This discussion will walk through the derivation of the forward and backward passes of the affine layer, ReLU, and BatchNorm as well as the Python implementations for the project.

1 Affine Layer Forward and Backward Pass Derivation

1.1 Forward Pass

The affine (or fully-connected) layer performs a linear transformation on the input. Given:

- $\mathbf{x} \in \mathbb{R}^{N\times D}$: The input, where N is the number of samples and $D = d_1 \times d_2 \times \cdots \times d_k$ is the flattened dimension of each input sample.
- $W \in \mathbb{R}^{D\times M}$: The weight matrix, where M is the dimension of the output vector.
- $\mathbf{b} \in \mathbb{R}^{M}$: The bias vector, which is added element-wise to each transformed input.

The affine transformation can be written as:

$$\mathbf{z} = \mathbf{x}W + \mathbf{b},$$

where $\mathbf{z} \in \mathbb{R}^{N\times M}$ is the output of the layer.

1.1.1 Steps for the Forward Pass

1. Reshape the input $\mathbf{x}$ from $(N, d_1, d_2, \ldots, d_k)$ to (N, D), where $D = d_1 \times d_2 \times \cdots \times d_k$. 2. Perform the matrix multiplication of the reshaped input and the weight matrix: $\mathbf{x}W$. 3. Add the bias term $\mathbf{b}$ to each row of the result, producing the final output $\mathbf{z}$:

$$\mathbf{z} = \mathbf{x}W + \mathbf{b}.$$

Thus, the forward pass computes:

$$\text{out} = \mathbf{x}W + \mathbf{b}.$$

1.2 Backward Pass Derivation

The goal of the backward pass is to compute the gradients of the loss L with respect to the inputs, weights, and biases. We are given the upstream gradient $\frac{\partial L}{\partial \mathbf{z}} = \text{dout}$, which has the same shape as $\mathbf{z}$. Now, we compute the following:

1.2.1 1. Gradient with respect to the input x

From the forward pass, we know that $\mathbf{z} = \mathbf{x}W + \mathbf{b}$. Applying the chain rule, the gradient with respect to the input $\mathbf{x}$ is:

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}}.$$

Since $\mathbf{z} = \mathbf{x}W + \mathbf{b}$, we have:

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = W^T.$$

Thus, the gradient with respect to the input $\mathbf{x}$ is:

$$\frac{\partial L}{\partial \mathbf{x}} = \text{dout} \cdot W^T.$$

Now, because the input $\mathbf{x}$ was originally reshaped from $(N, d_1, d_2, \ldots, d_k)$ to (N, D) during the forward pass, we need to reshape the gradient dx back to the original shape of $\mathbf{x}$:

$$\text{dx} = \left(\text{dout} \cdot W^T\right).\mathit{reshape}(\mathbf{x}.\text{shape}).$$

1.2.2 2. Gradient with respect to the weights W

Now, we compute the gradient of the loss with respect to the weights W. Again, applying the chain rule:

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial W}.$$

Since $\mathbf{z} = \mathbf{x}W + \mathbf{b}$, we have:

$$\frac{\partial \mathbf{z}}{\partial W} = \mathbf{x}^T.$$

Thus, the gradient with respect to W is:

$$\frac{\partial L}{\partial W} = \mathbf{x}^T \cdot \text{dout}.$$

In the forward pass, the input $\mathbf{x}$ was reshaped to (N, D). Therefore, we first reshape $\mathbf{x}$ to (N, D) before performing the matrix multiplication:

$$\text{dw} = \mathbf{x}.reshape(N, D)^T \cdot \text{dout}.$$

1.2.3 3. Gradient with respect to the biases b

Finally, we compute the gradient of the loss with respect to the biases $\mathbf{b}$. The bias term is added element-wise to each row of the output. Thus, the gradient of the loss with respect to $\mathbf{b}$ is the sum of the gradients over all samples in the batch:

$$\frac{\partial L}{\partial \mathbf{b}} = \sum_{i=1}^{N} \frac{\partial L}{\partial \mathbf{z}_i}.$$

This is simply the sum of the upstream gradient over the batch dimension:

$$\text{db} = \sum_{i=1}^{N} \text{dout}_i.$$

In practice, this is equivalent to computing:

$$\text{db} = \text{dout}.sum(\text{axis} = 0).$$

1.3 Final Backward Pass Expressions

To summarize, the backward pass for the affine layer computes:

- **Gradient with respect to the input x**:

$$\text{dx} = \left(\text{dout} \cdot W^T\right).reshape(\mathbf{x}.\text{shape}).$$

- **Gradient with respect to the weights** W:

$$\text{dw} = \mathbf{x}.reshape(N, D)^T \cdot \text{dout}.$$

- **Gradient with respect to the biases b**:

$$\text{db} = \text{dout}.sum(\text{axis} = 0).$$

2 ReLU Layer Forward and Backward Pass Derivation

2.1 Forward Pass

The ReLU (Rectified Linear Unit) is a common activation function that applies the non-linearity element-wise to its input. Given the input $\mathbf{x}$ of any shape, the ReLU function is defined as:

$$\text{ReLU}(x) = \max(0, x).$$

2.1.1 Steps for the Forward Pass

The forward pass computes the element-wise ReLU operation on the input:

$$\text{out}_i = \max(0, x_i),$$

for each element x_i in the input $\mathbf{x}$.

Thus, the forward pass output is:

$$\text{out} = \max(0, \mathbf{x}).$$

We also store the input $\mathbf{x}$ in the cache for use during the backward pass.

2.2 Backward Pass Derivation

In the backward pass, we want to compute the gradient of the loss L with respect to the input $\mathbf{x}$, given the upstream gradient $\frac{\partial L}{\partial \text{out}} = \text{dout}$.

2.2.1 1. Gradient with respect to the input x

From the forward pass, we know that:

$$\text{out}_i = \max(0, x_i).$$

The ReLU function is piecewise-defined:

$$\text{ReLU}(x) = \begin{cases} x, & \text{if } x > 0, \\ 0, & \text{if } x \leq 0. \end{cases}$$

To compute the gradient $\frac{\partial L}{\partial \mathbf{x}}$, we apply the chain rule. Since ReLU has no effect on negative values (outputs zero for $x \leq 0$), the gradient will propagate only through the elements where $x > 0$. For $x_i > 0$, $\frac{\partial \text{ReLU}(x_i)}{\partial x_i} = 1$, and for $x_i \leq 0$, $\frac{\partial \text{ReLU}(x_i)}{\partial x_i} = 0$.

Hence, the gradient with respect to the input $\mathbf{x}$ is:

$$\frac{\partial L}{\partial \mathbf{x}_i} = \begin{cases} \text{dout}_i, & \text{if } x_i > 0, \\ 0, & \text{if } x_i \leq 0. \end{cases}$$

This can be efficiently computed using element-wise multiplication with a mask that indicates where $x > 0$. In practice, this is implemented as:

$$\text{dx} = \text{dout} \cdot \mathbb{I}(x > 0),$$

where $\mathbb{I}(x > 0)$ is an indicator function that is 1 where $x > 0$ and 0 otherwise. In numpy, this is done as:

$$\text{dx} = \text{dout} \cdot \text{np.where}(x > 0, 1, 0)$$

2.3 Final Backward Pass Expression

To summarize, the backward pass for the ReLU layer computes:

$$\text{dx} = \text{dout} \cdot \mathbb{I}(x > 0).$$

This equation computes the gradient for each element of the input $\mathbf{x}$ by multiplying the upstream gradient dout by 1 for elements where $x > 0$ and by 0 for elements where $x \leq 0$.

3 Batch Normalization Forward and Backward Pass Derivation

3.1 Forward Pass

Batch normalization normalizes the input across a mini-batch to have a mean of zero and a variance of one, and then scales and shifts the normalized values using learnable parameters γ (scale) and β (shift). It helps in improving convergence during training by mitigating issues like covariate shift.

Given input $\mathbf{x} \in \mathbb{R}^{N \times D}$ where N is the batch size and D is the dimensionality of each input, the forward pass of batch normalization is computed as follows:

3.1.1 1. Compute the mean and variance

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i$$

$$\text{Var}(x) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2$$

3.1.2 2. Normalize the input

$$x_{norm} = \frac{x - \mu}{\sqrt{\text{Var}(x) + \epsilon}}$$

where ϵ is a small constant added for numerical stability.

3.1.3 3. Scale and shift using γ and β

$$\text{out} = \gamma \cdot x_{norm} + \beta$$

The output is the normalized input scaled by γ and shifted by β.

3.2 Backward Pass Derivation

To compute the backward pass, we aim to calculate the gradients of the loss L with respect to the input x, γ, and β given the upstream gradient $\frac{\partial L}{\partial \text{out}} = \text{dout}$.

Step-by-Step Derivation of the Backward Pass

3.2.1 1. Gradient with respect to β and γ

The gradients with respect to the shift parameter β and the scale parameter γ are straightforward:

$$\frac{\partial L}{\partial \beta} = \frac{\partial L}{\partial out_i} \cdot \frac{\partial out_i}{\partial \beta} = \sum_{i=1}^{N} \frac{\partial L}{\partial \text{out}_i} = \sum_{i=1}^{N} \text{dout}_i$$

$$\frac{\partial L}{\partial \gamma} = \frac{\partial L}{\partial out_i} \cdot \frac{\partial out_i}{\partial \gamma} = \sum_{i=1}^{N} \frac{\partial L}{\partial \text{out}_i} \cdot x_{norm,i} = \sum_{i=1}^{N} \text{dout}_i \cdot x_{norm,i}$$

3.2.2 2. Gradient with respect to x

To compute the gradient with respect to the input x, we use the chain rule. The forward pass involves several intermediate steps (mean subtraction, normalization, scaling, and shifting), so we propagate the gradients through each step.

1. **Gradient of the output with respect to the normalized input**:

$$\frac{\partial L}{\partial x_{norm,i}} = \frac{\partial L}{\partial out_i} \cdot \frac{\partial out_i}{\partial x_{norm,i}} = \text{dout}_i \cdot \gamma$$

Let dx_{norm} denote the gradient of the loss with respect to the normalized input:

$$\mathrm{dx}_{norm} = \mathrm{dout} \cdot \gamma$$

2. **Gradient with respect to the variance**: The variance affects the normalized input x_{norm} through the standard deviation:

$$\mathrm{std} = \sqrt{\mathrm{Var}(x) + \epsilon}$$

The gradient with respect to the variance is:

$$\frac{\partial L}{\partial \mathrm{Var}(x)} = \sum_{i=1}^{N} \frac{\partial L}{\partial x_{norm,i}} \cdot \frac{-0.5(x_i - \mu)}{(\mathrm{Var}(x) + \epsilon)^{3/2}}$$

Simplifying:

$$\mathrm{dvar} = \sum_{i=1}^{N} \mathrm{dx}_{norm,i} \cdot \frac{-(x_i - \mu)}{2(\mathrm{std})^3}$$

3. **Gradient with respect to the mean**: The mean affects both the normalized input and the variance:

$$\frac{\partial f}{\partial \mu} = \frac{\partial f}{\partial \hat{x}_i} \cdot \frac{\partial \hat{x}_i}{\partial \mu} + \frac{\partial f}{\partial \sigma^2} \cdot \frac{\partial \sigma^2}{\partial \mu}$$

From

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

and

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu)^2$$

$$\frac{\partial L}{\partial \mu} = \sum_{i=1}^{N} \frac{\partial L}{\partial x_{norm,i}} \cdot \frac{-1}{\mathrm{std}} + \frac{\partial L}{\partial \mathrm{Var}(x)} \cdot \frac{-2}{N} \sum_{i=1}^{N} (x_i - \mu)$$

Thus:

$$\mathrm{dmu} = \sum_{i=1}^{N} \mathrm{dx}_{norm,i} \cdot \frac{-1}{\mathrm{std}} + \mathrm{dvar} \cdot \frac{-2}{N} \sum_{i=1}^{N} (x_i - \mu)$$

4. **Gradient with respect to the input x**: Finally, the gradient with respect to the input x is:

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial x_{norm,i}} \cdot \frac{1}{\mathrm{std}} + \frac{\partial L}{\partial \mathrm{Var}(x)} \cdot \frac{2(x_i - \mu)}{N} + \frac{\partial L}{\partial \mu} \cdot \frac{1}{N}$$

Simplifying:

$$dx = \frac{1}{N \cdot \mathrm{std}} \left(N \cdot \mathrm{dx}_{norm} - \sum_{i=1}^{N} \mathrm{dx}_{norm} - x_{norm} \cdot \sum_{i=1}^{N} (\mathrm{dx}_{norm} \cdot x_{norm}) \right)$$

3.3 Final Backward Pass Equations

The final gradients are:

$$\frac{\partial L}{\partial x} = \frac{1}{N \cdot \mathrm{std}} \left(N \cdot \mathrm{dx}_{norm} - \sum \mathrm{dx}_{norm} - x_{norm} \cdot \sum (\mathrm{dx}_{norm} \cdot x_{norm}) \right)$$

$$\frac{\partial L}{\partial \gamma} = \sum_{i=1}^{N} \mathrm{dout}_i \cdot x_{norm,i}$$

$$\frac{\partial L}{\partial \beta} = \sum_{i=1}^{N} \mathrm{dout}_i$$

4 Layer Code Solutions

4.1 Affine Layer

4.1.1 Forward Pass

```
out = np.reshape(x, (x.shape[0], -1))
out = out.dot(w) + b
```

4.1.2 Backward Pass

```
dx = dout.dot(w.T).reshape(x.shape)
dw = x.reshape(x.shape[0], -1).T.dot(dout)
db = dout.sum(axis=0)
```

4.2 ReLU Layer

4.2.1 Forward Pass

```
out = np.maximum(x, 0)
```

4.2.2 Backward Pass

```
dx = dout * np.where(x > 0, 1, 0)
```