Optimizers

CS109A Introduction to Data Science

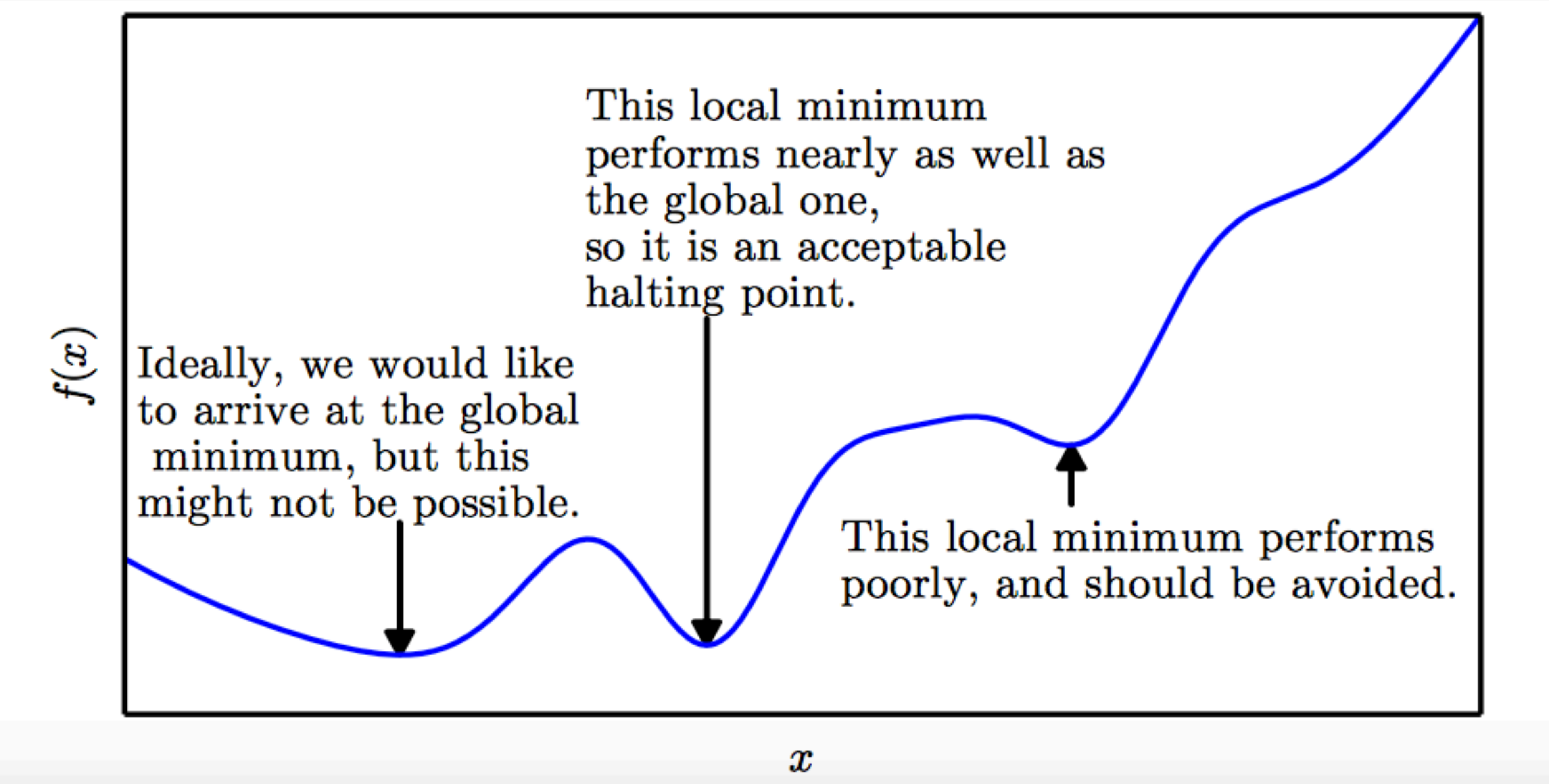
Pavlos Protopapas, Kevin Rader and Chris Tanner

Outline

Optimization

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

Local Minima



Goodfellow et al. (2016)

Local Minima

Old view: local minima is major problem in neural network training

Recent view:

- For sufficiently large neural networks, most local minima incur low cost
- Not important to find true global minimum

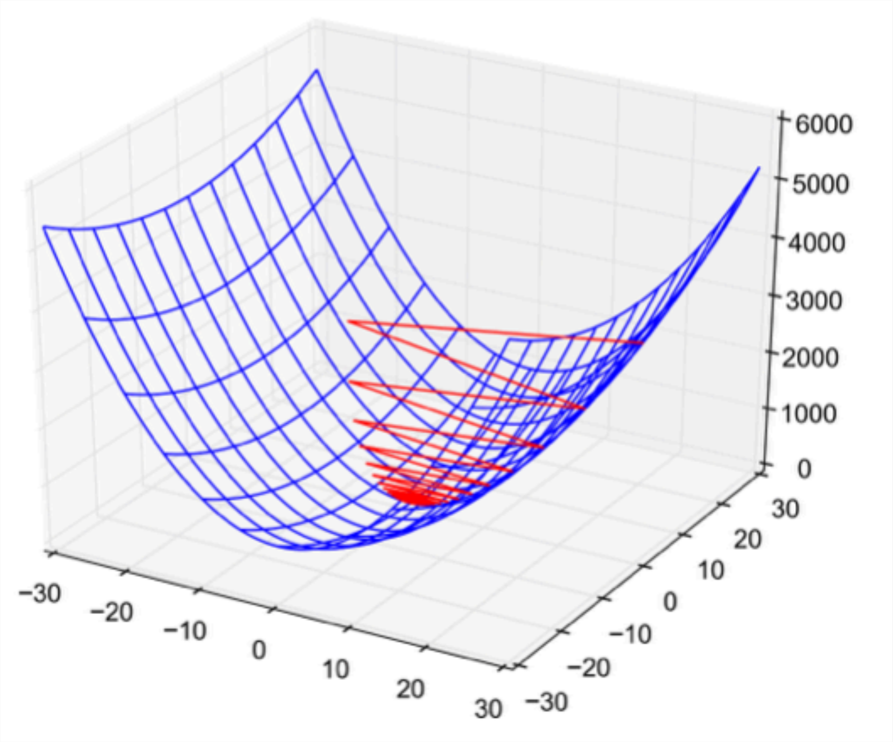
Poor Conditioning

Poorly conditioned Hessian matrix

- High curvature: small steps leads to huge increase

Learning is slow despite strong gradients

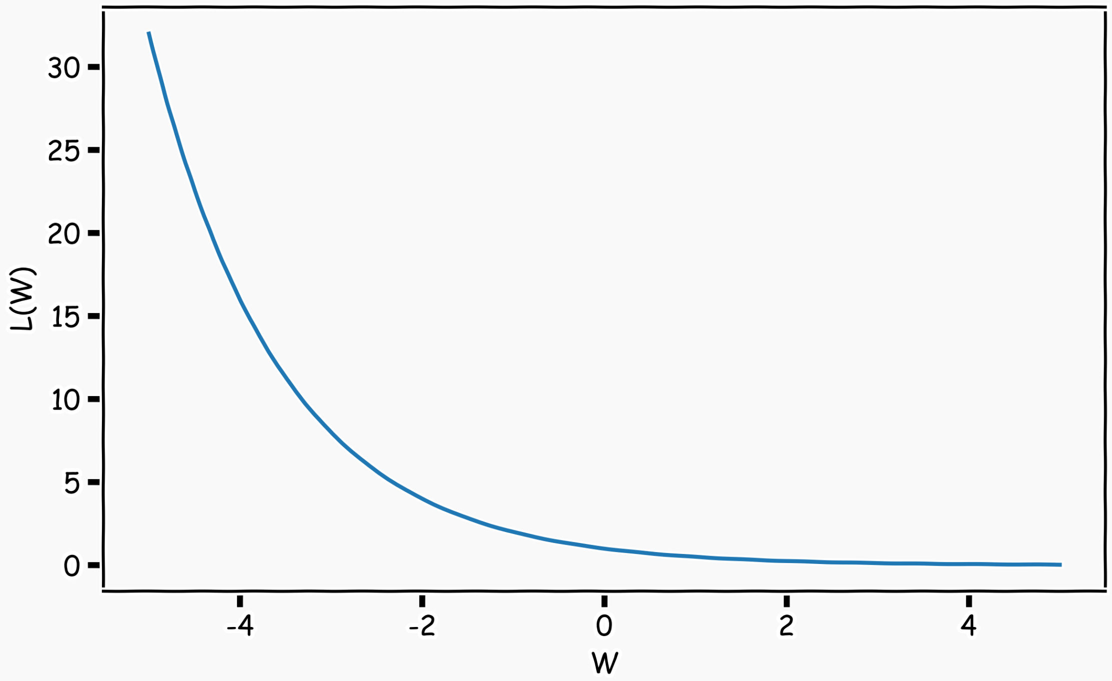
Oscillations slow down progress



No Critical Points

Some cost functions do not have critical points. In particular classification.

WHY?



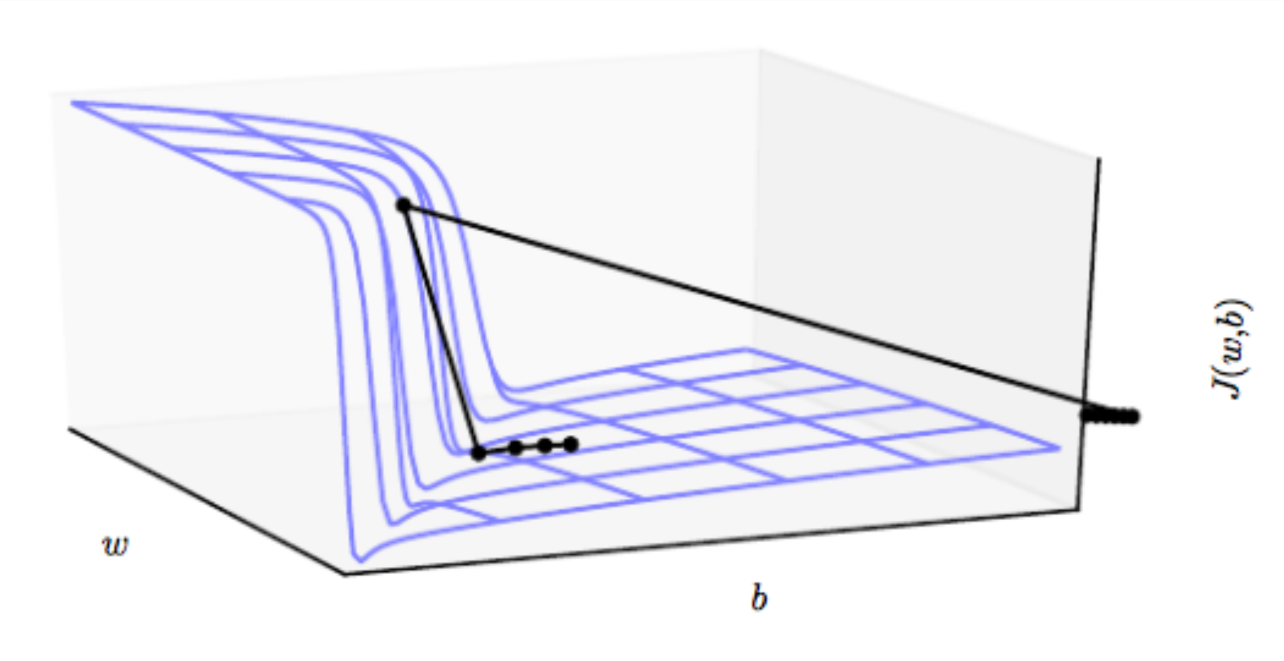
Exploding and Vanishing Gradients

Exploding gradients lead to cliffs

Can be mitigated using gradient clipping

$$\text{if } ||g|| > u$$

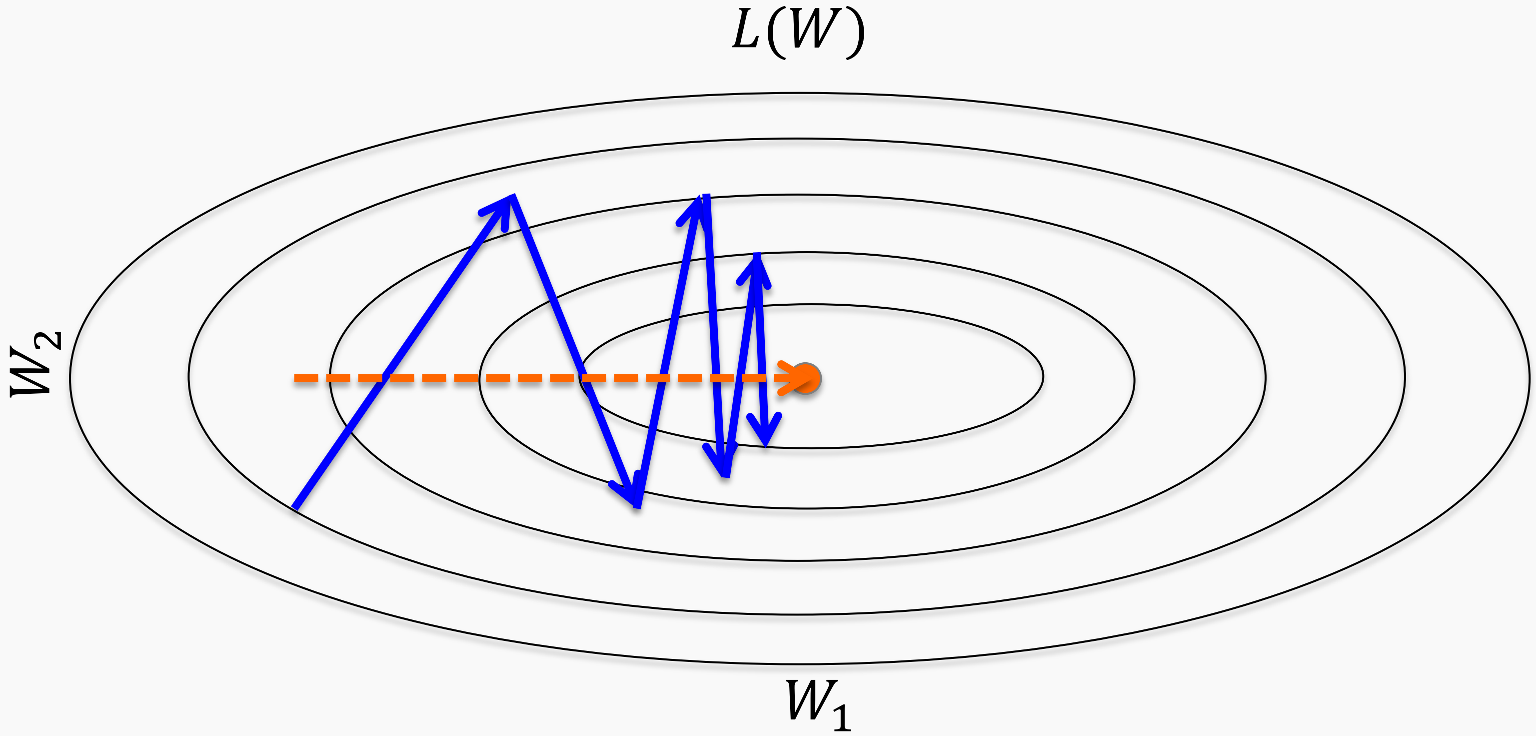
$$g \leftarrow \frac{gu}{||g||}$$



Goodfellow et al. (2016)

Momentum

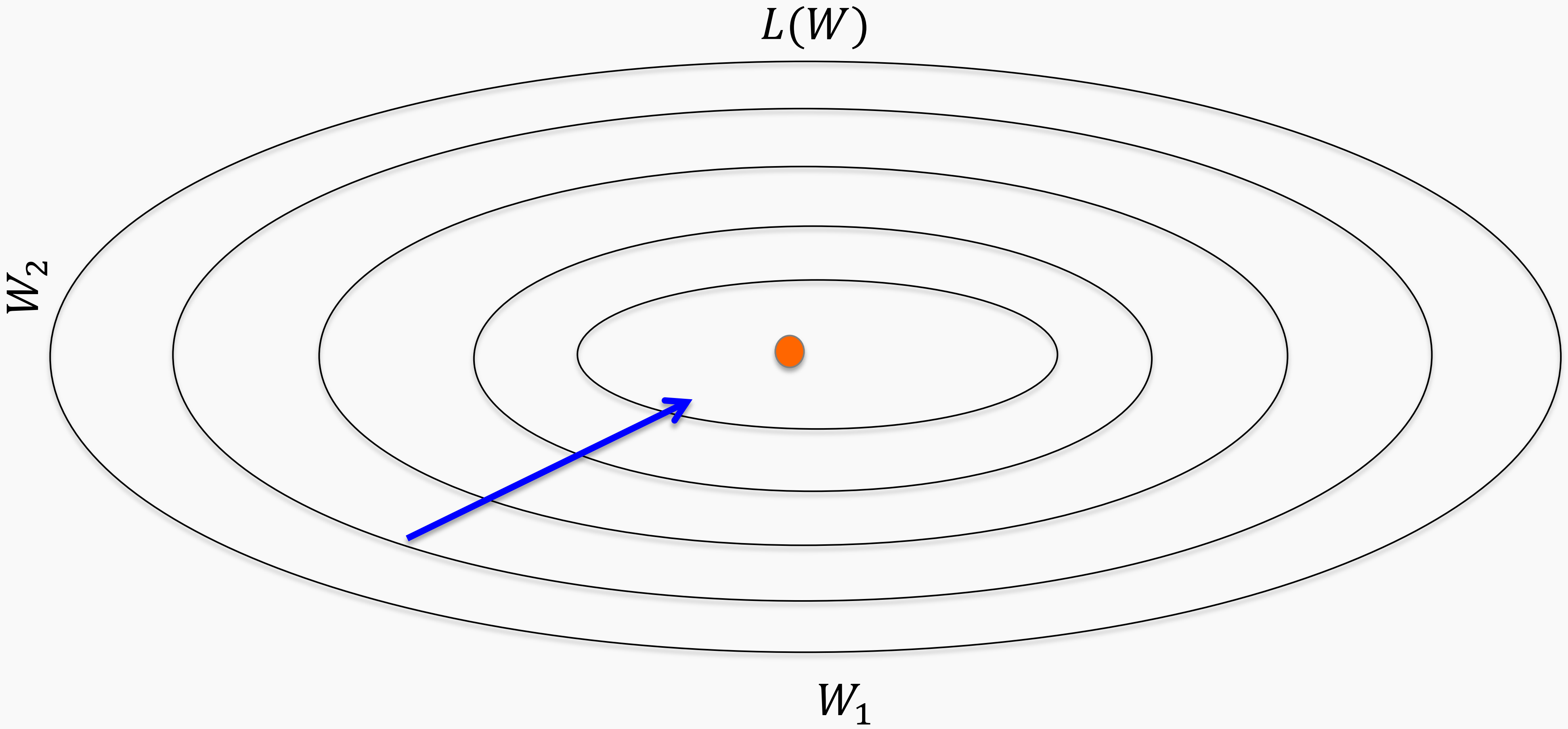
Oscillations because updates do not exploit curvature information



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

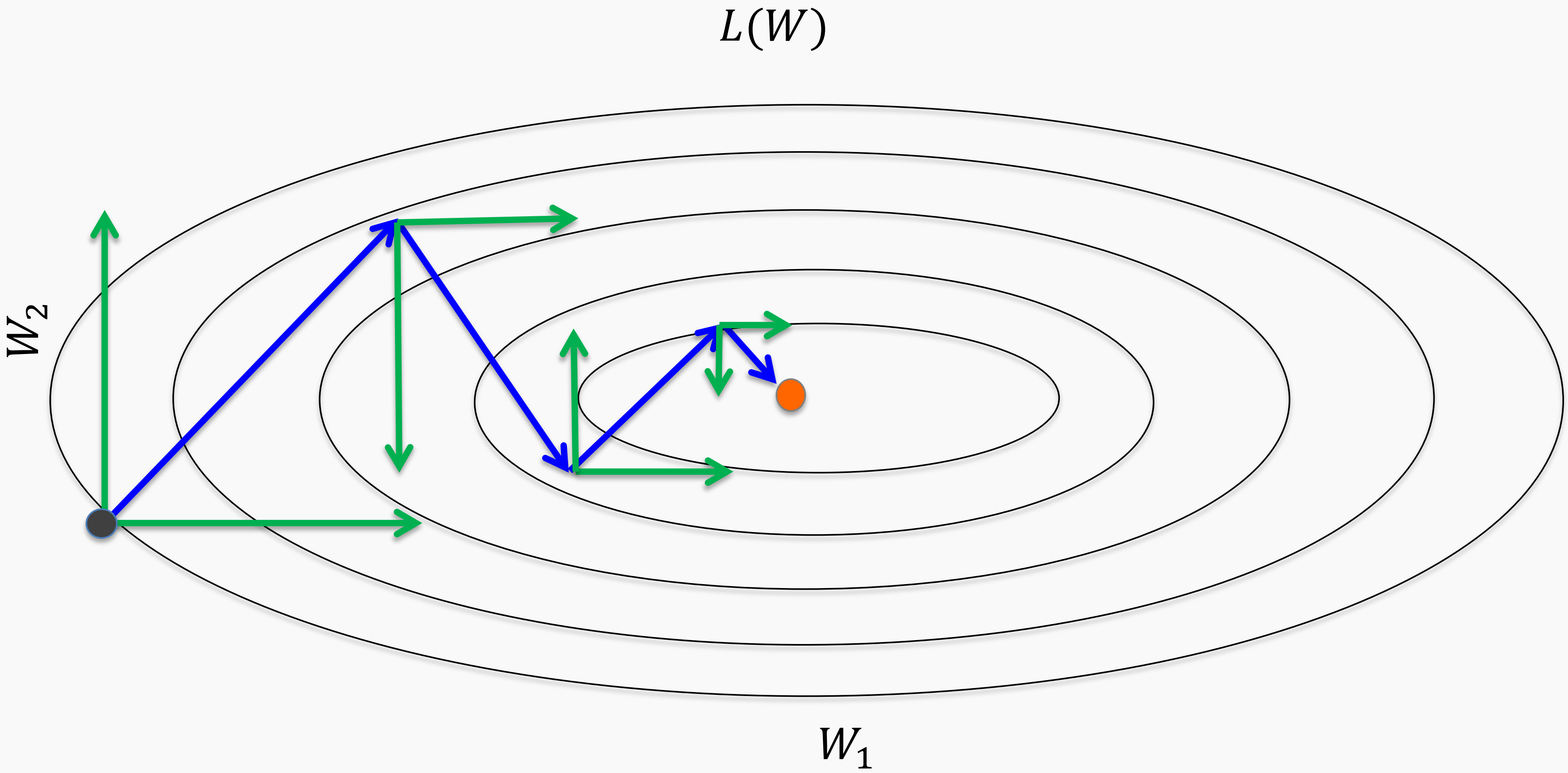
Momentum

Question: Why not this?



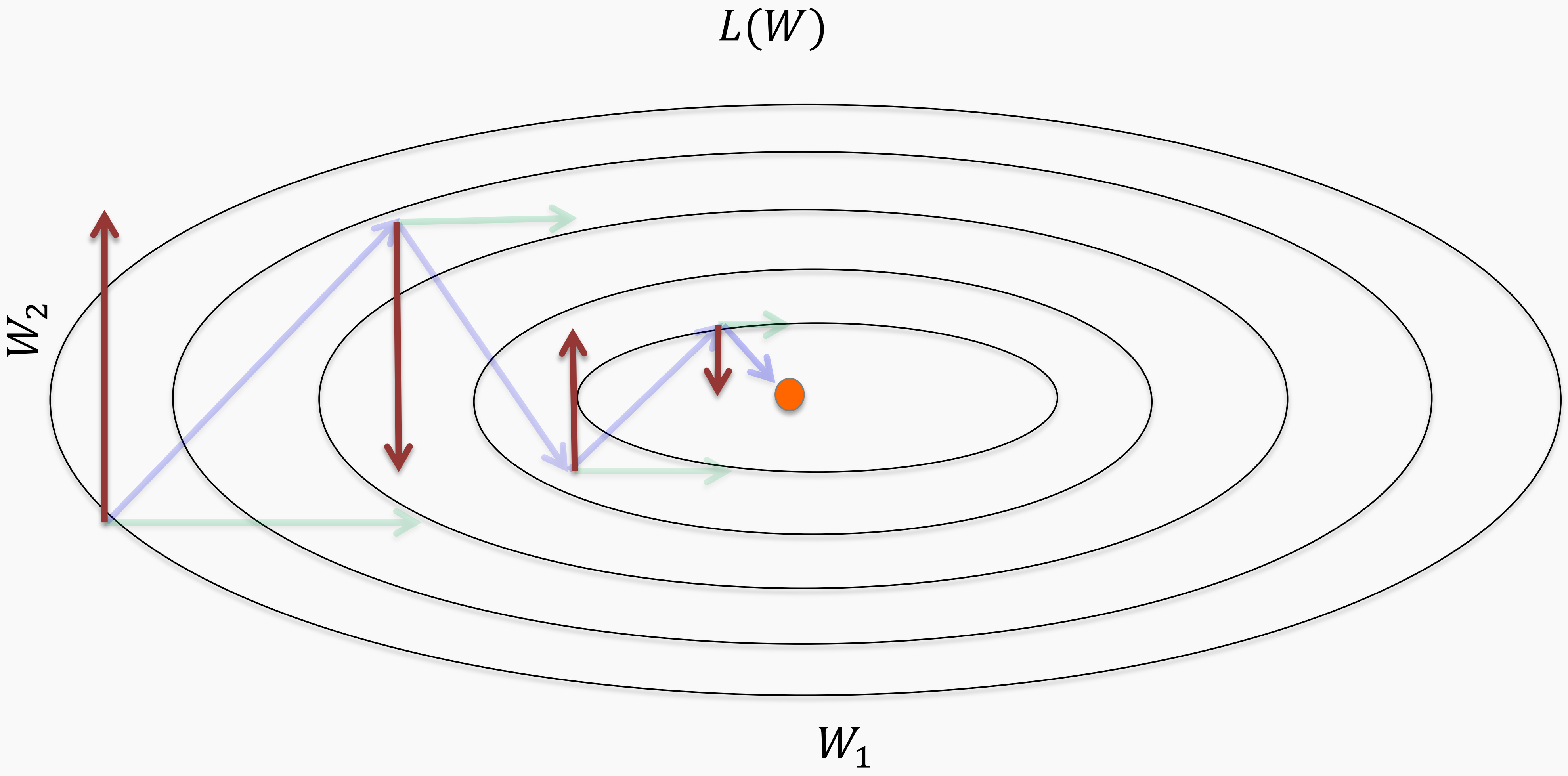
Momentum

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



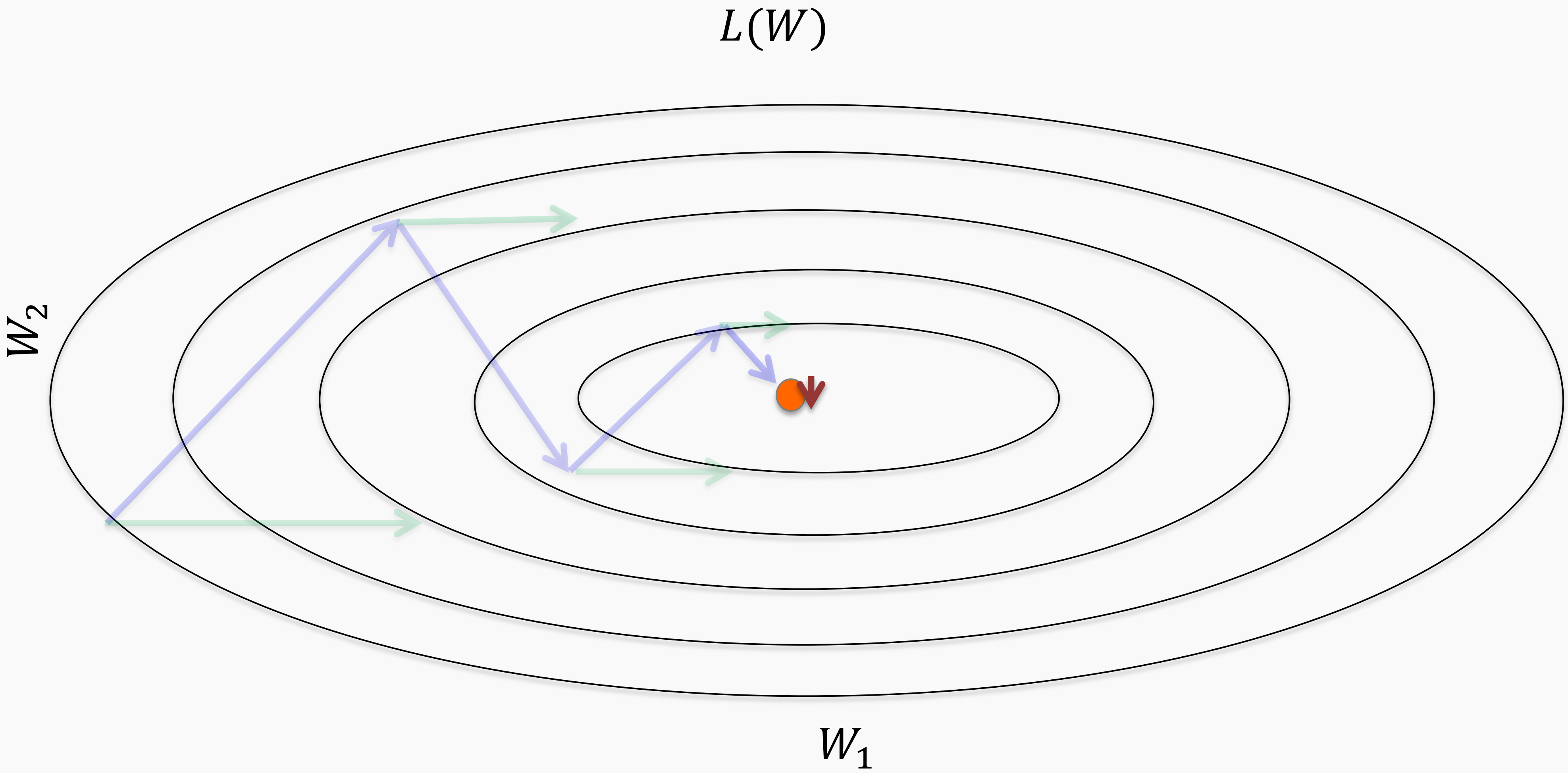
Momentum

Look each component at a time



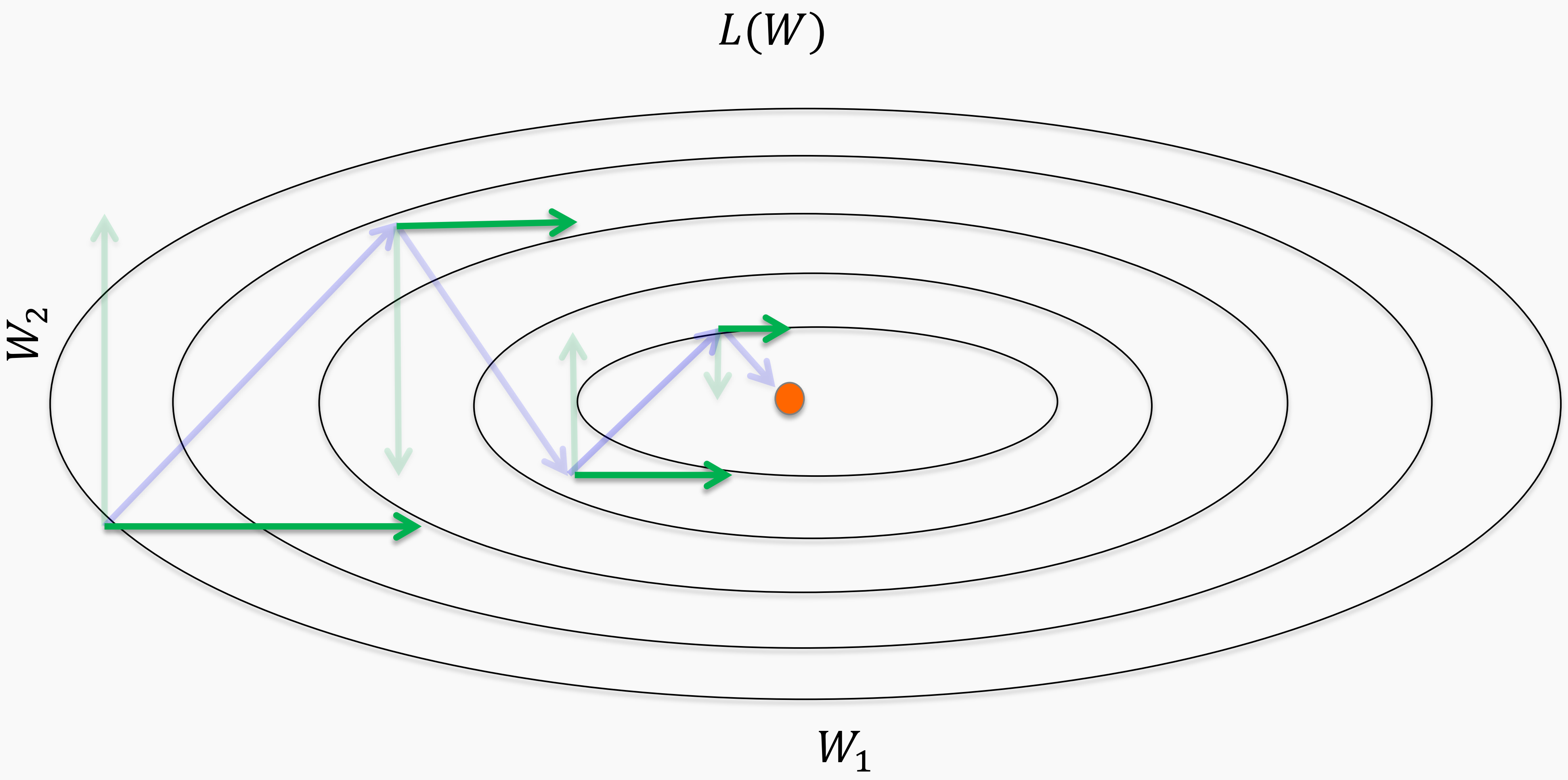
Momentum

Let us figure out an algorithm



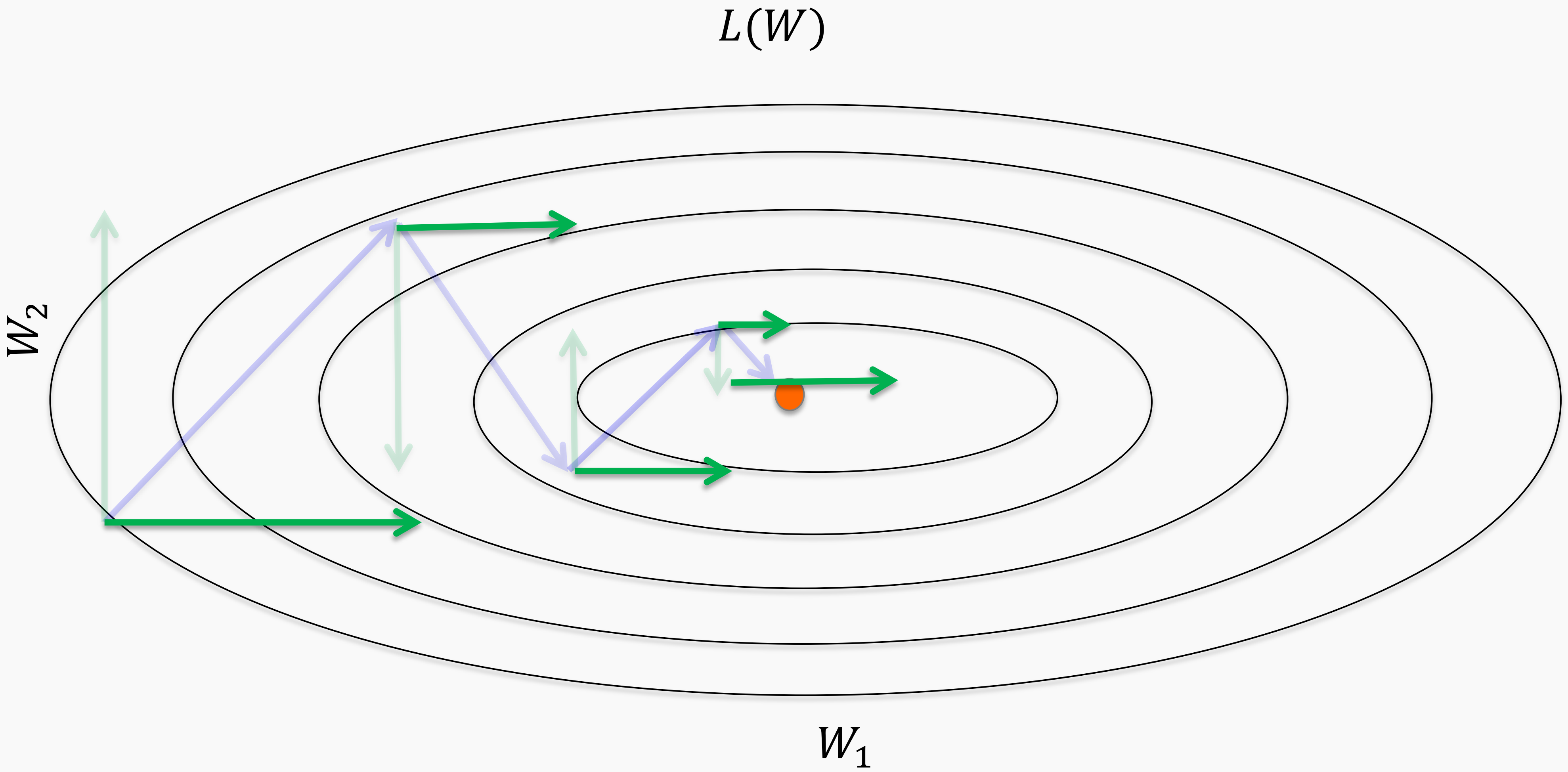
Momentum

Let us figure out an algorithm



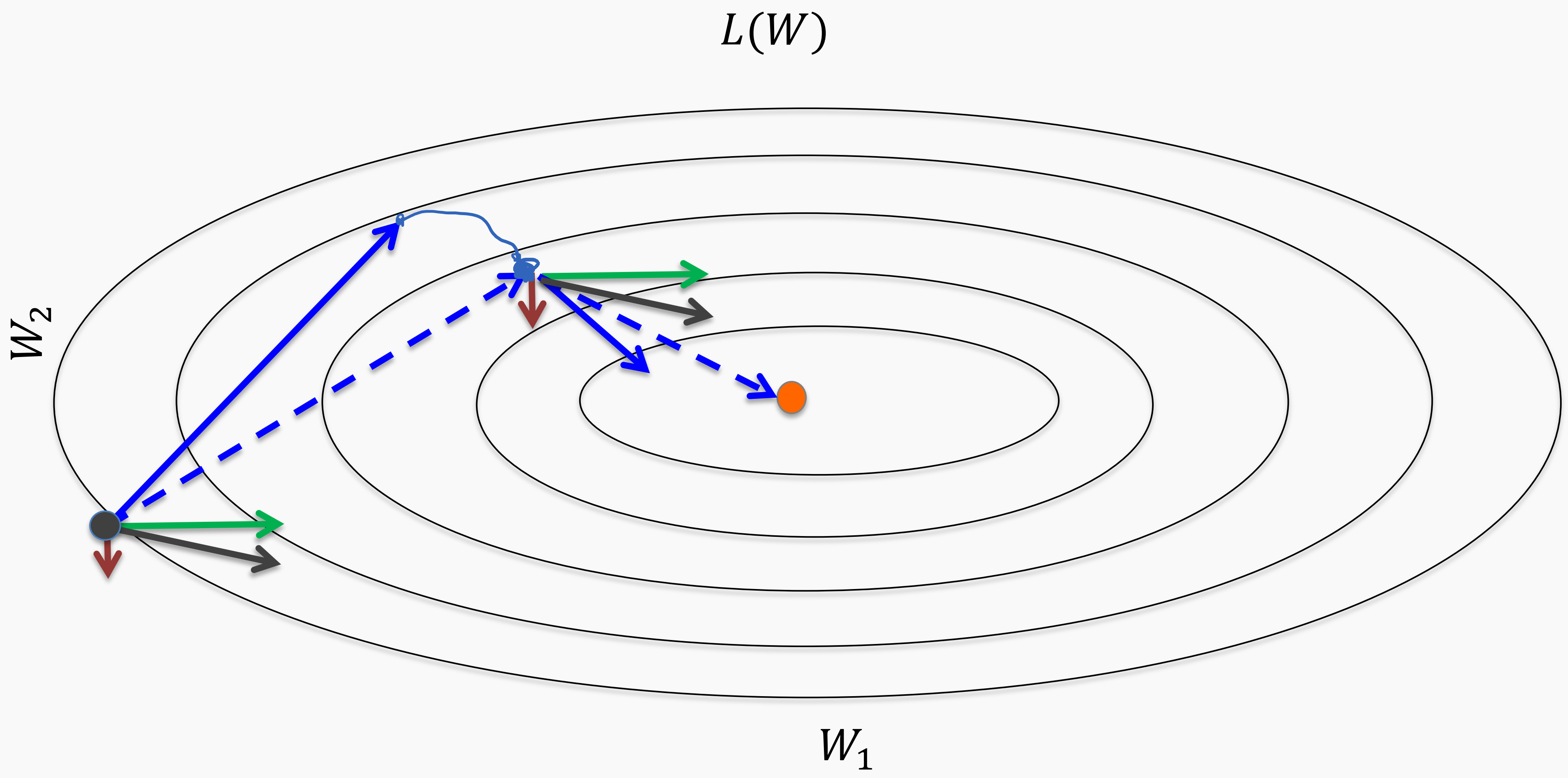
Momentum

Let us figure out an algorithm



Momentum

Let us figure out an algorithm



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:

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

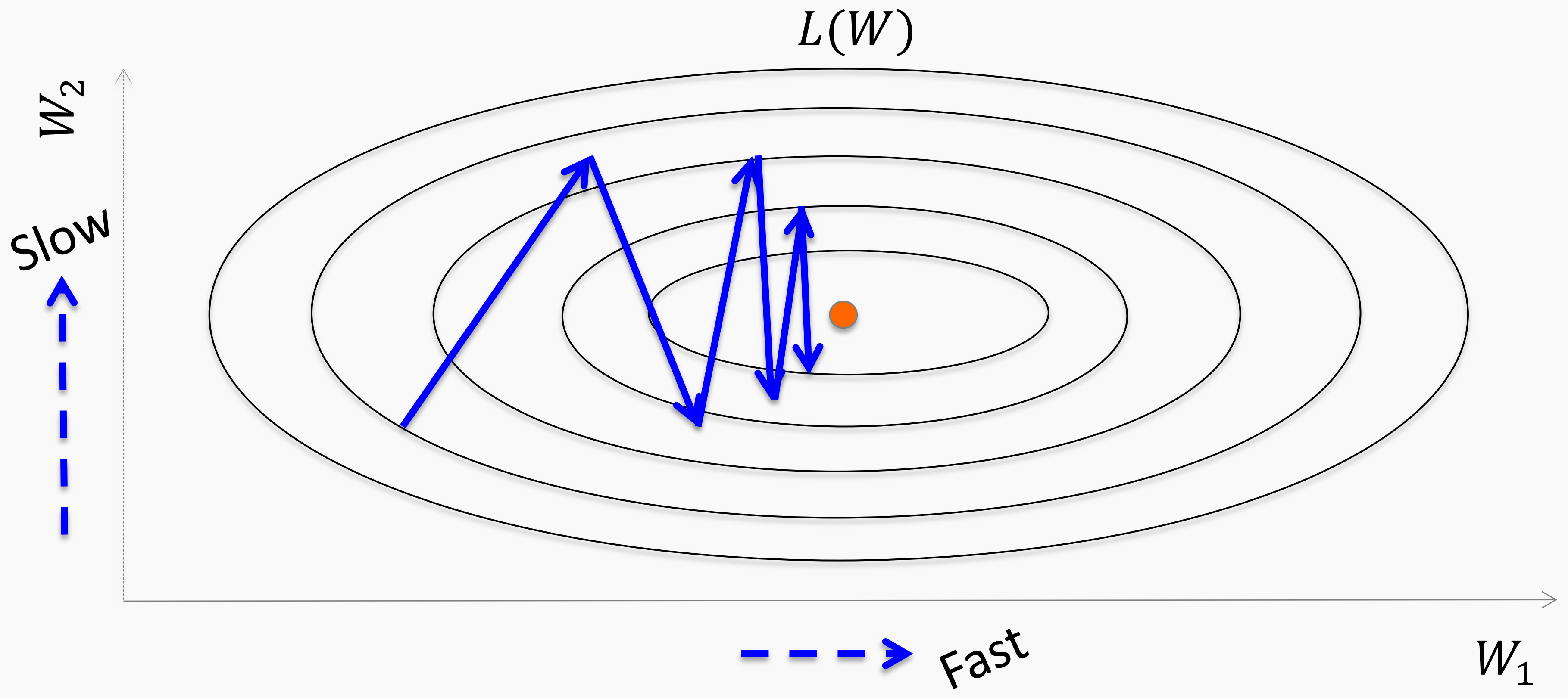
$\alpha \in [0,1)$ controls how quickly effect of past gradients decay

Outline

Optimization

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

Adaptive Learning Rates

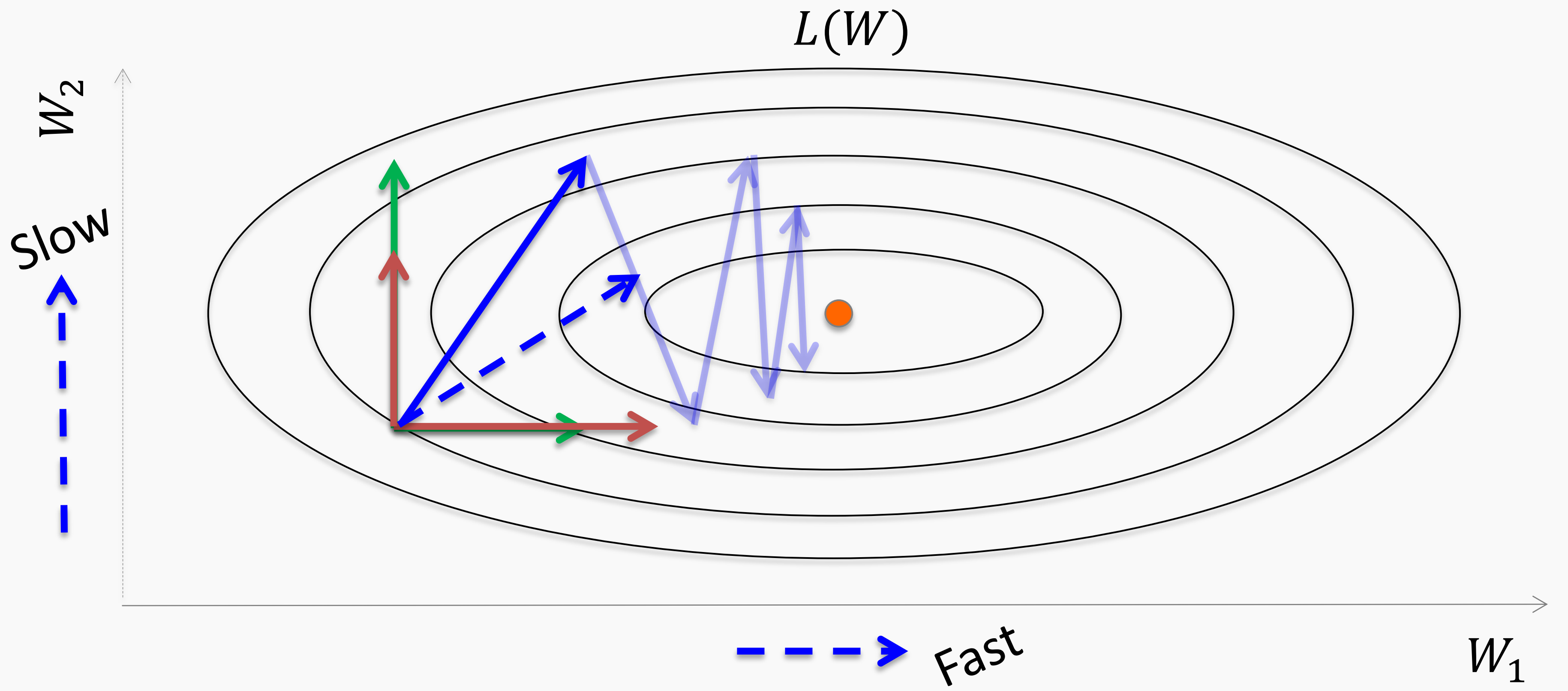


Oscillations along vertical direction

– Learning must be slower along parameter 2

Use a different learning rate for each parameter?

Adaptive Learning Rates

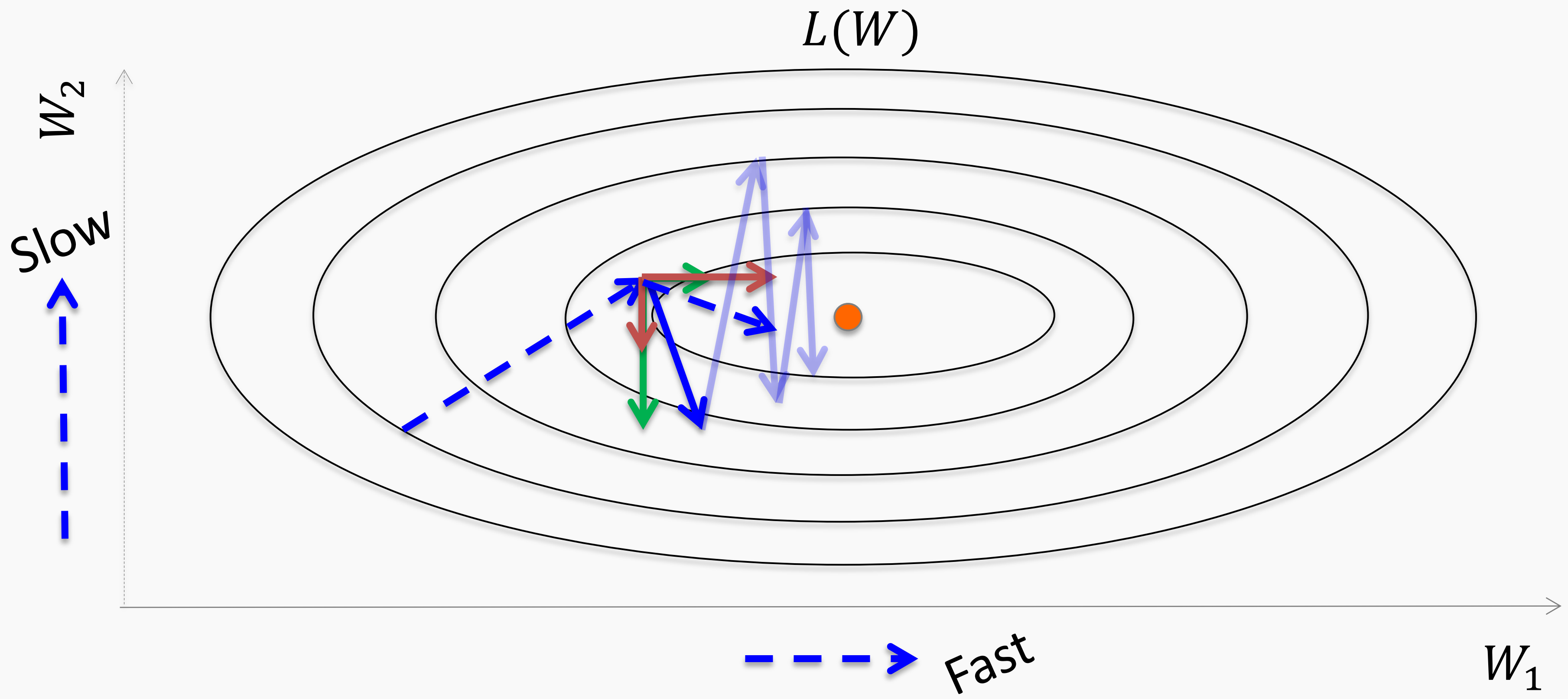


Oscillations along vertical direction

– Learning must be slower along parameter 2

Use a different learning rate for each parameter?

Adaptive Learning Rates

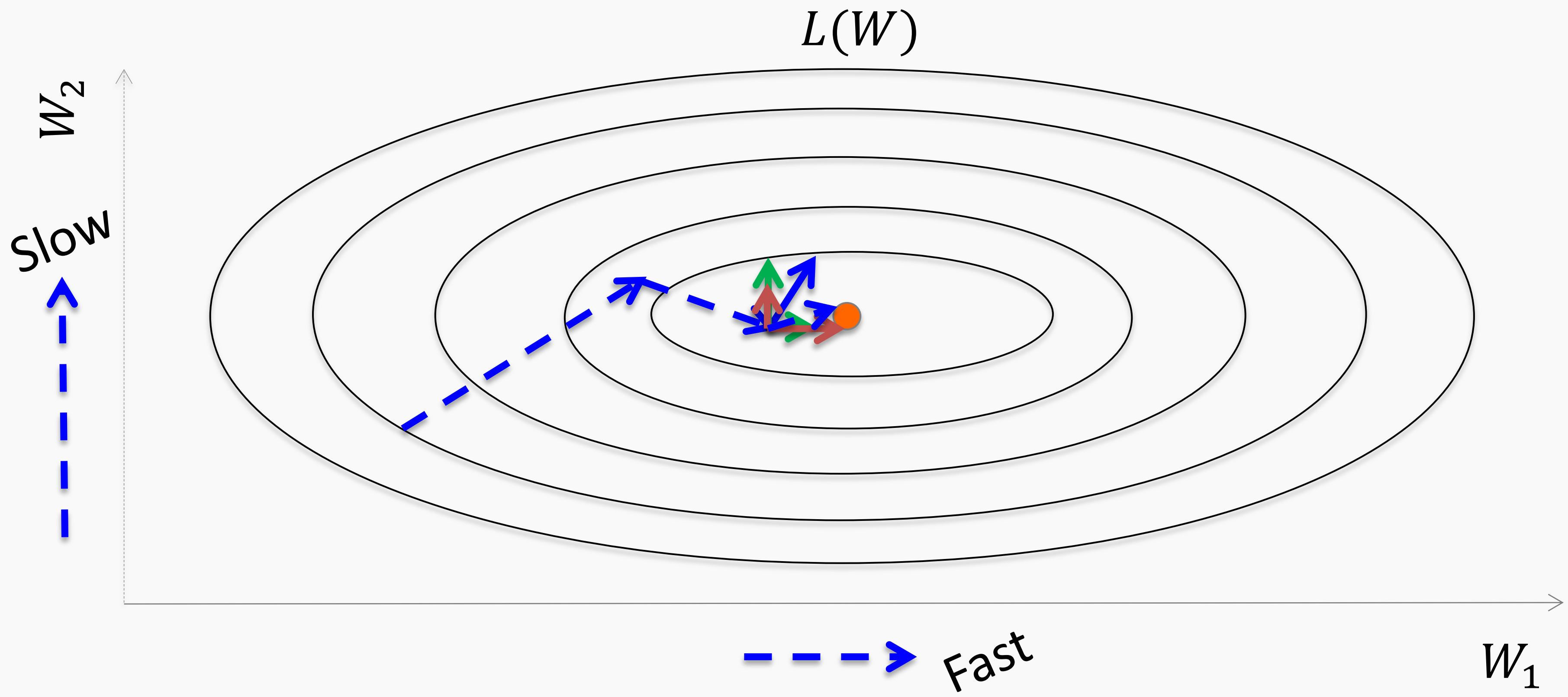


Oscillations along vertical direction

– Learning must be slower along parameter 2

Use a different learning rate for each parameter?

Adaptive Learning Rates

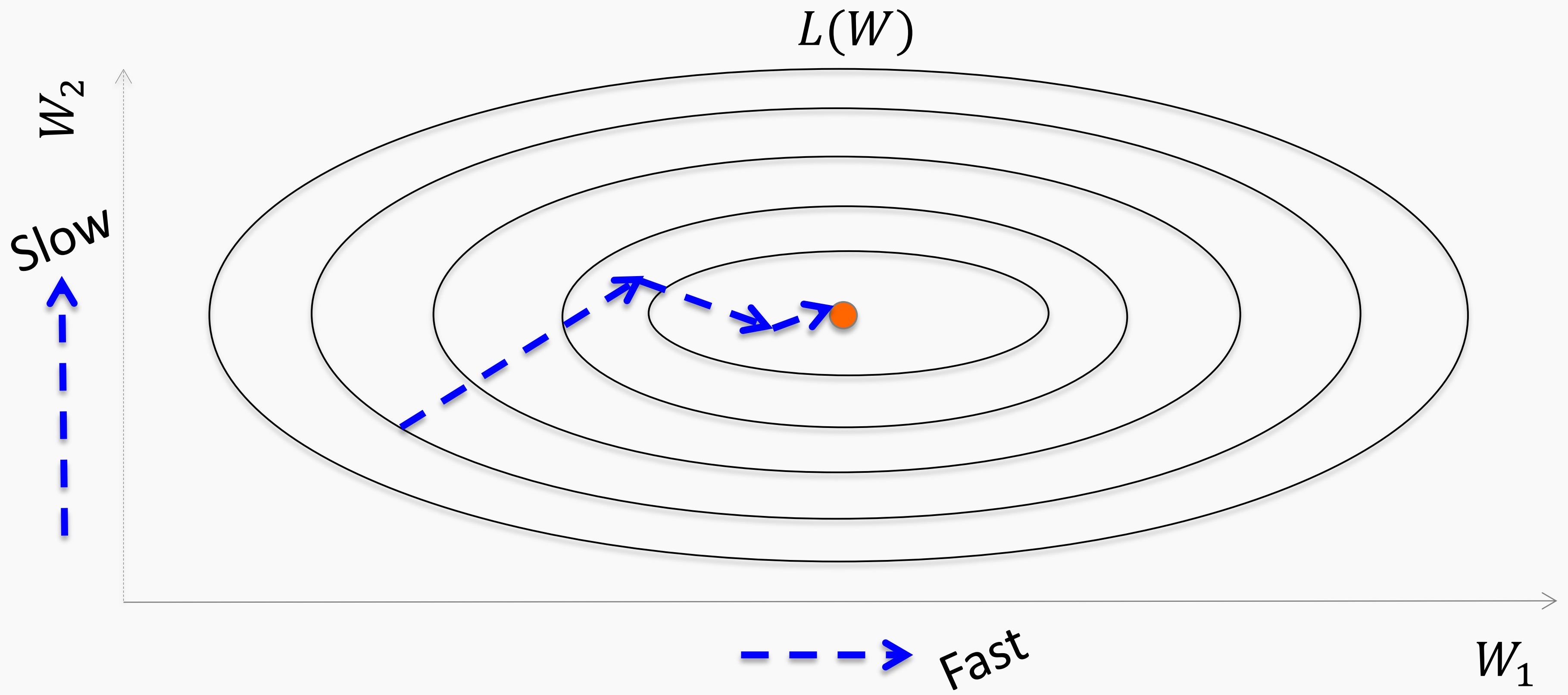


Oscillations along vertical direction

– 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

AdaGrad

δ is a small number, making sure this does not become too large

Old gradient descent:

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

We would like $\lambda' s$ not to be the same and inversely proportional to the $|g_i|$

$$W_i^* = W_i - \eta_i g_i \qquad \eta_i \propto \frac{1}{|g_i|} = \frac{1}{\delta + |g_i|}$$

New gradient descent with adaptive learning rate:

$$r_i^* = r_i + g_i^2 \qquad W_i^* = W_i - \frac{\epsilon}{\delta + \sqrt{r_i}} \mathrm{g_i}$$

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:

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

Also applies bias correction to v and r

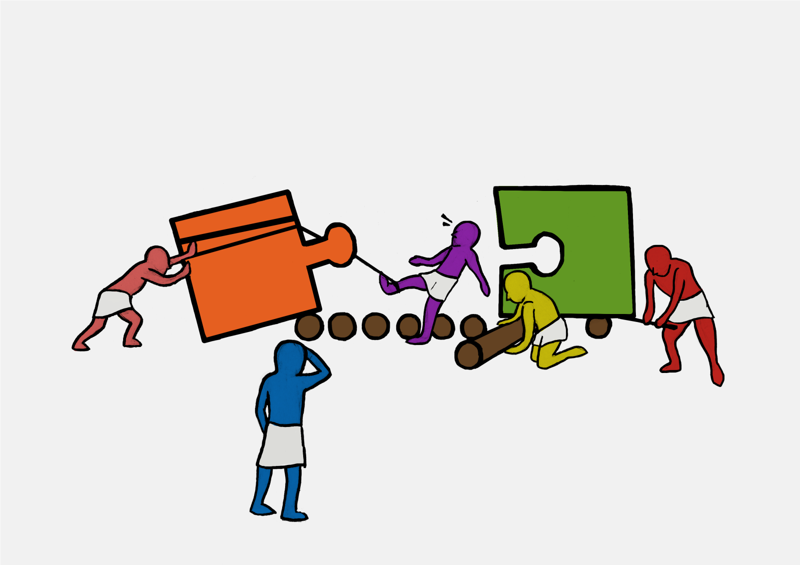
Works well in practice, it is fairly robust to hyper-parameters

Bias Correction

To performe bias correction on the two running average variables - ν and r use the following equations. Do this before they are used to update the weight.

- $\nu_{biascorr} = \nu/(1-\rho_1^t)$
- $r_{biascorr} = r/(1-\rho_2^t)$

where t is the number of the current iteration.



Momentum Weighting Parameter