Deep Learning

April 10, 2025

1 Metrics

1.1 Precision

Precision, also known as Positive Predictive Value, measures the accuracy of the positive predictions made by the model.

$$\text{Precision} = \frac{TP}{TP + FP}$$

where: - TP (True Positives) is the number of correct positive predictions. - FP (False Positives) is the number of incorrect positive predictions.

1.2 Recall

Recall, also known as Sensitivity, True Positive Rate, or Hit Rate, measures the ability of the model to identify all relevant instances of the positive class.

$$\text{Recall} = \frac{TP}{TP + FN}$$

where: - TP (True Positives) is the number of correct positive predictions. - FN (False Negatives) is the number of incorrect negative predictions (actual positives that were not predicted as such).

1.3 F1 Score

The F1 Score is the harmonic mean of precision and recall.

$$F1 \text{ Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

1.4 Accuracy

Accuracy measures the overall correctness of the model.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

where: - TN (True Negatives) is the number of correct negative predictions.

1.5 Specificity

Specificity, also known as True Negative Rate, measures the ability of the model to identify all negative instances.

$$\text{Specificity} = \frac{TN}{TN + FP}$$

1.6 Negative Predictive Value (NPV)

NPV measures the accuracy of negative predictions.

$$\text{NPV} = \frac{TN}{TN + FN}$$

1.7 False Positive Rate (FPR)

FPR measures the proportion of actual negatives that are incorrectly identified as positive.

$$\text{FPR} = \frac{FP}{FP + TN}$$

1.8 False Negative Rate (FNR)

FNR measures the proportion of actual positives that are incorrectly identified as negative.

$$\text{FNR} = \frac{FN}{FN + TP}$$

1.9 Matthews Correlation Coefficient (MCC)

MCC is a correlation coefficient between the observed and predicted binary classifications.

$$\text{MCC} = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

1.10 Balanced Accuracy

Balanced accuracy adjusts for imbalanced classes by taking the average of recall obtained on each class.

$$\text{Balanced Accuracy} = \frac{\text{Sensitivity} + \text{Specificity}}{2}$$

1.11 Mean Absolute Error (MAE)

$$L_{\text{MAE}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{1}$$

Derivative:

$$\frac{\partial L_{\text{MAE}}}{\partial \hat{y}_i} = \begin{cases} \frac{1}{n} & \text{if } \hat{y}_i < y_i \\ -\frac{1}{n} & \text{if } \hat{y}_i > y_i \end{cases} \tag{2}$$

1.12 Mean Squared Error (MSE)

$$L_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{3}$$

Derivative:

$$\frac{\partial L_{\text{MSE}}}{\partial \hat{y}_i} = -\frac{2}{n}(y_i - \hat{y}_i) \tag{4}$$

1.13 Binary Cross-Entropy (BCE)

$$L_{\text{BCE}}(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \tag{5}$$

Derivative:

$$\frac{\partial L_{\text{BCE}}}{\partial \hat{y}_i} = -\frac{1}{n} \left(\frac{y_i}{\hat{y}_i} - \frac{1 - y_i}{1 - \hat{y}_i} \right) \tag{6}$$

1.14 Cross-Entropy (CE)

$$L_{\text{CE}}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{n} y_i \log(\hat{y}_i) \tag{7}$$

Derivative:

$$\frac{\partial L_{\text{CE}}}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i} \tag{8}$$

1.15 Kullback-Leibler Divergence (KL Divergence)

$$L_{\text{KL}}(\mathbf{p}, \mathbf{q}) = \sum_{i=1}^{n} p_i \log \left(\frac{p_i}{q_i} \right) \tag{9}$$

Derivative:

$$\frac{\partial L_{\text{KL}}}{\partial q_i} = -\frac{p_i}{q_i} \tag{10}$$

1.16 Negative Log-Likelihood Loss (NLLLoss)

$$L_{\text{NLL}}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{n} \log(\hat{y}_{i,y_i}) \tag{11}$$

Derivative:

$$\frac{\partial L_{\text{NLL}}}{\partial \hat{y}_{i,y_i}} = -\frac{1}{\hat{y}_{i,y_i}} \tag{12}$$

1.17 Huber Loss

$$L_{\text{Huber}}(\mathbf{y}, \hat{\mathbf{y}}) = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{if } |y_i - \hat{y}_i| \leq \delta \\ \delta(|y_i - \hat{y}_i| - \frac{1}{2}\delta) & \text{otherwise} \end{cases} \tag{13}$$

Derivative:

$$\frac{\partial L_{\text{Huber}}}{\partial \hat{y}_i} = \begin{cases} -(y_i - \hat{y}_i) & \text{if } |y_i - \hat{y}_i| \leq \delta \\ -\delta \, \text{sign}(y_i - \hat{y}_i) & \text{otherwise} \end{cases} \tag{14}$$

1.18 Hinge Loss

$$L_{\text{Hinge}}(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{i=1}^{n} \max(0, 1 - y_i \hat{y}_i) \tag{15}$$

Derivative:

$$\frac{\partial L_{\text{Hinge}}}{\partial \hat{y}_i} = \begin{cases} 0 & \text{if } y_i \hat{y}_i \geq 1 \\ -y_i & \text{if } y_i \hat{y}_i < 1 \end{cases} \tag{16}$$

2 Variations of ReLU

2.1 ReLU (Rectified Linear Unit)

The ReLU function is defined as:

$$\text{ReLU}(x) = \max(0, x) \tag{17}$$

Its first-order derivative is:

$$\text{ReLU}'(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{18}$$

2.2 Leaky ReLU

The Leaky ReLU function introduces a small slope for negative values:

$$\text{LeakyReLU}(x) = \begin{cases} \alpha x & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{19}$$

Its first-order derivative is:

$$\text{LeakyReLU}'(x) = \begin{cases} \alpha & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{20}$$

where α is a small constant.

2.3 Parametric ReLU (PReLU)

The PReLU function is similar to Leaky ReLU but allows the slope to be learned:

$$\text{PReLU}(x) = \begin{cases} ax & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{21}$$

Its first-order derivative is:

$$\text{PReLU}'(x) = \begin{cases} a & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{22}$$

where a is a parameter learned during training.

2.4 ELU (Exponential Linear Unit)

The ELU function is defined as:

$$\text{ELU}(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{23}$$

Its first-order derivative is:

$$\text{ELU}'(x) = \begin{cases} \alpha e^x & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{24}$$

where α is a positive constant.

2.5 RReLU (Randomized Leaky ReLU)

The RReLU function introduces a randomized slope for negative values during training:

$$\text{RReLU}(x) = \begin{cases} \alpha x & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{25}$$

Its first-order derivative is:

$$\text{RReLU}'(x) = \begin{cases} \alpha & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{26}$$

where α is a random value in a given range during training and a fixed value during testing.

2.6 SELU (Scaled Exponential Linear Unit)

The SELU function is defined as:

$$\text{SELU}(x) = \lambda \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{27}$$

Its first-order derivative is:

$$\text{SELU}'(x) = \lambda \begin{cases} \alpha e^x & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{28}$$

where α and λ are fixed parameters.

2.7 GELU (Gaussian Error Linear Unit)

The GELU function is defined as:

$$\text{GELU}(x) = x\Phi(x) \tag{29}$$

where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution. A common approximation is:

$$\text{GELU}(x) \approx 0.5x \left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right)\right) \tag{30}$$

Its first-order derivative is more complex and often approximated in practice.

2.8 SiLU (Sigmoid Linear Unit or Swish)

The SiLU function is defined as:

$$\text{SiLU}(x) = x\sigma(x) \tag{31}$$

where $\sigma(x)$ is the sigmoid function. Its first-order derivative is:

$$\text{SiLU}'(x) = \sigma(x) + x\sigma(x)(1 - \sigma(x)) \tag{32}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$.

2.9 GLU (Gated Linear Unit)

The GLU function is defined as:

$$\text{GLU}(a, b) = a \otimes \sigma(b) \tag{33}$$

where a and b are inputs, σ is the sigmoid function, and $\otimes$ denotes element-wise multiplication. The first-order derivative is more complex and depends on the context in which it is used.

3 Optimizers

3.1 Momentum

Momentum helps accelerate gradient vectors in the right directions, thus leading to faster converging.

$$v_t = \beta v_{t-1} + \eta \nabla L(\theta_t) \tag{34}$$

$$\theta_{t+1} = \theta_t - v_t \tag{35}$$

where:

- v_t is the velocity at time step t.
- β is the momentum coefficient.
- η is the learning rate.
- $\nabla L(\theta_t)$ is the gradient of the loss function at time step t.
- θ_t are the parameters at time step t.

3.2 Nesterov Momentum

Nesterov Momentum is a variant of momentum that anticipates the gradient and uses the gradient at the approximated future position.

$$v_t = \beta v_{t-1} + \eta \nabla L(\theta_t - \beta v_{t-1}) \tag{36}$$

$$\theta_{t+1} = \theta_t - v_t \tag{37}$$

where:

- v_t is the velocity at time step t.
- β is the momentum coefficient.
- η is the learning rate.
- $\nabla L(\theta_t - \beta v_{t-1})$ is the gradient of the loss function at the anticipated future position.
- θ_t are the parameters at time step t.

3.3 Polyak's Momentum

Polyak's Momentum method uses the concept of momentum to stabilize convergence and avoid oscillations.

$$v_t = \beta v_{t-1} + \eta \nabla L(\theta_t) \tag{38}$$

$$\theta_{t+1} = \theta_t - (1 + \beta) v_t \tag{39}$$

where:

- v_t is the velocity at time step t.
- β is the momentum coefficient.
- η is the learning rate.
- $\nabla L(\theta_t)$ is the gradient of the loss function at time step t.
- θ_t are the parameters at time step t.

3.4 RMSProp (Root Mean Square Propagation)

RMSProp divides the learning rate by an exponentially decaying average of squared gradients.

$$v_t = \beta v_{t-1} + (1 - \beta)(\nabla L(\theta_t))^2 \tag{40}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} \nabla L(\theta_t) \tag{41}$$

where:

- v_t is the moving average of squared gradients at time step t.
- β is the decay rate.

- η is the learning rate.
- $\nabla L(\theta_t)$ is the gradient of the loss function at time step t.
- ϵ is a small constant for numerical stability.
- θ_t are the parameters at time step t.

4 Adam (Adaptive Moment Estimation)

Adam is an adaptive learning rate optimization algorithm that combines the advantages of two other popular methods: AdaGrad and RMSProp. It uses estimates of the first and second moments of the gradients to adapt the learning rate for each parameter.

4.1 First Moment Estimate

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{42}$$

where:

- m_t is the first moment estimate (mean of gradients).
- β_1 is the exponential decay rate for the first moment estimate.
- g_t is the gradient at time step t.

4.2 Second Moment Estimate

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{43}$$

where:

- v_t is the second moment estimate (uncentered variance of gradients).
- β_2 is the exponential decay rate for the second moment estimate.

4.3 Bias Correction

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{44}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{45}$$

where:

- $\hat{m}_t$ is the bias-corrected first moment estimate.
- $\hat{v}_t$ is the bias-corrected second moment estimate.

4.4 Parameter Update

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{46}$$

where:

- θ_{t+1} is the updated parameter.
- θ_t is the current parameter.
- η is the learning rate.
- ϵ is a small constant added for numerical stability.

5 Variants of Adam

5.1 AdamW (Adam with Weight Decay)

AdamW decouples weight decay from the gradient update, improving regularization and generalization.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{47}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{48}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{49}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{50}$$

$$\theta_{t+1} = \theta_t - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \theta_t \right) \tag{51}$$

where:

- λ is the weight decay coefficient.

5.2 AdaMax

AdaMax is a variant of Adam based on the infinity norm.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{52}$$

$$u_t = \max(\beta_2 u_{t-1}, |g_t|) \tag{53}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{54}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t + \epsilon} \hat{m}_t \tag{55}$$

5.3 NAdam (Nesterov-accelerated Adaptive Moment Estimation)

NAdam incorporates Nesterov momentum into Adam.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{56}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{57}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{58}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{59}$$

$$\theta_{t+1} = \theta_t - \eta \left(\frac{\beta_1 \hat{m}_t + (1 - \beta_1) g_t}{\sqrt{\hat{v}_t} + \epsilon} \right) \tag{60}$$

5.4 RAdam (Rectified Adam)

RAdam introduces a rectification term to the Adam optimizer.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{61}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{62}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{63}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{64}$$

$$\rho_t = \rho_\infty - \frac{2t\beta_2^t}{1-\beta_2^t} \tag{65}$$

$$r_t = \sqrt{\frac{(\rho_t - 4)(\rho_t - 2)\rho_\infty}{(\rho_\infty - 4)(\rho_\infty - 2)\rho_t}} \tag{66}$$

$$\theta_{t+1} = \theta_t - \eta r_t \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{67}$$

or, if $\rho_t \leq 4$:

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{68}$$

where:

- $\rho_\infty = \frac{2}{1-\beta_2} - 1$

6 Xavier Initialization

Xavier Initialization, also known as Glorot Initialization, aims to keep the scale of the gradients roughly the same in all layers, which is particularly useful for activation functions like sigmoid and Tanh.

This ensures the variance of the output is the same as the input, assuming a Gaussian distribution with:

- Mean: $\mathbb{E}[X] = 0$, $\mathbb{E}[W] = 0$
- Variance: Remember that $\mathbb{E}[X^2] = \text{Var}[X] + \mathbb{E}[X]^2$ and if X, Y are independent:

$$\text{Var}[XY] = \mathbb{E}[X^2 Y^2] - \mathbb{E}[XY]^2 \tag{69}$$

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y] \tag{70}$$

Therefore,

$$\text{Var}(s) = \text{Var}\left(\sum_i^n w_i x_i\right) = \sum_i^n \text{Var}(w_i x_i) \tag{71}$$

$$= \sum_i^n \left[\mathbb{E}(w_i)\right]^2 \text{Var}(x_i) + \left[\mathbb{E}(x_i)\right]^2 \text{Var}(w_i) + \text{Var}(x_i)\text{Var}(w_i) \tag{72}$$

$$= \sum_i^n \text{Var}(x_i)\text{Var}(w_i) = n(\text{Var}(w)\text{Var}(x)) \tag{73}$$

We want to ensure that the variance of the output $s = xw$ is the same as the input: $\text{Var}(s) = \text{Var}(x)$.

$$\text{Var}(w) = n(\text{Var}(w)\text{Var}(x)) \rightarrow \text{Var}(w) = \frac{1}{n} \tag{74}$$

Note that n is the number of input neurons for the layer of weights you want to initialize. This n is not the number N of input data $X \in \mathbb{R}^{N \times D}$, but $n = D$.

7 Regularization terms

The L2-regularization term is added to the loss function to penalize large weights. For a given loss function L and weights W, the regularized loss function L_{reg} can be written as:

$$L_{\text{reg}} = L + \frac{\lambda}{2}\|W\|_2^2$$

where $\frac{\lambda}{2}\|W\|_2^2$ is the L2-regularization term and $\|W\|_2^2 = \sum_i W_i^2$ is the squared Euclidean norm of the weights.

Gradient Descent Update Rule with L2-Regularization

In gradient descent, the weights are updated iteratively using the gradient of the loss function. The update rule for the weights W at iteration t without regularization is:

$$W^{(t+1)} = W^{(t)} - \eta \nabla_W L$$

When we include L2-regularization, the gradient of the regularized loss function L_{reg} with respect to the weights W becomes:

$$\nabla_W L_{\text{reg}} = \nabla_W L + \lambda W$$

Therefore, the update rule with L2-regularization is:

$$W^{(t+1)} = W^{(t)} - \eta \left(\nabla_W L + \lambda W\right)$$

Interpreting the Update Rule

Let's rewrite the update rule to separate the effect of L2-regularization:

$$W^{(t+1)} = W^{(t)} - \eta \nabla_W L - \eta \lambda W^{(t)}$$

Rearrange this to highlight the weight decay effect:

$$W^{(t+1)} = W^{(t)} \left(1 - \eta\lambda\right) - \eta \nabla_W L$$

Here, we can see that the term $W^{(t)} \left(1 - \eta\lambda\right)$ represents a multiplicative factor that scales the weights down. This multiplicative factor $1 - \eta\lambda$ is always less than 1 if $\eta\lambda > 0$, causing the weights to decay at each iteration. This is why L2-regularization is referred to as "weight decay."

Summary

The term "weight decay" comes from the effect of L2-regularization on the weights during the gradient descent update. The update rule with L2-regularization:

$$W^{(t+1)} = W^{(t)} \left(1 - \eta\lambda\right) - \eta \nabla_W L$$

shows that the weights are scaled down by a factor of $1 - \eta\lambda$ at each iteration, effectively decaying the weights over time. This regularization helps to prevent overfitting by encouraging smaller weights, leading to simpler and more generalizable models.

Other Regularization terms

1. Elastic Net Regularization

$$\text{Loss} = \text{Primary Loss} + \lambda_1 \sum_{i=1}^{n} |w_i| + \frac{1}{2}\lambda_2 \sum_{i=1}^{n} w_i^2$$

$$w \leftarrow w - \eta \left(\frac{\partial \text{Primary Loss}}{\partial w} + \lambda_1 \text{sign}(w) + \lambda_2 w \right)$$

2. Tikhonov Regularization (L2 Regularization)

$$\text{Loss} = \text{Primary Loss} + \frac{1}{2}\lambda \sum_{i=1}^{n} w_i^2$$

$$w \leftarrow w - \eta \left(\frac{\partial \text{Primary Loss}}{\partial w} + \lambda w \right)$$

3. Spectral Norm Regularization

$$\text{Loss} = \text{Primary Loss} + \lambda \|W\|_2$$

$$w \leftarrow w - \eta \left(\frac{\partial \text{Primary Loss}}{\partial w} + \lambda \frac{\partial \|W\|_2}{\partial w} \right)$$

4. Orthogonality Regularization

$$\text{Loss} = \text{Primary Loss} + \lambda \|W^T W - I\|_F^2$$

$$w \leftarrow w - \eta \left(\frac{\partial \text{Primary Loss}}{\partial w} + \lambda \frac{\partial \|W^T W - I\|_F^2}{\partial w} \right)$$

5. Max-Norm Regularization

$$\text{Loss} = \text{Primary Loss} + \lambda \sum_{i=1}^{n} \max(0, \|w_i\|_2 - c)$$

$$w \leftarrow w - \eta \left(\frac{\partial \text{Primary Loss}}{\partial w} \right)$$

After updating, project the weights onto the L_2 ball of radius c:

$$w_i \leftarrow \frac{w_i}{\max(1, \|w_i\|_2 / c)}$$

8 Batch Normalization

Given a fully connected layer with an output shape of 8×16, followed by a batch normalization layer, the operation can be expressed as $y = \text{BN}(x) = \gamma * x_{\text{norm}} + \beta$.

(a) Dimensions of the Parameters γ and β

- Since batch normalization is applied per feature (i.e., per column in the output of the fully connected layer), the parameters γ and β will each have dimensions equal to the number of features.
- For the given output shape 8×16:
  - γ and β will each have a dimension of 16.
- Therefore:
  - γ has shape $(16,)$
  - β has shape $(16,)$

(b) Derivation of the Gradients of γ and β

To derive the gradients of the loss L with respect to γ and β, we start by defining the batch normalization operation more formally.

Let:

- $x \in \mathbb{R}^{8\times16}$ be the input to the batch normalization layer.
- $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$ be the mean of the input x.
- $\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$ be the variance of the input x.
- $x_{\text{norm}} = \frac{x-\mu}{\sqrt{\sigma^2+\epsilon}}$ be the normalized input.
- $y = \gamma * x_{\text{norm}} + \beta$ be the output of the batch normalization layer.

We need to find the partial derivatives $\frac{\partial L}{\partial \gamma}$ and $\frac{\partial L}{\partial \beta}$.

1. Gradient with respect to β:

The gradient of the loss L with respect to β is the sum of the gradients of the loss with respect to each output y_i:

$$\frac{\partial L}{\partial \beta_j} = \sum_{i=1}^{N} \frac{\partial L}{\partial y_{ij}} \cdot \frac{\partial y_{ij}}{\partial \beta_j}$$

Since $\frac{\partial y_{ij}}{\partial \beta_j} = 1$, we get:

$$\frac{\partial L}{\partial \beta_j} = \sum_{i=1}^{N} \frac{\partial L}{\partial y_{ij}}$$

2. Gradient with respect to γ:

The gradient of the loss L with respect to γ is the sum of the gradients of the loss with respect to each output y_i multiplied by the normalized input x_{norm}:

$$\frac{\partial L}{\partial \gamma_j} = \sum_{i=1}^{N} \frac{\partial L}{\partial y_{ij}} \cdot \frac{\partial y_{ij}}{\partial \gamma_j}$$

Since $\frac{\partial y_{ij}}{\partial \gamma_j} = x_{\text{norm}_{ij}}$, we get:

$$\frac{\partial L}{\partial \gamma_j} = \sum_{i=1}^{N} \frac{\partial L}{\partial y_{ij}} \cdot x_{\text{norm}_{ij}}$$

(c) Using NumPy to Calculate Gradients

Let's assume:

- `dL_dy` is the gradient of the loss L with respect to the output y, with shape $(8, 16)$.
- `x_norm` is the normalized input, with shape $(8, 16)$.

Using NumPy, we can calculate the gradients as follows:

```
import numpy as np

# Assuming dL_dy and x_norm are given as NumPy arrays
dL_dy = np.random.randn(8, 16)  # Example gradient of the loss with respect to y
x_norm = np.random.randn(8, 16)  # Example normalized input

# Gradient with respect to beta
dL_dbeta = np.sum(dL_dy, axis=0)

# Gradient with respect to gamma
dL_dgamma = np.sum(dL_dy * x_norm, axis=0)

# Print the results
print("Gradient with respect to beta:", dL_dbeta)
print("Gradient with respect to gamma:", dL_dgamma)
```

In this code:

- `np.sum(dL_dy, axis=0)` computes the sum of `dL_dy` along the batch dimension (axis 0) to get $\frac{\partial L}{\partial \beta}$.
- `np.sum(dL_dy * x_norm, axis=0)` computes the sum of the element-wise product of `dL_dy` and `x_norm` along the batch dimension to get $\frac{\partial L}{\partial \gamma}$.

This gives us the gradients of the loss with respect to the parameters γ and β.

8.1 Batch Normalization Backpropagation in CNNs

Batch Normalization (BN) is a technique to normalize the inputs of each layer, improving training speed and stability. Here, we demonstrate the backpropagation through BN in CNNs using a small example.

8.2 Forward Pass

Consider a mini-batch of inputs $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$ with $\mathbf{x}_i \in \mathbb{R}^D$. The steps of the forward pass through BN are:

- Compute the mini-batch mean:

$$\mu_B = \frac{1}{m} \sum_{i=1}^{m} \mathbf{x}_i \tag{75}$$

- Compute the mini-batch variance:

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{x}_i - \mu_B)^2 \tag{76}$$

- Normalize the batch:

$$\hat{\mathbf{x}}_i = \frac{\mathbf{x}_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{77}$$

- Scale and shift:

$$\mathbf{y}_i = \gamma \hat{\mathbf{x}}_i + \beta \tag{78}$$

8.3 Backward Pass

Given the gradient of the loss L with respect to the output $\frac{\partial L}{\partial \mathbf{y}_i}$, we want to compute the gradients with respect to $\mathbf{x}_i$, γ, and β.

- Gradient with respect to β:

$$\frac{\partial L}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial L}{\partial \mathbf{y}_i} \tag{79}$$

- Gradient with respect to γ:

$$\frac{\partial L}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial L}{\partial \mathbf{y}_i} \cdot \hat{\mathbf{x}}_i \tag{80}$$

- Gradient with respect to $\hat{\mathbf{x}}_i$:

$$\frac{\partial L}{\partial \hat{\mathbf{x}}_i} = \frac{\partial L}{\partial \mathbf{y}_i} \cdot \gamma \tag{81}$$

- Gradient with respect to σ_B^2:

$$\frac{\partial L}{\partial \sigma_B^2} = \sum_{i=1}^{m} \frac{\partial L}{\partial \hat{\mathbf{x}}_i} \cdot (\mathbf{x}_i - \mu_B) \cdot -\frac{1}{2}(\sigma_B^2 + \epsilon)^{-3/2} \tag{82}$$

- Gradient with respect to μ_B:

$$\frac{\partial L}{\partial \mu_B} = \sum_{i=1}^{m} \left(\frac{\partial L}{\partial \hat{\mathbf{x}}_i} \cdot -\frac{1}{\sqrt{\sigma_B^2 + \epsilon}} \right) + \frac{\partial L}{\partial \sigma_B^2} \cdot \frac{-2}{m} \sum_{i=1}^{m} (\mathbf{x}_i - \mu_B) \tag{83}$$

- Gradient with respect to $\mathbf{x}_i$:

$$\frac{\partial L}{\partial \mathbf{x}_i} = \frac{\partial L}{\partial \hat{\mathbf{x}}_i} \cdot \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial L}{\partial \sigma_B^2} \cdot \frac{2(\mathbf{x}_i - \mu_B)}{m} + \frac{\partial L}{\partial \mu_B} \cdot \frac{1}{m} \tag{84}$$

8.4 Example Image

Below is a schematic example of the process:

(input) at (0, 0) [draw, rectangle, minimum width=1cm, minimum height=1cm] Input;
(mean) at (2, 1) [draw, rectangle, minimum width=1.5cm, minimum height=1cm] Mean; (variance) at (2, -1) [draw, rectangle, minimum width=1.5cm, minimum height=1cm] Variance;
(normalize) at (4, 0) [draw, rectangle, minimum width=2cm, minimum height=1cm] Normalize;
(scale) at (6, 0.5) [draw, rectangle, minimum width=1.5cm, minimum height=1cm] Scale; (shift) at (6, -0.5) [draw, rectangle, minimum width=1.5cm, minimum height=1cm] Shift;
(output) at (8, 0) [draw, rectangle, minimum width=1cm, minimum height=1cm] Output;
[-¿] (input) – (mean); [-¿] (input) – (variance); [-¿] (mean) – (normalize); [-¿] (variance) – (normalize); [-¿] (normalize) – (scale); [-¿] (normalize) – (shift); [-¿] (scale) – (output); [-¿] (shift) – (output);

9 Generative Adversarial Networks (GANs)

9.1 Original GAN Loss Functions

9.1.1 Discriminator Loss

$$L_D = -\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}[\log D(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))] \tag{85}$$

9.1.2 Generator Loss

$$L_G = -\mathbb{E}_{\mathbf{z}\sim p_\mathbf{z}}[\log D(G(\mathbf{z}))] \tag{86}$$

9.2 Non-Saturating GAN Loss

$$L_G = \mathbb{E}_{\mathbf{z}\sim p_\mathbf{z}}[\log(1 - D(G(\mathbf{z})))] \tag{87}$$

9.3 Least Squares GAN (LSGAN)

9.3.1 Discriminator Loss

$$L_D = \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[(D(\mathbf{x}) - 1)^2] + \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p_\mathbf{z}}[D(G(\mathbf{z}))^2] \tag{88}$$

9.3.2 Generator Loss

$$L_G = \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p_\mathbf{z}}[(D(G(\mathbf{z})) - 1)^2] \tag{89}$$

9.4 Wasserstein GAN (WGAN)

9.4.1 Critic (Discriminator) Loss

$$L_D = -\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_\mathbf{z}}[D(G(\mathbf{z}))] \tag{90}$$

9.4.2 Generator Loss

$$L_G = -\mathbb{E}_{\mathbf{z}\sim p_\mathbf{z}}[D(G(\mathbf{z}))] \tag{91}$$

9.5 Wasserstein GAN with Gradient Penalty (WGAN-GP)

9.5.1 Critic (Discriminator) Loss with Gradient Penalty

$$\begin{aligned} L_D = &- \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_\mathbf{z}}[D(G(\mathbf{z}))] \\ &+ \lambda \mathbb{E}_{\hat{\mathbf{x}}\sim p_{\hat{\mathbf{x}}}}\left[(||\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})||_2 - 1)^2\right] \end{aligned} \tag{92}$$

10 RNNs

$$h_t = \text{ReLU}(W_h h_{t-1} + W_x x_t) \tag{93}$$

$$\frac{\partial h_t}{\partial W_h} = \frac{\partial \text{A}(z_t)}{\partial z_t} \cdot \left(W_h \cdot \frac{\partial h_{t-1}}{\partial W_h} + h_{t-1}\right) \tag{94}$$

$$\frac{\partial h_t}{\partial W_x} = \frac{\partial \text{A}(z_t)}{\partial z_t} \cdot \left(W_h \cdot \frac{\partial h_{t-1}}{\partial W_x} + x_t\right) \tag{95}$$

$$\frac{\partial h_t}{\partial x_\tau} = \frac{\partial \text{A}(z_t)}{\partial z_t} \cdot \left(W_h \cdot \frac{\partial h_{t-1}}{\partial x_\tau} + W_x \cdot \delta_{\tau t}\right) \tag{96}$$

11 GRUs

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \tag{97}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \tag{98}$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1})) \tag{99}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{100}$$

12 LSTMs

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{101}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{102}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{103}$$

$$\tilde{C}_t = \tanh(W_C x_t + U_C h_{t-1} + b_C) \tag{104}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{105}$$

$$h_t = o_t \odot \tanh(C_t) \tag{106}$$

Exploding and Vanishing Gradients in RNNs

Recurrence Relation

For a standard RNN, the hidden state h_t at time t is given by:

$$h_t = \phi(W_h h_{t-1} + W_x x_t) \tag{107}$$

where ϕ is the activation function, typically tanh or ReLU. For simplicity, we consider ϕ to be the identity function:

$$h_t = W_h h_{t-1} + W_x x_t \tag{108}$$

Backpropagation Through Time (BPTT)

During BPTT, the gradient of the loss L with respect to the weight matrix W_h is computed by unrolling the RNN through time:

$$\frac{\partial L}{\partial W_h} = \sum_{t=1}^{T} \frac{\partial L}{\partial h_t} \frac{\partial h_t}{\partial W_h} \tag{109}$$

To understand the gradient flow, we need to look at the partial derivative of the hidden state with respect to the previous hidden state:

$$\frac{\partial h_t}{\partial h_{t-1}} = W_h \tag{110}$$

So, the gradient with respect to h_{t-1} can be written as:

$$\frac{\partial L}{\partial h_{t-1}} = \frac{\partial L}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial L}{\partial h_t} W_h \tag{111}$$

By recursively applying this through time, we get:

$$\frac{\partial L}{\partial h_0} = \left(\prod_{t=1}^{T} W_h \right) \frac{\partial L}{\partial h_T} \tag{112}$$

Eigenvalues and Gradient Behavior

The behavior of the gradients as they propagate backward through time is governed by the eigenvalues of the weight matrix W_h:

Exploding Gradients

If the eigenvalues λ_i of W_h are greater than 1, the gradients will grow exponentially as they propagate backward through time:

$$\left| \frac{\partial L}{\partial h_0} \right| = \left| \left(\prod_{t=1}^{T} \lambda_i \right) \frac{\partial L}{\partial h_T} \right| = \left| \lambda_i^T \frac{\partial L}{\partial h_T} \right| \tag{113}$$

Since $|\lambda_i| > 1$, $|\lambda_i^T|$ increases exponentially with T, causing the gradients to explode.

Vanishing Gradients

If the eigenvalues λ_i of W_h are less than 1, the gradients will decay exponentially as they propagate backward through time:

$$\left| \frac{\partial L}{\partial h_0} \right| = \left| \left(\prod_{t=1}^{T} \lambda_i \right) \frac{\partial L}{\partial h_T} \right| = \left| \lambda_i^T \frac{\partial L}{\partial h_T} \right| \tag{114}$$

Since $|\lambda_i| < 1$, $|\lambda_i^T|$ decreases exponentially with T, causing the gradients to vanish.

Mathematical Summary

Exploding Gradients

When $|\lambda_i| > 1$:

$$\left| \frac{\partial L}{\partial h_t} \right| = \left| \lambda_i^t \frac{\partial L}{\partial h_T} \right| \tag{115}$$

As t increases, $|\lambda_i^t|$ grows exponentially, leading to exploding gradients.

Vanishing Gradients

When $|\lambda_i| < 1$:

$$\left| \frac{\partial L}{\partial h_t} \right| = \left| \lambda_i^t \frac{\partial L}{\partial h_T} \right| \tag{116}$$

As t increases, $|\lambda_i^t|$ decays exponentially, leading to vanishing gradients.