INF721 - Deep Learning
L12: Normalization

Prof. Lucas N. Ferreira
Universidade Federal de Viçosa

2024/2

1 Introduction

Normalization is a fundamental technique in deep learning that helps accelerate and stabilize the training of neural networks. In this lecture, we explore three main types of normalization:

- Input Normalization
- Batch Normalization
- Layer Normalization

2 Input Normalization

When training neural networks, having input features with significantly different scales can make the optimization process challenging. Consider the following house price dataset:

Size (m^2)	Number of Beds.	Nearest Subway Station (m)	Price (1000's of USD)
152	4	7200	1550
229	3	3000	2286
84	1	1500	2930
...	...	...	...
95	3	12000	196

Table 1: House Price Prediction Dataset

Note how the "Nearest Subway Station" feature has a much larger scale compared to the other features. To understand why these different scales matter, let's consider training a linear regression model using stochastic gradient descent:

Impact of Different Scales

Initial conditions:

- Weights: $\mathbf{w} = [0, 0, 0]$
- Learning rate: $\alpha = 0.1$
- Update rule: $\mathbf{w} = \mathbf{w} - \alpha(\hat{y}^{(i)} - y^{(i)})\mathbf{x}^{(i)}$

For the first example:

$$\begin{aligned}\mathbf{w} &= [0, 0, 0] + 155 \cdot [152, 4, 7200] \\ &= [23560, 620, 1116000]\end{aligned}$$

Notice how the weight corresponding to the "Nearest Subway Station" feature gets updated by a much larger amount due to its scale!

Having different scales in the input features make the error landscape have different curvatures along different dimensions, as shown in the figure below for two parameters w and b:

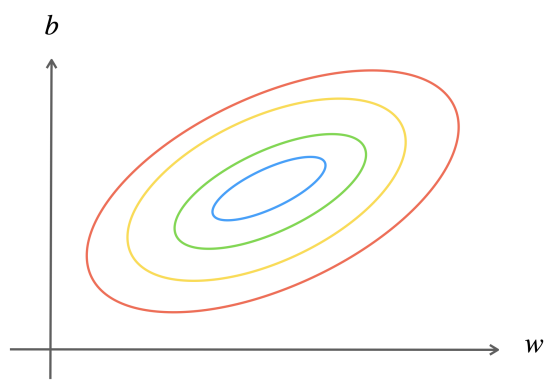

Figure 1: Error Landscape with Different Feature Scales

This significant difference in feature scales leads to:

- Different magnitudes in gradients
- Slower convergence
- Numerical instability
- Difficulty in selecting an appropriate learning rate

2.1 Formulation

Given a dataset $\mathbf{X}$ with m examples, input normalization transforms each feature to have zero mean and unit variance:

$$\mathbf{x}_{\text{norm}}^{(i)} = \frac{\mathbf{x}^{(i)} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \tag{1}$$

where:

$$\boldsymbol{\mu} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}^{(i)} \quad \text{(mean)}$$
$$\boldsymbol{\sigma}^2 = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}^{(i)} - \boldsymbol{\mu})^2 \quad \text{(variance)}$$

Let's walk through the normalization process for our house price dataset:

1. Original data as a matrix $\mathbf{X}$:

$$\mathbf{X} = \begin{bmatrix} 152 & 229 & 84 & 95 \\ 4 & 3 & 1 & 3 \\ 7200 & 3000 & 1500 & 12000 \end{bmatrix} \tag{2}$$

   , where rows represent features: size (m^2), number of bedrooms, and subway distance (m).

2. Computing Mean

   Mean vector $\boldsymbol{\mu}$ for each feature:

$$\boldsymbol{\mu} = \frac{1}{4}\sum_{i=1}^{4}\mathbf{x}^{(i)} \qquad = \frac{1}{4}\begin{bmatrix} 152+229+84+95 \\ 4+3+1+3 \\ 7200+3000+1500+12000 \end{bmatrix} = \begin{bmatrix} 140 \\ 2.75 \\ 5925 \end{bmatrix}$$

3. Compute Variance

   For each feature, subtract mean and square:

$$(\mathbf{X} - \boldsymbol{\mu}) = \begin{bmatrix} 12 & 89 & -56 & -45 \\ 1.25 & 0.25 & -1.75 & 0.25 \\ 1275 & -2925 & -4425 & 6075 \end{bmatrix} \tag{3}$$

   Square each element:

$$(\mathbf{X} - \boldsymbol{\mu})^2 = \begin{bmatrix} 144 & 7921 & 3136 & 2025 \\ 1.5625 & 0.0625 & 3.0625 & 0.0625 \\ 1625625 & 8555625 & 19580625 & 36905625 \end{bmatrix} \tag{4}$$

Compute variance $\boldsymbol{\sigma}^2$ by taking mean of squared differences:

$$\boldsymbol{\sigma}^2 = \frac{1}{4}\sum_{i=1}^{4}(\mathbf{x}^{(i)} - \boldsymbol{\mu})^2 \qquad = \begin{bmatrix} 3306.5 \\ 1.1875 \\ 16666875 \end{bmatrix}$$

Take square root to get standard deviation:

$$\boldsymbol{\sigma} = \begin{bmatrix} 57.50 \\ 1.09 \\ 4082.50 \end{bmatrix} \tag{5}$$

4. Normalize Input

   Apply the normalization formula to each feature:

$$\mathbf{X}_{\text{norm}} = \frac{\mathbf{X} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \tag{6}$$

   For each feature j and example i:

$$x^{(i)}_{\text{norm},j} = \frac{x^{(i)}_j - \mu_j}{\sigma_j} \tag{7}$$

   This gives us:

$$\mathbf{X}_{\text{norm}} = \begin{bmatrix} 0.209 & 1.548 & -0.974 & -0.783 \\ 1.147 & 0.229 & -1.606 & 0.229 \\ 0.312 & -0.717 & -1.084 & 1.489 \end{bmatrix} \tag{8}$$

When you normalize the input features, you transform the error landscape into a more isotropic shape, which helps the optimization algorithm converge faster and more reliably. See figure below for an example with only two parameters w and b:

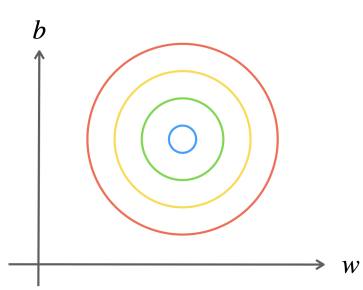

Figure 2: Error Landscape after Input Normalization

2.2 Implementation

Here's a simple implementation in NumPy:

Input Normalization in NumPy

```python
def normalize_input(X):
    """
    X: input data of shape (n_features, n_examples)
    """
    mean = np.mean(X, axis=1, keepdims=True)
    std = np.std(X, axis=1, keepdims=True)
    X_norm = (X - mean) / (std + 1e-8)
    return X_norm, mean, std
```

Important: The same $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ used to normalize the training set must be used for validation and test sets!

3 Batch Normalization

While input normalization helps with the first layer, the distributions of activations in deeper layers can still shift during training, a phenomenon known as internal covariate shift. Batch normalization (BatchNorm) addresses this by normalizing the activations at each layer. For a minibatch of size m, BatchNorm normalizes the pre-activations $\mathbf{z}^{[l]}$ of layer l:

$$
\begin{aligned}
\boldsymbol{\mu}_B &= \frac{1}{m}\sum_{i=1}^{m}\mathbf{z}^{[l](i)} \\
\boldsymbol{\sigma}_B^2 &= \frac{1}{m}\sum_{i=1}^{m}(\mathbf{z}^{[l](i)} - \boldsymbol{\mu}_B)^2 \\
\hat{\mathbf{z}}^{[l](i)} &= \frac{\mathbf{z}^{[l](i)} - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \epsilon}} \\
\mathbf{z}_{\mathrm{BN}}^{[l](i)} &= \boldsymbol{\gamma} \odot \hat{\mathbf{z}}^{[l](i)} + \boldsymbol{\beta}
\end{aligned}
$$

where $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learnable parameters that allow the network to recover the original representation if needed.

4 Layer Normalization

BatchNorm has limitations when the batch size is small, as the estimates of mean and variance become noisy. Layer normalization (LayerNorm) addresses

this by normalizing across features instead of across the batch. For each example i, LayerNorm normalizes across all $n^{[l]}$ units in layer l:

$$\mu^{(i)} = \frac{1}{n^{[l]}} \sum_{j=1}^{n^{[l]}} z_j^{[l](i)}$$

$$(\sigma^{(i)})^2 = \frac{1}{n^{[l]}} \sum_{j=1}^{n^{[l]}} (z_j^{[l](i)} - \mu^{(i)})^2$$

$$\hat{z}_j^{[l](i)} = \frac{z_j^{[l](i)} - \mu^{(i)}}{\sqrt{(\sigma^{(i)})^2 + \epsilon}}$$

$$z_j^{[l](i)_{\text{LN}}} = \gamma_j \hat{z}_j^{[l](i)} + \beta_j$$

5 Examples

Let's consider a concrete example with real values to understand how batch and layer normalization work in practice. Given input matrix $\mathbf{X}$ and weight matrix $\mathbf{W}^{[1]}$:

$$\mathbf{X} = \begin{bmatrix} 1.0 & 2.0 & -1.0 & 0.0 \\ 0.5 & 1.0 & 0.5 & -1.0 \\ 0.0 & 0.0 & 1.0 & -0.5 \end{bmatrix}, \quad \mathbf{W}^{[1]} = \begin{bmatrix} 0.1 & -0.2 & 0.1 \\ 0.2 & 0.1 & -0.1 \\ -0.1 & 0.2 & 0.1 \end{bmatrix} \tag{9}$$

With bias vector $\mathbf{b}^{[1]} = [0, 0, 0]^T$, the pre-activation matrix $\mathbf{Z}^{[1]}$ is:

$$\mathbf{Z}^{[1]} = \begin{bmatrix} 0.2 & 0.4 & -0.1 & -0.15 \\ -0.15 & -0.3 & 0.45 & -0.2 \\ 0.05 & 0.1 & -0.05 & 0.05 \end{bmatrix} \tag{10}$$

5.1 Batch Normalization Computation

For batch normalization, we normalize across examples (columns) for each feature (row):

Computing mean $\boldsymbol{\mu}_B$ (across columns):

$$\boldsymbol{\mu}_B = \begin{bmatrix} 0.08 \\ -0.05 \\ 0.03 \end{bmatrix} \tag{11}$$

Computing variance $\boldsymbol{\sigma}_B^2$ (across columns):

$$\boldsymbol{\sigma}_B^2 = \begin{bmatrix} 0.05 \\ 0.08 \\ 0.02 \end{bmatrix} \tag{12}$$

Normalized matrix:

$$\mathbf{Z}_{BN}^{[1]} = \begin{bmatrix} 0.50 & 1.39 & -0.83 & -1.05 \\ -0.34 & -0.85 & 1.70 & -0.51 \\ 0.22 & 1.14 & -1.60 & 0.22 \end{bmatrix} \tag{13}$$

5.2 Layer Normalization Computation

For layer normalization, we normalize across features (rows) for each example:

Computing mean $\boldsymbol{\mu}_L$ (across rows):

$$\boldsymbol{\mu}_L = \begin{bmatrix} 0.03 & 0.06 & 0.10 & -0.10 \end{bmatrix} \tag{14}$$

Computing variance $\boldsymbol{\sigma}_L^2$ (across rows):

$$\boldsymbol{\sigma}_L^2 = \begin{bmatrix} 0.02 & 0.08 & 0.06 & 0.01 \end{bmatrix} \tag{15}$$

Normalized matrix:

$$\mathbf{Z}_{LN}^{[1]} = \begin{bmatrix} 1.16 & 1.16 & -0.80 & -0.46 \\ -1.27 & -1.27 & 1.40 & -0.92 \\ 0.11 & 0.11 & -0.60 & 1.38 \end{bmatrix} \tag{16}$$

The figure below illustrates the difference between BatchNorm and LayerNorm. Note how BatchNorm normalizes across examples (columns) while LayerNorm normalizes across features (rows).

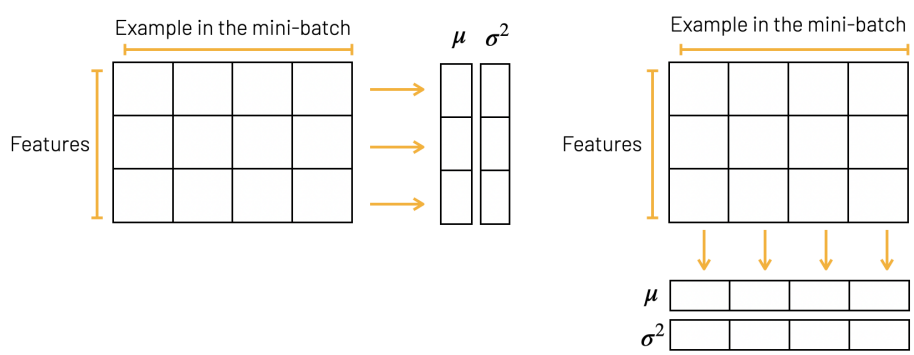

Figure 3: Comparison of BatchNorm and LayerNorm

6 Conclusion

Normalization is a crucial technique in deep learning that helps stabilize and accelerate training. Here are some key takeaways:

- **Input Normalization**:

1. Normalizes features across the entire dataset.
2. Applied only once before training.
3. Always apply input normalization as a preprocessing step.

- **Batch Normalization**:

  1. Normalizes activations across examples in a minibatch.
  2. Use BatchNorm when batch size is sufficiently large (e.g, 32 or more).
  3. BatchNorm is more approapriate for computer vision problems (and hence CNNs) because features depend on the statistical parameters between different samples.

- **Layer Normalization**:

  1. Normalizes activations across features for each example independently.
  2. Use LayerNorm when batch size is small.
  3. LayerNorm is more appropriate for Natual Lenguage Processing problems because the different features of a single sample are actually the variations in words over time, and the feature relationships within the sample are very close.

Moreover, keep track of normalization statistics (mean, variance) for inference and be careful with the axis of normalization in your implementation!

Exercises

1. Given the following input matrix X:

$$\mathbf{X} = \begin{bmatrix} 4 & 6 & 2 \\ 8 & 12 & 4 \end{bmatrix}$$

   Calculate the normalized values for the first row after applying input normalization. Round to 2 decimal places.

   (a) $[-0.82, 1.63, -0.82]$
   (b) $[-1.00, 1.41, -0.41]$
   (c) $[-0.71, 1.41, -0.71]$
   (d) $[-0.58, 1.15, -0.58]$

2. Consider a neural network layer with batch size of 4 and 3 features. Given the following pre-activation values for one batch:

$$\mathbf{Z}^{[1]} = \begin{bmatrix} 2 & -1 & 0 & 1 \\ -2 & 3 & -1 & 0 \\ 4 & 1 & 2 & -1 \end{bmatrix}$$

   When applying batch normalization to the first feature (first row), what is the normalized value for the first example (first column)?

   (a) 0.89
   (b) 1.22
   (c) 0.76
   (d) 1.41

3. What is the key difference between BatchNorm and LayerNorm in terms of how they normalize the activations?

   (a) BatchNorm normalizes across channels while LayerNorm normalizes across spatial dimensions
   (b) BatchNorm normalizes across the batch dimension while LayerNorm normalizes across the feature dimension
   (c) BatchNorm normalizes each feature independently while LayerNorm normalizes all features together
   (d) BatchNorm uses learnable parameters while LayerNorm uses fixed parameters

4. Given a pre-activation matrix for a single example:

$$\mathbf{z}^{[1]} = \begin{bmatrix} 4 & 2 & -2 \end{bmatrix}$$

   Calculate the LayerNorm output for the first element. Round to 2 decimal places.

(a) 0.82
(b) 1.22
(c) 1.41
(d) 0.91

5. When should you prefer LayerNorm over BatchNorm? Select all that apply:

   (a) When working with small batch sizes
   (b) When working with NLP tasks
   (c) When working with CNNs for computer vision
   (d) When working with fixed batch sizes