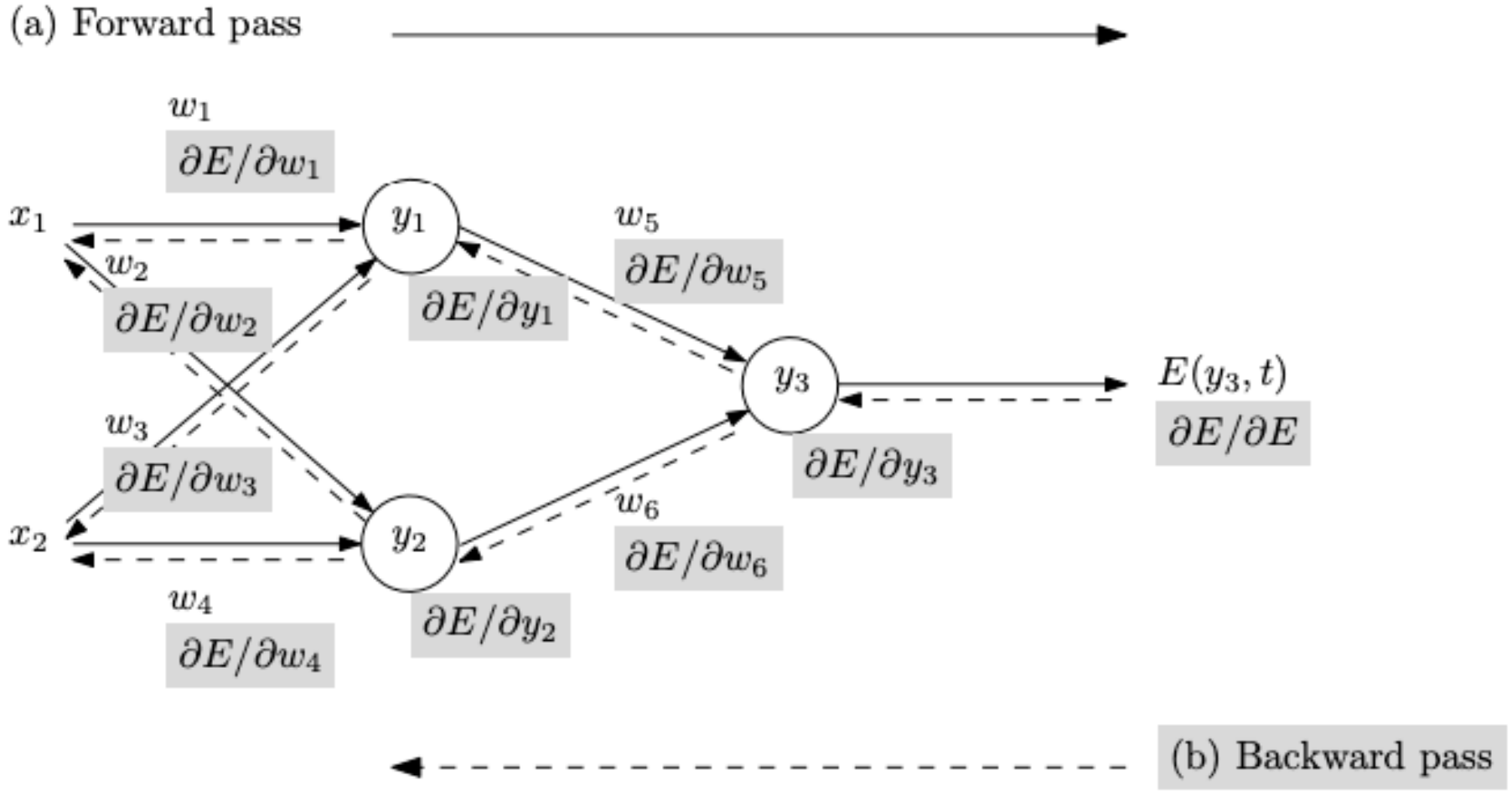
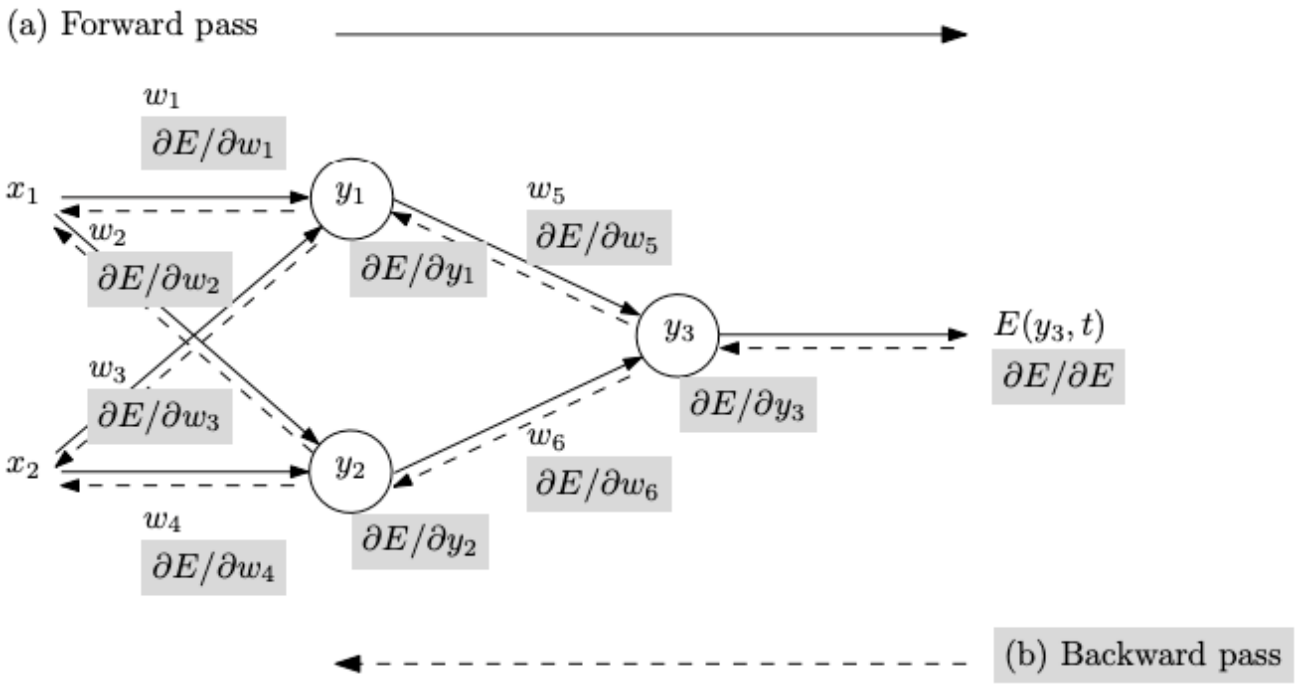
Exploding & Vanishing Gradients

MUKESH KUMAR


(a) Forward pass
w_1
$\partial E/\partial w_1$
x_1
y_1
w_5
$\partial E/\partial w_5$
w_2
$\partial E/\partial w_2$
$\partial E/\partial y_1$
y_3
$E(y_3, t)$
$\partial E/\partial E$
w_3
$\partial E/\partial w_3$
$\partial E/\partial y_3$
w_6
x_2
y_2
$\partial E/\partial w_6$
w_4
$\partial E/\partial y_2$
$\partial E/\partial w_4$
(b) Backward pass

Gradient w.r.to w1

$$\partial E/\partial w_1 = \partial E/\partial E * \partial E/\partial w_5 * \partial E/\partial y_1 * \partial E/\partial w_1$$

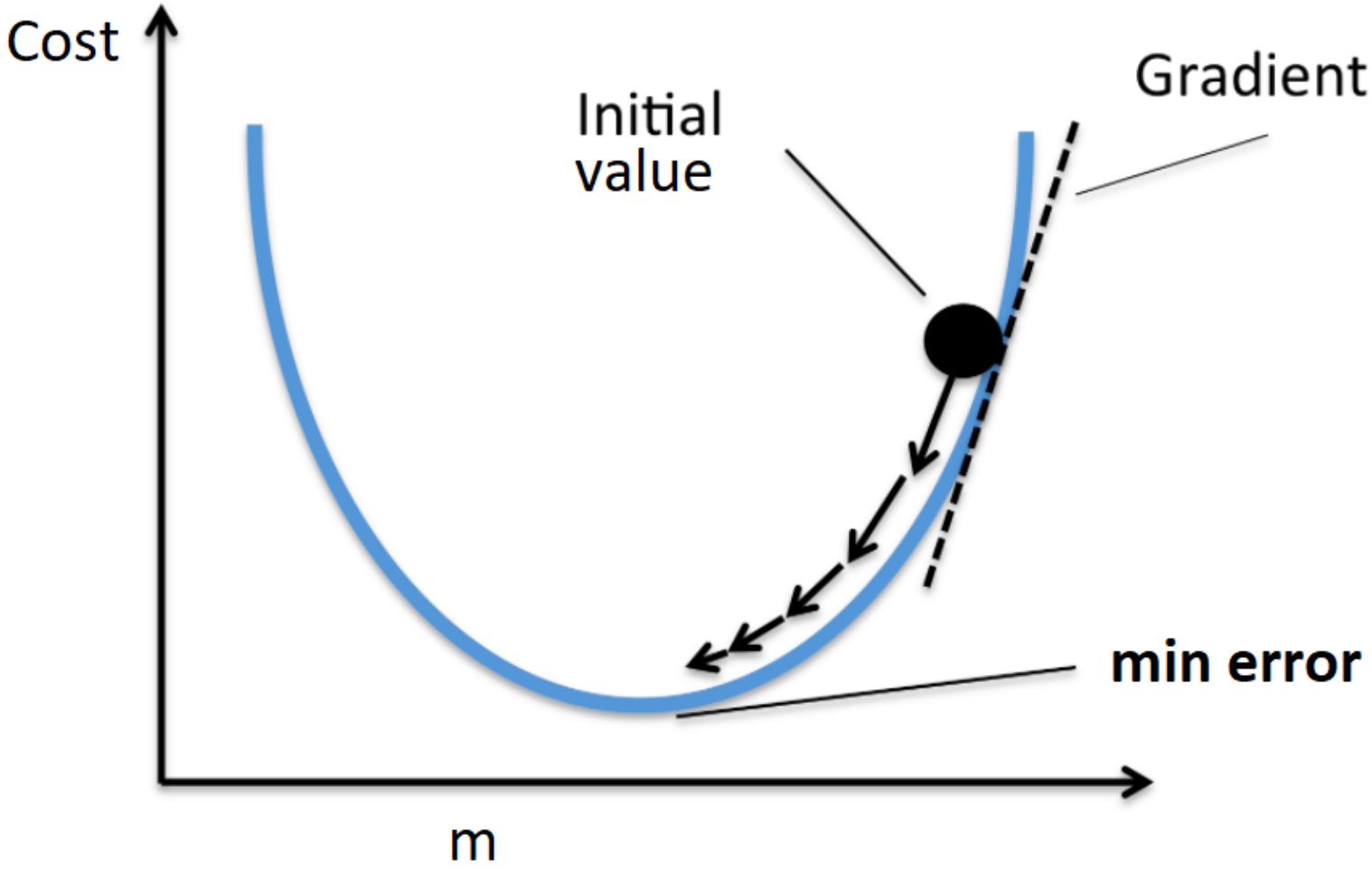


Gradient Descent Weight Calculation

- When the gradient is too low the old weight will be almost equal to new weight

- When the gradient is too high , change in weight will be huge causing the model parameter to oscillate and it would never converge to optimum value

$$W_{new} = W_{old} - \alpha \underbrace{\frac{dJ}{dW}}_{\text{gradient}}$$


Cost
Initial
value
Gradient
min error
m

Exploding Gradients

- **Definition:** This occurs when gradients become excessively large during backpropagation.

- Because the gradients are so large, weight updates become huge, which can cause the model's parameters to "explode" to large values.

Exploding Gradients

- **Cause**: Exploding gradients are usually caused by repeated multiplication of gradients in deep networks, especially if weights have values larger than 1. In such cases, the gradient increases exponentially as it backpropagates through layers.

- **Effect**: The training process can become unstable, and the model's parameters may oscillate wildly or diverge to extremely large values.

Exploding Gradients

- **Solutions**:

- **Gradient Clipping**: Limits the maximum gradient values to a set threshold, preventing them from getting too large.

- **Normalization**: Batch Normalization can help reduce the risk of exploding gradients by normalizing layer outputs.

Gradient Clipping

- **Gradient clipping** is a technique used to prevent the gradients from becoming too large, which can lead to the exploding gradient problem in deep or recurrent neural networks.

```
# Compile the model with a gradient-clipped optimizer
optimizer = tf.keras.optimizers.SGD(learning_rate=0.01, clipnorm=1.0)  # Set clipnorm=1.0

model.compile(optimizer=optimizer, loss='mse')
```

How Gradient Clipping Works

- Gradient clipping involves setting a threshold for the gradients.
- During backpropagation, if the computed gradient exceeds this threshold, it is scaled down to ensure it remains within the acceptable range.

Diminishing (Vanishing) Gradients

- **Definition:** This happens when gradients become very small as they backpropagate through the network, making it difficult for the network to learn from early layers.

- With diminishing gradients, the weights associated with earlier layers receive little to no updates during training.

Diminishing (Vanishing) Gradients

- **Effect**: The training process slows down significantly or stalls, as the model can't learn features effectively from earlier layers.

- **Solutions**:
- **Activation Functions**: ReLU and variants like Leaky ReLU or ELU can reduce vanishing gradients because they don't squash outputs as much as sigmoid or tanh.
- **Initialization Schemes**: Proper weight initialization, such as He initialization or Xavier initialization, can help by setting up appropriate starting weight magnitudes.

Activation functions that commonly contribute to vanishing or exploding gradient

Sigmoid

Vanishing Gradients:

- Sigmoid squashes input values to a range between 0 and 1, causing small gradients for large input values (either positive or negative).

- During backpropagation, this small gradient diminishes layer by layer.

Exploding Gradients:

- When the input is close to zero, the sigmoid function's slope is steep, potentially causing larger gradient values for certain weight updates.

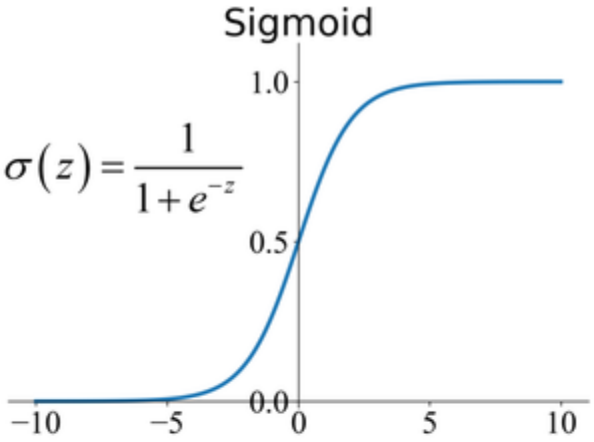
Tanh (Hyperbolic Tangent)

Vanishing Gradients:

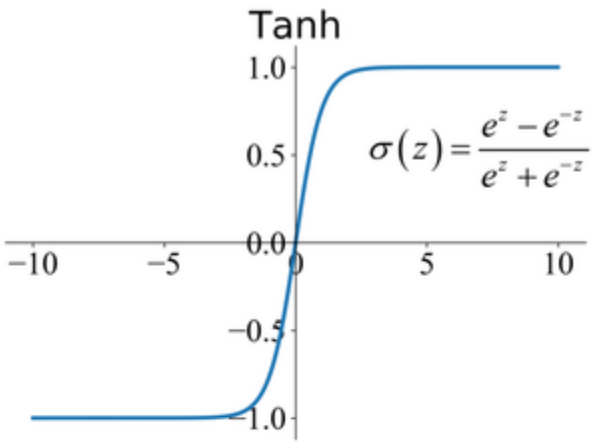
- Like the sigmoid function, tanh squashes values, but within a range of -1 to 1.
- While it has a stronger gradient in the range around 0, it still suffers from vanishing gradients when values are saturated (i.e., far from zero).

- **Exploding Gradients**: Near zero, tanh has a higher gradient, which can contribute to large weight updates.

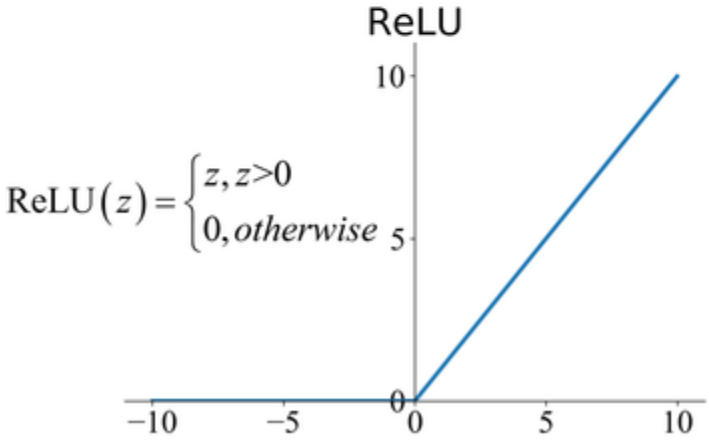
This is why alternative functions like **ReLU**, **Leaky ReLU**, and **ELU** are often preferred in deep networks due to their ability to maintain more stable gradient magnitudes.


Sigmoid
$\sigma(z) = \frac{1}{1+e^{-z}}$
1.0
0.5
0.0
−10
−5
0
5
10

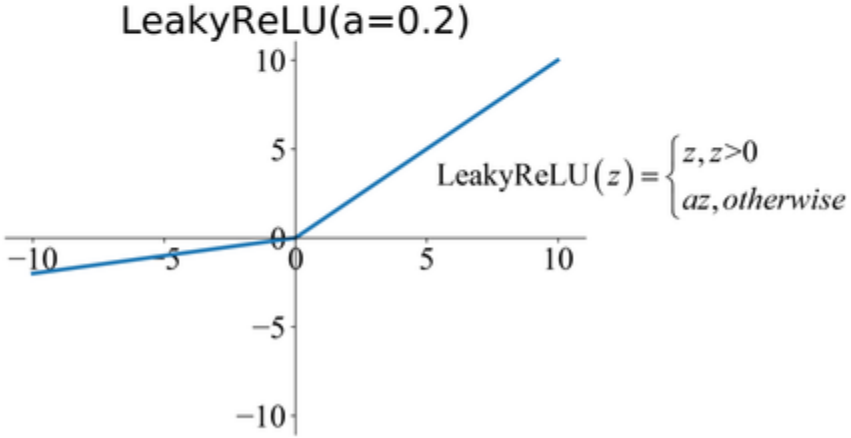
(a)


Tanh
$\sigma(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$
1.0
0.5
0.0
−0.5
−1.0
−10
−5
0
5
10

(b)


ReLU
$\mathrm{ReLU}(z) = \begin{cases} z, z>0 \\ 0, otherwise \end{cases}$
10
5
0
−10
−5
0
5
10

(c)


LeakyReLU(a=0.2)
$\mathrm{LeakyReLU}(z) = \begin{cases} z, z>0 \\ az, otherwise \end{cases}$
10
5
0
−5
−10
−10
−5
0
5
10

(d)

Leaky ReLU

- **Formula**:

$$f(x) = \max(\alpha x, x)$$

Here, α is a small constant (often 0.01) and it allows a small, non-zero gradient for negative values of x. So, instead of outputting 0 for negative inputs, Leaky ReLU outputs a small negative value determined by α.

ELU (Exponential Linear Unit)

Formula:

For ELU, the activation function is defined as:

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(\exp(x) - 1) & \text{if } x < 0 \end{cases}$$

Where α is a hyperparameter (usually a small positive value, like 1).