

18AIC301J: DEEP LEARNING TECHNIQUES

B. Tech in ARTIFICIAL INTELLIGENCE, 5th semester

Faculty: **Dr. Athira Nambiar**

Section: A, slot:D

Venue: TP 804

Academic Year: 2022-22

UNIT-2

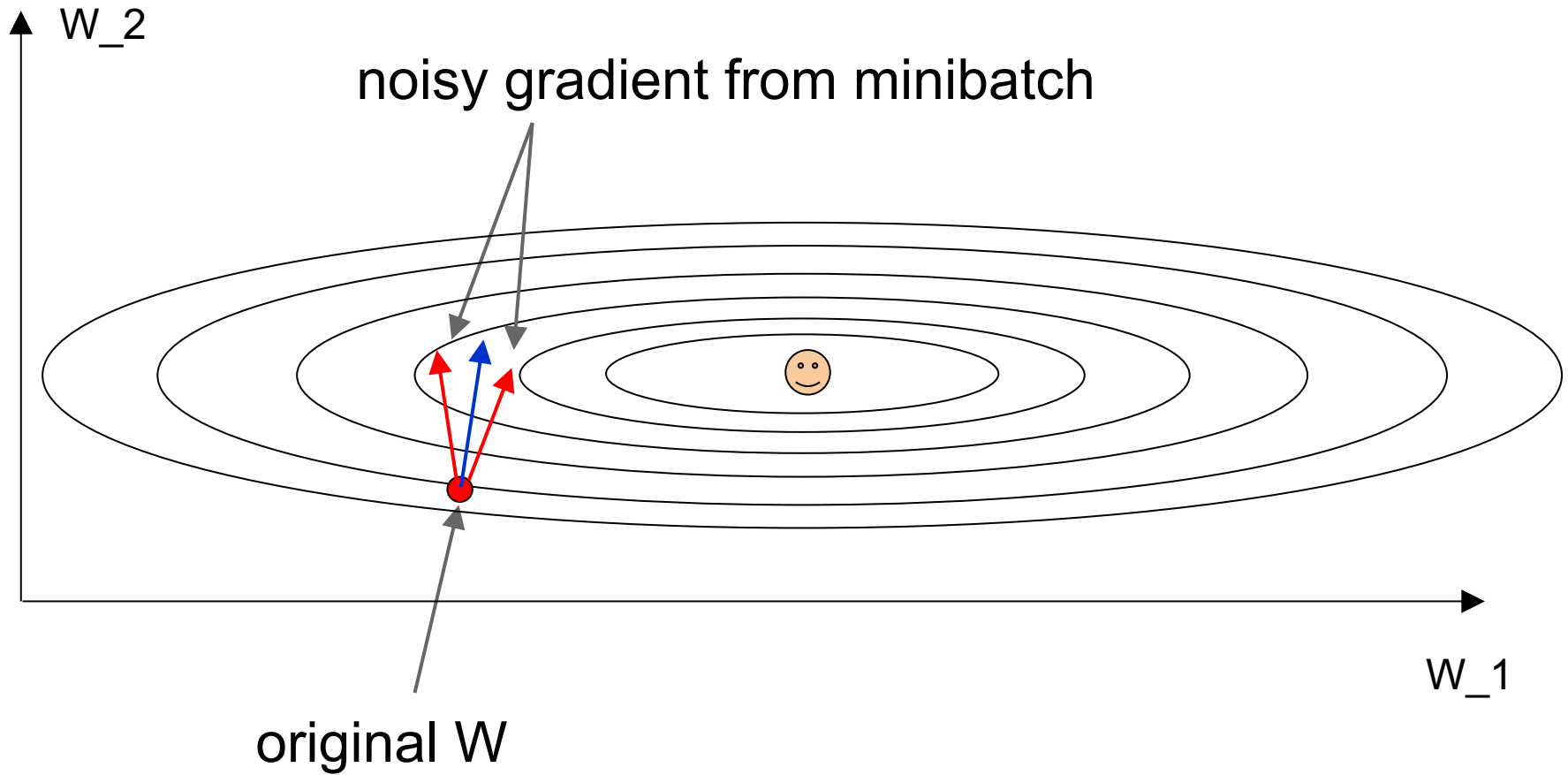
Limitations of gradient descent learning algorithm, Contour maps,
Momentum based gradient descent, Nesterov accelerated gradient Descent, AdaGrad, RMSProp, Adam learning Algorithm, Stochastic gradient descent
Implement linear regression with stochastic gradient descent
Mini-batch gradient descent, Bias Variance tradeoff, Overfitting in deep neural networks,
Hyperparameter tuning, Regularization: L2 regularization, Dataset Augmentation and Early stopping
Implement linear regression with stochastic mini-batch gradient descent and compare the results with previous exercise.
Dimensionality reduction, Principal Component Analysis, Singular value decomposition
Autoencoders, Relation between PCA and Autoencoders, Regularization in Autoencoders
Optimizing neural networks using L2 regularization, Dropout, data augmentation and early stopping.

Challenges of mini-batch SGD

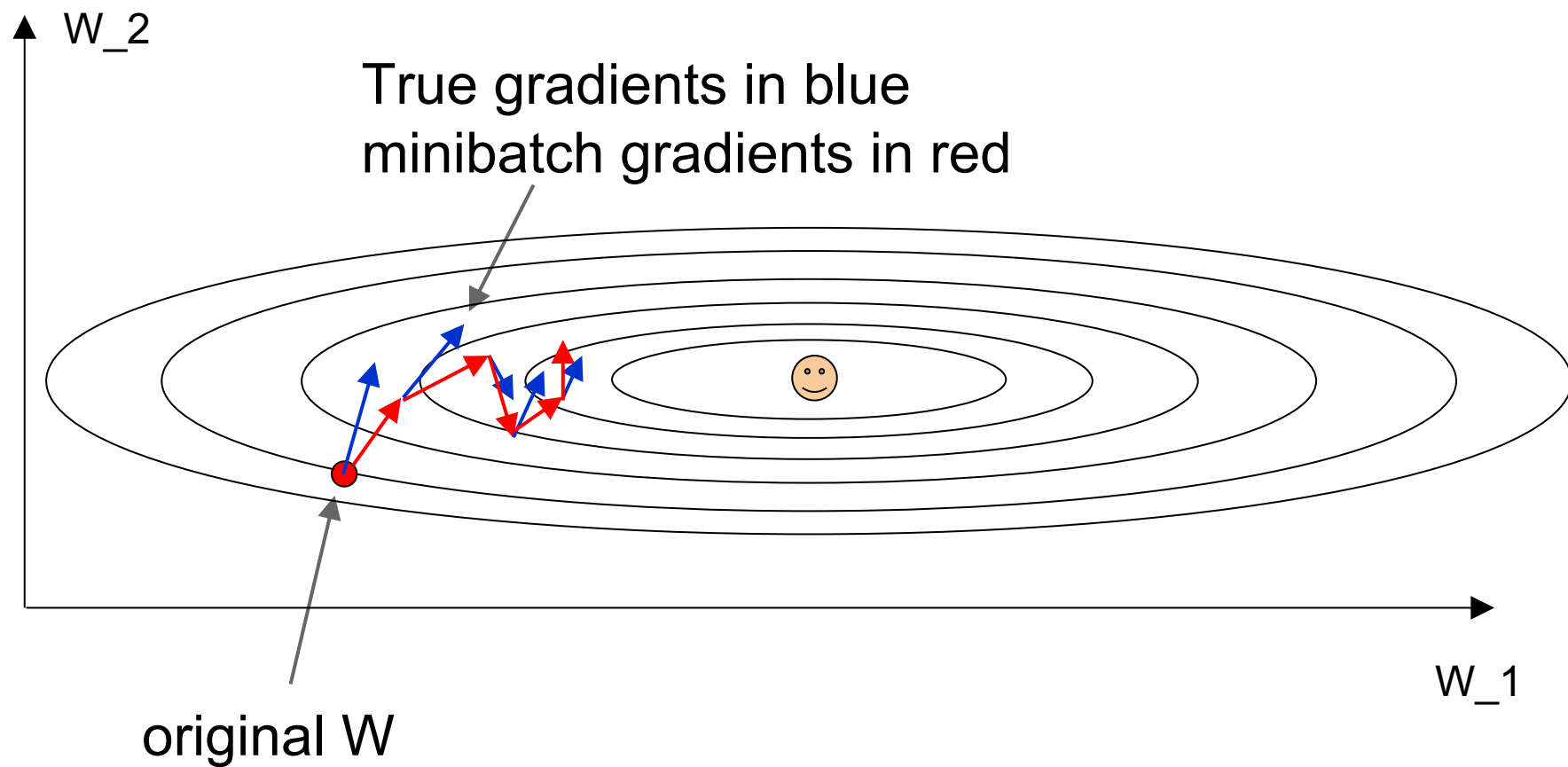
Vanilla mini-batch gradient descent, does not guarantee good convergence, but offers a few challenges that need to be addressed:

- Choosing a proper learning rate can be difficult. A learning rate that is too small leads to painfully slow convergence, while a learning rate that is too large can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge.
- Additionally, the same learning rate applies to all parameter updates. If our data is sparse and our features have very different frequencies, we might not want to update all of them to the same extent, but perform a larger update for rarely occurring features.
- Another key challenge of minimizing highly non-convex error functions common for neural networks is avoiding getting trapped in their numerous suboptimal local minima.

Stochastic Gradient

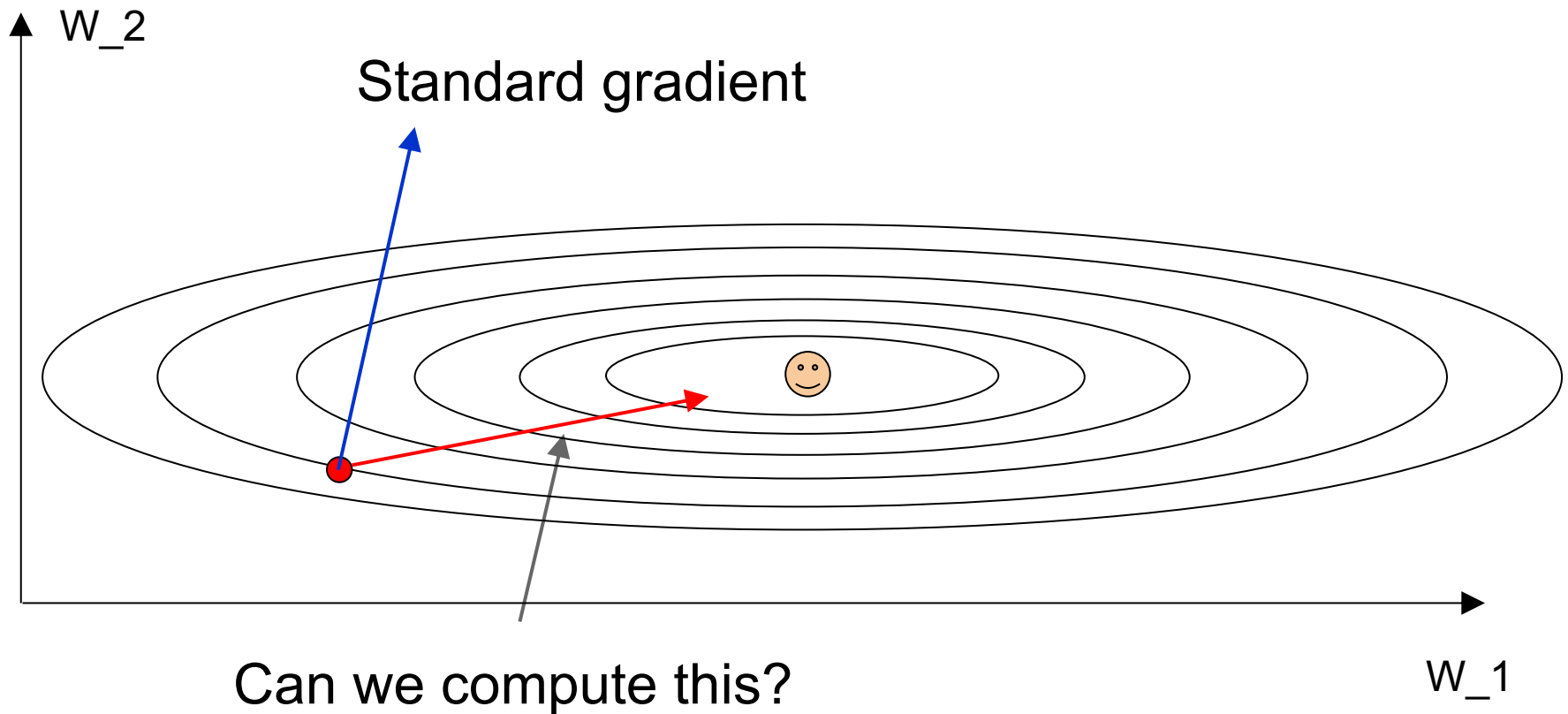


Stochastic Gradient Descent



Gradients are noisy but still make good progress on average

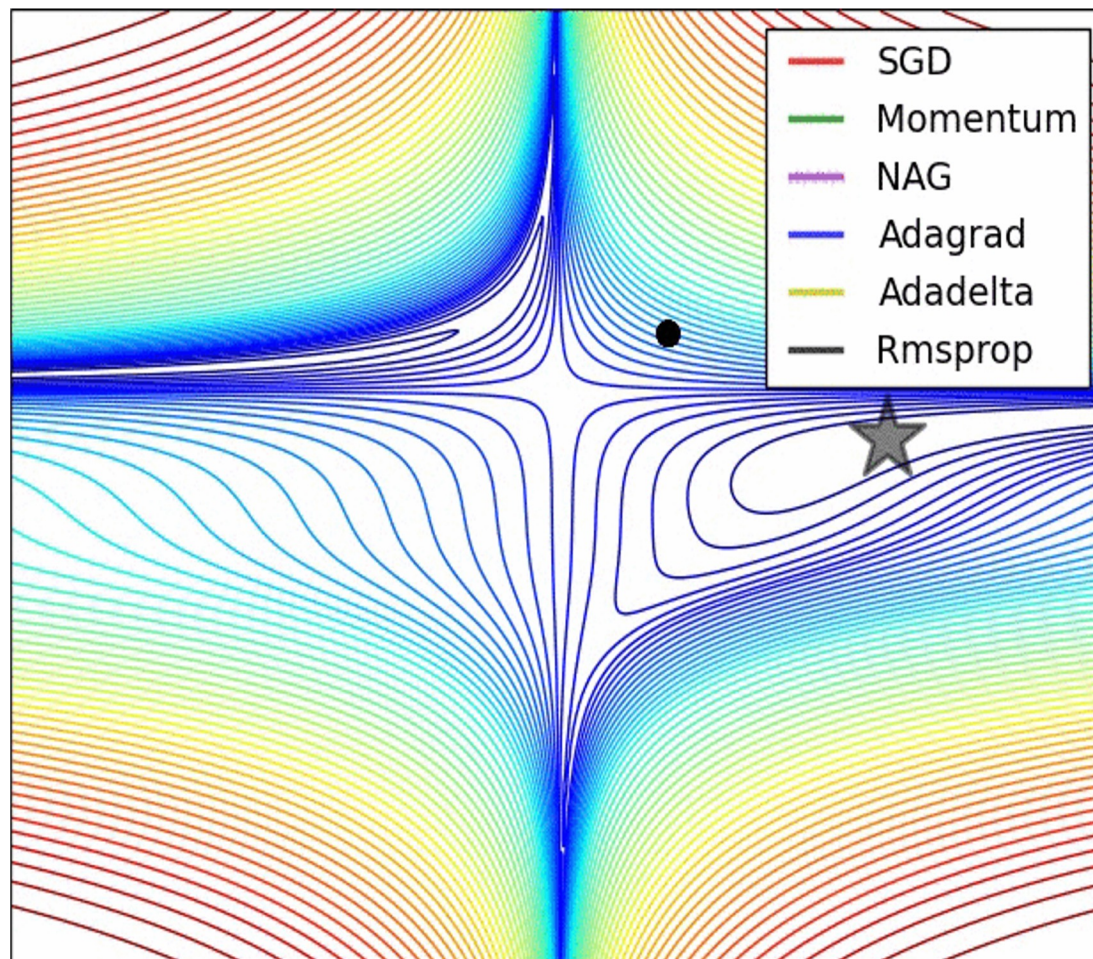
You might be wondering....



Optimizers

- There exist multiple variants of SGD that differ by taking into account previous weight updates when computing the next weight update, rather than just looking at the current value of the gradients.
- There is, for instance, SGD with momentum, as well as Adagrad, RMSProp, and several others.
- Such variants are known as **optimization methods or optimizers**.

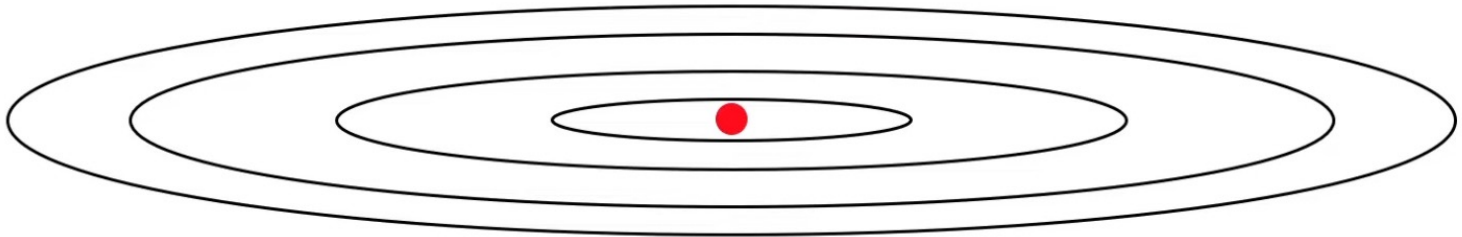
Optimizers



Momentum based gradient descent

Momentum addresses two issues with SGD: convergence speed and local minima.

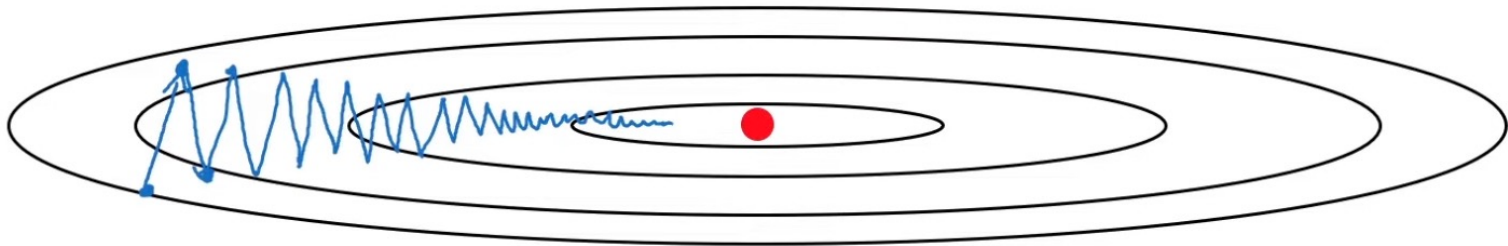
We are trying to optimize a cost function which has contours like this:
(red dot - minimum)



Start a gradient descent and then at each iteration it end up heading to the minimum point.

Momentum based gradient descent

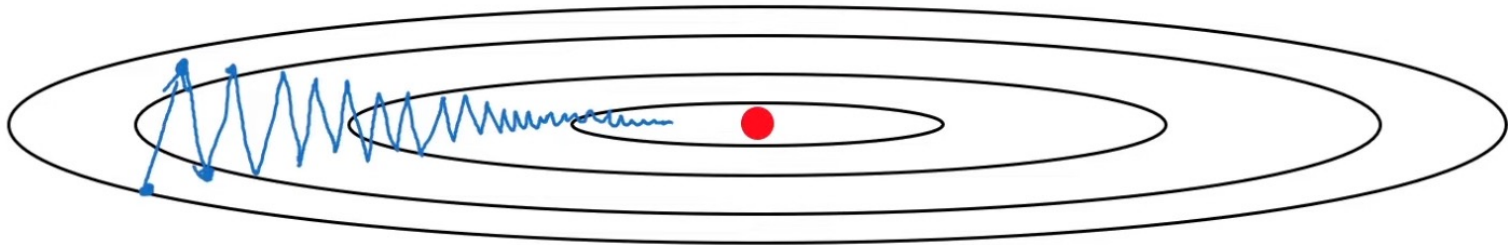
Vertical side of the ellipse..



Gradient descent will take a lot of steps / slowly oscillate towards the minimum.

These up-and-down oscillations slows down gradient learning and prevents from using a much larger learning rate.

Momentum based gradient descent



On the vertical axis, we want the learning to be slower (we don't want the oscillations), but along the horizontal axis, we want faster learning. i.e. we want to aggressively move from left to right to that minimum.

Solution: Gradient descent with momentum

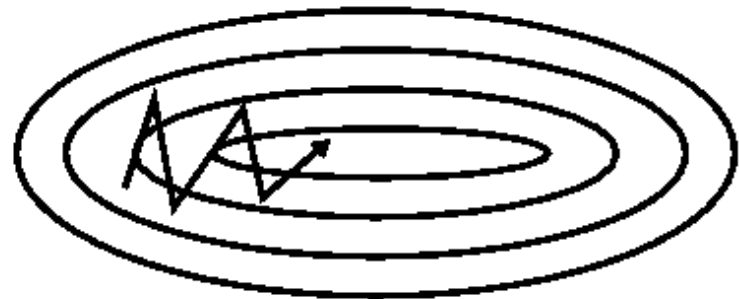
Momentum based gradient descent

Momentum

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another which are common around local optima. In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum



SGD



SGD with momentum

Momentum based gradient descent

The gradient descent with momentum algorithm borrows the idea from physics.

Imagine rolling down a ball inside of a frictionless bowl. Instead of stopping at the bottom, the momentum it has accumulated pushes it forward, and the ball keeps rolling back and forth.

We can apply the concept of momentum to our vanilla gradient descent algorithm. In each step, in addition to the regular gradient, it also adds on the movement from the previous step.

Mathematically, it is commonly expressed as:

$$\begin{aligned} \text{delta} &= - \text{learning_rate} * \text{gradient} + \text{previous_delta} * \text{decay_rate} \\ \text{theta} &+= \text{delta} \end{aligned}$$

Momentum based gradient descent

Momentum is a method that helps accelerate SGD in the relevant direction and dampens oscillations.

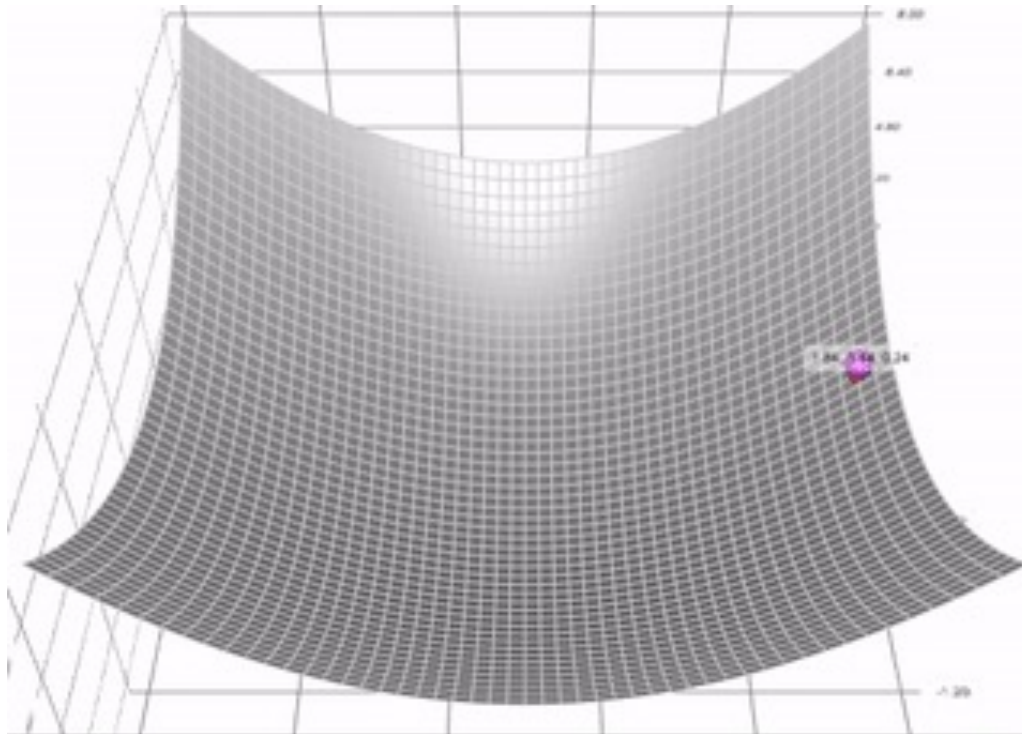
It does this by adding a fraction γ of the update vector of the past time step to the current update vector:

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

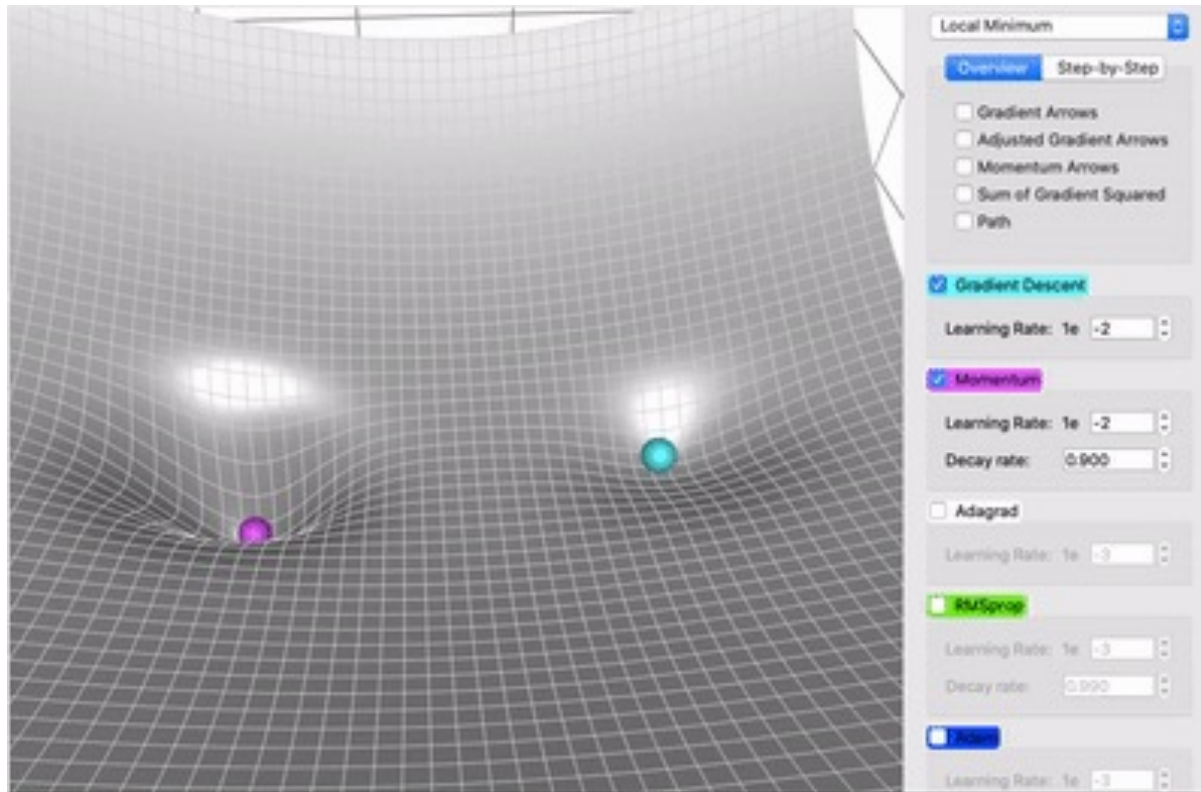
Essentially, when using momentum, we push a ball down a hill. The ball accumulates momentum as it rolls downhill, becoming faster and faster on the way (until it reaches its terminal velocity if there is air resistance, i.e. $\gamma < 1$).

Momentum based gradient descent

Updating the parameter w based not only on the current gradient value but also on the previous parameter update.



Momentum based gradient descent

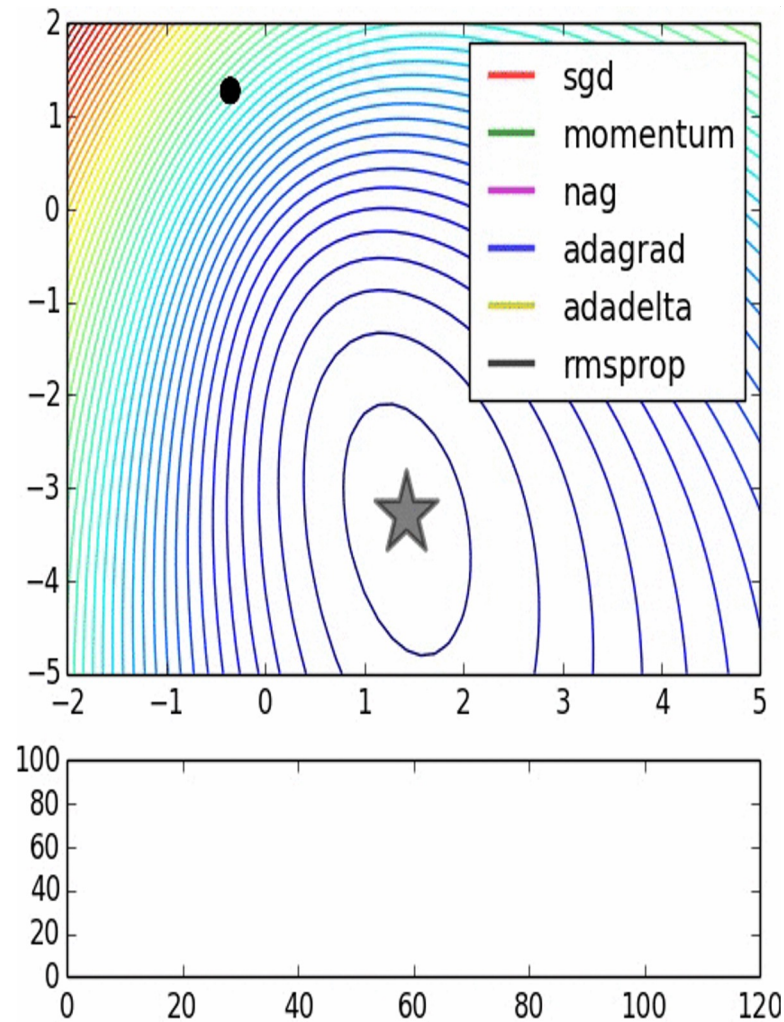


Momentum (magenta) vs. Gradient Descent (cyan) on a surface with a global minimum (the left well) and local minimum (the right well)

<https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c>

Momentum based gradient descent

SGD
vs
Momentum



notice momentum overshooting the target, but overall getting to the minimum much faster than vanilla SGD.

Nesterov accelerated gradient descent

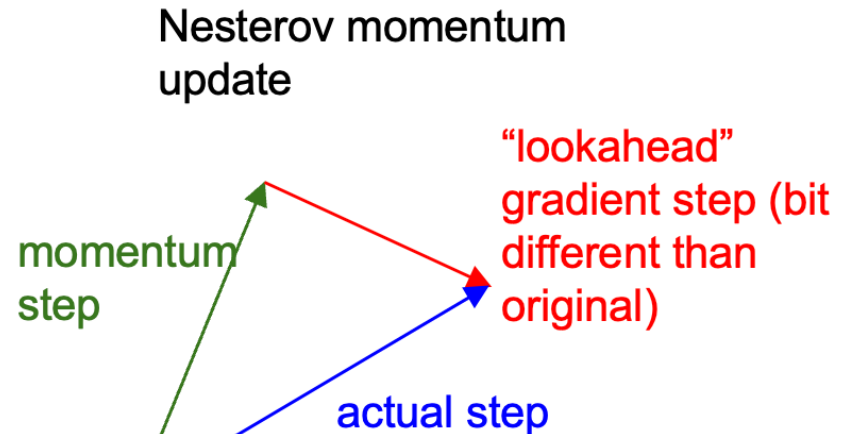
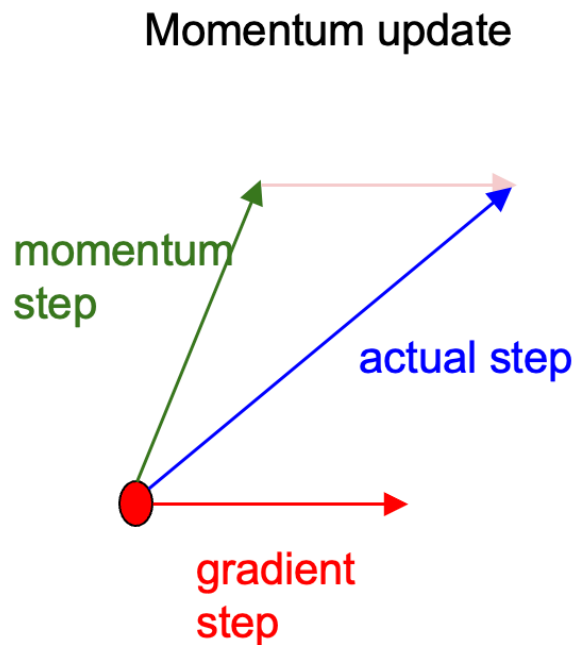
- However, a ball that rolls down a hill, blindly following the slope, is highly unsatisfactory.
- We'd like to have a smarter ball, a ball that has a notion of where it is going so that it knows to slow down before the hill slopes up again.
- Nesterov accelerated gradient (NAG) is a way to give our momentum term this kind of prescience.

Nesterov accelerated gradient descent

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

- We know that we will use our momentum term γv_{t-1} to move the parameters θ .
- Computing $\theta - \gamma v_{t-1}$ thus gives us an approximation of the next position of the parameters (the gradient is missing for the full update), a rough idea where our parameters are going to be.
- Helps to calculate the gradient not w.r.t. to our current parameters θ but w.r.t. the approximate future position of our parameters:

Nesterov accelerated gradient descent



Instead of evaluating gradient at the current position (red circle), we know that our momentum is about to carry us to the tip of the green arrow. With Nesterov momentum, we therefore evaluate the gradient at this “look-ahead” position.

AdaGrad learning algorithms

Adagrad (Adaptive Gradient Algorithm)

- Whatever the optimizer we learned till SGD with momentum, the learning rate remains constant.
- In Adagrad optimizer, there is **no momentum concept** so, it is much simpler compared to SGD with momentum.
- The idea behind Adagrad is **to use different learning rates for each parameter base on iteration**.
- The reason behind the need for different learning rates is that the learning rate for sparse features parameters needs to be higher compare to the dense features parameter because the frequency of occurrence of sparse features is lower.

AdaGrad learning algorithms

$$\text{SGD} \Rightarrow w_t = w_{t-1} - \eta \frac{\partial L}{\partial w_{t-1}}$$

$$\text{Adagrad} \Rightarrow w_t = w_{t-1} - \eta'_t \frac{\partial L}{\partial w_{t-1}}$$

$$\text{where } \eta'_t = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}}$$

ε is a small +ve number to avoid divisibility by 0

$$\alpha_t = \sum_{i=1}^t \left(\frac{\partial L}{\partial w_{t-1}} \right)^2 \quad \text{summation of gradient square}$$

In the Adagrad optimizer equation, the learning rate has been modified in such a way that it will automatically decrease because the summation of the previous gradient square will always keep on increasing after every time step.


AdaGrad learning algorithms

```
# Adagrad update  
cache += dx**2  
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

Added element-wise scaling of the gradient based on the historical sum of squares in each dimension

RMSProp learning algorithms

```
# Adagrad update  
cache += dx**2  
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```



```
# RMSProp  
cache = decay_rate * cache + (1 - decay_rate) * dx**2  
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

RMSprop optimizer adjusts the Adagrad in a simple way in an attempt to reduce its aggressive monotonically decreasing learning rate.

It uses a moving average of squared gradients, instead.

RMSProp learning algorithms

The RMSprop optimizer is similar to the gradient descent algorithm with momentum.

The RMSprop optimizer restricts the oscillations in the vertical direction.

Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster.

The difference between RMSprop and gradient descent is on how the gradients are calculated.

The following slide shows how the gradients are calculated for the RMSprop and gradient descent with momentum.

The value of momentum is denoted by β and is usually set to 0.9.

RMSProp learning algorithms

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw$$

$$v_{db} = \beta \cdot v_{db} + (1 - \beta) \cdot db$$

$$W = W - \alpha \cdot v_{dw}$$

$$b = b - \alpha \cdot v_{db}$$

Gradient descent with momentum

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$

$$v_{db} = \beta \cdot v_{db} + (1 - \beta) \cdot db^2$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

RMSprop optimizer

Adam learning algorithms

```
# Adam
m = beta1*m + (1-beta1)*dx # update first moment
v = beta2*v + (1-beta2)*(dx**2) # update second moment
x += - learning_rate * m / (np.sqrt(v) + 1e-7)
```

```
# Adam
m = beta1*m + (1-beta1)*dx # update first moment
v = beta2*v + (1-beta2)*(dx**2) # update second moment
x += - learning_rate * m / (np.sqrt(v) + 1e-7)
```

momentum

RMSProp-like

Looks a bit like RMSProp with momentum

Summary

- **Momentum:** is another method to produce better effective gradients.
- ADAGRAD, RMSprop diagonally scale the gradient.
- ADAM scales and applies momentum.

Learning Resources

- Charu C. Aggarwal, Neural Networks and Deep Learning, Springer, 2018.
- Eugene Charniak, Introduction to Deep Learning, MIT Press, 2018.
- Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, 2016.
- Michael Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.
- Deng & Yu, Deep Learning: Methods and Applications, Now Publishers, 2013.
- <https://ruder.io/optimizing-gradient-descent/index.html#challenges>
- Notes from Andrej Karpathy, Fei-Fei Li, Justin Johnson cs231n
- <https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c>

Thank you