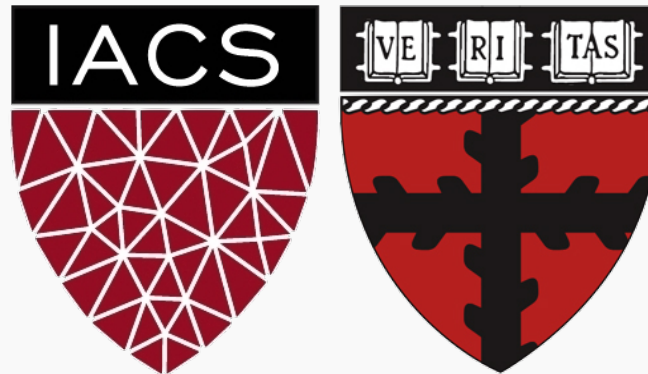


Optimizers

CS109B Data Science 2

Pavlos Protopapas, Mark Glickman





Brute Force



Greedy Search



**Non-Convex
optimization
using Gradient
Descent**

Outline

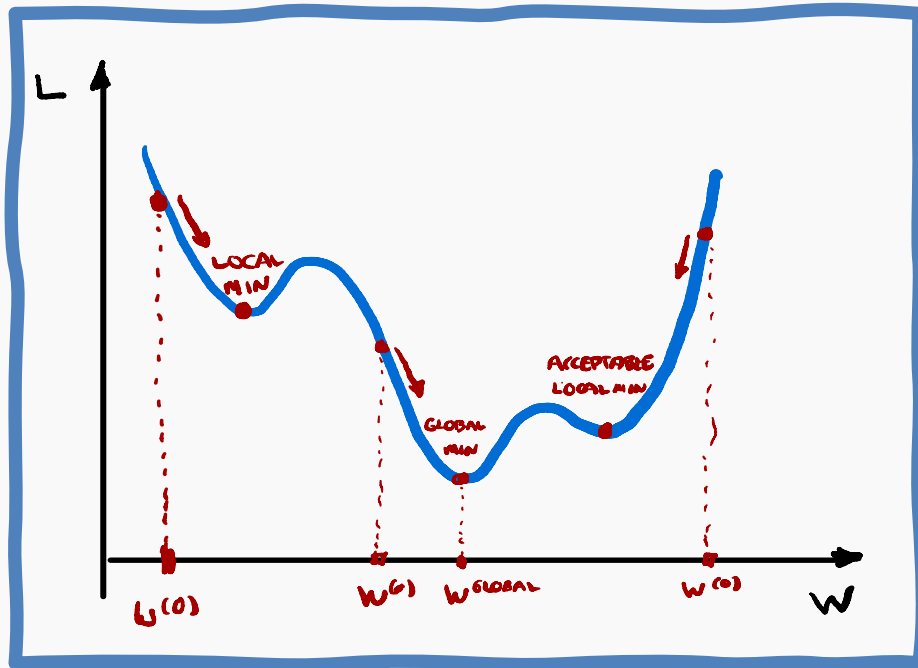
- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- Adam

Outline

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

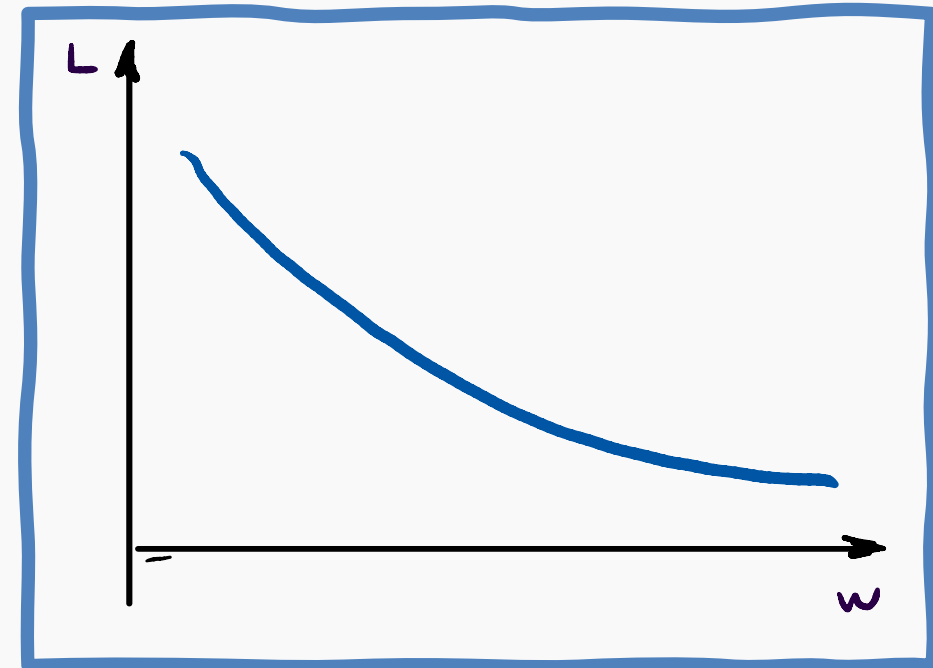
Challenges in Optimization

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.

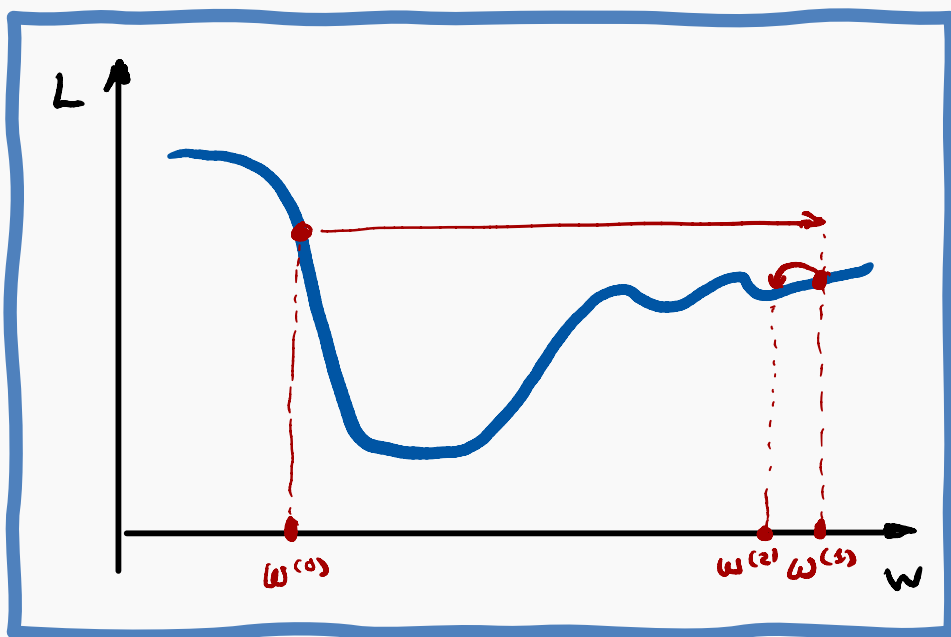
No critical points



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

Challenges in Optimization

Exploding Gradients

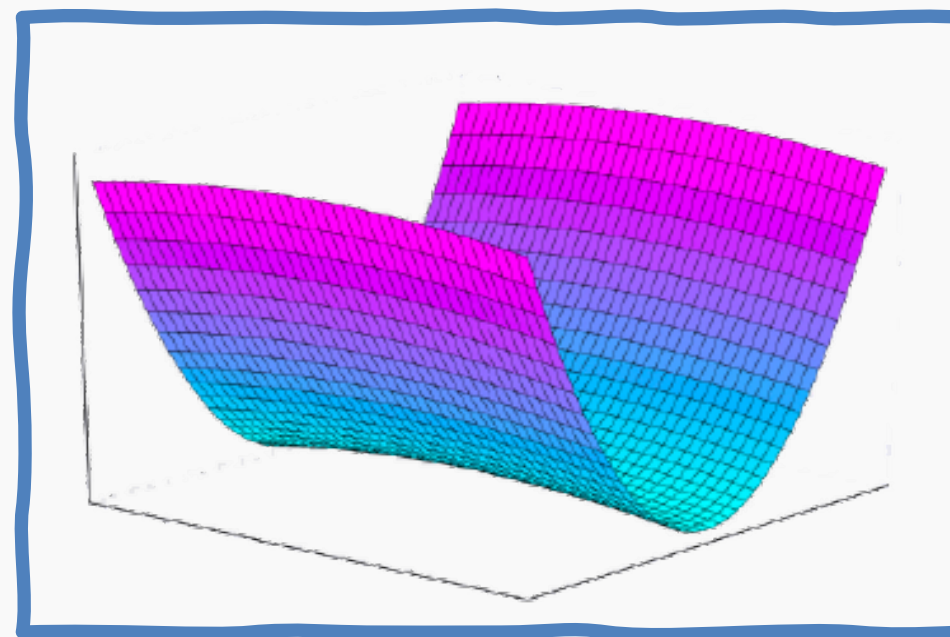


Exploding gradients due to cliffs. Can be mitigated using **gradient clipping**:

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

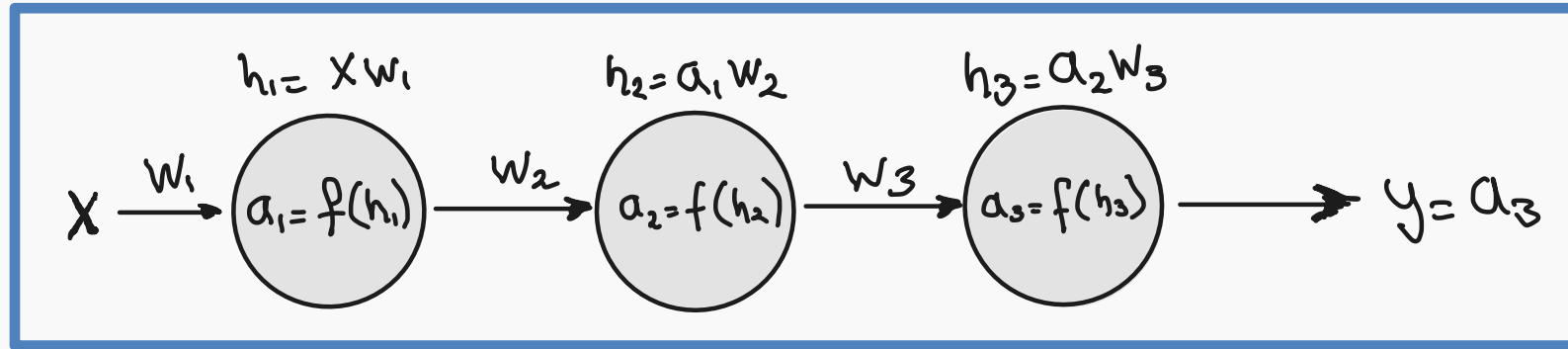
where u is user defined threshold.

Poor Conditioning



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

Challenges in Optimization: Vanishing Gradients



$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial a_2} \frac{\partial a_2}{\partial h_2} \frac{\partial h_2}{\partial W_2}$$

Diagram illustrating the chain rule for the gradient of the loss L with respect to weight W_2 :

- $\frac{\partial L}{\partial y}$ (red box) points to $f'()$.
- $\frac{\partial y}{\partial h_3}$ (red box) points to $f'()$.
- $\frac{\partial h_3}{\partial a_2}$ (green box) points to W_3 .
- $\frac{\partial a_2}{\partial h_2}$ (red box) points to $f'()$.
- $\frac{\partial h_2}{\partial W_2}$ (green box) points to a_1 .

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial y} f'() W_3 f'() a_1$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial a_2} \frac{\partial a_2}{\partial h_2} \frac{\partial h_2}{\partial a_1} \frac{\partial a_1}{\partial h_1} \frac{\partial h_1}{\partial W_1}$$

Diagram illustrating the chain rule for the gradient of the loss L with respect to weight W_1 :

- $\frac{\partial L}{\partial y}$ (red box) points to $f'()$.
- $\frac{\partial y}{\partial h_3}$ (red box) points to $f'()$.
- $\frac{\partial h_3}{\partial a_2}$ (green box) points to W_3 .
- $\frac{\partial a_2}{\partial h_2}$ (red box) points to $f'()$.
- $\frac{\partial h_2}{\partial a_1}$ (green box) points to W_2 .
- $\frac{\partial a_1}{\partial h_1}$ (red box) points to $f'()$.
- $\frac{\partial h_1}{\partial W_1}$ (green box) points to X .

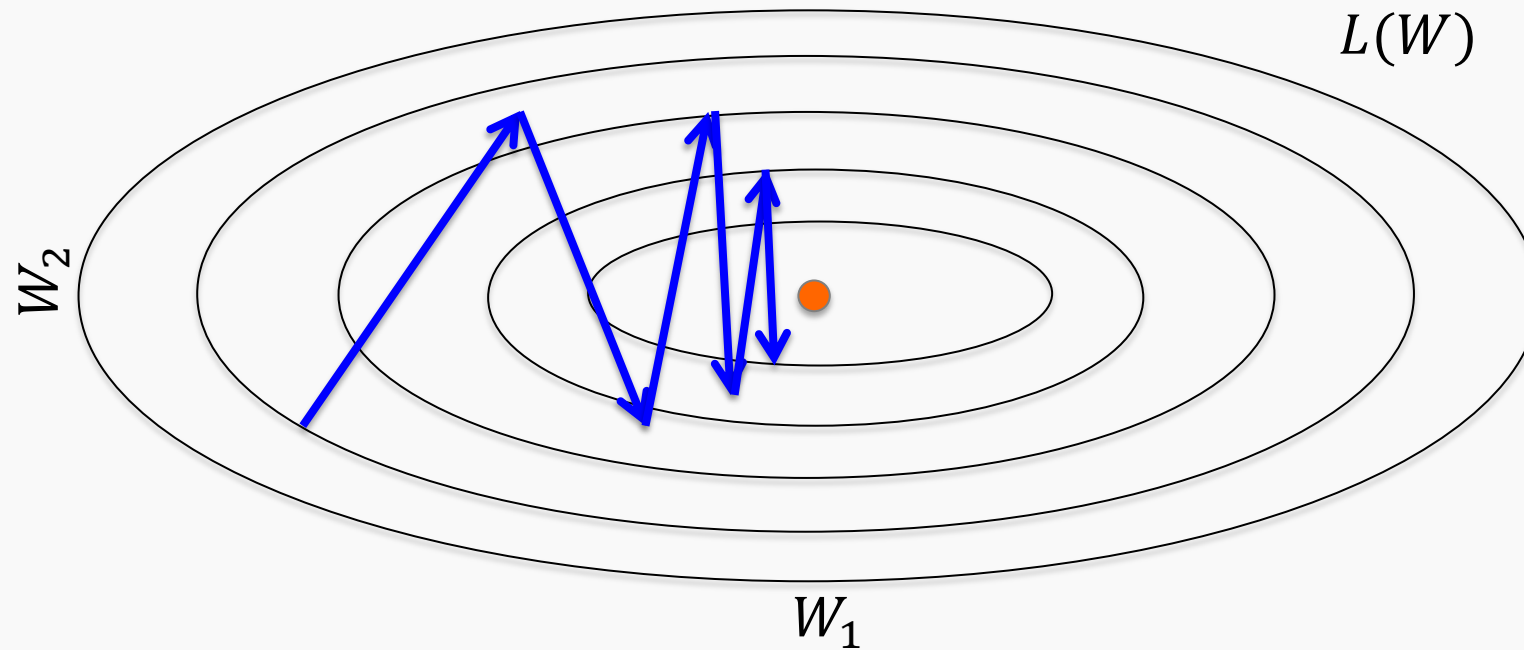
$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial y} f'() W_3 f'() W_2 f'() X$$

Outline

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

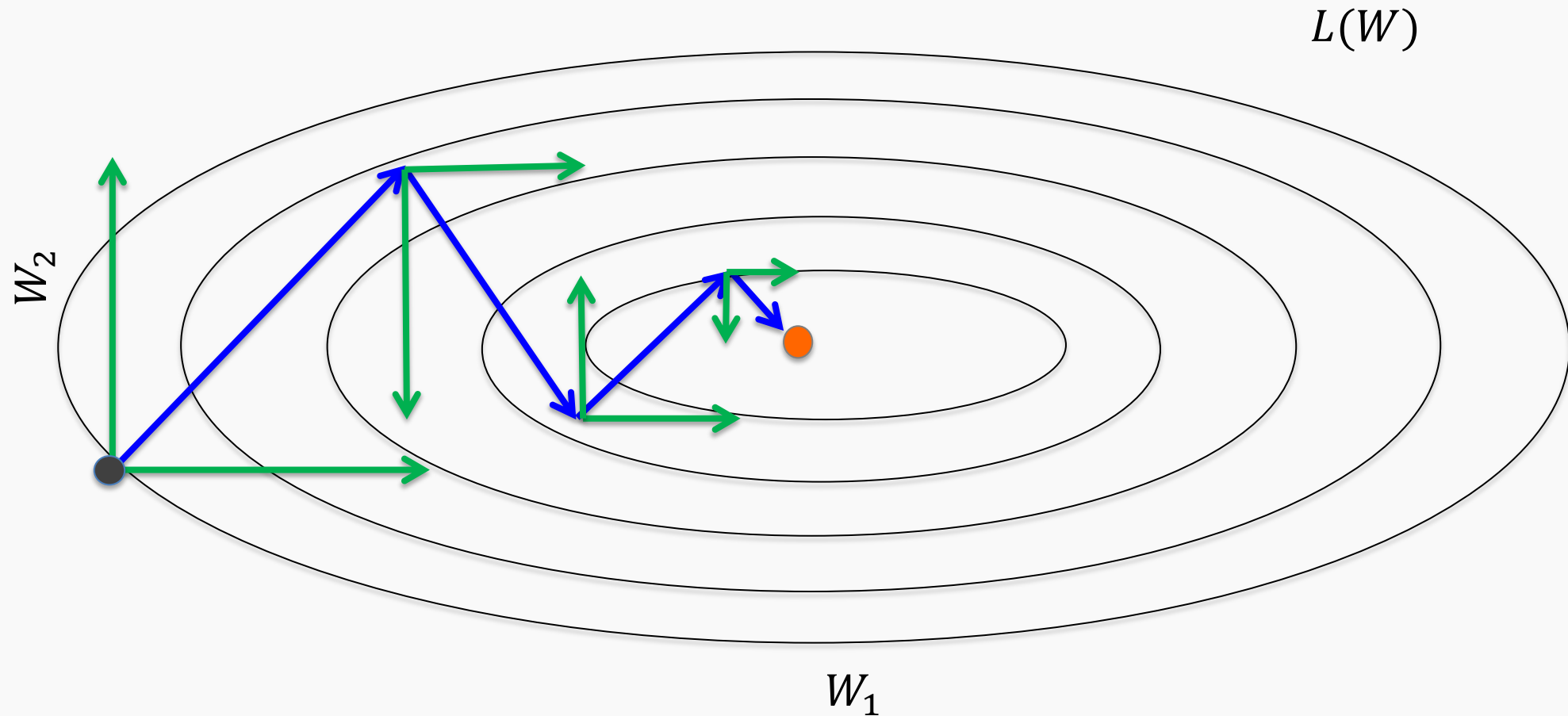
Momentum

Simple Gradient Descent oscillates because updates do not exploit curvature information



Momentum

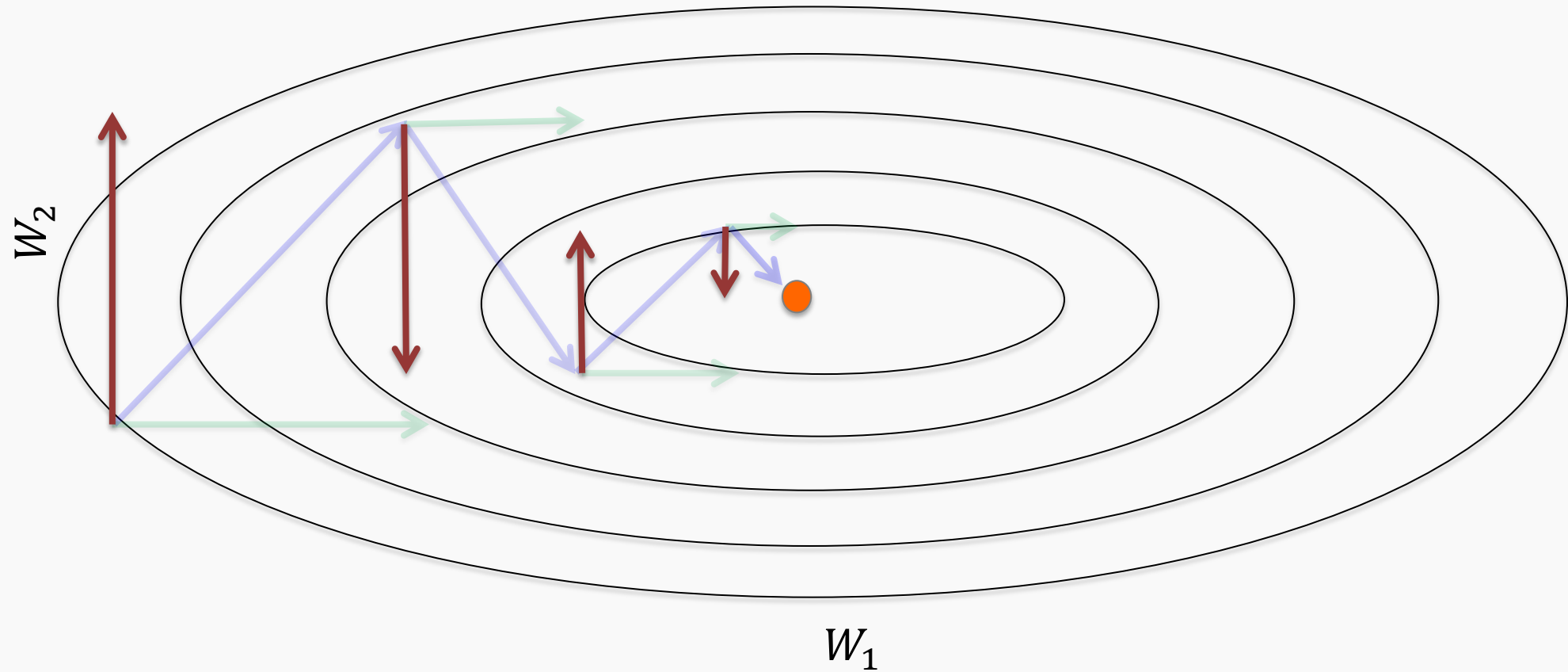
Let us figure out an algorithm which will converge to the minimum faster. We first examine the partial derivatives of the loss



Momentum

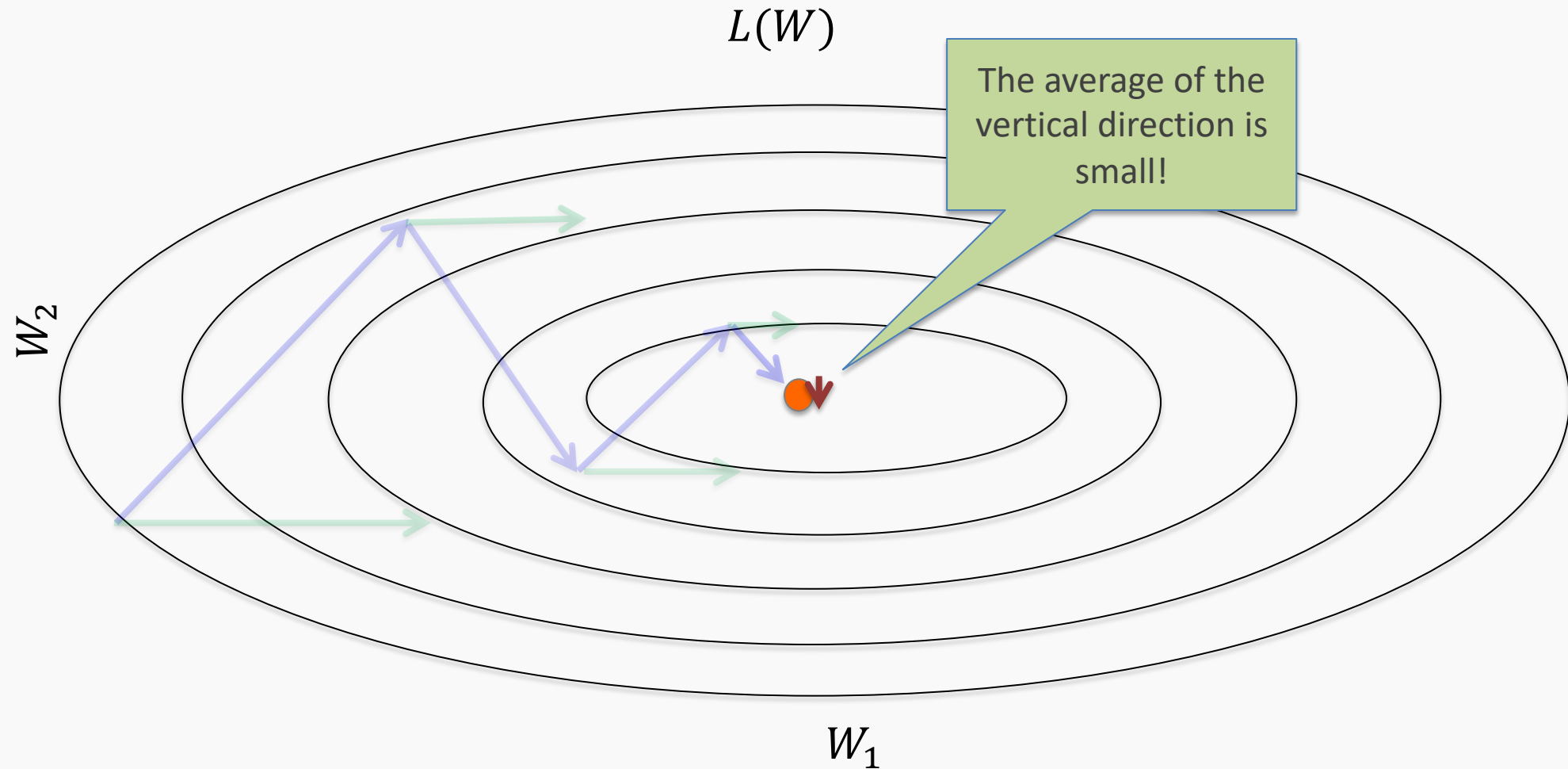
Look each component at a time. And see the average behavior of each component

$$L(W)$$



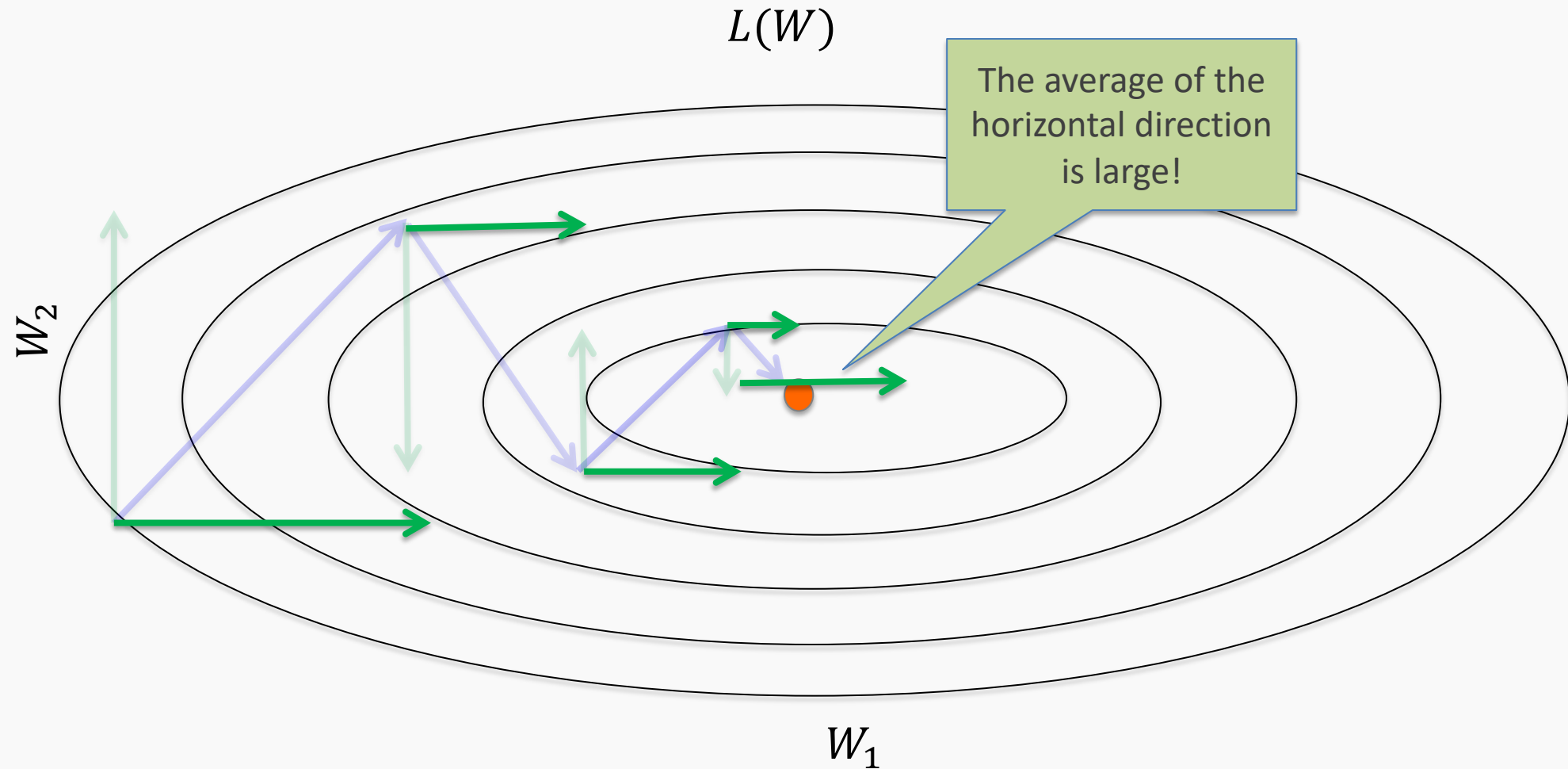
Momentum

Let us figure out an algorithm



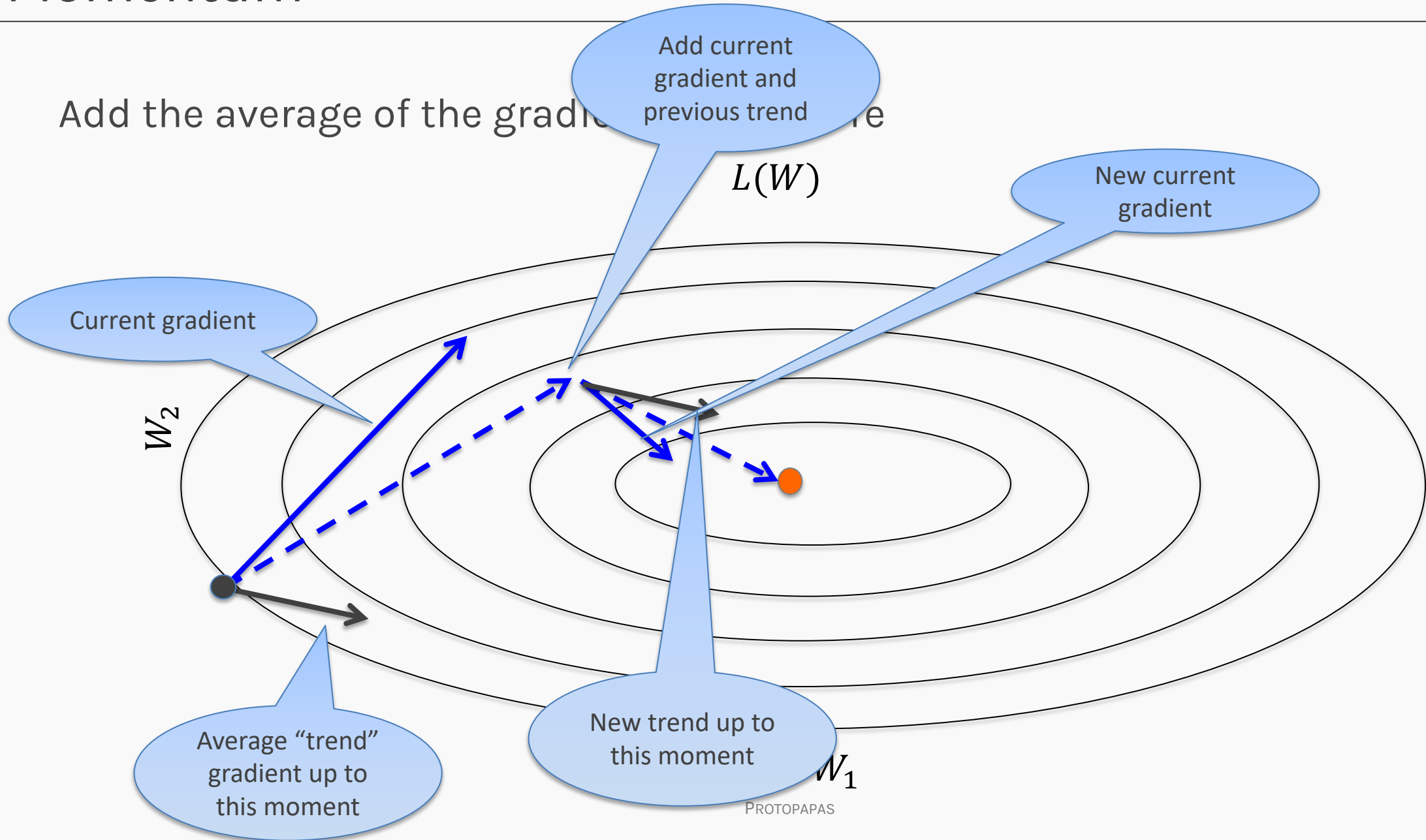
Momentum

Let us figure out an algorithm



Momentum

Add the average of the gradient and previous trend



Momentum

f is the Neural Network

Old gradient descent:

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

Momentum

f is the Neural Network

Old gradient descent:

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

$$W^* = W - \eta g$$

If $\alpha = 0$ old SGD
If $\alpha = 1$ we only consider the trend
Typical: $\alpha = 0.9 - 0.99$

New gradient descent with momentum

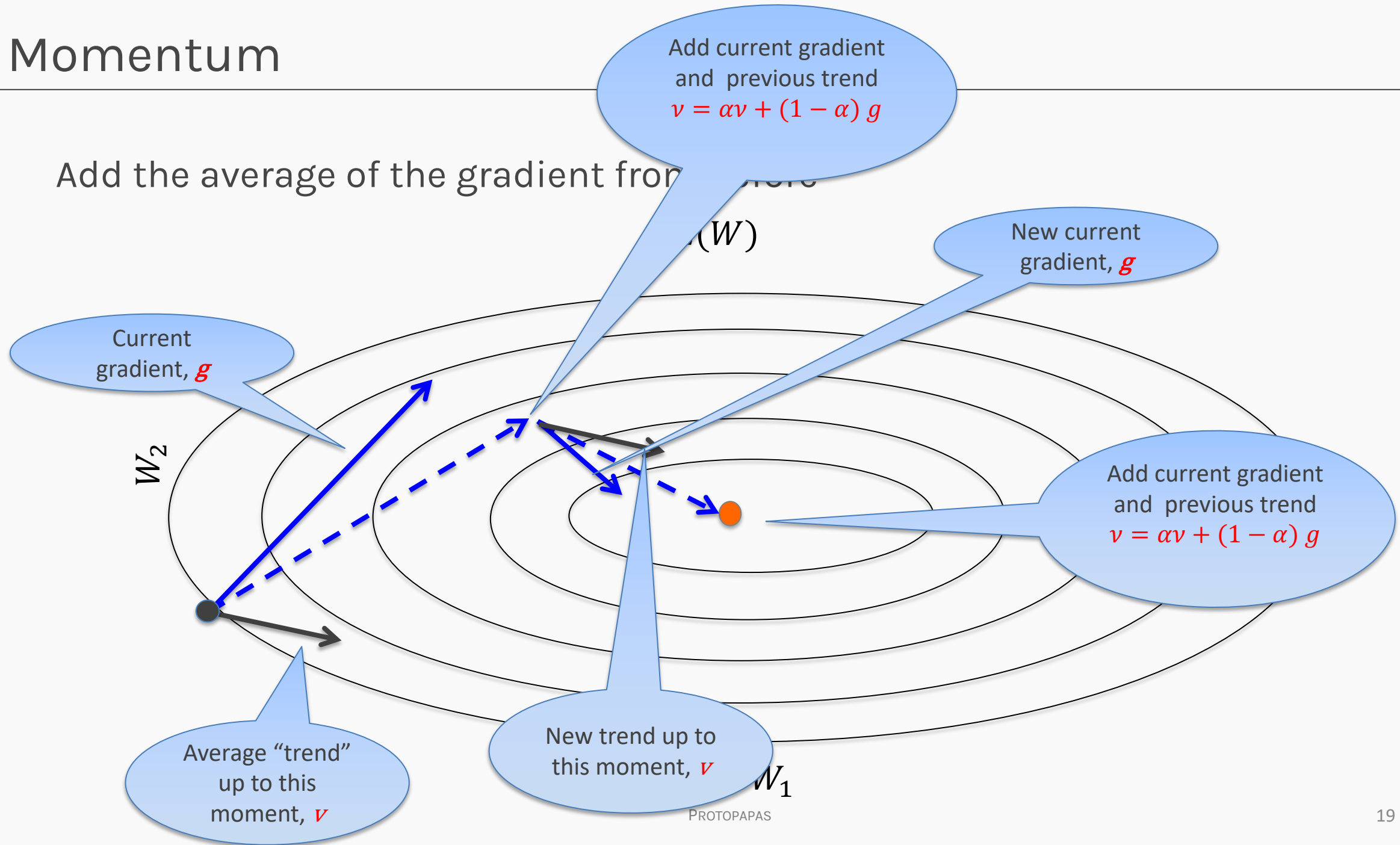
$$v = \alpha v + (1 - \alpha) g$$

$$W^* = W - \eta v$$

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

Momentum

Add the average of the gradient from previous steps



Nesterov Momentum

It is a slightly different version of the momentum update that has recently been gaining popularity. And it has better theoretical convergence guarantees (at least for convex functions).

The idea is to look ahead of the weights and apply an interim update:

$$v = \alpha v + (1 - \alpha) g$$

$$\tilde{W} = W - \eta v$$

Nesterov Momentum

$$v = \alpha v + (1 - \alpha) g$$

$$\tilde{W} = W - \eta v$$

Re-calculate the gradient, \tilde{g} , with the new weight, \tilde{W}

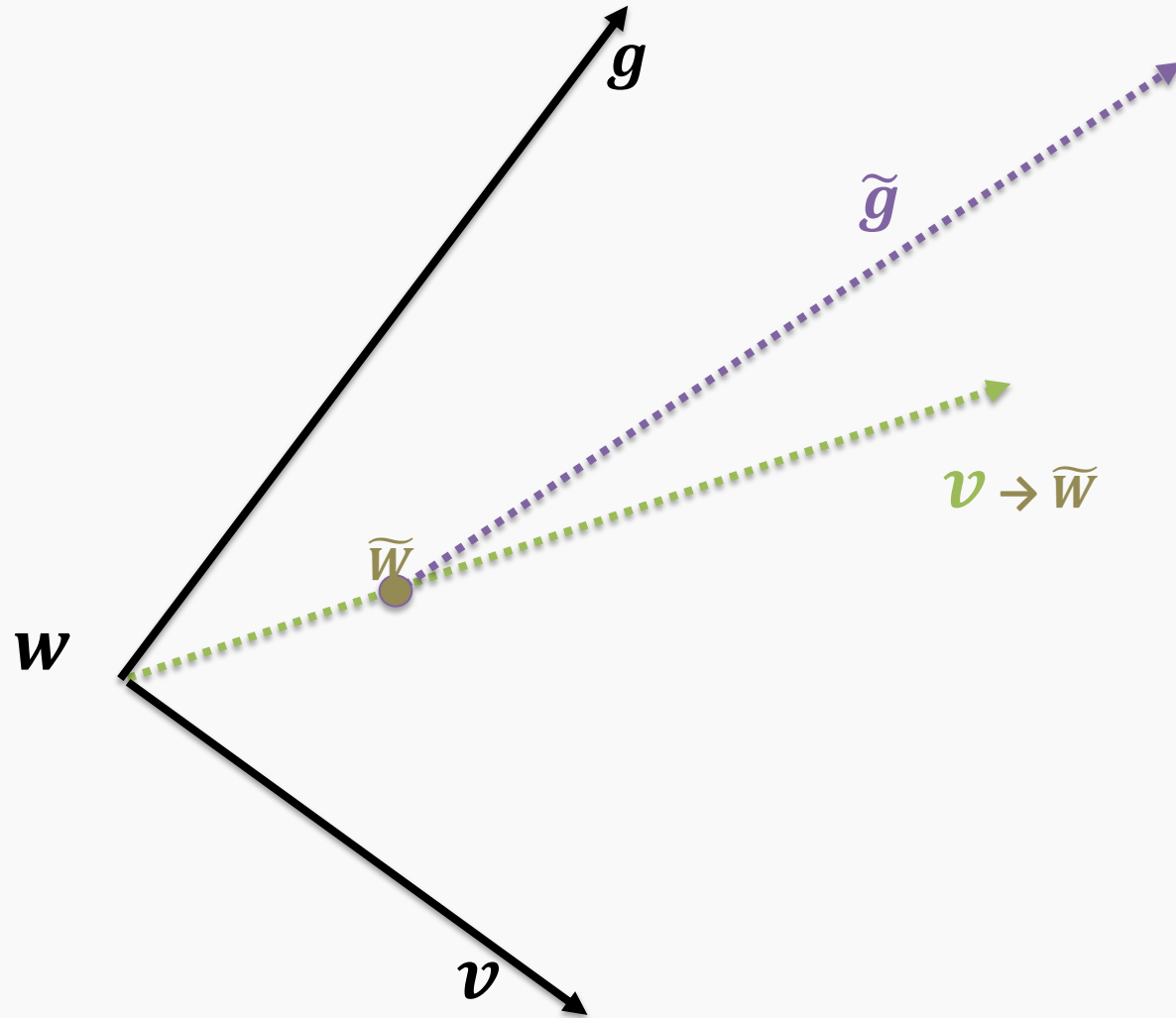
$$\tilde{g} = \frac{1}{m} \sum_i \nabla_W L(f(x_i; \tilde{W}), y_i)$$

Find a new momentum, \tilde{v} , with the new intermediate gradient, \tilde{g} . And update using gradient descent.

$$\tilde{v} = \alpha \tilde{v} + (1 - \alpha) \tilde{g}$$

$$W = W - \eta \tilde{v}$$

Nesterov Momentum visually

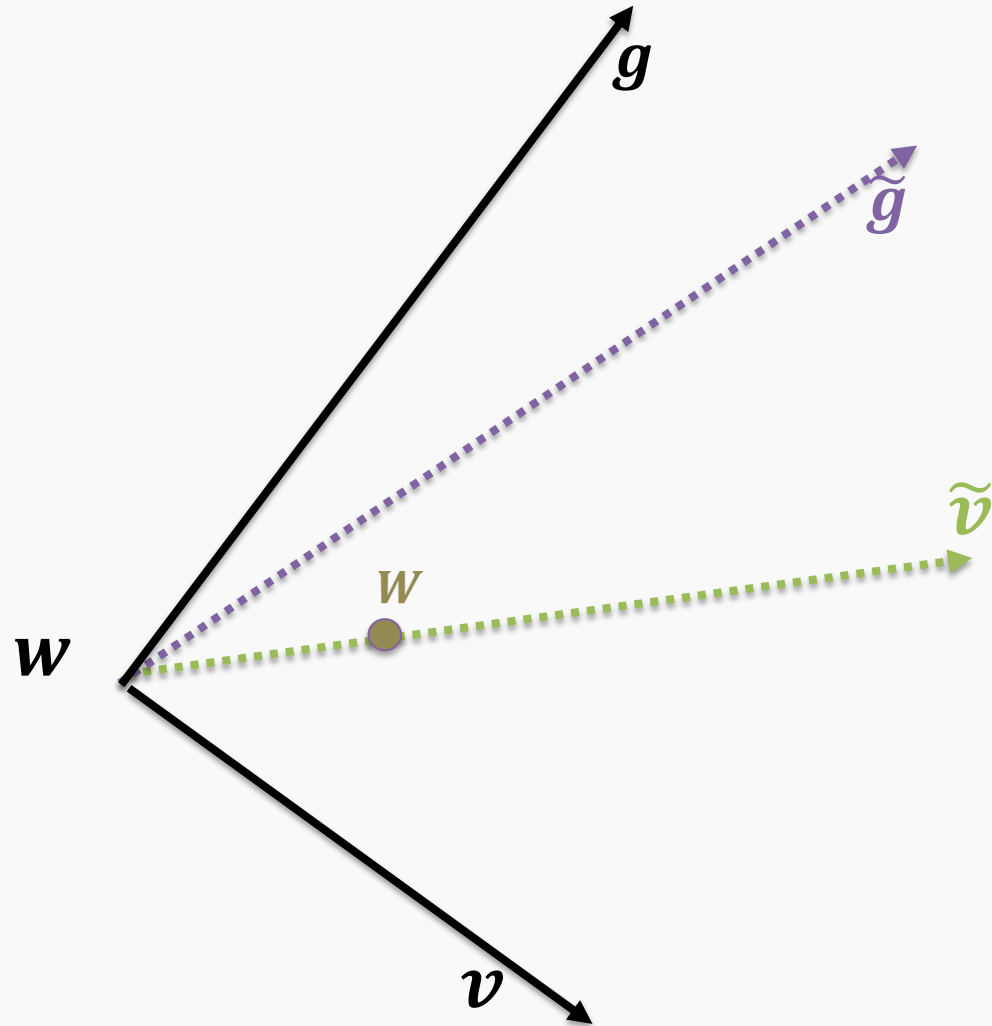


$$v = \alpha v + (1 - \alpha) g$$

$$\tilde{W} = W - \eta v$$

$$\tilde{g} = \frac{1}{m} \sum_i \nabla_W L(f(x_i; \tilde{W}), y_i)$$

Nesterov Momentum visually



$$v = \alpha v + (1 - \alpha) g$$

$$\tilde{W} = W - \eta v$$

$$\tilde{g} = \frac{1}{m} \sum_i \nabla_W L(f(x_i; \tilde{W}), y_i)$$

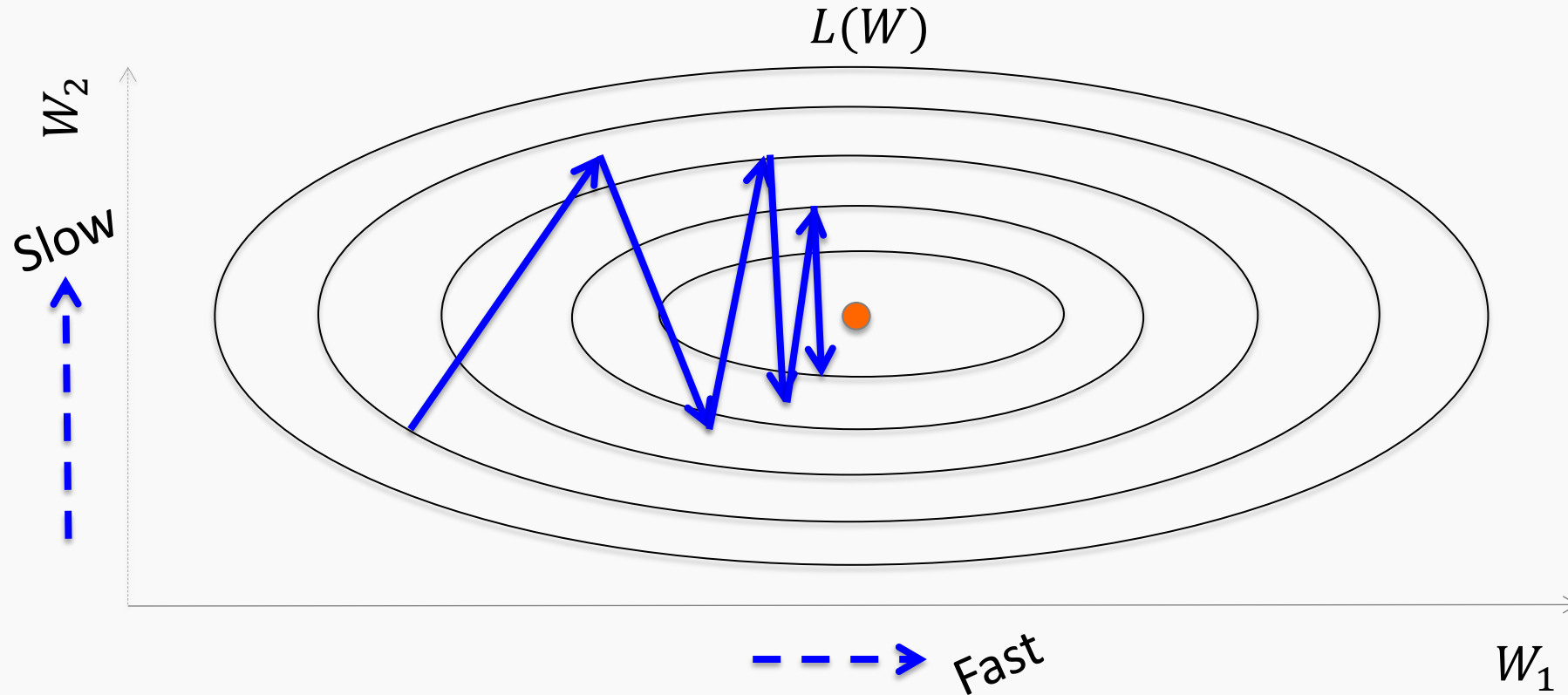
$$\tilde{v} = \alpha \tilde{v} + (1 - \alpha) \tilde{g}$$

$$W = W - \eta \tilde{v}$$

Outline

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

Adaptive Learning Rates

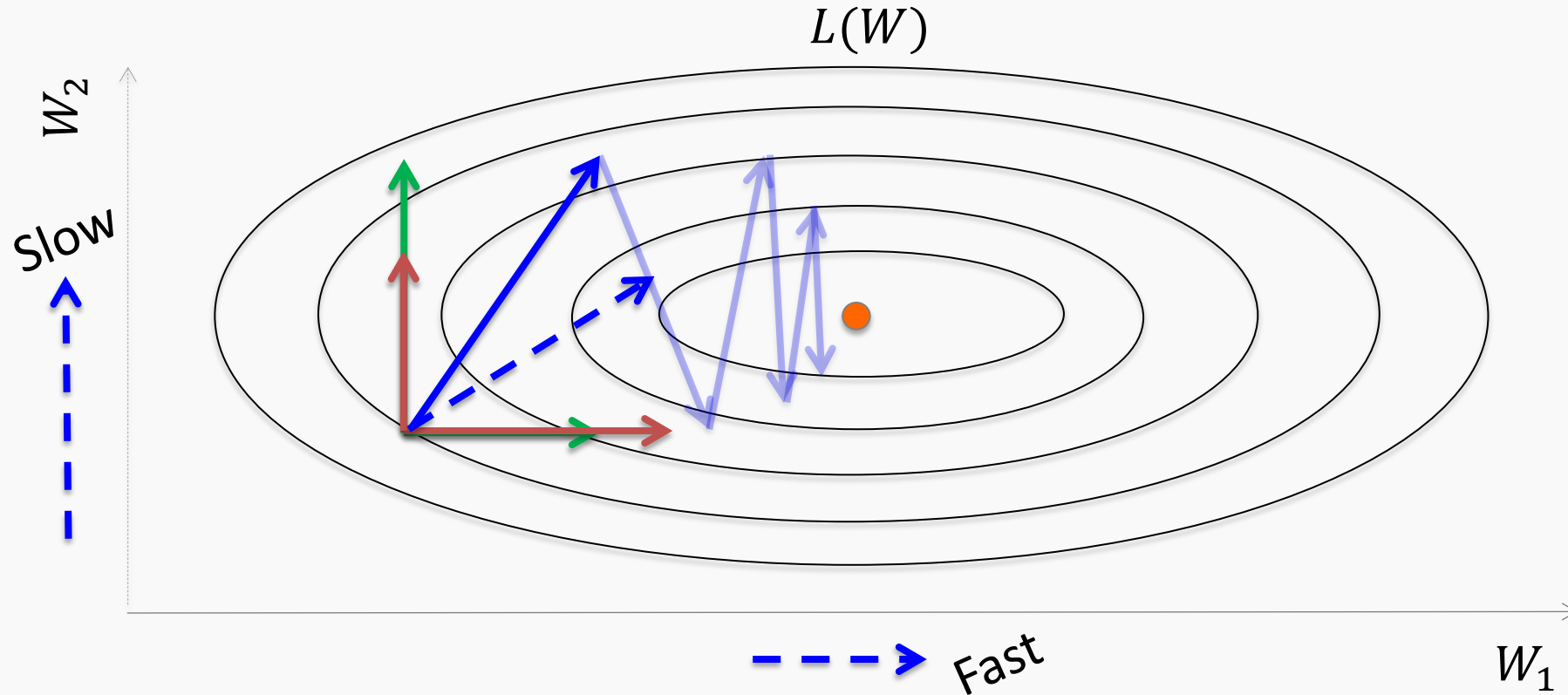


Oscillations along vertical direction

- Learning must be slower along parameter W_2

Use a different learning rate for each parameter?

Adaptive Learning Rates

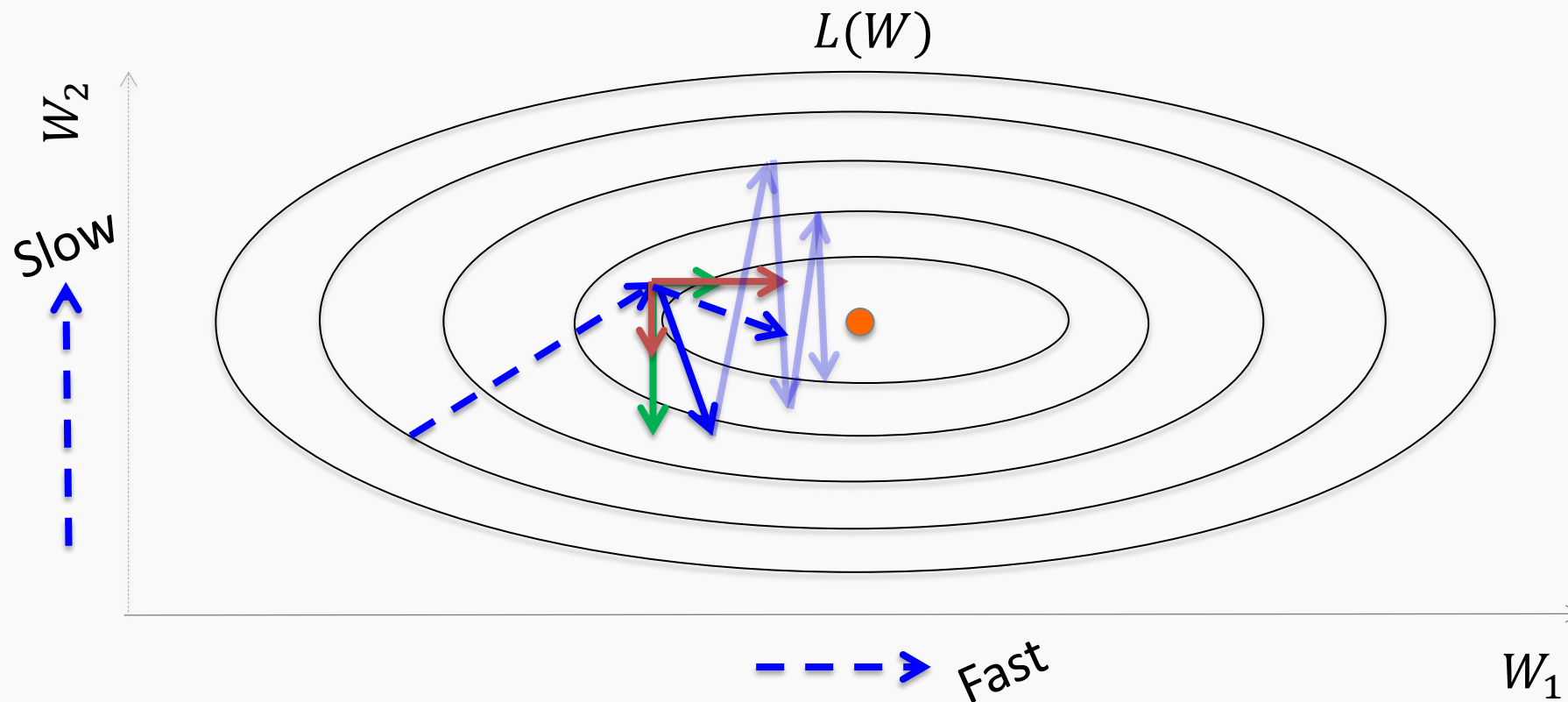


Oscillations along vertical direction

- Learning must be slower along parameter W_2

Use a different learning rate for each parameter?

Adaptive Learning Rates

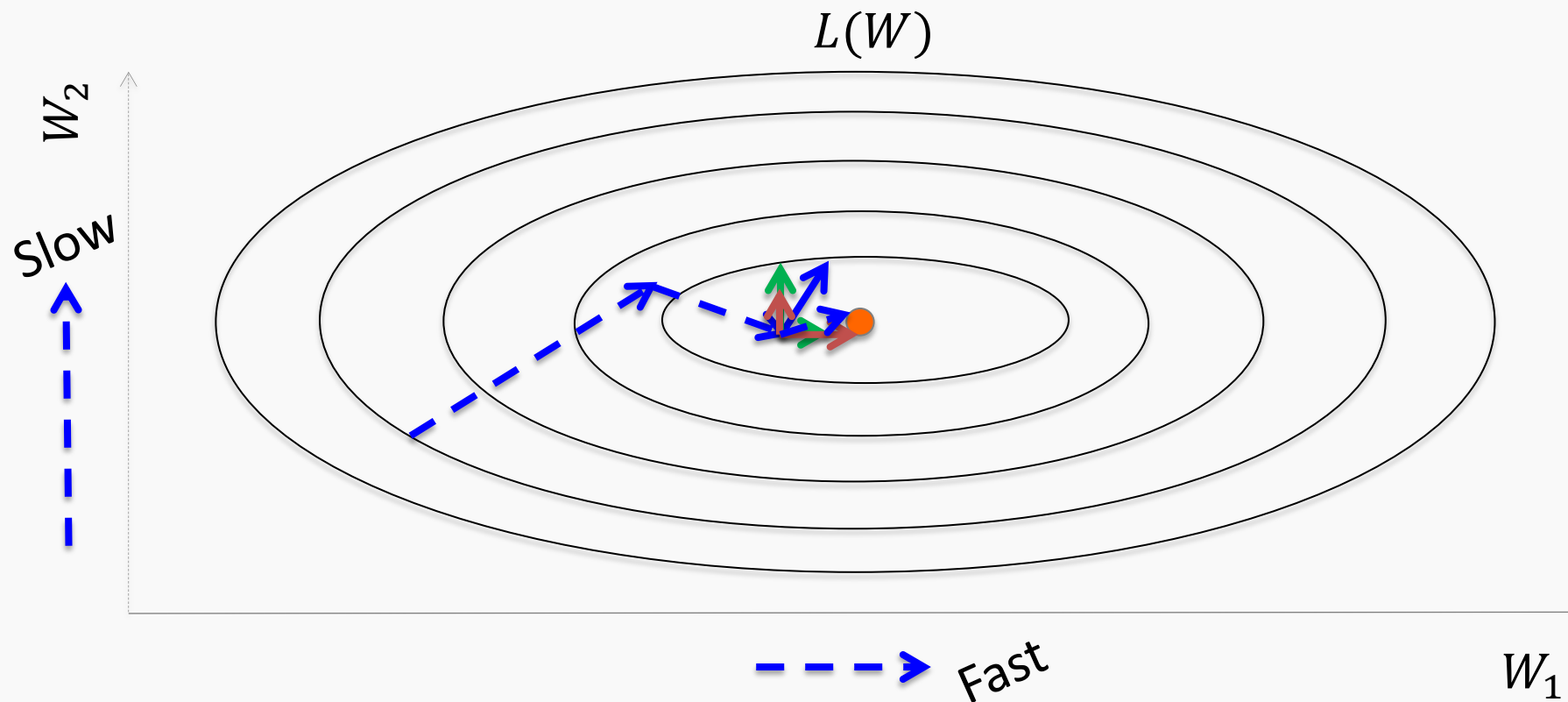


Oscillations along vertical direction

- Learning must be slower along parameter W_2

Use a different learning rate for each parameter?

Adaptive Learning Rates

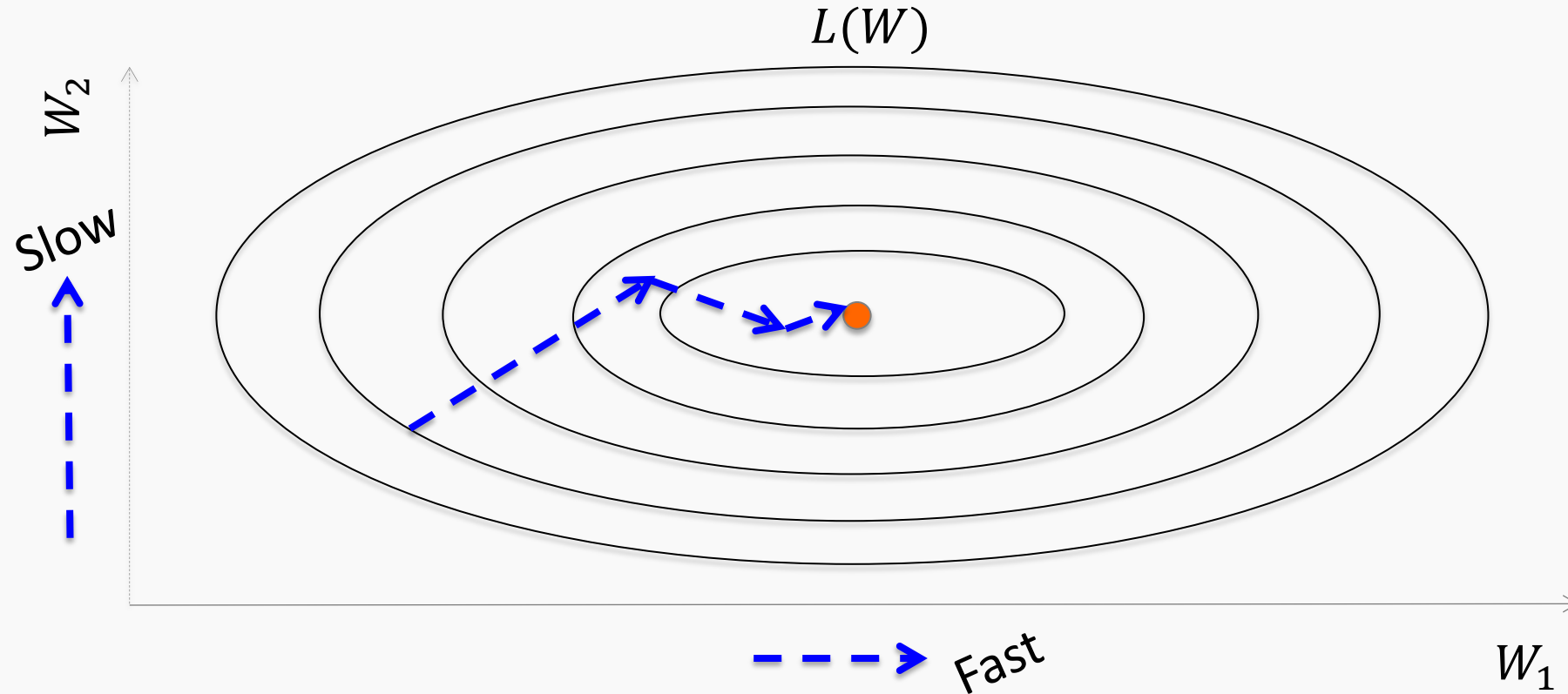


Oscillations along vertical direction

- Learning must be slower along parameter W_2

Use a different learning rate for each parameter?

Adaptive Learning Rates



With different learning rates we can control the oscillations.

AdaGrad

Old gradient descent:

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

AdaGrad

Old gradient descent:

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

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

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

AdaGrad

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

Old gradient descent:

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

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

$$W_i^* = W_i - \eta_i g_i$$

$$\eta_i \propto \frac{1}{|g_i|} = \frac{\epsilon}{\delta + |g_i|}$$

AdaGrad

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

Old gradient descent:

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

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

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

New gradient descent with adaptive learning rate:

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

RMSProp

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

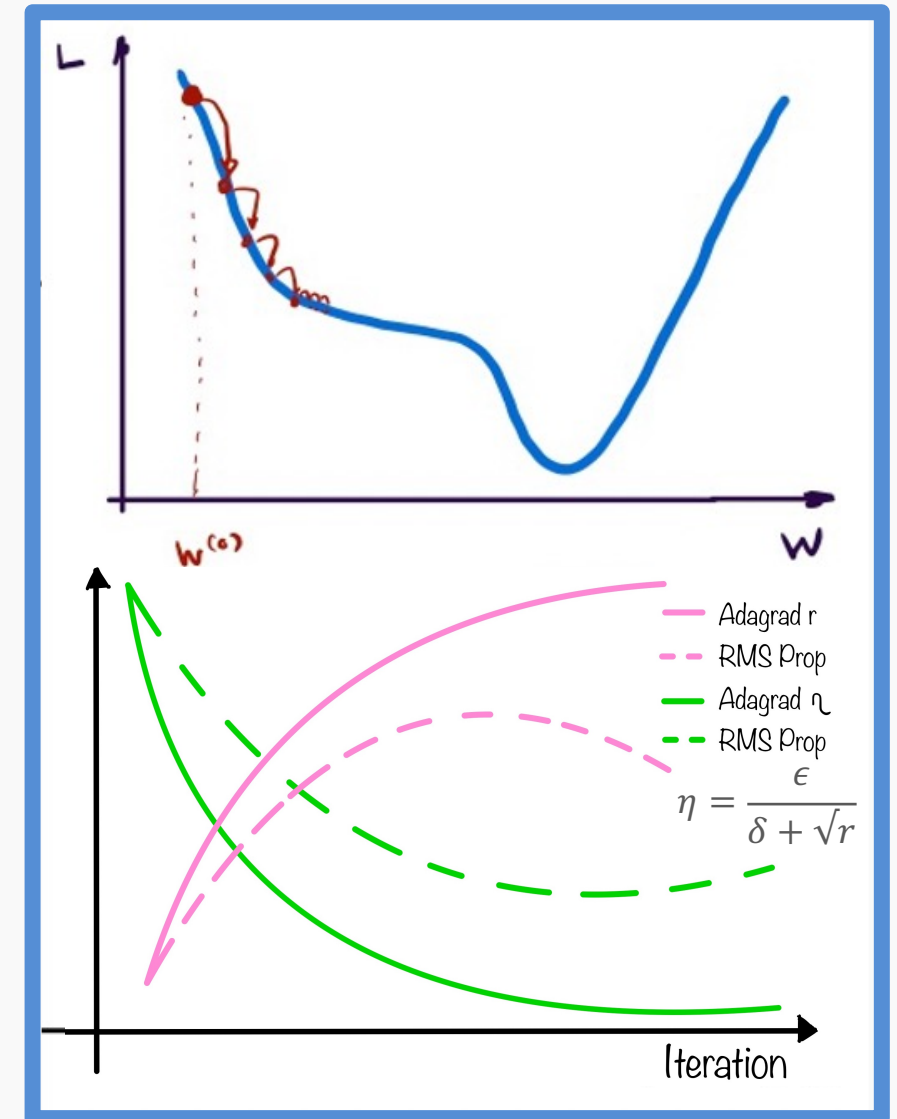
AdaGrad

$$r_i^* = r_i + g_i^2$$

RMSProp

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

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



Outline

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

Adam: RMSProp + Momentum

- Estimate **first** moment:

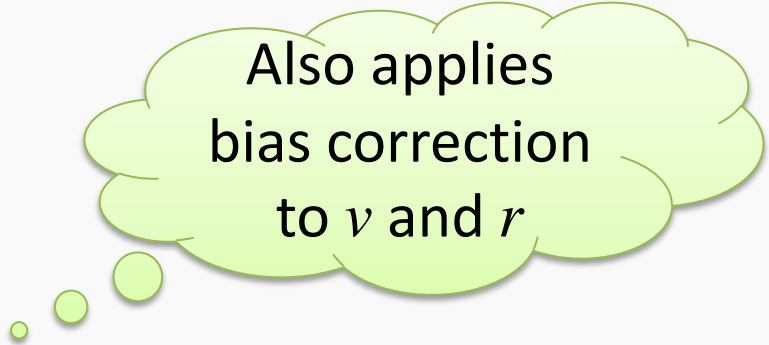
$$v_i = \rho_1 v_i + (1 - \rho_1) g_i$$

- 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 robust to hyper-
parameters

Bias Correction

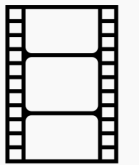
To perform bias correction on the two running average variables, we use the following equations. We do this before we update weights.

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

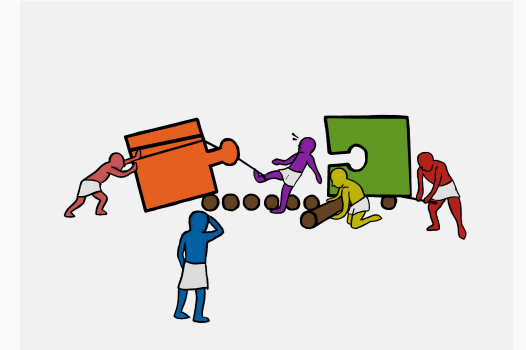
$$r_{corr} = \frac{r}{1 - \rho_2^t}$$

Where t is the number of the current iteration.

1st and 2nd moment gradient estimates are started off with both estimates being zero. Hence those initial values for which the true value is not zero, would bias the results.



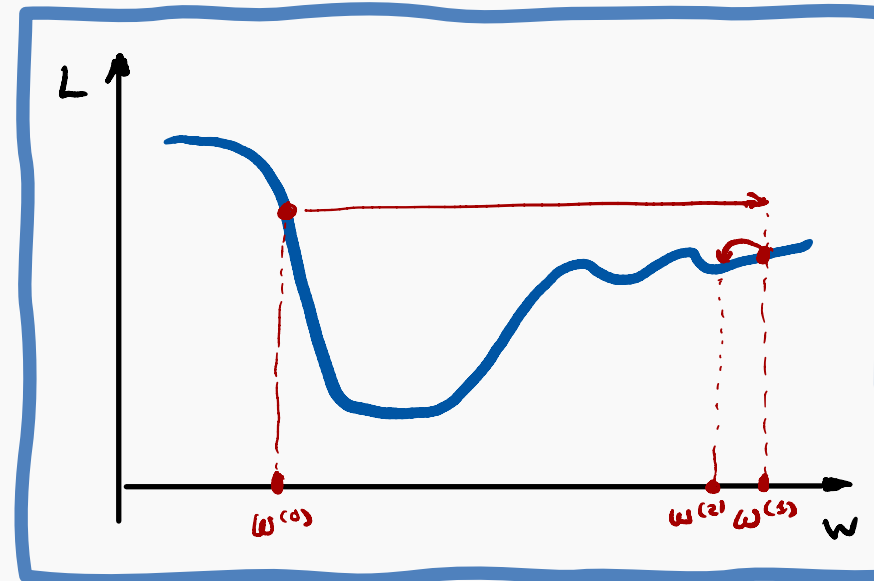
Exercise: Clipping



The aim of this exercise is to understand gradient clipping and learning rate decay.

- Implement a function to clip exploding gradients
- Experiment with different learning rates, clipping threshold

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



Exercise: RMS Prop vs Learning rate decay

The aim of this exercise is to visualize various learning rate scheduling strategies

- Make a choice for the learning rate, decay rate and starting point for the weight
- Based on the choices, visualize the loss landscape to see how quickly each strategy converges to the local minima
- Change the parameters to see if the updates are consistent with the equations for the various strategies

