ECE 884 Deep Learning

Lecture 7: Regularization and Training

02/09/2021

Review of last lecture

- Loss Function
  - What it is, what is it for?
  - Example: cross-entropy loss function

Today’s lecture

- Regularization
  - What it is, what is it for?
  - Examples
- Training

Problem of Loss Function

Loss functions encourage good performance on training data
but we really care about test data

Credit: Justin

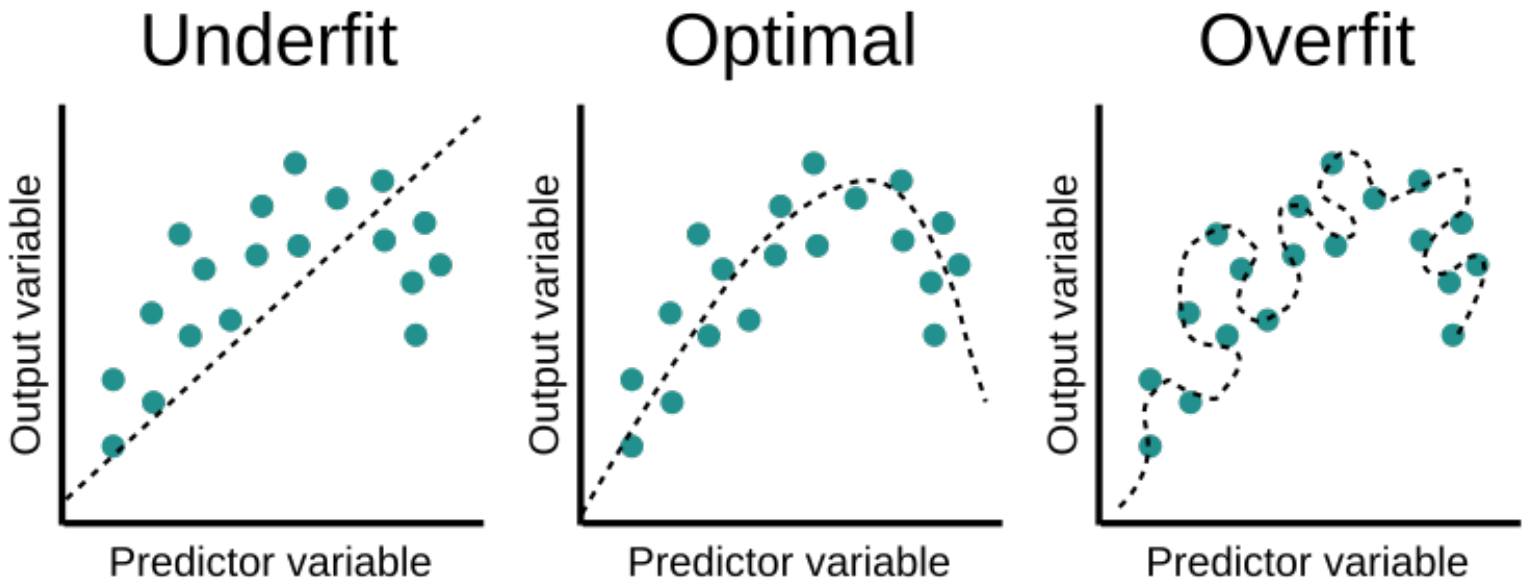
Overfitting

A model is **overfit** when it performs too well on the training data, but has poor performance for unseen data

Credit: Justin

Overfitting

A model is **overfit** when it performs too well on the training data, but has poor performance for unseen data



Regularization: Beyond Training Error

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}$$

Data loss: Model predictions should match training data

Credit: Justin

Regularization: Beyond Training Error

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

λ is a hyperparameter giving regularization strength

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Credit: Justin

Example Regularization

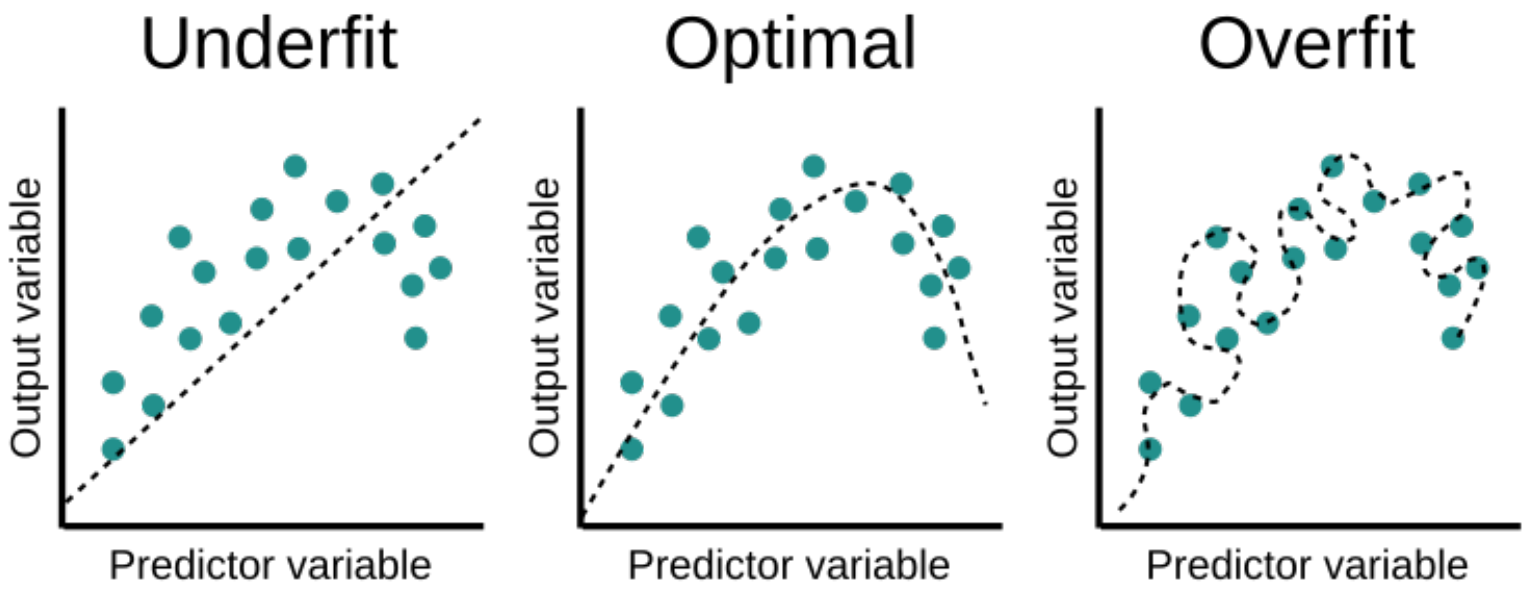
L2 regularization: $R(W) = \sum_{k,l} W_{k.l}^2$

L1 regularization: $R(W) = \sum_{k,l} |W_{k,l}|$

Credit: Justin

Regularization Effect#1

Select simpler model with lower order



Regularization Effect#2

Select model with more features

L2 Regularization

$$R(W) = \sum_{k,l} W_{k.l}^2$$

$$x = [1, 1, 1, 1]$$
$$w_1 = [1, 0, 0, 0]$$
$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1^T x = w_2^T x = 1$$

Same predictions, so data loss will always be the same

Credit: Justin

Regularization Effect#2

Select model with more features

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

L2 Regularization

$$R(W) = \sum_{k,l} W_{k.l}^2$$

L2 regularization prefers weights to be "spread out"

$$w_1^T x = w_2^T x = 1$$

Same predictions, so data loss will always be the same

Credit: Justin

Summary

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

Loss function consists of **data loss** to fit the training data and **regularization** to prevent overfitting

Credit: Justin

Training

What is the purpose of training?

Find the model weights W that achieve the lowest total loss based on the training dataset

Training

What is the purpose of training?

Training == Loss Optimization Problem

$$w^* = \arg\min_{w} L(w)$$

Training

What is the purpose of training?

Find the model weights W that achieve the lowest total loss based on the training dataset

$$w^* = \arg\min_{w} L(w)$$

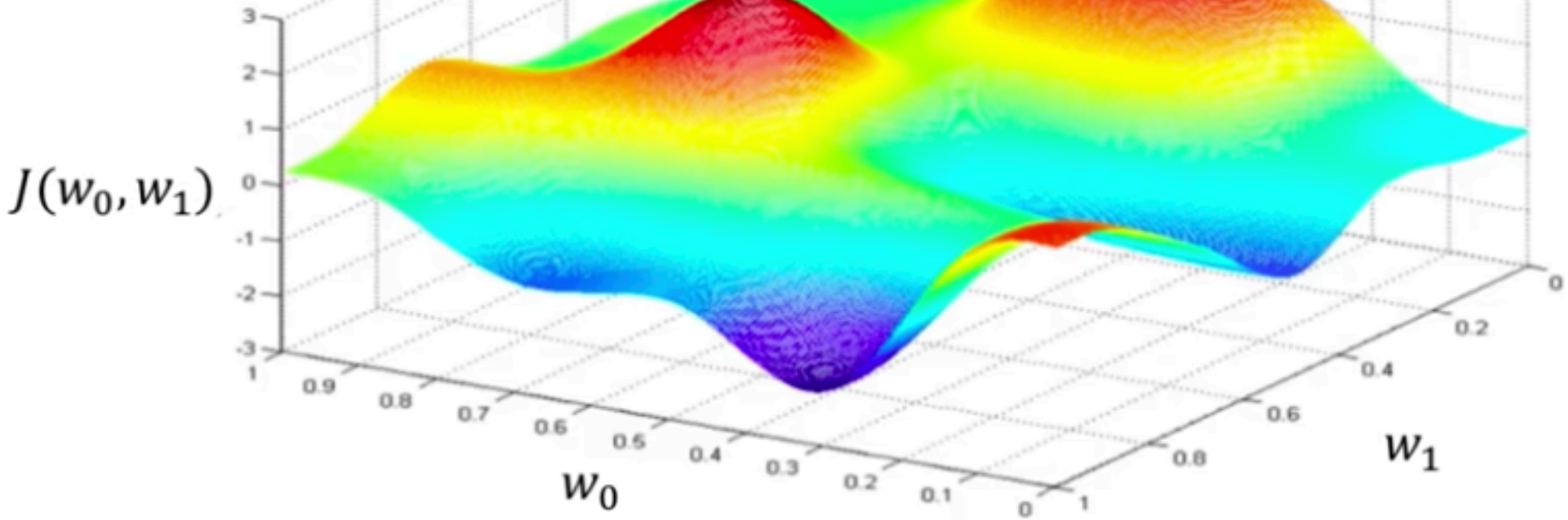
Remember:

$$\boldsymbol{W} = \{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \dots\}$$

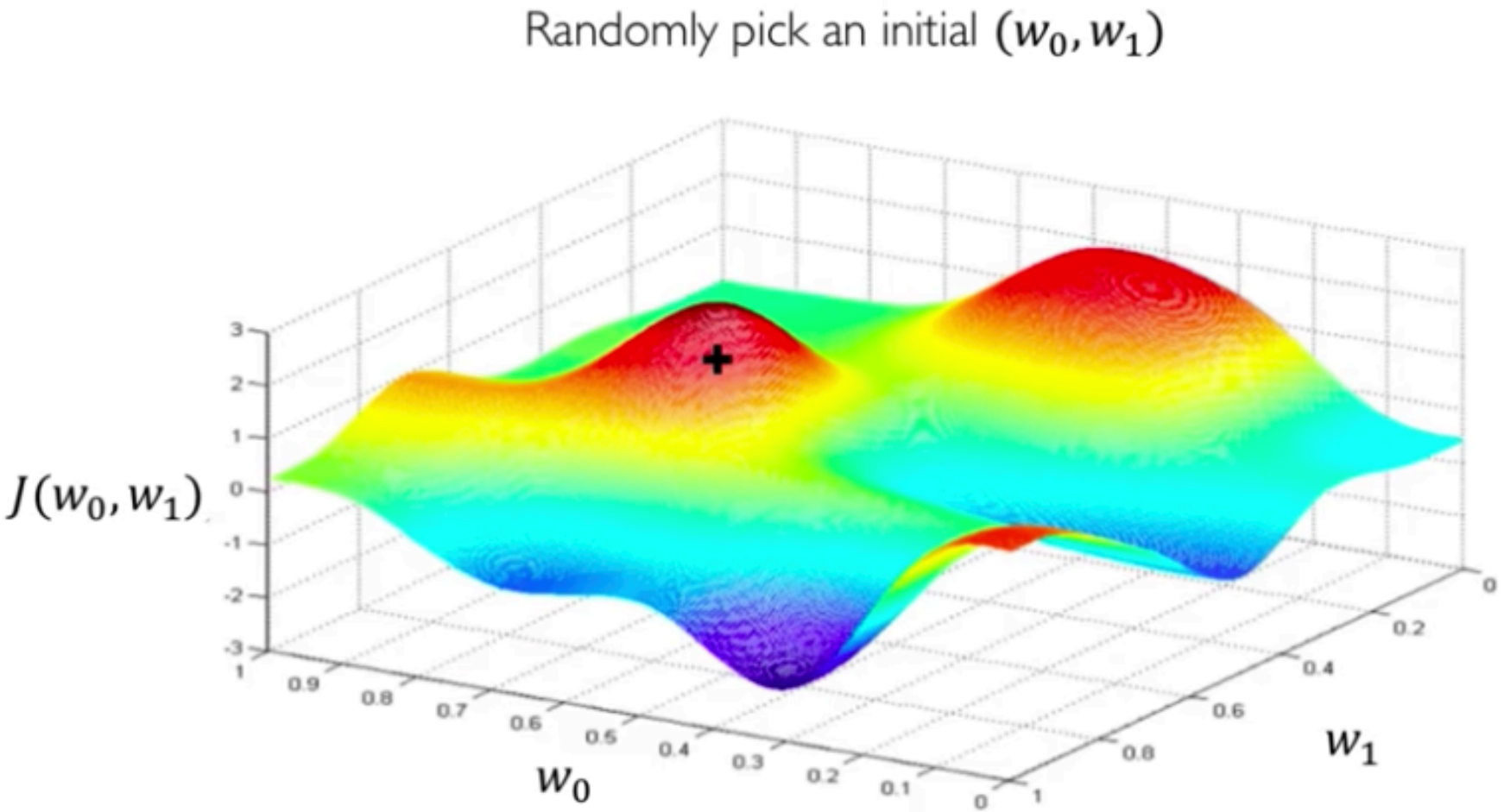
Optimization Landscape

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} J(\boldsymbol{W})$$

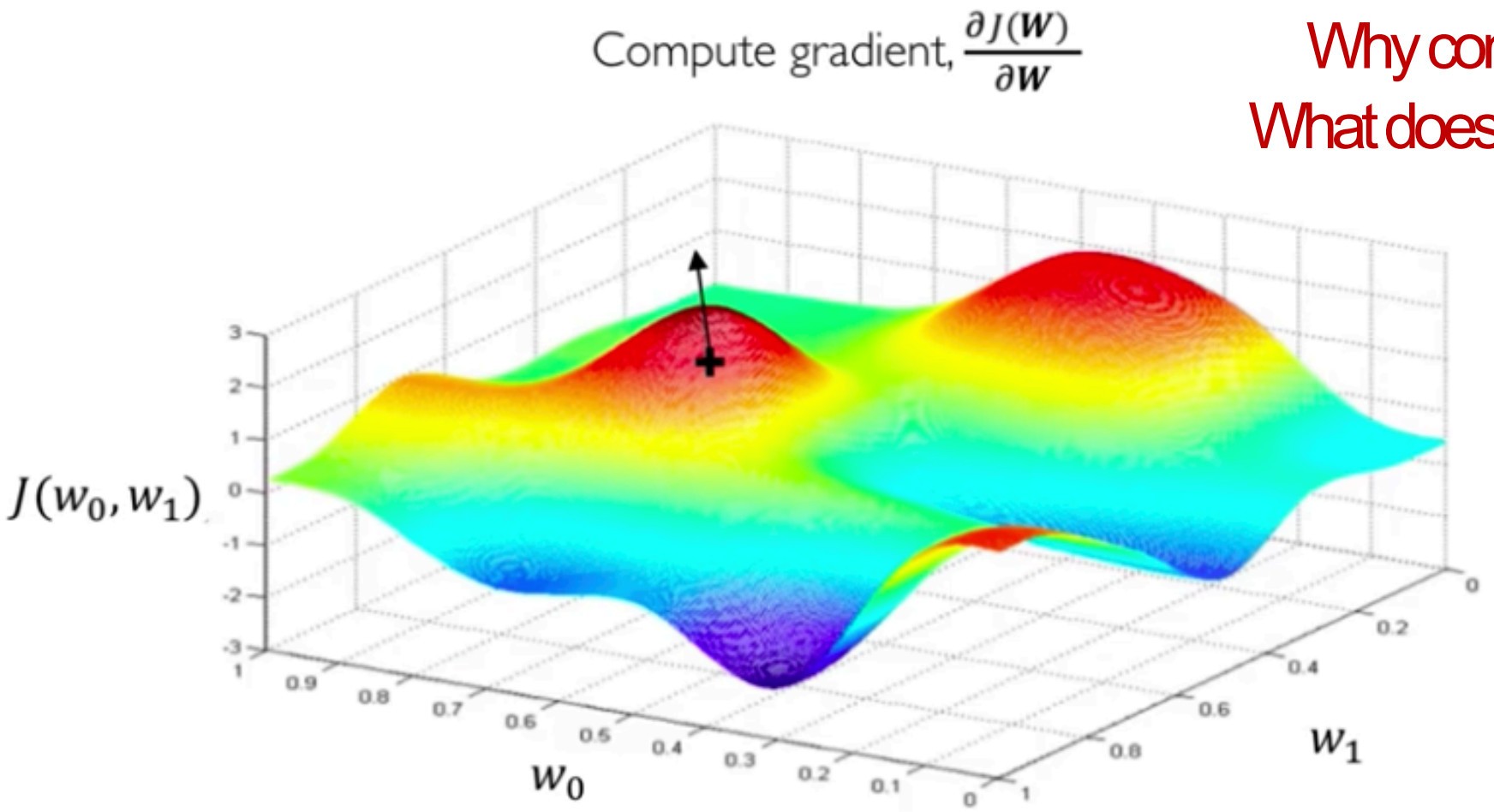
Remember:
Our loss is a function of the network weights!



Optimization Method: Gradient Decent



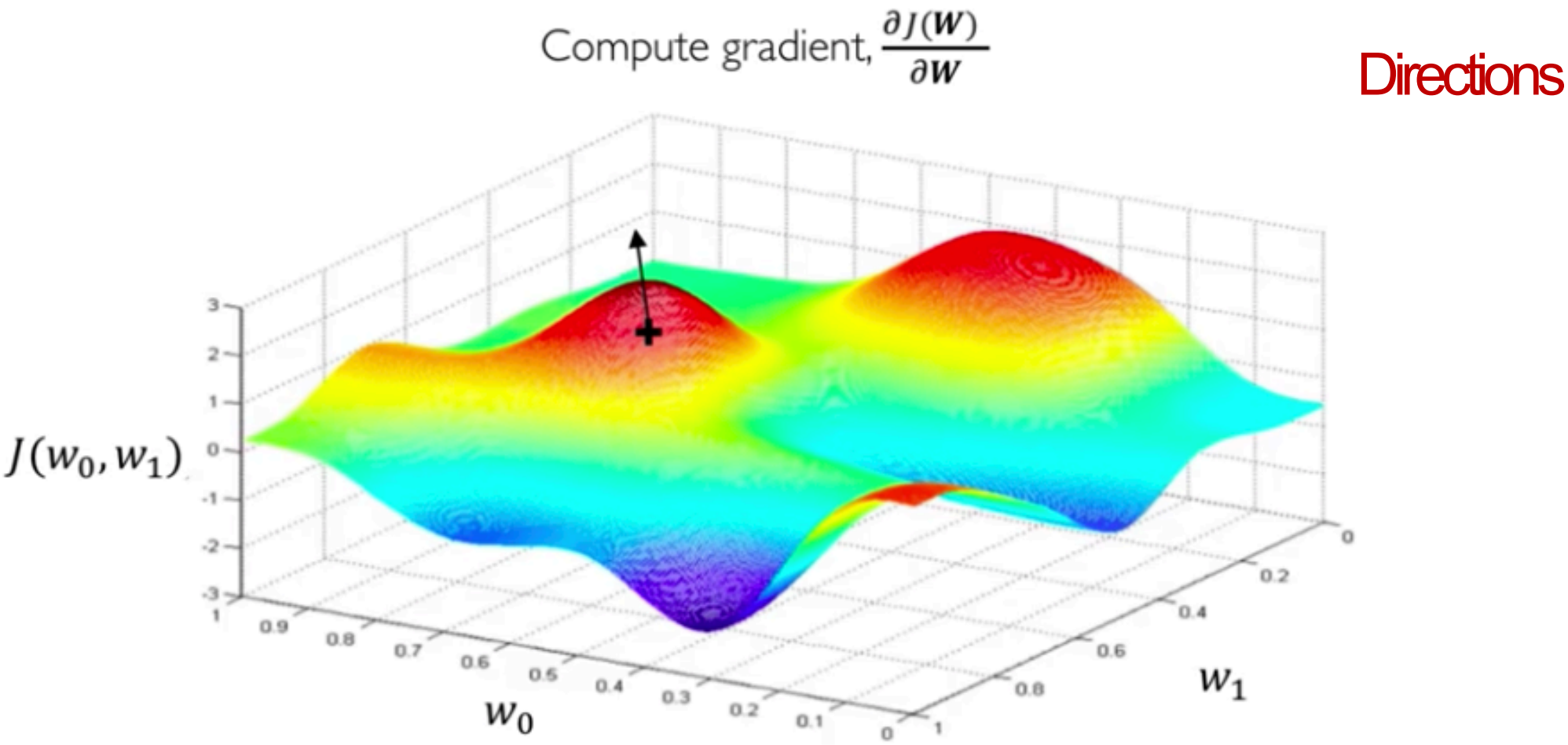
Optimization Method: Gradient Decent



Why computing gradient?

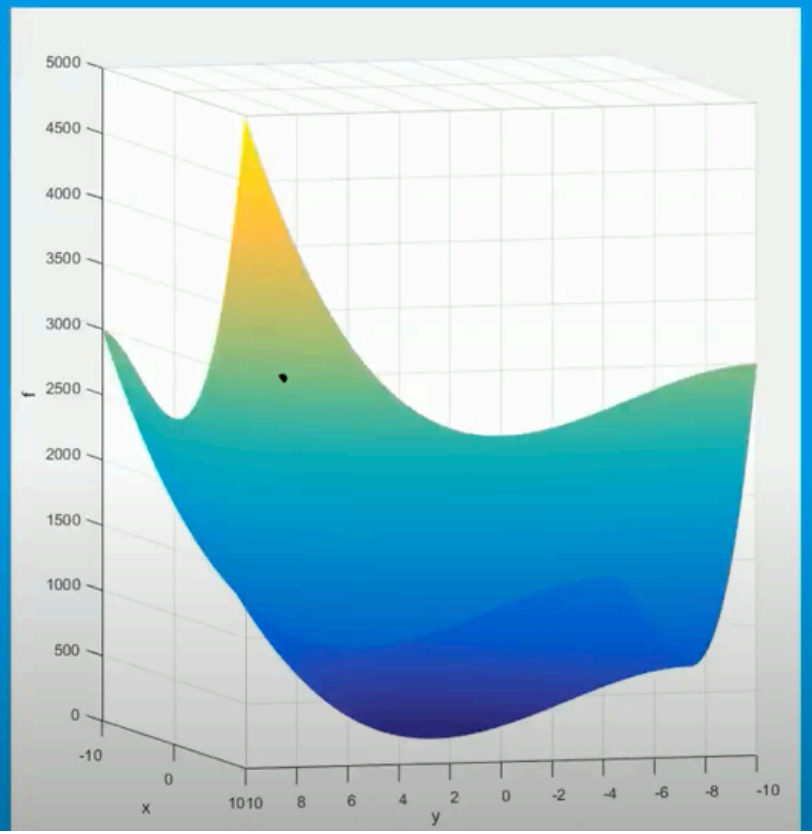
What does gradient represent?

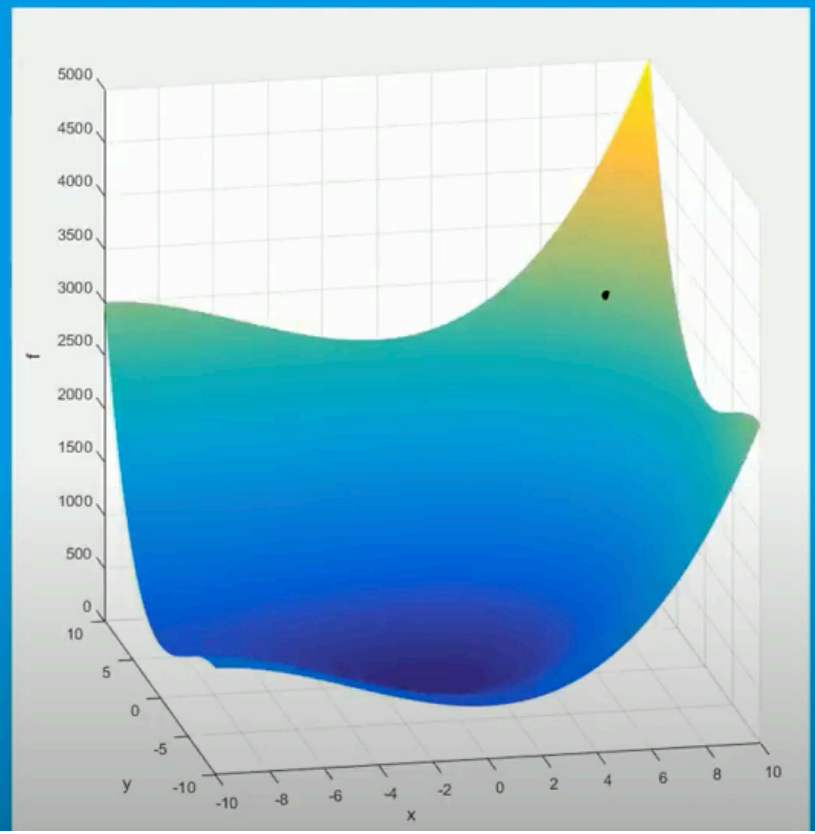
Optimization Method: Gradient Decent



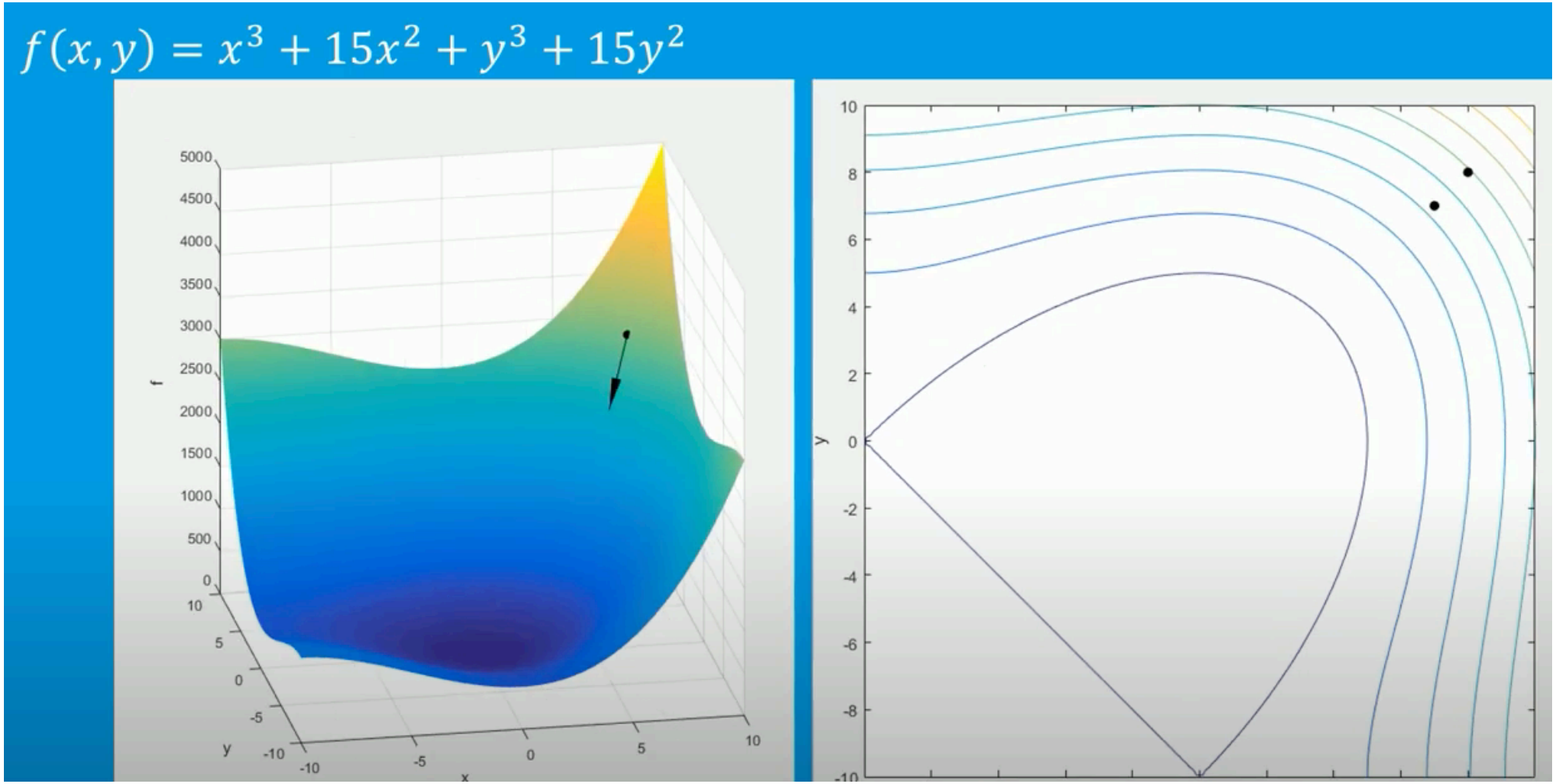
Optimization Method: Gradient Decent

$$f(x,y) = x^3 + 15x^2 + y^3 + 15y^2$$

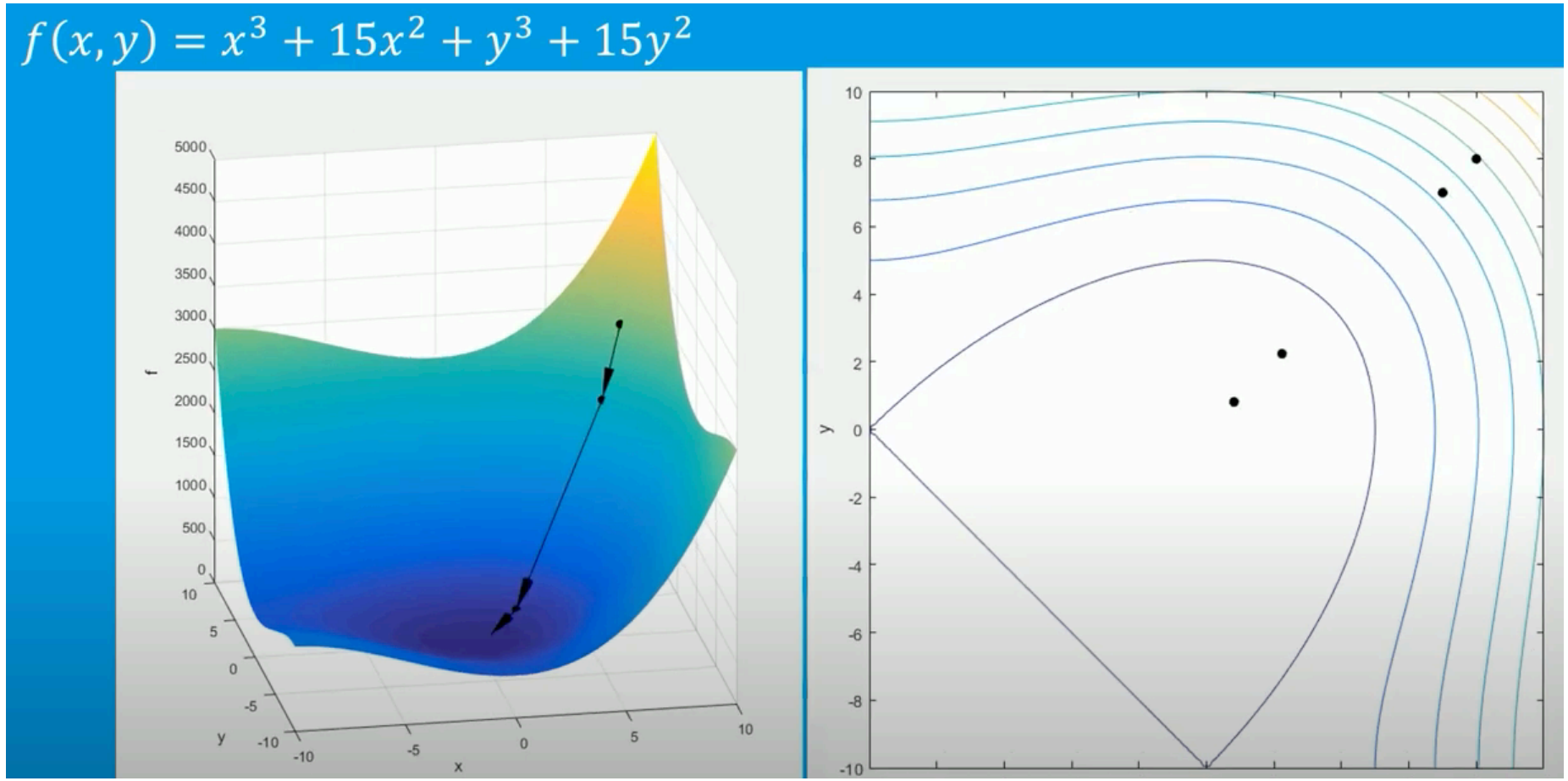




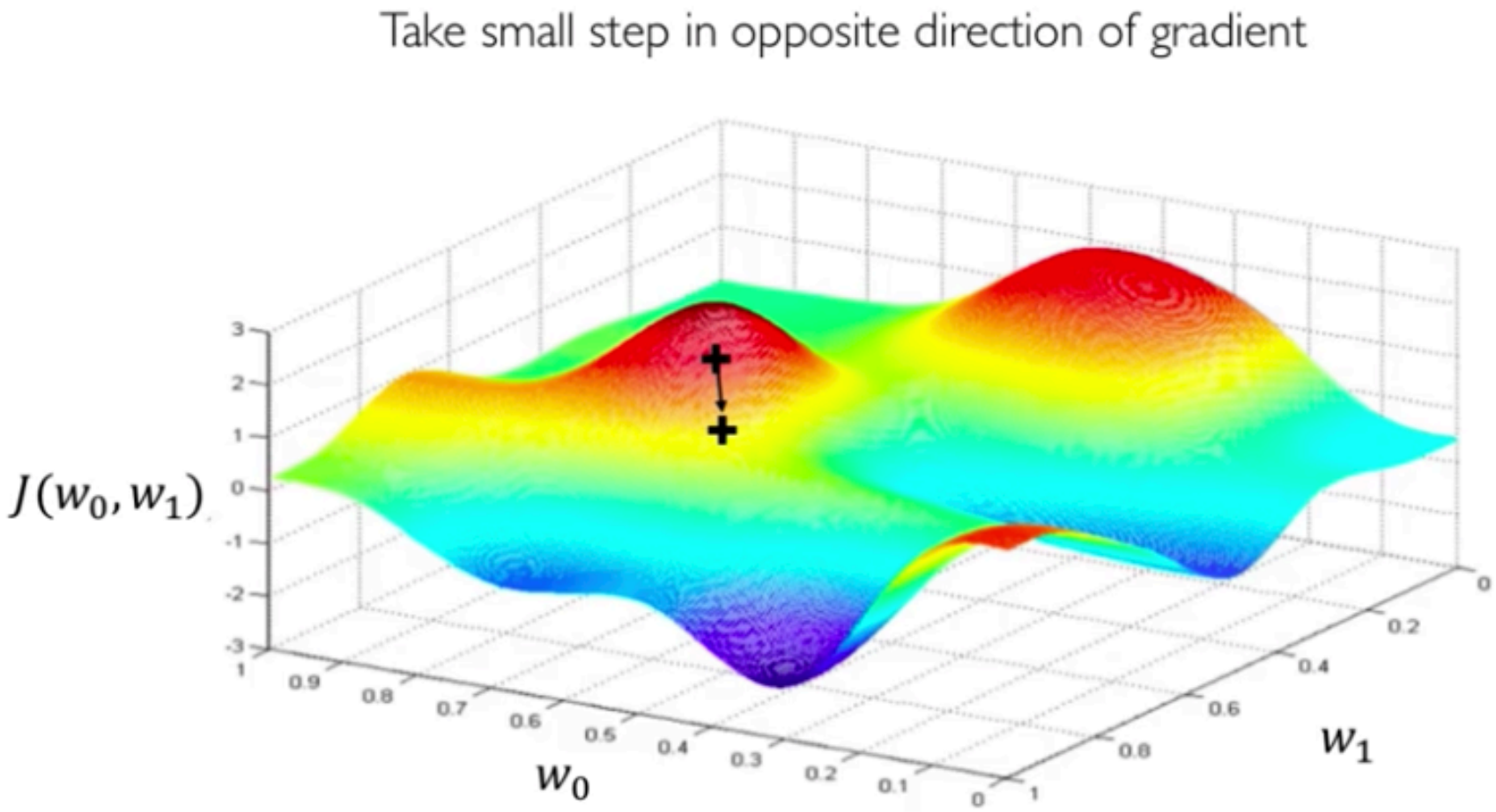
Optimization Method: Gradient Decent



Optimization Method: Gradient Decent

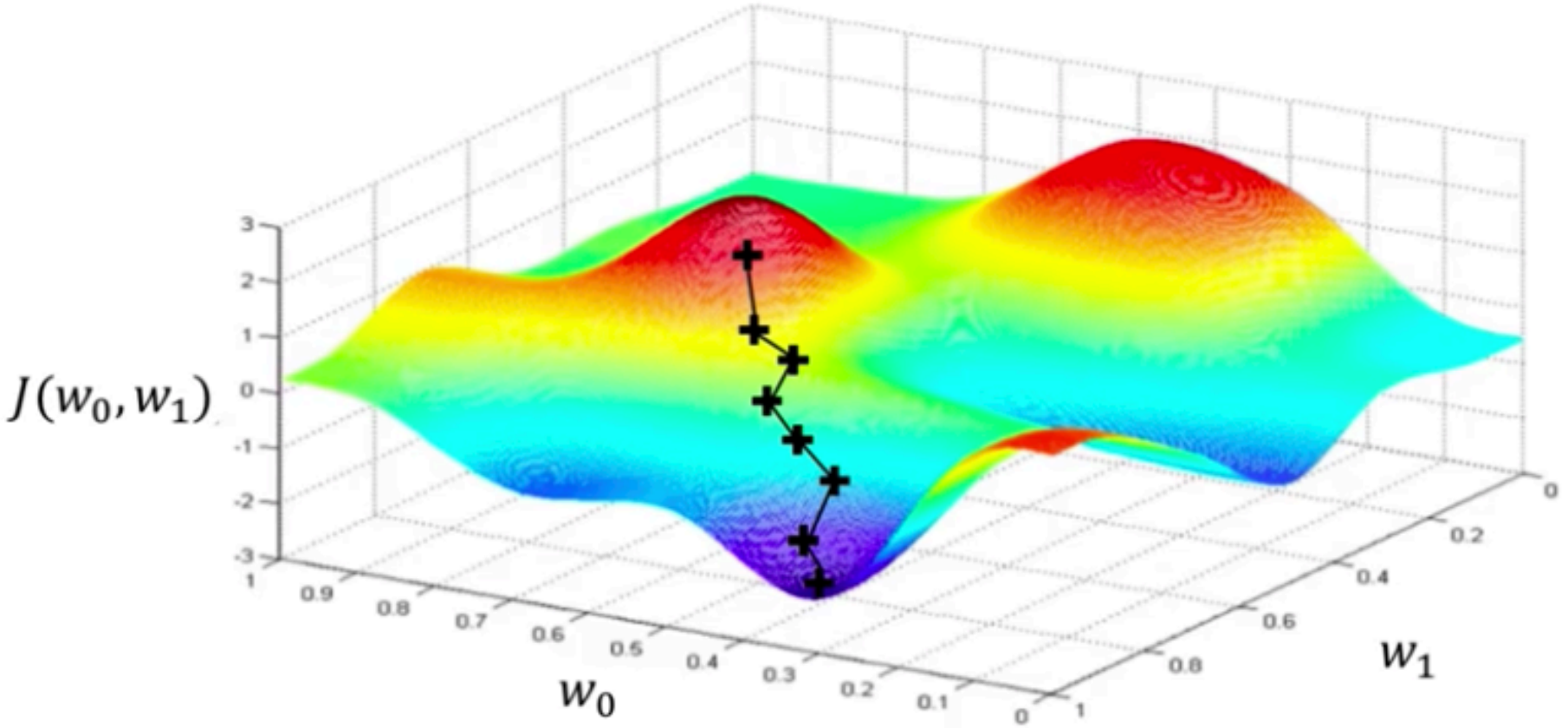


Optimization Method: Gradient Decent



Optimization Method: Gradient Decent

Repeat until convergence



Optimization Method: Gradient Decent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Batch Gradient Descent

Problem: This is very expensive when N is large!

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

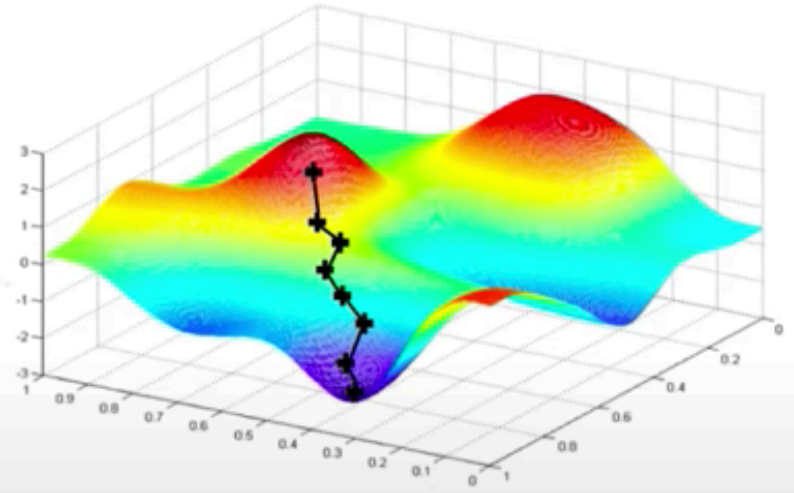
Stochastic Gradient Descent (SGD)

Single point to calculate GD

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point i
4. Compute gradient, $\frac{\partial J_i(W)}{\partial W}$
5. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
6. Return weights

Easy to compute but **very noisy** (stochastic)!



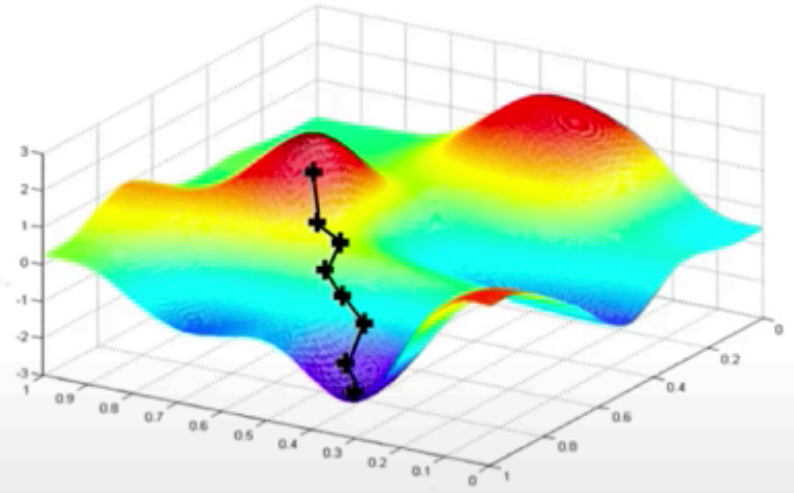
Stochastic Gradient Descent (SGD)

Problem: calculated gradient decent is noisy!

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point i
4. Compute gradient, $\frac{\partial J_i(W)}{\partial W}$
5. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
6. Return weights

Easy to compute but **very noisy** (stochastic)!

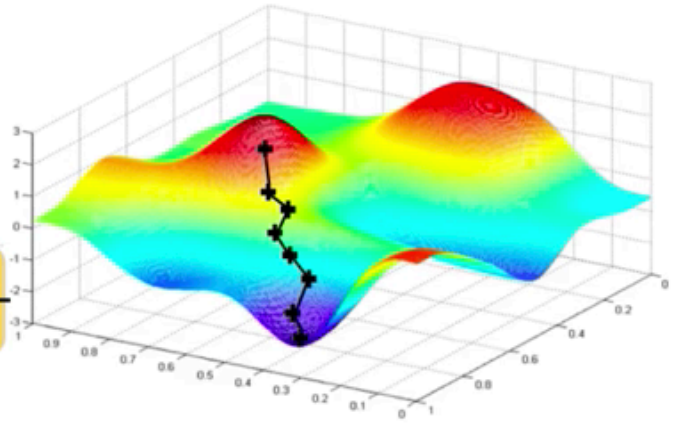


Stochastic Gradient Descent (SGD)

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(W)}{\partial W} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(W)}{\partial W}$
5. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
6. Return weights

Fast to compute and a much better estimate of the true gradient!



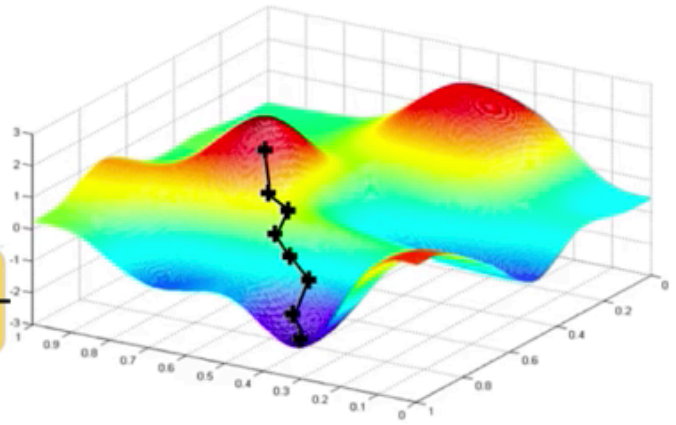
Approximate sum using a **minibatch** of 32 / 64 / 128 examples

Hyperparameters of Stochastic Gradient Descent (SGD)

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(W)}{\partial W} = \frac{1}{B}\sum_{k=1}^{B}\frac{\partial J_k(W)}{\partial W}$
5. Update weights, $W \leftarrow W - \eta\frac{\partial J(W)}{\partial W}$
6. Return weights

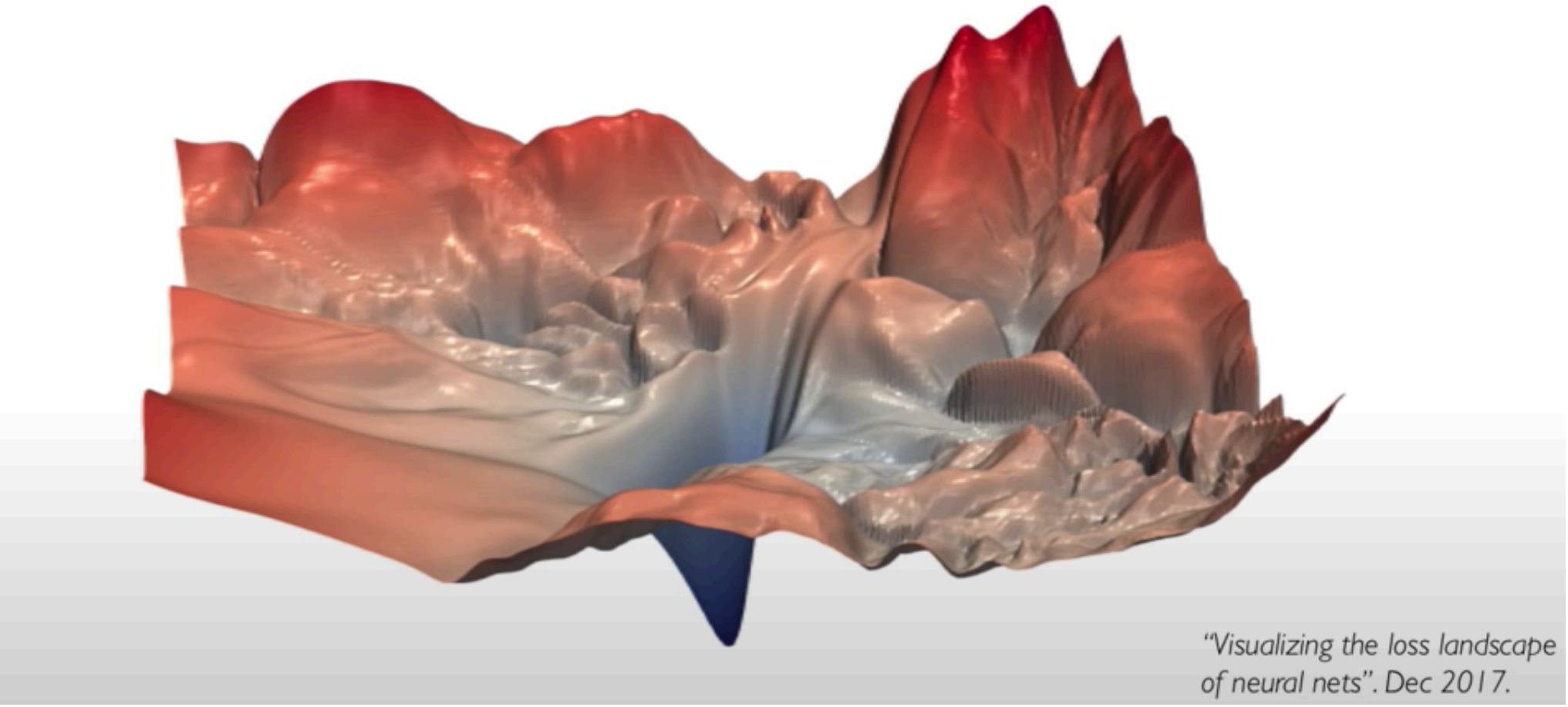
Fast to compute and a much better estimate of the true gradient!



Hyperparameters:

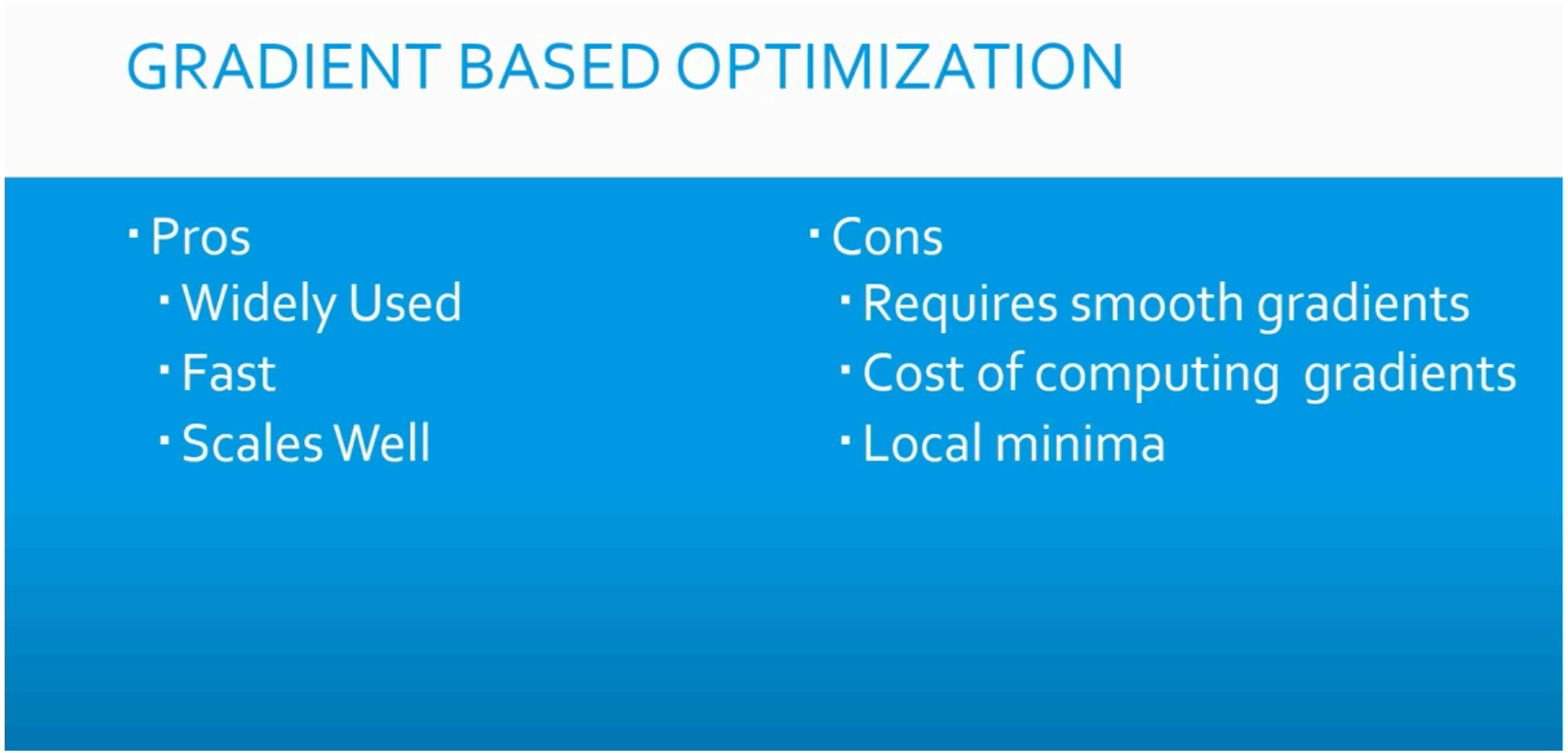
- Weight initialization
- Number of steps
- Learning rate
- Batch size
- Data sampling

Initialization Matters a lot!



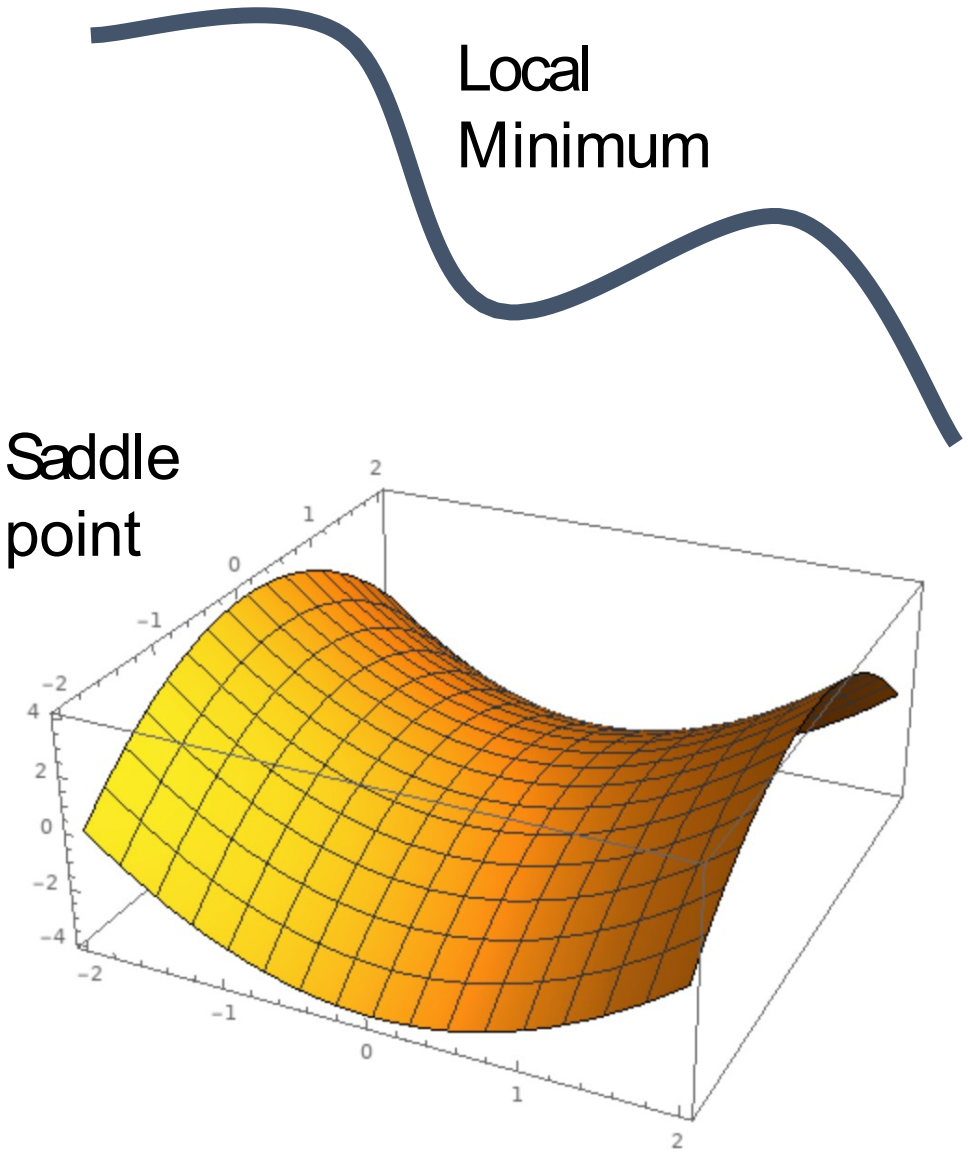
"Visualizing the loss landscape of neural nets". Dec 2017.

Pros and Cons



Problem#1 with SGD

What if the loss function has a **local minimum** or **saddle point**?

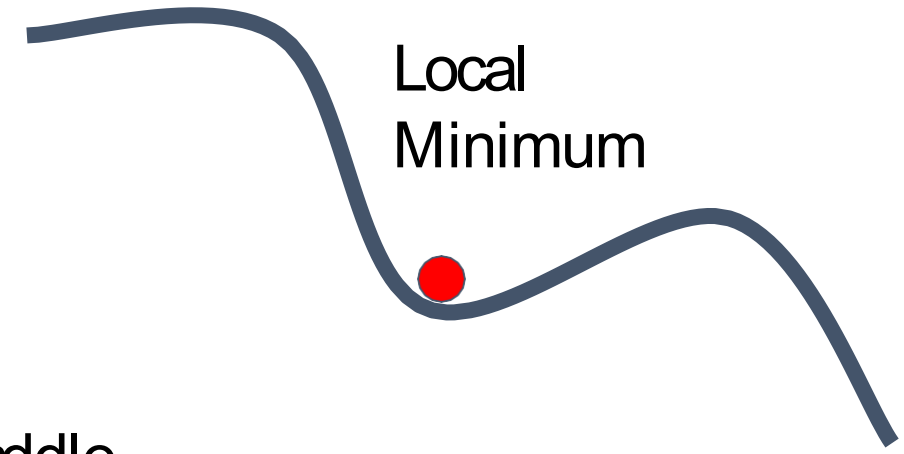


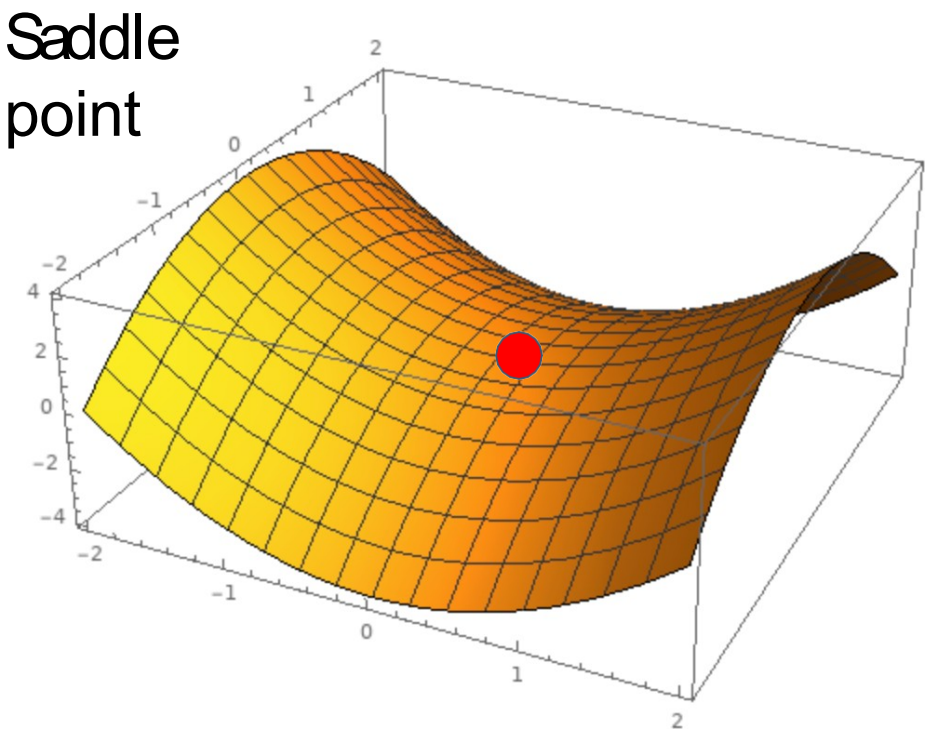
Credit: Justin

Problem#1 with SGD

What if the loss function has a **local minimum** or **saddle point**?

Zero gradient, gradient descent gets stuck





Credit: Justin

SGD+Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

V is velocity as a moving average of gradients.

Credit: Justin

SGD+Momentum

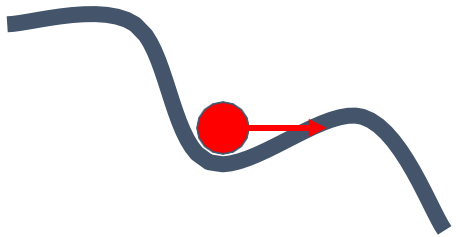
SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

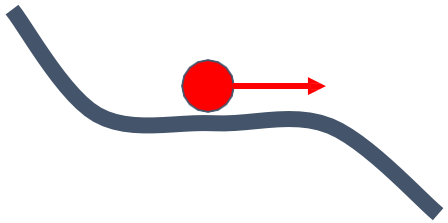
SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

Local Minima

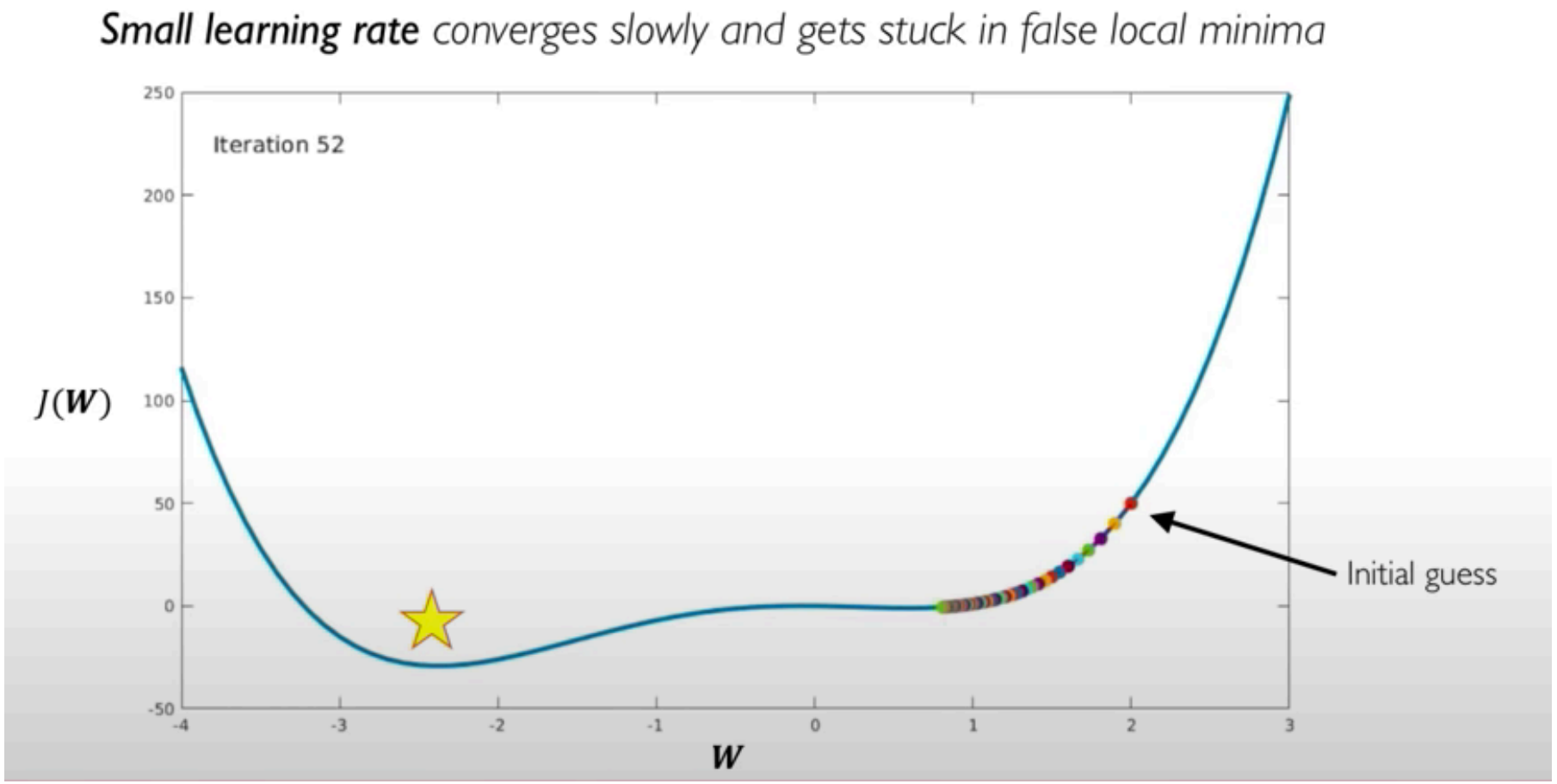


Saddle points

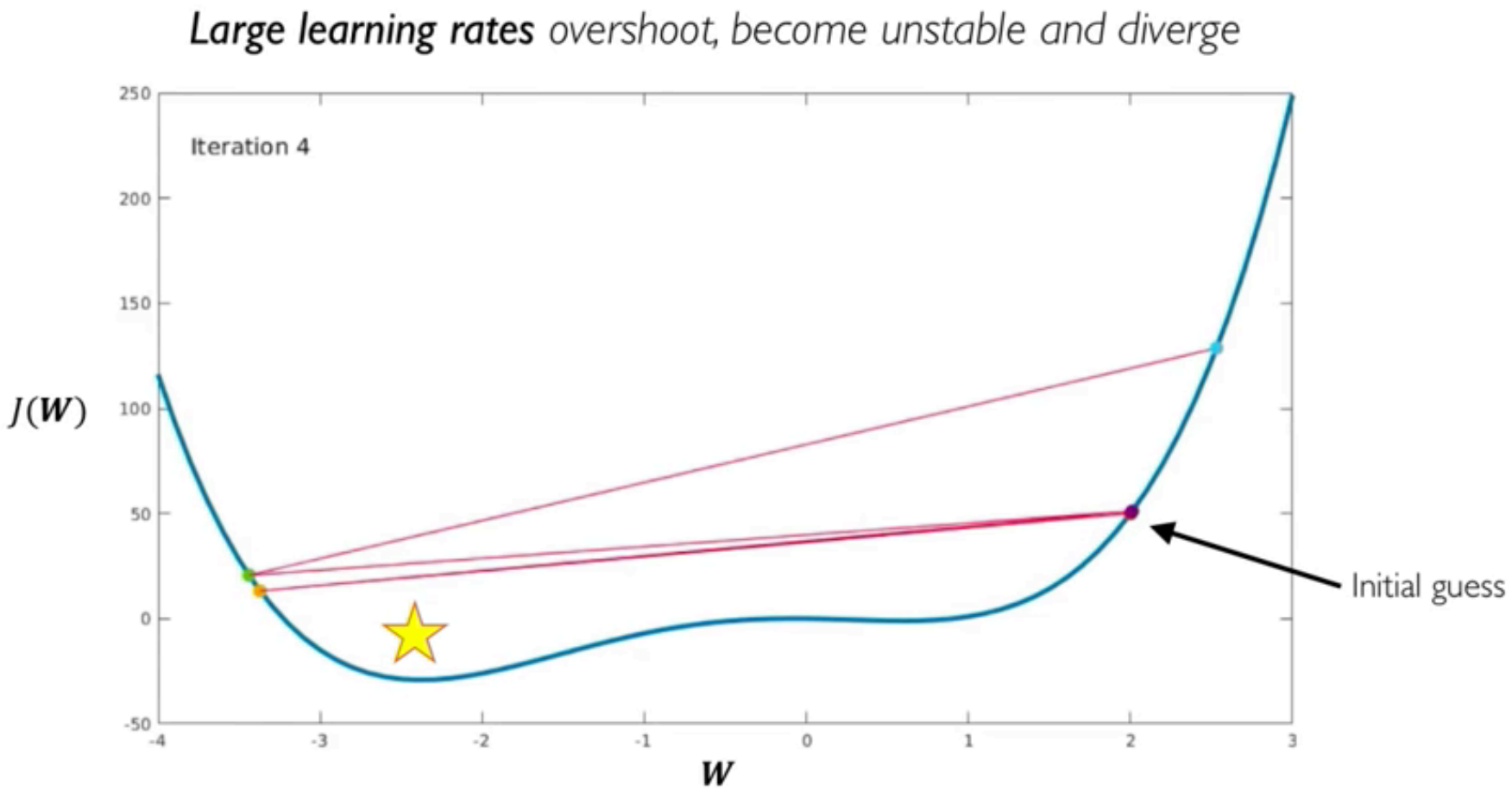


Credit: Justin

Problem#2 with SGD:Learning Rate

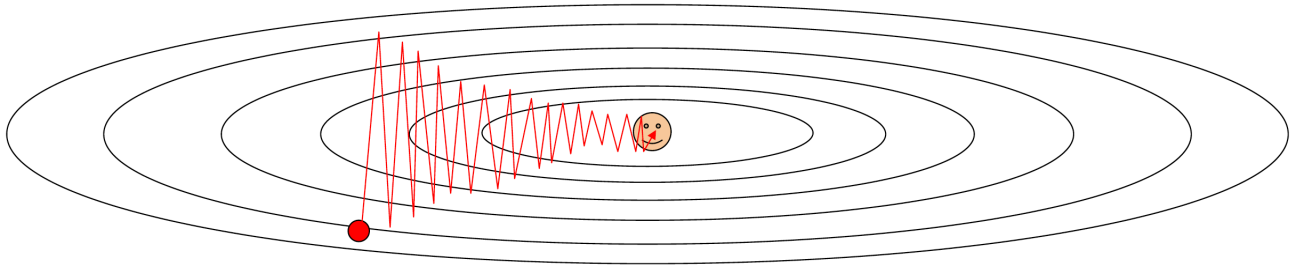


Problem#2 with SGD:Learning Rate



Optimizers with Adaptive Learning Rate

- RMSProp
- AdaGrad



- Adam == Momentum + RMSProp

Per-parameter learning rate

- AdamW

Learning rate decay

In practice:

- **Adam / AdamW** is a good default choice in many cases **SGD+Momentum** can outperform Adam but may require more tuning

Any Question ?