduction to a single value can be either $mean(\mathcal{L})$ or $sum(\mathcal{L})$

- For errors close to zero, it behaves like the L2 loss, which means it's quadratic in that region. As the error magnitude increases, it transitions to behaving like the L1 loss, which is linear for larger errors. This dual behavior allows the loss to strike a balance between the two norms, making it more robust to outliers while still accounting for smaller errors.
- Robustness against outliers makes SmoothL1 suitable for datasets containing noisy or outlier-ridden samples; prevents undue influence on the training process from those data points that deviate significantly from the norm.
- introduces a scale factor, often set at 0.5 - choosing an appropriate scale factor can be challenging since it depends on the distribution of the errors in the dataset

2.4 Negative log likelihood loss

Definition 2.4 (Smooth L1)**.**

$$\ell(x, y) = \boldsymbol{\mathcal{L}} = \{l_1, \dots, l_N\}^T, \quad l_n = -w_{y_n} x_{n,y_n}, \quad w_c = weight[c] \cdot 1\{c \neq ignore_{index}\}$$

$$\ell(x, y) = \begin{cases} \sum_{n=1}^{N} \frac{1}{\sum_{n=1}^{N} w_{y_n}} l_n, & if\ reduction = mean \\ \sum_{n=1}^{N} l_n, & if\ reduction = sum \end{cases}$$

Assumption: network's output can be interpreted as log likelihoods, usually representing class probabilities.

- LL loss is calculated for each data sample with each term representing the negative logarithm of the network's output probability for the ground truth class
- The weights w_c are used to assign different importance to different classes. It's designed to maximize the likelihood of the correct class while minimizing the likelihoods of other classes
- not limited to likelihoods, key idea is to drive the network to favor the correct class while penalizing other classes
- flexible to assign different weights to classes, valuable when training data exhibits class imbalance which helps counterbalance this issue during training
- an alternative approach to handling class imbalance is to increase frequency of rae classes during training.

2.5 Cross Entropy (CE) Loss

Definition 2.5 (Cross Entropy)**.**

$$loss(x, class) = -\log(\frac{e^{x[class]}}{\sum_j e^{x[j]}}) = -x[class] + \log(\sum_j e^{x[j]})$$

- Combines LogSoftmax and NLLLoss
- Useful for classification problems with C classes
- Classes can be weighted
- In classification problems with C classes, the CE loss functions by taking the LogSoftmax of the input scores and then applying the NLL loss It's designed to make the output of the LogSoftmax as large as possible for the correct class while minimizing the scores for other classes. This is achieved by taking the negative logarithm of the softmax probability for the correct class.
- Conceptually, it takes the scores, passes them through a softmax function, takes the logarithm of those values, and then optimizes to maximize the correct class's output while minimizing those of other classes
- When you backpropagate through the LogSoftmax operation, it has the effect of making the scores of incorrect classes as small as possible, thereby focusing on improving the correct class's score. In simpler terms, it aims to minimize the negative score of the correct class and adds a log term to diminish the influence of scores from other classes.

2.6 Binary Cross Entropy (BCE) Loss

Definition 2.6 (Binary Cross Entropy)**.**

$$\ell(x, y) = \mathcal{L} = \{l_1, \dots, l_N\}^T,$$
$$l_n = -w_n[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)]$$

- x_n represents the predicted probability for the positive class
- y_n is the ground truth label (0 or 1) for that observation
- w_n is a weights associated with that sample

With further reduction to a single value can be either $mean(\mathcal{L})$ or $sum(\mathcal{L})$

- negative weighted sum of two entropy terms for each observation in the batch
- the input probabilities, x_n, must lie in the [0, 1] range to ensure the loss's validity
- Suitable for binary classification tasks where each observation belongs to one of two classes.

2.7 Kullback-Leibler Divergence Loss

Definition 2.7 (Kullback-Leibler Divergence Loss)**.**

$$\ell(x, y) = \mathcal{L} = \{l_1, \dots, l_N\}^T, \quad l_n = y_n \cdot (\log y_n - x_n)$$

- y_n the ground truth probability
- x_n predicted probability
- suitable when your target distribution is represented as a one-hot distribution, where a single category is assigned a value of 1 and others are 0
- lacks incorporation of a softmax or log-softmax operation, suffers with numerical stability issues (especially with small probabilities)
- strong in variational autoencoders (VAEs)

2.8 Margin Ranking Loss/Ranking Losses/Contrastive Loss

Definition 2.8 (Margin Ranking Loss/Ranking Losses/Contrastive Loss)**.**

$$loss(x, y) = \max(0, -y \cdot (x_1 - x_2) + margin)$$

- useful to push classes as far away as possible and fro metric learning
- take category that scores is closest or higher than correct one change until difference is at least in the margin
- It finds particular use in metric learning, a task where the objective is to learn a meaningful distance metric between data points
- To better understand its practical application, consider scenarios where you aim to identify if two inputs belong to the same class (similar) or different classes (dissimilar). Instead of predicting specific values or labels, you're concerned with how these inputs relate in terms of similarity.
- Start by extracting features from two inputs and obtaining embedded representations for them
- using a metric function, like Euclidean distance, measure the similarity between these representations
- The goal is to train the feature extractors to produce similar representations for similar inputs and distinct representations for dissimilar inputs

n essence, Margin Ranking Losses operate around the concept of pushing one input's score above another's by a defined margin. If the difference in scores satisfies this condition, the cost is zero. However, if the difference falls below the margin, the cost increases linearly. This loss is particularly handy in cases like classification, where you compare the scores of the correct answer (x_1) with the highest-scoring incorrect answer (x_2) in the mini-batch. Keep in mind, though, that Margin Ranking Losses aren't confined to classification tasks alone. They also shine in energy-based models, where they exert downward pressure on the correct answer's score and push the incorrect answer's score upward.

2.9 Triplet Margin Loss

Definition 2.9 (Triplet Margin Loss)**.**

$$\ell(x, y) = \mathcal{L} = \{l_1, \ldots, l_N\}^T,$$
$$l_n(x_a, x_p, x_n) = \max(0, m + |f(x_a) - f(x_p)| - |f(x_a) - f(x_n)|$$

- a: actual
- p: positive sample
- n: negative sample
- Make samples from same classes close and different classes far away
- all about making samples from the same classes come close to each other, while simultaneously pushing samples from different classes farther apart
- objective is to ensure that the distance between a "good pair" (similar samples) is smaller than the distance between a "bad pair" (dissimilar samples).

- The actual distances themselves don't need to be small, but rather the relative difference matters more. In essence, we aim to reduce the distance between the "good" pair while simultaneously increasing the distance between the "bad" pair by at least a certain margin, denoted as 'm'
- focuses on training models to generate meaningful representations that capture the relative similarities or differences between data point
- One common scenario is in Siamese networks, where two inputs pass through a shared neural network, and their resulting vectors are compared
- Consider a scenario where we feed two images of the same category into a CNN, yielding two vectors. In this case, we want the distance between these vectors to be minimized. Conversely, for two images of different categories, we want the distance between their vectors to be maximized.

2.10 Cosine Embedding Loss

Definition 2.10 (Cosine Embedding Loss)**.**

$$loss(x, y) = \begin{cases} a - cos(x_1, x_2), & if y = 1 \\ \max(0, cos(x_1, x_2) - margin), & if y = -1 \end{cases}$$

- Measure whether two inputs are similar or dissimilar (normalized Euclidiean distance)
- compares similarity or dissimilarity between two input vectors x_1 and x_2.
- $y = 1 or -1$ indicates similar or dissimilar respectively
- For pairs labeled as similar (y=1), the loss tries to minimize the angle between the vectors, effectively making their cosine similarity close to 1
- For pairs labeled as dissimilar (y=-1), the loss aims to ensure that the cosine similarity is smaller than a specified margin. The margin is a hyperparameter and is usually a small positive value
- Use cosine similarity over euclidean distance because it focuses on direction of vector instead of magnitude
- make similar vectors point in the same direction, while dissimilar vectors should be separated but not necessarily be antipodal
- This is why we often set the margin to a small positive value, maximizing the use of the "equatorial space" for dissimilar vectors

2.11 Summary

- The choice of loss depends on the desired output (e.g., classification vs. regression)
- Loss functions are a hot topic of research.
- It informs how the overall system behaves during training
- Don't get scared by the equations. If you look closely the underlying ideas are very simple