Adam

Radek Bartyzal

MI-SZI

TBD

Training of neural nets

Goal

Set weights w and biases b in such way that the network approximates the desired outputs $y(x)$ for every input x.

Training of neural nets

Goal

Set weights w and biases b in such way that the network approximates the desired outputs $y(x)$ for every input x.

How to get there?

1. choose a function that tells you the error of the network = cost function
2. minimize the cost function

Training of neural nets

Goal

Set weights w and biases b in such way that the network approximates the desired outputs $y(x)$ for every input x.

How to get there?

1. choose a function that tells you the error of the network = cost function
2. minimize the cost function

Cost function

Quantifies how well the network approximates the desired outputs $y(x)$ for every input x.

Training 1: Cost function

= loss function = objective function.
Quantifies how well the network approximates the desired outputs $y(x)$ for every input x.

Mean Squared Error (MSE)

w = weights

b = biases

n = number of inputs

x = one input

a = output of network for x

$y(x)$ = desired output for x

$$C(w, b) = \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

Training 1: Cost function

Our goal is to minimize the cost function = move w and b in a direction that lowers the value of the cost function.

Why Mean Squared Error ?

- **smooth function of weights and biases** - even small changes of w or b result in a change of the function value
- **easily derivable** - we need derivation of the cost function to calculate the direction in which we should change the w and b

Training 2: Gradient descent

Gradient

Vector = direction in which the function increases the most in value.

Training 2: Gradient descent

Gradient

Vector = direction in which the function increases the most in value.

Gradient descent

1. calculate gradient of the cost function
2. take step in opposite direction = change w and b in a way that lowers the value of the cost function

Training 2: Gradient descent

Gradient

Vector = direction in which the function increases the most in value.

Gradient descent

1. calculate gradient of the cost function
2. take step in opposite direction = change w and b in a way that lowers the value of the cost function

Learning rate = η

How large is the step we take.

Training 3: Backpropagation

Backpropagation algorithm

Efficient way of calculating the gradient of the cost function with respect to any neuron = to any weight or bias.

Tells us in which direction should we move the weights and biases to reduce the error.

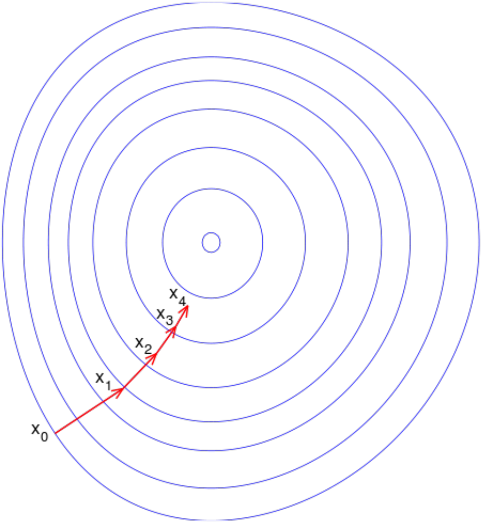
Works by propagating the error back from the output layer to the input layer.

Mini-Batch Gradient Descent

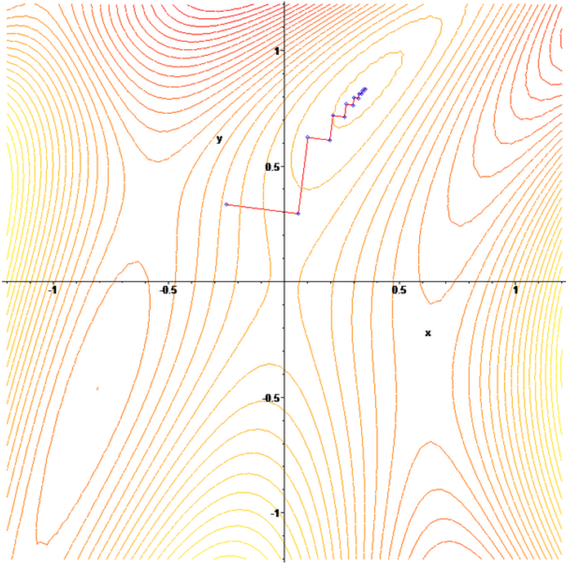
Algorithm 1: Mini-batch Stochastic Gradient Descent

$\theta = \theta_0$
for $t \leftarrow 1$ **to** T **do**
    shuffle(*dataset*)
    foreach *mini_batch* $\in$ *dataset* **do**
        $g = 0$
        foreach $i \in$ *mini_batch* **do**
            $g = g + \nabla L(\hat{y}_i, f(\theta, x_i))$
        $g = \frac{1}{B} g$
        $\theta = \theta - \eta g$

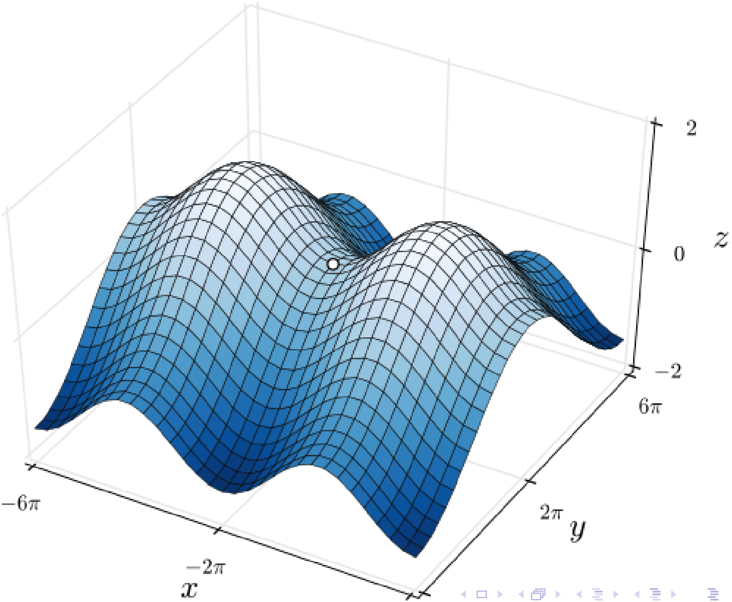
Visualization of Gradient Descent



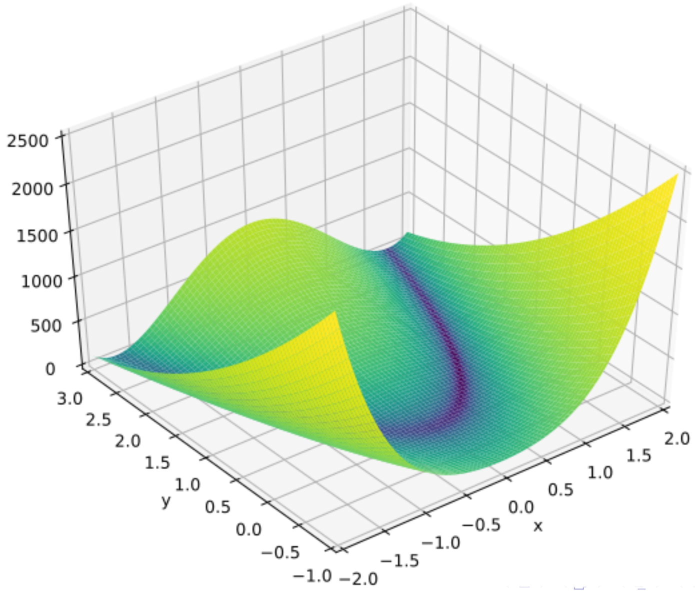
More realistic space



Saddle point



Rosenbrock function = long descending valley



Problems of SGD

- same learning rate for each parameter $\implies$ Adaptive Gradient:

$$\theta_{t+1,i}^{AdaGrad} = \theta_{t,i} - \frac{\eta}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2} + \epsilon} g_{t,i} \tag{1}$$

- slow traversal of saddle points $\implies$ momentum [1]:

$$\begin{aligned} g_t &= \gamma g_{t-1} + \eta \nabla_\theta L(\theta) \\ \theta &= \theta - g_t \end{aligned} \tag{2}$$

Momentum

Figure: Without momentum [1].

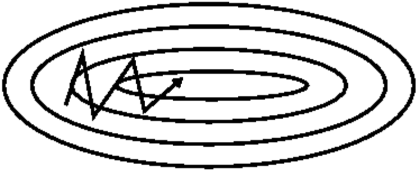


Figure: With momentum [1].

Adam

First and second order moments:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \tag{3}$$

Bias corrected:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{(1-\beta_1^t)} \\ \hat{v}_t &= \frac{v_t}{(1-\beta_2^t)} \end{aligned} \tag{4}$$

Parameter update:

$$\theta_{t+1}^{Adam} = \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t \tag{5}$$

Sources

1. Ruder, Sebastian. "An overview of gradient descent optimization algorithms." arXiv preprint arXiv:1609.04747 (2016).