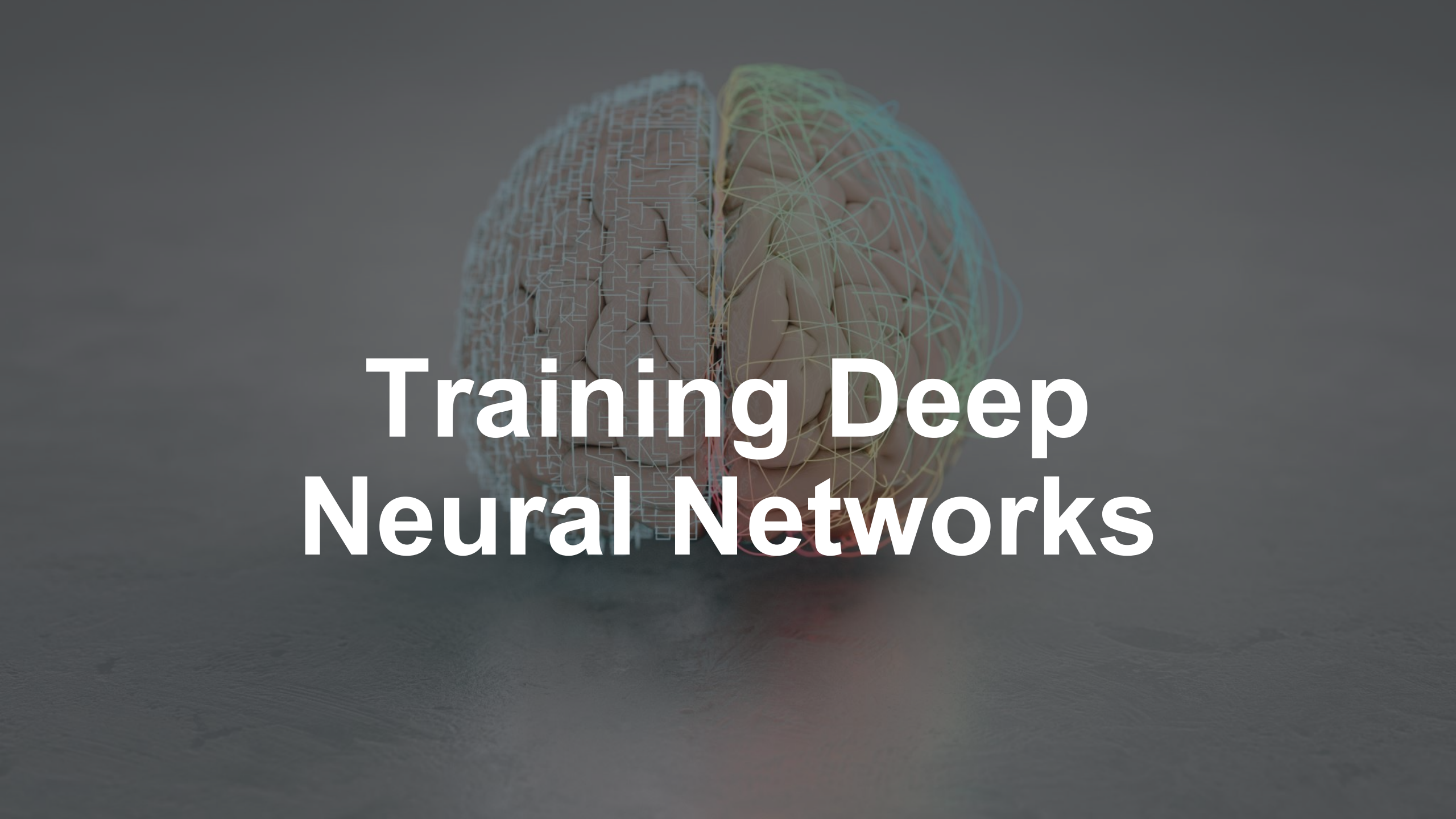
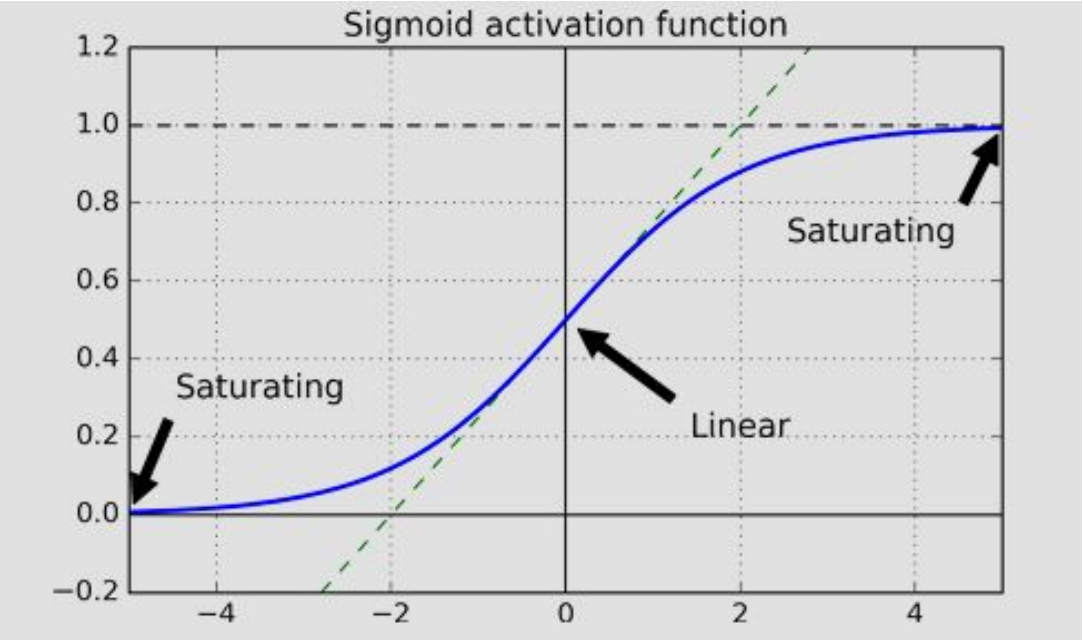


Training Deep Neural Networks

The Vanishing/Exploding Gradients Problem

- Deep networks suffer from unstable gradients
- Gradients can become extremely small (vanish) or large (explode)
- This makes training slow or impossible
- Caused by repeated multiplication of gradients during backpropagation
- Impacts:
  - Early layers learn very slowly
  - Training stalls or diverges



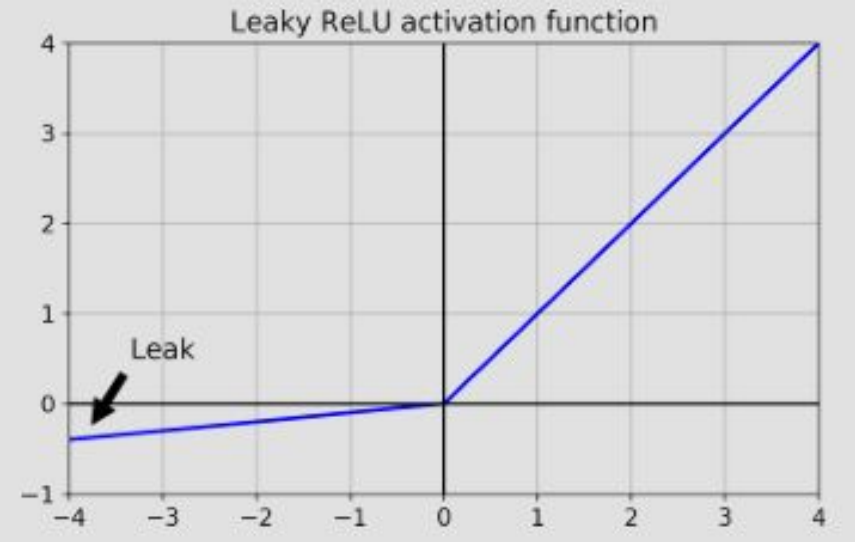
Glorot and He Initialization

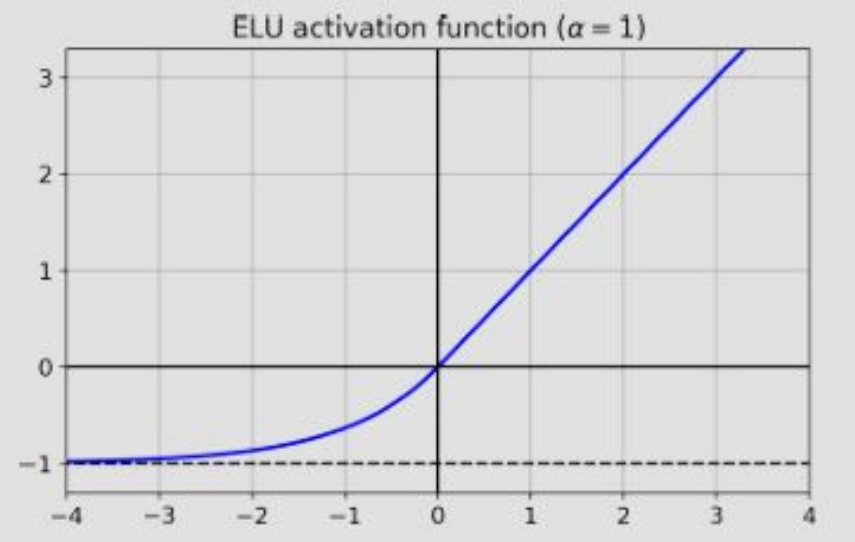
- Proper weight initialization is critical
- The variance of the outputs of each layer to be equal to the variance of its inputs
- Initializing all weights and biases to zero makes all neurons in a given layer perfectly identical
  - Backpropagation will remain identical.
  - Model will act as if it had only one neuron per layer: it won't be too smart.

- **Glorot (Xavier) Initialization:**
  - For activation functions like tanh and logistic
  - Weights initialized with variance = $\frac{2}{n_{in}+n_{out}}$
- **He Initialization:**
  - For ReLU and its variants
  - Variance = $\frac{2}{n_{in}}$
- Helps maintain stable gradients through layers

Nonsaturating Activation Functions

- **Saturating functions:** Sigmoid, Tanh → gradients vanish
- **Nonsaturating functions:** ReLU → avoids vanishing gradients
    - **Dying ReLUs:** during training, some neurons effectively “die,” meaning they stop outputting anything other than 0.
    - A neuron dies when its weights get tweaked in such a way that the weighted sum of its inputs are negative for all instances in the training set.
    - Gradient Descent does not affect it anymore because the gradient of the ReLU function is zero when its input is negative.





Nonsaturating Activation Functions

- **Improved variants:**
  - **Leaky ReLU:** Allows small gradient when unit is off

$$LeakyReLU_{\alpha}(z) = \max(\alpha z, z)$$

    - The hyperparameter α defines how much the function "leaks": it is the slope of the function for z < 0 and is typically set to 0.01.
    - This small slope ensures that leaky ReLUs never die; they can go into a long coma, but they have a chance to eventually wake up.
  - **Exponential Linear Unit (ELU):** Smooth and avoids dead neurons

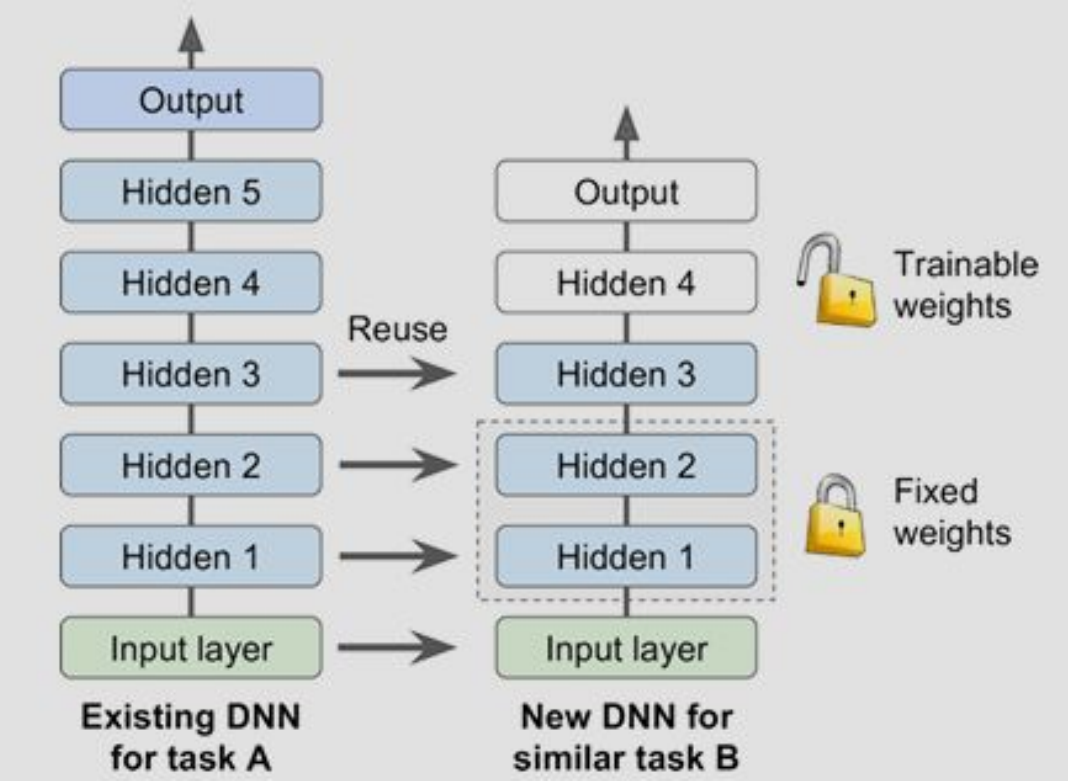
$$ELU_{\alpha}(z) = \begin{cases} \alpha(e^{z} - 1), & if\ z < 0 \\ z, & if\ z \geq 0 \end{cases}$$

Gradient Clipping

- Used to combat **exploding gradients**
- Clips gradients during backpropagation so that they never exceed some threshold
- Common in RNNs and deep networks
- Two types:
  - **Value Clipping:** Clip by value
  - **Norm Clipping:** Clip by norm

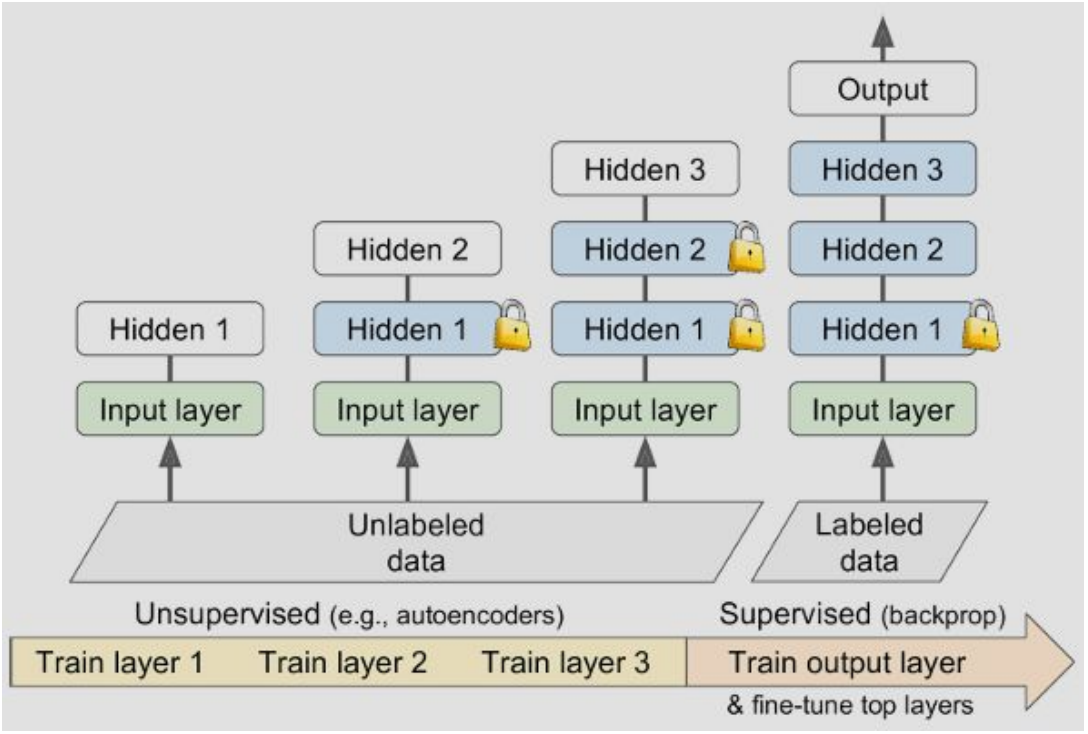
Reusing Pretrained Layers

- **Transfer Learning:**
  - Use layers from a pre-trained model
  - Freeze early layers, retrain later layers
  - Especially useful when data is scarce



Reusing Pretrained Layers

- **Unsupervised Pretraining**
  - Train layers with unlabeled data first
  - Fine-tune with labeled data



Faster Optimizers

-
- **Gradient Descent (Vanilla)**

$$\theta_{t+1} \leftarrow \theta_t - \eta \nabla_\theta J(\theta_t)$$

- **Intuitive Analogy:** The hiker looks at the steepness of the ground directly under their feet and takes a small step exactly in the steepest downhill direction. They then stop, check the steepness again, and take another small step. This is simple but can be slow and inefficient, especially on flat plains or noisy, rocky paths.

Faster Optimizers

- **Momentum Optimization:**
  - Accelerates SGD by considering past gradients
  1. $m_t \leftarrow \beta m_{t-1} + \eta \nabla_\theta J(\theta_t)$, initially $m_0 = 0$
  2. $\theta_{t+1} \leftarrow \theta_t - m_t$

  Exponentially decaying average weights

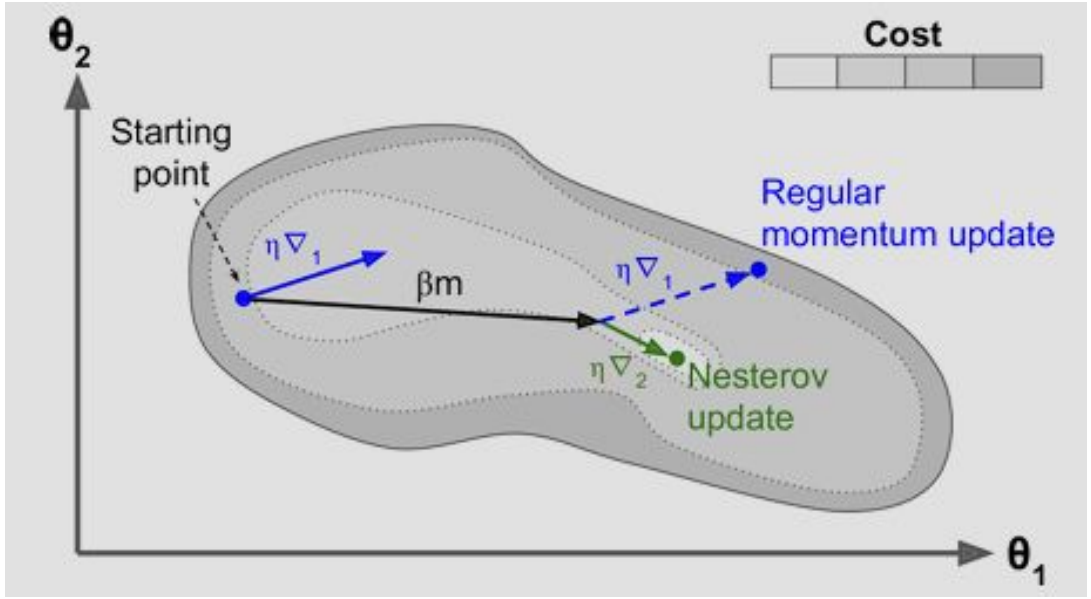
  - The algorithm introduces a new hyperparameter β, called the momentum, which must be set between 0 (high friction) and 1 (no friction).
  - Due to the momentum, the optimizer may overshoot a bit, then come back, overshoot again, and oscillate like this many times before stabilizing at the minimum.
- **Intuitive Analogy:** The hiker now has momentum, like a heavy ball rolling down the hill. Instead of stopping at each step, they build up speed in a consistent direction. This helps them roll through small bumps (noisy gradients) and flat areas faster. On a consistent slope, they go faster and faster.

Faster Optimizers

- **Nesterov Accelerated Gradient (NAG):**
  - Foresees (Look-ahead) the future position for better updates
  - Measures the gradient of the cost function not at the local position θ but slightly ahead in the direction of the momentum, at $\theta + \beta m$

  1. $m_t \leftarrow \beta m_{t-1} + \eta \nabla_\theta J(\theta_t + \beta m_{t-1})$
  2. $\theta_{t+1} \leftarrow \theta_t - m_t$

- **Intuitive Analogy:** This is a smarter hiker with momentum. They think, "Based on my current speed, where will I be in a moment?" They then look at the slope *at that future location* and adjust their direction *now*. If they see a turn coming up ahead, they start leaning into it early, preventing overshooting.

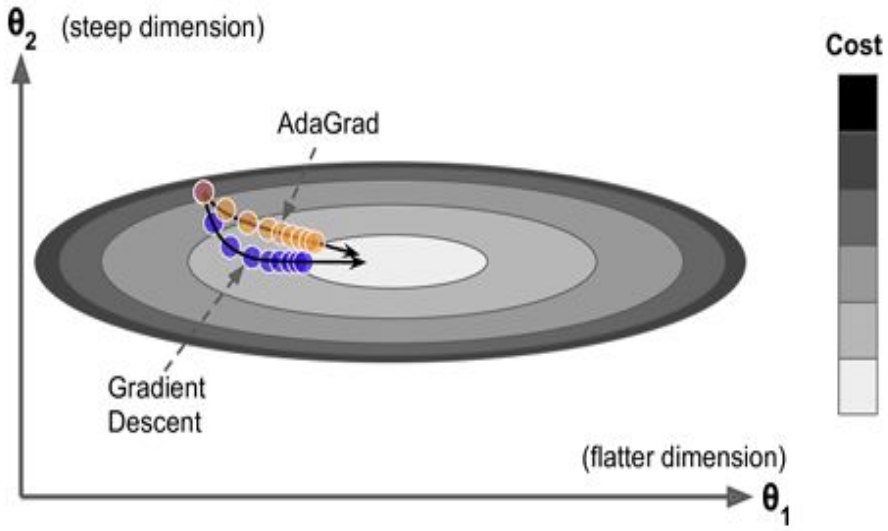


Faster Optimizers

- **AdaGrad:**
  - Adapts learning rate per parameter
  - **Adaptive learning rate:** Algorithm decays the learning rate faster for steep dimensions than for dimensions with gentler slopes.

1. $s_t \leftarrow s_{t-1} + \nabla_\theta J(\theta_t) \otimes \nabla_\theta J(\theta_t)$
2. $\theta_{t+1} \leftarrow \theta_t - \frac{\eta \nabla_\theta J(\theta_t)}{\sqrt{s_t + \epsilon}}$

- **Intuitive Analogy:** The hiker now has a personalized step size for every part of the terrain. For gentle slopes, they take larger steps. For rare, steep slop, they take smaller steps. This is great for sparse data but eventually, the "brakes" become so strong that the hiker stops moving entirely.
- **Drawback:** The learning rate gets scaled down so much that the algorithm ends up stopping entirely before reaching the global optimum.



Faster Optimizers

- **RMSProp:**
  - Accumulate only the gradients from the most recent iterations (as opposed to all the gradients since the beginning of training).
  - It does so by using exponential decay in the first step

1. $s_t \leftarrow \beta s_{t-1} + (1-\beta)\nabla_\theta J(\theta_t) \otimes \nabla_\theta J(\theta_t)$
2. $\theta_{t+1} \leftarrow \theta_t - \frac{\eta \nabla_\theta J(\theta_t)}{\sqrt{s_t + \epsilon}}$

  - The decay rate β is typically set to 0.9.

- **Intuitive Analogy:** RMSProp fixes AdaGrad's aggressive, monotonically decreasing learning rate. The hiker now uses a "moving average" of recent steepness to decide their step size. A sudden steep slope (large gradient) will cause a temporary small step, but this effect will fade away over time if it doesn't persist, preventing the learning rate from vanishing.

Faster Optimizers

- **Adam (Adaptive Moment Estimation):** Combines Momentum and RMSProp
  1. $m_t \leftarrow \beta_1 m_{t-1} - (1-\beta_1)\nabla_\theta J(\theta_t)$
  2. $s_t \leftarrow \beta_2 s_{t-1} + (1-\beta_2)\nabla_\theta J(\theta_t) \otimes \nabla_\theta J(\theta_t)$
  3. $\widehat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$
  4. $\widehat{s_t} \leftarrow \frac{s_t}{1-\beta_2^t}$

  Bias correction
  5. $\theta_{t+1} \leftarrow \theta_t - \frac{\eta \hat{m}_t}{\sqrt{\hat{s}_t + \epsilon}}$
- t represents the iteration number

- **Intuitive Analogy:** Adam is the most sophisticated hiker. It behaves like a heavy ball (momentum) that rolls down the hill, but it also has personalized, adaptive step sizes for each foot and direction (like RMSProp). Furthermore, it has a "warm-up" period (bias correction) to prevent taking overly large steps at the very beginning of the journey.

Learning Rate Scheduling

- Adjust learning rate during training

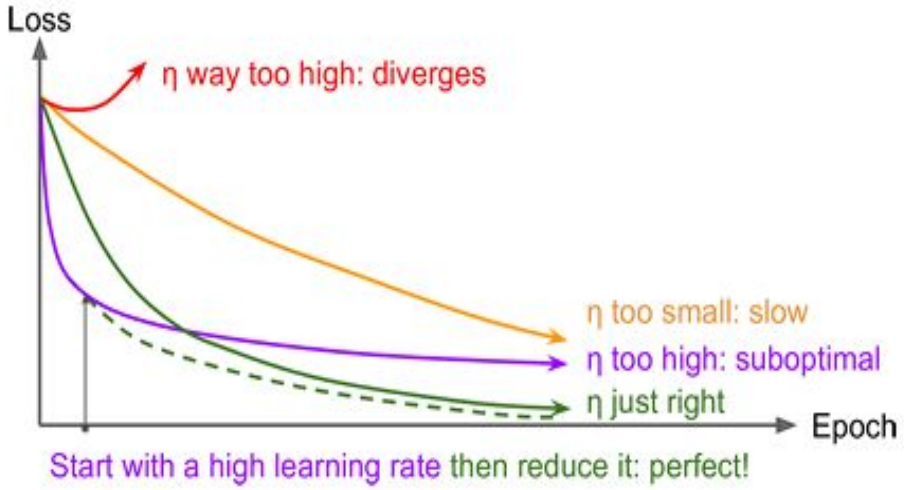
- **Power Scheduling:** Gradually decreases LR

$$\eta(t) = \frac{\eta_0}{\left(1 + \frac{t}{s}\right)^c}$$

- **Exponential Scheduling:** Drops LR exponentially

$$\eta(t) = \eta_0 \times 0.1^{\frac{t}{s}}$$

- **Piecewise Constant:** Drops at fixed intervals ($\eta_0 = 0.1$ for 5 epochs)
- **Performance Scheduling:** Reduce the learning rate by factor of λ when validation error stops improving
- **1Cycle Scheduling:** Cyclic learning rate for faster convergence



Regularization Techniques

- **L1 & L2 Regularization:**
  - Penalize large weights
- **Dropout:**
  - Randomly turn off neurons during training
  - Prevents over-reliance on specific neurons
  - The hyperparameter p is called the dropout rate, and it is typically set between 10% and 50%.