

Indian Institute of Information Technology, Allahabad



Introduction to Machine Learning Optimization

By

Dr. Shiv Ram Dubey

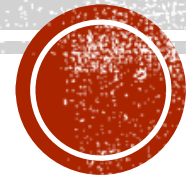
Associate Professor

Computer Vision and Biometrics Lab

Department of Information Technology

Indian Institute of Information Technology, Allahabad

Web: <https://profile.iitaa.ac.in/srdubey/>



DISCLAIMER

The content (text, image, and graphics) used in this slide are adopted from many sources for academic purposes. Broadly, the sources have been given due credit appropriately. However, there is a chance of missing out some original primary sources. The authors of this material do not claim any copyright of such material.

Optimization

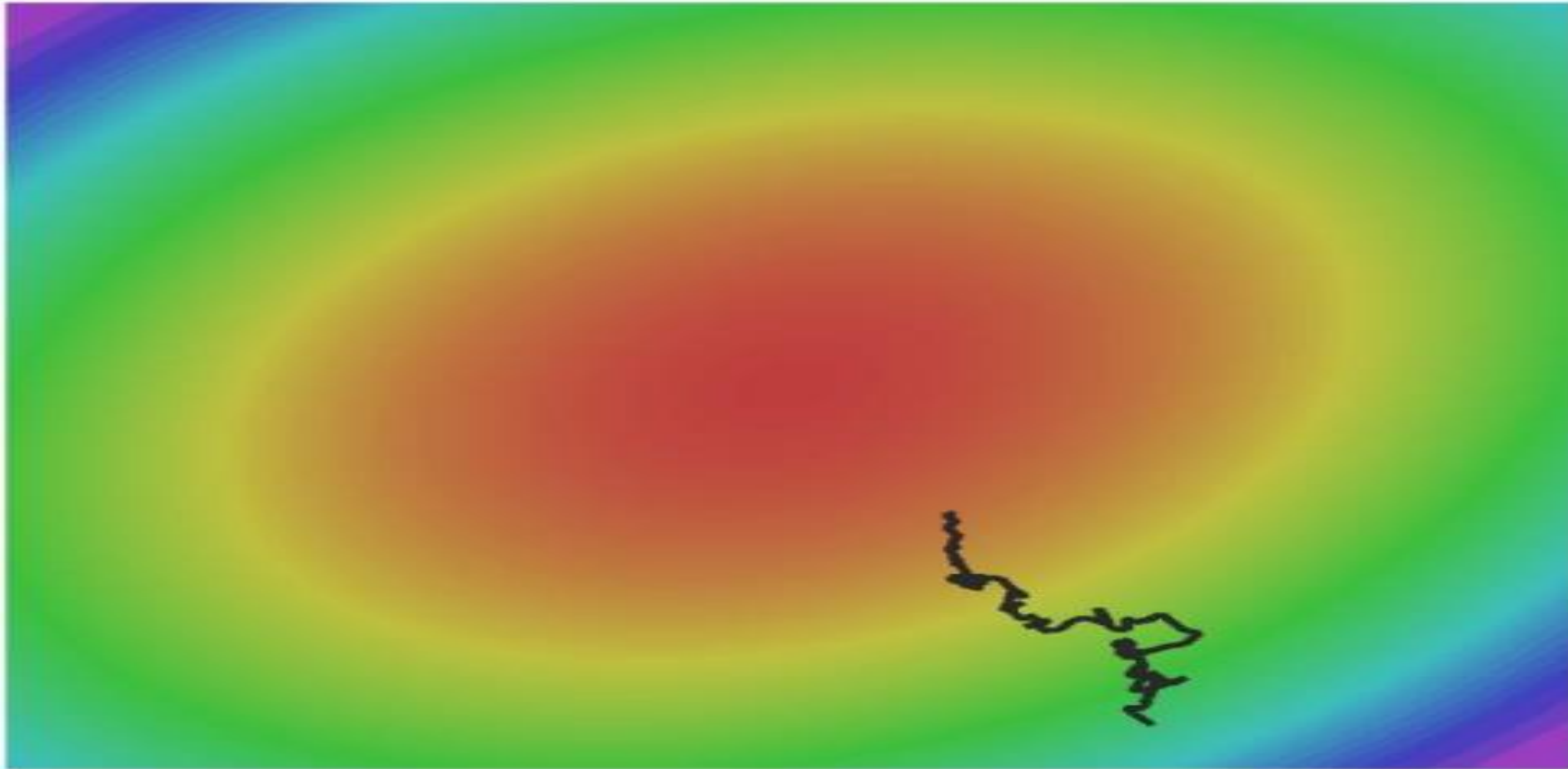


Image Source: cs231n, Stanford University

Optimization



Mini-batch SGD

Loop:

1. **Sample** a batch of data
2. **Forward** prop it through the graph (network), get loss
3. **Backprop** to calculate the gradients
4. **Update** the parameters using the gradient

Stochastic Gradient Descent (SGD)

The procedure of repeatedly evaluating the **gradient of loss function** and then performing a **parameter update**.

Vanilla (Original) Gradient Descent:

```
while True:
    dx = compute_gradient(x)
    x -= learning_rate * dx
```

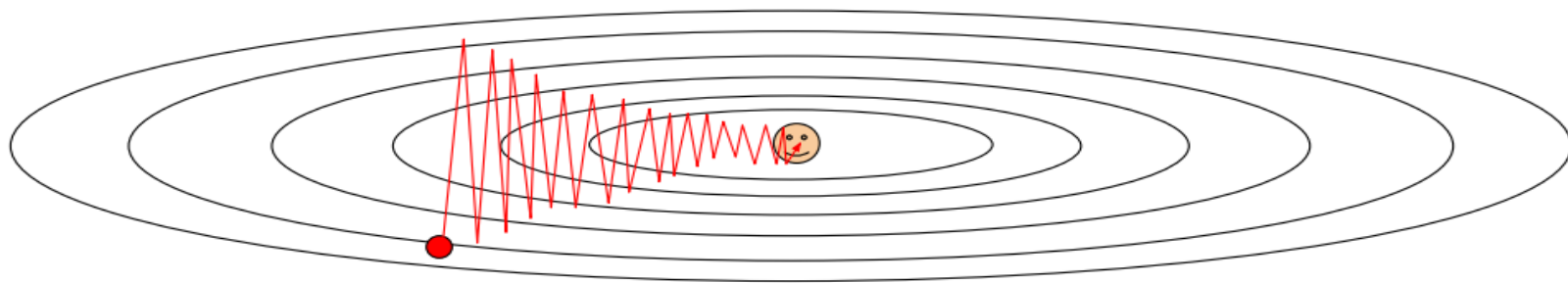
SGD: Problems

What if loss changes quickly in one direction and slowly in another?

SGD: Problems

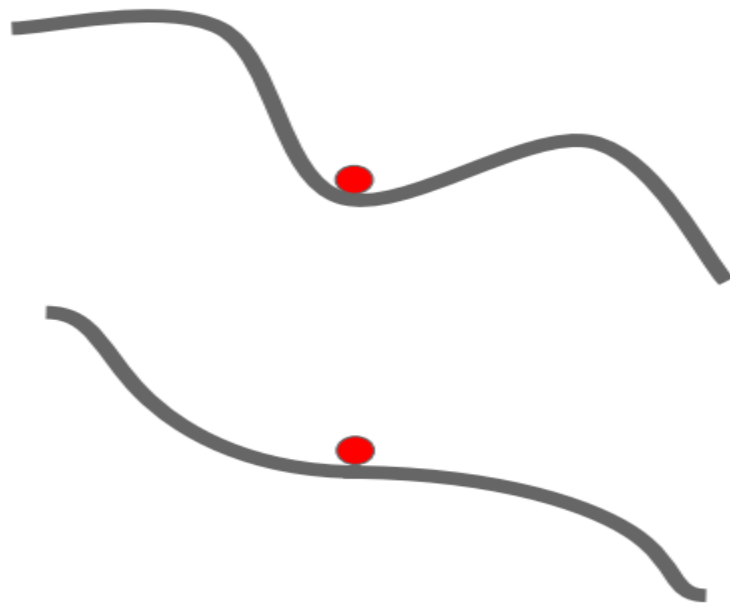
What if loss changes quickly in one direction and slowly in another?

Very slow progress along shallow dimension, jitter along steep direction



SGD: Problems

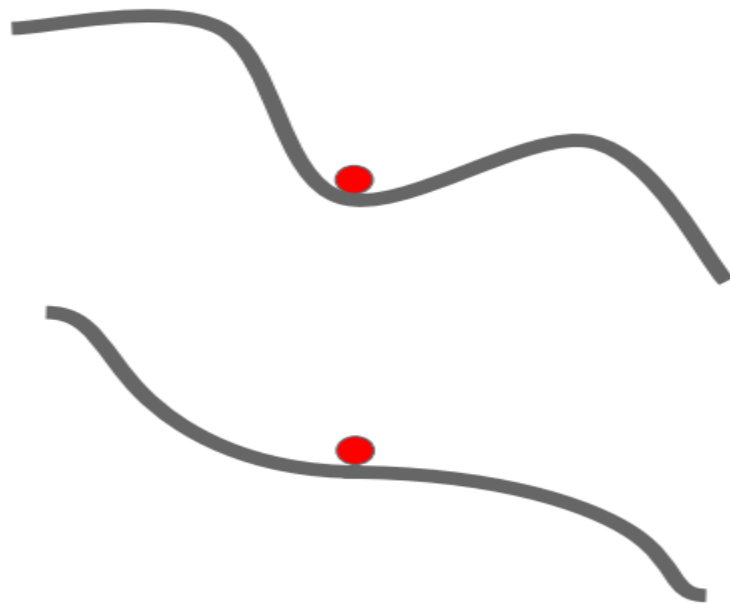
What if the loss function has a **local minima** or **saddle point**?



SGD: Problems

What if the loss function has a **local minima** or **saddle point**?

Zero gradient,
gradient descent
gets stuck

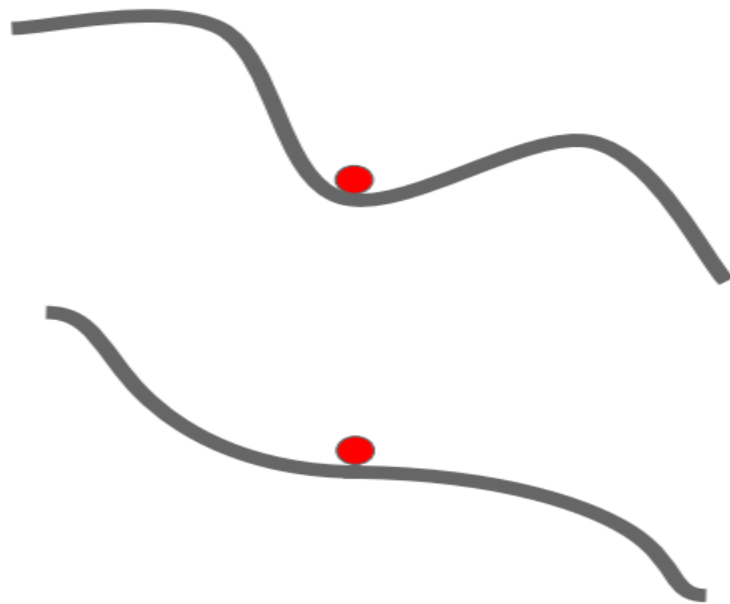


SGD: Problems

What if the loss function has a **local minima** or **saddle point**?

Zero gradient,
gradient descent
gets stuck

Saddle points much more
common in high dimension



Dauphin et al, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization", NIPS 2014

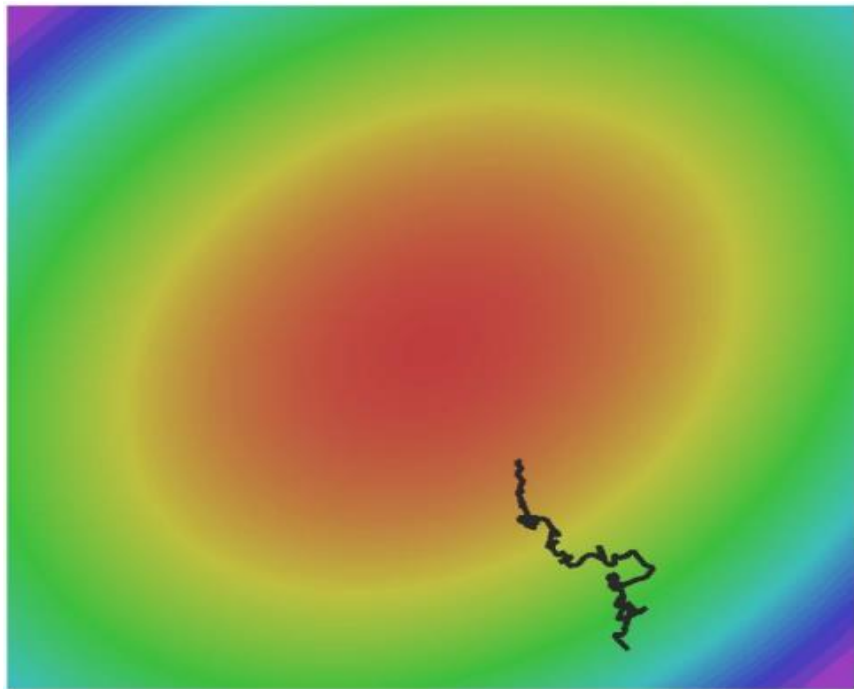
Source: cs231n

SGD: Problems

Our gradients come from **minibatches** so they can be **noisy!**

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W)$$



SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:  
    dx = compute_gradient(x)  
    x -= learning_rate * dx
```

SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
    dx = compute_gradient(x)
    x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
    dx = compute_gradient(x)
    x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
    dx = compute_gradient(x)
    vx = rho * vx + dx
    x -= learning_rate * vx
```

SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
    dx = compute_gradient(x)
    x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
    dx = compute_gradient(x)
    vx = rho * vx + dx
    x -= learning_rate * vx
```

- Build up “velocity” in any direction that has consistent gradient
- Rho gives “friction”; typically rho=0.9 or 0.99

SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

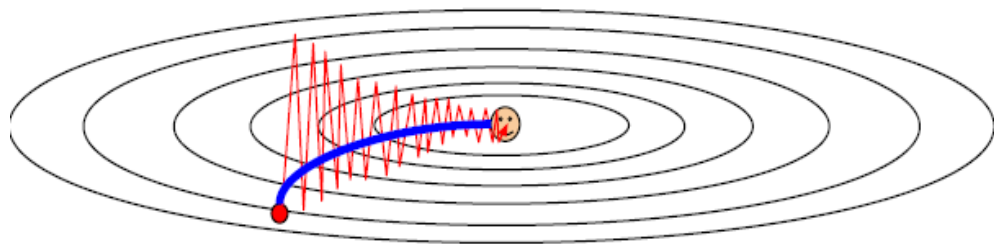
```
while True:
    dx = compute_gradient(x)
    x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
    dx = compute_gradient(x)
    vx = rho * vx + dx
    x -= learning_rate * vx
```



AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

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

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

What happens to the step size over long time?

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

What happens to the step size over long time?

Effective learning rate diminishing problem

RMSProp

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

AdaGrad



```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

Adam

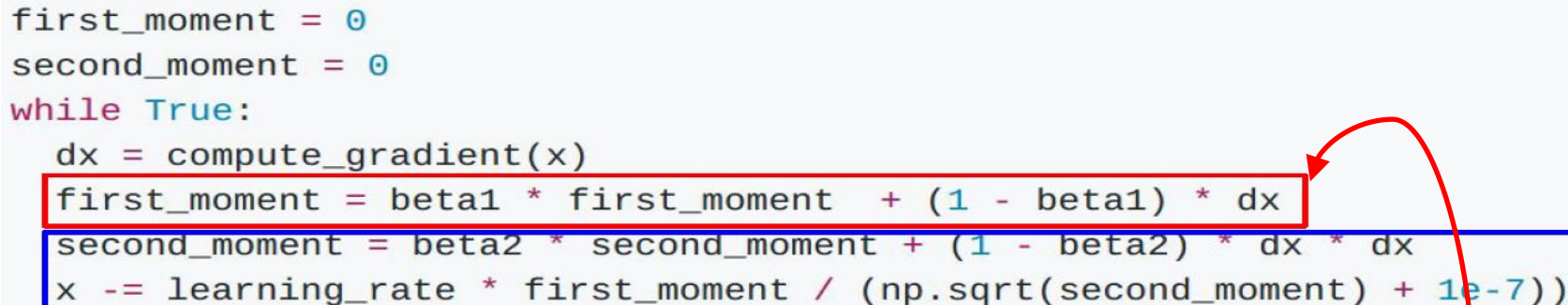
Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

```
first_moment = 0
second_moment = 0
while True:
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Adam

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

```
first_moment = 0
second_moment = 0
while True:
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

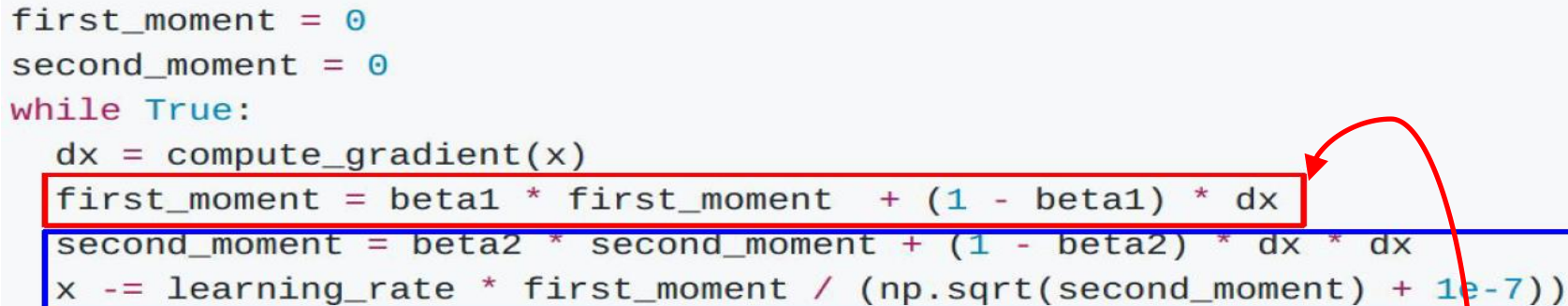


Sort of like RMSProp with Momentum

Adam

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

```
first_moment = 0
second_moment = 0
while True:
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```



Sort of like **RMSProp** with **Momentum**

Problem:

Initially, second_moment=0 and beta2=0.999

After 1st iteration, second_moment -> close to zero

So, very large step for update of x

Adam (with Bias correction)

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

```
first_moment = 0
second_moment = 0
for t in range(num_iterations):
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

AdaGrad/
RMSProp

Bias Correction

Bias correction for the fact that first and second moment estimates start at zero

Momentum

Adam (with Bias correction)

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

```
first_moment = 0
second_moment = 0
for t in range(num_iterations):
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

AdaGrad/
RMSProp

Bias Correction

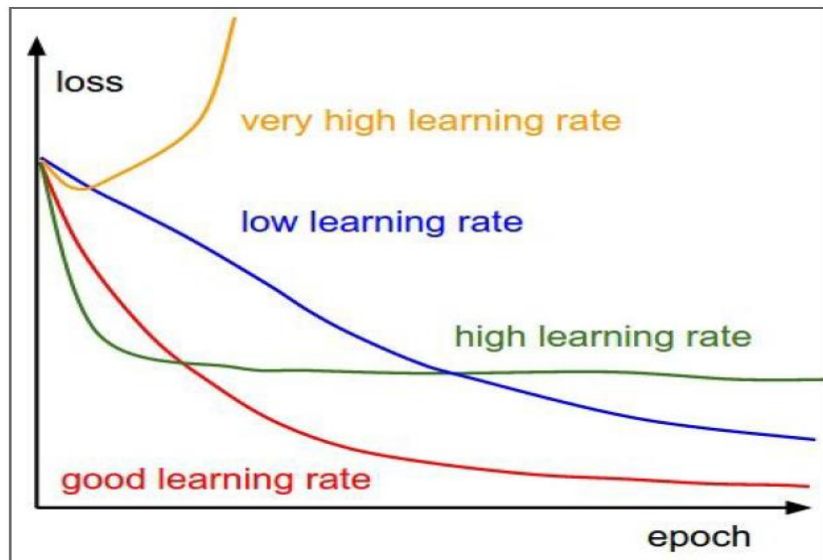
Momentum

Bias correction for the fact that first and second moment estimates start at zero

Adam with **beta1 = 0.9**,
beta2 = 0.999, and **learning_rate = 1e-3 or 5e-4**
is a **great starting point** for many models!

Learning Rate

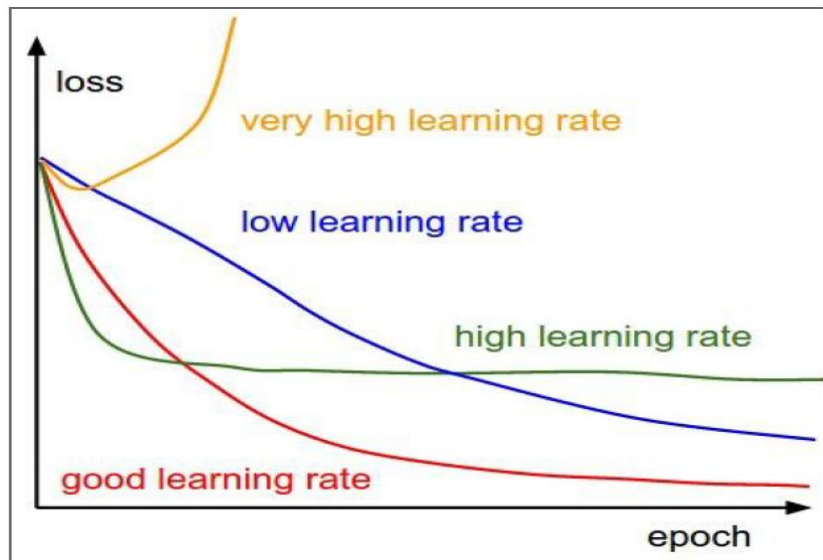
SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



Q: Which one of these learning rates is best to use?

Learning Rate

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



=> Learning rate decay over time!

step decay:

e.g. decay learning rate by half every few epochs.

exponential decay:

$$\alpha = \alpha_0 e^{-kt}$$

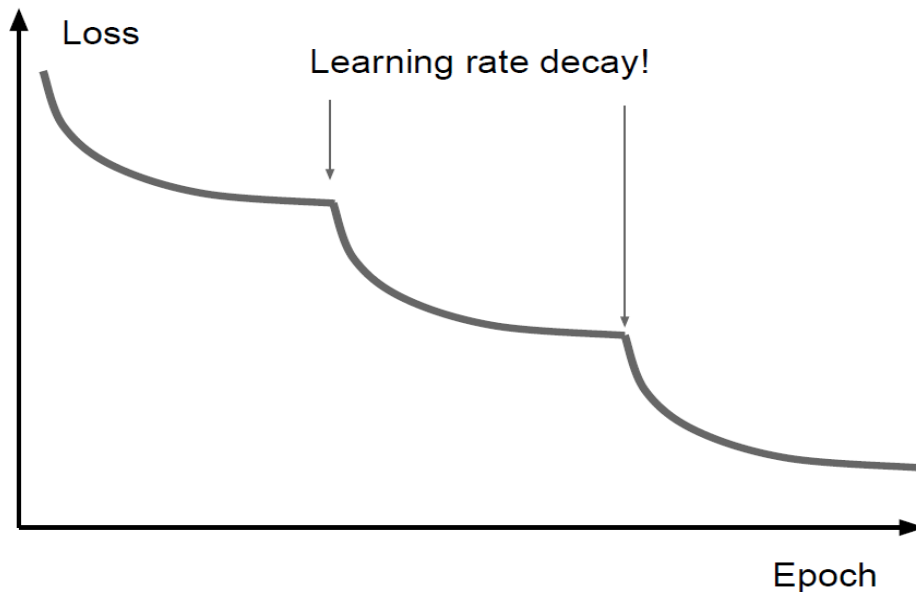
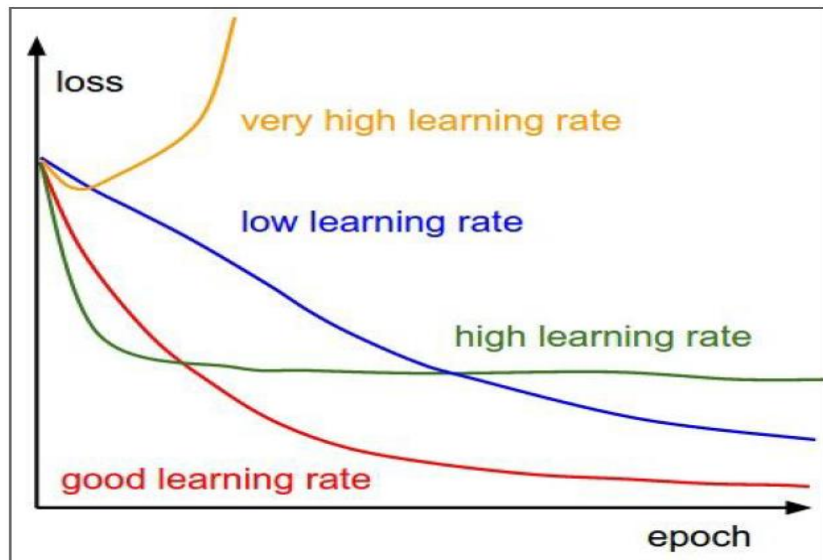
?

1/t decay:

$$\alpha = \alpha_0 / (1 + kt)$$

Learning Rate

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



Learning Rate

In Practice:

- **Learning rate** with step decay is commonly used
 - **Step decay:** reduce rate by a constant factor every few epochs, e.g., by 0.5 every 5 epochs, 0.1 every 20 epochs
 - **Manual:** watch validation error and reduce learning rate whenever it stops improving
 - “Patience” hyperparameter: number of epochs without improvement before reducing learning rate
- **Warmup:** train with a low learning rate for a first few epochs, or linearly increase learning rate before transitioning to normal decay schedule ([Goyal et al.](#), 2018)

Acknowledgement

Thanks to the following courses and corresponding researchers for making their teaching/research material online

- Deep Learning, Stanford University
- Introduction to Deep Learning, University of Illinois at Urbana-Champaign
- Introduction to Deep Learning, Carnegie Mellon University
- Convolutional Neural Networks for Visual Recognition, Stanford University
- Natural Language Processing with Deep Learning, Stanford University
- And Many More

THANK YOU