Optimizers

CS1090B Data Science II – Spring 2025

Pavlos Protopapas, Natesh Pillai, and Chris Gumb

Sanil Edwin


Brute Force
Greedy Search
Non-Convex optimization using Gradient Descent

Outline

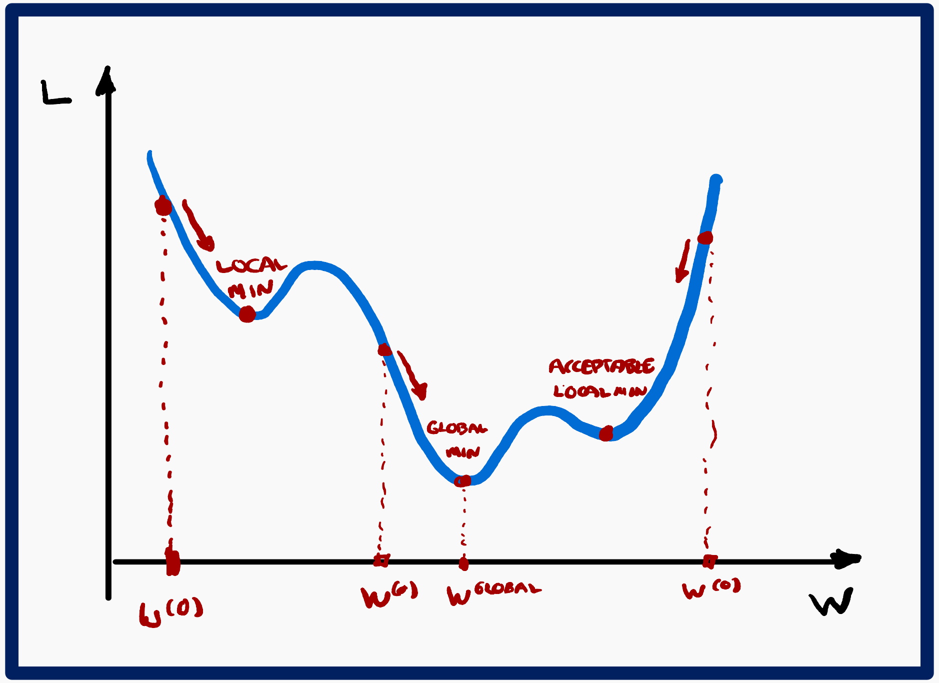
- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- Adam

Outline

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

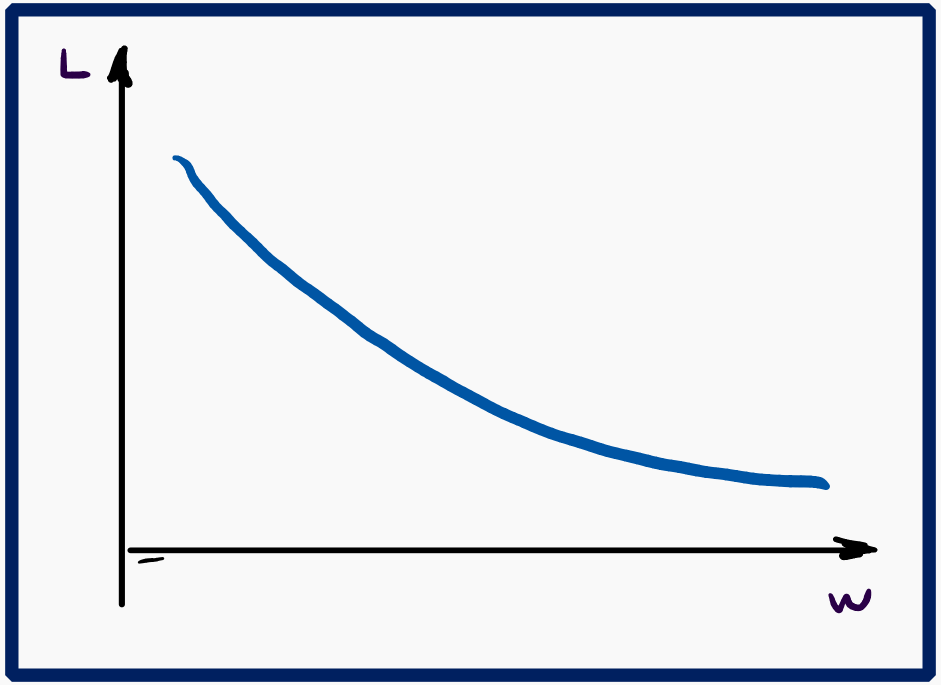
Challenges in Optimization

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.

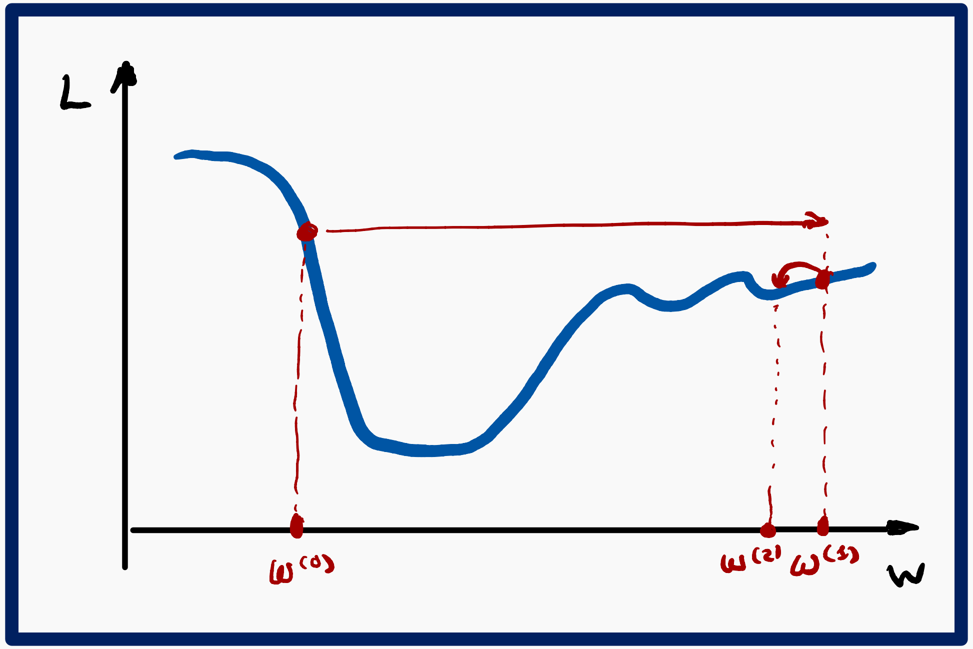
No critical points



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

Challenges in Optimization

Exploding Gradients

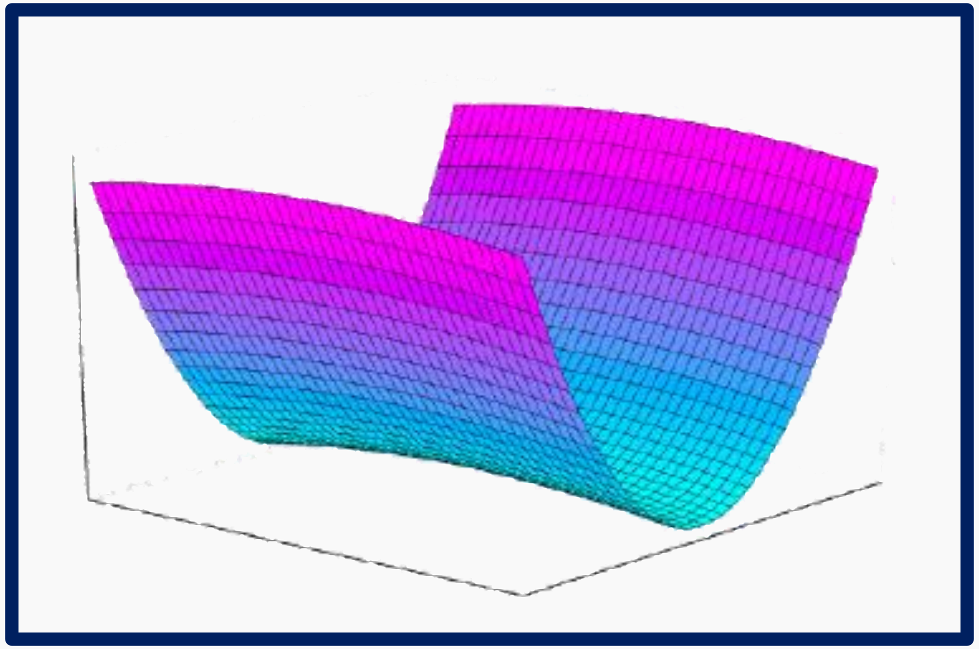


Exploding gradients due to cliffs. Can be mitigated using **gradient clipping:**

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

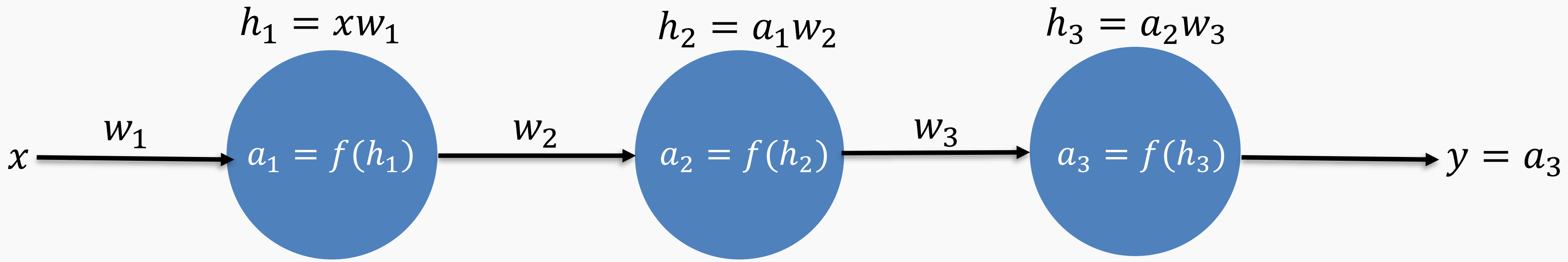
where u is user defined threshold.

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



$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial y} \underbrace{\frac{\partial y}{\partial h_3}}_{f'()} \underbrace{\frac{\partial h_3}{\partial a_2}}_{w_3} \underbrace{\frac{\partial a_2}{\partial h_2}}_{f'()} \underbrace{\frac{\partial h_2}{\partial w_2}}_{a_1}$$

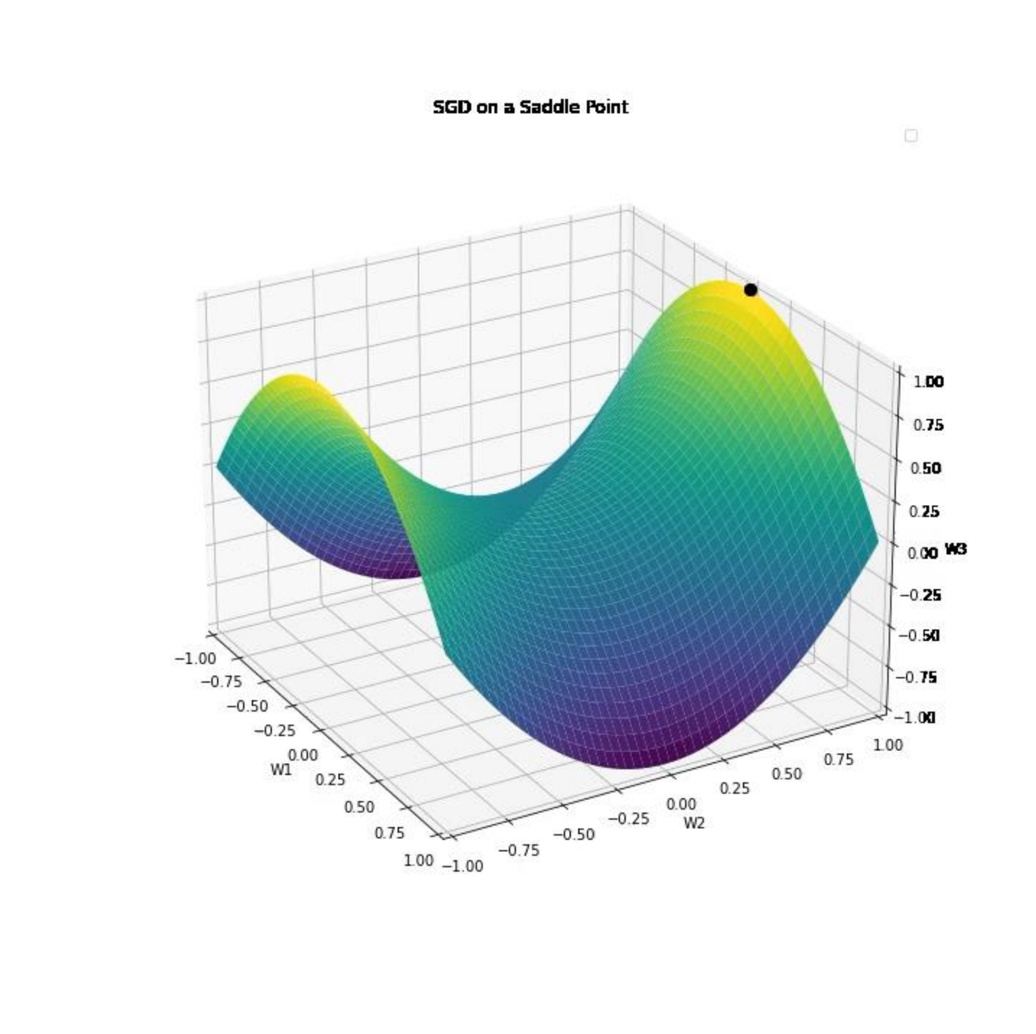
$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial y} \underbrace{\frac{\partial y}{\partial h_3}}_{f'()} \underbrace{\frac{\partial h_3}{\partial a_2}}_{w_3} \underbrace{\frac{\partial a_2}{\partial h_2}}_{f'()} \underbrace{\frac{\partial h_2}{\partial a_1}}_{w_2} \underbrace{\frac{\partial a_1}{\partial h_1}}_{f'()} \underbrace{\frac{\partial h_1}{\partial w_1}}_{x}$$

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

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

Challenges in Optimization | Saddle Points

- In large N-dimensional domains, local minima are **extremely rare**.
- Saddle points are very common in high-dimensional spaces.
- These saddle points make it notoriously hard for SGD to escape, as the gradient is close to zero in all dimensions.



Escaping saddle points

Somewhat counterintuitively, the best way to escape saddle points is to just move in any direction quickly.

Then we can get somewhere with more substantial curvature for a more informed update.

Game Time

What is the role of the backpropagation algorithm?

A. Updating the weights

B. Computing the loss function

C. Computing the gradients

D. None of the above

Game Time

What is the role of the backpropagation algorithm?

A. Updating the weights

B. Computing the loss function

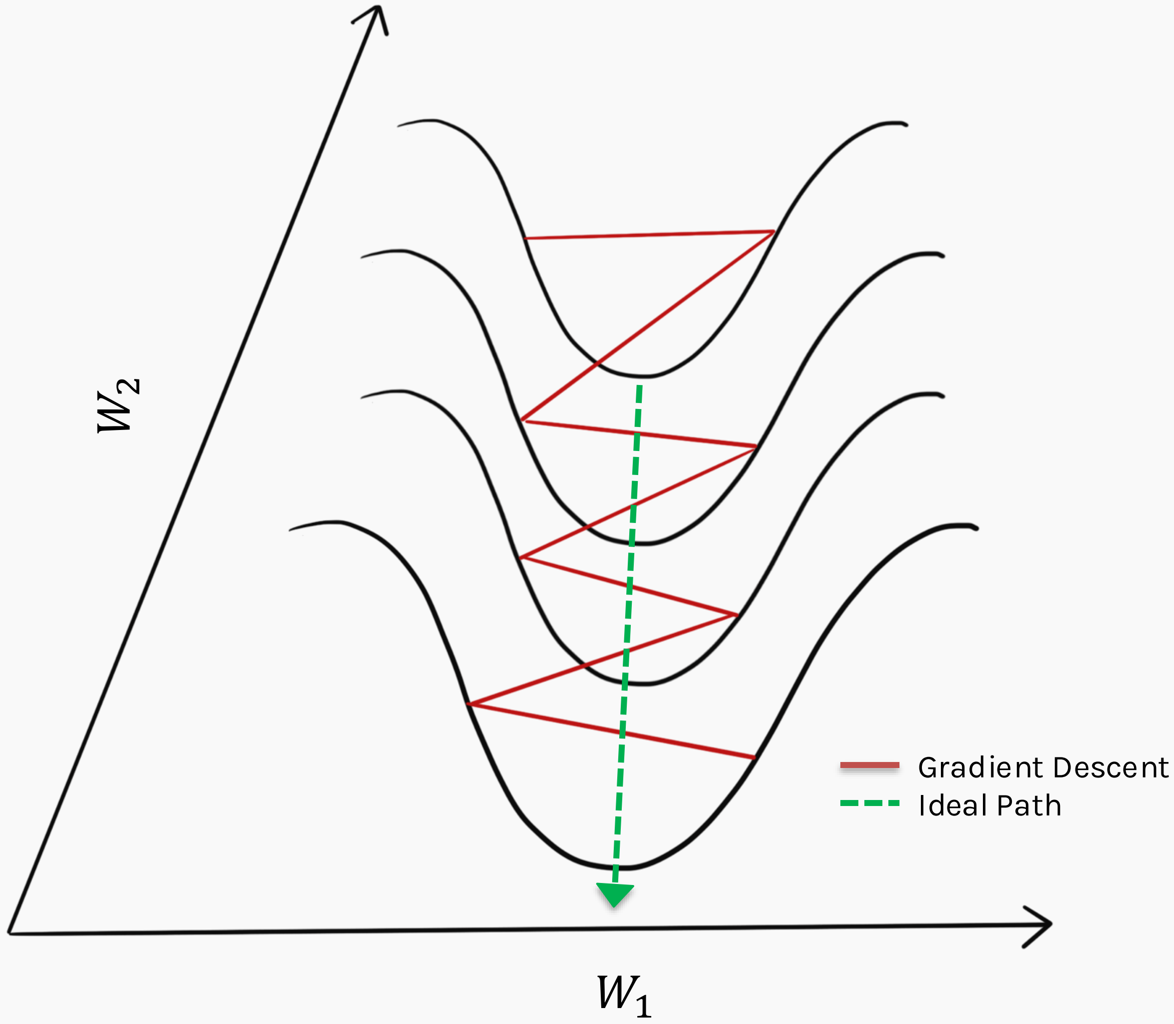
C. Computing the gradients

D. None of the above

Outline

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

Why do we need momentum?

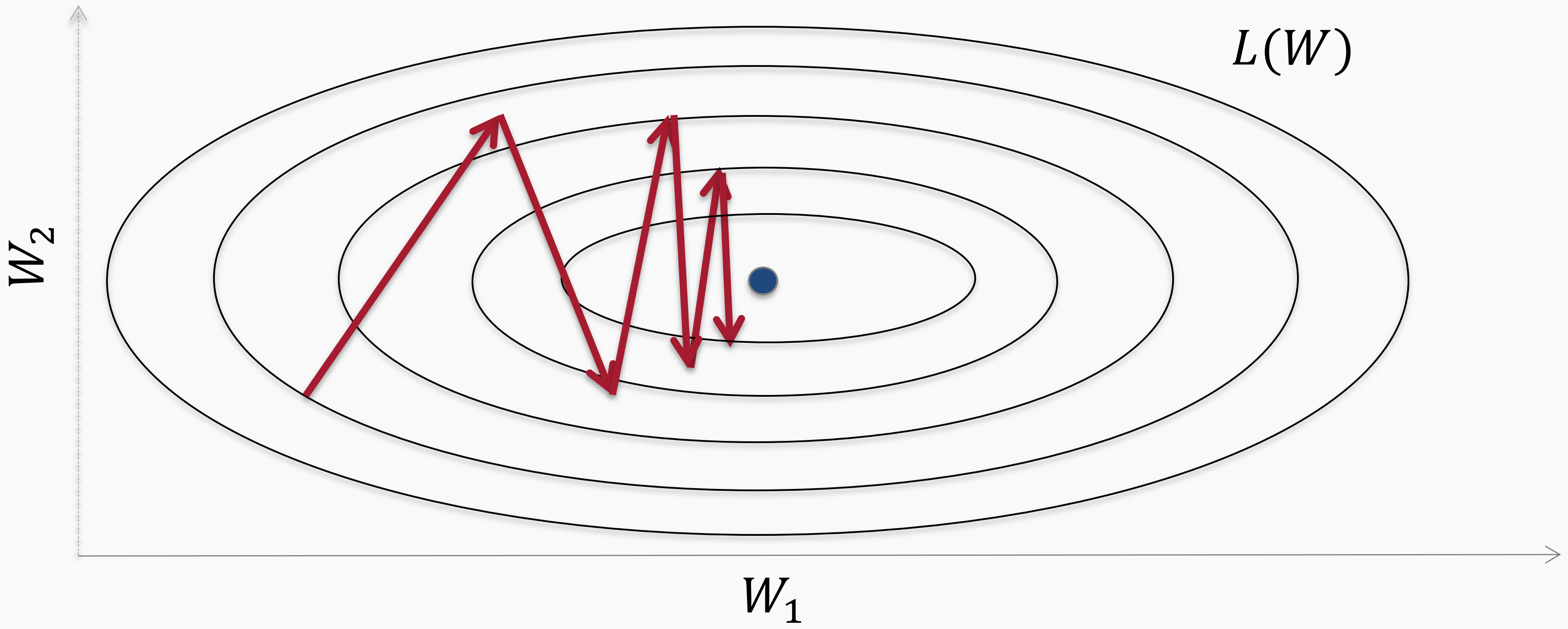


The gradients oscillating along the ridges, making the descent lot slower to the minima.

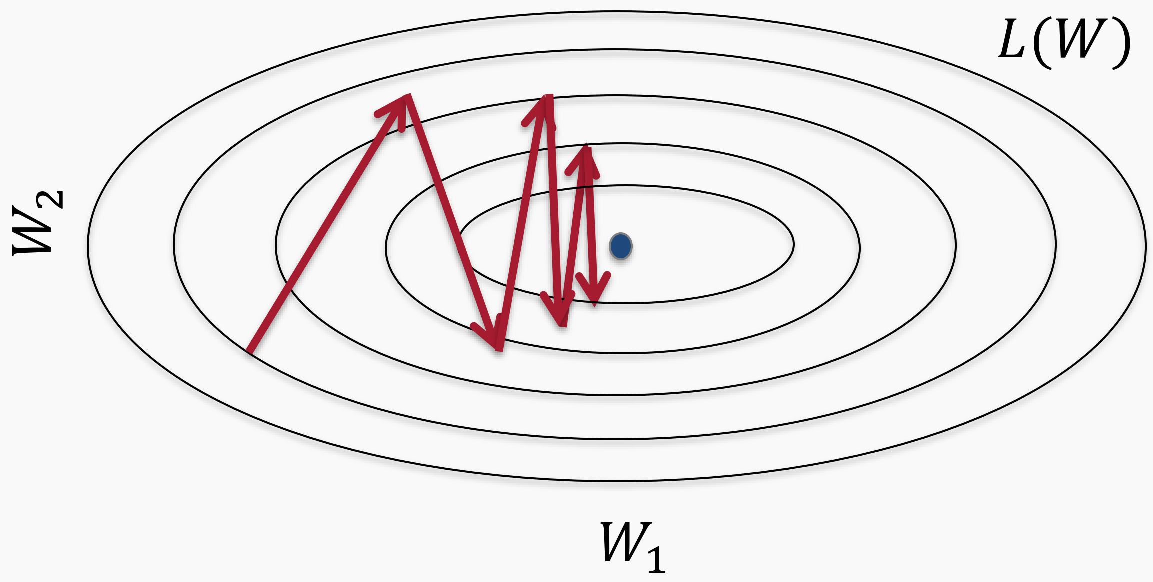
The optimization may become too slow to be practical and even appear to halt altogether, creating the false impression of a local minimum.

We need momentum to accelerate our search in the direction of minima.

Momentum



Momentum

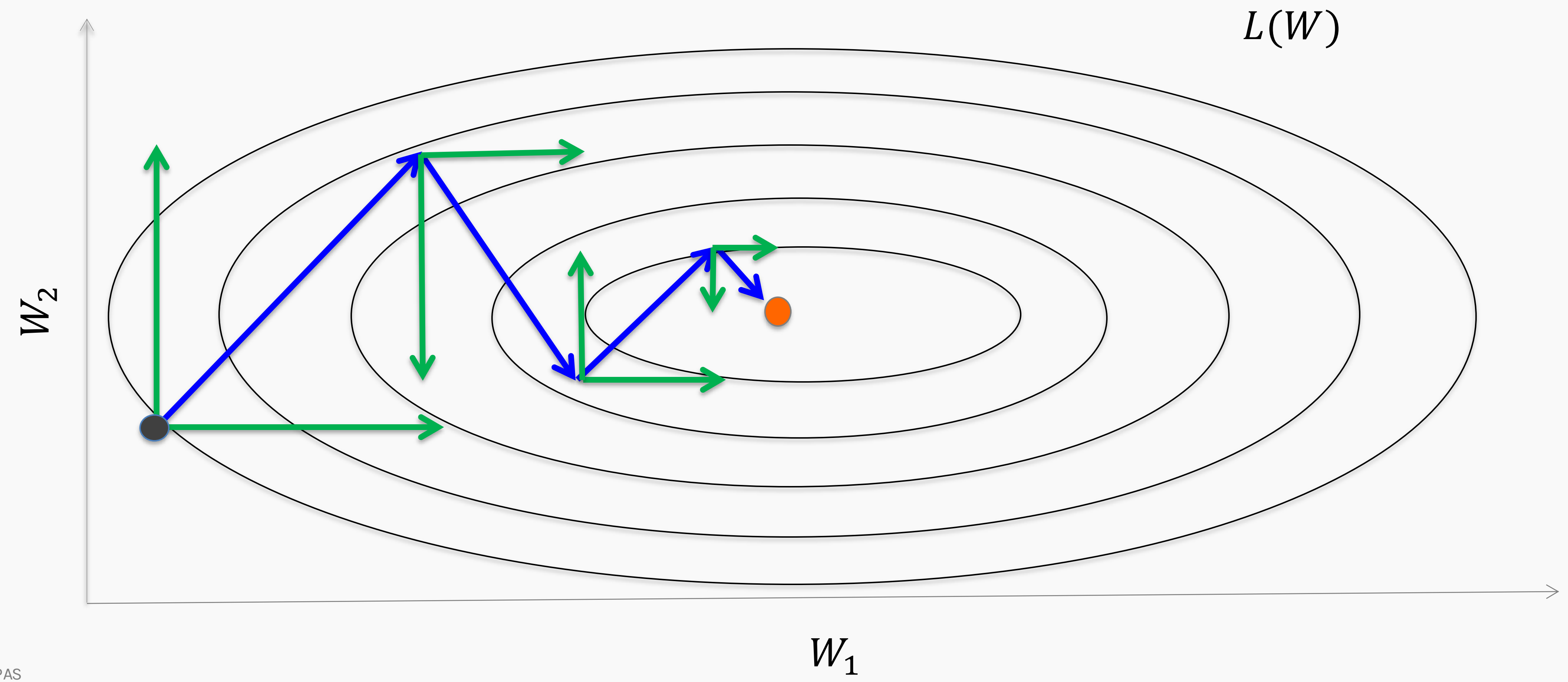


- Simple Gradient Descent oscillates because updates do not exploit curvature information
- The gradients oscillate along the ridges, making the descent to the minima a lot slower.

- The optimization may become too slow on saddle points to be practical, creating the false impression of a local minimum.
- We need ‘something’ to accelerate our search in the direction of minima.

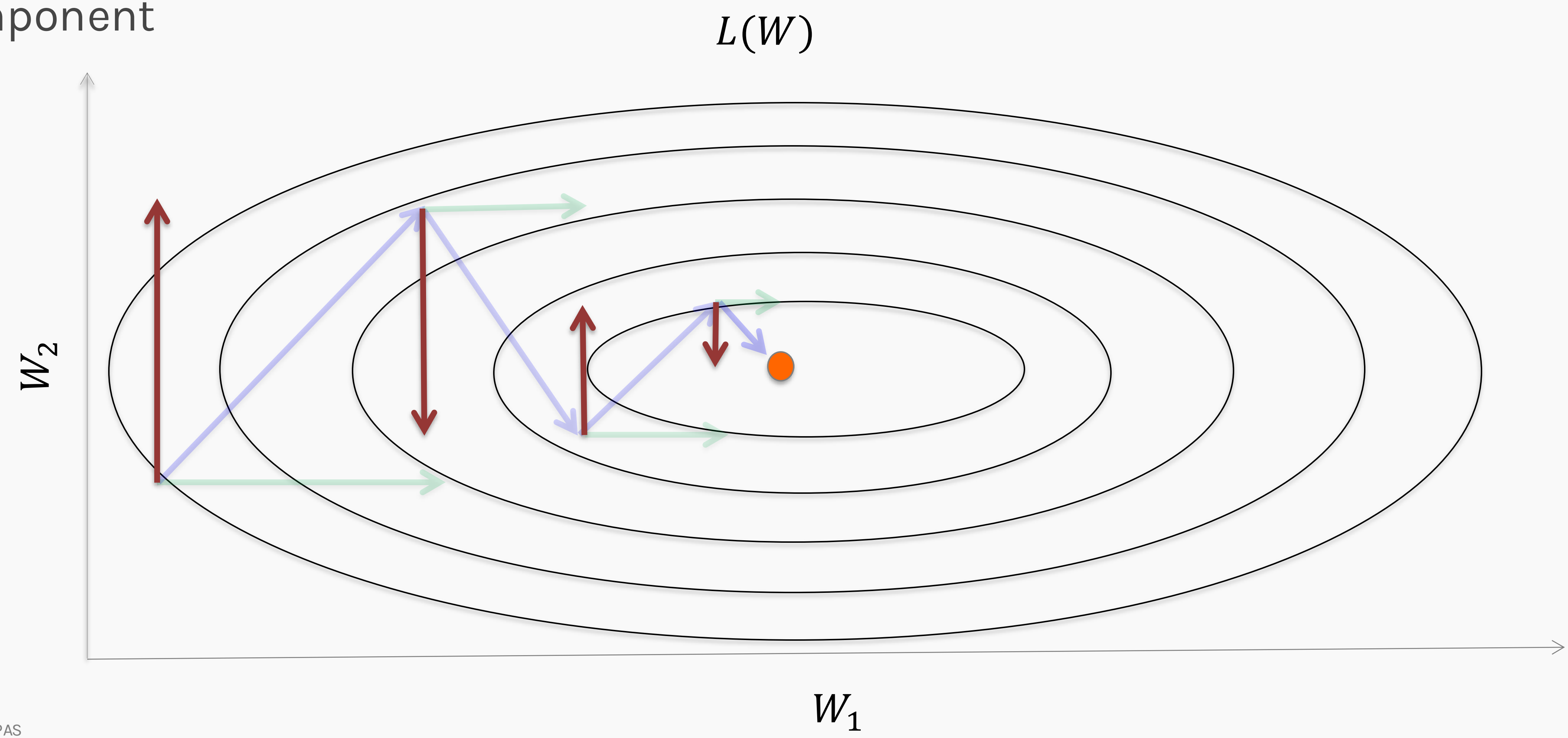
Momentum

Let us figure out an algorithm which will converge to the minimum faster and avoid saddle points. We first examine the gradient of the loss



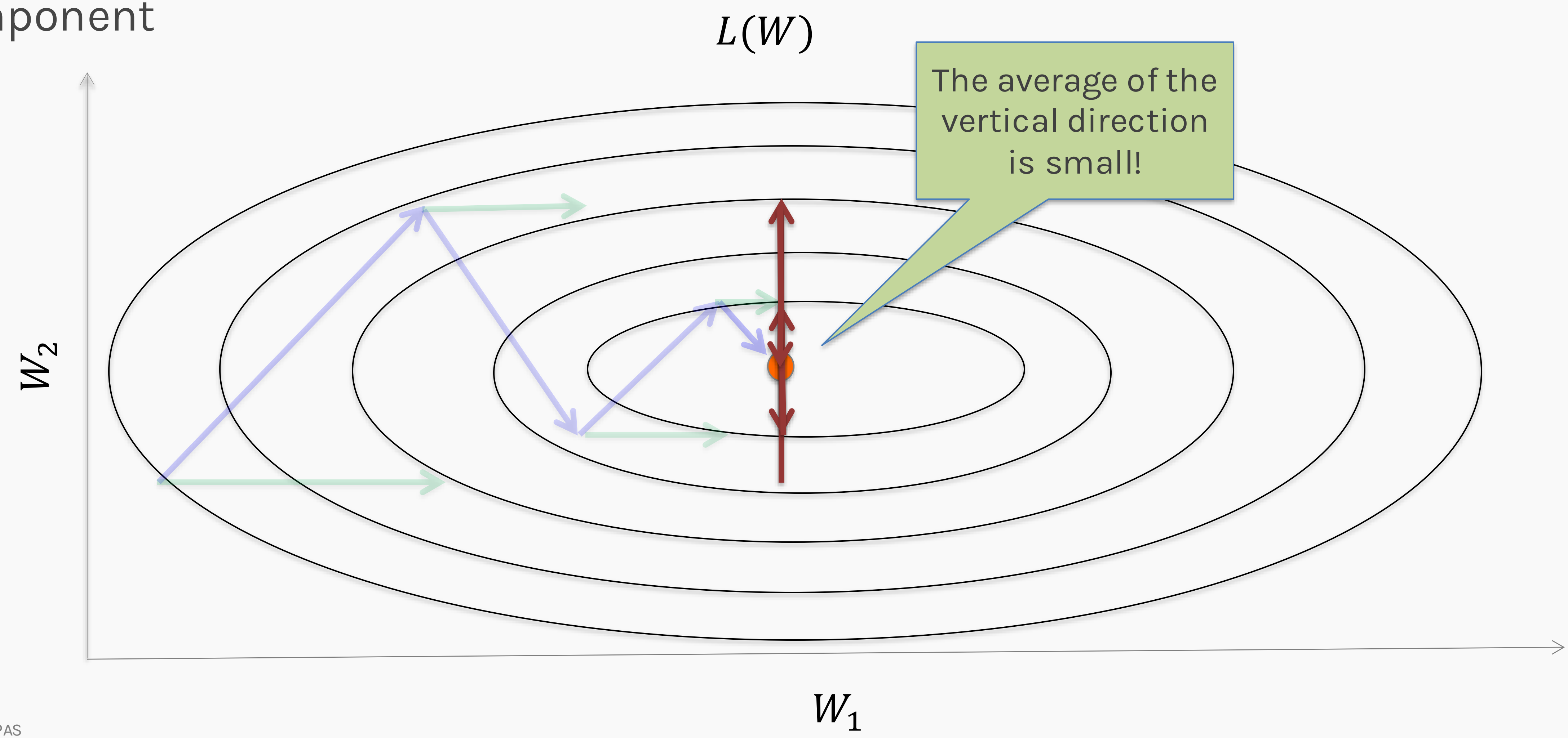
Momentum

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



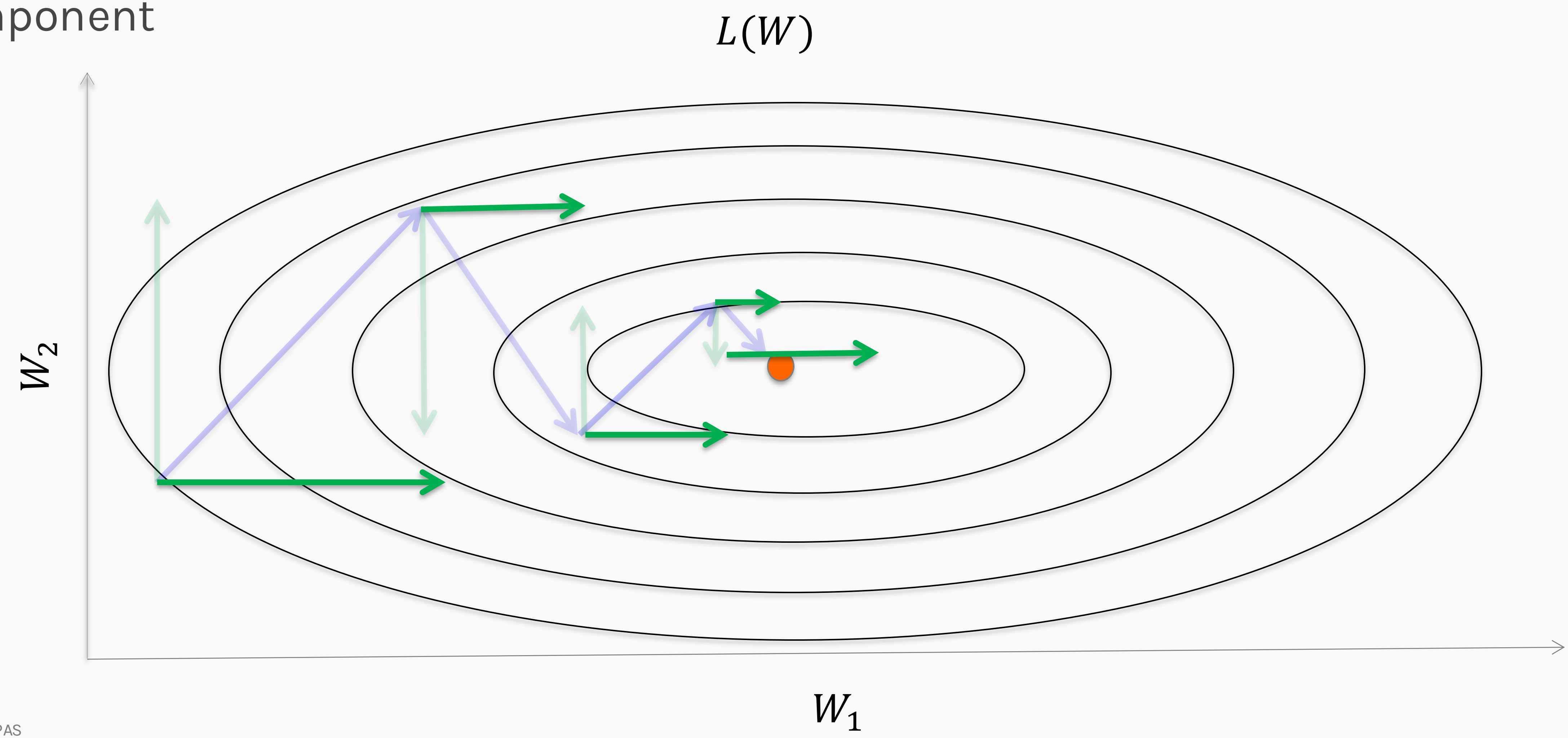
Momentum

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



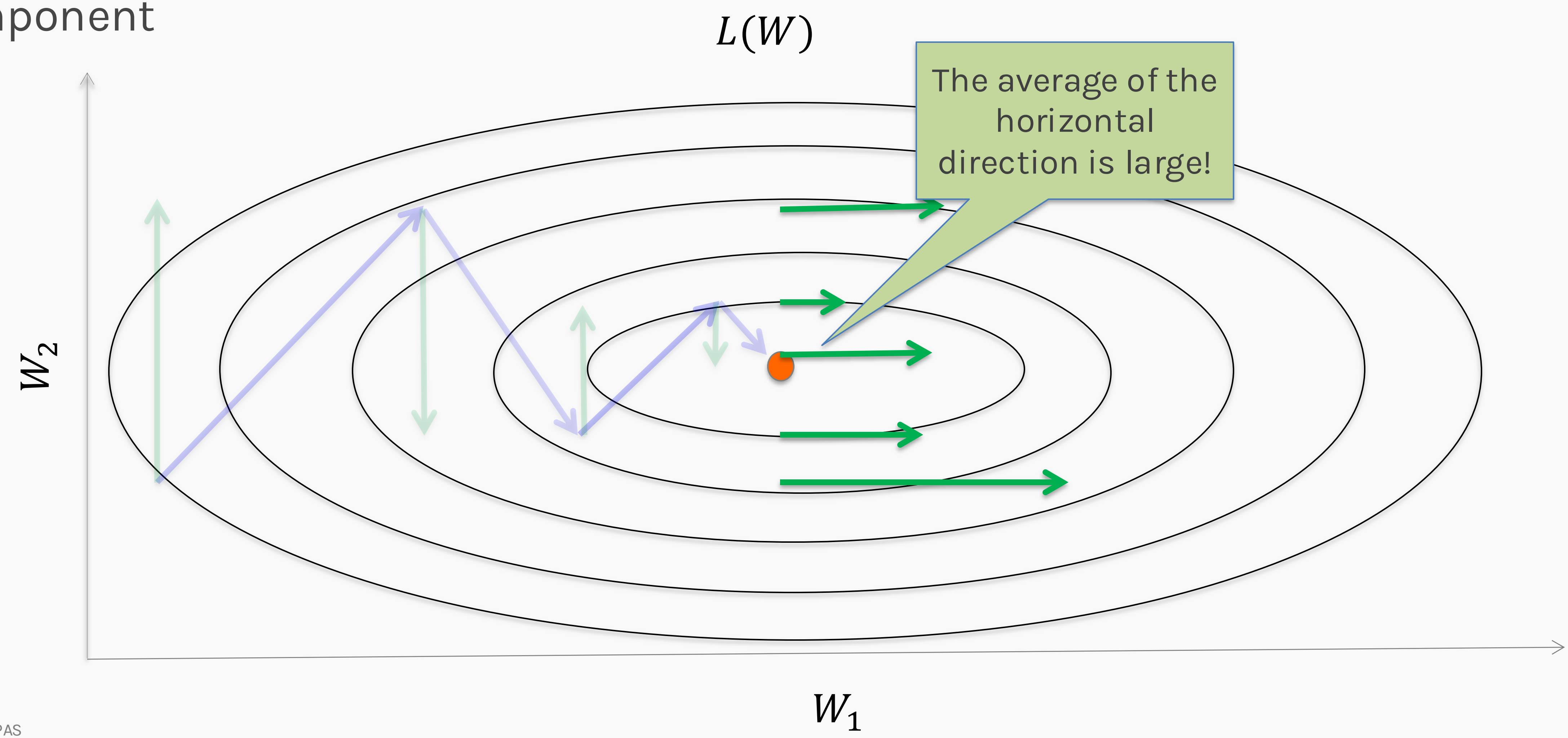
Momentum

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



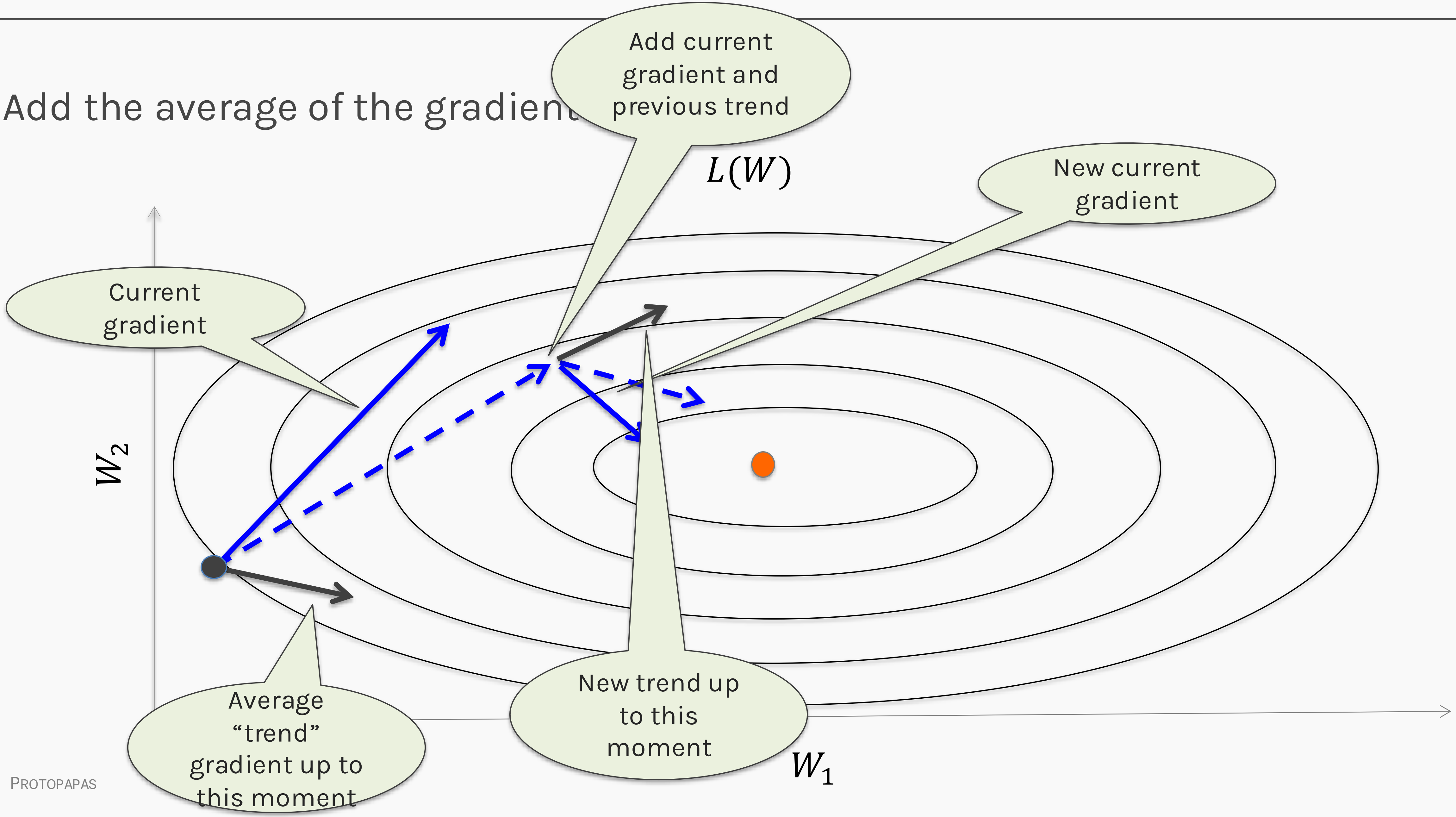
Momentum

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



Momentum

Add the average of the gradient



Momentum

Old gradient descent:

$$W^{(t+1)} = W^{(t)} - \eta g^{(t)} \qquad\qquad g^{(t)} = \frac{1}{m}\sum_i \nabla_W L_i \Big|_{W^{(t)}}$$

Momentum

Gradient descent in one dimension:

$$W^{(t+1)} = W^{(t)} - \eta \frac{dL}{dW}$$

Where is the derivative evaluated? Or, at what value of W?

$$W^{(t+1)} = W^{(t)} - \eta \frac{dL}{dW}\Big|_{W^{(t)}}$$

At $W^{(t)}$

Gradient descent in multiple dimensions:

$$W^{(t+1)} = W^{(t)} - \eta \, \nabla_W L \Big|_{W^{(t)}}$$

Momentum

Gradient descent in one dimension:

$$W^{(t+1)} = W^{(t)} - \eta \frac{dL}{dW}$$

$$W^{(t+1)} = W^{(t)} - \eta \frac{dL}{dW}\bigg|_{W^{(t)}}$$

Gradient descent in multiple dimensions:

$$W^{(t+1)} = W^{(t)} - \eta \, \nabla_W L \Big|_{W^{(t)}} = W^{(t)} - \eta \, \frac{1}{m} \sum_i \nabla_W L_i \Big|_{W^{(t)}}$$

Momentum

$$W^{(t+1)} = W^{(t)} - \eta \, \nabla_W L \Big|_{W^{(t)}} = W^{(t)} - \eta \, \underbrace{\frac{1}{m} \sum_i \nabla_W L_i \Big|_{W^{(t)}}}_{g^{(t)}}$$

$$g^{(t)} = \frac{1}{m} \sum_i \nabla_W L_i \Big|_{W^{(t)}}$$

Momentum

$$W^{(t+1)} = W^{(t)} - \eta \nabla_W L \Big|_{W^{(t)}} = W^{(t)} - \eta \underbrace{\frac{1}{m} \sum_i \nabla_W L_i \Big|_{W^{(t)}}}_{g^{(t)}}$$

current weights

f is the Neural Network

y_i is the true label

$$g^{(t)} = \frac{1}{m} \sum_i \nabla_W L_i \Big|_{W^{(t)}} = \frac{1}{m} \sum_i \nabla_W L\left(f\left(x_i; W^{(t)}\right), y_i\right)$$

Momentum

Old gradient descent:

$$W^{(t+1)} = W^{(t)} - \eta g^{(t)} \qquad g^{(t)} = \frac{1}{m} \sum_i \nabla_W L_i \Big|_{W^{(t)}}$$

Momentum

Old gradient descent:

$$W^{(t+1)} = W^{(t)} - \eta g^{(t)} \qquad g^{(t)} = \frac{1}{m}\sum_i \nabla_W L_i \Big|_{W^{(t)}}$$

If $\alpha = 0$ old SGD
If $\alpha = 1$ we only consider the trend
Typically: $\alpha \in [0.9, 0.99]$

New gradient descent with momentum:

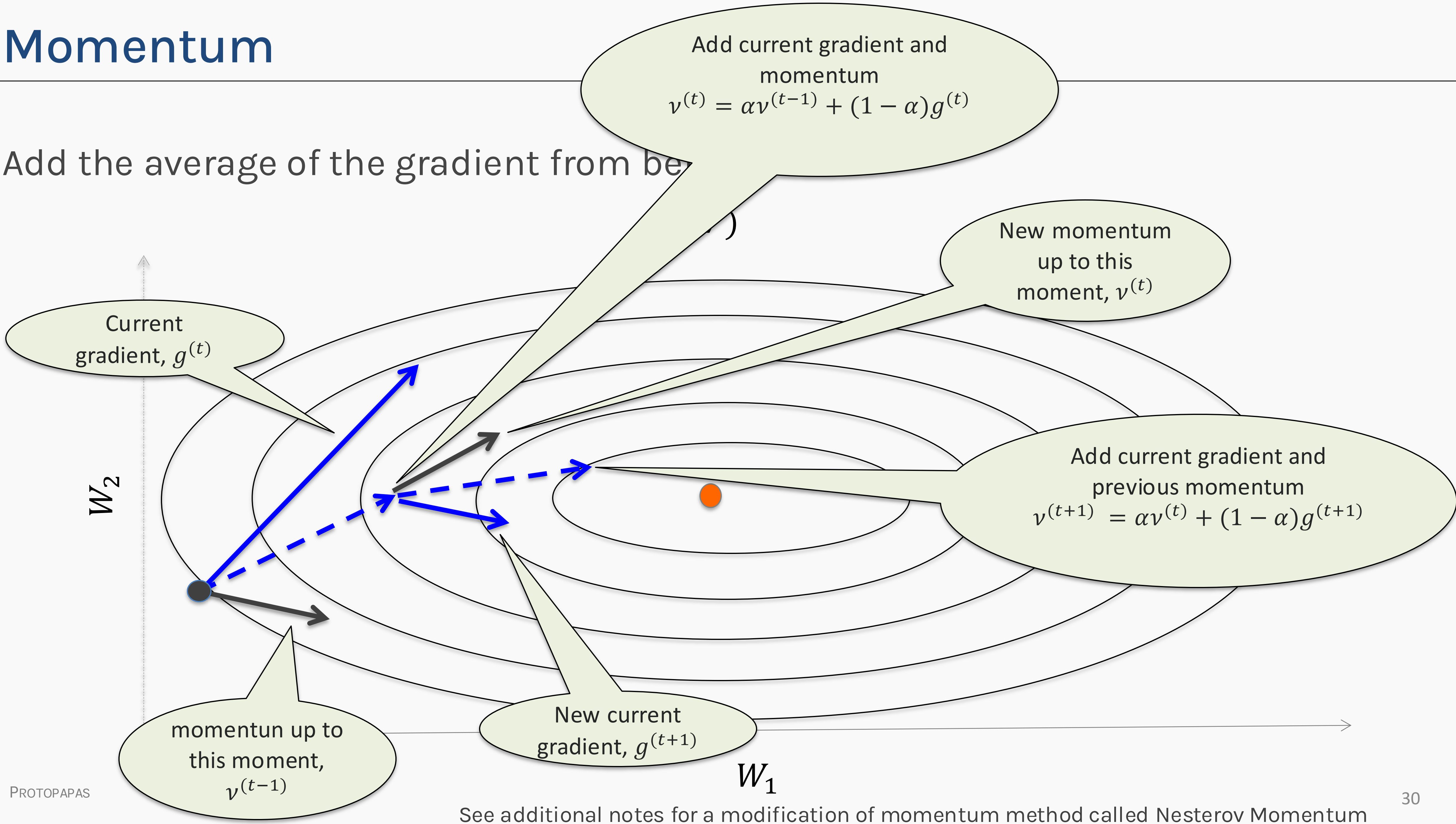
$$v^{(t)} = \alpha v^{(t-1)} + (1-\alpha) g^{(t)} \qquad W^{(t+1)} = W^{(t)} - \eta v^{(t)}$$

Momentum (trend)

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

Momentum

Add the average of the gradient from be

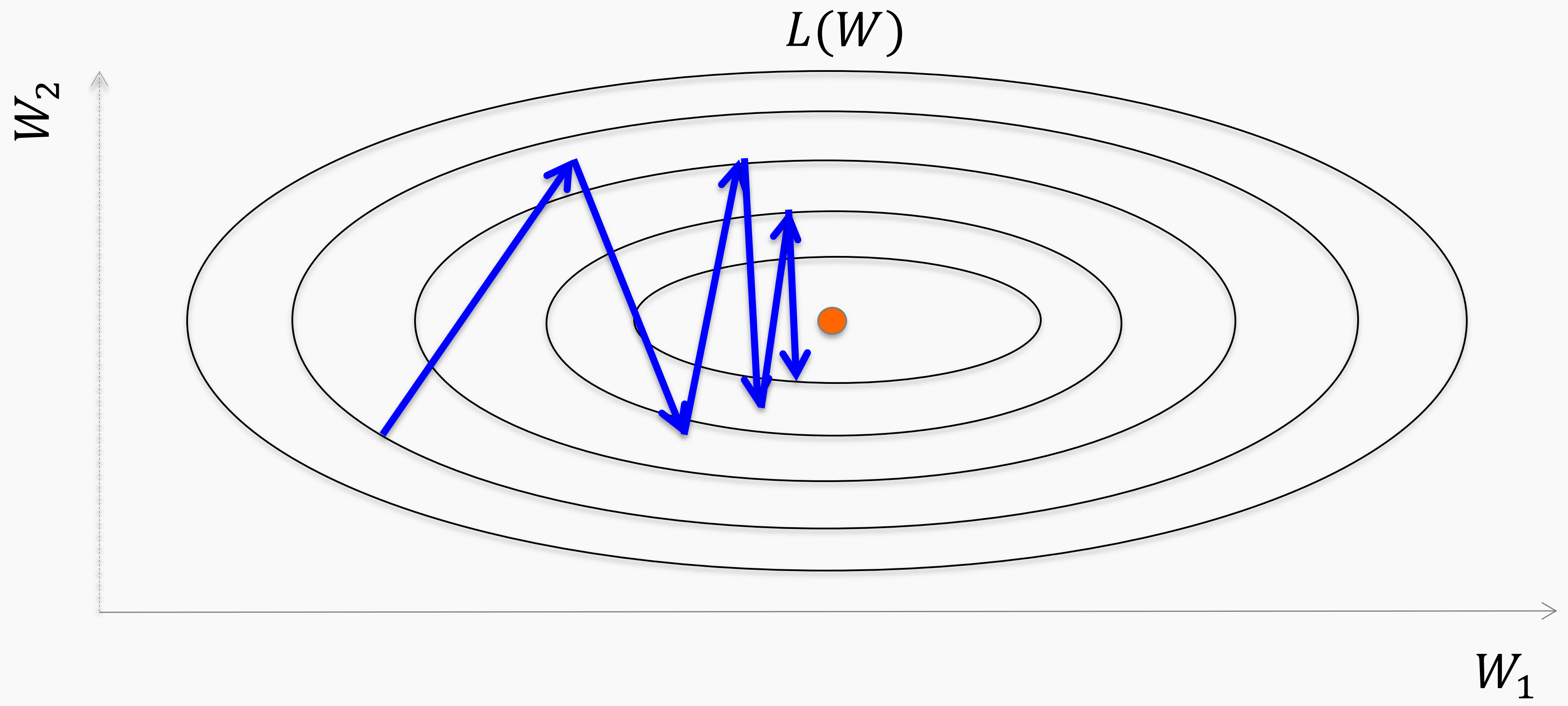


See additional notes for a modification of momentum method called Nesterov Momentum

Outline

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

Adaptive Learning Rates

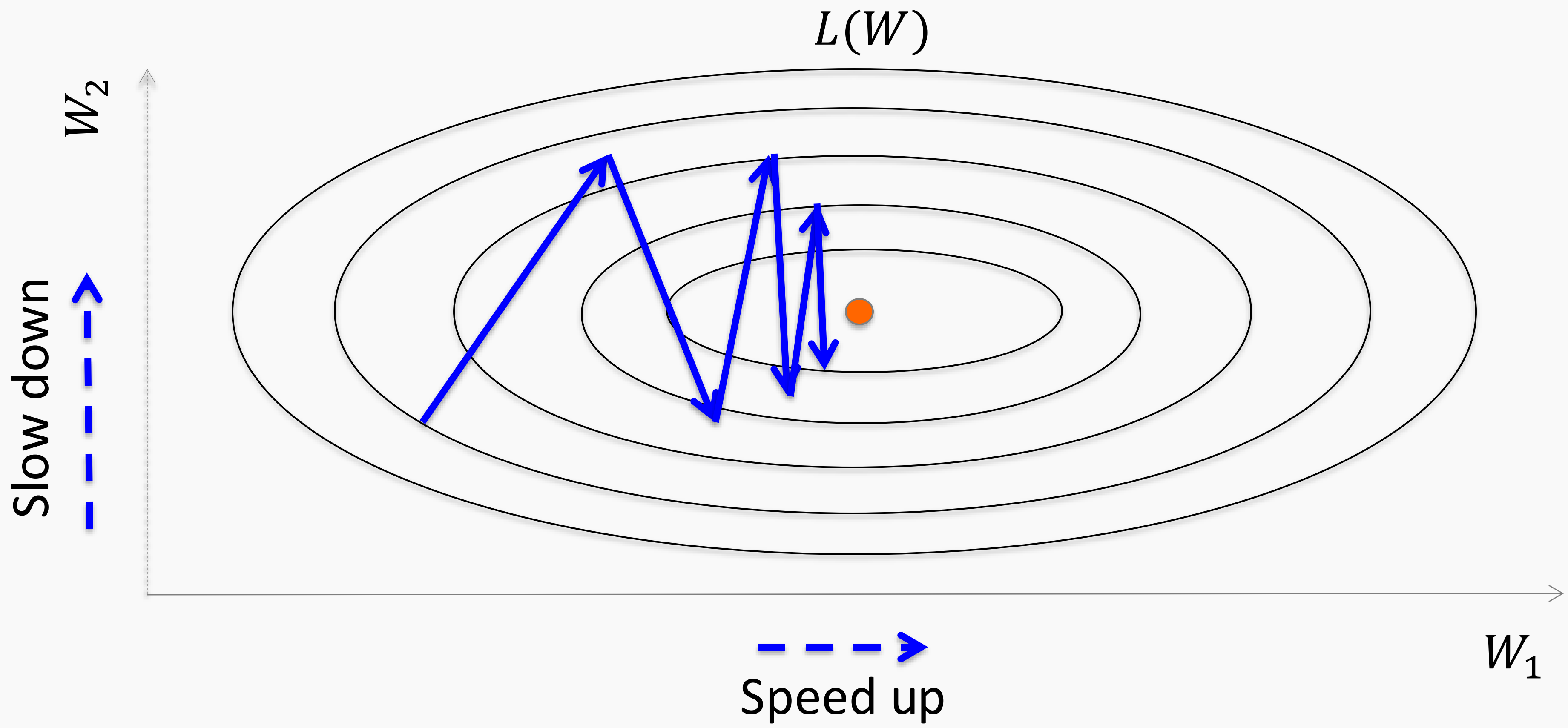


We observe oscillations along vertical direction

- Learning must be slower along parameter W_2 than W_1

Use a different learning rate for each parameter?

Adaptive Learning Rates

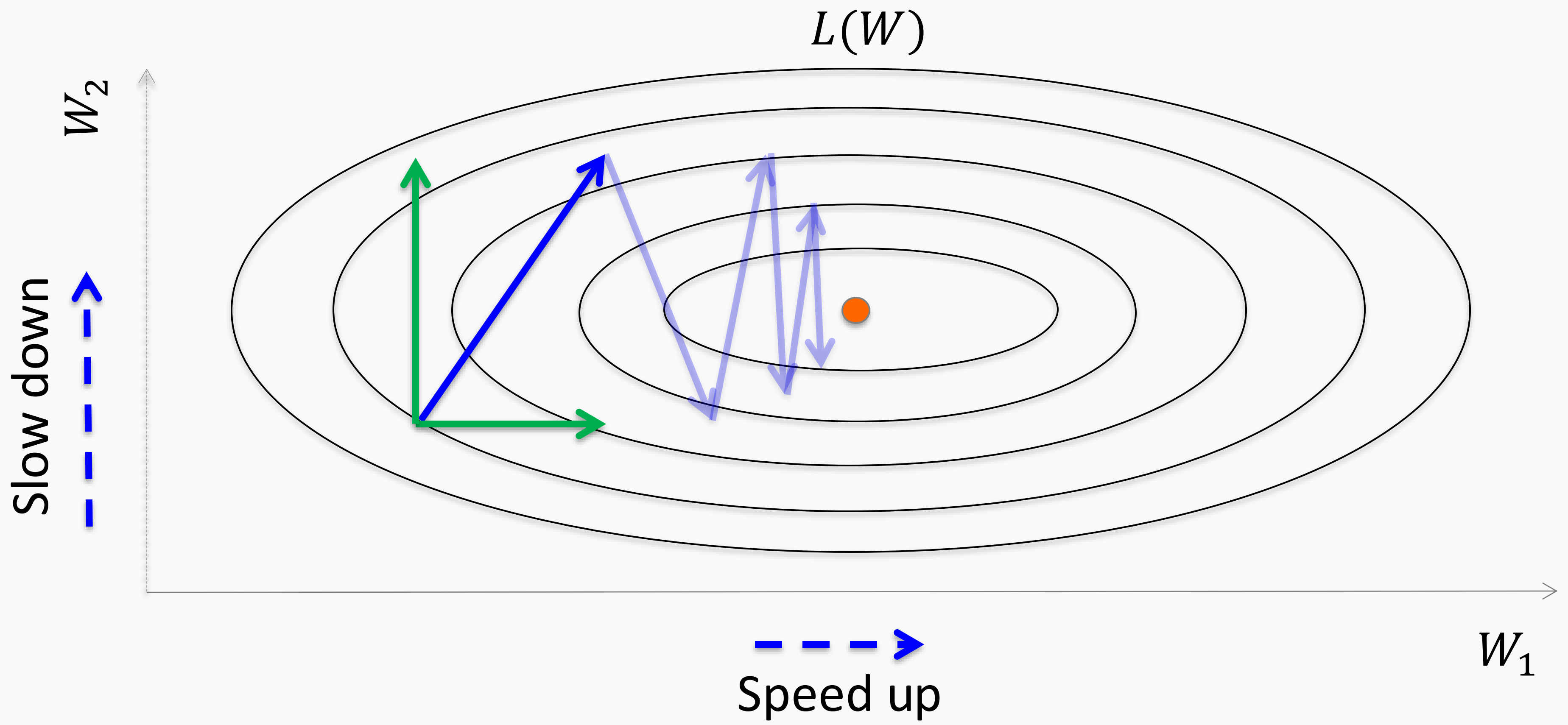


We observe oscillations along vertical direction

- Learning must be slower along parameter W_2 than W_1

Use a different learning rate for each parameter?

Adaptive Learning Rates

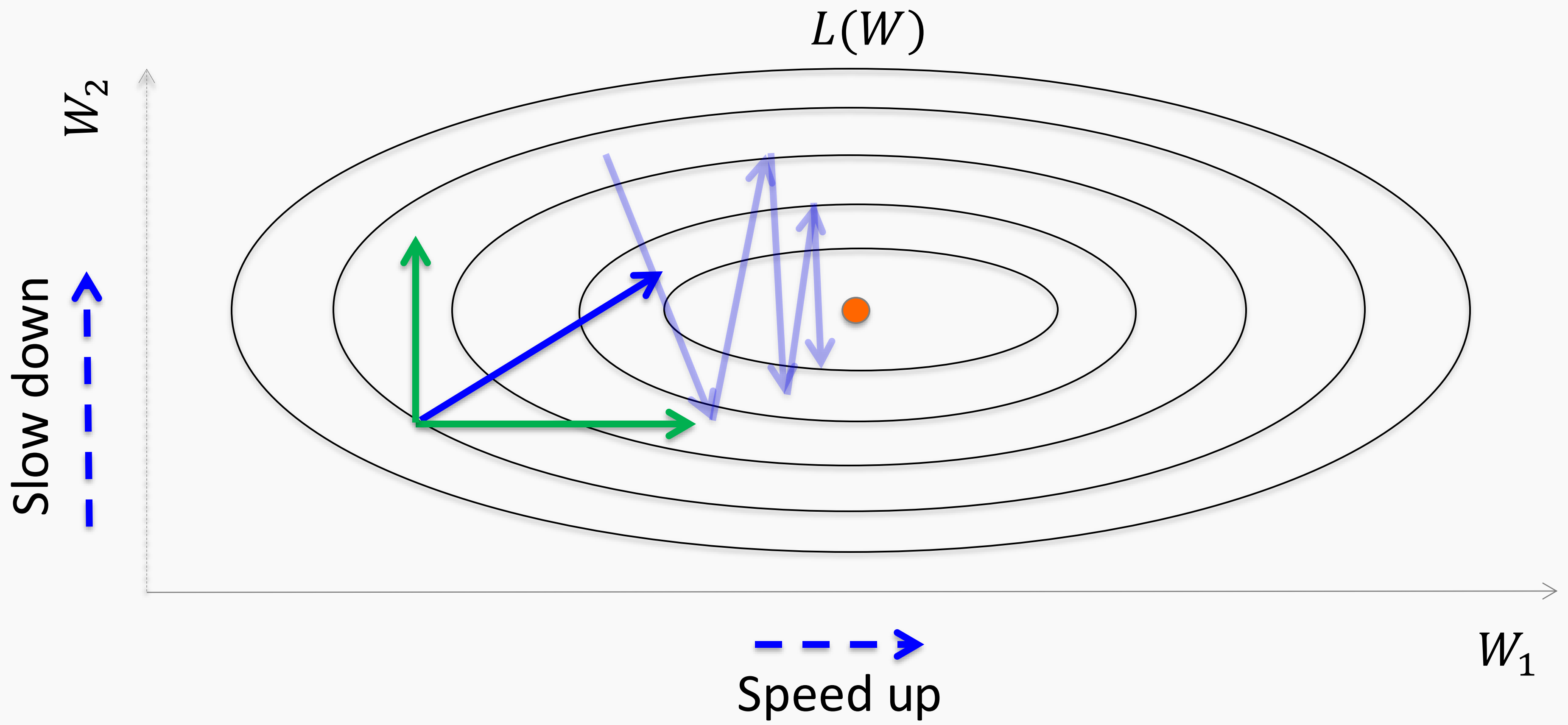


We observe oscillations along vertical direction

– Learning must be slower along parameter W_2 than W_1

Use a different learning rate for each parameter?

Adaptive Learning Rates

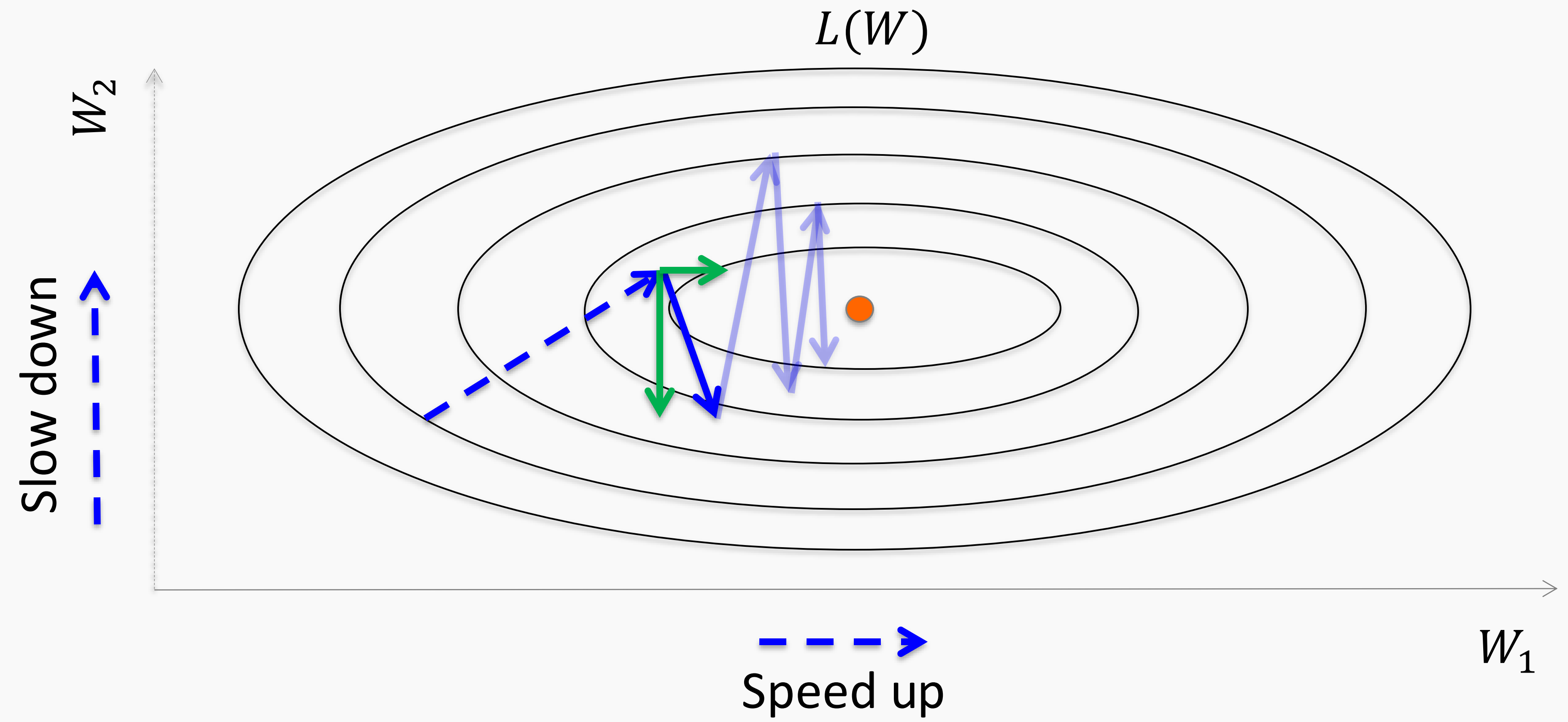


We observe oscillations along vertical direction

– Learning must be slower along parameter W_2 than W_1

Use a different learning rate for each parameter?

Adaptive Learning Rates

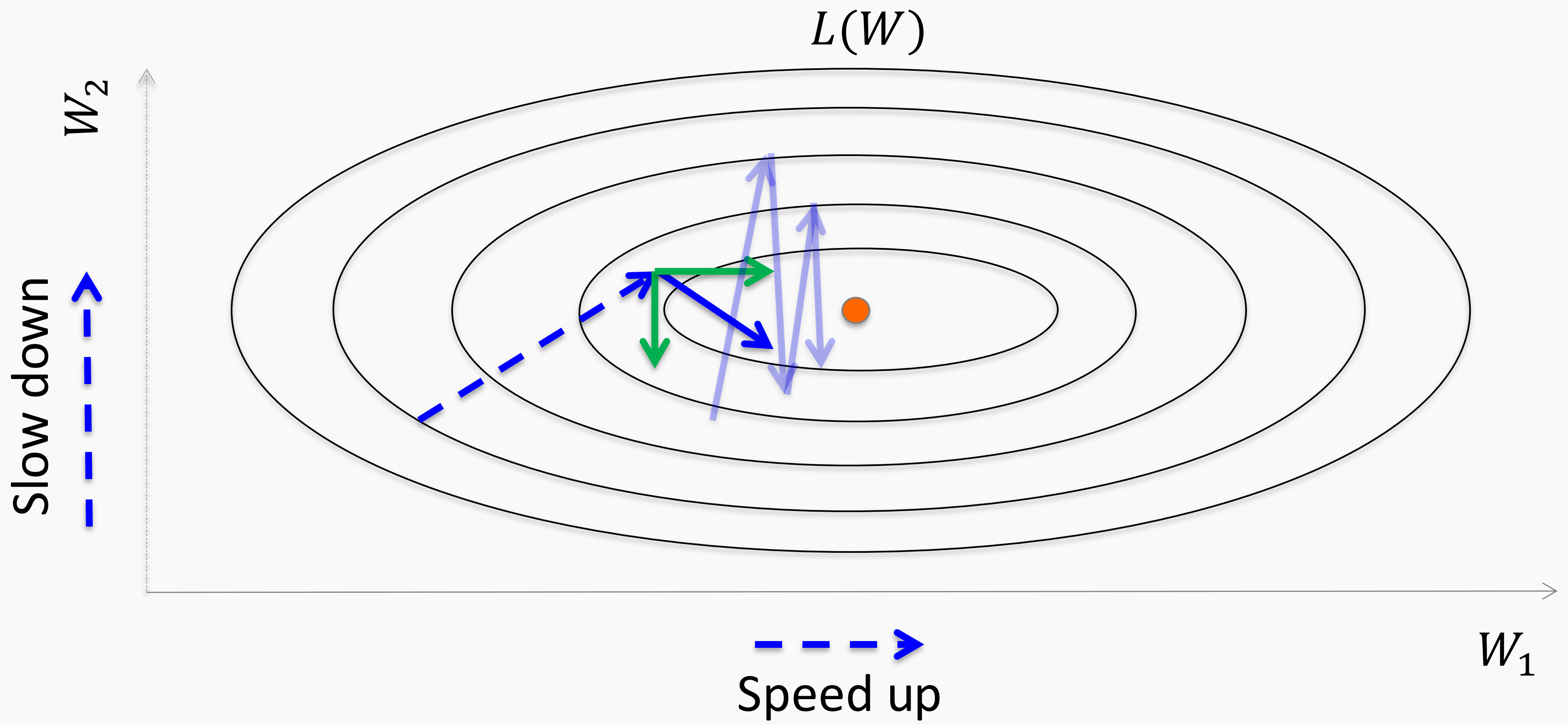


We observe oscillations along vertical direction

- Learning must be slower along parameter W_2 than W_1

Use a different learning rate for each parameter?

Adaptive Learning Rates

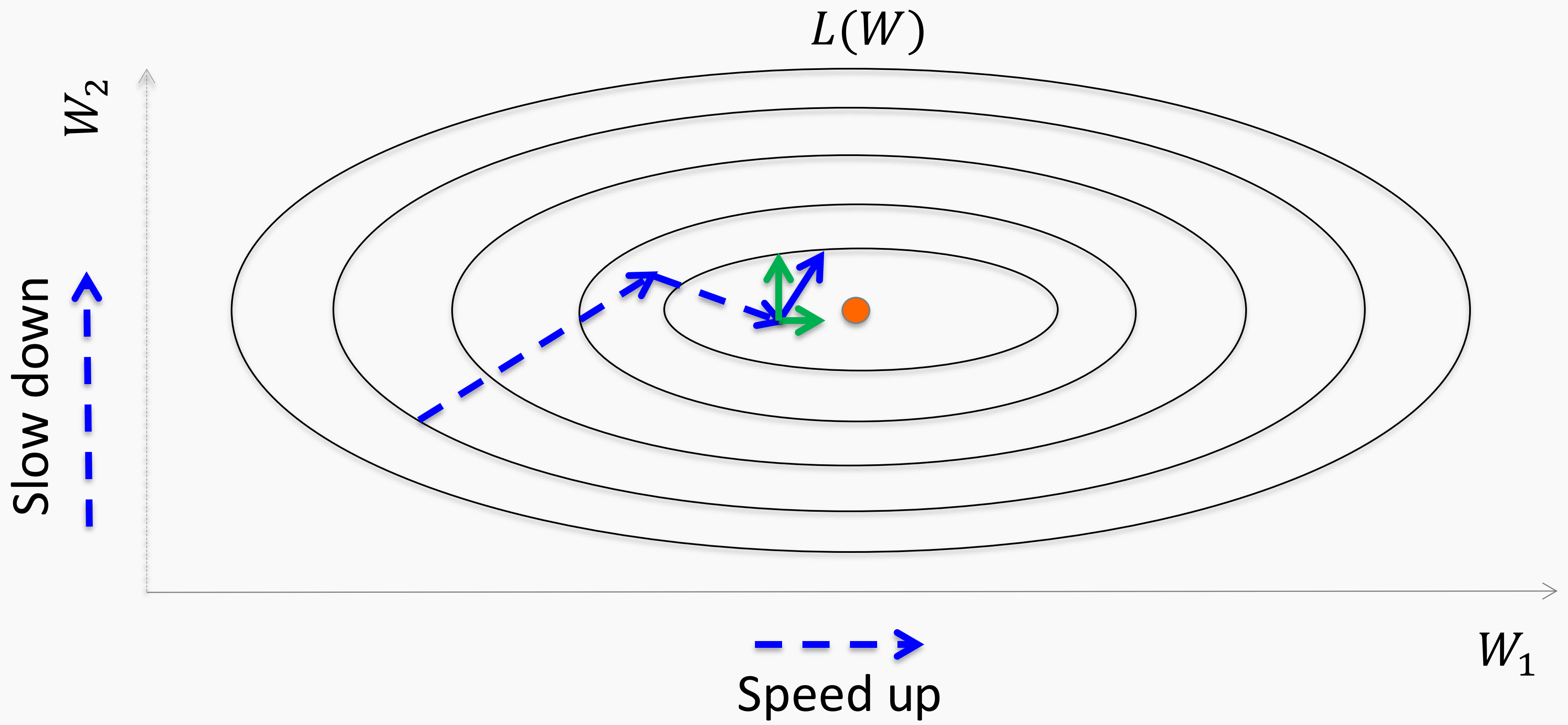


We observe oscillations along vertical direction

– Learning must be slower along parameter W_2 than W_1

Use a different learning rate for each parameter?

Adaptive Learning Rates

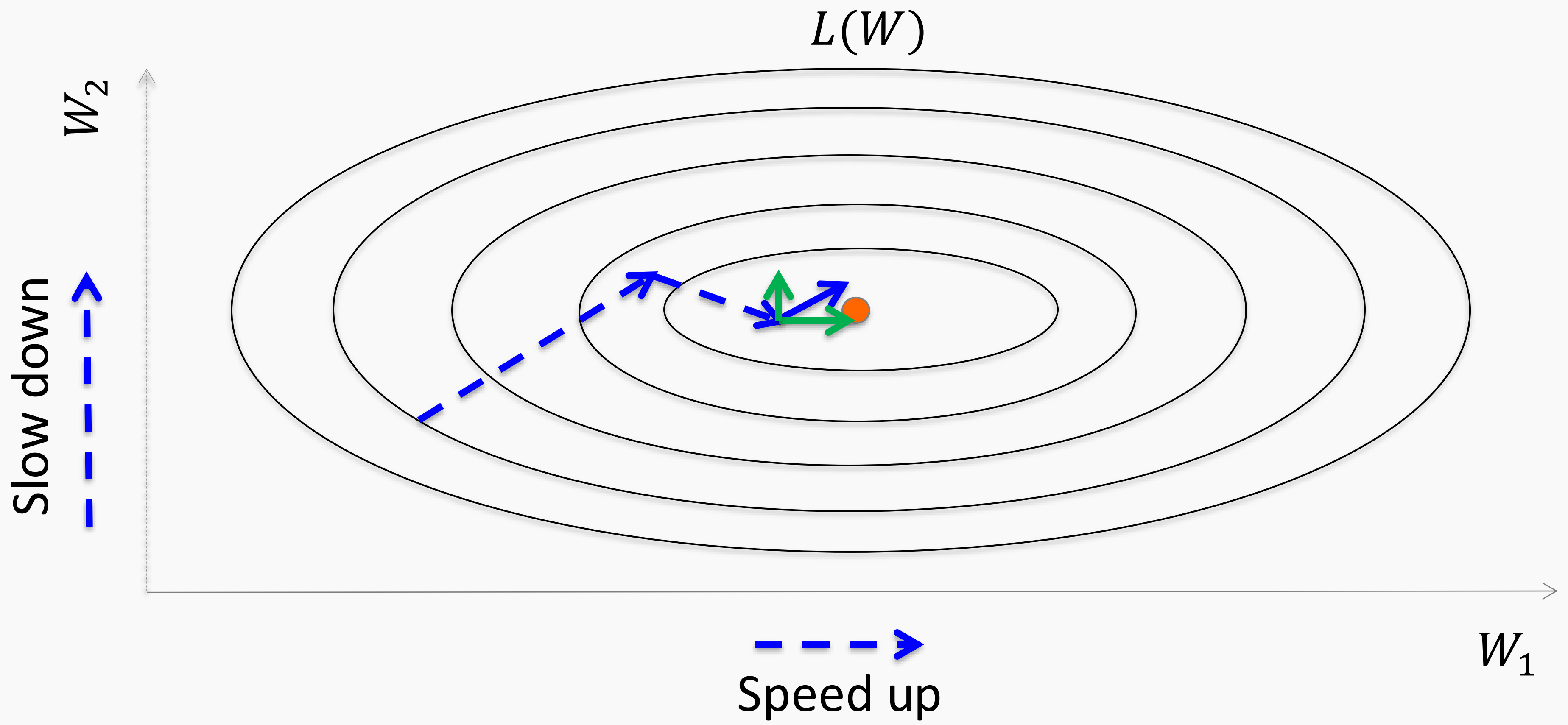


We observe oscillations along vertical direction

- Learning must be slower along parameter W_2 than W_1

Use a different learning rate for each parameter?

Adaptive Learning Rates

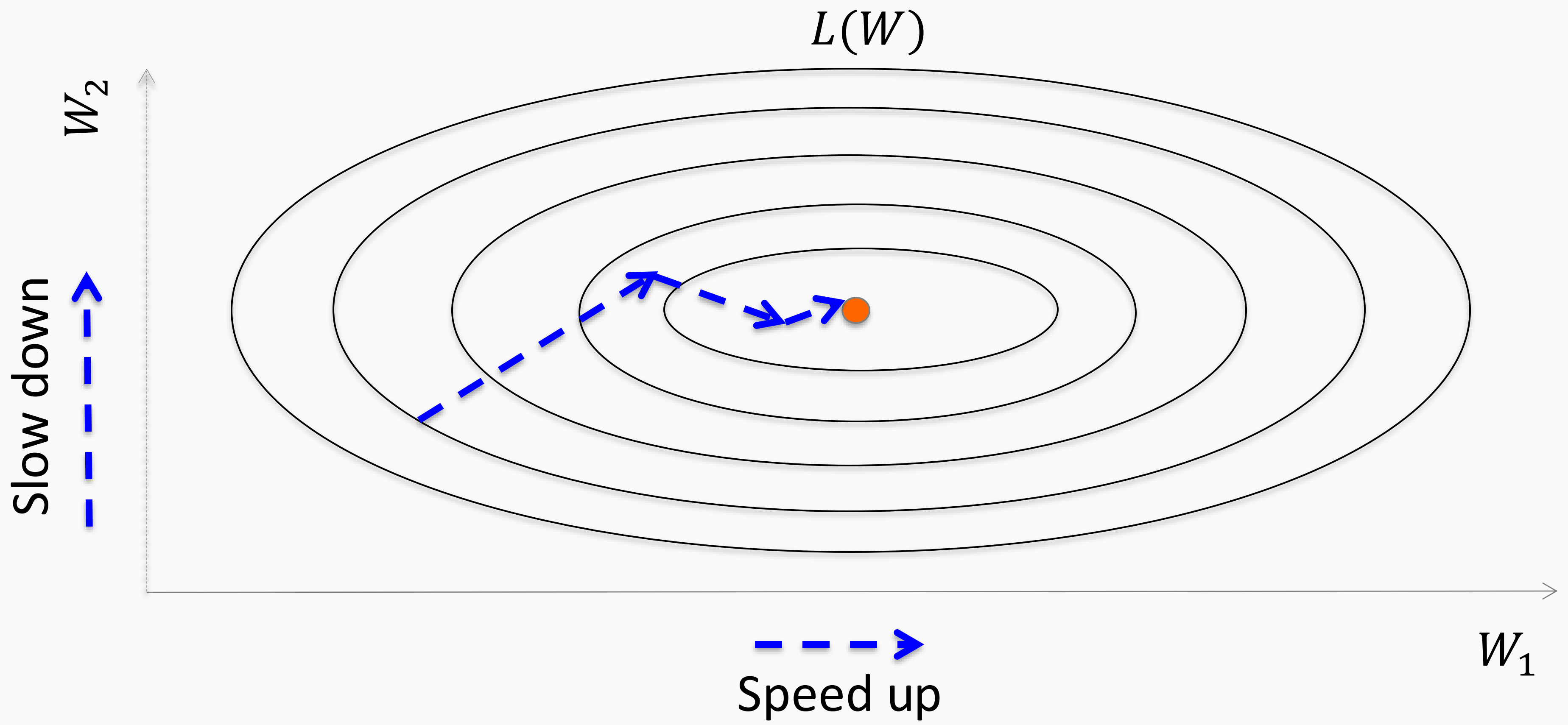


We observe oscillations along vertical direction

– Learning must be slower along parameter W_2 than W_1

Use a different learning rate for each parameter?

Adaptive Learning Rates



We observe oscillations along vertical direction

- Learning must be slower along parameter W_2 than W_1

Use a different learning rate for each parameter?

AdaGrad

Old gradient descent:

$$\boxed{W^{(t+1)} = W^{(t)} - \eta g^{(t)}}$$

We would like to speed up or slow down, which means we want $\eta's$ to vary for different weights and change with iterations.

$$W_j^{(t+1)} = W_j^{(t)} - \eta_j^{(t)} g_j^{(t)}$$

We also aim to accelerate when derivatives are small and decelerate when derivatives are large. That means, $\eta_j^{(t)}$, should be inversely proportional to the value of $|g_j|$. To do that, we define a variable, r, as the cumulative sum of squared gradients.

$$r_j^{(t)} = r_j^{(t-1)} + g_j^{(t)^2}$$

AdaGrad

$$r_j^{(t)} = r_j^{(t-1)} + g_j^{(t)^2}$$

we can use it to adapt the learning rates.

$$\boxed{W_j^{(t+1)} = W_j^{(t)} - \frac{\epsilon}{\sqrt{r_j^{(t)}}} g_j^{(t)}}$$

AdaGrad

$$r_j^{(t)} = r_j^{(t-1)} + g_j^{(t)^2}$$

we can use it to adapt the learning rates.

$$W_j^{(t+1)} = W_j^{(t)} - \frac{\epsilon}{\delta + \sqrt{r_j^{(t)}}} g_j^{(t)}$$

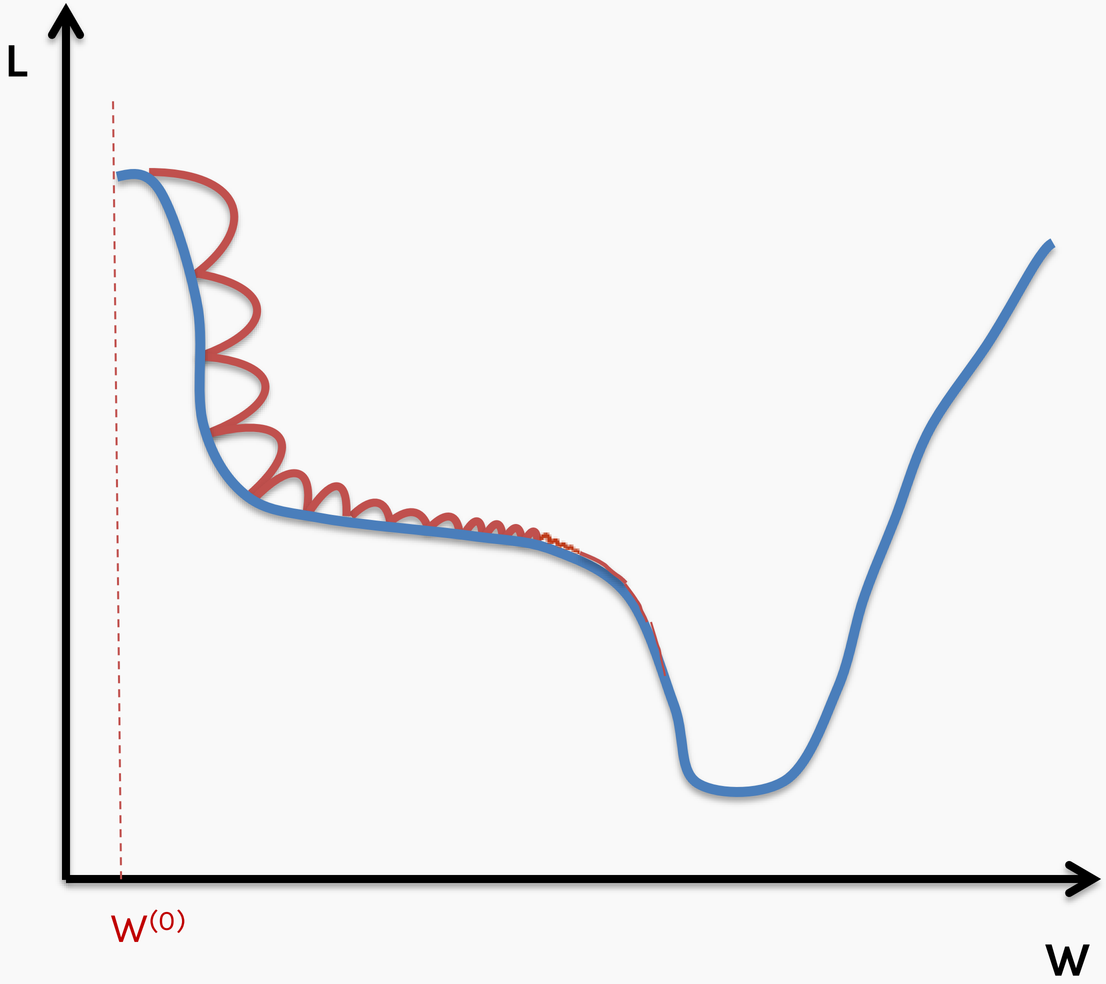
ϵ is now the learning rate to be specified by us.

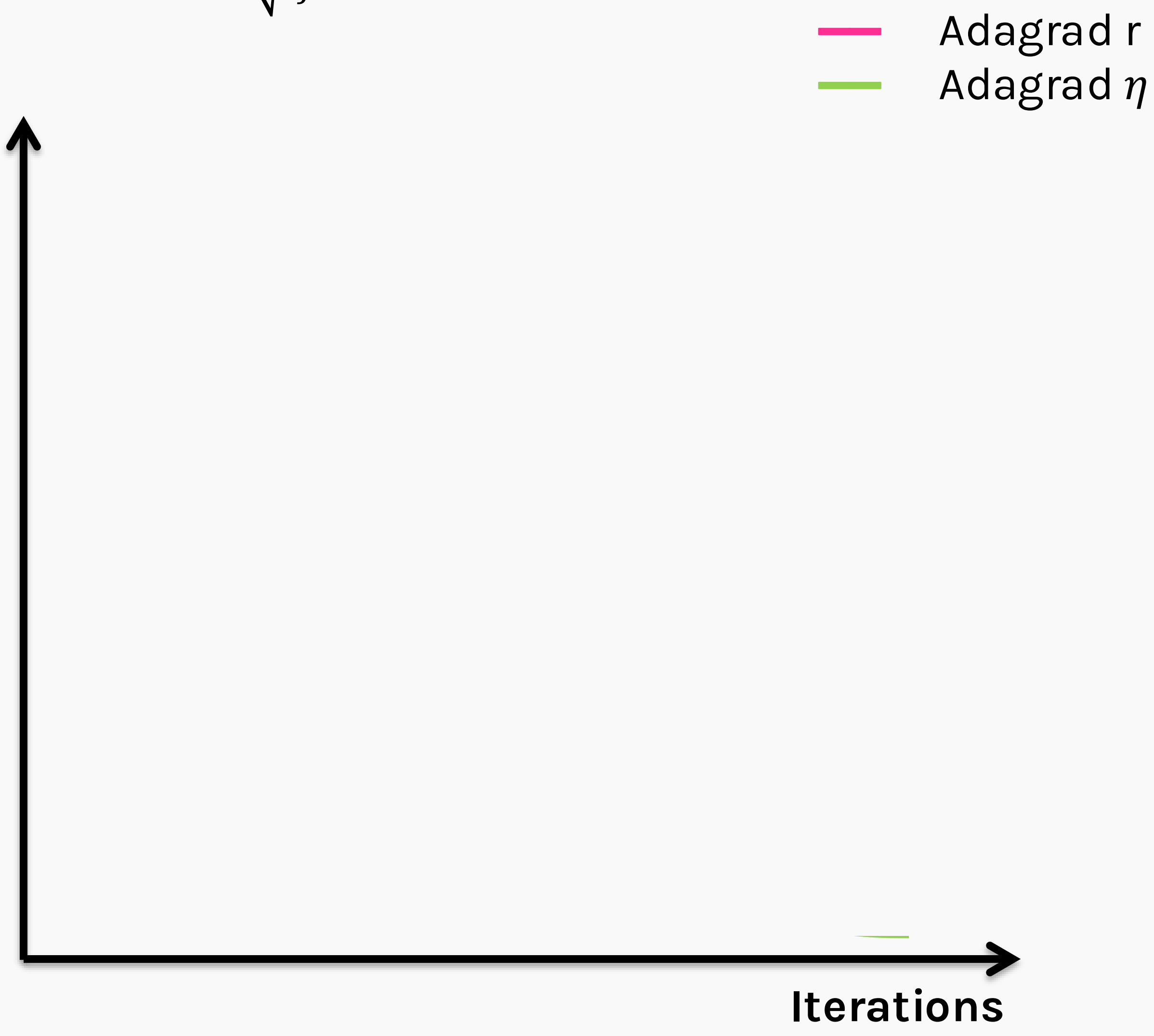
$\eta = \frac{\epsilon}{\delta + \sqrt{r_j^{(t)}}}$ is the effective learning rate

δ is a small number, ensuring that η does not become too large

$$\text{AdaGrad: } r_j^{(t)} = r_j^{(t-1)} + {g_j^{(t)}}^2$$

$$\eta_j^{(t)} = \frac{\epsilon}{\delta + \sqrt{r_j^{(t)}}}$$

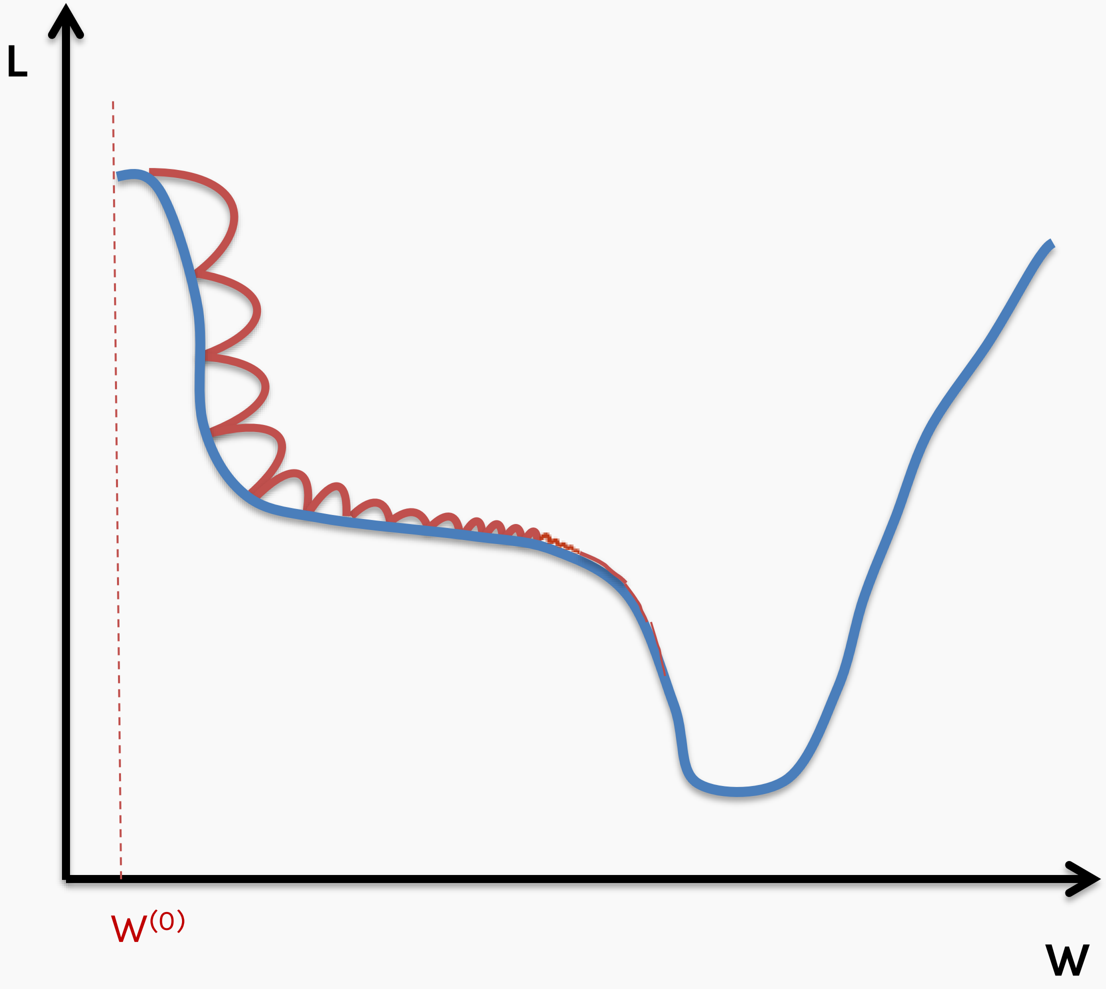


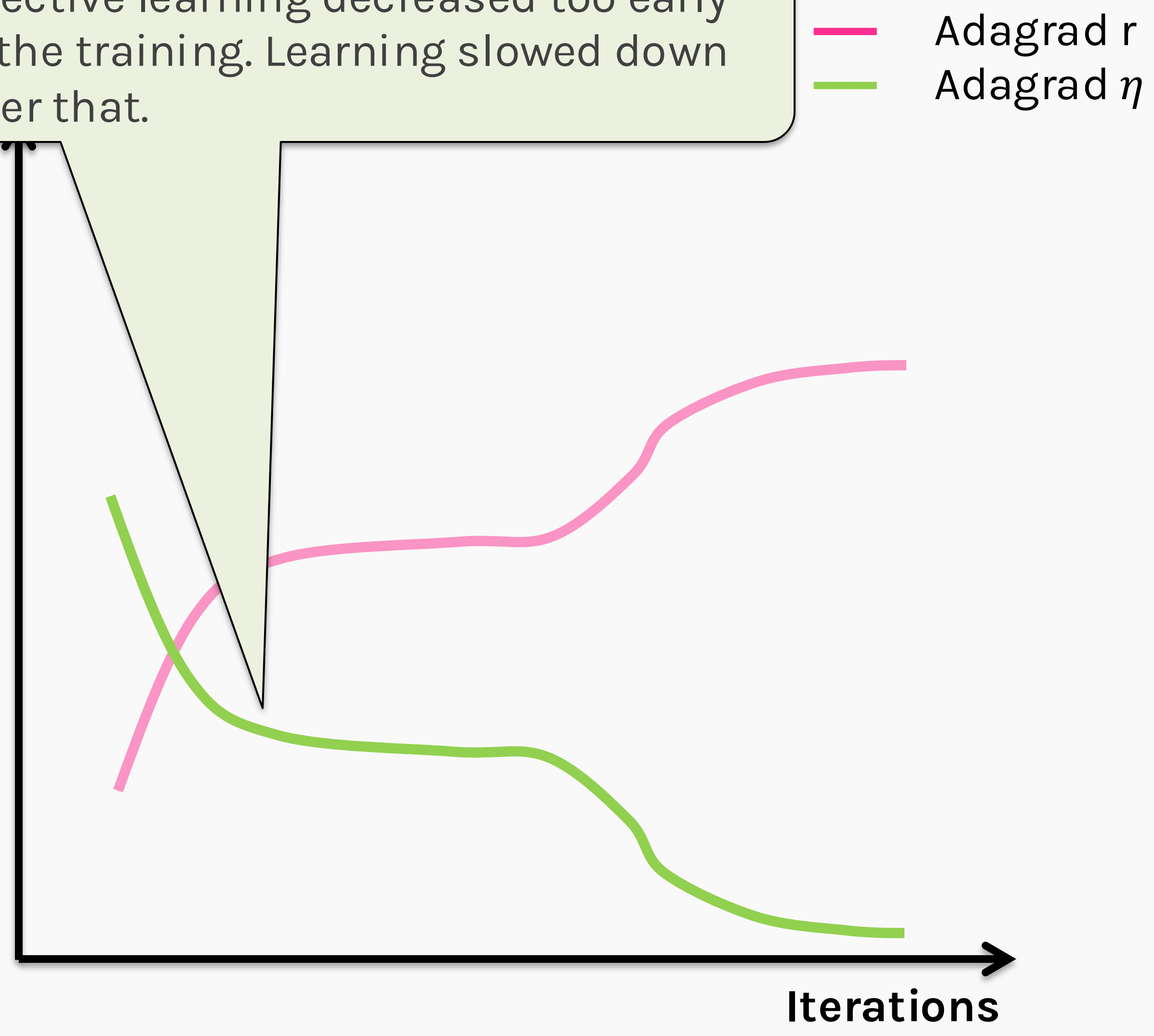


AdaGrad: $r_j^{(t)} = r_j^{(t-1)} + {g_j^{(t)}}^2$

$$\eta_j^{(t)} = \frac{\epsilon}{\delta + \sqrt{r_j^{(t)}}}$$

Effective learning decreased too early in the training. Learning slowed down after that.





RMSProp

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

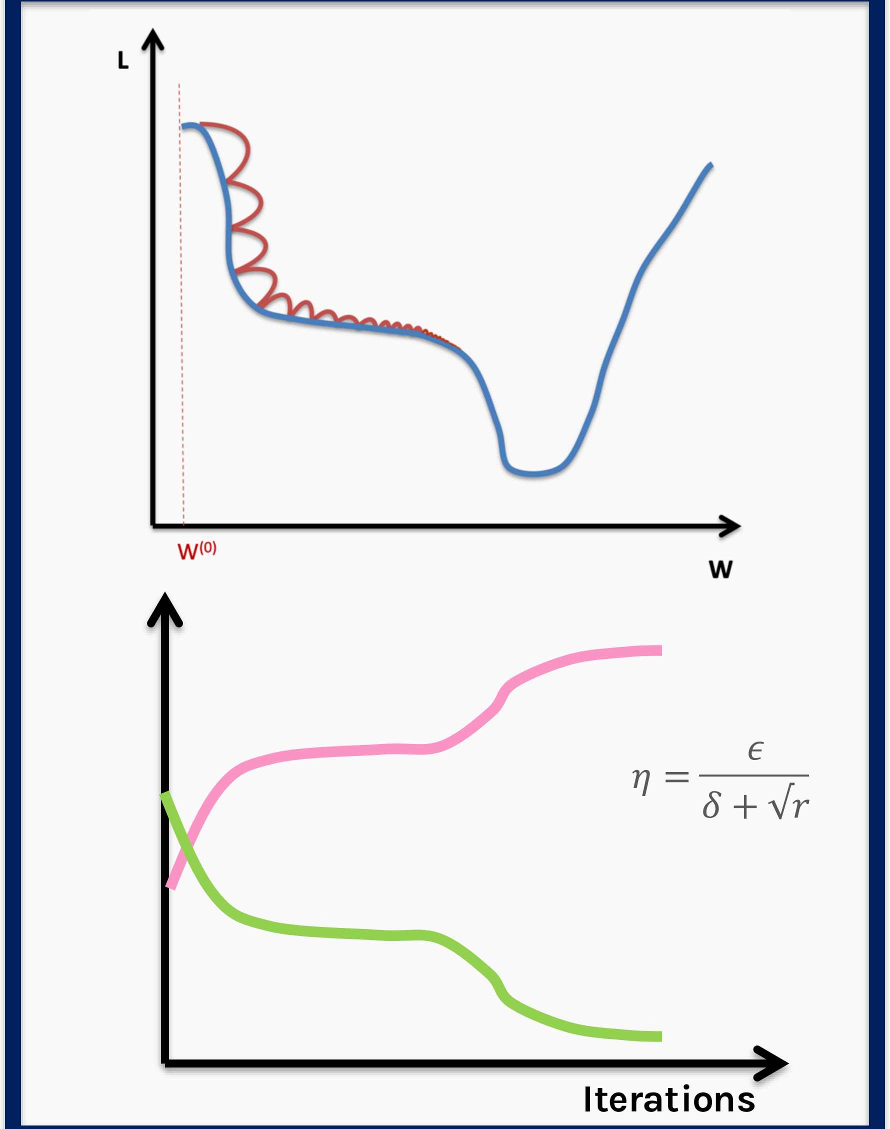
AdaGrad

$$r_j^{(t)} = r_j^{(t-1)} + {g_j^{(t)}}^2$$

RMSProp

$$r_j^{(t)} = \rho r_j^{(t-1)} + (1-\rho)\, {g_j^{(t)}}^2$$

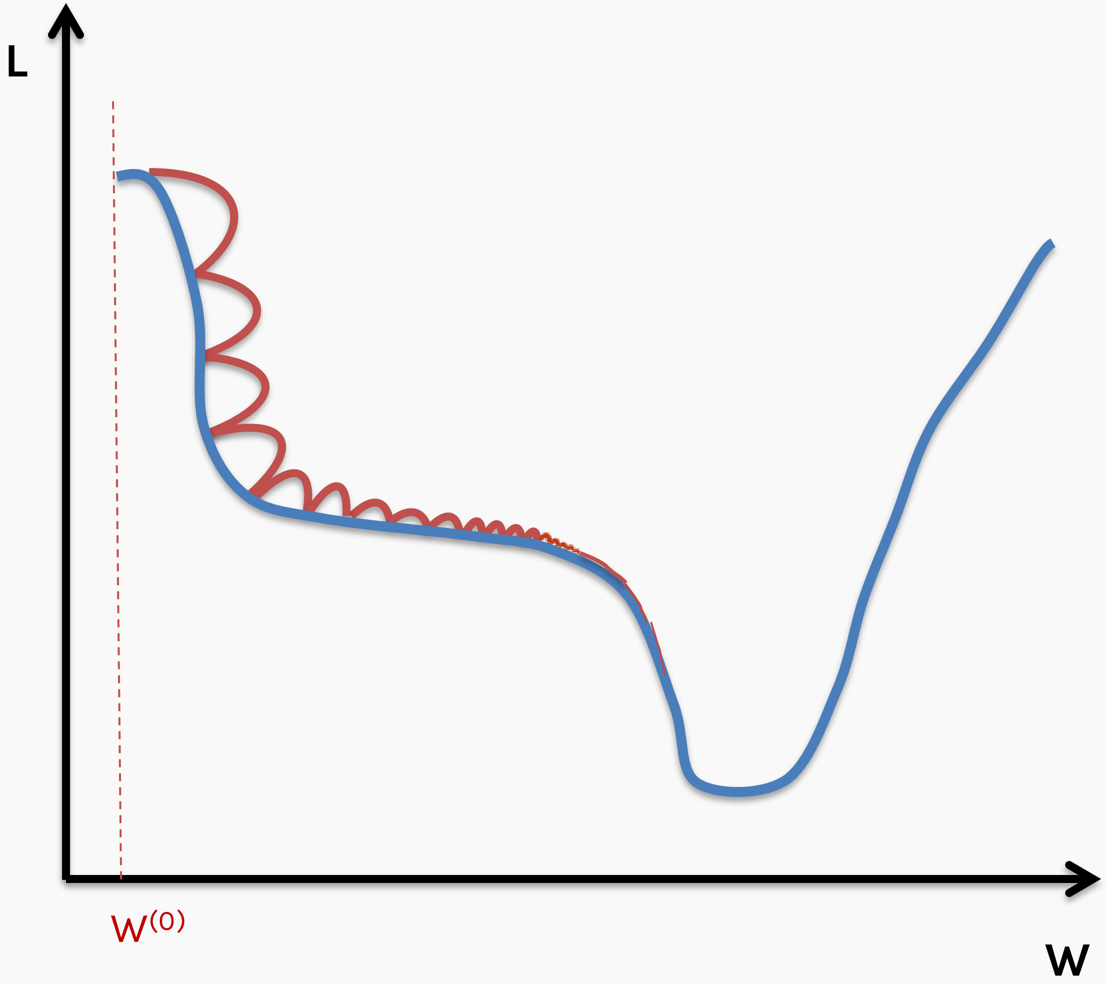
$$W_j^{(t+1)} = W_j^{(t)} - \frac{\epsilon}{\delta + \sqrt{r_j^{(t)}}} g_j^{(t)}$$

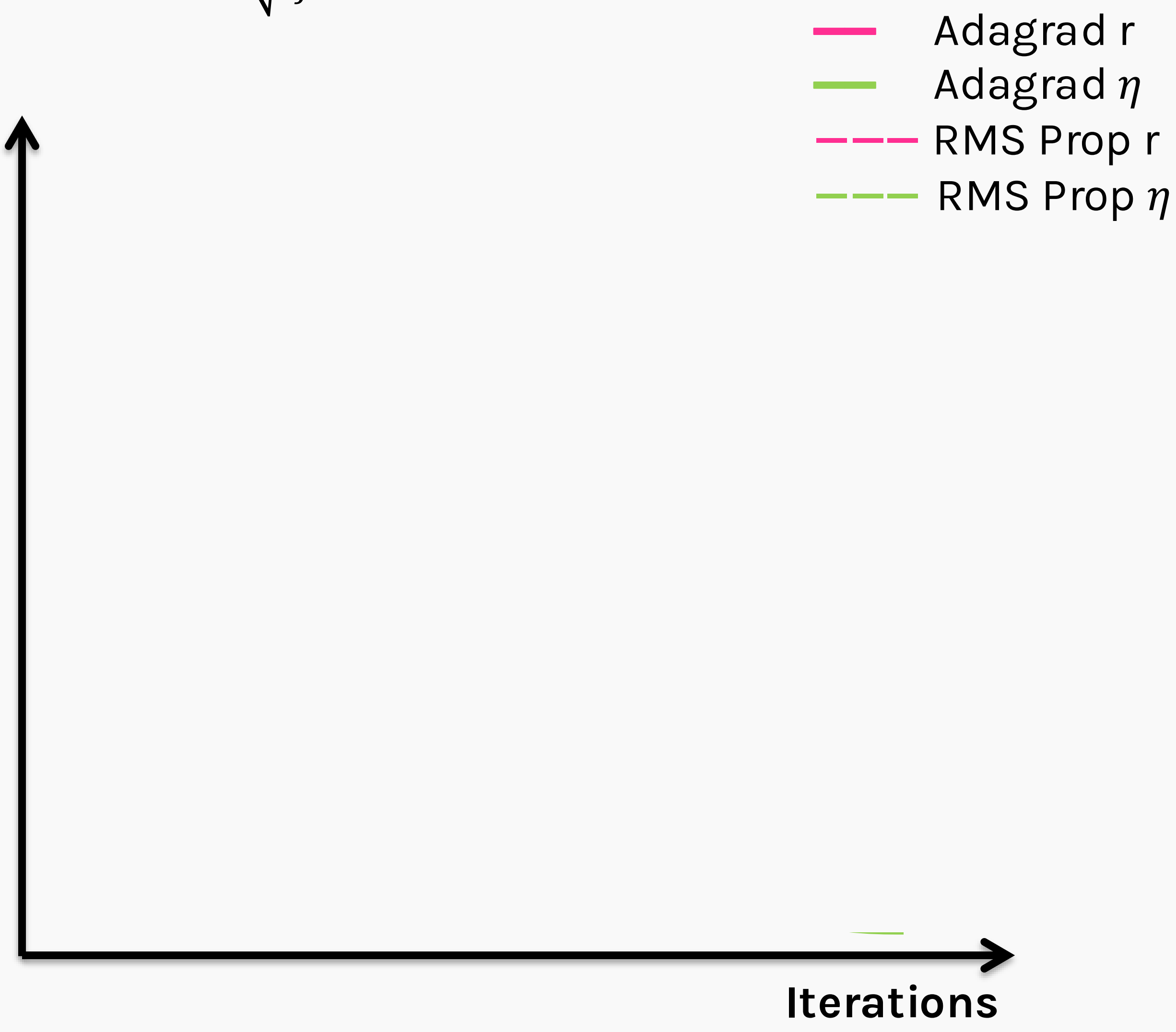


$$\text{RMSProp: } r_j^{(t)} = \rho r_j^{(t-1)} + (1\text{-}\,\rho)\, {g_j^{(t)}}^2$$

$$\text{AdaGrad: } r_j^{(t)} = r_j^{(t-1)} + {g_j^{(t)}}^2$$

$$\eta_j^{(t)} = \frac{\epsilon}{\delta + \sqrt{r_j^{(t)}}}$$

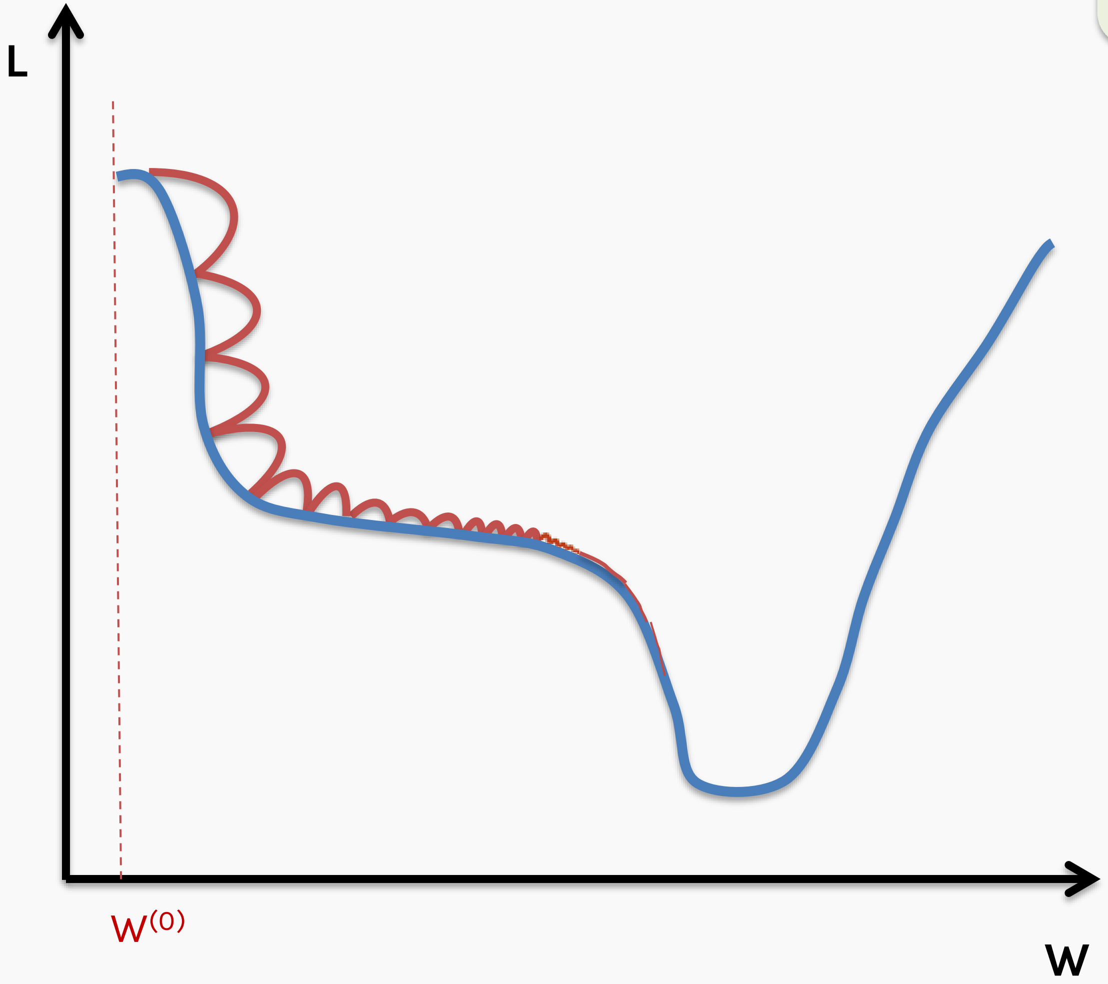




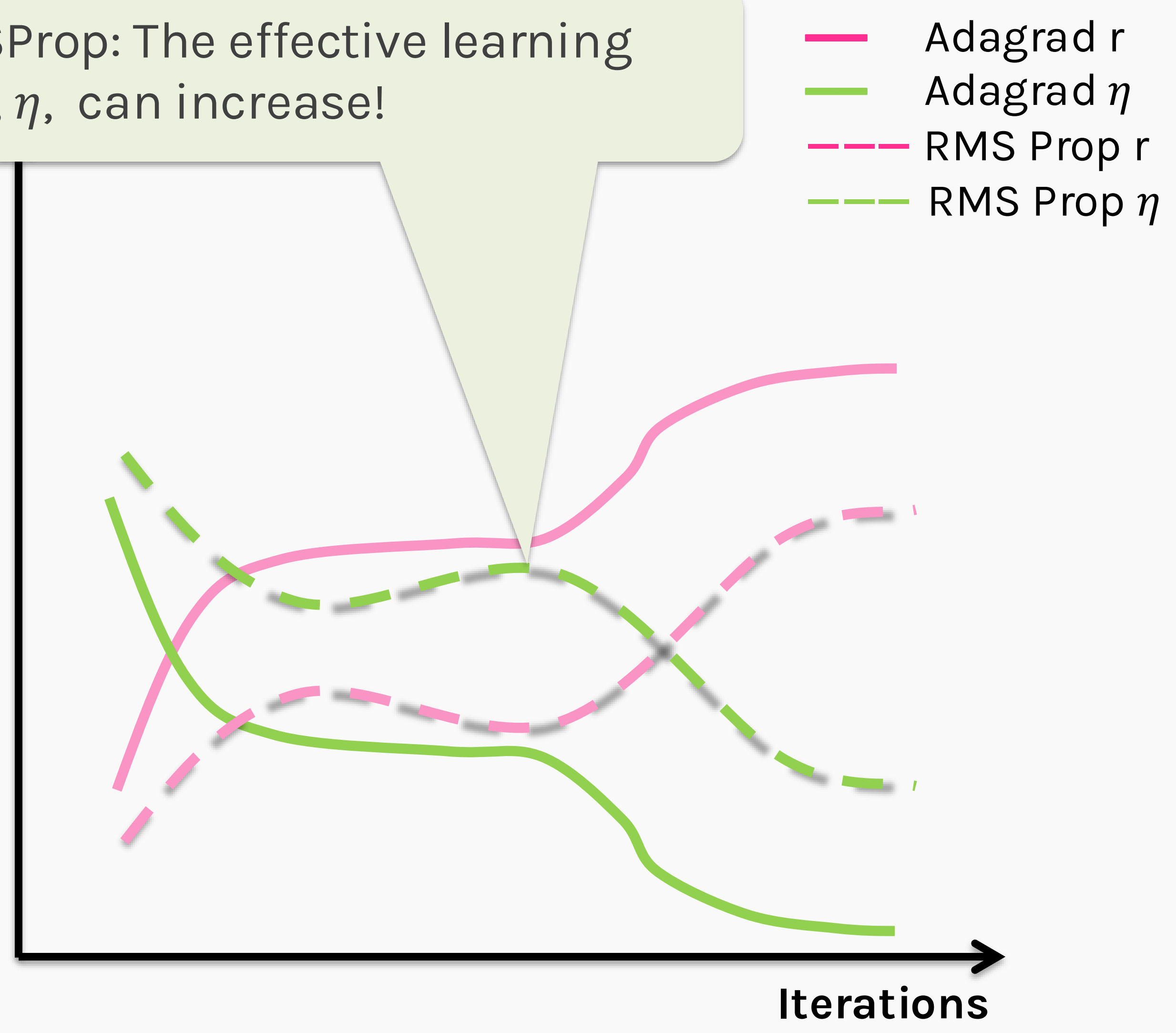
RMSProp: $r_j^{(t)} = \rho r_j^{(t-1)} + (1-\rho)\, {g_j^{(t)}}^2$

AdaGrad: $r_j^{(t)} = r_j^{(t-1)} + {g_j^{(t)}}^2$

$\eta_j^{(t)} = \dfrac{\in}{\delta + \sqrt{r_j^{(t)}}}$



RMSProp: The effective learning rate, η, can increase!



Outline

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

Adam: RMSProp + Momentum

TF/Keras and pytorch use:
$\beta_1 = \rho_1$ and $\beta_2 = \rho_2$
Typical values are $\beta_1 = 0.9, \beta_2 = 0.99$

- Estimate first moment:

$$\boxed{v_j^{(t)} = \rho_1 v_j^{(t-1)} + (1-\rho_1) g_j^{(t)}}$$

- Estimate second moment:

$$\boxed{r_j^{(t)} = \rho_2 r_j^{(t-1)} + (1-\rho_2) {g_j^{(t)}}^2}$$

- Update parameters:

$$\boxed{W_j^{(t+1)} = W_j^{(t)} - \frac{\epsilon}{\delta + \sqrt{r_j^{(t)}}} v_j^{(t)}}$$

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.

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

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

ρ to the power of t

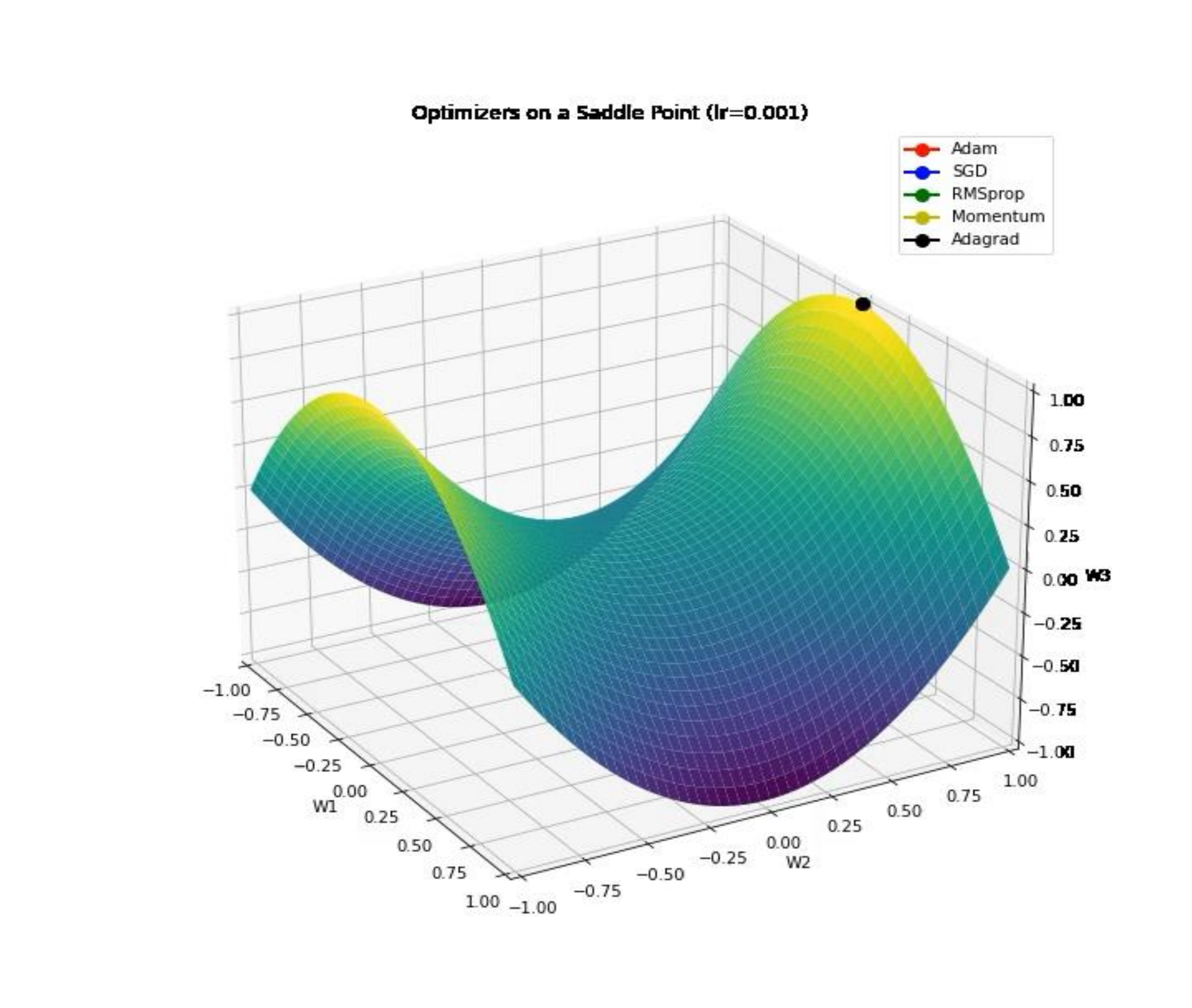
Where t is the number of the current iteration.

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. See notes for an explanation.

See additional notes on how to derive these corrections!

Saddle Point Revisited

Now, let's revisit our saddle point problem and see how different optimizers perform on a saddle point



Thank you