Optimizers in Deep Learning

Atul Balaji (ab5246)

Domain: Neural Networks (Deep Learning)

Topic: Exploring different Optimizers for training neural networks

Expert: Atul Balaji. I have taken courses on machine learning and deep learning and worked on projects in these areas in my undergraduate and graduate studies. Some other members of the team are also familiar with this topic as they are in the Machine Learning track.

User: Sairam Haribabu. He is a MS CS student at Columbia. He is interested in working on machine learning projects and has started to code an emotion classification model. He observed that there are many different optimizers available in the framework, but is not sure which to use. So, he wants to learn the difference between them and the advantages and disadvantages of each of them.

Media: We can have 2-3 videos (total of 5-7 mins) covering different optimizers like Gradient descent, SGD, Momentum, Adagrad, Adam, etc. We can also have graphs showing the performance of different optimizers on a given test function. Also, the formulae and a few points of text for each approach will be shown to summarize the key ideas learned.

Quiz: The quiz will involve multiple choice questions about the concept behind an optimizer, comparison between different optimizers, their formula and which optimizer should be preferred for a given function. A few (3-5) questions will be asked after each lesson (covering one type of optimizer). After all lessons are complete, we will also have a final quiz combining all concepts learned so far.

Positive piece of feedback: This topic is very relevant to students and has a large user pool.

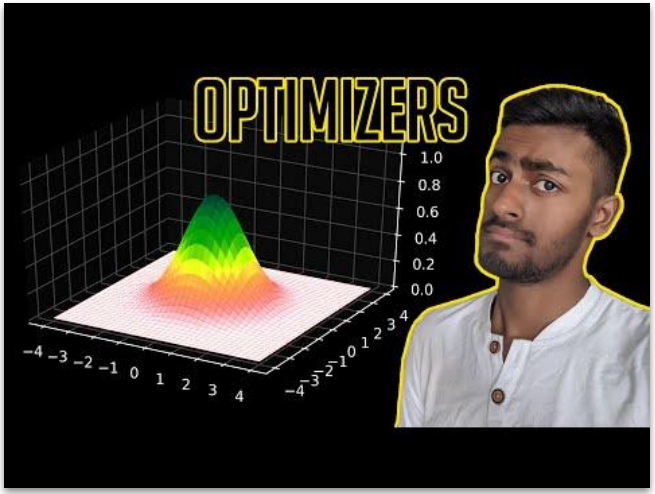
Cautionary piece of feedback: Try not to turn this into a lecture. The three topics (backprop, optimizers and convolution) feel too bloated to successfully teach in 10 minutes, so focus on a singular aspect.

Low Fidelity Prototype

Optimizers in Deep Learning

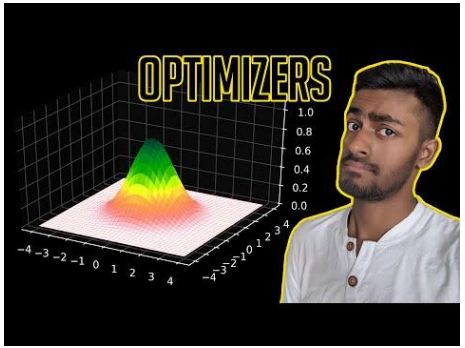
In this site you can learn about some common optimizers used in neural networks and how they work.

Click below to start.



START: Gradient Descent (1/6)

Gradient Descent (GD)



It is the simplest optimizer used in machine learning.

Parameter Update equation:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta).$$

θ: Parameter, η: Learning rate, **J(θ)**: Loss function

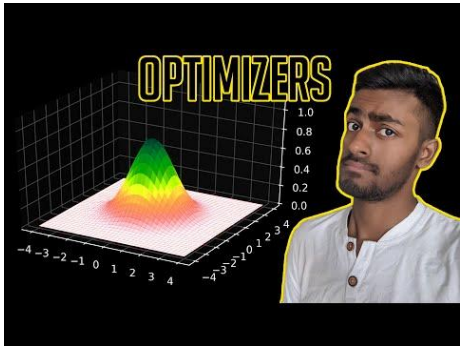
Batch gradient descent: Parameter update happens only once in an epoch (a full pass over the dataset). The gradient is summed up over entire dataset, so the update happens in **large jumps** which leads to poor convergence.

Stochastic gradient descent (SGD): Parameter update happens with every data point. This causes frequent but very **noisy** updates, which again leads to poor convergence.

Minibatch gradient descent: This is midway between the above two approaches. Parameter updates are done after seeing each batch of data, containing N samples (usually 32 or 64). This achieves **better convergence**.

NEXT: Momentum (2/6)

Momentum



Momentum takes past gradients into account to calculate the parameter update. The momentum term increases when the gradients point in the same direction and reduces when the gradients change direction.

Parameter update equation:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

v: Momentum term
γ: Momentum parameter
(usually set as 0.9)

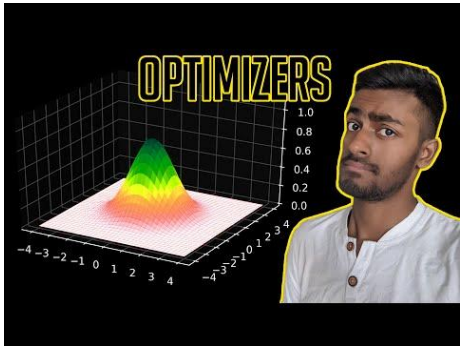
Advantage: It is useful when dealing with multiple dimensions (parameters). Momentum can provide larger updates to dimensions with larger gradients, making convergence faster.

SGD without momentum SGD with momentum

PREVIOUS: Gradient Descent (1/6)

NEXT: Nesterov Momentum (3/6)

Nesterov Momentum (Acceleration)



In momentum without acceleration, the updates keep getting larger, and may overshoot the goal. Nesterov momentum allows us to have a notion of acceleration so that we know when to slow down before the goal is reached, to prevent overshooting.

Parameter update equation:

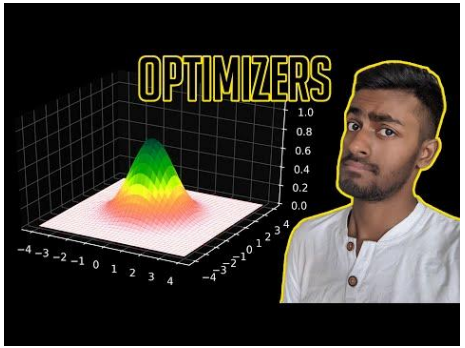
$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

Advantage: The modified parameter value in the loss function serves as a correction to the momentum term and acts as an anticipatory update, which prevents the updates from going too fast and results in better responsiveness.

PREVIOUS: Momentum (2/6)

NEXT: Adagrad (4/6)

Adagrad



Adagrad adapts the learning rate to the parameters, performing smaller updates for parameters associated with frequently occurring features, and larger updates for parameters associated with infrequent features.

Parameter update equation:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \nabla_{\theta_{t,i}} J(\theta_{t,i})$$

$\theta_{t,i}$: Parameter i at epoch t

$G_{t,ii}$: sum of squares of gradients of θ_i up to epoch t

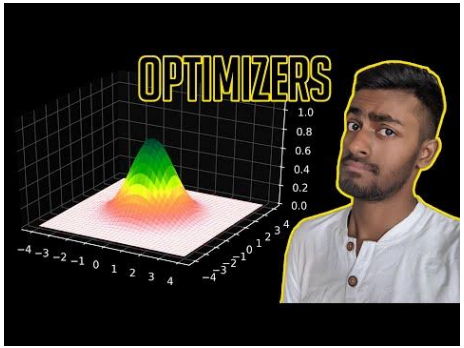
$$G_{t,ii} = \nabla^2_{\theta_{1,i}} J(\theta_{1,i}) + \nabla^2_{\theta_{2,i}} J(\theta_{2,i}) + \cdots + \nabla^2_{\theta_{t,i}} J(\theta_{t,i})$$

Advantage: Due to the adaptive learning rate, it is good at dealing with sparse data, with multiple parameters. No need to manually tune the learning rate.

PREVIOUS: Nesterov Momentum (3/6)

NEXT: Adadelta (5/6)

Adadelta



In Adagrad, the G_t values keep increasing with t, so the learning rate becomes very low with time, making converge slow. In Adadelta, this is mitigated by computing G_t recursively with a discount factor $\gamma < 1$.

Parameter update equation:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \nabla_{\theta_{t,i}} J(\theta_{t,i})$$

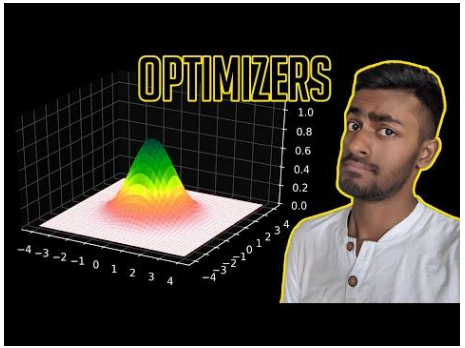
$$G_{t,ii} = \gamma G_{t-1,ii} + (1-\gamma) \nabla^2_{\theta_{t,i}} J(\theta_{t,i})$$

Advantage: Exponentially lower weight is given to gradients from earlier iterations. This leads to faster convergence due to smaller G and large enough learning rates even at large values of t.

PREVIOUS: Adagrad (4/6)

NEXT: Adam (6/6)

Adam (Adaptive Momentum Estimation)



Adam adds a notion of momentum to Adadelta. This momentum is captured by $E[G_{t,ii}]$ which is the expected sum of squares of past gradients. So, as the expected sum rises, momentum builds.

Parameter update equation:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[G_{t,ii}] + \epsilon}} \times \mathrm{E}[g_{t,i}]$$

$g_{t,i}$: Gradient **of** Parameter i at epoch t

β: decay factor (set as 0.9)

$$\mathrm{E}[g_{t,i}] = \beta E[g_{t-1,i}] + (1-\beta)\nabla_{\theta_{t,i}} J(\theta_{t,i})$$

Advantage: It has faster convergence and better accuracy, making it the default optimizer of choice in most problems.

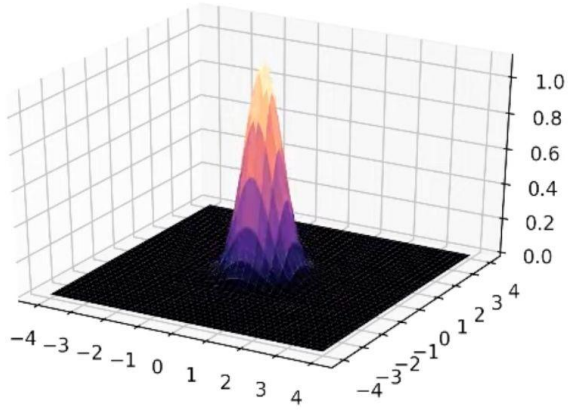
PREVIOUS: Adadelta (5/6)

NEXT: Review

Review

Optimizers

Gradient Descent:	$\theta = \theta - \alpha \,.\nabla_\theta J(\theta)$
Stochastic Gradient Descent	$\theta = \theta - \alpha \,.\nabla_\theta J(\theta; sample)$
Mini-Batch Gradient Descent	$\theta = \theta - \alpha \,.\nabla_\theta J(\theta; N\ samples)$
SGD + Momentum	$v = \gamma.v + \eta.\nabla_\theta J(\theta)$ $\theta = \theta - \alpha\, v$
SGD + Momentum + Acceleration	$v = \gamma.v + \eta.\nabla_\theta J(\theta - \gamma.v)$ $\theta = \theta - \alpha\, v$
Adagrad	$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \nabla_{\theta_{t,i}} J(\theta_{t,i})$
Adadelta	$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[G_{t,ii}] + \epsilon}} \nabla_{\theta_{t,i}} J(\theta_{t,i})$
Adam	$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[G_{t,ii}] + \epsilon}} \times \mathrm{E}[g_{t,i}]$



PREVIOUS: Adam (6/6)

NEXT: Quiz

Quiz

1. Which of the following is a disadvantage of adagrad?

- Learning rate should be manually tuned
- Learning rate decreases to zero quickly
- It makes the same updates to all parameters
- It fails on sparse data

NEXT: (2/5)

Quiz

1. Which of the following is a disadvantage of adagrad?

Correct ★

- Learning rate should be manually tuned
- Learning rate decreases to zero quickly
- It makes the same updates to all parameters
- It fails on sparse data

NEXT: (2/5)

Quiz

2. Select the true statement.

- Momentum with acceleration leads to increased overshooting
- Stochastic gradient descent is better than minibatch gradient descent
- Adadelta produces higher learning rates than Adagrad
- Momentum reduces the updates to dimensions when gradient is steep

NEXT: (3/5)

Quiz

2. Select the true statement.

Wrong ✖

- Momentum with acceleration leads to increased overshooting
- Stochastic gradient descent is better than minibatch gradient descent
- Adadelta produces higher learning rates than Adagrad
- Momentum reduces the updates to dimensions when gradient is steep

NEXT: (3/5)

Quiz

3. Which optimizer does this formula describe?

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

- Adagrad
- Momentum
- Nesterov Momentum
- Adadelta

NEXT: (4/5)

Quiz

4. Which is the right formula for Adam?

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[G_{t,ii}] + \epsilon}} \times \mathrm{E}[g_{t,i}]$$
$$\mathrm{E}[g_{t,i}] = \beta E[g_{t-1,i}] + (1-\beta)\nabla_{\theta_{t,i}} J(\theta_{t,i})$$

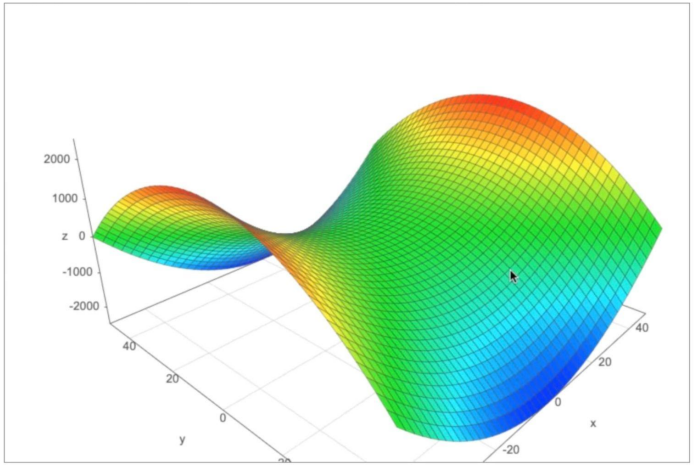
$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}} + \epsilon} \nabla_{\theta_{t,i}} J(\theta_{t,i})$$
$$G_{t,ii} = \gamma G_{t-1,ii} + (1-\gamma)\nabla^2_{\theta_{t,i}} J(\theta_{t,i})$$

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

NEXT: (5/5)

Quiz

5. Which optimizer(s) will converge fast on the function $f(x,y) = x^2 - y^2$ as shown?



- Adagrad
- Momentum
- Nesterov Momentum
- Adadelta

SUBMIT QUIZ

Quiz

Your score is ⅘.

Congratulations!