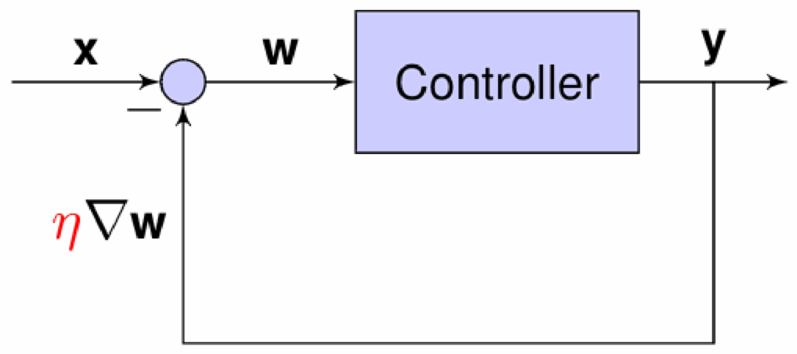
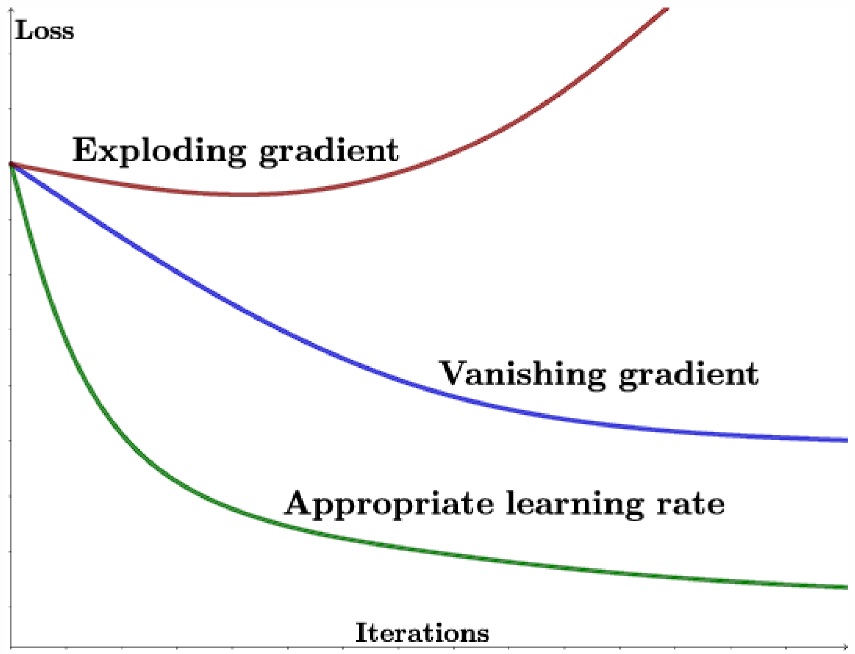
Vanishing and Exploding Gradients Problem

Vanishing and Exploding Gradients



Analogy to control theory



- If η is to high $\mapsto$ **positive feedback** $\mapsto$ loss grows **without bounds**
- If η is to small $\mapsto$ **negative feedback** $\mapsto$ **gradient vanishes**
- Choice of η is **critical** for learning

Vanishing and Exploding Gradients

Vanishing and exploding gradients are **common problems** encountered during the training of deep neural networks. These issues primarily arise **during back-propagation**, where gradients are used to update the weights of the network.

- Vanishing Gradients
- Exploding Gradients

Note: Backpropagation is the process by which neural networks learn. It involves **calculating the gradient of the loss function concerning each weight by the chain rule**, propagating these gradients backward through the network.

Vanishing Gradients

The vanishing gradient problem occurs when the **gradients of the loss function** concerning the weights become **very small**. This can slow down the learning process significantly, as the **weight updates become negligible**.

- **Activation Functions:** Non-linear activation functions like the **Sigmoid** or **Tanh** can squash input values to a very small range (0 to 1 for Sigmoid and -1 to 1 for Tanh). The input values into a small range, leading to small gradients.
- **Deep Networks:** The problem is exacerbated in deep networks, where the chain rule is applied many times. Even small gradients can multiply to become extremely small.
- **Weight Initialization:** Poor initialization can exacerbate the problem, but it is not the root cause.

Consequences of Vanishing Gradients

- The network learns **very slowly** or **stops** learning altogether because the weights are not updated effectively.
- Lower layers (**closer to the input**) learn very slowly because they **receive much smaller gradient signals.**

Practical Mitigations of Vanishing Gradients

- **ReLU Activation Function:** The Rectified Linear Unit (ReLU) activation function does not suffer from vanishing gradients for positive inputs since its derivative is either 0 or 1. Note: Sigmoid and Tanh activation functions are often associated with the vanishing gradient problem, which can hinder the training of deep neural networks.

- Weight Initialization: Techniques like **He** or **Xavier** initialization help maintain a balance in the gradients.

- Batch Normalization: This normalizes the input of each layer, helping maintain a stable distribution of activations and gradients.

Characteristics of Sigmoid Activation Function

Characteristics:

- The Sigmoid function squashes its input into the range (0, 1).
- The output of the Sigmoid function is given by:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Characteristics of Sigmoid Activation Function

- When the input x is very large or very small, the derivative of the Sigmoid function becomes very close to zero.
- The derivative of the Sigmoid function is:

$$f'(x) = f(x)(1 - f(x))$$

Since $f(x)$ approaches 0 or 1 for very large or very small x, $f'(x)$ approaches 0.

- During back-propagation, gradients are multiplied by these small derivatives, causing them to shrink exponentially and leading to very small updates to the weights in the earlier layers of the network.

Characteristics of Tanh Activation Function

Characteristics:

- The Tanh function squashes its input into the range (-1, 1).
- The output of the Tanh function is given by:

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Characteristics of Tanh Activation Function

- Similar to the Sigmoid function, when the input x is very large or very small, the derivative of the Tanh function becomes very close to zero.
- The derivative of the Tanh function is:

$$f'(x) = 1 - \tanh^2(x)$$

Since $\tanh(x)$ approaches 1 or -1 for very large or very small x, $f'(x)$ approaches 0.

- This causes the gradients to vanish, leading to small updates to the weights during back-propagation.

Why Sigmoid and Tanh Might Still Be Used?

Why Sigmoid and Tanh Might Still Be Used

Despite the vanishing gradient problem, there are scenarios where Sigmoid and Tanh functions might still be used:

- **Output Layers:** Sigmoid is often used in the output layer for binary classification problems because it outputs a probability between 0 and 1.

- **LSTM Networks:** In LSTM (Long Short-Term Memory) networks, the Sigmoid function is used within gates to regulate the flow of information.

Alternatives to Mitigate Vanishing Gradients

To address the vanishing gradient problem, alternative activation functions are often used:

1. **ReLU (Rectified Linear Unit)**:
   - The ReLU function is defined as $f(x) = \max(0, x)$.
   - It does not saturate in the positive domain, and its derivative is 1 for positive x, which helps maintain stronger gradients during back-propagation.
2. **Leaky ReLU**:
   - A variant of ReLU that allows a small, non-zero gradient when x is negative.
   - Defined as $f(x) = \max(\alpha x, x)$, where α is a small constant.

Alternatives to Mitigate Vanishing Gradients

To address the vanishing gradient problem, alternative activation functions are often used:

3. **Parametric ReLU (PReLU):**
    - Similar to Leaky ReLU but with α being a learnable parameter.
4. **ELU (Exponential Linear Unit):**
    - Combines benefits of ReLU and mitigates some of its downsides by allowing outputs to be negative and smoothing the transition.

Practical Mitigations of Vanishing Gradients

- ReLU Activation Function: The Rectified Linear Unit (ReLU) activation function does not suffer from vanishing gradients for positive inputs since its derivative is either 0 or 1. Note: Sigmoid and Tanh activation functions are often associated with the vanishing gradient problem, which can hinder the training of deep neural networks.

- **Weight Initialization:** Techniques like **He** or **Xavier** initialization help maintain a balance in the gradients.

- Batch Normalization: This normalizes the input of each layer, helping maintain a stable distribution of activations and gradients.

`He` & `Xavier` Initialization

- **He Initialization:** Best for ReLU and its variants. It uses a higher variance (2/n) to fulfill the fact that ReLU activation only allows positive values.

- **Xavier Initialization:** Suitable for sigmoid and tanh activations. It balances the variance between the input and output, preventing the gradients from vanishing or exploding.

Exploding Gradients Problem

Vanishing and Exploding Gradients

Vanishing and exploding gradients are **common problems** encountered during the training of deep neural networks. These issues primarily arise **during back-propagation**, where gradients are used to update the weights of the network.

- ❖ Vanishing Gradients
- ❖ Exploding Gradients

Note: Backpropagation is the process by which neural networks learn. It involves **calculating the gradient of the loss function concerning each weight by the chain rule**, propagating these gradients backward through the network.

Exploding Gradients Problem

The exploding gradient problem occurs when the gradients **grow exponentially** during back-propagation. This can lead to very large weight updates, causing the network to become unstable and diverge.

- **Initialization:** Poor initialization of weights can contribute to exploding gradients. If weights are initialized with large values, it can cause large activations, which in turn cause large gradients.

- **Deep Networks:** Similar to the vanishing gradient problem, the chain rule applied over many layers can cause gradients to grow exponentially.

- **Consequences:**
  - The network weights can take very large values, causing numerical instability.
  - The learning process can become unstable, and the model might not converge.

Mitigation Strategies

- **Gradient Clipping:** Clip gradients during back-propagation to a maximum threshold to prevent them from growing too large.
- **Initialization:** Proper weight initialization can also help mitigate exploding gradients.
- **Use of Optimizers:** Advanced optimization algorithms like **RMSprop**, **Adam**, or **Adagrad** can adapt learning rates dynamically to help manage exploding gradients.

Gradient Clipping

Gradient clipping is a technique used to address the problem of **exploding gradients**, which can occur during the training of deep neural networks. Exploding gradients can cause numerical instability and interrupt the model’s convergence. Common threshold values range between 0.1 and 5, depending on the model architecture and the nature of the data. In RNNs or LSTMs, a common threshold is around 1.0 to 5.0. However, these values are not universal, and different models might require different thresholds.

Reasons to use gradient clipping-

- ❖ **Unstable Training:** Substantial gradients can cause the weights to be updated in very large steps, leading to unstable training and divergence.

- ❖ **Numerical Issues:** Large gradients can cause numerical overflow or underflow, leading to NaNs (Not a Number) or infinite values in the weights.

Norm Clipping

Gradient clipping mitigates these issues by limiting the size of the gradients during back-propagation.

1. **Norm Clipping**:

   - This method clips the gradient if its norm exceeds a predefined threshold.
   - For a gradient vector $\mathbf{g}$, if $||\mathbf{g}|| > \text{threshold}$, then the gradient is scaled down:

$$\mathbf{g} \leftarrow \frac{\mathbf{g}}{||\mathbf{g}||} \times \text{threshold}$$

   - This ensures that the gradient's norm does not exceed the threshold, preventing excessively large updates to the weights.

Norm Clipping

Example of Norm Clipping:

Let's say we have a threshold value of 5. During back-propagation, if the gradient vector $\mathbf{g}$ has a norm greater than 5, we scale it down:

1. Compute the norm of the gradient: $||\mathbf{g}||$.
2. If $||\mathbf{g}|| > 5$:

$$\mathbf{g} \leftarrow \frac{\mathbf{g}}{||\mathbf{g}||} \times 5$$

Note: If ||g||<5, the norm clipping mechanism does not alter the gradient vector. Norm clipping only scales down the gradient vector when its norm exceeds the specified threshold (in this case, 5). If the norm of the gradient vector is **already less than the threshold**, **no scaling is necessary**.

Norm Clipping

Let's compute the result of gradient clipping if you have a gradient norm value of 27 and you're using a clipping threshold = 5.

How Gradient Clipping Works

The formula for clipping a gradient vector g to a maximum norm τ is:

$$g_{\text{clipped}} = g \times \frac{\tau}{||g||}$$

Given Values

- Current gradient norm: $||g|| = 27$
- Clipping threshold: $\tau = 5$

Applying the Formula

1. **Calculate the scaling factor**:

$$\text{scaling factor} = \frac{\tau}{||g||} = \frac{5}{27} \approx 0.185$$

2. **Scale the gradient**:
   - Multiply the original gradient g by the scaling factor of **0.185**.
   - This scales down the gradient so that its new norm is exactly $\tau = 5$.

Value Clipping

Gradient clipping mitigates these issues by limiting the size of the gradients during back-propagation.

2. **Value Clipping**:

- This method clips each component of the gradient independently.
- For each component g_i of the gradient vector $\mathbf{g}$, if $g_i > \text{max\_value}$ or $g_i < \text{min\_value}$, then:

$$g_i \leftarrow \max(\min(g_i, \text{max\_value}), \text{min\_value})$$

- This ensures that each individual gradient component stays within a specified range.

Value Clipping

Example of Value Clipping:

- A gradient vector $\mathbf{g} = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix}$
- $\text{max\_value} = 5$
- $\text{min\_value} = -5$

Initial Gradient Vector

Let's assume the initial gradient vector is:

$$\mathbf{g} = \begin{bmatrix} 7 \\ -6 \\ 3 \end{bmatrix}$$

The formula for value clipping is:

$$g_i \leftarrow \max(\min(g_i, \text{max\_value}), \text{min\_value})$$

Value Clipping

We will apply the clipping formula to each component of the gradient vector $\mathbf{g}$:

1. For $g_1 = 7$:

$$g'_1 = \max(\min(7,5),-5) = \max(5,-5) = 5$$

2. For $g_2 = -6$:

$$g'_2 = \max(\min(-6,5),-5) = \max(-6,-5) = -5$$

3. For $g_3 = 3$:

$$g'_3 = \max(\min(3,5),-5) = \max(3,-5) = 3$$

Value Clipping

Initial Gradient Vector

Let's assume the initial gradient vector is:

$$\mathbf{g} = \begin{bmatrix} 7 \\ -6 \\ 3 \end{bmatrix}$$

New Gradient Vector

After applying value clipping, the new gradient vector $\mathbf{g}'$ is:

$$\mathbf{g}' = \begin{bmatrix} 5 \\ -5 \\ 3 \end{bmatrix}$$

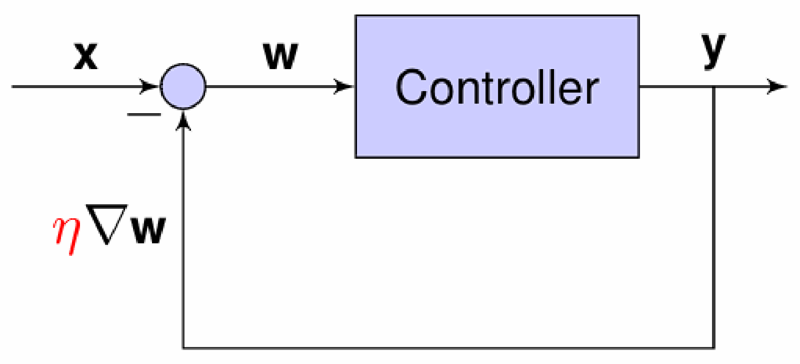
When to Use Gradient Clipping?

- **Recurrent Neural Networks (RNNs):** Particularly useful in training RNNs, which are prone to both vanishing and exploding gradients due to their sequential nature.

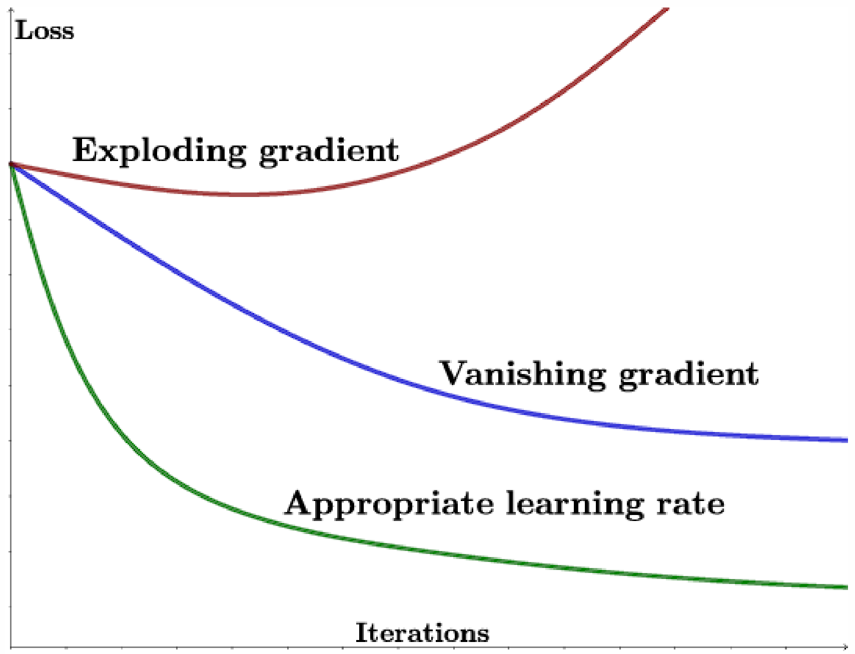
- **Deep Neural Networks:** In very deep networks where gradients can grow significantly, gradient clipping can help stabilize training.

- **Training with High Learning Rates:** When using higher learning rates, gradients can become large, making clipping a useful technique to maintain stability.

Vanishing and Exploding Gradients



Analogy to control theory



- If η is to high $\mapsto$ **positive feedback** $\mapsto$ loss grows **without bounds**
- If η is to small $\mapsto$ **negative feedback** $\mapsto$ **gradient vanishes**
- Choice of η is **critical** for learning