Optimizers

Pavlos Protopapas

Brute Force

Greedy Search

Non-Convex optimization using Gradient Descent

Outline

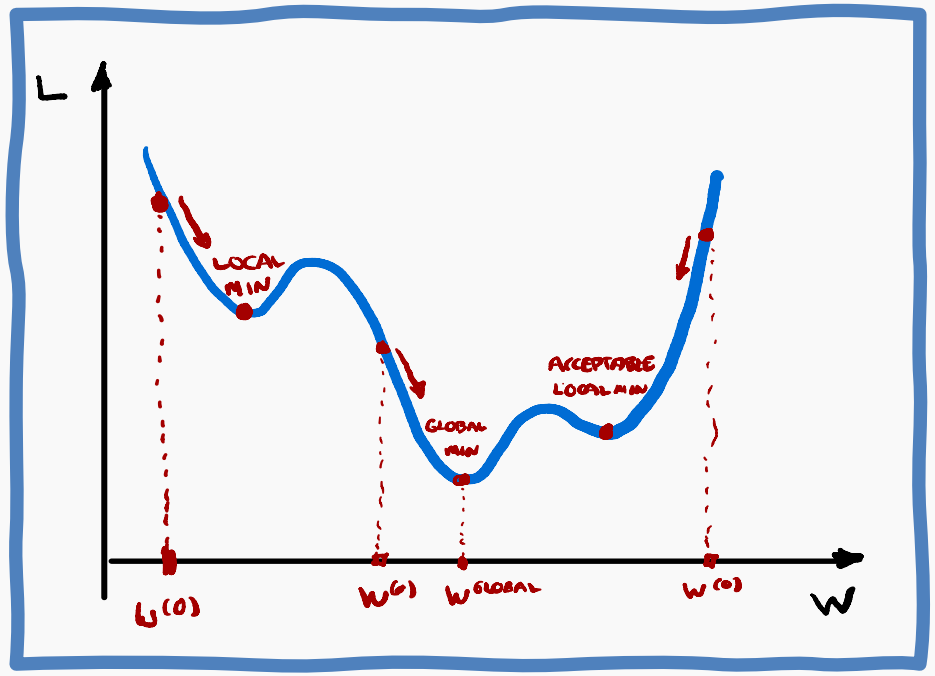
- Challenges in Optimization
- Momentum
- Adaptive Learning Rate

Outline

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

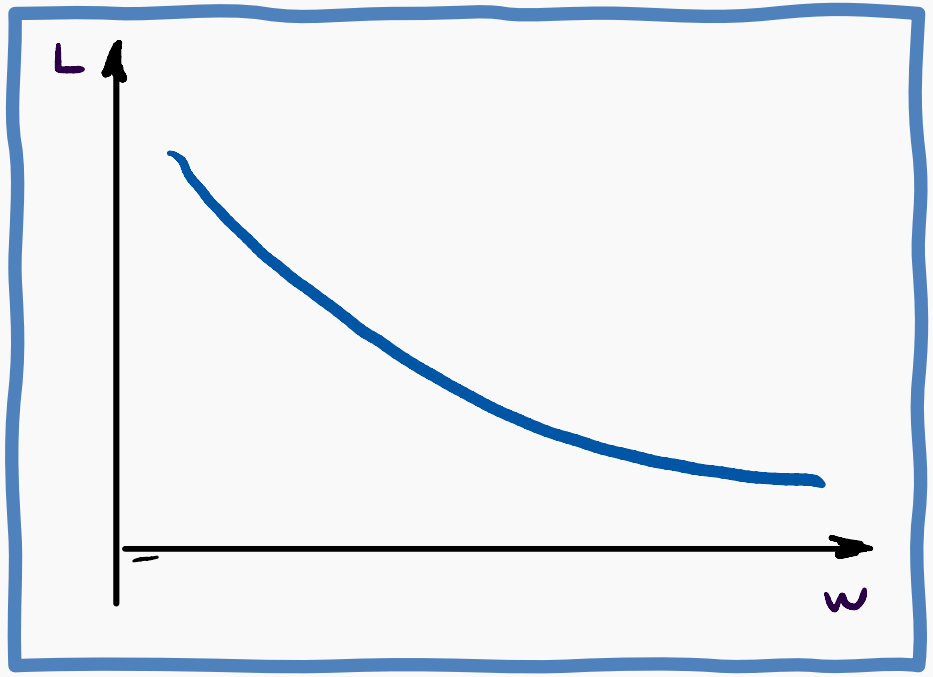
Challenges in Optimization

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.

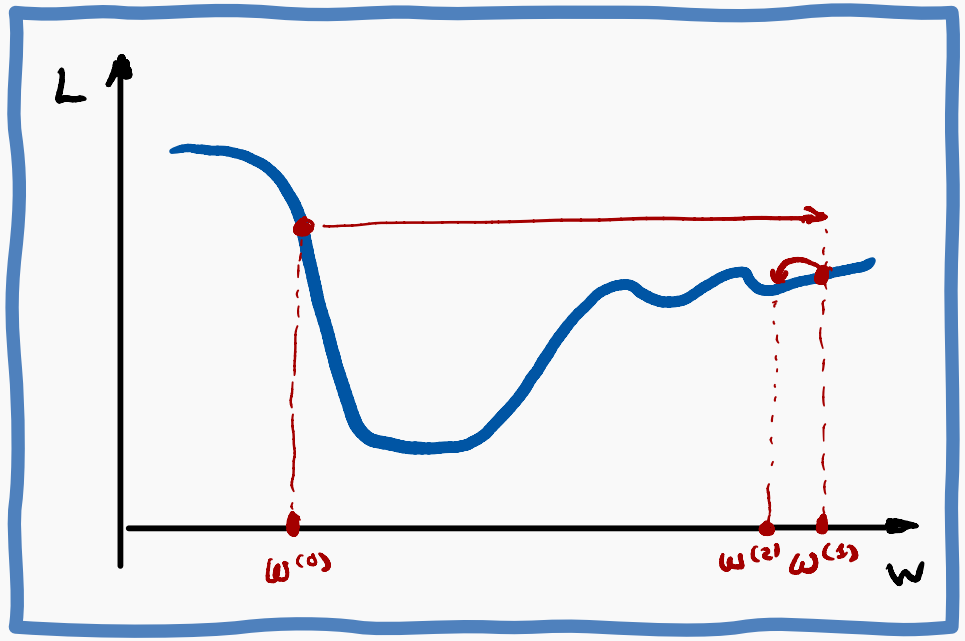
No critical points



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 Gradients

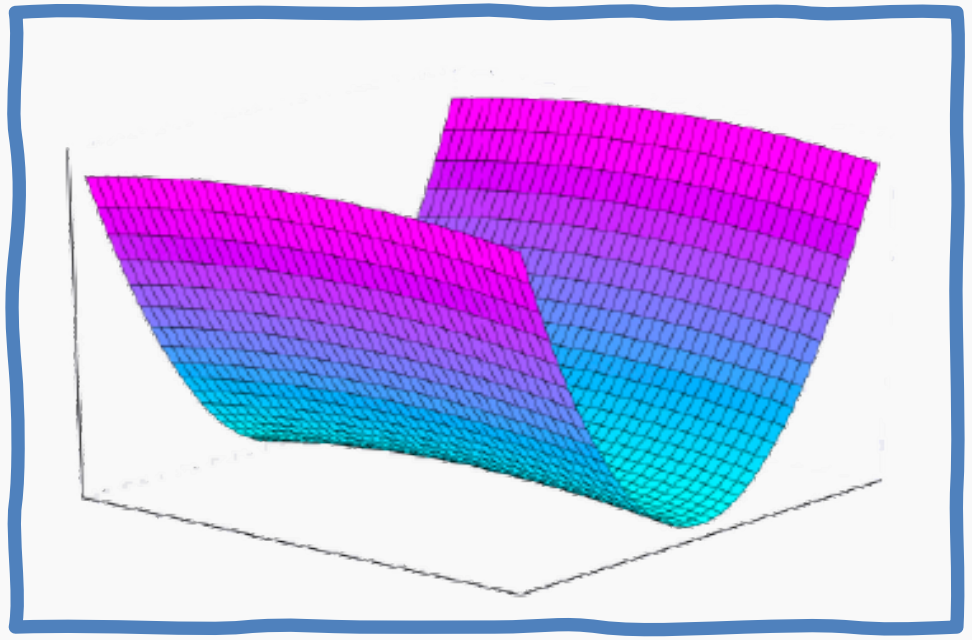


Exploding gradients due to cliffs. Can be mitigated using gradient clipping:

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

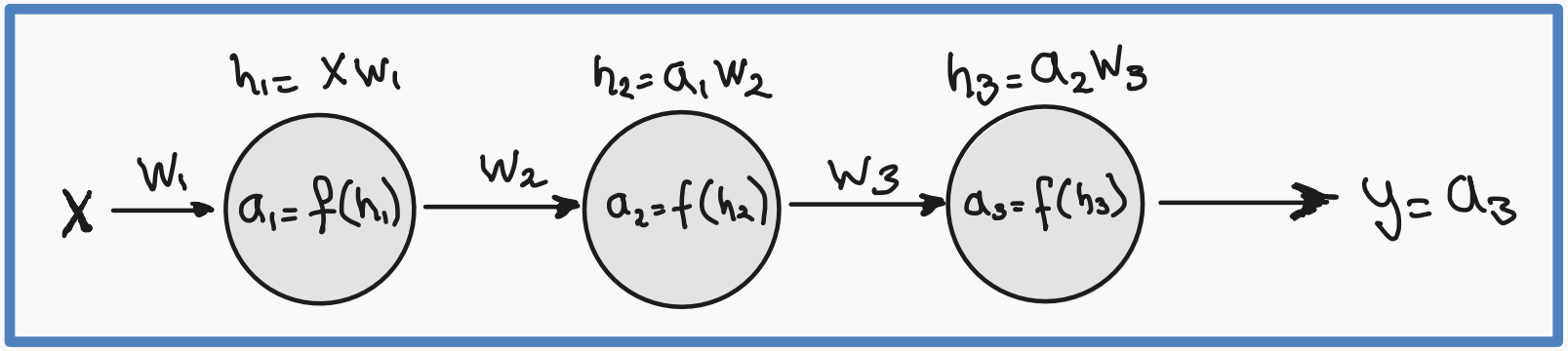
where u is user defined threshold.

Poor Conditioning



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

Challenges in Optimization: Vanishing Gradients



$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial a_2} \frac{\partial a_2}{\partial h_2} \frac{\partial h_2}{\partial w_2}$$

$\frac{\partial y}{\partial h_3} \rightarrow f'()$, $\frac{\partial h_3}{\partial a_2} \rightarrow w_3$, $\frac{\partial a_2}{\partial h_2} \rightarrow f'()$, $\frac{\partial h_2}{\partial w_2} \rightarrow a_1$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial y} f'() w_3 f'() . a_1$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial a_2} \frac{\partial a_2}{\partial h_2} \frac{\partial h_2}{\partial a_1} \frac{\partial a_1}{\partial h_1} \frac{\partial h_1}{\partial w_1}$$

$\frac{\partial y}{\partial h_3} \rightarrow f'()$, $\frac{\partial h_3}{\partial a_2} \rightarrow w_3$, $\frac{\partial a_2}{\partial h_2} \rightarrow f'()$, $\frac{\partial h_2}{\partial a_1} \rightarrow w_2$, $\frac{\partial a_1}{\partial h_1} \rightarrow f'()$, $\frac{\partial h_1}{\partial w_1} \rightarrow X$

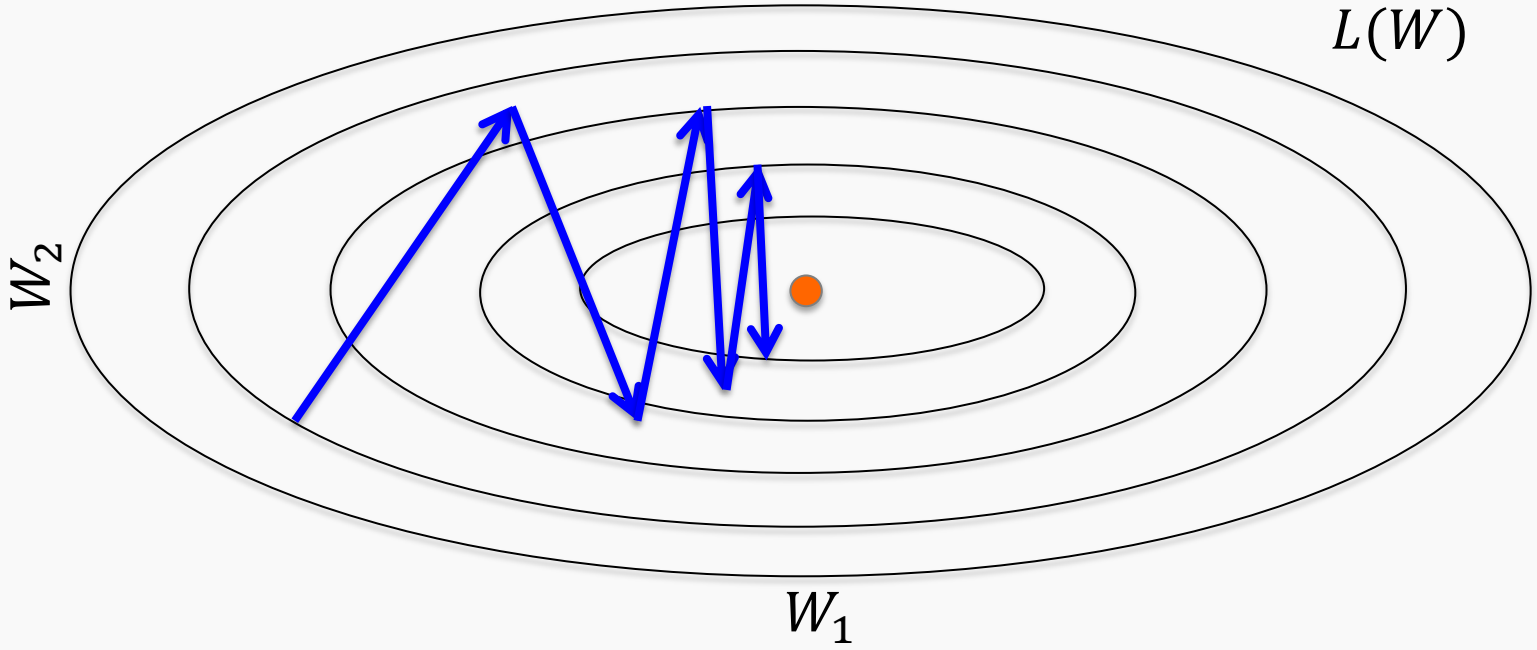
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} f'() w_3 f'() w_2 f'() X$$

Outline

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

Momentum

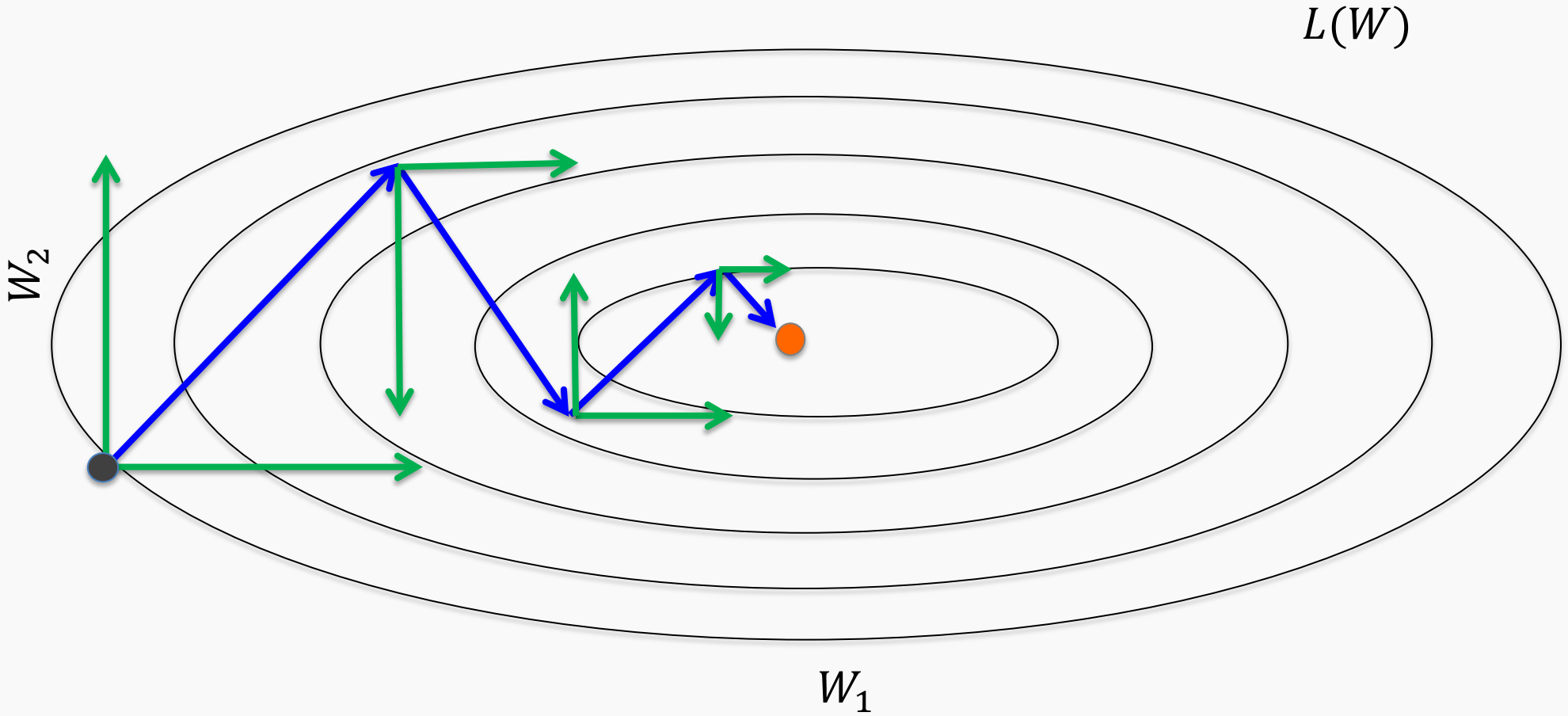
Simple Gradient Descent oscillates because updates do not exploit curvature information



Momentum

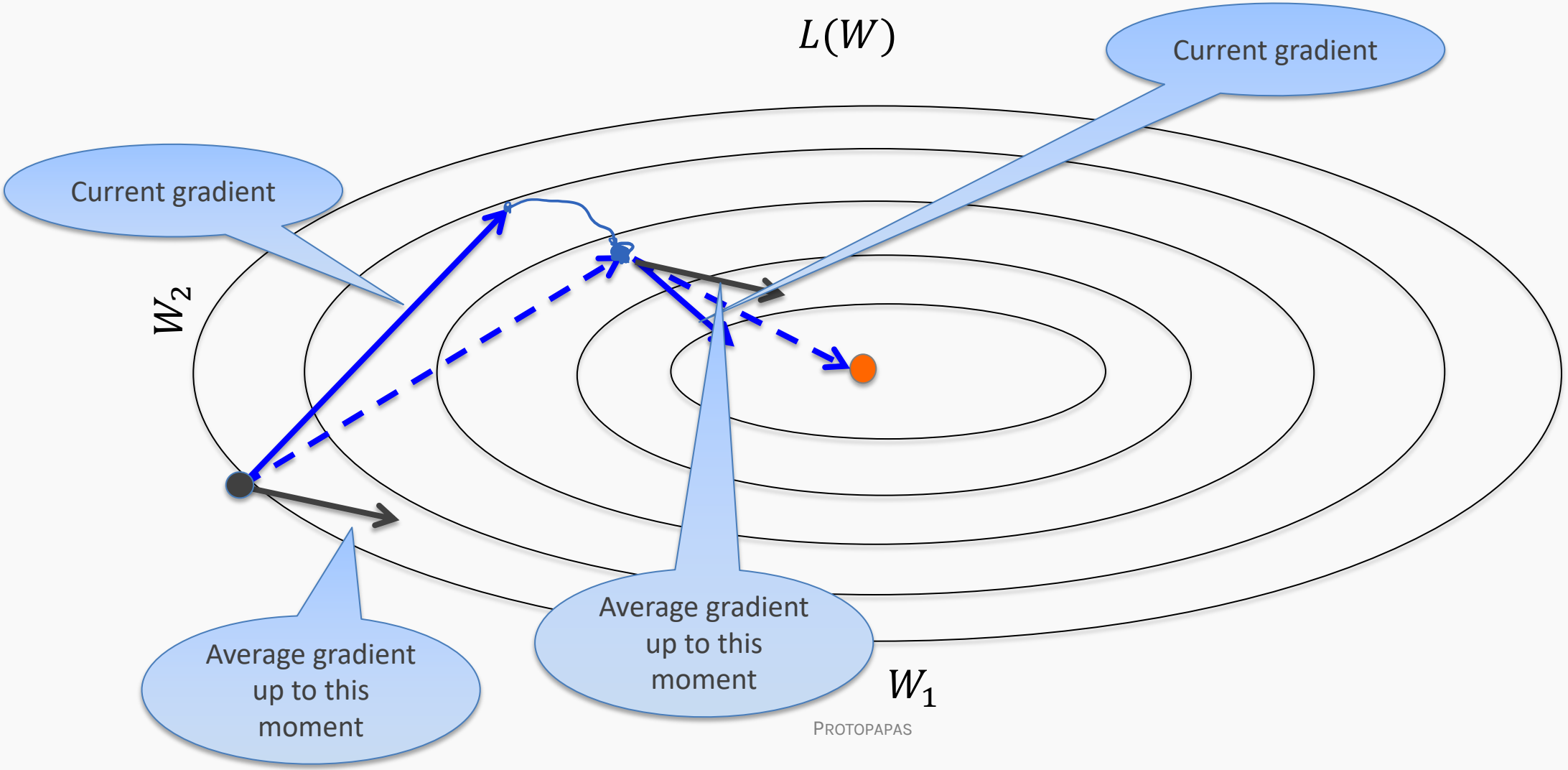
Let us figure out an algorithm which will lead us to the minimum faster.

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



Momentum

Add the average of the gradient from before



Momentum

f is the Neural Network

Old gradient descent:

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

Momentum (Nesterov)

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:

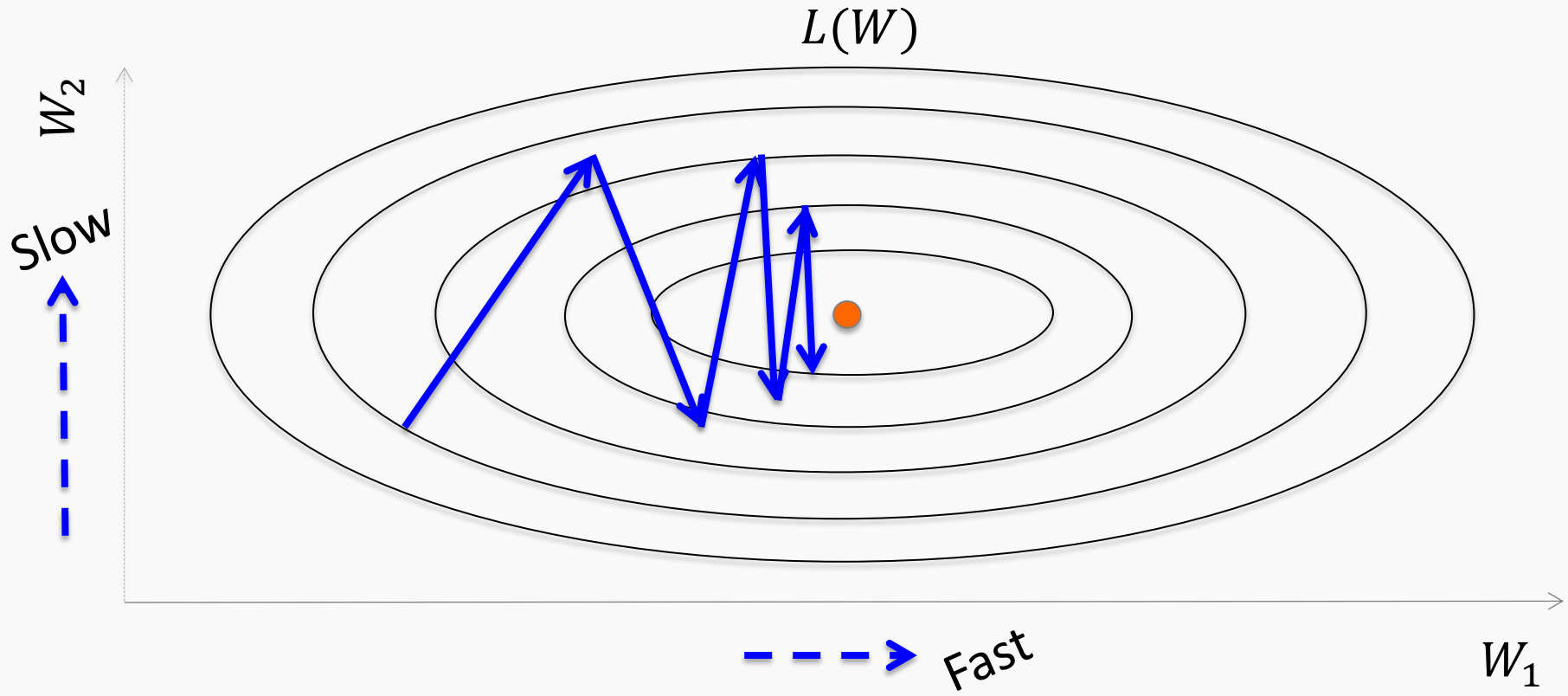
$$v = \alpha v + (1-\alpha)\, g \qquad W^* = W - \eta v$$

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

Outline

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

Adaptive Learning Rates

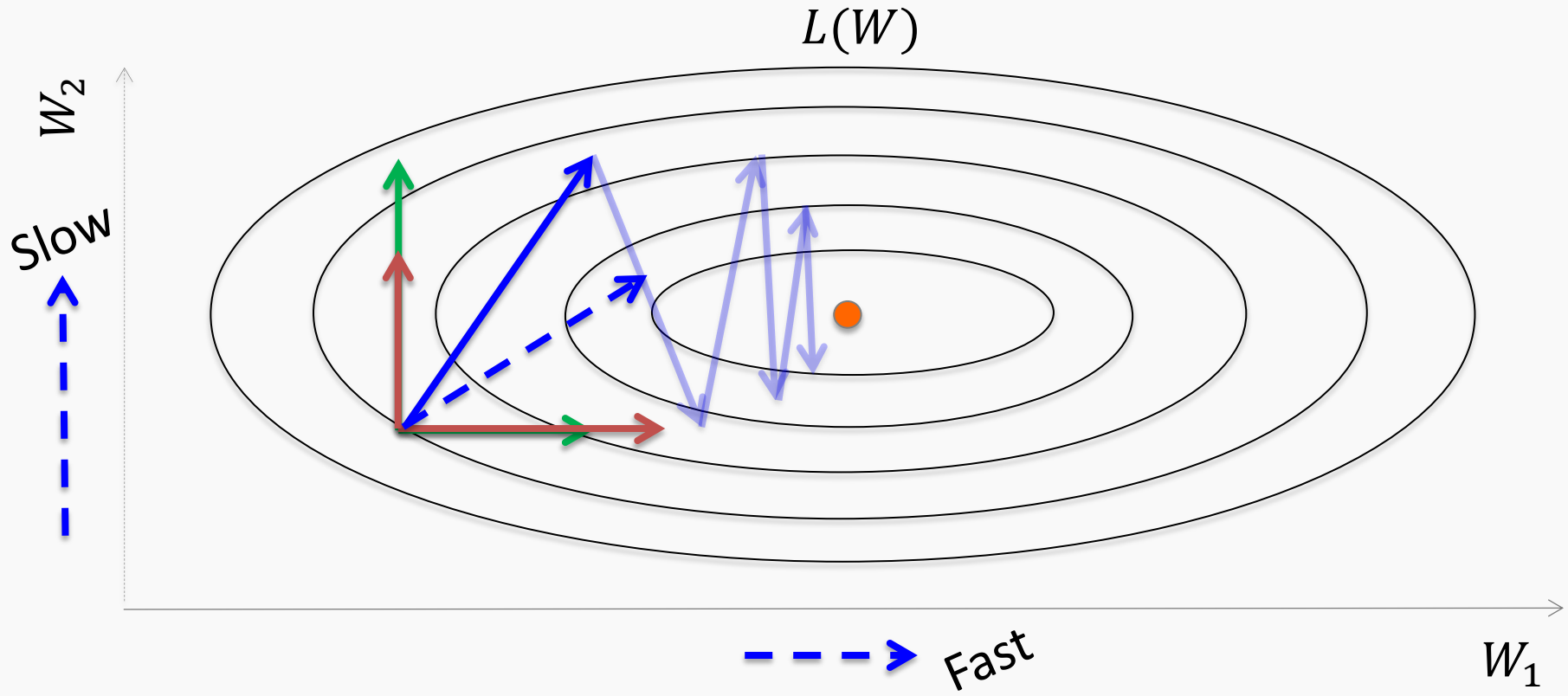


Oscillations along vertical direction

– Learning must be slower along parameter W_2

Use a different learning rate for each parameter?

Adaptive Learning Rates

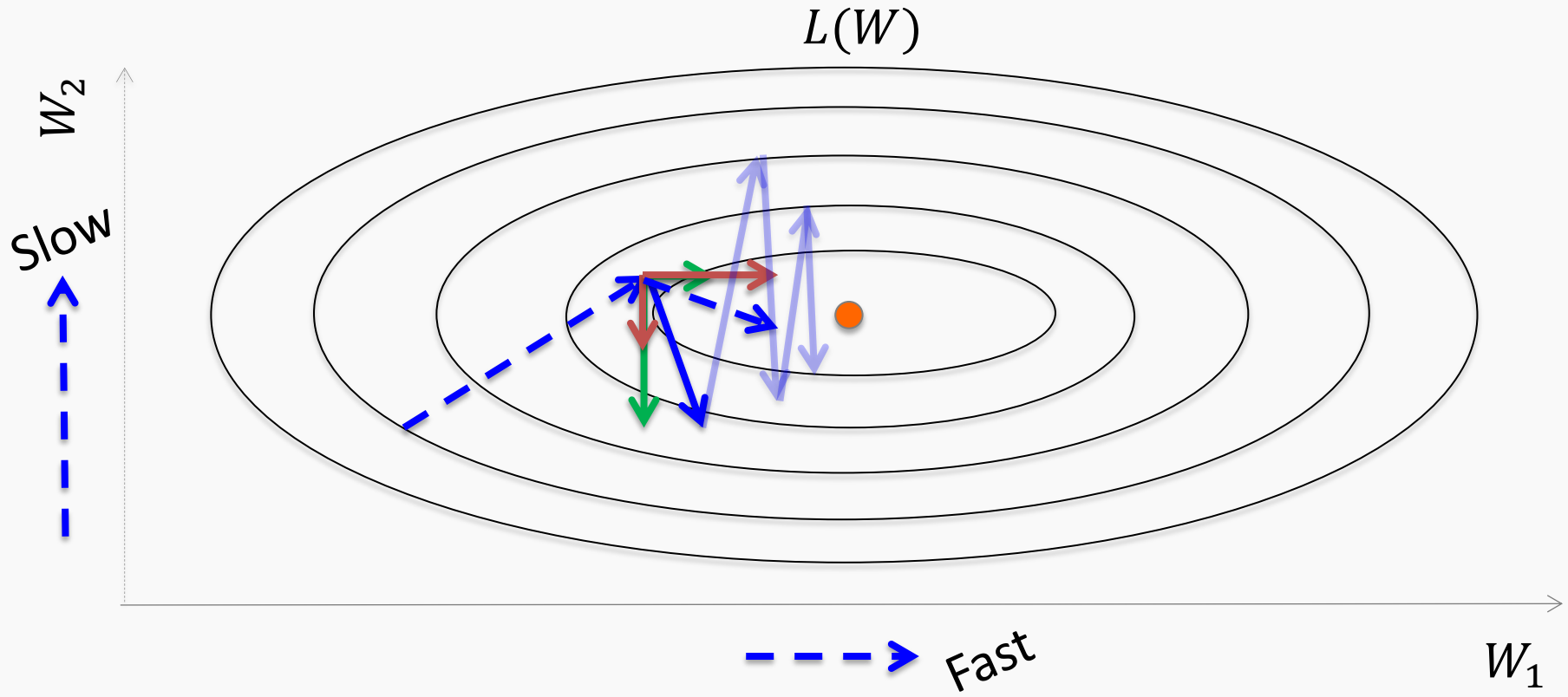


Oscillations along vertical direction

– Learning must be slower along parameter W_2

Use a different learning rate for each parameter?

Adaptive Learning Rates

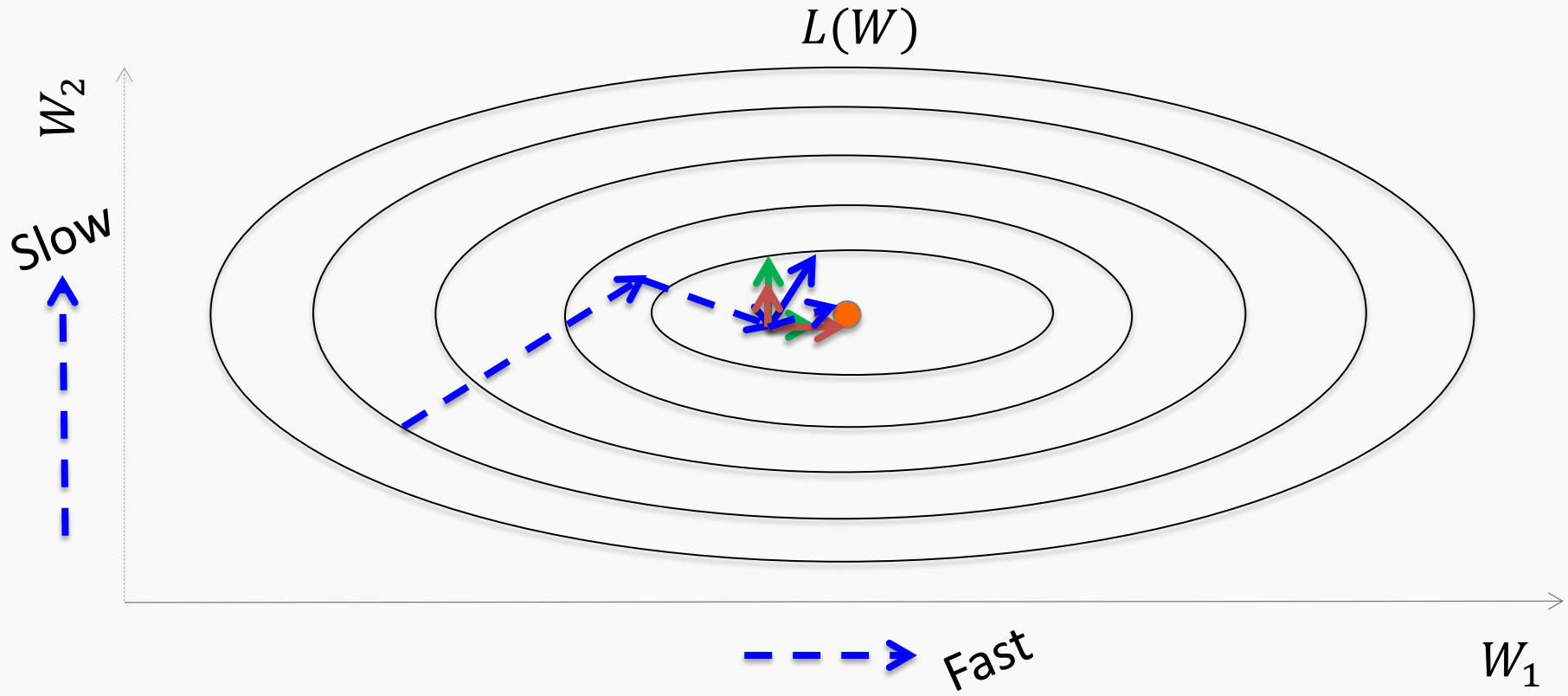


Oscillations along vertical direction

– Learning must be slower along parameter W_2

Use a different learning rate for each parameter?

Adaptive Learning Rates

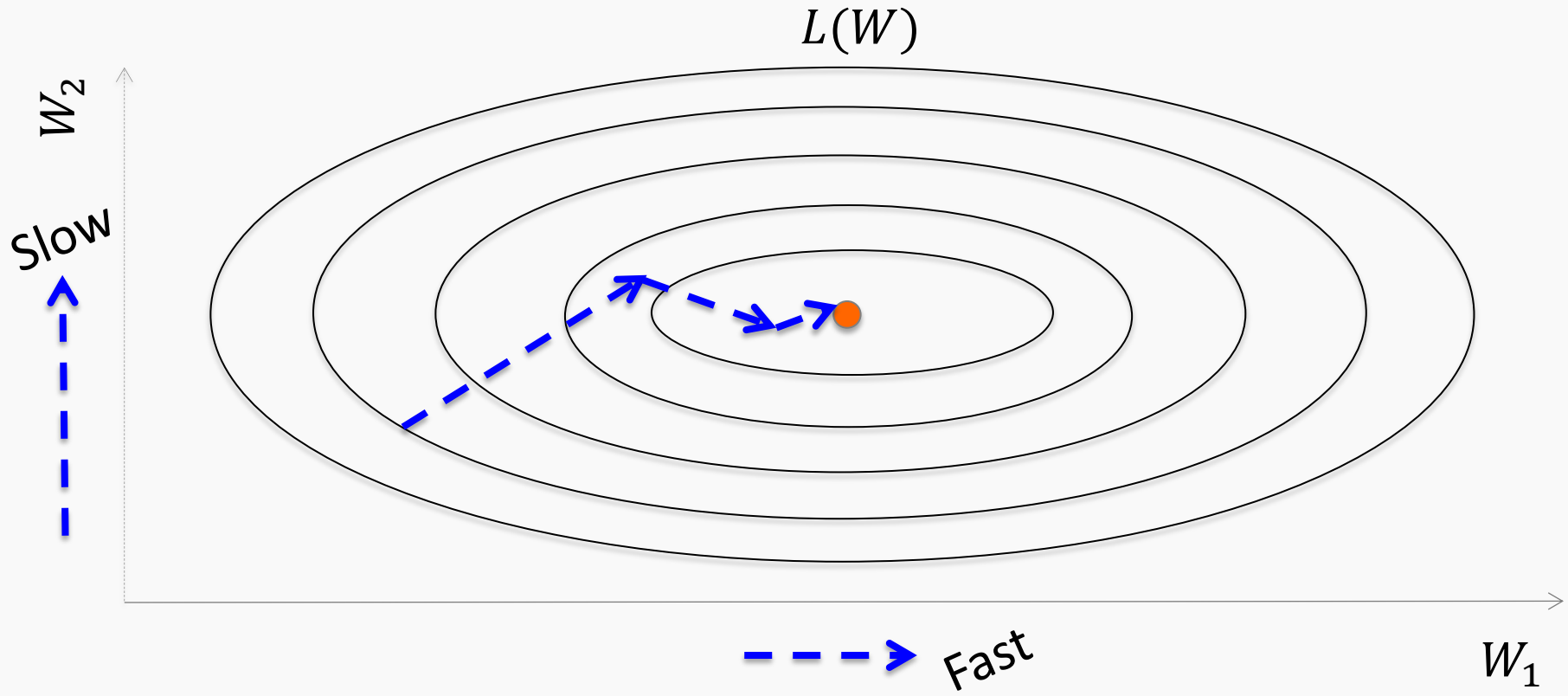


Oscillations along vertical direction

– 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:

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

AdaGrad

Old gradient descent:

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

We would like $\eta's$ 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:

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

We would like $\eta's$ 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:

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

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

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

New gradient descent with adaptive learning rate:

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

RMSProp

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

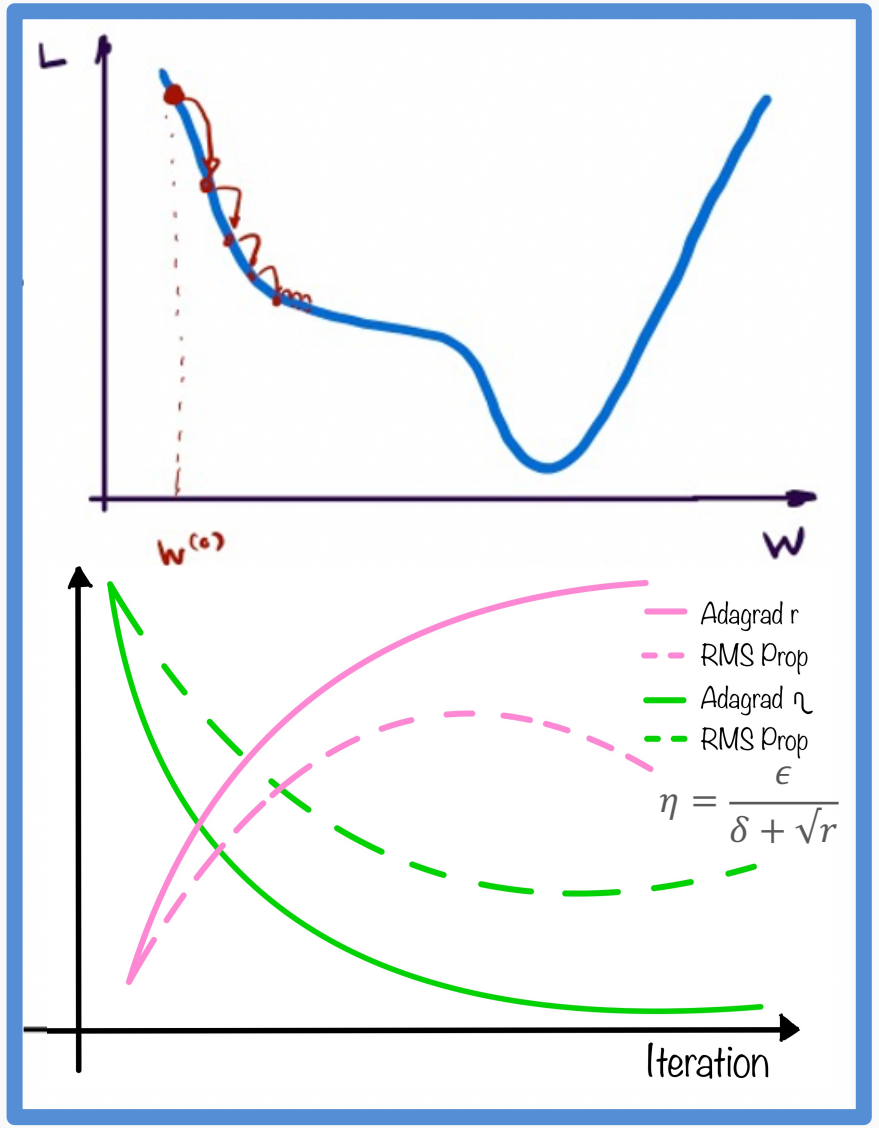
AdaGrad

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

RMSProp

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

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



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.

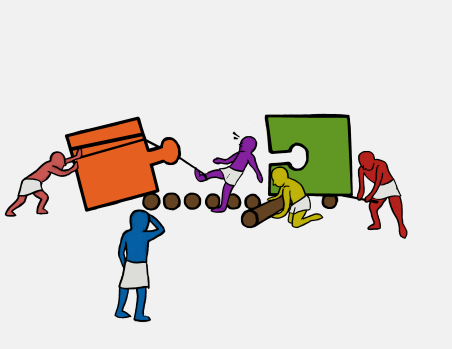
$$v_{corr} = \frac{v}{1 - \rho_1^t}$$

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

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.

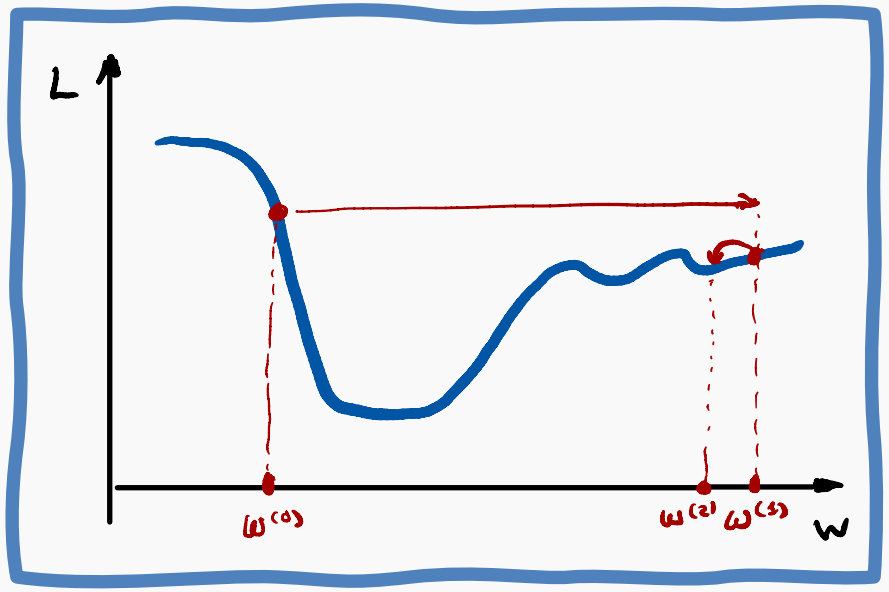
Exercise: Clipping



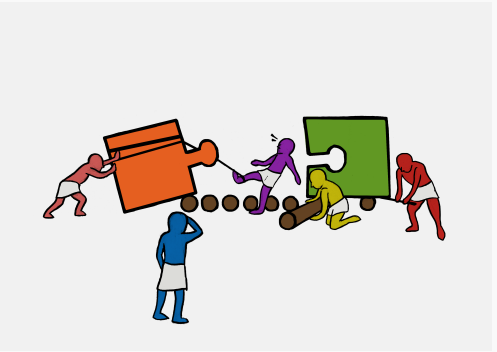
The aim of this exercise is to understand gradient clipping and learning rate decay.

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

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

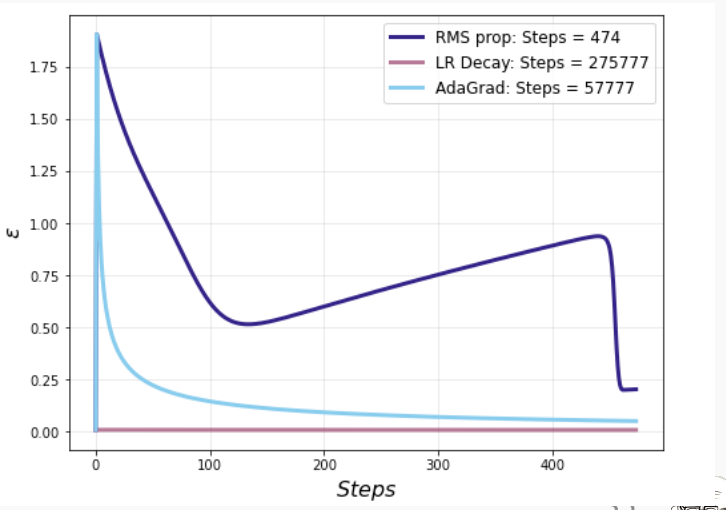


Exercise: RMS Prop vs Learning rate decay

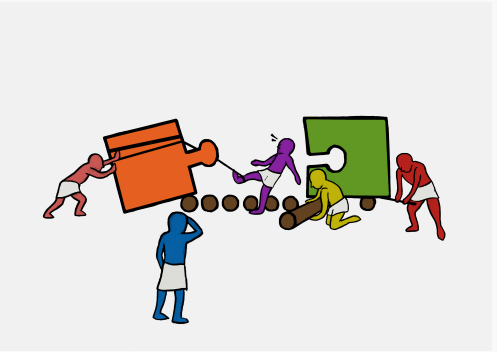


The aim of this exercise is to visualize various learning rate scheduling strategies

- 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



Exercise: Adam Optimizer



The aim of this exercise is to understand gradient descent with Adam optimizer

- Write a function to implement gradient descent with Adam
- Make sure to account for bias correction for 1st and 2nd moment gradient
- Use the helper function to visualize how the descent works and compare the speed of convergence with vanilla gradient descent