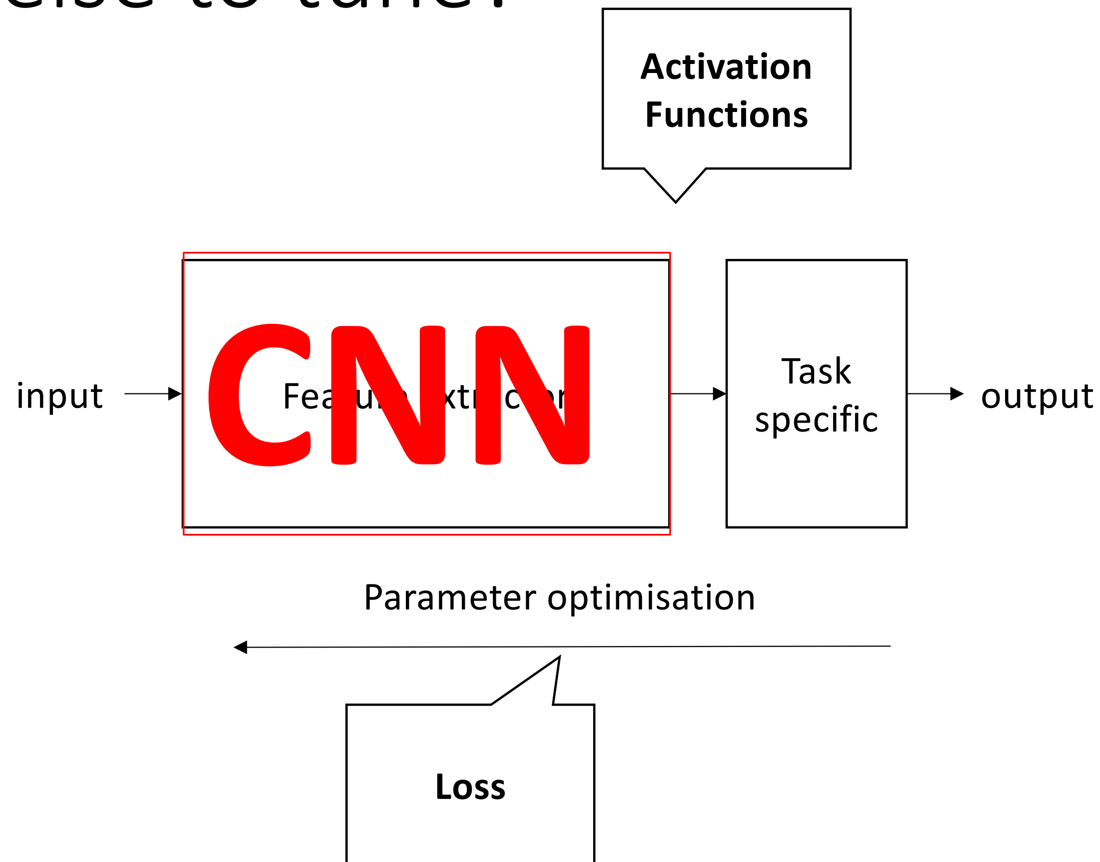
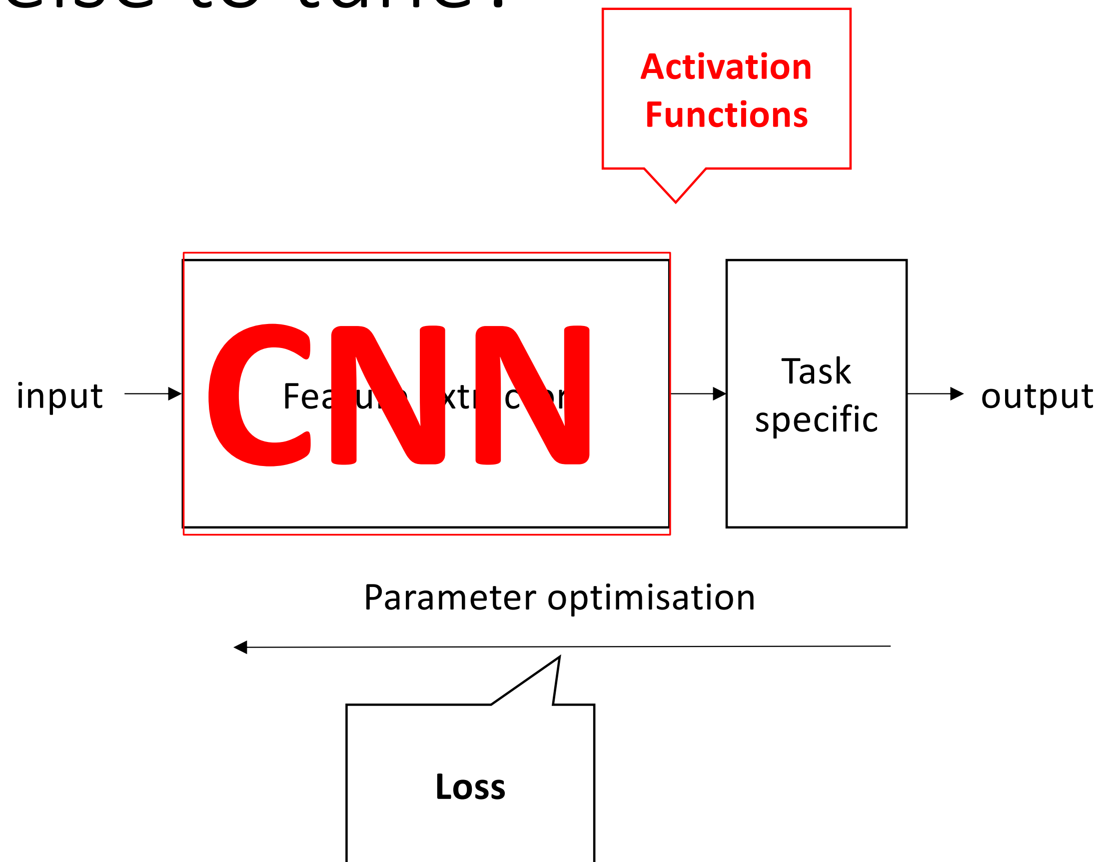
Deep Learning – Activation Functions

Bernhard Kainz

Where else to tune?



Where else to tune?

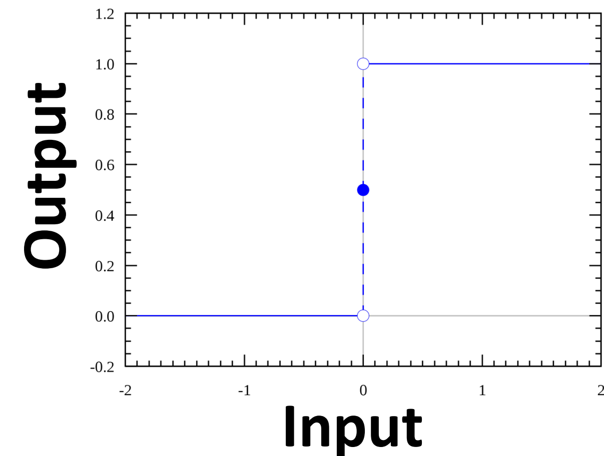


Activation functions

- Consider a neuron,

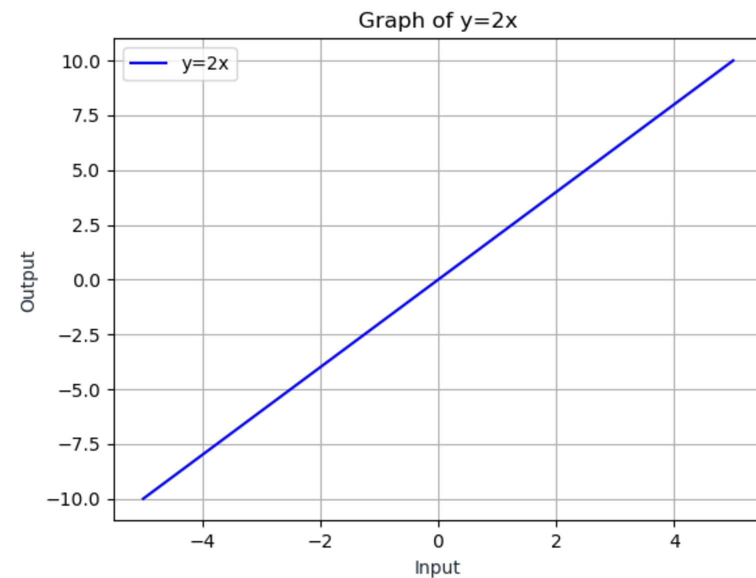
$$\text{Input} = \sum (w_i \cdot input) + b_i$$

- Naïve activation:
  - If the value of Y is above a certain value, declare it activated.
  - Not differentiable, no backprop



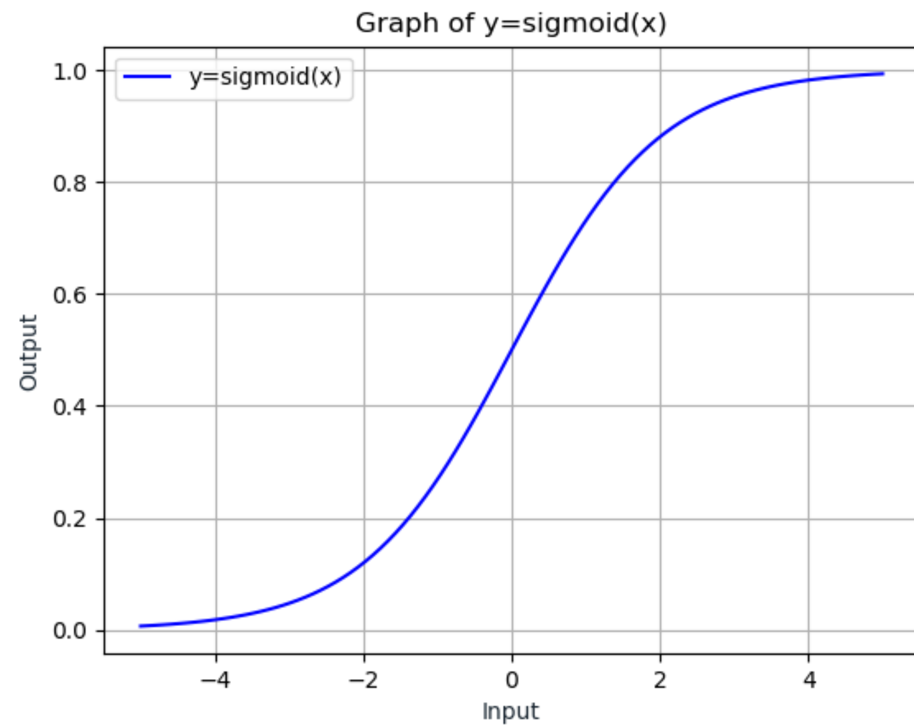
Linear activation

- $Output = c \cdot x$
- Constant gradient, no relationship to x during backprop



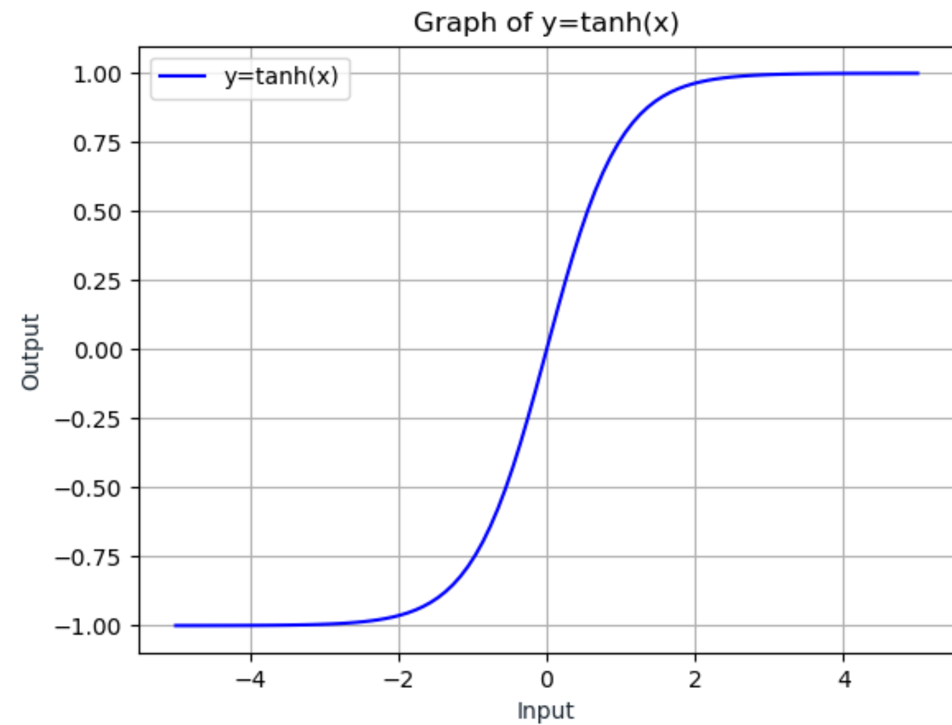
Sigmoid function

- $Output = \frac{1}{1+e^{-input}}$
- Nonlinear



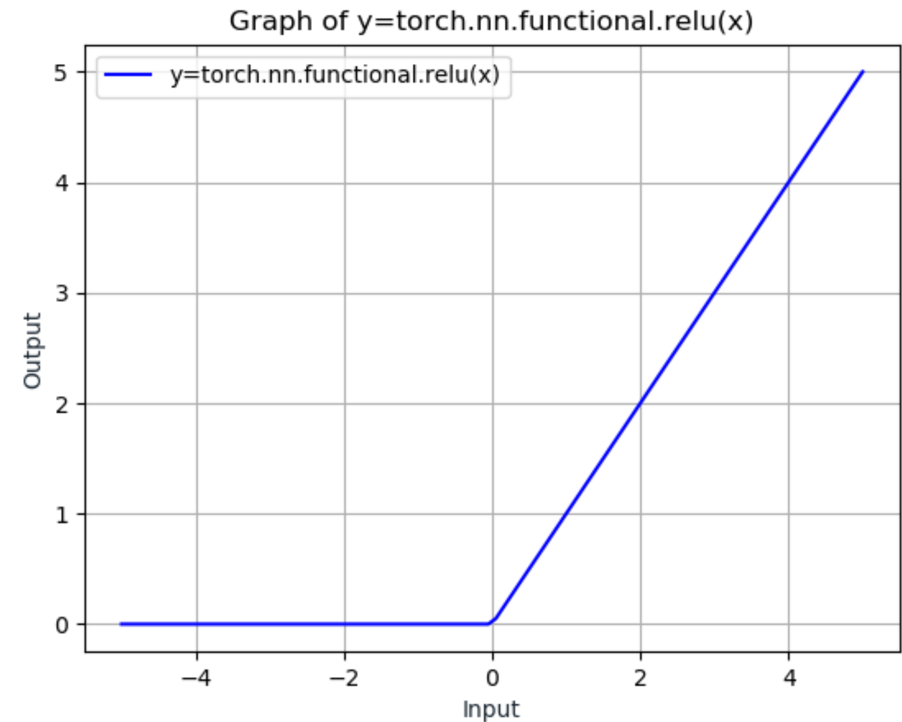
tanh function

- $Output = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Scaled sigmoid



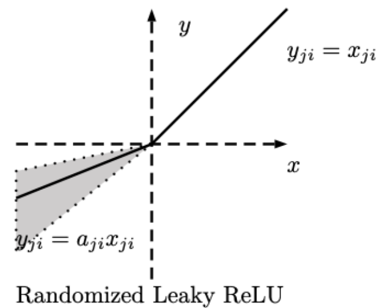
ReLU

- $Output = \max(0, x)$
- Efficient
- combinations of ReLU and ReLU are non linear
- Bound: [0, inf)
- dying ReLu problem

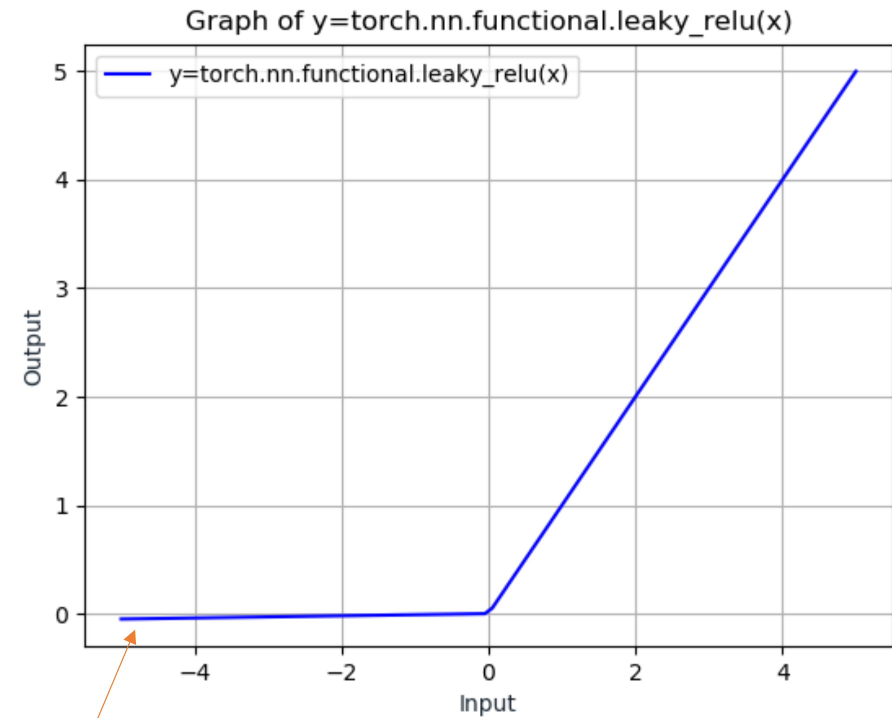


Leaky ReLU

- $Output = \begin{cases} x & for\ x \geq 0 \\ e.g.\ 0.01 \cdot x & for\ x < 0 \end{cases}$
- Mitigates dying ReLu

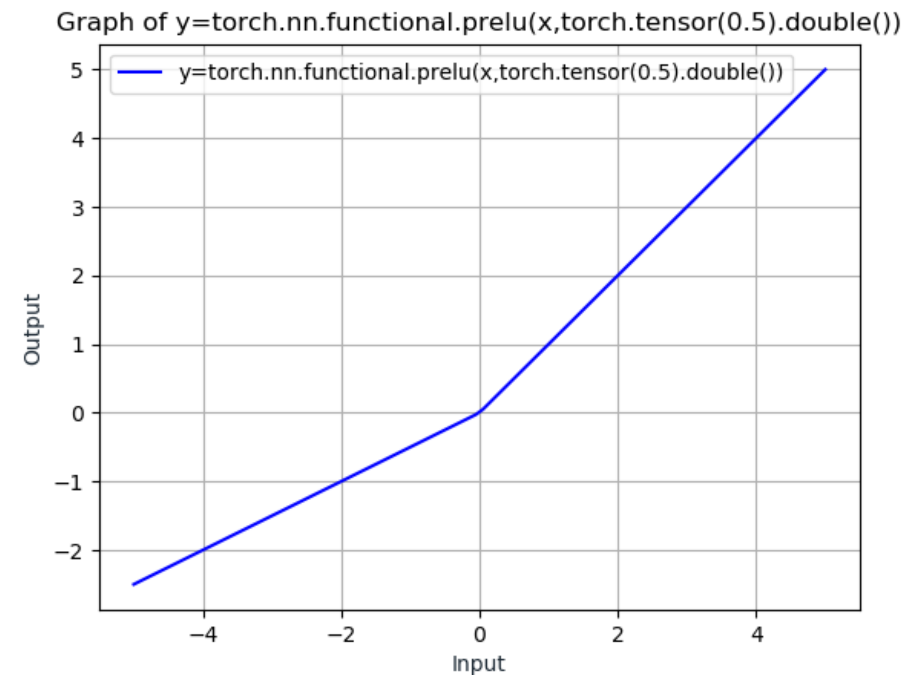


atcold.github.io



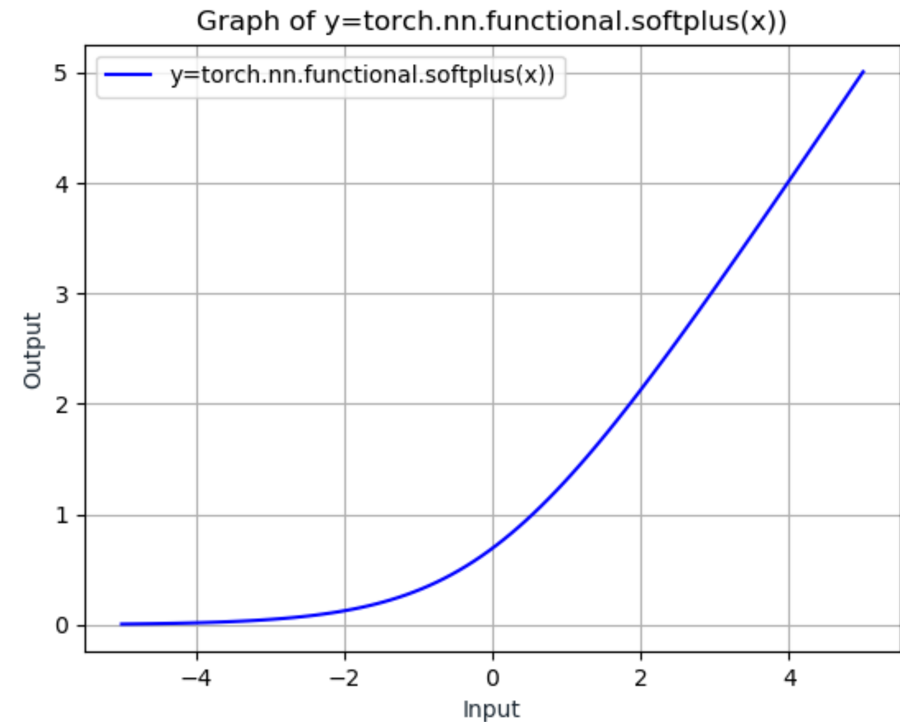
PReLU

- $Output = \begin{cases} x & for\ x \geq 0 \\ a \cdot x & for\ x < 0 \end{cases}$
- a is learnable
- Either one or a separate a is used for each input channel



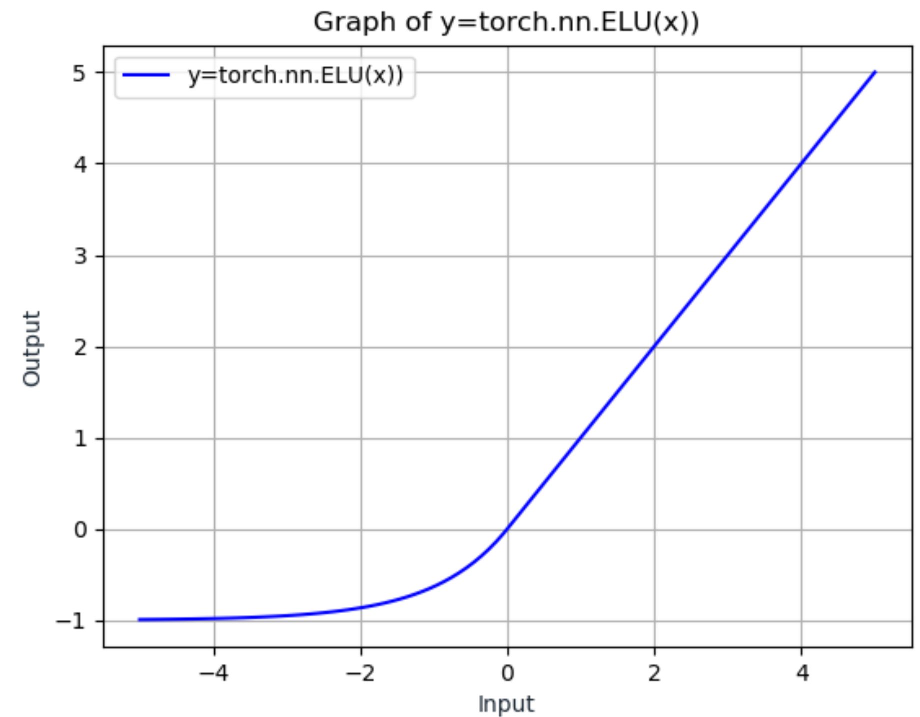
SoftPlus

- $Output = \frac{1}{\beta} \cdot \log\left(1 + e^{(\beta \cdot x)}\right)$
- Smooth approximation of ReLU
- Output always positive
- Numerical stability:
  use linear if
  $(\beta \cdot x) > threshold$



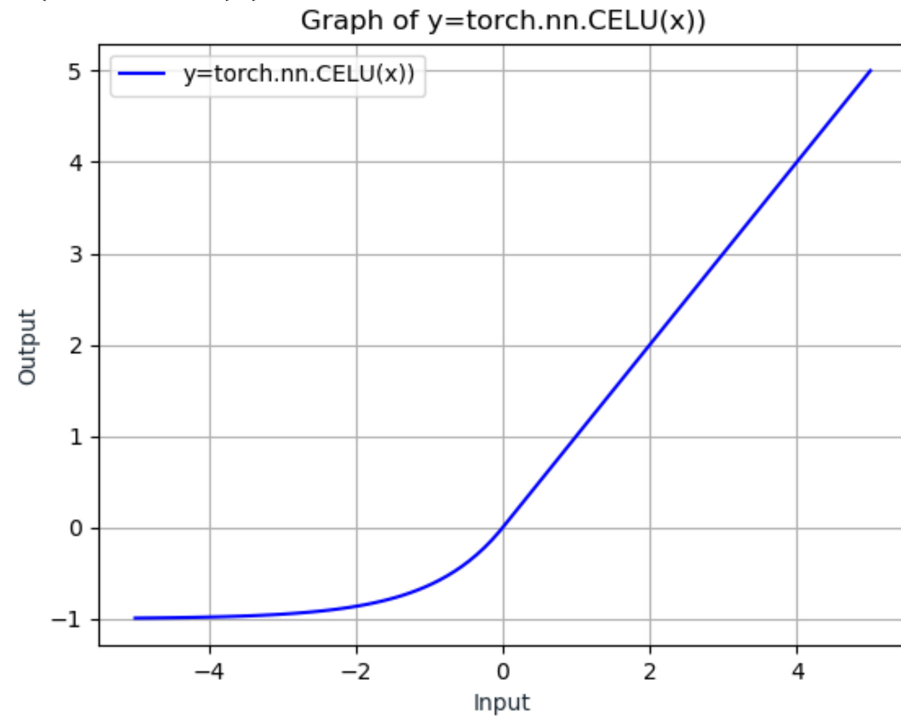
ELU

- $Output = \max(0, x) + \min(0, \alpha \cdot (e^{x} - 1))$
- Element-wise



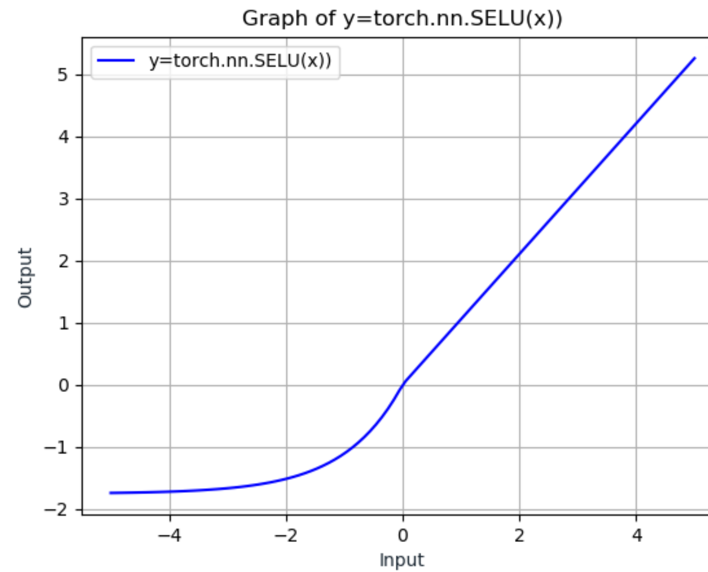
CELU

- $Output = \max(0, x) + \min(0, \alpha \cdot (e^{\frac{x}{\alpha}} - 1))$
- Element-wise
- Barron (2017) https://arxiv.org/abs/1704.07483



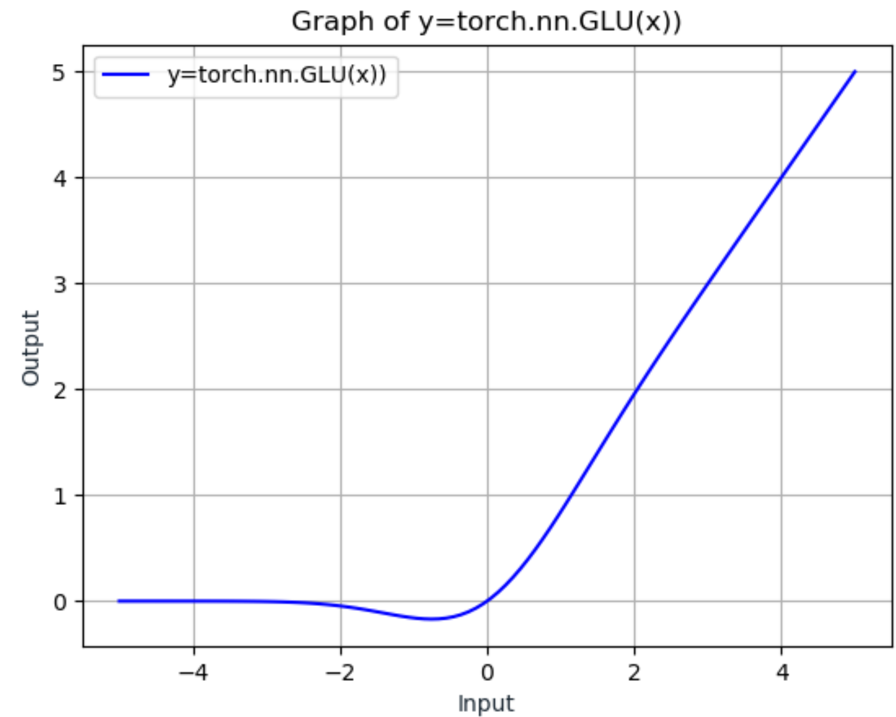
SELU

- $Output = scale \cdot (\max(0, x) + \min(0, \alpha \cdot (e^{x} - 1)))$
- with α = 1.6732632423543772848170429916717 and scale = 1.0507009873554804934193349852946



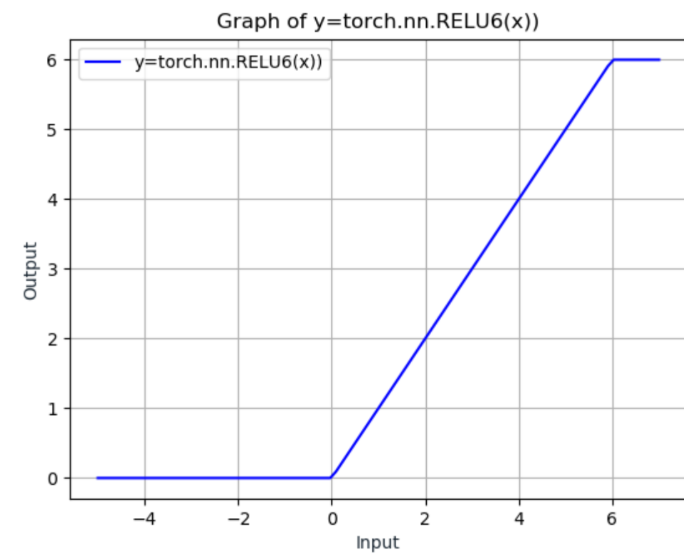
GELU

- $Output = x \cdot \Phi(x)$
- $\Phi(x)$ is the Cumulative Distribution Function for Gaussian Distribution.



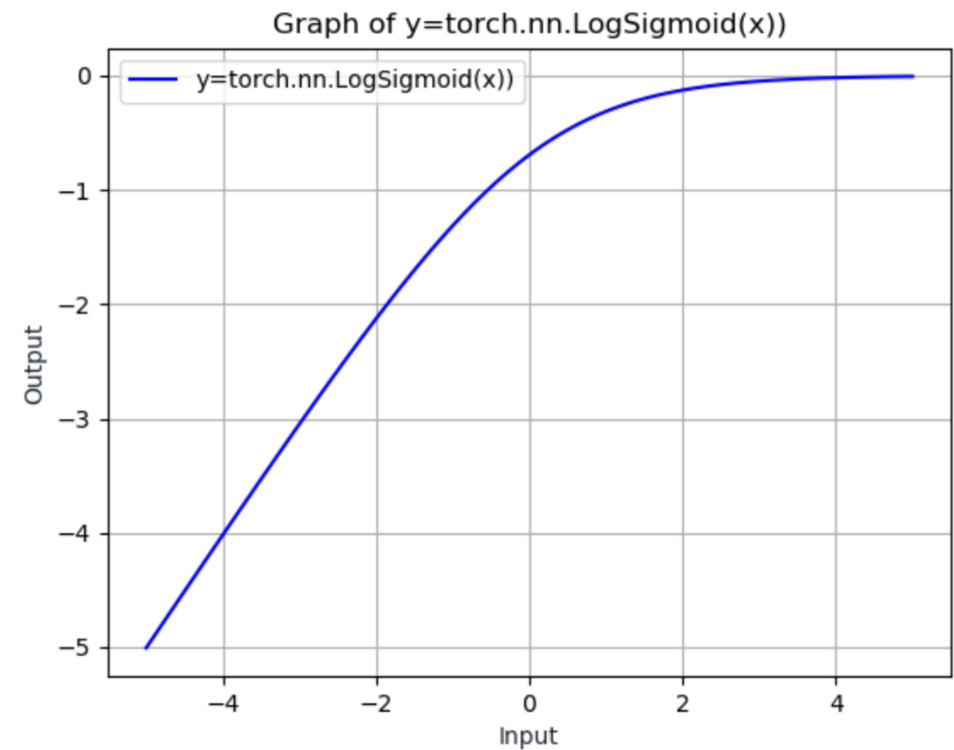
ReLU6

- $Output = \min(\max(0, x), 6)$



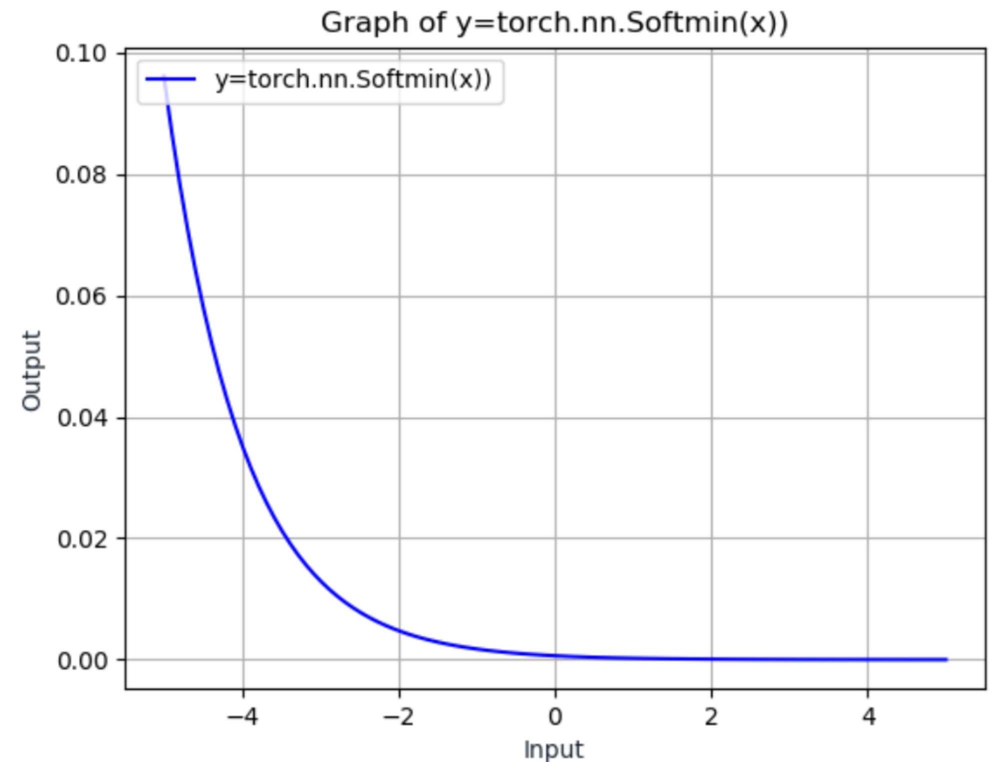
LogSigmoid

- $Output = \log(\frac{1}{1+e^{-x}})$
- Element-wise



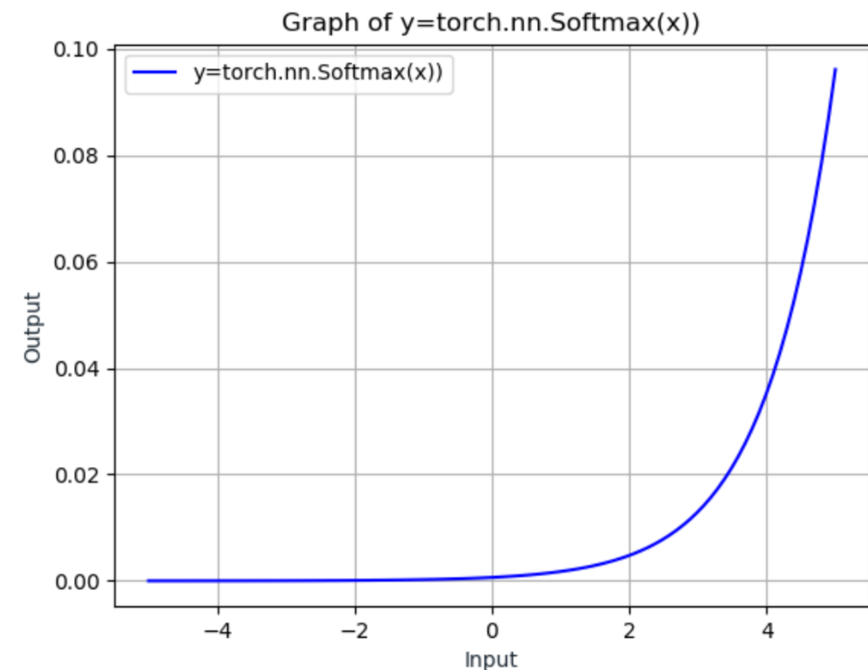
Softmin

- $Output = \frac{e^{-x_i}}{\sum_j e^{-x_j}}$
- Applies the Softmin function 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 up to 1.



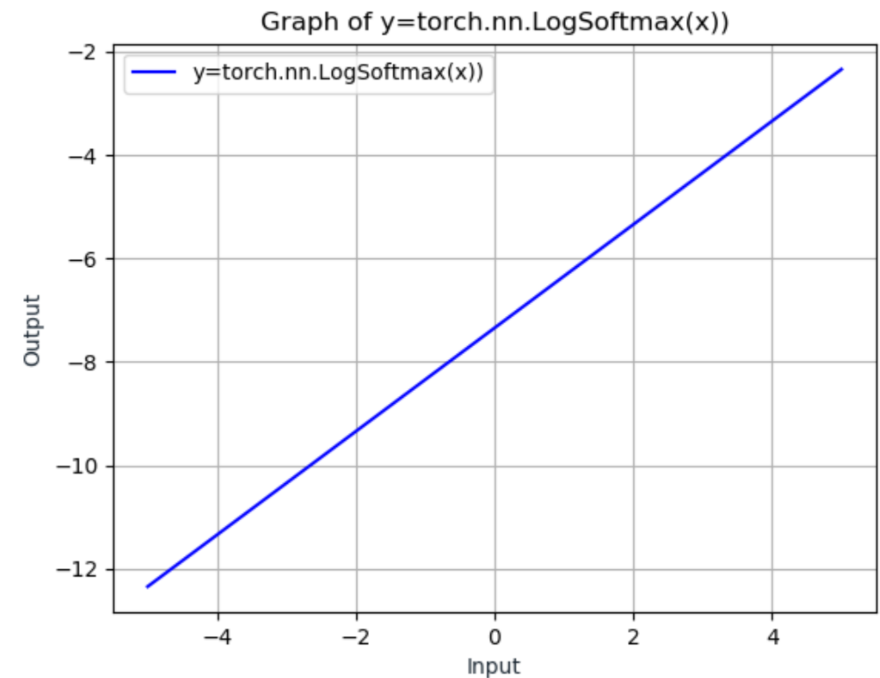
Softmax

- $Output = \frac{e^{x_i}}{\sum_j e^{x_j}}$
- Applies the Softmax function 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 up to 1.



LogSoftmax

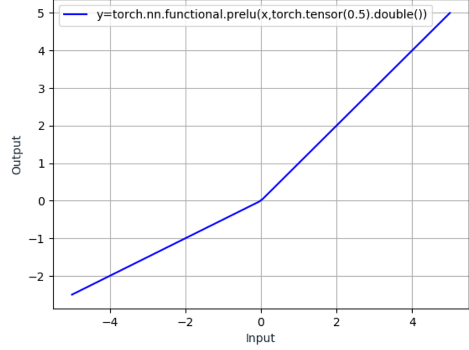
- $Output = \log(\frac{e^{x_i}}{\sum_j e^{x_j}})$
- Applies the log(Softmax) function to an n-dimensional input Tensor

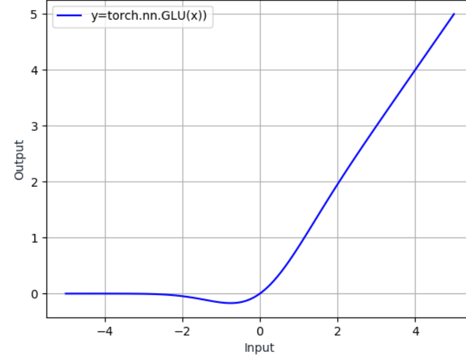


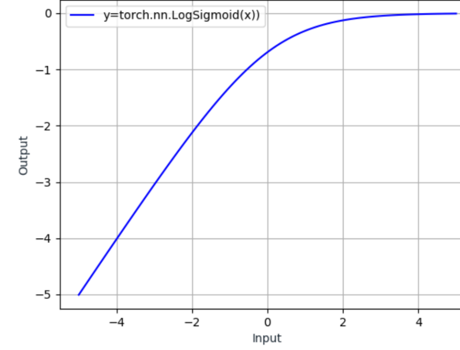
Periodic activations, SIREN

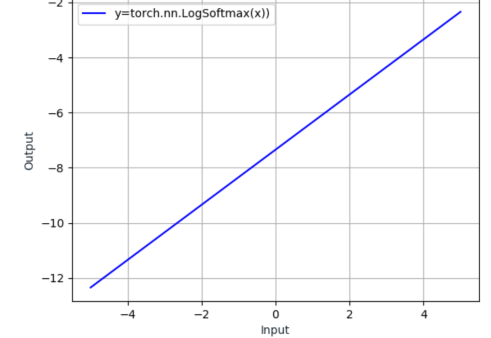
- Difficult convergence properties for general problems
- Has been used for implicit representations (find a continuous function that represents sparse input data, e.g. an image)
- https://vsitzmann.github.io/siren/

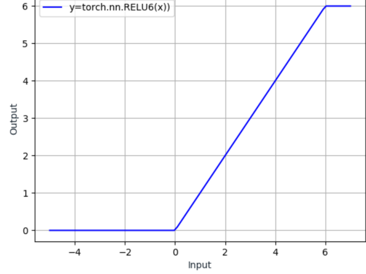
$$\Phi(\mathbf{x}) = \mathbf{W}_n (\phi_{n-1} \circ \phi_{n-2} \circ \ldots \circ \phi_0)(\mathbf{x}) + \mathbf{b}_n, \quad \mathbf{x}_i \mapsto \phi_i(\mathbf{x}_i) = \sin(\mathbf{W}_i \mathbf{x}_i + \mathbf{b}_i).$$

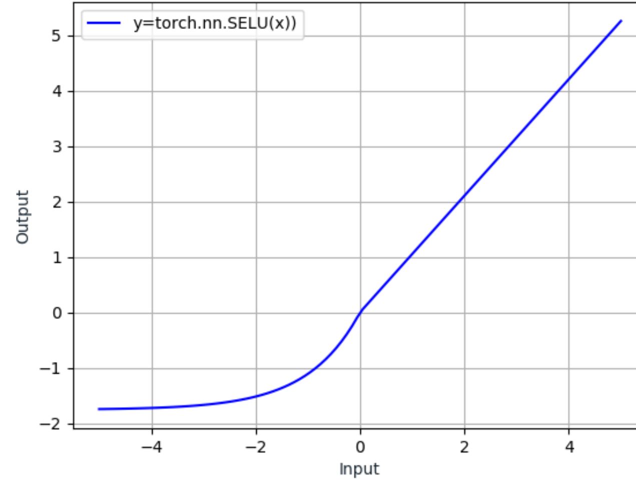

Graph of y=torch.nn.functional.prelu(x,torch.tensor(0.5).double())
y=torch.nn.functional.prelu(x,torch.tensor(0.5).double())
Output
Input

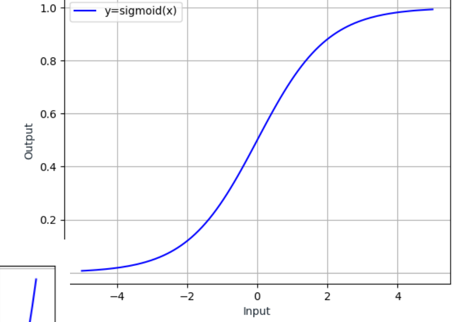

Graph of y=torch.nn.GLU(x))
y=torch.nn.GLU(x))
Output
Input

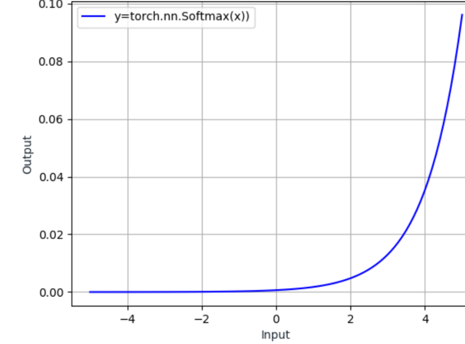

Graph of y=torch.nn.LogSigmoid(x))
y=torch.nn.LogSigmoid(x))
Output
Input


Graph of y=torch.nn.LogSoftmax(x))
y=torch.nn.LogSoftmax(x))
Output
Input


Graph of y=torch.nn.RELU6(x))
y=torch.nn.RELU6(x))
Output
Input


Graph of y=torch.nn.SELU(x))
y=torch.nn.SELU(x))
Output
Input


y=sigmoid(x)
Output
Input


Graph of y=torch.nn.Softmax(x))
y=torch.nn.Softmax(x))
Output
Input

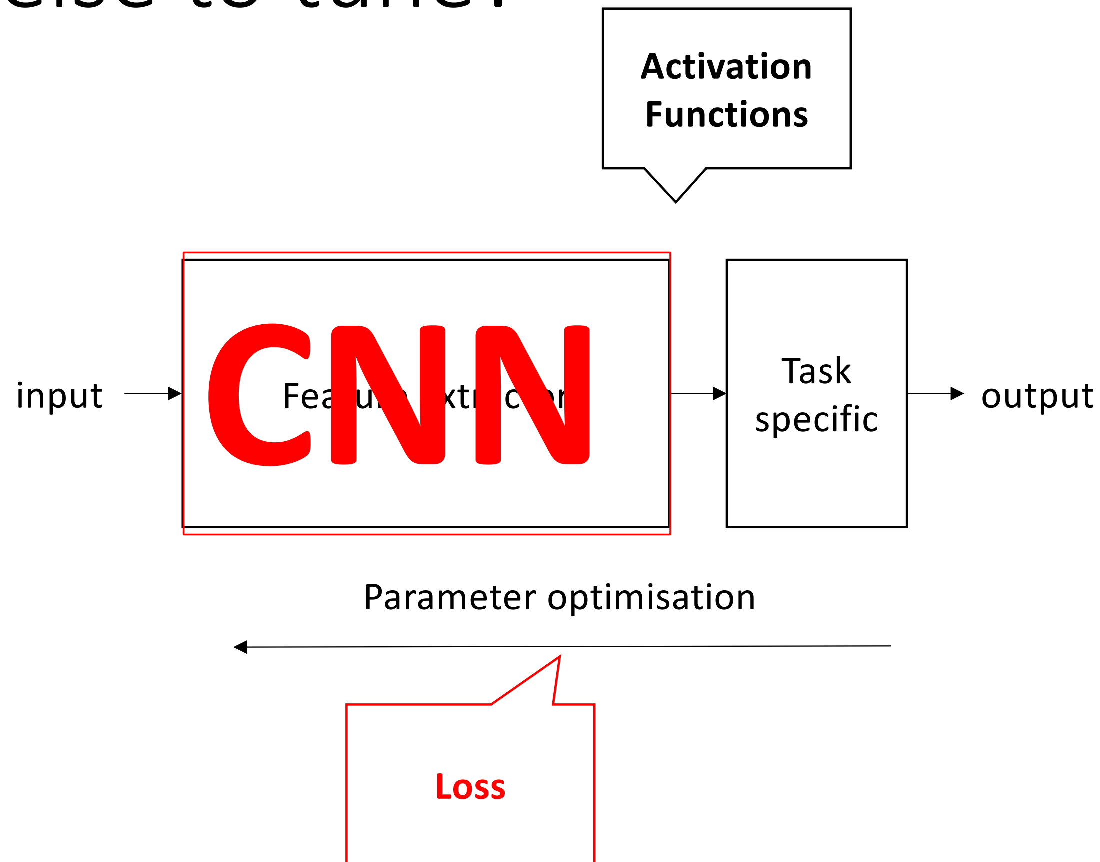
What do we learn from this?

- Which function to use depends on the nature of the targeted problem.
- Most often you will be fine with ReLUs for classification problems. If the network does not converge, use leakyReLUs or PReLUs, etc.
- Tanh is quite ok for regression and continuous reconstruction problems.
- The representative power of you training set will usually outweigh the contribution of a smartly chosen activation function.

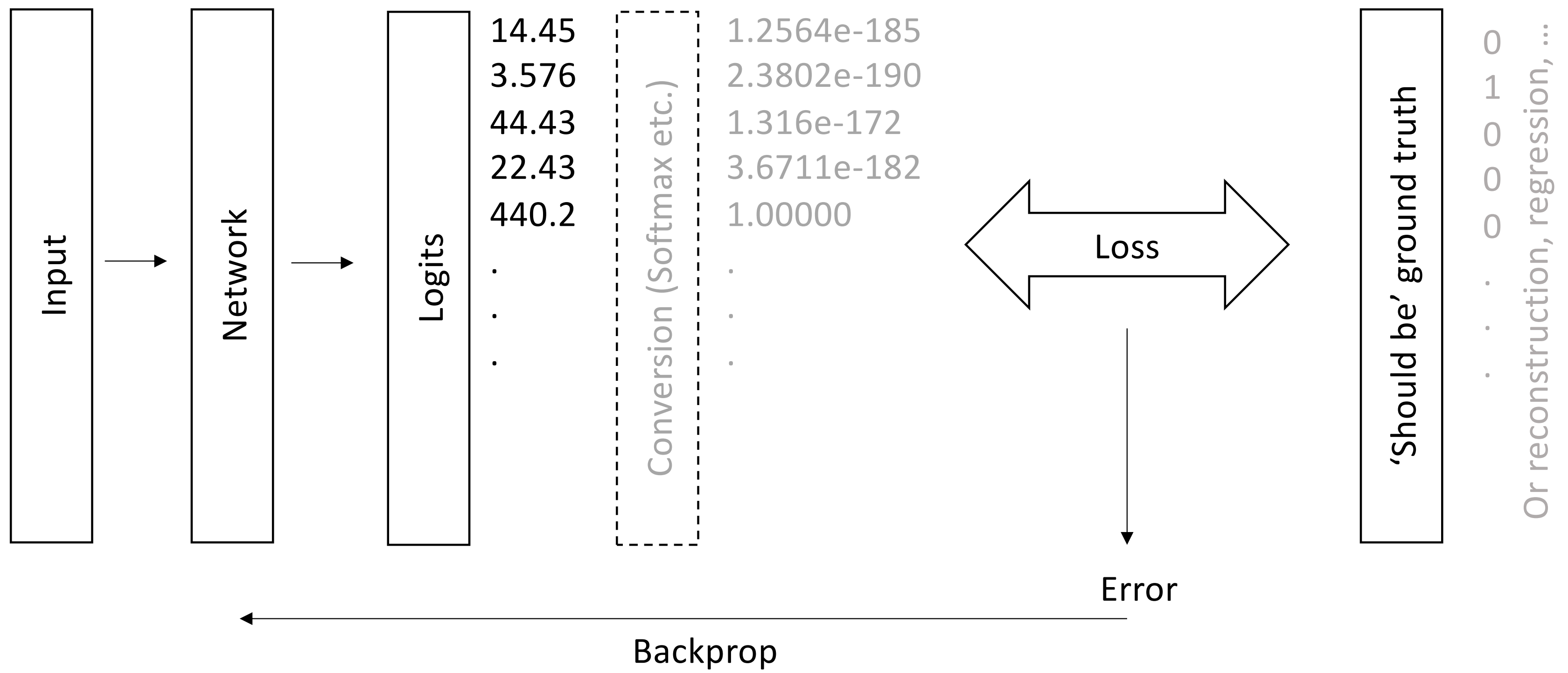
Deep Learning – Loss Functions

Bernhard Kainz

Where else to tune?



Loss functions

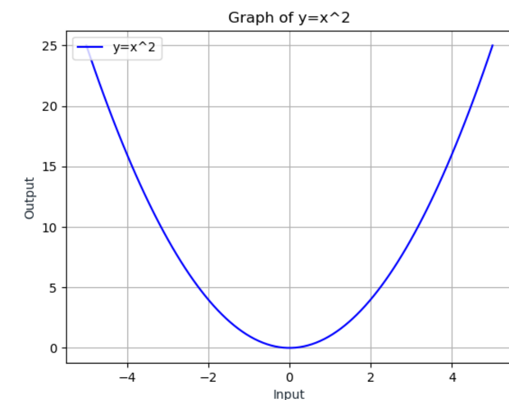


L2 Norm, mean squared error

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

Reduction to a single value can be either $mean(\mathcal{L})$ or sum($\mathcal{L}$).

pytorch: `nn.MSELoss()`



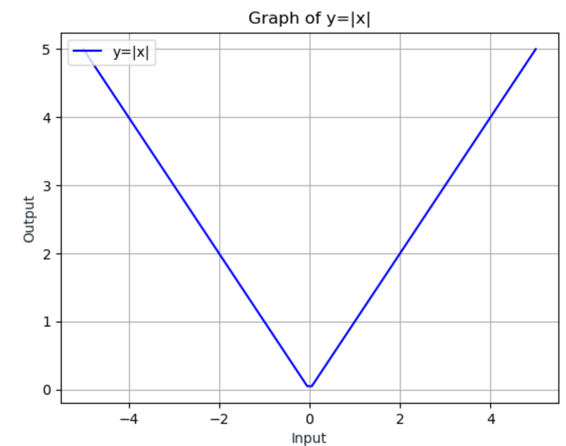
L1 Norm

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

Reduction to a single value can be either $mean(\mathcal{L})$ or sum($\mathcal{L}$).

Use for robust regression (noisy data)

pytorch: `nn.L1Loss()`

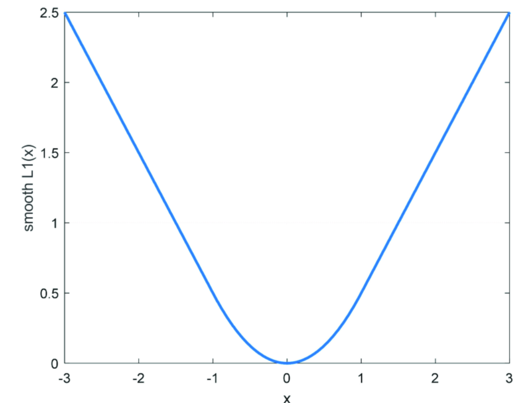


Smooth L1

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

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

pytorch: `nn.SmoothL1Loss()`



Negative log likelihood loss

- Assumption: Network output represents log liklihoods.
- Make the desired output as large as possible and all others as small as possible

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

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

pytorch: `nn.NLLLoss()`
Implementation valid for all such problems, not only likelihoods.

Cross Entropy (CE) Loss

- Combines LogSoftmax and NLLLoss
- Useful for classification problems with C classes

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

Classes can also be weighted.

The losses are averaged across observations for each minibatch.

pytorch: `nn.CrossEntropyLoss()`

Binary Cross Entropy (BCE) Loss

- CE loss for only two classes

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

Reduction to a single value can be either $mean(\mathcal{L})$ or sum($\mathcal{L}$).

pytorch: `nn.BCELoss()`

Requires [0,1] probabilities. If this cannot be guaranteed, use `nn.BCEWithLogitsLoss()`

Kullback-Leibler Divergence Loss

- Measures distance between distributions

$$\ell(x, y) = \mathcal{L} = \{l_1, \dots, l_N\}^T,$$

$$l_n = y_n \cdot (log y_n - log x_n) = y_n \left(log \left(\frac{y_n}{x_n} \right) \right)$$

Pytroch: `nn.KLDivLoss()`

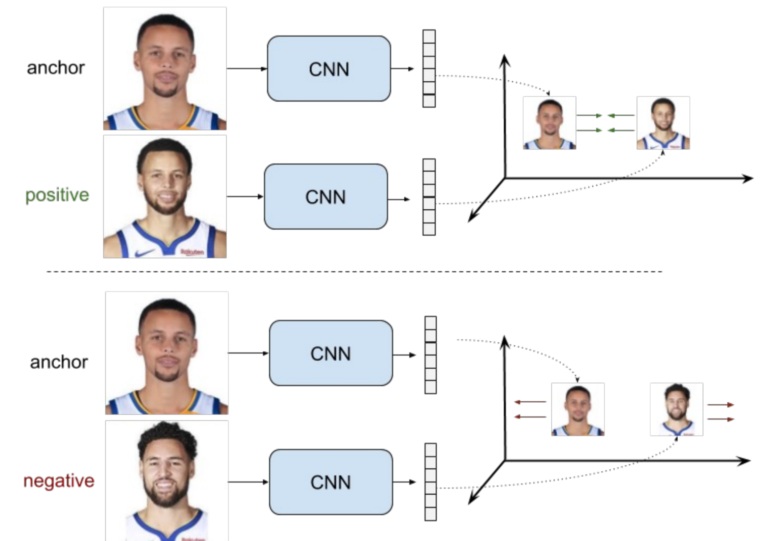
Margin Ranking Loss/Ranking Losses/Contrastive loss

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

Useful to push classes as far away as possible and for metric learning

Practical: take category that scores is closest or higher than correct one change until difference is at least the margin

pytorch: `nn.MarginRankingLoss()`



https://gombru.github.io/

Triplet Margin Loss

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

actual → x_a; positive sample → x_p; negative sample → x_n

Make samples from same classes close and different classes far away.

Objective: Distance for the good pair has to be smaller than distance to the bad pair. Actual distance does not need to be small, just smaller.

Used for metric learning and Siamese networks

pytorch: `nn.TripletMarginLoss()`

Triplet Margin Loss

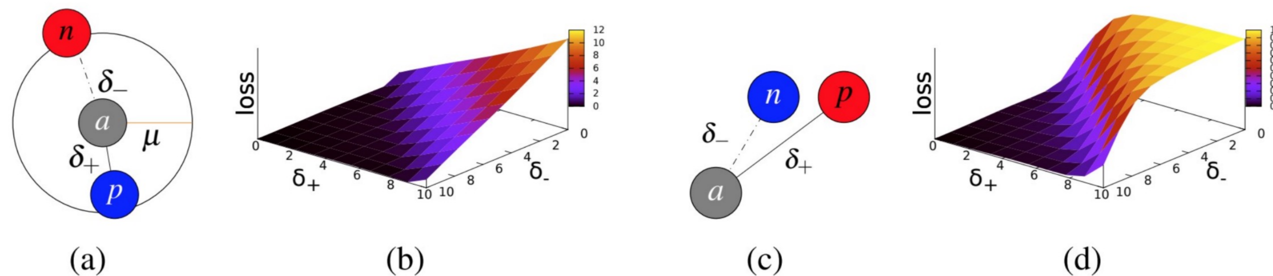


Figure 1: (a) Margin ranking loss. It seeks to push n outside the circle defined by the margin μ, and pull p inside. (b) Margin ranking loss values in function of δ_-, δ_+ (c) Ratio loss. It seeks to force δ_+ to be much smaller than δ_-. (d) Ratio loss values in function of δ_-, δ_+

$$\boldsymbol{\delta}_+ = ||f(\boldsymbol{a}) - f(\boldsymbol{p})||_2 \text{ and } \boldsymbol{\delta}_- = ||f(\boldsymbol{a}) - f(\boldsymbol{n})||_2.$$

$$\lambda(\delta_+, \delta_*) = max(0, \mu + \delta_+ - \delta_*)$$

http://www.bmva.org/bmvc/2016/papers/paper119/index.html

Cosine Embedding Loss

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

Measure weather two inputs are similar or dissimilar

Basically a normalised Euclidian distance

pytroch: `nn.CosineEmbeddingLoss()`

What do we learn from this

- 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.