Regularization
Gradient Descent with Momentum
RMSprop
Adam

13th December 2021

Overview

1. Regularization
2. Mini-batch gradient descent
3. Gradient descent with momentum
4. RMSprop
5. Adam

Regularization

What is regularization and why do we need it?

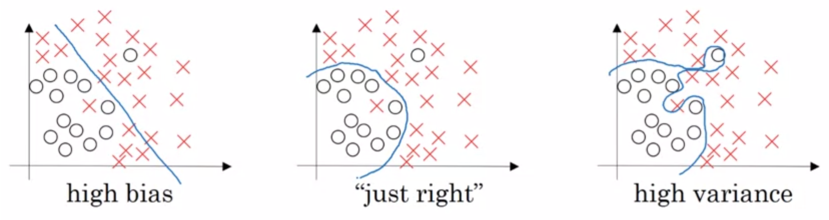


Figure: Bias-Variance [1]

[1] `https://www.coursera.org/learn/deep-neural-network`

Regularization - Logistic Regression

$$
\begin{aligned}
z^{(i)} &= w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \ldots + w_d x_d^{(i)} \\
a^{(i)} &= \sigma(z^{(i)})
\end{aligned}
$$

L2 regularization:

$$\min_{\mathbf{w}} L2(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} L^{(i)}(a^{(i)}, y^{(i)}) + \frac{\lambda}{2N} \|\mathbf{w}\|_2^2$$

where $\|\mathbf{w}\|_2^2 = \sum_{j=1}^{d} w_j^2$.

L1 regularization:

$$\min_{\mathbf{w}} L1(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} L^{(i)}(a^{(i)}, y^{(i)}) + \frac{\lambda}{2N} \|\mathbf{w}\|_1$$

where $\|\mathbf{w}\|_1 = \sum_{j=1}^{d} |w_j|$.
Note: We can omit regularization on w_0.

Regularization - Neural Network

$$a^{(i)} = f^{[K]}(\mathbf{W}^{[K]}(\dots(f^{[2]}(\mathbf{W}^{[2]}(f^{[1]}(\mathbf{W}^{[1]}\mathbf{x}^{(i)}))))))$$

L2 regularization:

$$\min_{\mathbf{w}} L2(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} L^{(i)}(a^{(i)}, y^{(i)}) + \frac{\lambda}{2N}\sum_{k=1}^{K}\left\|\mathbf{W}^{[k]}\right\|_F^2$$

where $\left\|\mathbf{W}^{[k]}\right\|_2^2 = \sum_{i=1}^{n^{[k]}}\sum_{j=1}^{n^{[k-1]}}(w_{ij}^{[k]})^2$ is the Frobenius norm of the weight matrix in layer k.

L1 regularization:

$$\min_{\mathbf{w}} L1(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} L^{(i)}(a^{(i)}, y^{(i)}) + \frac{\lambda}{2N}\sum_{k=1}^{K}\left\|\mathbf{W}^{[k]}\right\|_1$$

where $\left\|\mathbf{W}^{[k]}\right\|_1 = \sum_{i=1}^{n^{[k]}}\sum_{j=1}^{n^{[k-1]}}|w_{ij}^{[k]}|$.

Note: We can omit regularization on $w_{i0}^{[k]}$.

Regularization - Neural Network

$\nabla L(\mathbf{W^{[k]}})$ is the gradient of the loss function without regularization obtained from backpropagation.

The gradient of the loss function with L2 regularization can be computed to update the weight matrix of layer k as:

$$
\begin{aligned}
\nabla L2(\mathbf{W^{[k]}}) &\longleftarrow \nabla L(\mathbf{W^{[k]}}) + \frac{\lambda}{N}\mathbf{W^{[k]}} \\
\mathbf{W^{[k]}} &\longleftarrow \mathbf{W^{[k]}} - \gamma \nabla L2(\mathbf{W^{[k]}}) \\
\mathbf{W^{[k]}} &\longleftarrow \mathbf{W^{[k]}} - \gamma \left(\nabla L(\mathbf{W^{[k]}}) + \frac{\lambda}{N}\mathbf{W^{[k]}} \right) \\
\mathbf{W^{[k]}} &\longleftarrow \left(1 - \frac{\gamma\lambda}{N}\right)\mathbf{W^{[k]}} - \gamma \nabla L(\mathbf{W^{[k]}})
\end{aligned}
$$

Gradient descent for L-layer neural network

Result: weights $\mathbf{W}^{[k]}$ for all layers.

for $k \leftarrow 1$ **to** L **do**

  $\mathbf{W}^{[k]} \longleftarrow \text{random}()$ *[Random initialization]*

end

for $t \leftarrow 1$ **to** $max\_iterations$ **do**

  for $k \leftarrow 1$ **to** L **do**

    $\mathbf{A^{[k]}} \longleftarrow f(\mathbf{Z^{[k]}}) \longleftarrow f(\mathbf{W}^{[k]} \cdot \mathbf{A}^{[k-1]})$ *[Forward propagation]*

  end

  for $k \leftarrow L$ **to** 1 **do**

    $\nabla L(\mathbf{Z^{[k]}}) \longleftarrow \nabla L(\mathbf{A^{[k]}}) \circ f'(\mathbf{Z^{[k]}})$

    $\nabla L(\mathbf{W^{[k]}}) \longleftarrow \frac{1}{N} \nabla L(\mathbf{Z^{[k]}}) \cdot \mathbf{A}^{[k-1]T}$ *[Backpropagation]*

    $\nabla L(\mathbf{A^{[k-1]}}) \longleftarrow \mathbf{W^{[k]T}} \cdot \nabla L(\mathbf{Z^{[k]}})$

  end

  for $k \leftarrow 1$ **to** L **do**

    $\mathbf{W}^{[k]} \longleftarrow \mathbf{W}^{[k]} - \gamma \nabla L(\mathbf{W^{[k]}})$ *[Update parameters]*

  end

end

Batch gradient descent

If the training set is too large, it takes a lot of time to process the whole training set for each step of gradient descent.

Mini-batch gradient descent

If the training set is too large, it takes a lot of time to process the whole training set for each step of gradient descent.
$\longrightarrow$ Divide the training set into multiple mini-batches of **reasonable** sizes.

$$
\begin{aligned}
\mathbf{X} &= [\mathbf{x}^{(1)} \quad \mathbf{x}^{(2)} \dots \mathbf{x}^{(s)} \quad | \quad \mathbf{x}^{(s+1)} \dots \mathbf{x}^{(2s)} \quad | \quad \mathbf{x}^{(2s+1)} \dots \quad | \quad \dots \mathbf{x}^{(N)}] \\
&= [\mathbf{X}_{\{1\}} \quad | \quad \mathbf{X}_{\{2\}} \quad | \quad \dots | \quad \mathbf{X}_{\{N/s\}}] \\
\mathbf{Y} &= [y^{1} \quad y^{2} \dots y^{s} \quad | \quad y^{s+1} \dots y^{2s} \quad | \quad y^{2s+1} \dots \quad | \quad \dots y^{N}] \\
&= [\mathbf{Y}_{\{1\}} \quad | \quad \mathbf{Y}_{\{2\}} \quad | \quad \dots | \quad \mathbf{Y}_{\{N/s\}}]
\end{aligned}
$$

At each iteration t, perform gradient descent using one mini-batch of training examples $(\mathbf{X}_{\{t\}}; \mathbf{Y}_{\{t\}})$.

Mini-batch gradient descent

- Mini-batch size $s = N \longrightarrow$ Normal gradient descent.
- Mini-batch size $s = 1 \longrightarrow$ Stochastic gradient descent.
- Mini-batch sizes are normally set as powers of 2, e.g., $s = 64, 128, 256, 512$

Problem with gradient descent

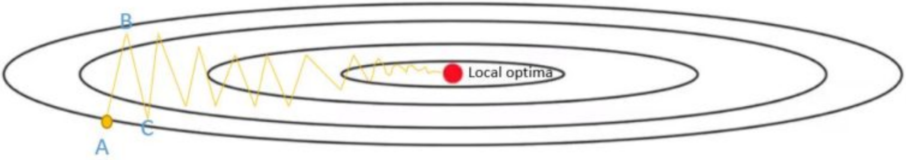


Figure: Gradient descent [2]

[2] https://engmrk.com/gradient-descent-with-momentum/.

Problem with gradient descent

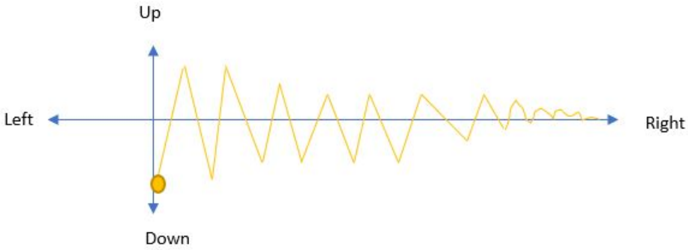
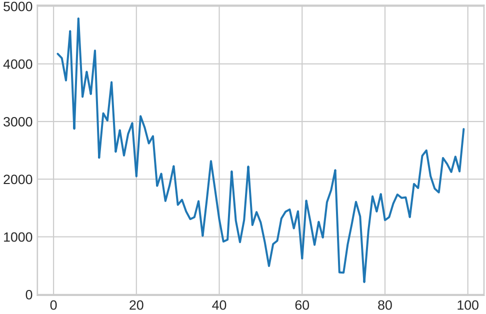


Figure: Gradient descent [3]

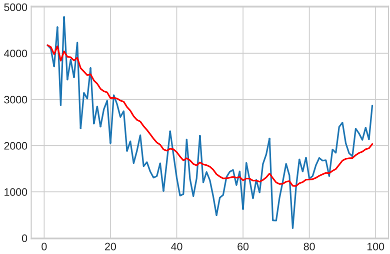
[3] https://engmrk.com/gradient-descent-with-momentum/.

Exponentially weighted moving average

Example: How to model/forecast the stock/share price of a certain company?

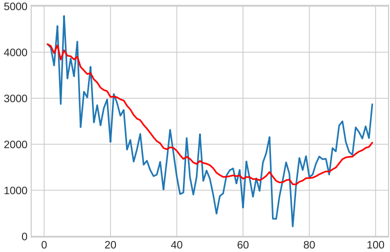


Exponentially weighted moving average



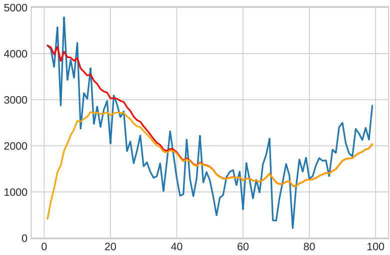
$$m_{100} = \beta m_{99} + (1-\beta) price_{100}$$
$$m_{99} = \beta m_{98} + (1-\beta) price_{99}$$
$$\dots$$
$$m_t = \beta m_{t-1} + (1-\beta) price_t$$
$$\dots$$
$$m_1 = \beta m_0 + (1-\beta) price_1$$

Exponentially weighted moving average



$$
\begin{aligned}
\beta &= 0.9 \\
m_{100} &= 0.9 m_{99} + (1 - 0.9) price_{100} \\
&= 0.1 price_{100} + 0.9 m_{99} \\
&= 0.1 price_{100} + 0.9 (0.1 price_{99} + 0.9 m_{98}) \\
&= 0.1 price_{100} + 0.9 \cdot 0.1 price_{99} + (0.9)^2 m_{98} \\
&= 0.1 price_{100} + 0.09 price_{99} + (0.9)^2 (0.1 price_{98} + 0.9 m_{97}) \\
&= 0.1 price_{100} + 0.09 price_{99} + 0.081 price_{98} + (0.9)^3 m_{97} \\
&= \dots
\end{aligned}
$$

Exponentially weighted moving average



Actually, without the below correction, we will get the orange line instead of the red line. Why?

$$m_t = \beta m_{t-1} + (1-\beta) price_t$$
$$m_t^{corrected} = \frac{m_t}{1-\beta^t}$$

Gradient descent with momentum

Update weights at iteration t:

$$
\begin{aligned}
\mathbf{g}_{\{t\}} &\longleftarrow \nabla L(\mathbf{W}_{\{t-1\}}) \\
\mathbf{m}_{\{t\}} &\longleftarrow \beta_1 \mathbf{m}_{\{t-1\}} + (1-\beta_1)\mathbf{g}_{\{t\}} \\
\mathbf{W}_{\{t\}} &\longleftarrow \mathbf{W}_{\{t-1\}} - \gamma \mathbf{m}_{\{t\}}
\end{aligned}
$$

Initialization: $\mathbf{m}_{\{0\}} = \mathbf{0}$

Hyperparameters: $\beta_1 = 0.9$; learning rate γ

Gradient descent with momentum

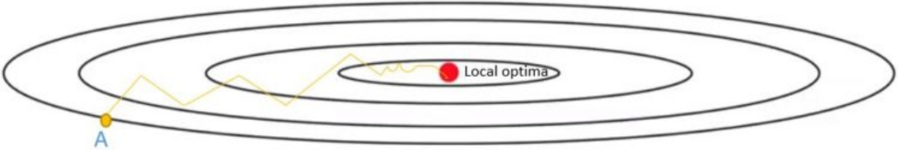


Figure: Gradient descent with momentum [4]

[4] https://engmrk.com/gradient-descent-with-momentum/.

Problem with gradient descent

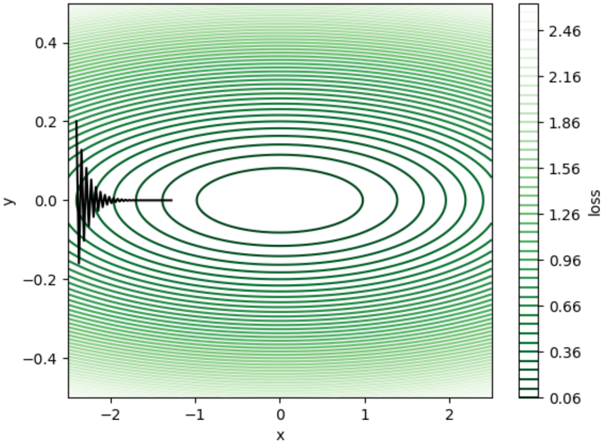


Figure: Gradient descent [5]

[5] https://github.com/hengluchang/visualizing_momentum.

Gradient descent with momentum

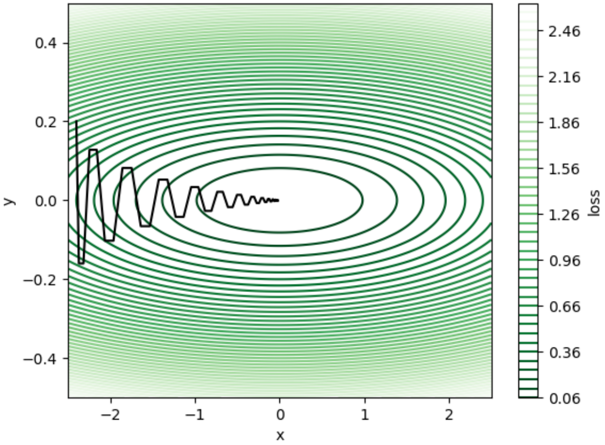


Figure: Gradient descent with momentum $\beta_1 = 0.8$ [6]

[6] `https://github.com/hengluchang/visualizing_momentum`.

RMSprop

Update weights at iteration t:

$$\begin{aligned}
\mathbf{g}_{\{t\}} &\longleftarrow \nabla L(\mathbf{W}_{\{t-1\}}) \\
\mathbf{s}_{\{t\}} &\longleftarrow \beta_2 \mathbf{s}_{\{t-1\}} + (1-\beta_2)\mathbf{g}_t^2 \\
\mathbf{W}_{\{t\}} &\longleftarrow \mathbf{W}_{\{t-1\}} - \gamma \frac{\mathbf{g}_{\{t\}}}{\sqrt{\mathbf{s}_{\{t\}}} + \epsilon}
\end{aligned}$$

Initialization: $\mathbf{s}_{\{0\}} = \mathbf{0}$

Hyperparameters: $\beta_2 = 0.9$; $\epsilon = 10^{-8}$; learning rate γ

Adam - Adaptive moment estimation

Update weights at iteration t:

$$
\begin{aligned}
\mathbf{g}_{\{t\}} &\longleftarrow \nabla L(\mathbf{W}_{\{t-1\}}) \\
\mathbf{m}_{\{t\}} &\longleftarrow \beta_1 \mathbf{m}_{\{t-1\}} + (1-\beta_1)\mathbf{g}_{\{t\}} \\
\mathbf{s}_{\{t\}} &\longleftarrow \beta_2 \mathbf{s}_{\{t-1\}} + (1-\beta_2)\mathbf{g}_t^2 \\
\mathbf{m}_{\{t\}}^{corrected} &\longleftarrow \frac{\mathbf{m}_{\{t\}}}{1-\beta_1^t} \\
\mathbf{s}_{\{t\}}^{corrected} &\longleftarrow \frac{\mathbf{s}_{\{t\}}}{1-\beta_2^t} \\
\mathbf{W}_{\{t\}} &\longleftarrow \mathbf{W}_{\{t-1\}} - \gamma \frac{\mathbf{m}_{\{t\}}^{corrected}}{\sqrt{\mathbf{s}_{\{t\}}^{corrected}} + \epsilon}
\end{aligned}
$$

Initialization: $\mathbf{m}_{\{0\}} = \mathbf{s}_{\{0\}} = \mathbf{0}$

Hyperparameters: $\beta_1 = 0.9$; $\beta_2 = 0.999$; $\epsilon = 10^{-8}$; learning rate γ

References

1. Visualizing Gradient Descent with Momentum in Python: `https://github.com/hengluchang/visualizing_momentum`.
2. Coursera - Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization: `https://www.coursera.org/learn/deep-neural-network/`
3. Coursera - Neural Networks and Deep Learning: `https://www.coursera.org/learn/neural-networks-deep-learning/`