Objectives

- Optimizers
  - RMSProp
  - Adam
- Calculus Review
- Intro to Backpropagation

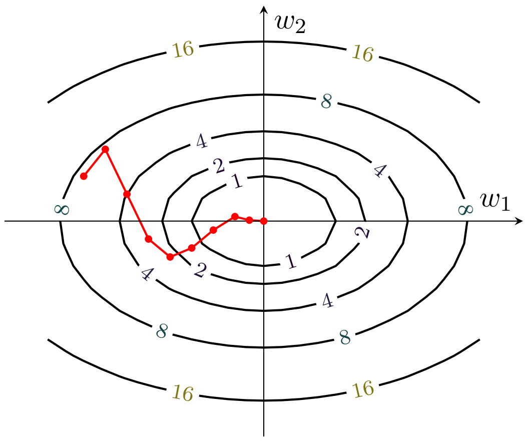
1 Optimizers

1.1 RMSProp

- Root Mean Square Propagation
- Proposed by Geoffrey Hinton during a series of Coursera lectures in 2012

RMSProp typically has a slow convergence rate, since it oscillates more than it moves left to right. However, it can be updated differently in different dimensions, meaning we can adjust the rate at which the gradient descent oscillates up/down as it moves left to right

RMSProp Descent Path on Loss Contours



Standard:

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}, \nabla C = \begin{bmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_n} \end{bmatrix} \rightarrow w = w - \alpha \nabla C$$

We can adjust this equation:

- w_1 should increase at a faster rate than w_2
- This allows for faster convergence, with the gradient descent moving left/right more than oscillating up/down

<u>How?</u>

- Scale components *individually*
- Hadamard Products
  - Element-wise multiplication
  - Example:

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 3w_1 \\ 2w_2 \end{bmatrix}$$

We can apply element-wise operations to RMSProp's update equations:

$$v_{t+1} = \beta v_t + (1 - \beta) \nabla C(w_2)^2$$

$$w_{t+1} = w_t - \alpha \frac{\nabla C(w_t)}{\sqrt{v} + \epsilon}$$

- w = weights
- v = Exponentially decaying average of the squared gradients
- $\nabla C(w)^2$ is an element-wise product:

$$\begin{bmatrix} C_1 \\ C_2 \\ \vdots \end{bmatrix} \odot \begin{bmatrix} C_1 \\ C_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} C_1^2 \\ C_2^2 \\ \vdots \end{bmatrix}$$

- $\frac{\nabla C(w_t)}{\sqrt{v}+\epsilon}$ is an elemental-wise division:

$$\frac{\nabla C}{\sqrt{v}} = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_n \end{bmatrix} \oslash \begin{bmatrix} \sqrt{v_1} \\ \sqrt{v_2} \\ \vdots \\ \sqrt{v_n} \end{bmatrix} = \begin{bmatrix} C_1/\sqrt{v_1} \\ C_2/\sqrt{v_2} \\ \vdots \\ C_n/\sqrt{v_n} \end{bmatrix}$$

If v_1 is large, $C_1/\sqrt{v_1}$ will shrink, making it oscillate less and move more to the right.

$\epsilon \approxeq 10^-8$ typically. This is used to prevent division by 0 without affecting results in a noticeable way. This process is known as **numerical stability**.

1.2 Adam

- Adam = RMSProp + Momentum = "Adaptive Moments"
  - 1st moment = mean (Momentum)
  - 2nd moment = variance (RMSProp)
- Combines the best of both worlds
- Works well with almost all architectures
  - ANN
  - CNN
  - RNN
  - etc...

Adam's update equations:

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1)\nabla C(w)$$
$$v_{t+1} = \beta + 2v + t + (1 - \beta_2)\nabla C(w)^2$$

- C = cost $\equiv L$ = loss
- m = mean (Momentum)
- v = **variance** (NOT velocity)

$$\longrightarrow w_{t+1} = w_t - \alpha \frac{m_{t+1}}{\sqrt{v_{t+1}} + \epsilon}$$

With initializations:

- v = 0
- m = 0
- w = 0 (Arbitrarily chosen; can be random)

We can make some slight changes to account for bias:

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1-\beta_1^t}$$

- at $m_1 \to 1 - \beta_1^1 \neq 0 \approx 0.999...$

$$\hookrightarrow \hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_2^t}$$

$$\hookrightarrow \boxed{w_{t+1} = w_t - \alpha \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}} + \epsilon}}$$

We now have our new bias corrected equation made up of the **bias-correction terms** $\hat{m}_{t+1}$ and $\hat{v}_{t+1}$

2 Calculus Review

2.1 Differential Calculus

Differential calculus is concerned with the study of how quantities change. A central theme is understanding how a function changes with respect to its inputs.

$$\begin{tikzcd} x \arrow[r] & w_1 \arrow[r] & z_1 \arrow[r] & w_2 \arrow[r] & z_2 \arrow[r] & y \end{tikzcd}$$

This diagram represents a simple computational graph. Variables x, w_1, w_2, etc., flow through functions and operations to compute an output y.

2.2 Goal: Minimize Cost

We aim to minimize some cost function, such as the squared loss:

$$\text{Loss} = (y - \hat{y})^2$$

2.3 Change and Derivatives

To achieve this, we need to understand how the output changes with respect to changes in the input — this is captured by the derivative:

$$\frac{d}{dx}$$

The derivative tells us how quantities flow and change with respect to one another.

2.4 Single Variable Case

In the single variable case, the derivative is:

$$\frac{dz}{dx}$$

This helps in understanding how a small change in x affects z.

2.5 The Chain Rule

The chain rule is a fundamental rule in differential calculus that allows us to compute the derivative of a composition of functions.

If we have two functions f and g, then:

$$(f \circ g)'(x) = f'(g(x)) \cdot g'(x)$$

This is essential for calculating derivatives when functions are composed (e.g., layers in a neural network).

2.5.1 Function Composition

$$F(x) = f(g(h(x)))$$

Applying the chain rule:

$$\frac{dF}{dx} = f'(g(h(x))) \cdot g'(h(x)) \cdot h'(x)$$

This generalizes to many layers of nested functions.

2.6 Neural Network Example

Assume a simple feedforward computation:

$$\begin{aligned} x &= \text{input} \\ z_1 &= w_1 x \\ z_2 &= w_2 z_1 \\ y &= f(z_2) \end{aligned}$$

If our loss is based on y, then we want to compute:

$$\frac{d\mathcal{L}}{dw_2} = \frac{d\mathcal{L}}{dy} \cdot \frac{dy}{dz_2} \cdot \frac{dz_2}{dw_2}$$

This shows how gradients ”flow backward” through the network using the chain rule, which is the core idea behind **backpropagation**.

2.7 Example: Applying the Chain Rule to a Composition

Let:

$$\mathcal{L} = \mathcal{L}(z^{(4)}) = 3z^{(4)} = 6$$

Suppose we are interested in computing the derivative with respect to earlier variables using the chain rule.

Assume:

$$z^{(4)} = a^{(3)}w^{(4)} + b^{(4)}$$

Then:

$$\frac{d\mathcal{L}}{dz^{(4)}} = 3, \quad \text{so} \quad \frac{d\mathcal{L}}{da^{(3)}} = \frac{d\mathcal{L}}{dz^{(4)}} \cdot \frac{dz^{(4)}}{da^{(3)}} = 3 \cdot w^{(4)}$$

Example Using a Quadratic Function

If we let:

$$z^{(4)} = (x-3)^2$$

Then:

$$\frac{d\mathcal{L}}{dx} = \frac{d\mathcal{L}}{dz^{(4)}} \cdot \frac{dz^{(4)}}{dx} = 3 \cdot 2(x-3) = 6(x-3)$$

2.8 Backpropagation Flow

The gradient continues backward layer by layer, following the chain rule. Each step requires computing:

$$\frac{d\mathcal{L}}{dz^{(l)}} = \frac{d\mathcal{L}}{dz^{(l+1)}} \cdot \frac{dz^{(l+1)}}{dz^{(l)}}$$

This recursive application forms the core of backpropagation in neural networks.

In the next section, we walk through this process step-by-step using detailed notation and examples.

2.9 Backpropagation: Chain Rule

To understand backpropagation, we rely heavily on the chain rule from calculus.

Chain Rule (2 Functions)

$$(f \circ g)'(x) = f'(g(x)) \cdot g'(x)$$

For a composition of three functions:

$$[f \circ g \circ h](x) = f'(g(h(x))) \cdot g'(h(x)) \cdot h'(x)$$

Using Leibniz notation (derivative of y with respect to x):

$$\frac{dy}{dx} = \frac{dy}{dz} \cdot \frac{dz}{dw} \cdot \frac{dw}{dx}$$

Alternatively, in Newton notation: $\dot{y}$ denotes the derivative with respect to time.

Example Setup

Let:

$$\begin{aligned}
w_0 &= x \\
w_1 &= h(w_0) \\
w_2 &= g(w_1) \\
w_3 &= f(w_2) = f(g(h(x))) = y
\end{aligned}$$

We are interested in computing:

$$\frac{dy}{dx}$$

To do so, we take derivatives step-by-step:

$$\frac{dy}{dx} = \frac{dy}{dw_2} \cdot \frac{dw_2}{dw_1} \cdot \frac{dw_1}{dx}$$

3 Backpropagation Flow

Backpropagation Chain

In a typical neural network layer:

$$z^{(l)} = a^{(l-1)}w^{(l)} + b^{(l)}$$

$$a^{(l)} = \sigma(z^{(l)})$$

Where: - σ is an activation function (e.g., sigmoid or ReLU). - $a^{(l)}$ is the activation at layer l. - $z^{(l)}$ is the linear combination before activation.

Gradient Flow

From the output layer backward, we compute:

$$\frac{\partial C}{\partial a^{(L)}} = 2(a^{(L)} - y)$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)}), \quad \frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}$$

Using the chain rule:

$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}$$

This tells us how a small change in a weight affects the overall cost. This is the essence of **backpropagation**.

References