What are loss functions?

Training the neural network is similar to how humans learn. We give data to the model, it predicts something and we tell it whether the prediction is correct or not. The model then corrects its mistakes. The model does this repeatedly until it reaches a certain level of accuracy, decided by us. Telling the model that the prediction was wrong is crucial for it to learn well. This where the loss function comes in. It tells the model how far off its estimation was from the actual value. While communicating with a human is easier, to tell so to a machine we need a medium(pun intended). This communication needs a ***how*** and a ***what***. The ***How*** is the programming language Python and the ***What*** is the Mathematics. In this post, I'll go through some *Hows*, *Whats* and the intuition behind them.

For a quick recap of how neural networks train, have a look at this amazing post. My post is meant for people who are familiar with Deep Learning. For nitty-gritty details refer Pytorch Docs.

Mean Absolute Error

`torch.nn.L1Loss`

Measures the mean absolute error.

$$\text{loss}(x, y) = |x - y|$$

where *x* is the actual value and *y* is the predicted value.

What does it mean?

It measures the numerical distance between the estimated and actual value. It is the simplest form of error metric. The absolute value of the error is taken because if we don't then negatives will cancel out the positives. This isn't useful to us, rather it makes it more unreliable.

The lower the value of MAE, better is the model. We can not expect its value to be zero, because it might not be practically useful. This leads to wastage of resources. For example, if our model's loss is within 5% then it is alright in practice, and making it more precise may not really be useful.

When to use it?

+ Regression problems
+ Simplistic model
+ As neural networks are usually used for complex problems, this function is rarely used.

Mean Square Error Loss

`torch.nn.MSELoss`

It measures the mean squared error (squared L2 norm).

$$\text{loss}(x, y) = (x - y)^2$$

where *x* is the actual value and *y* is the predicted value.

What does it mean?

The squaring of the difference of prediction and actual value means that we're amplifying large losses. If the classifier is off by 200, the error is 40000 and if the classifier is off by 0.1, the error is 0.01. This penalizes the model when it makes large mistakes and incentivizes small errors.

When to use it?

+ Regression problems.
+ The numerical value features are not large.
+ Problem is not very high dimensional.

Smooth L1 Loss

`torch.nn.SmoothL1Loss`

Also known as Huber loss, it is given by —

$$\text{loss}(x, y) = \begin{cases} 0.5(x - y)^2, & \text{if } |x - y| < 1 \\ |x - y| - 0.5, & \text{otherwise} \end{cases}$$

What does it mean?

It uses a squared term if the absolute error falls below 1 and an absolute term otherwise. It is less sensitive to outliers than the mean square error loss and in some cases prevents exploding gradients. In mean square error loss, we square the difference which results in a number which is much larger than the original number. These high values result in exploding gradients. This is avoided here as for numbers greater than 1, the numbers are not squared.

When to use it?

+ Regression.

+ When the features have large values.
+ Well suited for most problems.

Negative Log-Likelihood Loss

`torch.nn.NLLLoss`

The negative log-likelihood loss:

$$\text{loss}(x, y) = -(\log y)$$

What does it mean?

It maximizes the overall probability of the data. It penalizes the model when it predicts the correct class with smaller probabilities and incentivizes when the prediction is made with higher probability. The logrithm does the penalizing part here. Smaller the probabilities, higher will be its logrithm. The negative sign is used here because the probabilities lie in the range [0, 1] and the logrithms of values in this range is negative. So it makes the loss value to be positive.

When to use it?

+ Classification.
+ Smaller quicker training.
+ Simple tasks.

Cross-Entropy Loss

`torch.nn.CrossEntropyLoss`

Measures the cross-entropy between the predicted and the actual value.

$$\text{loss}(x, y) = -\sum x \log y$$

where *x* is the probability of true label and *y* is the probability of predicted label.

What does it mean?

Cross-entropy as a loss function is used to learn the probability distribution of the data. While other loss functions like squared loss penalize wrong predictions, cross entropy gives a greater penalty when incorrect predictions are predicted with high confidence. What differentiates it with negative log loss is that cross entropy also penalizes wrong but confident predictions and correct

but less confident predictions, while negative log loss does not penalize according to the confidence of predictions.

When to use it?
+ Classification tasks
+ For making confident model i.e. model will not only predict accurately, but it will also do so with higher probability.
+ For higher precision/recall values.

Kullback-Leibler divergence

`torch.nn.KLDivLoss`

KL divergence gives a measure of how two probability distributions are different from each other.

$$\text{loss}(x, y) = \ y \cdot (\log y - x)$$

where *x* is the probability of true label and *y* is the probability of predicted label.

What does it mean?
It is quite similar to cross entropy loss. The distinction is the difference between predicted and actual probability. This adds data about information loss in the model training. The farther away the predicted probability distribution is from the true probability distribution, greater is the loss. It does not penalize the model based on the confidence of prediction, as in cross entropy loss, but how different is the prediction from ground truth. It usually outperforms mean square error, especially when data is not normally distributed. The reason why cross entropy is more widely used is that it can be broken down as a function of cross entropy. Minimizing the cross-entropy is the same as minimizing KL divergence.
KL = — xlog(y/x) = xlog(x) — xlog(y) = Entropy — Cross-entropy

When to use it?
+ Classification
+ Same can be achieved with cross entropy with lesser computation, so avoid it.

Margin Ranking Loss

`torch.nn.MarginRankingLoss`

It measures the loss given inputs *x1*, *x2*, and a label tensor *y* with values (1 or -1). If y == 1 then it assumed the first input should be ranked higher than the second input, and vice-versa for y == -1.

$$\text{loss}(x, y) = \max(0, -y * (x1 - x2) + \text{margin})$$

What does it mean?

The prediction y of the classifier is based on the ranking of the inputs *x1* and x2. Assuming margin to have the default value of 0, if *y* and *(x1-x2)* are of the same sign, then the loss will be zero. This means that *x1/x2* was ranked higher(for *y=1/-1*), as expected by the data. If *y* and *(x1-x2)* are of the opposite sign, then the loss will be the non-zero value given by *y * (x1-x2)*. This means that either *x2* was ranked higher when *x1* should have been ranked higher or vice versa. Although its usage in Pytorch in unclear as much open source implementations and examples are not available as compared to other loss functions.

When to use it?

+ GANs.
+ Ranking tasks.

Hinge Embedding Loss

`torch.nn.HingeEmbeddingLoss`

Measures the loss given an input tensor *x* and a labels tensor *y* containing values (1 or -1). It is used for measuring whether two inputs are similar or dissimilar.

$$\text{loss}(x, y) = \begin{cases} x\ , & \text{if } y\ = 1, \\ \max\{0, \Delta - x\ \}, & \text{if } y\ = -1, \end{cases}$$

What does it mean?

The prediction y of the classifier is based on the value of the input *x*. Assuming margin to have the default value of 1, if *y=-1*, then the loss will be maximum of 0 and *(1 — x)*. If *x* > 0 loss will be *x* itself (higher value), if 0<*x*<1 loss will be *1 — x* (smaller value) and if *x* < 0 loss will be 0 (minimum value). For *y* =1, the loss is as high as the value of *x*.

When to use it?

+ Learning nonlinear embeddings
+ Semi-supervised learning
+ Where similarity or dissimilar of two inputs is to be measured.

Cosine Embedding Loss

`torch.nn.CosineEmbeddingLoss`

It measures the loss given inputs *x1*, *x2*, and a label tensor *y* containing values (1 or -1). It is used for measuring whether two inputs are similar or dissimilar.

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

What does it mean?

The prediction y of the classifier is based on the cosine distance of the inputs *x1* and x2. Cosine distance refers to the angle between two points. It can be easily found out by using dot products as:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{|\mathbf{A}| \ |\mathbf{B}|}$$

As cosine lies between - 1 and + 1, loss values are smaller. This aids in computation. Assuming margin to have the default value of 0, if *y* =1, the loss is (1 - cos(*x1, x2)*). For *y=-1*, then the loss will be maximum of 0 and cos*(x1, x2)*. If cos*(x1, x2)* > 0 loss will be cos*(x1, x2)* itself (higher value), and if cos*(x1, x2)* < 0 loss will be 0 (minimum value).

When to use it?

+ Learning nonlinear embeddings
+ Semi-supervised learning
+ Where similarity or dissimilar of two inputs is to be measured.