Optimizers

CS109A Introduction to Data Science

Pavlos Protopapas, Kevin Rader and Chris Tanner


Brute Force
Greedy Search
Non-Convex optimization using Gradient Descent

Outline

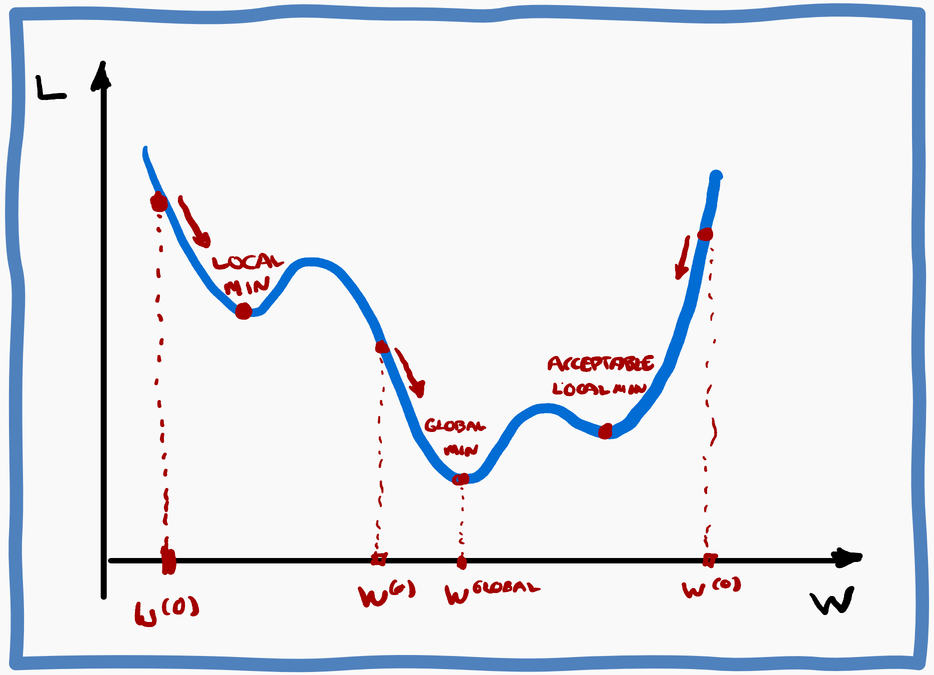
- Challenges in Optimization
- Momentum
- Adaptive Learning Rate

Outline

- **Challenges in Optimization**
- Momentum
- Adaptive Learning Rate

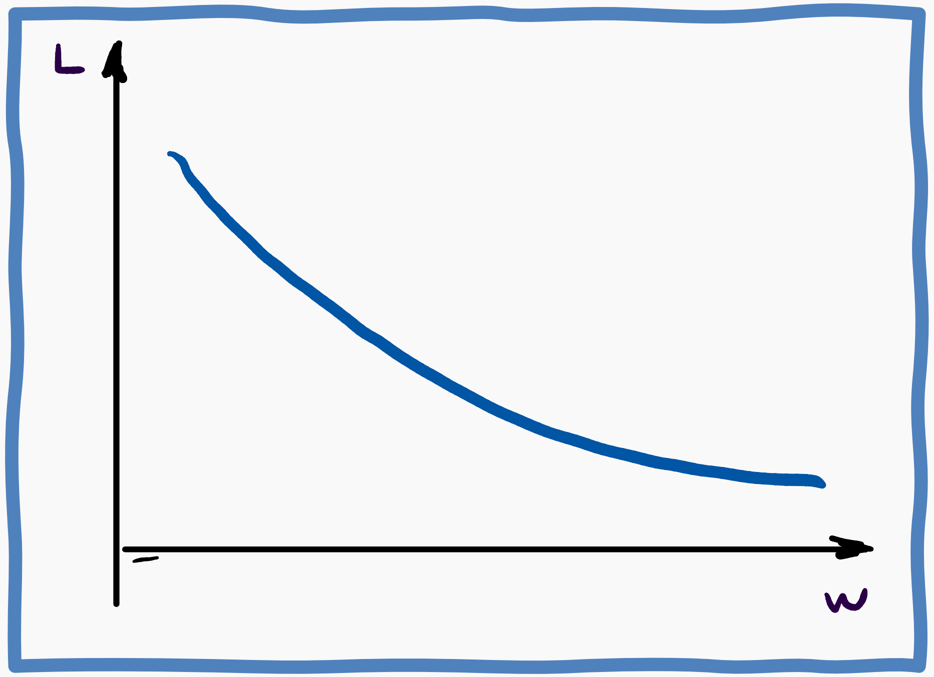
Challenges in Optimization

Local Minima



Ideally, we would like to arrive at the global minimum, but this might not be possible. Some local minima performs as well as the global one, so it is an acceptable stopping point.

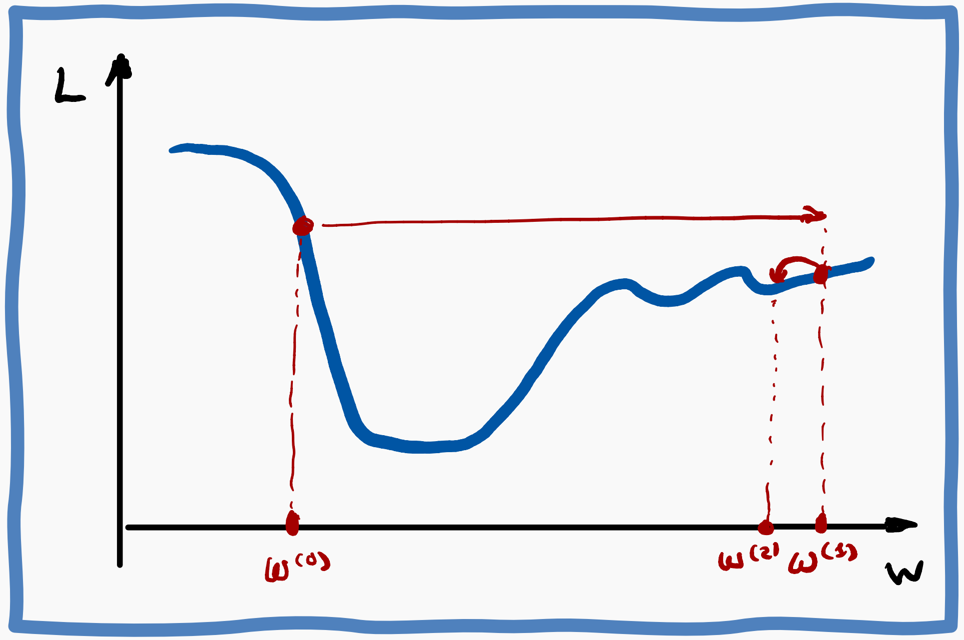
No critical points



Some cost functions do not have critical points. In particular for classification when $p(y = 1)$ is never zero or one.

Challenges in Optimization

Exploding and vanishing Gradients

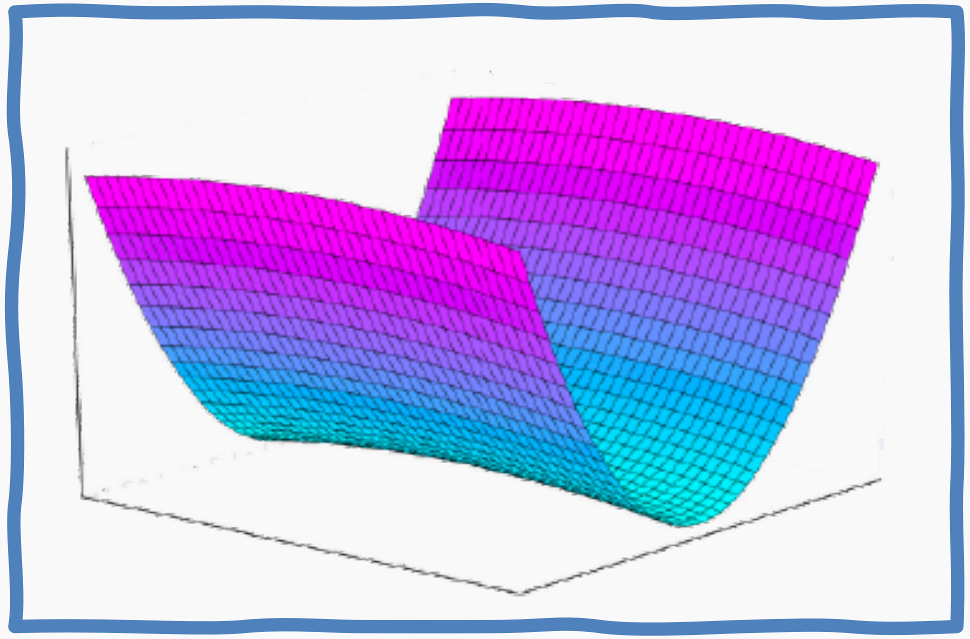


Exploding gradients due to cliffs. Can be mitigated using gradient clipping:

$$\text{if } \left\|\frac{\partial L}{\partial W}\right\| > u: \quad \frac{\partial L}{\partial W} = u$$

where u is user defined threshold.

Poor Conditioning



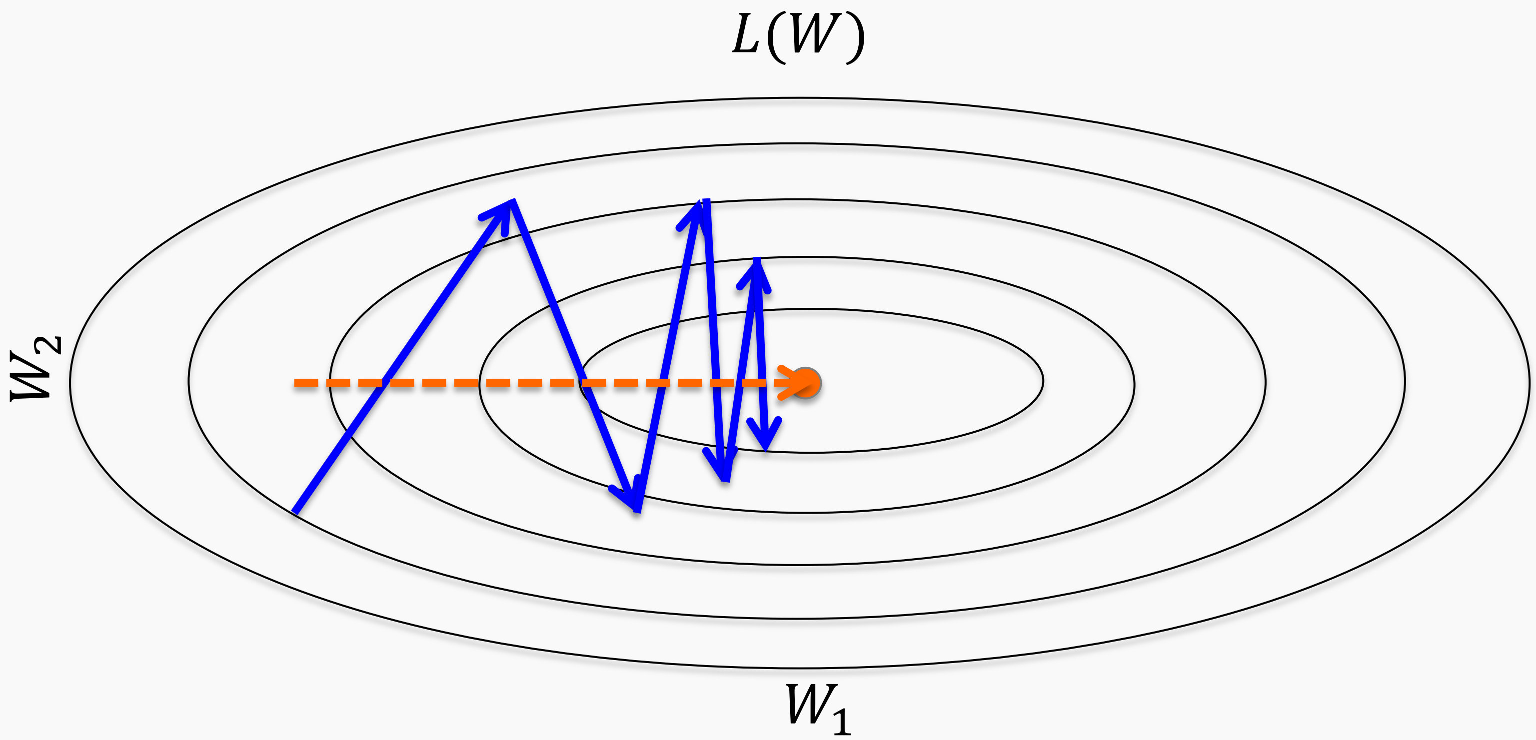
Poorly conditioned Hessian matrix. High curvature: small steps leads to huge increase. Learning is slow despite strong gradients. Oscillations slow down progress.

Outline

- Challenges in Optimization
- **Momentum**
- Adaptive Learning Rate

Momentum

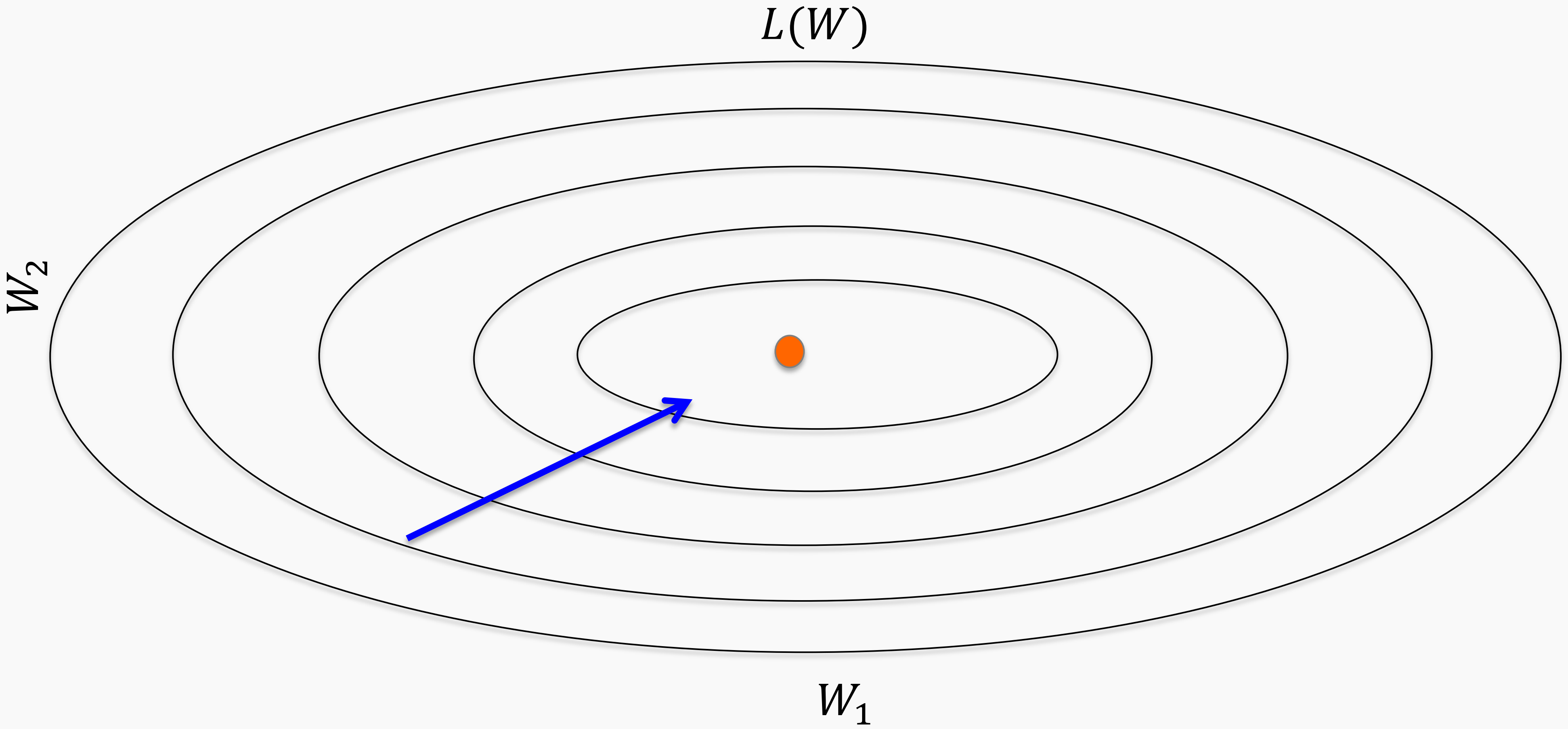
Oscillations because updates do not exploit curvature information



Average gradient presents faster path to optimal: vertical components cancel out

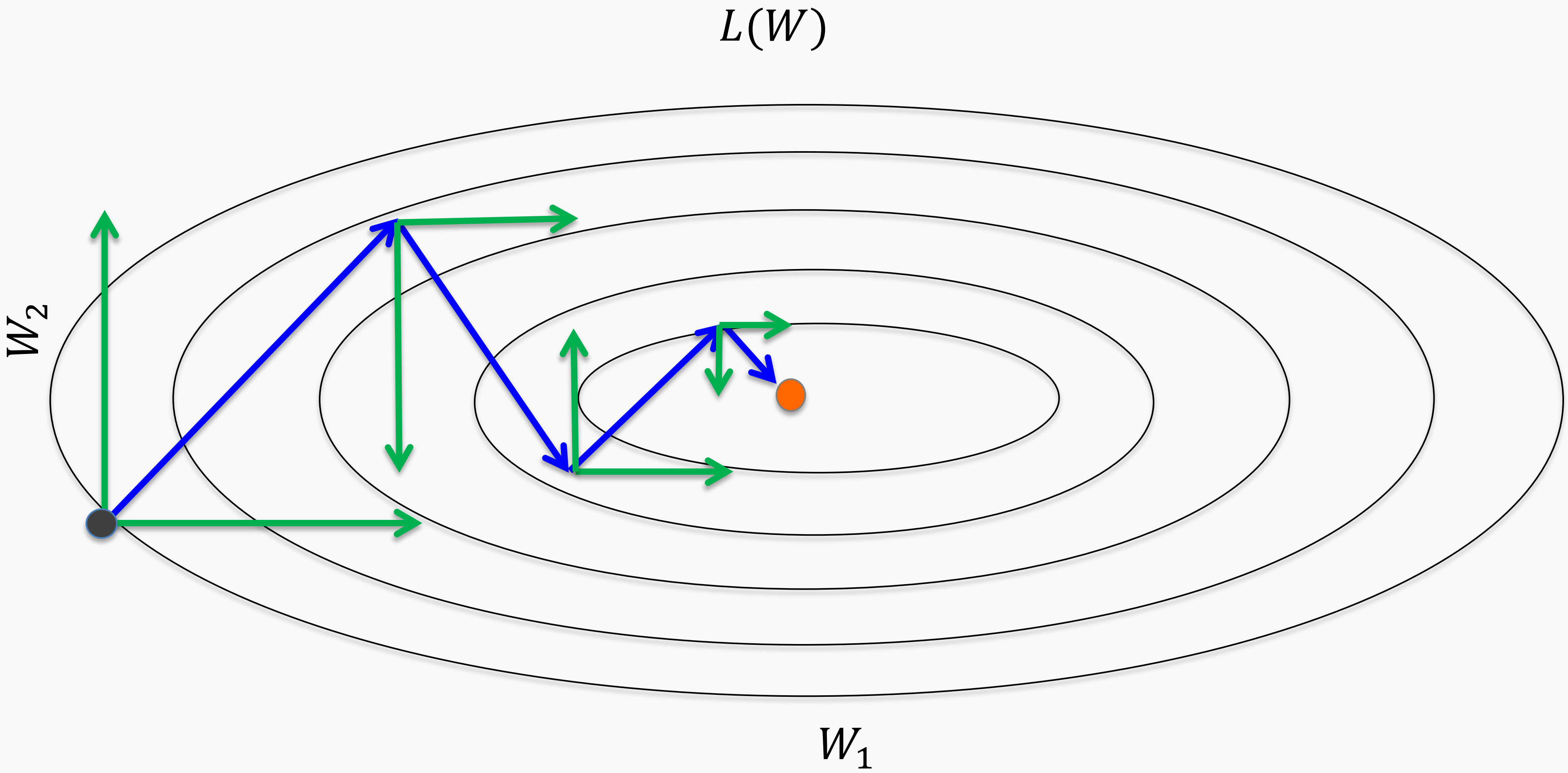
Momentum

Question: Why not this?



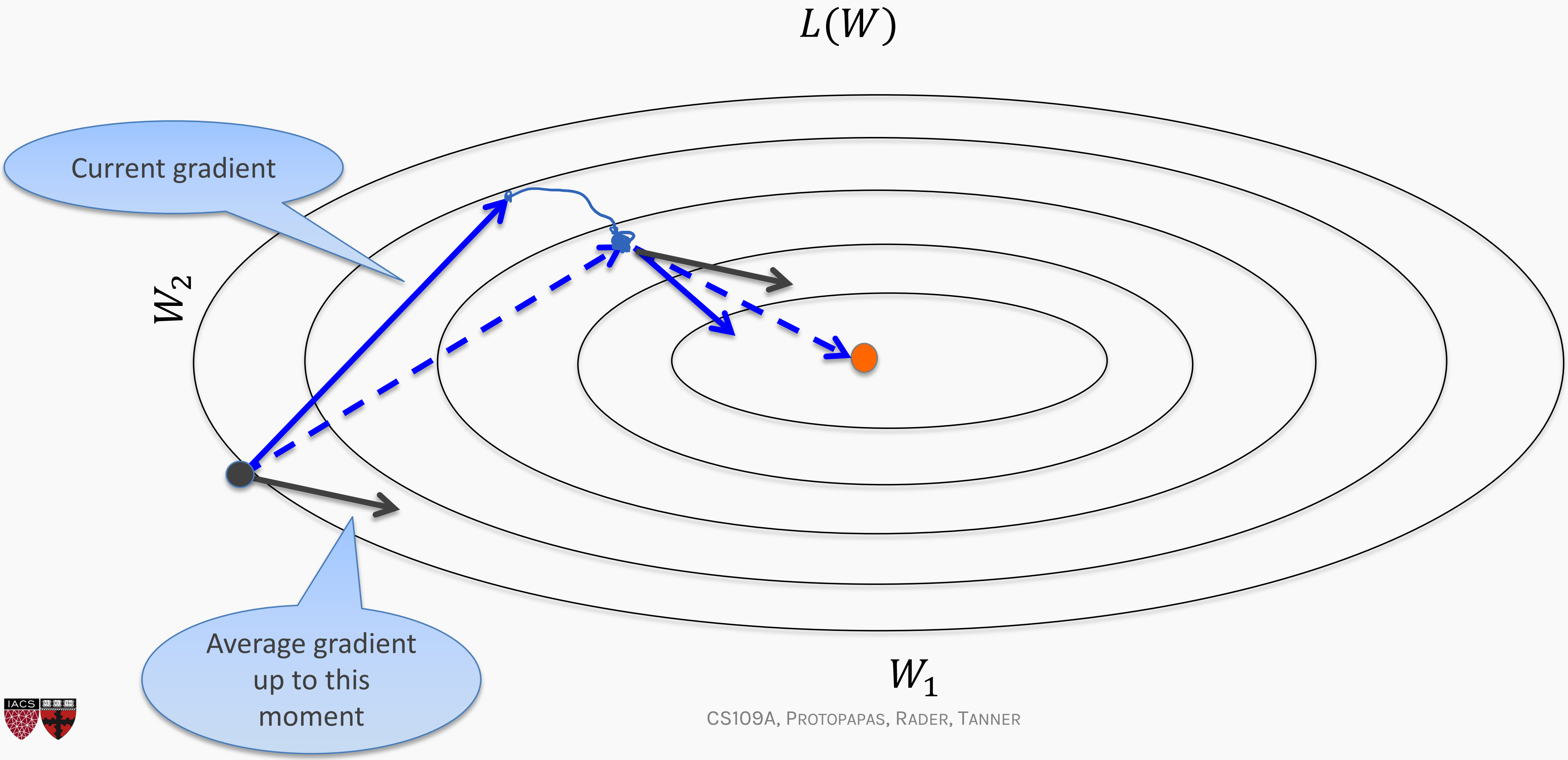
Momentum

Let us figure out an algorithm which will lead us to the minimum faster.



Momentum

Add the average of the gradient from before



Momentum

f is the Neural Network

Old gradient descent:

$$g = \frac{1}{m}\sum_i \nabla_W L(f(x_i; W), y_i) \qquad W^* = W - \eta g$$

New gradient descent with momentum:

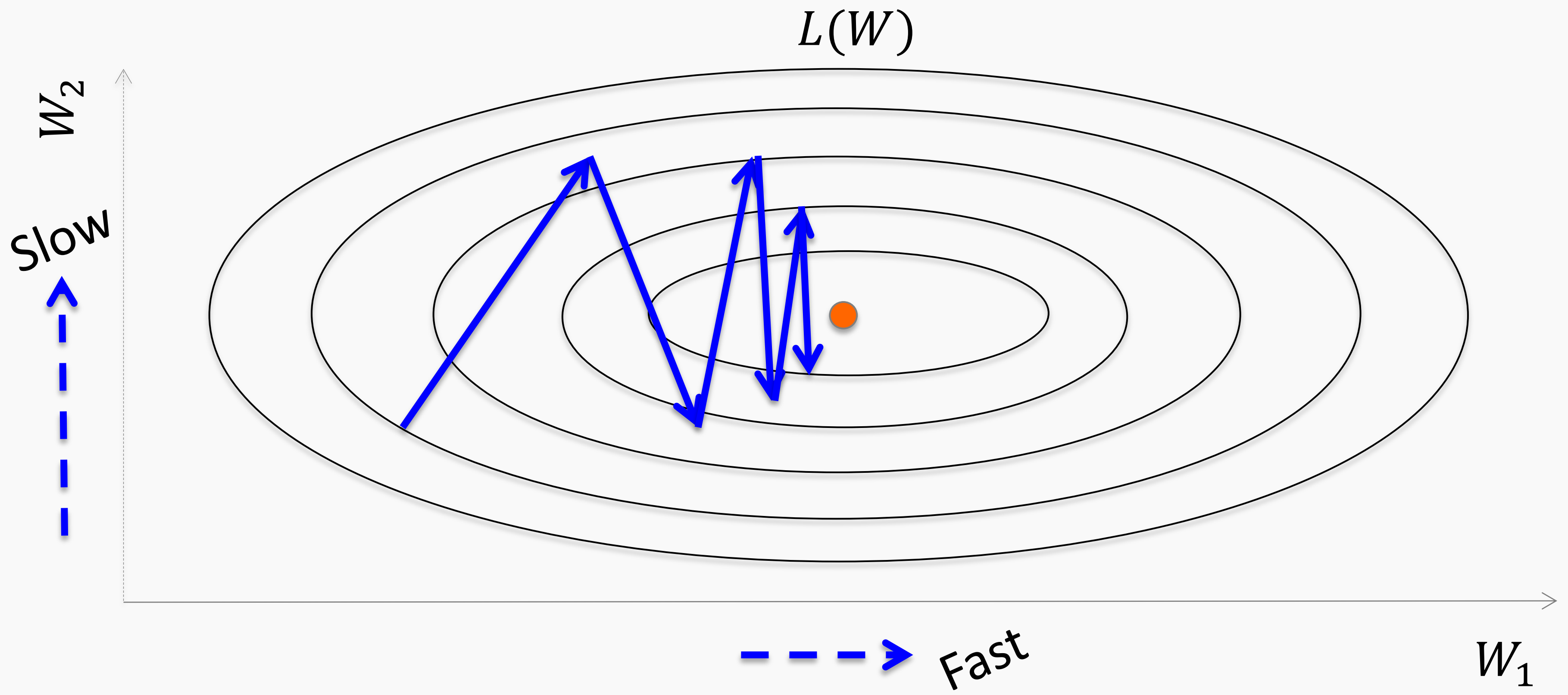
$$\nu = \alpha\nu + (1-\alpha)\, g \qquad W^* = W - \eta\nu$$

$\alpha \in [0,1)$ controls how quickly effect of past gradients decay

Outline

- Challenges in Optimization
- Momentum
- **Adaptive Learning Rate**

Adaptive Learning Rates

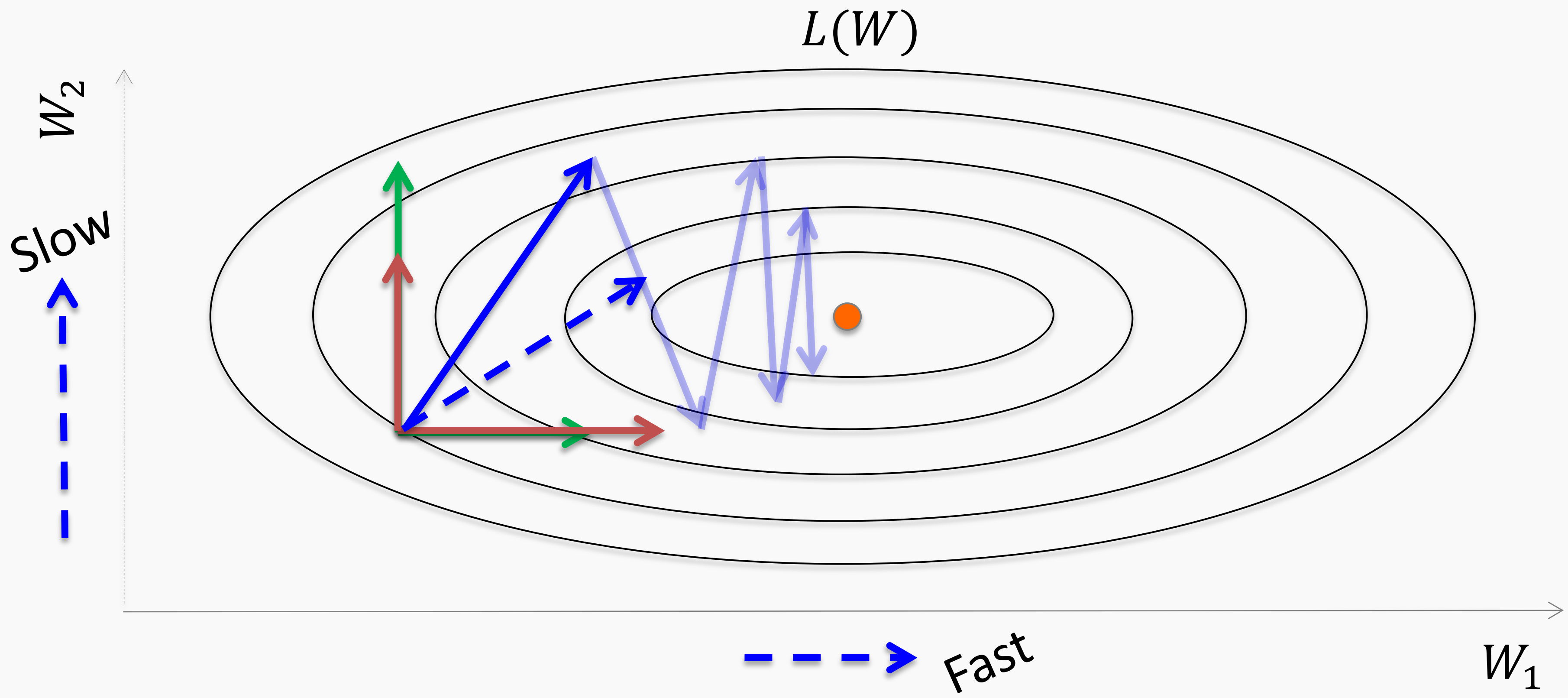


Oscillations along vertical direction

– Learning must be slower along parameter 2

Use a different learning rate for each parameter?

Adaptive Learning Rates

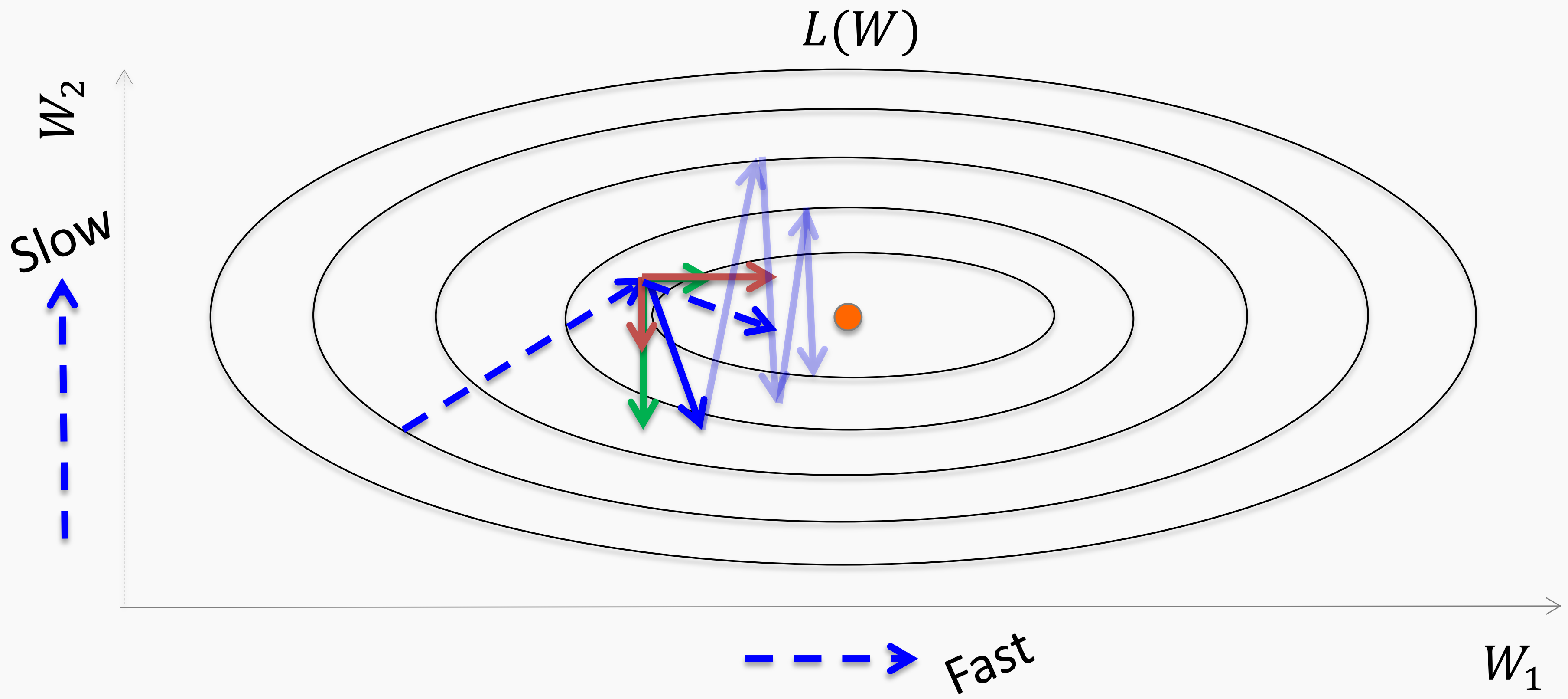


Oscillations along vertical direction

- Learning must be slower along parameter 2

Use a different learning rate for each parameter?

Adaptive Learning Rates

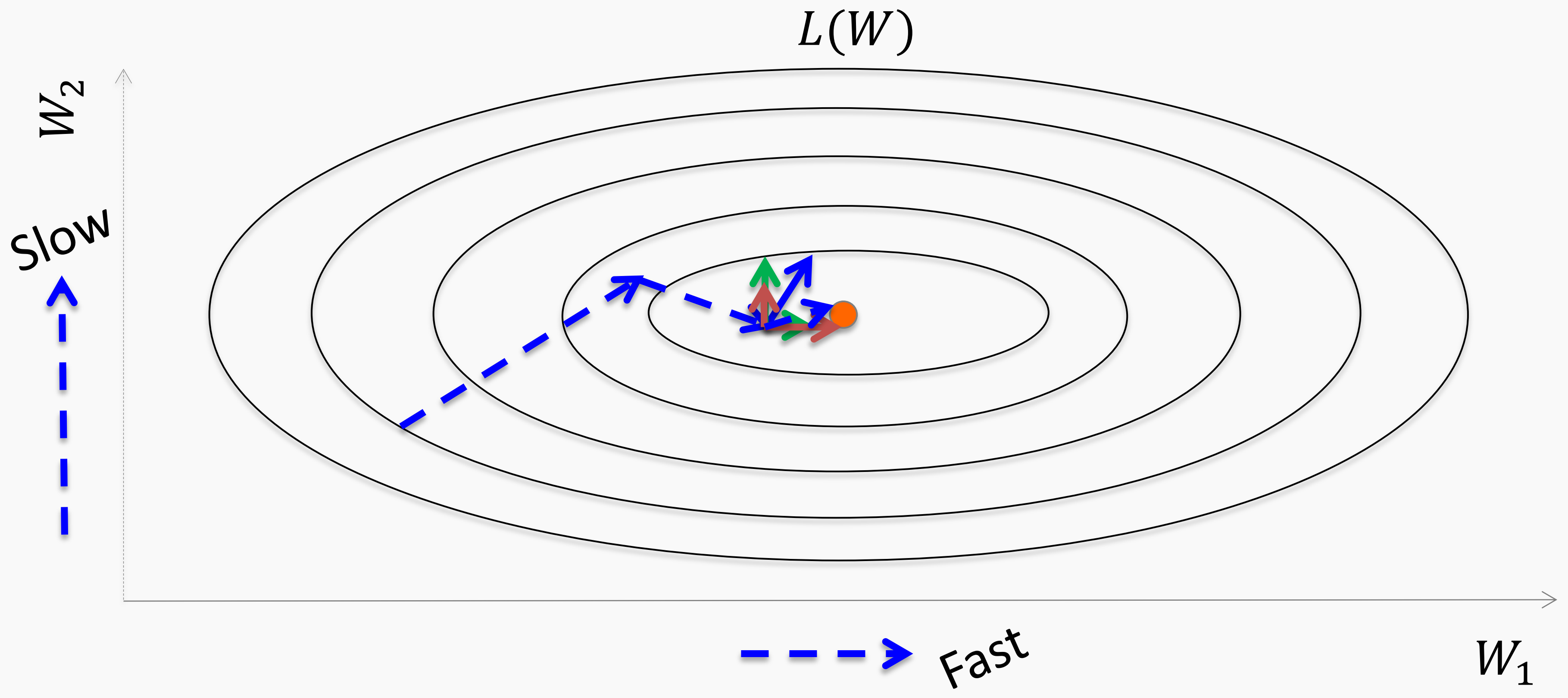


Oscillations along vertical direction

– Learning must be slower along parameter 2

Use a different learning rate for each parameter?

Adaptive Learning Rates

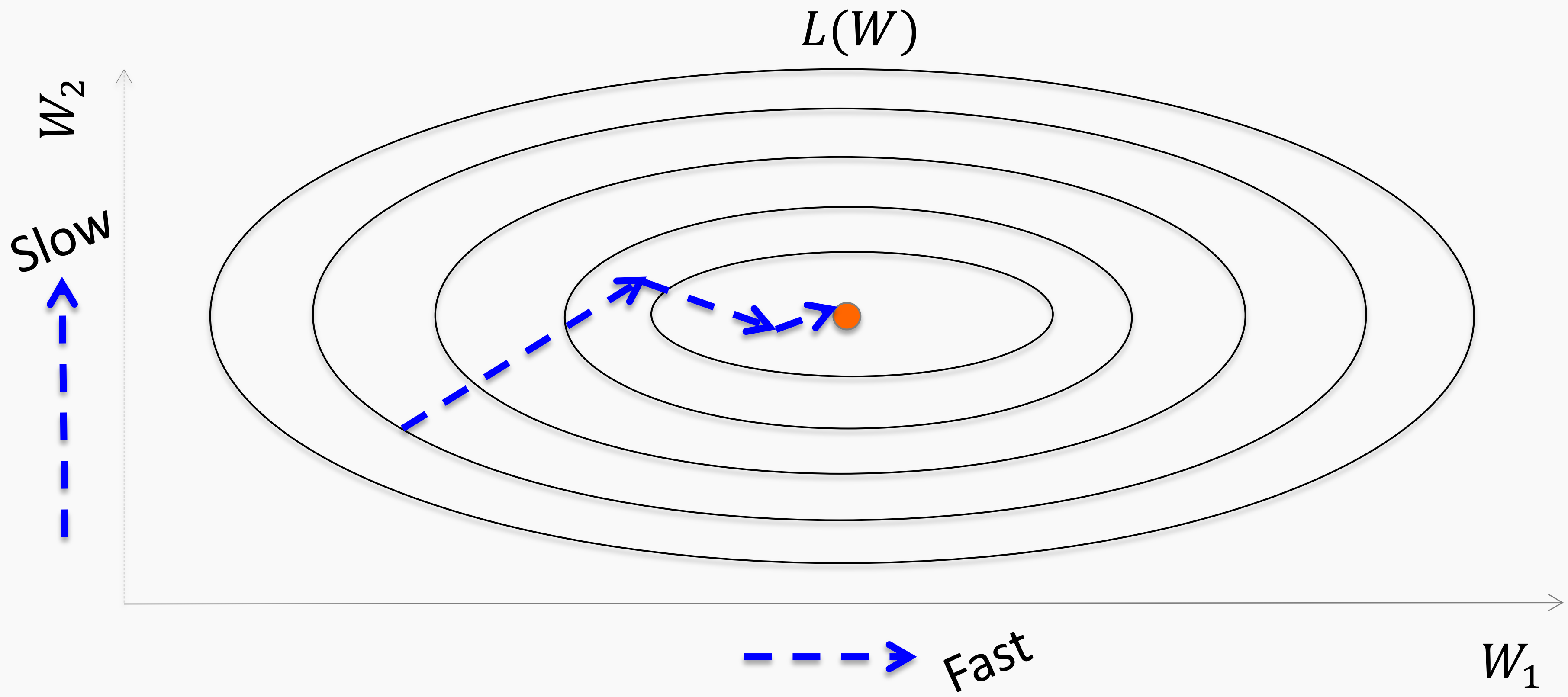


Oscillations along vertical direction

– Learning must be slower along parameter 2

Use a different learning rate for each parameter?

Adaptive Learning Rates



Oscillations along vertical direction

– Learning must be slower along parameter 2

Use a different learning rate for each parameter?

AdaGrad

- Accumulate squared gradients:

$$r_i = r_i + g_i^2$$

g is the gradient

- Update each parameter:

$$W_i = W_i - \frac{\epsilon}{\delta + \sqrt{r_i}} g_i$$

Inversely proportional to cumulative gradient

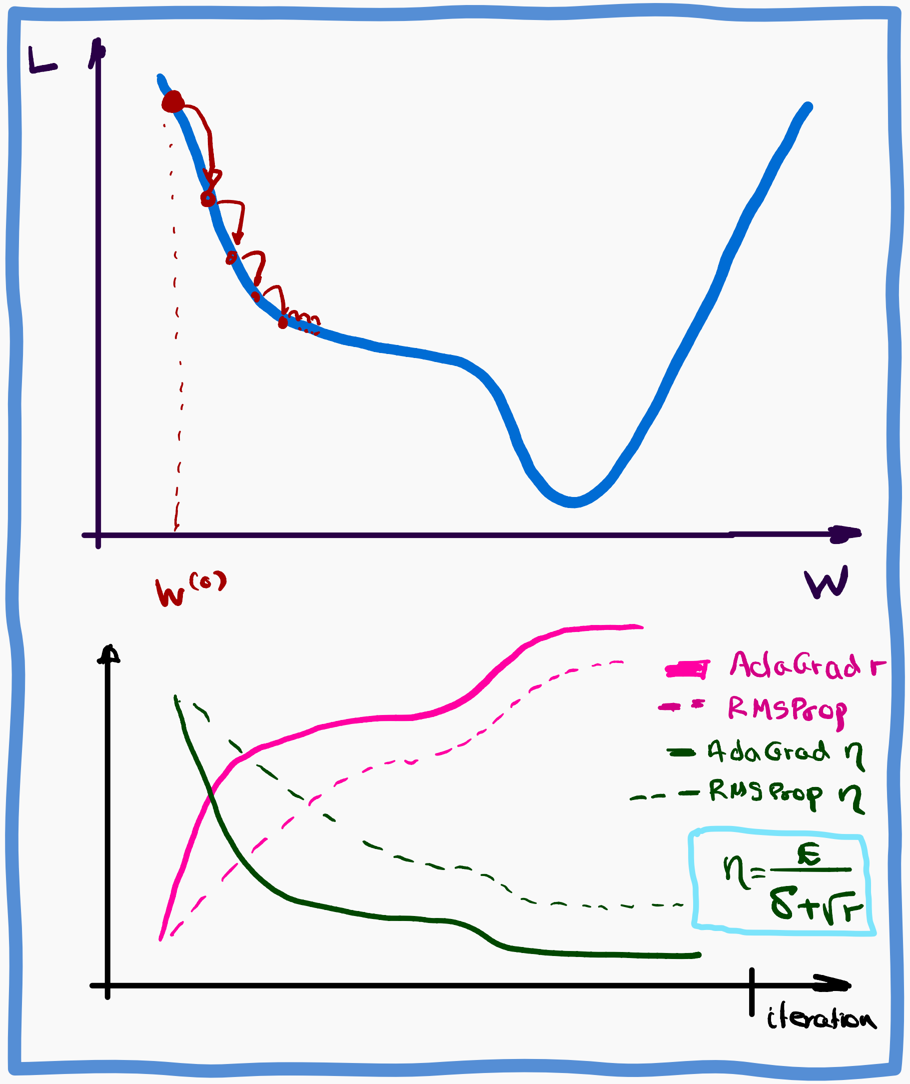
- Greater progress along gently sloped directions

RMSProp

- For non-convex problems, AdaGrad can prematurely decrease learning rate
- Use exponentially weighted average for gradient accumulation

$$r_i = \rho r_i + (1-\rho) g_i^2$$

$$W_i = W_i - \frac{\epsilon}{\delta + \sqrt{r_i}} g_i$$



Adam

- RMSProp + Momentum
- Estimate first moment:

$$v_i = \rho_1 v_i + (1-\rho_1) g_i$$

- Estimate second moment:

$$r_i = \rho_2 r_i + (1-\rho_2) g_i^2$$

- Update parameters:

$$W_i = W_i - \frac{\epsilon}{\delta + \sqrt{r_i}} v_i$$

Also applies bias correction to v and r

Works well in practice, it is fairly robust to hyper-parameters

Bias Correction

To perform bias correction on the two running average variables, we use the following equations. We do this before we update weights.

$$v_{corr} = \frac{v}{1 - \rho_1^t}$$

$$r_{corr} = \frac{r}{1 - \rho_2^t}$$

Where t is the number of the current iteration.