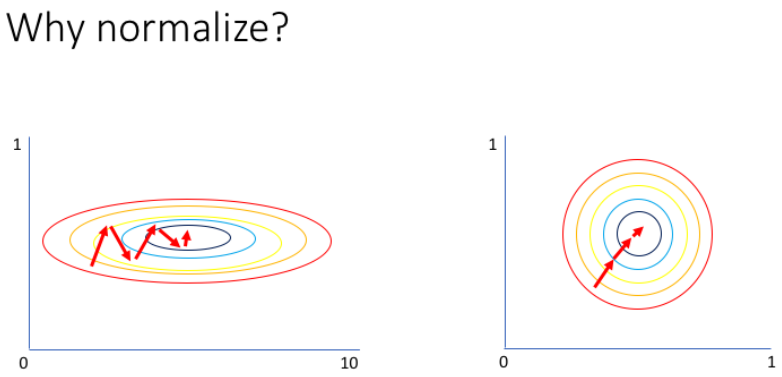
AI Coffe Club (11/12/2019)
Normalization

Last modification: December 11, 2019

1 Why is Normalization Important?

Consider a simple scenario in which the neural network takes two inputs (let's say x_0 and x_1) and such variables have significantly different ranges $x_0 = [0, 1]$ and $x_1 = [0, 10]$. The goal of the network is to learn a set of weights to combine such inputs throuh linear combinations and non-linear activations. This implies that the weights associated to each input will belong to different ranges.

Such phenomena has a direct impact in the topology of the loss function as shown below:



This will indirectly lead to the network placing more emphasis on specific parameter gradients just because they produce larger activations. This fact makes training harder and hurts convergence which in turn leads to needing more iterations to train a network properly.

2 How to Normalize the Input

An easy way to avoid this problem is to normalize the input to have zero mean and unit variance using the standardization approach (putting the data in the range $[-1, 1]$).

In some cases, standardization can hurt performance depending on the activation layers. For instance, ReLU activations tend to show better performance if the data is just scaled to the $[0, 1]$ range instead of standardized to $[-1, 1]$.

3 How to Normalize between Layers

The aforementioned approach solves the problem of normalizing the input but that only affects significantly the first layers of the network. Unfortunately, the distribution of each layer's input changes during training since the weights from the previous layers change as we iterate through the training set.

This leads to slower training (requiring lower learning rates to reach convergence), requires careful initialization, and makes it really hard to train certain models with saturating non-linearities. This problem is named internal covariance shift.

However, knowing that input normalization helps, this intuition can be extended to the whole network to help it learn more effectively by making normalization a part of the architecture.

We would like to normalize the activations of a given layer so that we improve the learning of the weights that connect to the next layer. This technique is known as batch normalization [1].

There is a debate whether to normalize the input sum z or the activations a, but in practice most people normalize the input sum.

Considering z_i^l (the input sum from a layer l for each observation of the batch i, we compute the batch mean $\mu_{\mathcal{B}}^l$ and variance ${\sigma^2}_{\mathcal{B}}^l$:

$$\mu_{\mathcal{B}}^l = \frac{1}{m} \sum_{i=0}^{m} z_i^l \tag{1}$$

$${\sigma^2}_{\mathcal{B}}^l = \frac{1}{m} \sum_{i=0}^{m} (z_i^l - \mu_{\mathcal{B}}^l)^2 \tag{2}$$

Using those values, we can normalize the input sums to have zero mean and unit variance as follows:

$$z_{inorm}^l = \frac{z_i^l - \mu_{\mathcal{B}}^l}{\sqrt{{\sigma^2}_{\mathcal{B}}^l + \epsilon}} \tag{3}$$

(Note that a small ϵ is added for numerical reasons to avoid a possible division by zero).

[1] `https://arxiv.org/pdf/1502.03167.pdf`

Nevertheless, in some cases we don't want to explicitly normalize to zero mean and unit variance (this is actually bad for some activation functions such as sigmoid) but rather we allow the network to learn the optimal distribution by scaling the normalized values by γ and shifting by β:

$$\hat{z}_i^l = \gamma z_{i_{norm}}^l + \beta \tag{4}$$

(Note: γ and β are learned parameters which are used across all batches).