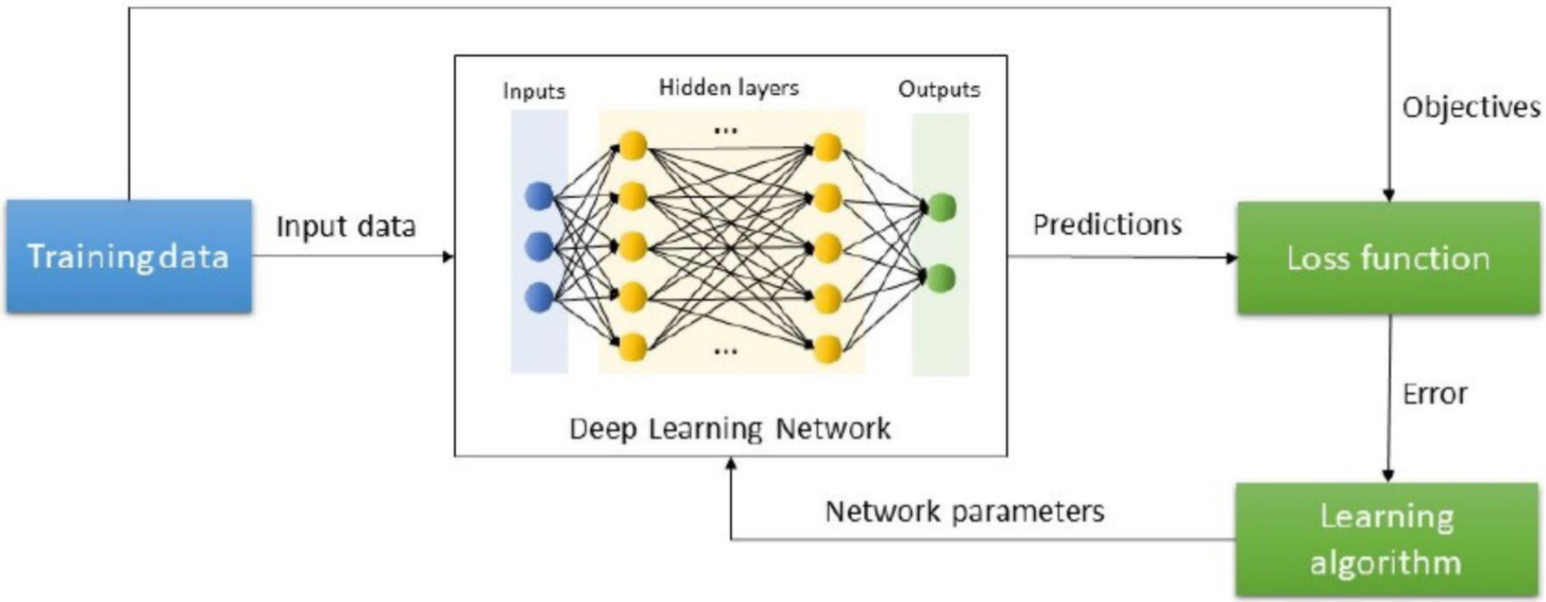
Neural Networks

Fatemeh Mansoori

University of Isfahan

Some of the slide are based on slides from the machine learning course by sharifi zarchi

Training Process



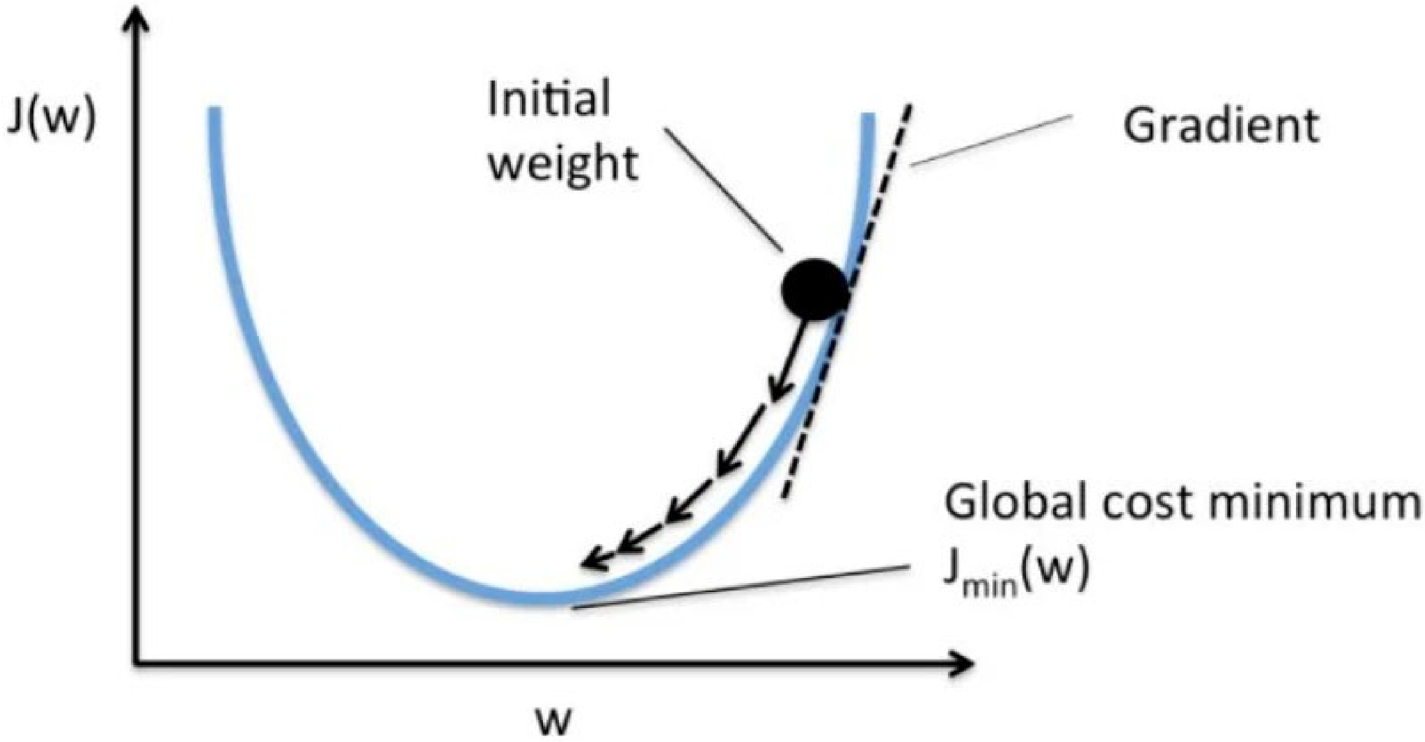
Optimization

- **Goal**: Find the value of x where $f(x)$ is at a **minimum** or **maximum**.
- In neural networks, we aim to minimize **prediction error** by finding the optimal weights w^*:

$$w^* = \arg\min_{w} J(w)$$

- Simply put: determine the **direction to step** that will quickly **reduce loss**.

Gradient Descent



Gradient Descent

Gradient Descent: Minimizes the loss function by updating weights based on the gradient.
Weight Update Rule:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{\partial L}{\partial w}$$

Where:

- η is the learning rate (step size).
- $\frac{\partial L}{\partial w}$ is the gradient of the loss function with respect to w.

Example: Gradient Descent and Updating Weights

Example Problem:

- Initial weight: $w_0 = 2$
- Learning rate: $\eta = 0.1$
- Loss function: $L(w) = (y - wx)^2$

  Example: For $x = 3$, $y = 10$, and $w_0 = 2$,

Gradient Calculation:

$$\frac{\partial L}{\partial w} = -2x(y - wx)$$

$$\frac{\partial L}{\partial w} = -24, \quad w_{\text{new}} = 4.4$$

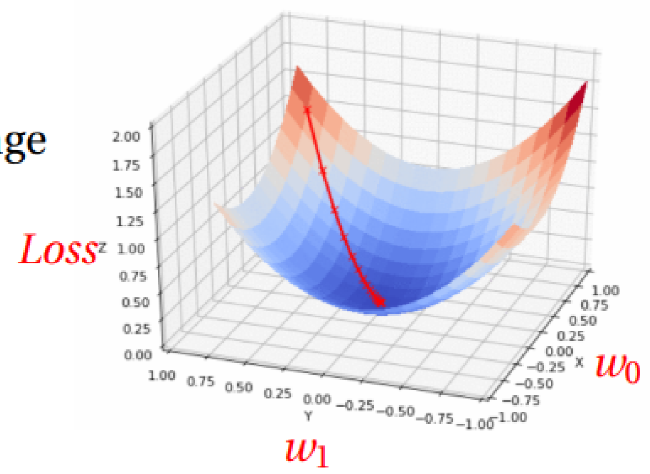
Gradient Descent: Formula and Process

Weight Update Formula:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{\partial L}{\partial w}$$

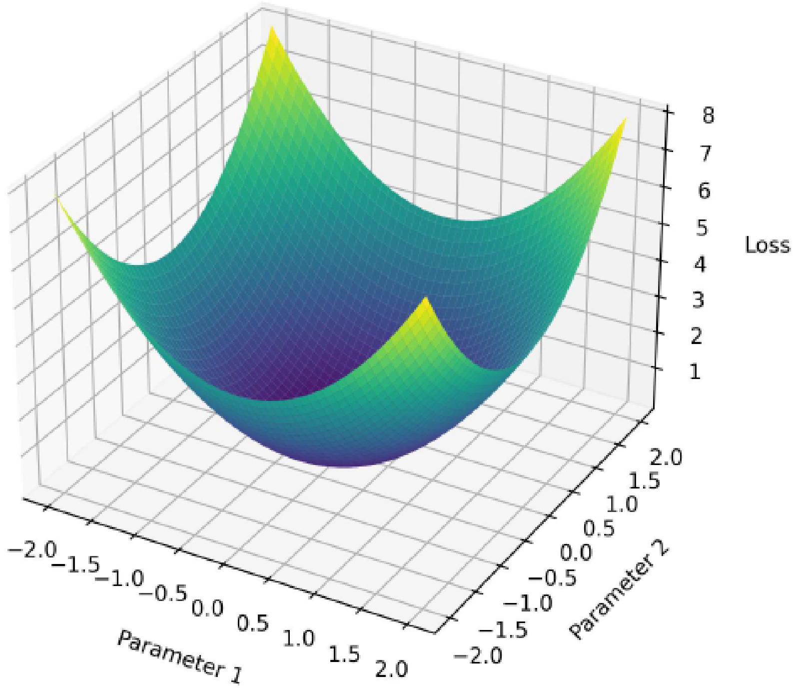
Steps in Gradient Descent:

- Compute the gradient of the loss function.
- Update the weights using the update rule.
- Repeat until convergence.
- Image adapted from Data Science Stack Exchange



Convexity and Optimization

- **Convex Functions**:
  - A function is **convex** if any line segment between points on the curve lies **above or on** the curve.
  - Convex functions are easier to optimize, as they have a single **global minimum**.
  - Numerical methods like **Gradient Descent** are guaranteed to reach the global minimum in convex functions.



Non-Convex functions and challenges

- **Non-Convex Functions**:
  - Characterized by multiple **local minima** and **saddle points**.
  - **Global Minimum**: Overall lowest point.
  - **Local Minimum**: Lower than nearby points, but not the lowest overall.
  - **Saddle Points**: Regions where the gradient is close to zero but can increase or decrease in other directions.
- Finding the **global minimum** is more complex in non-convex functions.

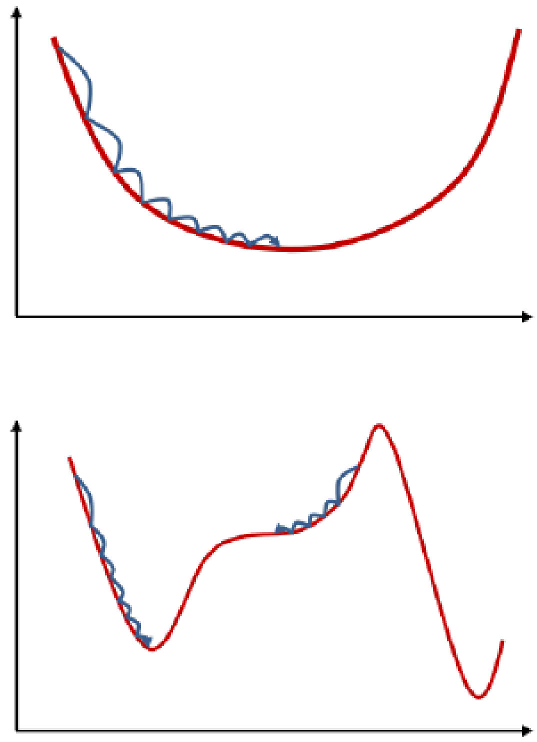


Figure 2: Convex (top) vs. Non-Convex (bottom) functions. Source: (CMU, 11-785)

Loss function in NN

- The **loss surface** shows how error changes based on network weights.
- For neural networks, the loss surface is typically **non-convex** due to multiple layers, nonlinear activations, and complex parameter interactions, resulting in **multiple local minima** and **saddle points**.
- In large networks, most local minima yield similar error values close to the **global minimum**; this is less true in smaller networks.

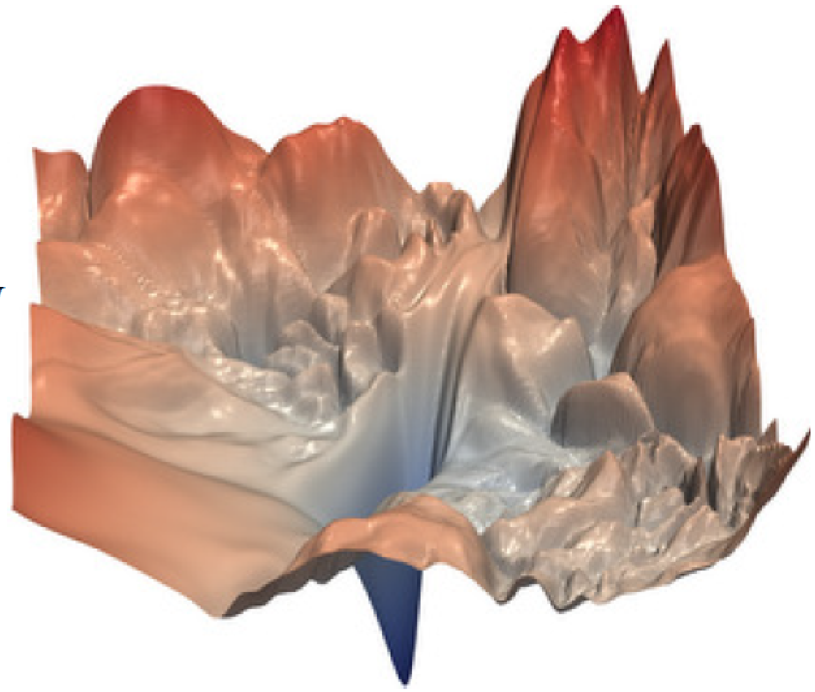


Figure 3: Loss surface of ResNet56. Source: GitHub: Loss Landscape

Gradient Descent Overview

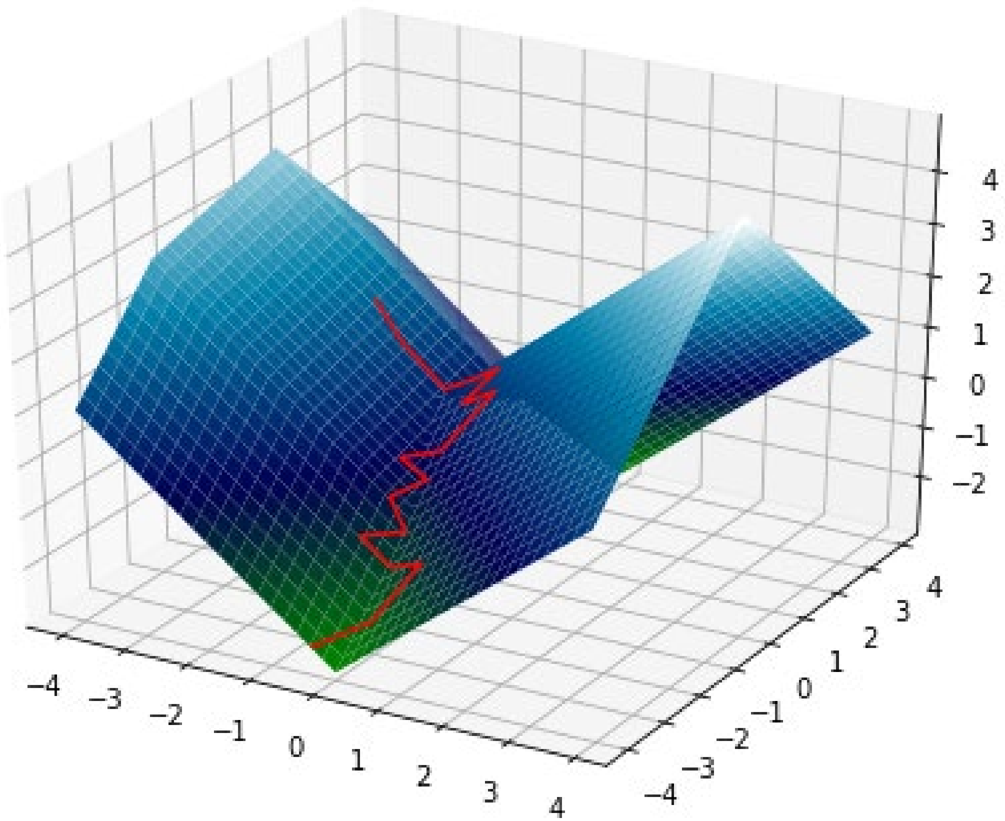
- **Gradient Descent**: As mentioned earlier in this course, Gradient Descent is an iterative method to minimize error by updating weights in the direction of the **negative gradient**:

$$w_{t+1} = w_t - \eta \nabla J(w_t)$$

  where η is the **learning rate**.

- **Types of Gradient Descent**:
  - **Batch**: Full dataset for stable but slow updates.
  - **Stochastic (SGD)**: One data point for fast, noisy updates.
  - **Mini-Batch**: Small batches, balancing speed and stability.

SGD and Saddle point



Problems with Gradient descent

- **High Variability (SGD)**: Quick in steep directions but slow in shallow ones, causing **jitter and slow progress**.
- **Local Minima and Saddle Points**: Risk of **sub-optimal solutions** or long convergence times in flat regions.
- **Noisy Updates**: Using individual points or mini-batches introduces noise, affecting stable convergence.

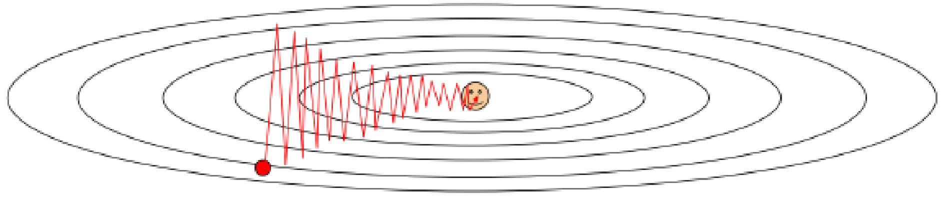


Figure 4: SGD Variability (CS231n, Stanford)

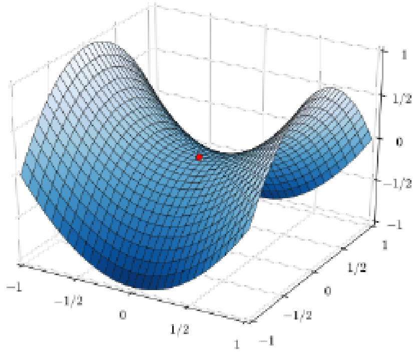


Figure 5: Saddle Point. Source: Wikipedia

Improvement to Gradient Descent — Momentum

- One of the tricky aspects of Gradient Descent is dealing with steep slopes
  - Gradient is large there, you could take a large step when you actually want to go slowly and cautiously
  - This could result in bouncing back and forth, thus slowing down the training
- With Gradient Descent you make an update to the weights at each step, based on the gradient and the learning rate
  - Adjust the gradient
  - Adjust the learning rate

Momentum vs SGD

- Momentum is a way to adjust the gradient
- In SGD we look only at the current gradient and ignore all the past gradients along the path we took
    - there is a sudden anomaly in the loss curve, your trajectory may get thrown
    - using Momentum, let the past gradients guide overall direction
- how far in the past do you go?
- does every gradient from the past count equally?
    - this would make sense that things from the recent past should count more than things from the distant past
- Momentum algorithm uses the exponential moving average of the gradient, instead of the current gradient value.
- Which algorithm use momentum to updating the gradient ?
    - Stochastic Gradient Descent with Momentum (SGDM)

```
v = beta * v + (1 - beta) * gradient
parameters = parameters - learning_rate * v
```

SGDM (example)

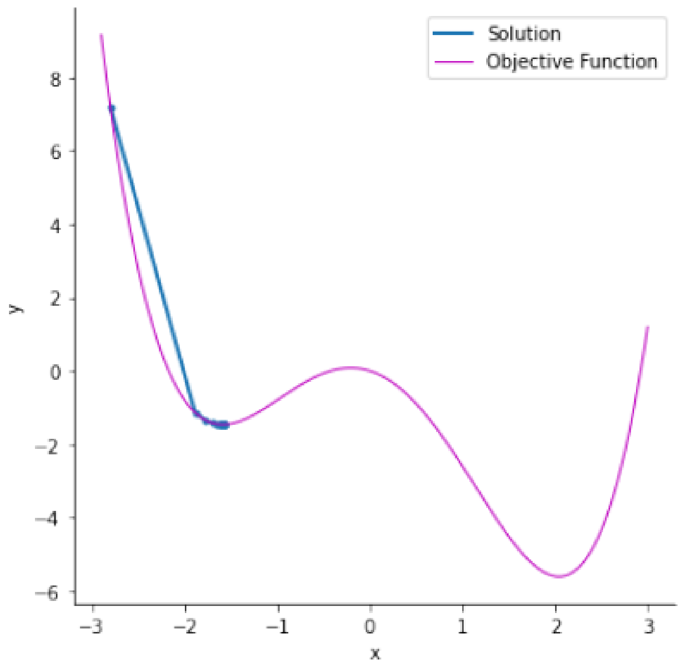


Figure 7: Stochastic gradient descent without momentum stops at a local minimum. Source: Akash Ajagekar (SYSEN 6800 Fall 2021)

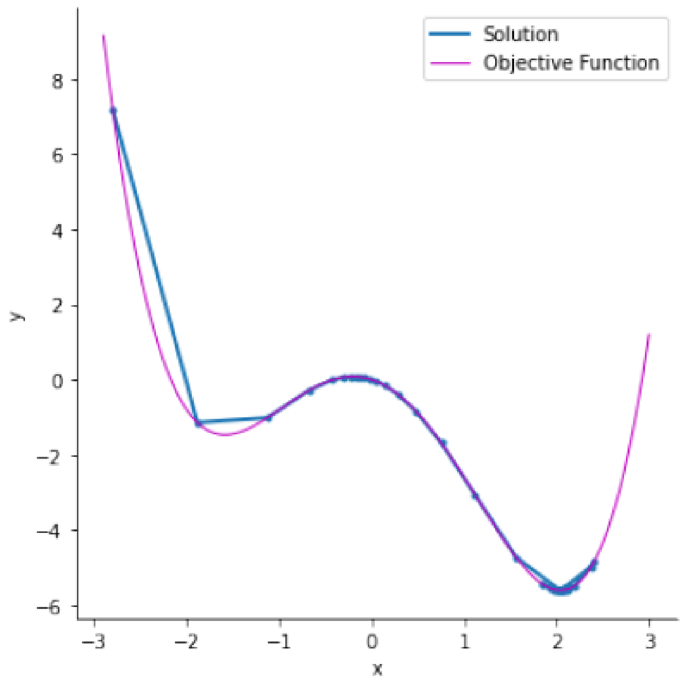


Figure 8: Stochastic gradient descent with momentum stops at the global minimum. Source: Akash Ajagekar (SYSEN 6800 Fall 2021)

Modify Learning Rate

- So for we have a constant learning rate
  - keeping the learning rate constant from one iteration to the next
  - gradient updates are using the same learning rate for all parameters.
- Modify the learning rate based on the gradient
- Use of past gradients (for each parameter separately) to choose the learning rate for that parameter.
- Optimizer algorithms that do this
  - Adagrad (Adaptive Gradient)
  - Adadelta
  - RMS Prop. (Root Mean Square Propagation)

Algorithm for updating learning rate

- for a parameter that has a steep slope
  - the gradients are large and the squares of the gradients are really large and always positive
  - the algorithm calculates the learning rate by dividing the accumulated squared gradients by a larger factor. This allows it to slow down on steep slopes.
- for shallower slopes
  - the accumulation is small and so the algorithm divides the accumulated squares by a smaller factor to compute the learning rate. This boosts the learning rate for gentle slopes.

$$v_t = v_{t-1} + \left[\frac{\delta L}{\delta w_t}\right]^2$$

- Adagrad squares the past gradients and adds them up weighting all of them equally

$$w_{t+1} = w_t - \frac{\alpha_t}{(v_t+\varepsilon)^{1/2}} * \left[\frac{\delta L}{\delta w_t}\right]$$

- RMSProp also squares the past gradients but uses their exponential moving average, thus giving more importance to recent gradients.

$$w_{t+1} = w_t - \frac{\alpha_t}{(v_t+\varepsilon)^{1/2}} * \left[\frac{\delta L}{\delta w_t}\right]$$

$$v_t = \beta v_{t-1} + (1-\beta) * \left[\frac{\delta L}{\delta w_t}\right]^2$$

Adam optimizer algorithm

- Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.
- Adam realizes the benefits of both SGDM and RMSProp.
  - **Root Mean Square Propagation** (RMSProp)
  - Stochastic gradient descent with momentum
- algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 control the decay rates of these moving averages.

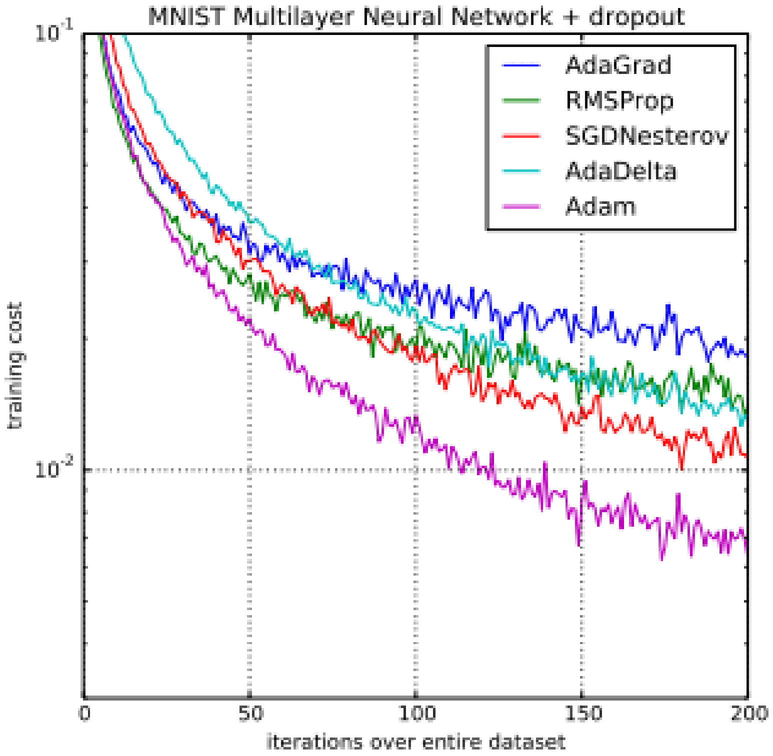
Adam

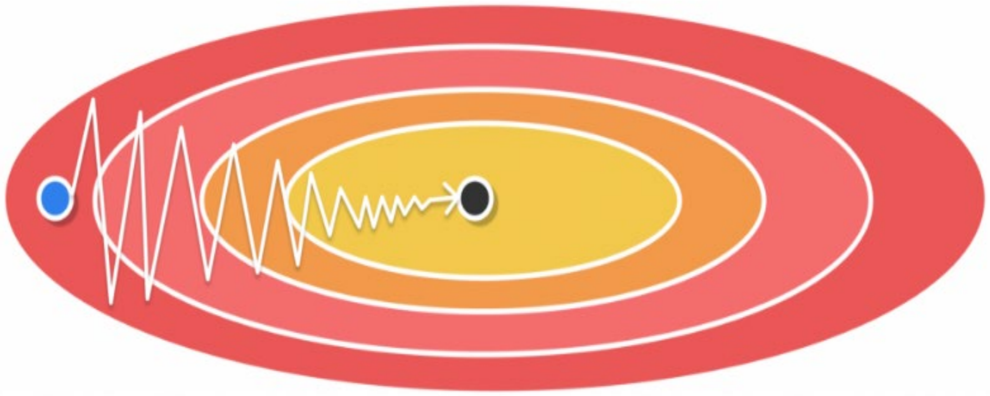
- Adam is a popular algorithm in the field of deep learning because it achieves good results fast.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\delta L}{\delta w_t}\right]$$

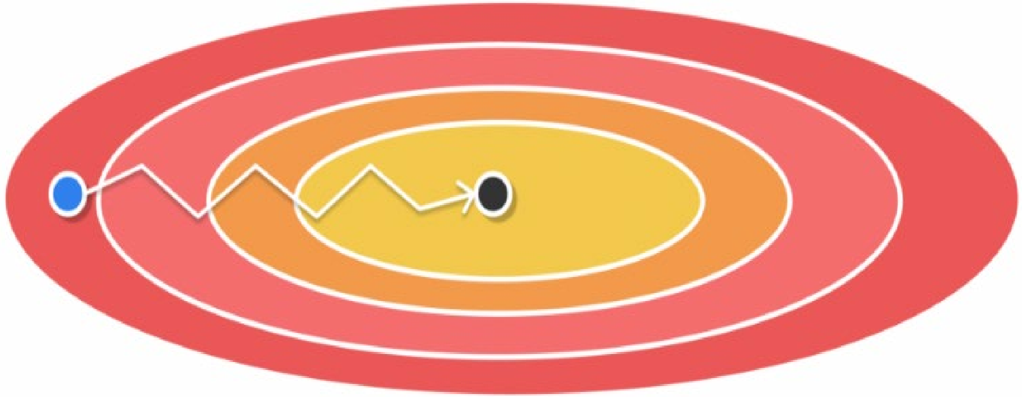
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\delta L}{\delta w_t}\right]^2$$

$$w_{t+1} = w_t - m_t\left(\frac{\alpha}{\sqrt{\widehat{v_t}}+\varepsilon}\right)$$

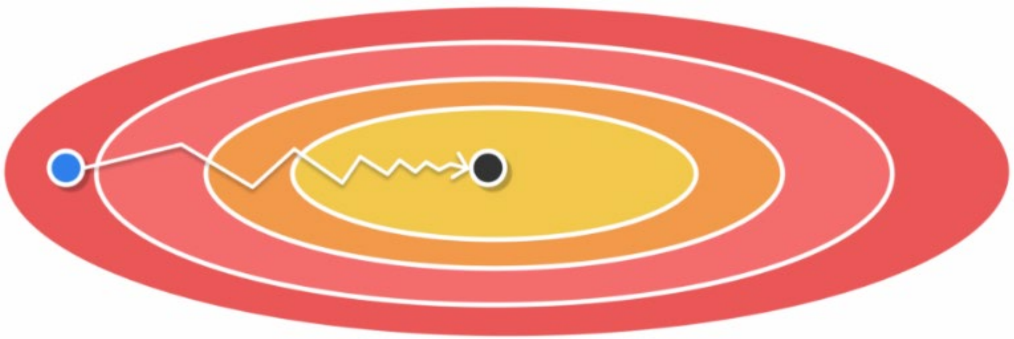




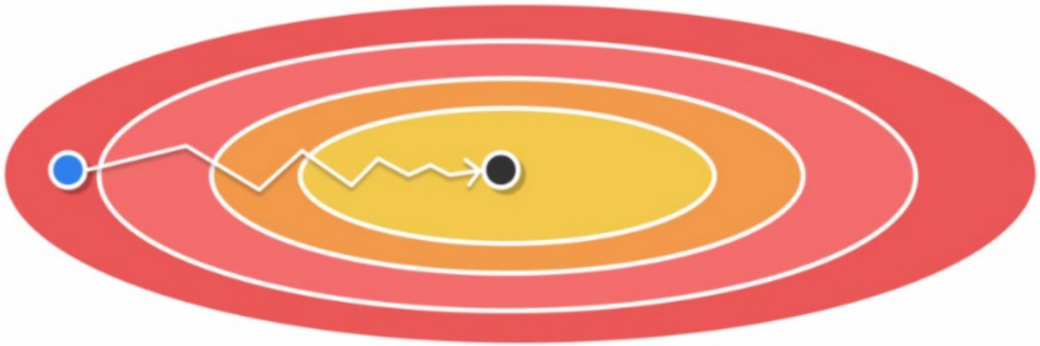
Example of an optimization problem with gradient descent in a ravine area. The starting point is depicted in blue and the local minimum is shown in black.



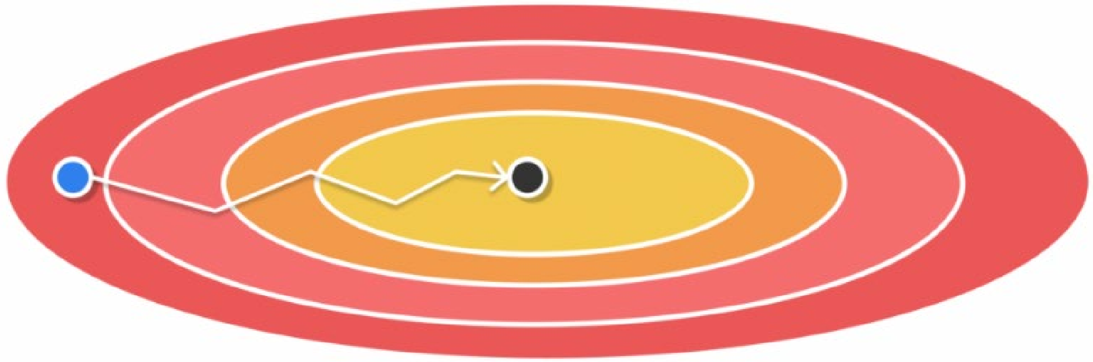
Optimization with Momentum



Optimization with AdaGrad



Optimization with RMSProp



Optimization with Adam