

# OPTIMIZING NEURAL NETWORKS

# TODAY'S CLASS

## GOALS

1. Review gradient calculation
2. Mini-batch gradient descent
3. Cover initialization
4. Optimization strategies

# BACKPROP

## BACKWARD PASS

Using the results of the forward pass, apply the chain rule to compute the derivatives for each layer

Compute  $\frac{\partial l_D(\theta)}{\partial h^k}$

Then compute  $\frac{\partial l_D(\theta)}{\partial W^k}$  and  $\frac{\partial l_D(\theta)}{\partial h^{k-1}}$

Repeat till  $W^1$

$$\begin{aligned} h^0 &= x \\ h^1 &= f^1(h^0, W^1) \\ h^2 &= f^2(h^1, W^2) \\ &\vdots \quad \nearrow \\ h^i &= f^i(h^{i-1}, W^i) \\ &\vdots \quad \nearrow \\ h^k &= f^k(h^{k-1}, W^k) \\ &\quad \uparrow \quad \nearrow \\ &\quad \frac{\partial l_D(\theta)}{\partial h^k} \end{aligned}$$

# LAYERWISE GRADIENT

DENSE LAYERS

$$h^i, z^i \in \mathbb{R}^{m \times n_i}, h^{i-1} \in \mathbb{R}^{m \times n_{i-1}}, W^i \in \mathbb{R}^{n_i \times n_{i-1}}$$

$$\begin{aligned}\frac{\partial l(\theta)}{\partial W^i} &= \frac{\partial l(\theta)}{h^i} \frac{\partial h^i}{\partial W^i} \\ &= \left( \partial_{h^i} l(\theta) \odot \frac{\partial \sigma(z^i)}{\partial z^i} \right)^T h^{i-1}\end{aligned}$$

$$\begin{aligned}\frac{\partial l(\theta)}{\partial h^{i-1}} &= \frac{\partial l(\theta)}{h^i} \frac{\partial h^i}{\partial h^{i-1}} \\ &= \left( \partial_{h^i} l(\theta) \odot \frac{\partial \sigma(z^i)}{\partial z^i} \right) W^i\end{aligned}$$

# BATCH GRADIENT DESCENT

## OPTIMIZING NNS

$$\theta^{k+1} \leftarrow \theta^k - \eta \nabla l_D(\theta)$$

“batch gradient descent” – gradient is computed on the full training dataset

- Very slow for lots of data
- Stochastic gradient descent is often faster
- The gradient of a single data point might be too low quality to be that efficient.
- Single data point may under utilize hardware
  - Parallel hardware (multi-core CPUs, GPUs, etc) can process more data at a time

# MINI-BATCH GRADIENT DESCENT

## OPTIMIZING NNS IN PRACTICE

Sample a mini-batch of  $q \ll m$  data points,  $D_q$

Perform a stochastic gradient descent step on  $l_{D_q}(\theta)$

$$\theta^{k+1} = \theta^k - \eta \nabla l_{D_q}(\theta^k)$$

Sample new  $D_q$  and repeat

# MINI-BATCH GRADIENT DESCENT

## OPTIMIZING NNS IN PRACTICE

In practice, we don't randomly sample data

The first mini-batch is the first set of  $q$  data points, i.e.,  $X^1 = X_{1:q,:}$ ,  $Y^1 = y_{1:q,:}$ .

The second is the second set of  $q$  data points, and so on, i.e.,  $X^2 = X_{q+1:2q,:}$ ,  $Y^2 = y_{q+1:2q,:}$

Repeat until all the data is processed. Then start over.

We call processing the whole dataset once an *epoch*.

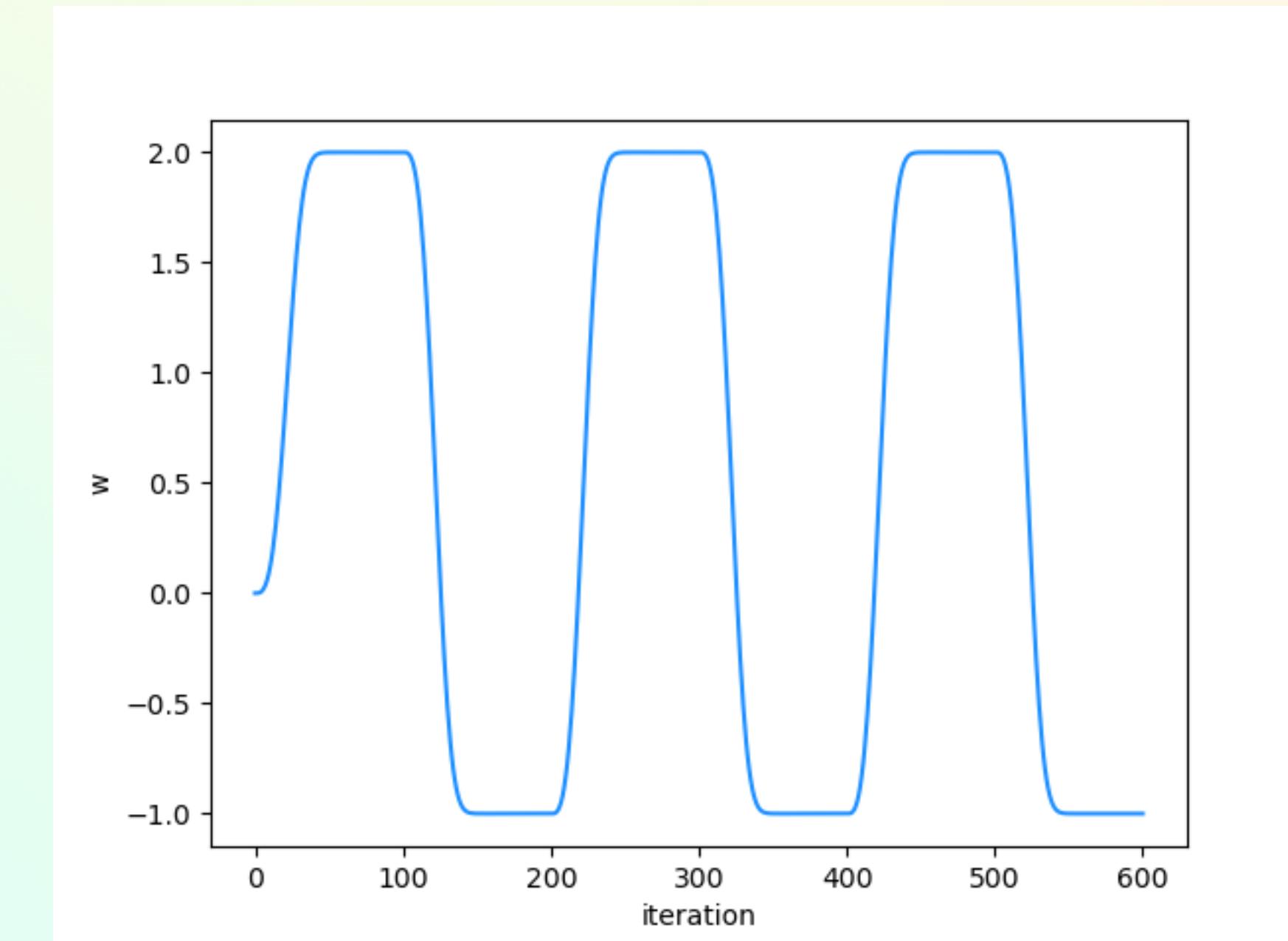
A mini-batch is often called a batch, and we do not mean performing batch gradient descent.

# MINI-BATCH GRADIENT DESCENT

## OPTIMIZING NNS IN PRACTICE

The dataset might be stored in a correlated (nonrandom) way.

$$X = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ 1 \\ 2 \\ \vdots \end{bmatrix} \quad Y = \begin{bmatrix} 2 \\ 4 \\ \vdots \\ -1 \\ -2 \\ \vdots \end{bmatrix}$$



Processing the data sequentially on this data set would first optimized towards  $f_*(x) = 2x$  then towards  $f_*(x) = -x$

The model oscillates between being good at only one of these functions.

# MINI-BATCH GRADIENT DESCENT

## OPTIMIZING NNS IN PRACTICE

We want to find the model that performs well across all the data (averaged across data points)

The correlations in the data prevent us from getting this behavior

In SGD/Mini-batch SGD, we assume the datasets are each independent and identically distributed (i.i.d.)

Solution: Randomize data order before using it.

# MINI-BATCH GRADIENT DESCENT

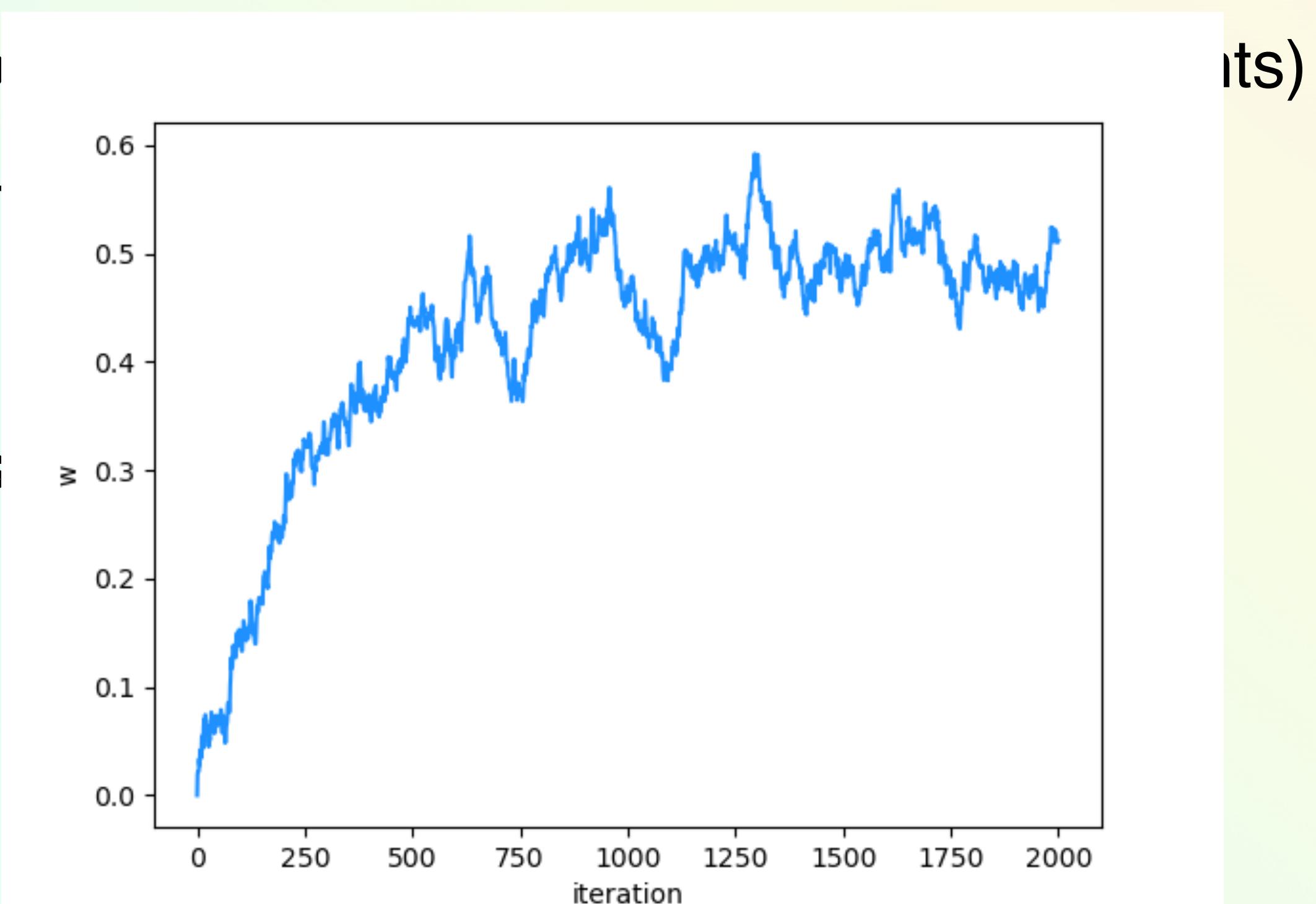
## OPTIMIZING NNS IN PRACTICE

We want to find the model that performs well across all data points

The correlations in the data prevent us from getting there

In SGD/Mini-batch SGD, we assume the datasets are i.i.d. (independently and identically distributed)

Solution: Randomize data order before using it.



# MINI-BATCH GRADIENT DESCENT

## CHOOSING $q$

$q = 1$  – may not use all parallel hardware, gradients might be very noisy

$q = m$  – full batch gradient descent, usually too slow.

$q \in \{16, 32, 64, 128, 256\}$

- typical in practice
- Choose based on network size and computational resources

Increasing  $q$  increases the quality of the gradient estimate but has diminishing returns

SGD has a regularizing effect on the weights, where using small  $q$  can reduce overfitting.

# SGD AND FLAT OPTIMA

WHITEBOARD

# NORMALIZING DATA

MAKING GRADIENT DESCENT WORK WELL

If we get real-world data, it can look like anything

$$X = \begin{bmatrix} 1 & 100 & 0.01 & 1 \\ -0.9 & 101 & 0.02 & 2 \\ -1.1 & 102 & 0.03 & 3 \\ 0.2 & 150 & 0.05 & 4 \end{bmatrix}, Y = \begin{bmatrix} 1000 \\ 998 \\ 1010 \\ 800 \end{bmatrix}$$

# NORMALIZING DATA

## MAKING GRADIENT DESCENT WORK WELL

If we get real-world data, it can look like anything

$$X = \begin{bmatrix} 1 & 100 & 0.01 & 1 \\ -0.9 & 101 & 0.02 & 2 \\ -1.1 & 102 & 0.03 & 3 \\ 0.2 & 150 & 0.05 & 4 \end{bmatrix}, Y = \begin{bmatrix} 1000 \\ 998 \\ 1010 \\ 800 \end{bmatrix}$$

If we have a model  $f(x, W, b) = xW^\top + b$

- If  $b = 0$  at the start, it has to slowly learn to get close to 1000
- The targets have large variations, then the prediction errors will also have large variations
- For  $w$  having the same scale for each feature, large features will contribute more to the prediction.

# NORMALIZING DATA

MAKING GRADIENT DESCENT WORK WELL

A key step in any machine-learning system is to normalize the data

- Each feature & target should be on the same scale and centered

Normalizations:

- Make each feature have a mean of 0 and a standard deviation of 1
- Map features to  $[-1, 1]$  or  $[0, 1]$
- For regression problems, it is best to normalize targets to mean 0 standard deviation 1

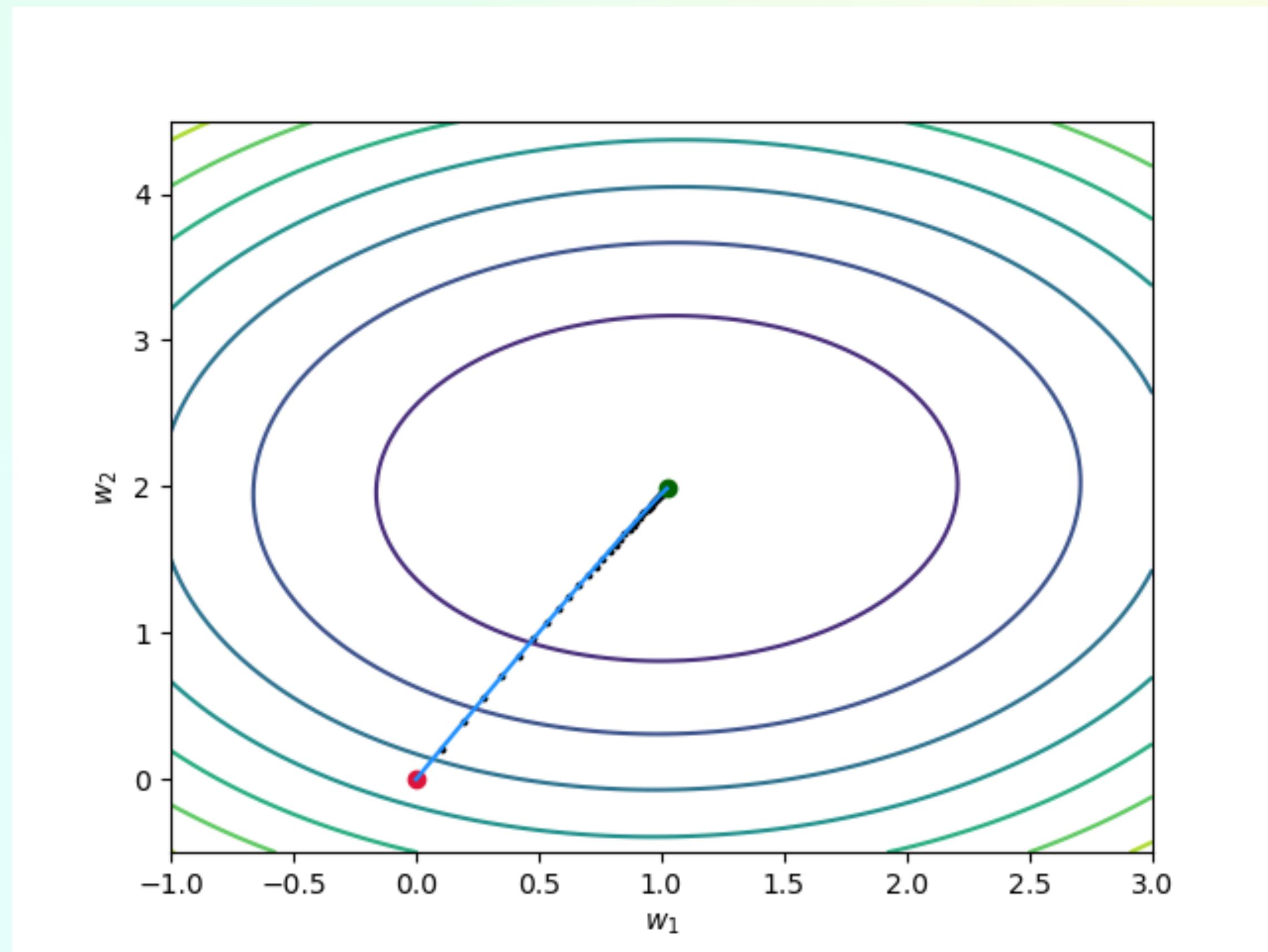
# NORMALIZING DATA

MAKING GRADIENT DESCENT WORK WELL

Normalized data:

$$x_1 \sim \mathcal{N}(0,1)$$

$$x_2 \sim \mathcal{N}(0,1)$$



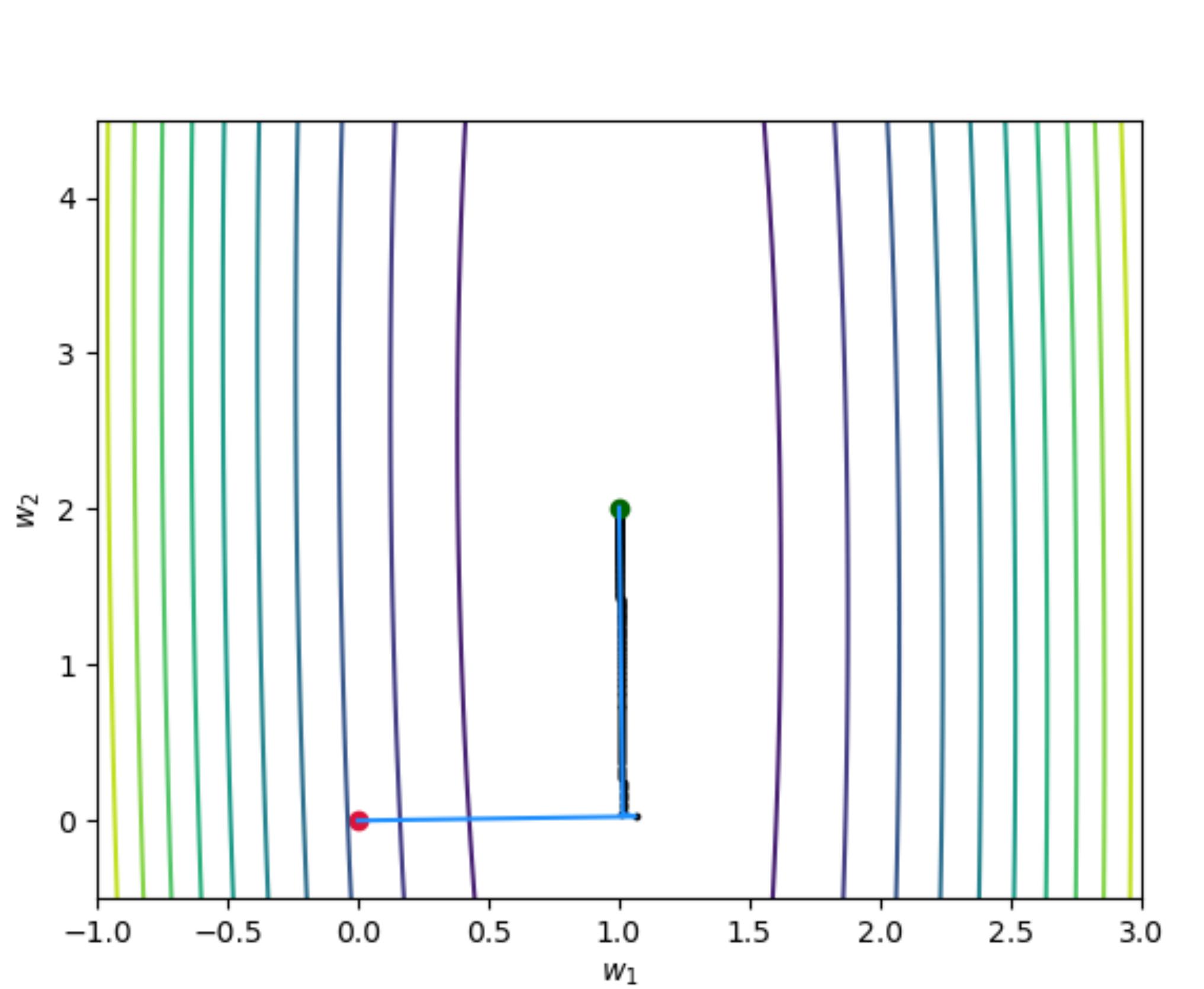
# NORMALIZING DATA

MAKING GRADIENT DESCENT WORK WELL

Not normalized data:

$$x_1 \sim \mathcal{N}(0, 10^2)$$

$$x_2 \sim \mathcal{N}(0, 1)$$



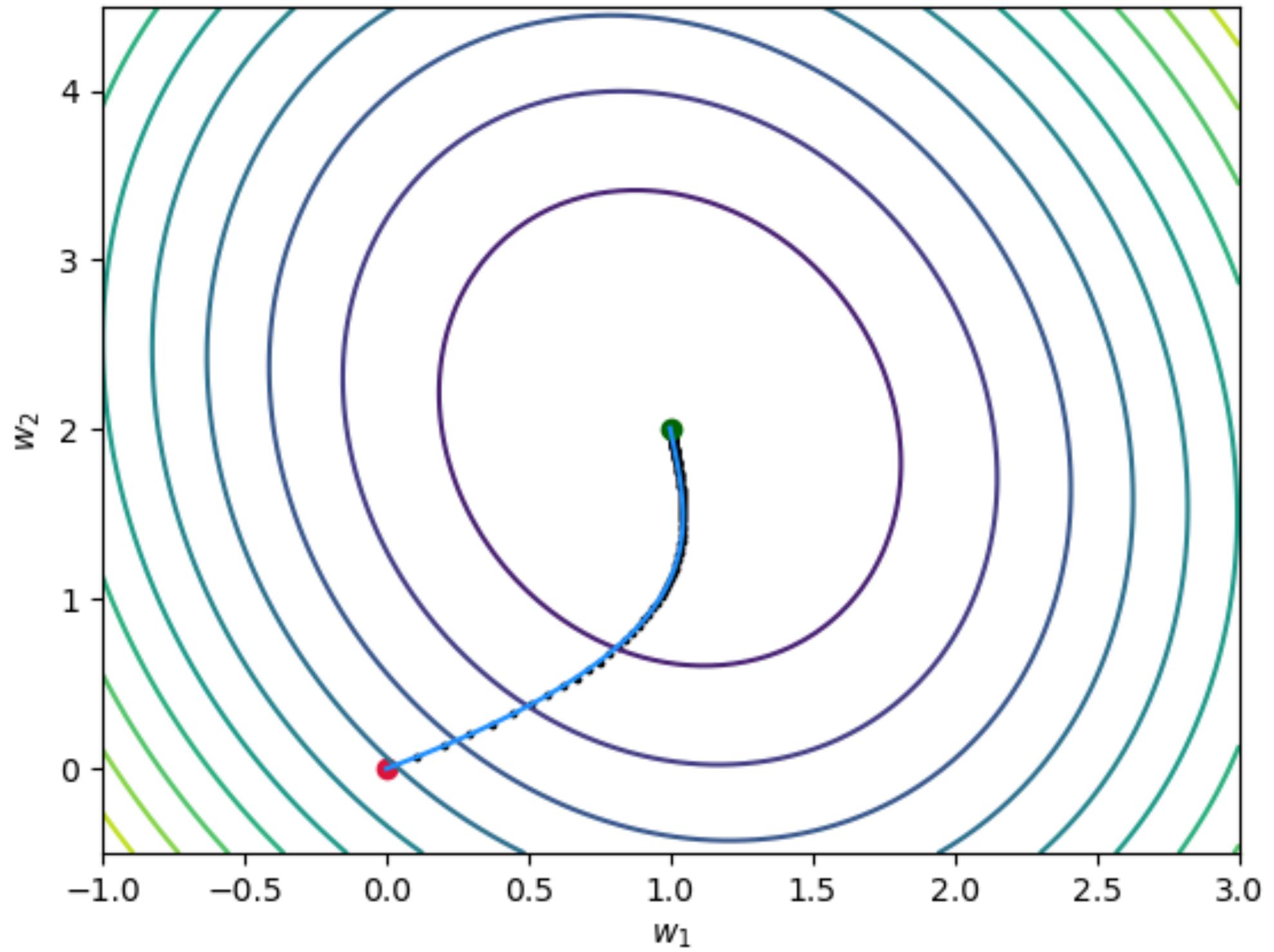
# NORMALIZING DATA

MAKING GRADIENT DESCENT WORK WELL

Not normalized data:

$$x_1 \sim \mathcal{N}(3, 1)$$

$$x_2 \sim \mathcal{N}(-2, 0.3^2)$$



# QUIZ

# INITIALIZING NEURAL NETWORKS

## STARTING WELL

We (the user) have to choose the initial weights  $\theta^0$ .

With linear, we initialized the weights with all zeros.

What if we do this in the neural network?

# QUIZ

# INITIALIZING NEURAL NETWORKS

## STARTING WELL

Zero initialization of weights

No derivative information to earlier layers:

$$\frac{\partial l(\theta)}{\partial h^{i-1}} = \left( \partial_{h^i} l(\theta) \odot \frac{\partial \sigma(z^i)}{\partial z^i} \right) W^i = 0$$

No information about  $h^0$  goes to later layers

$$z^1 = h^0 W^{1\top} = 0$$

The network can only learn the average target  $E[Y]$

# QUIZ

# INITIALIZING NEURAL NETWORKS

## CONSTANT INITIALIZATION

Initialize all weights with a constant  $c$

Derivatives are not zero; all hidden units will output the same value.

All hidden units in the layer will have the same output.

# INITIALIZING NEURAL NETWORKS

## CONSTANT INITIALIZATION

$$f_1^1(x_{1,.}, W^1) = \sigma(x_{1,.} W_{1,.}^1) = \sum_{k=1}^{n_0} x_{1,k} W_{1,k}^1 = \sum_{k=1}^{n_0} x_{1,k} c = c \sum_{k=1}^{n_0} x_{1,k}$$

$$f_2^1(x_{1,.}, W^1) = \sigma(x_{1,.} W_{2,.}^1) = \sum_{k=1}^{n_0} x_{1,k} W_{2,k}^1 = \sum_{k=1}^{n_0} x_{1,k} c = c \sum_{k=1}^{n_0} x_{1,k}$$

We want to make it so each hidden unit represents a different feature.

# INITIALIZING NEURAL NETWORKS

## CONSTANT INITIALIZATION

$$f_1^1(x_{1,.}, W^1) = \sigma(x_{1,.} W_{1,.}^1) = \sum_{k=1}^{n_0} x_{1,k} W_{1,k}^1 = \sum_{k=1}^{n_0} x_{1,k} c = c \sum_{k=1}^{n_0} x_{1,k}$$

$$f_2^1(x_{1,.}, W^1) = \sigma(x_{1,.} W_{2,.}^1) = \sum_{k=1}^{n_0} x_{1,k} W_{2,k}^1 = \sum_{k=1}^{n_0} x_{1,k} c = c \sum_{k=1}^{n_0} x_{1,k}$$

We want to make it so each hidden unit represents a different feature.

The derivatives will also be the same, e.g.,  $\frac{\partial l(\theta)}{\partial W_{j,k}^i} = \frac{\partial l(\theta)}{\partial W_{j',k}^i}$  and

thus cannot learn to represent different features

# INITIALIZING NEURAL NETWORKS

RANDOM VECTORS

$$W_{j,k}^i \sim \mathcal{N}(0, s^2) \text{ or } W_{j,k}^i \sim U(-s, s)$$

# INITIALIZING NEURAL NETWORKS

## RANDOM VECTORS

$$W_{j,k}^i \sim \mathcal{N}(0, s^2) \text{ or } W_{j,k}^i \sim U(-s, s)$$

For two random vectors  $X, Y \in \mathbb{R}^n$

As  $n \rightarrow \infty$ ,  $\frac{X^\top}{\|X\|} \frac{Y}{\|Y\|} \rightarrow 0$ , i.e., the two vectors are orthogonal (perpendicular)

For large  $n$ , the two vectors  $X$  and  $Y$  will point in different directions

- $W_{j,\cdot}^i$  and  $W_{j',\cdot}^i$  will make it so  $h_{\cdot,j}^i$  and  $h_{\cdot,j'}^i$  represent different aspects of the data

# INITIALIZING NEURAL NETWORKS

HOW LARGE SHOULD THE WEIGHTS BE?

What should the scale  $s$  be for the weight initialization?

Linear (scalar) network:  $f(\theta) = x(W^k \dots W^3 W^2 W^1)^T$

$\forall i, W^i \in \mathbb{R}$

If each weight is  $|W^i| > 1$ , then  $|h^k|$  is going to grow exponentially with depth  $k$

If each weight is  $|W^i| < 1$ , then  $|h^k|$  is going to shrink exponentially to 0 with depth  $k$

# INITIALIZING NEURAL NETWORKS

## CONSISTENCY ACROSS LAYERS

Desirable Properties:

$\forall i, \mathbf{E}[h^i] = 0$  – mean activation is zero

$\forall i, \text{Var}(h^i) = \text{Var}(h^{i-1})$  variance is the same across all layers

Want each layer to look like normalized data

# INITIALIZING NEURAL NETWORKS

## CONSISTENCY ACROSS LAYERS

Desirable Properties:

$\forall i, \mathbf{E}[h^i] = 0$  – mean activation is zero

$\forall i, \text{Var}(h^i) = \text{Var}(h^{i-1})$  variance is the same across all layers

This is achieved (for tanh activation functions) using the initialization

$$W_{j,k}^i \sim \mathcal{N}\left(0, \frac{1}{n_{i-1}}\right)$$

Called: Xavier initialization

Reading: <https://www.deeplearning.ai/ai-notes/initialization/index.html>

# XAVIER INITIALIZATION

## REASON IT WORKS

Assumptions:

1. initialization weights are such that  $\tanh(x) \approx x$ , which happens when  $x$  is close to 0.
2. Inputs to the layer are i.i.d.
3. Initial weights are i.i.d.
4. Weights and inputs are mutually independent

# XAVIER INITIALIZATION

REASON IT WORKS

$$\text{Var}(h^i) \approx \text{Var}(z^i) \quad (\text{assume } \tanh(x) \approx x)$$

$$\begin{aligned} \text{Var}(z_j^i) &= \text{Var} \left( \sum_{k=1}^{n_{i-1}} W_{j,k}^i h_{1,k}^{i-1} \right) && \text{independence of weights and inputs} \\ &= \sum_{k=1}^{n_{i-1}} \text{Var} \left( W_{j,k}^i h_{1,k}^{i-1} \right) \end{aligned}$$

# XAVIER INITIALIZATION

REASON IT WORKS

$$\text{Var}(XY) = \mathbf{E}[XY]^2 - \mathbf{E}[X]\mathbf{E}[Y]$$

$$\text{Var}(XY) = \mathbf{E}[X]^2\text{Var}(Y) + \text{Var}(X)\mathbf{E}[Y]^2 + \text{Var}(X)\text{Var}(Y)$$

# XAVIER INITIALIZATION

REASON IT WORKS

$$\text{Var}(XY) = \mathbf{E}[XY]^2 - \mathbf{E}[X]\mathbf{E}[Y]$$

$$\text{Var}(XY) = \mathbf{E}[X]^2\text{Var}(Y) + \text{Var}(X)\mathbf{E}[Y]^2 + \text{Var}(X)\text{Var}(Y)$$

$$\text{Var}\left(W_{j,k}^i h_{1,k}^{i-1}\right) = \underbrace{\mathbf{E}[W_{j,k}^i]^2}_{(a)} \text{Var}(h_{1,k}^{i-1}) + \underbrace{\text{Var}(W_{j,k}^i)\mathbf{E}[h_{1,k}^{i-1}]^2}_{(b)} + \text{Var}(W_{j,k}^i)\text{Var}(h_{1,k}^{i-1})$$

(a) = 0 initialized weights with mean 0

(b) = 0 assume inputs are normalized to zero mean and standard deviation of one.

# XAVIER INITIALIZATION

REASON IT WORKS

$$\text{Var}(h^i) \approx \text{Var}(z^i)$$

$$\begin{aligned}\text{Var}(z_j^i) &= \text{Var} \left( \sum_{k=1}^{n_{i-1}} W_{j,k}^i h_{1,k}^{i-1} \right) = \sum_{k=1}^{n_{i-1}} \text{Var} \left( W_{j,k}^i h_{1,k}^{i-1} \right) \\ &= \sum_{k=1}^{n_{i-1}} \text{Var}(W_{j,k}^i) \text{Var}(h_{1,k}^{i-1}) \\ &= \sum_{k=1}^{n_{i-1}} \frac{1}{n_{i-1}} \text{Var}(h_{1,k}^{i-1}) \\ &= \text{Var}(h^{i-1})\end{aligned}$$

$$\text{Var}(h^i) = \text{Var}(h^{i-1})$$

# MANY INITIALIZATIONS

## DIFFERENT PROPERTIES

$\mathcal{N}\left(0, \frac{1}{n_{i-1}}\right)$  – activations have the same variance at each layer

$\mathcal{N}\left(0, \frac{1}{n_i}\right)$  – gradients have the same variance at each layer

$\mathcal{N}\left(0, \frac{2}{n_i + n_{i-1}}\right)$  – balances both

# MANY INITIALIZATIONS

## DIFFERENT PROPERTIES

$\mathcal{N}\left(0, \frac{1}{n_{i-1}}\right)$  – activations have the same variance at each layer

$\mathcal{N}\left(0, \frac{1}{n_i}\right)$  – gradients have the same variance at each layer

$\mathcal{N}\left(0, \frac{2}{n_i + n_{i-1}}\right)$  – balances both

Rescale the variance based on the activation function. ReLU multiply by 2.

# ADAPTIVE STEP SIZE ALGORITHMS

## BASICS

$$\theta^{k+1} = \theta^k - \eta \nabla l_D(\theta^k)$$

This update assumes that a change in each weight has an equal impact on the loss function corresponding to the magnitude of the gradient term.

- We showed why this wasn't true in linear function approximation

# ADAPTIVE STEP SIZE ALGORITHMS

## BASICS

Changing some weights will have a bigger/smaller impact on the loss function

Changing the output of an early layer will have a chain of impacts on the later layers.

Cannot easily design a step size to have the loss decrease by a constant factor, e.g.,

$$\frac{l(\theta^{k+1})}{l(\theta^k)} \approx \alpha < 1$$

# ADAPTIVE STEP SIZE ALGORITHMS

## BASICS

$$\frac{\partial l(\theta)}{\partial W^i} = \left( \partial_{h^i} l(\theta) \odot \frac{\partial \sigma(z^i)}{\partial z^i} \right)^T h^{i-1}$$

If  $|h_{1,k}^{i-1}|$  is small on average then the change in  $W_{\cdot,k}^i$  will be smaller

- Not because it has less impact on  $l(\theta)$

Activation function may shrink the gradients, making them smaller at earlier layers.

- Change in an earlier layer may be more important, but the derivative is small

# ADAPTIVE STEP SIZE ALGORITHMS

## BASICS

Idea #1: rescale step size for each parameter so that weight changes are roughly on the same scale

$$V_{j,k}^i \approx \mathbf{E} \left[ \left( \partial_{W_{j,k}^i} l(\theta) \right)^2 \right]$$

$$W_{j,k}^i \leftarrow W_{j,k}^i - \frac{\eta}{\sqrt{V_{j,k}^i}} \partial_{W_{j,k}^i} l(\theta)$$

# ADAPTIVE STEP SIZE ALGORITHMS

## BASICS

Idea #1: rescale step size for each parameter so that weight changes are roughly on the same scale

$$V_{j,k}^i \approx \mathbf{E} \left[ \left( \partial_{W_{j,k}^i} l(\theta) \right)^2 \right]$$

$$W_{j,k}^i \leftarrow W_{j,k}^i - \frac{\eta}{\sqrt{V_{j,k}^i}} \partial_{W_{j,k}^i} l(\theta)$$

Properties:

- derivative has large scale, this will adapt the step size to capture the scale
- If the derivative has high variance (noisy / less confident that it is a good direction), then the step size will be smaller.
- If the derivative has a low variance, the step size will be larger.

# ADAPTIVE STEP SIZE ALGORITHMS

RMSPROP

$V_{j,k}^i \approx \mathbf{E} \left[ \left( \partial_{W_{j,k}^i} l(\theta) \right)^2 \right]$  need to approximate the square of the derivative

Use a moving average of past derivatives

$\beta \in (0,1)$  – moving average parameter

$$V_{j,k}^i \leftarrow \beta V_{j,k}^i + (1 - \beta) \left( \partial_{W_{j,k}^i} l_D(\theta) \right)^2$$

Derivative from  
mini-batch  $D$

# ADAPTIVE STEP SIZE ALGORITHMS

## RMSPROP

$\beta \in (0,1)$  – moving average parameter

$\epsilon \in (0,1)$  – numerical stability parameter

$$v_0 \leftarrow 0,$$

For  $t \in \{1,2,\dots\}$  do

$$g_t \leftarrow \nabla l_D(\theta^t)$$

$$v_t \leftarrow \beta v_{t-1} + (1 - \beta) g_t^2$$

$$\theta^{t+1} \leftarrow \theta^t - \frac{\eta}{\sqrt{v_t + \epsilon}} g_t$$

# ADAPTIVE STEP SIZE ALGORITHMS

## RMSPROP

Idea #2: use a moving average of the gradients

A single mini-batch gradient calculation may be inaccurate.

An average of recent gradient estimates can better approximate the gradient on the full dataset

$$M_{j,k}^i \leftarrow \beta M_{j,k}^i + (1 - \beta) \partial_{W^{j,k}} l_D(\theta)$$

# ADAPTIVE STEP SIZE ALGORITHMS

## ADAM

$\beta_1 \in (0,1)$  – moving average parameter for gradient mean

$\beta_2 \in (0,1)$  – moving average parameter for gradient variance

$\epsilon \in (0,1)$

$m_0 \leftarrow 0, v_0 \leftarrow 0$

For  $t \in \{1,2,\dots\}$  do

$$g_t \leftarrow \nabla l_D(\theta^t)$$

$$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\theta^{t+1} \leftarrow \theta^t - \frac{\eta}{\sqrt{v_t + \epsilon}} m_t$$

# ADAPTIVE STEP SIZE ALGORITHMS

## ADAM

$\beta_1 \in (0,1)$  – moving average parameter for gradient mean

$\beta_2 \in (0,1)$  – moving average parameter for gradient variance

$\epsilon \in (0,1)$

$m_0 \leftarrow 0, v_0 \leftarrow 0$

For  $t \in \{1,2,\dots\}$  do

$$g_t \leftarrow \nabla l_D(\theta^t)$$

$$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\theta^{t+1} \leftarrow \theta^t - \frac{\eta}{\sqrt{v_t + \epsilon}} m_t$$

Initial values of  $m_t$  and  $v_t$  will  
be near zero

# ADAPTIVE STEP SIZE ALGORITHMS

## ADAM

$\beta_1 \in (0,1)$  – moving average parameter for gradient mean

$\beta_2 \in (0,1)$  – moving average parameter for gradient variance

$\epsilon \in (0,1)$

$m_0 \leftarrow 0, v_0 \leftarrow 0$

For  $t \in \{1,2,\dots\}$  do

$$g_t \leftarrow \nabla l_D(\theta^t)$$

$$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$$

$$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$$

$$\theta^{t+1} \leftarrow \theta^t - \frac{\eta}{\sqrt{\widehat{v}_t} + \epsilon} \widehat{m}_t$$

# ADAPTIVE STEP SIZE ALGORITHMS

## ADAM

$\beta_1 \in (0,1)$  – moving average parameter for gradient mean

$\beta_2 \in (0,1)$  – moving average parameter for gradient variance

$\epsilon \in (0,1)$

$m_0 \leftarrow 0, v_0 \leftarrow 0$

For  $t \in \{1,2,\dots\}$  do

$$g_t \leftarrow \nabla l_D(\theta^t)$$

$$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$$

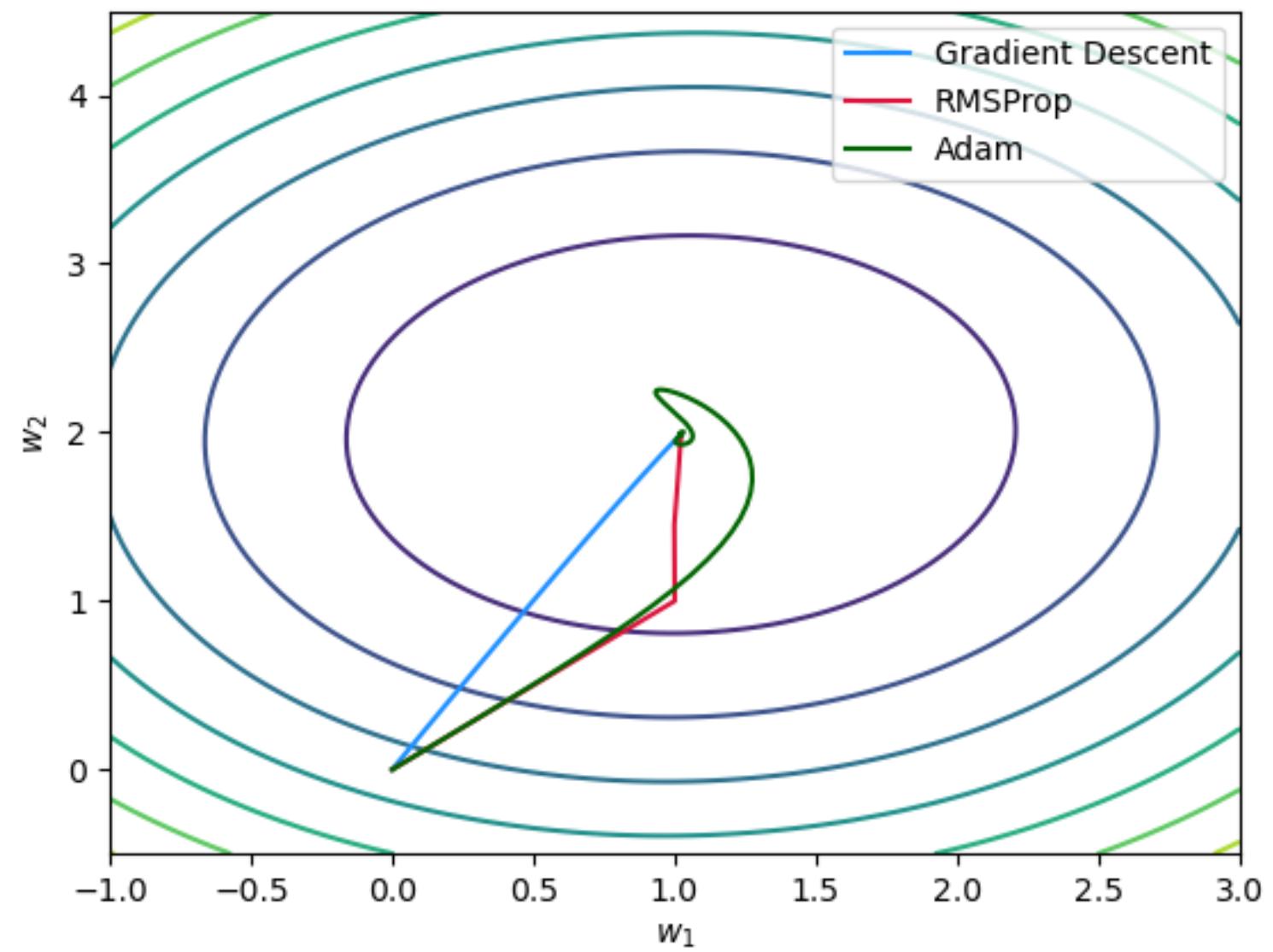
$$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$$

$$\theta^{t+1} \leftarrow \theta^t - \frac{\eta}{\sqrt{\widehat{v}_t + \epsilon}} \widehat{m}_t$$

$$\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$$

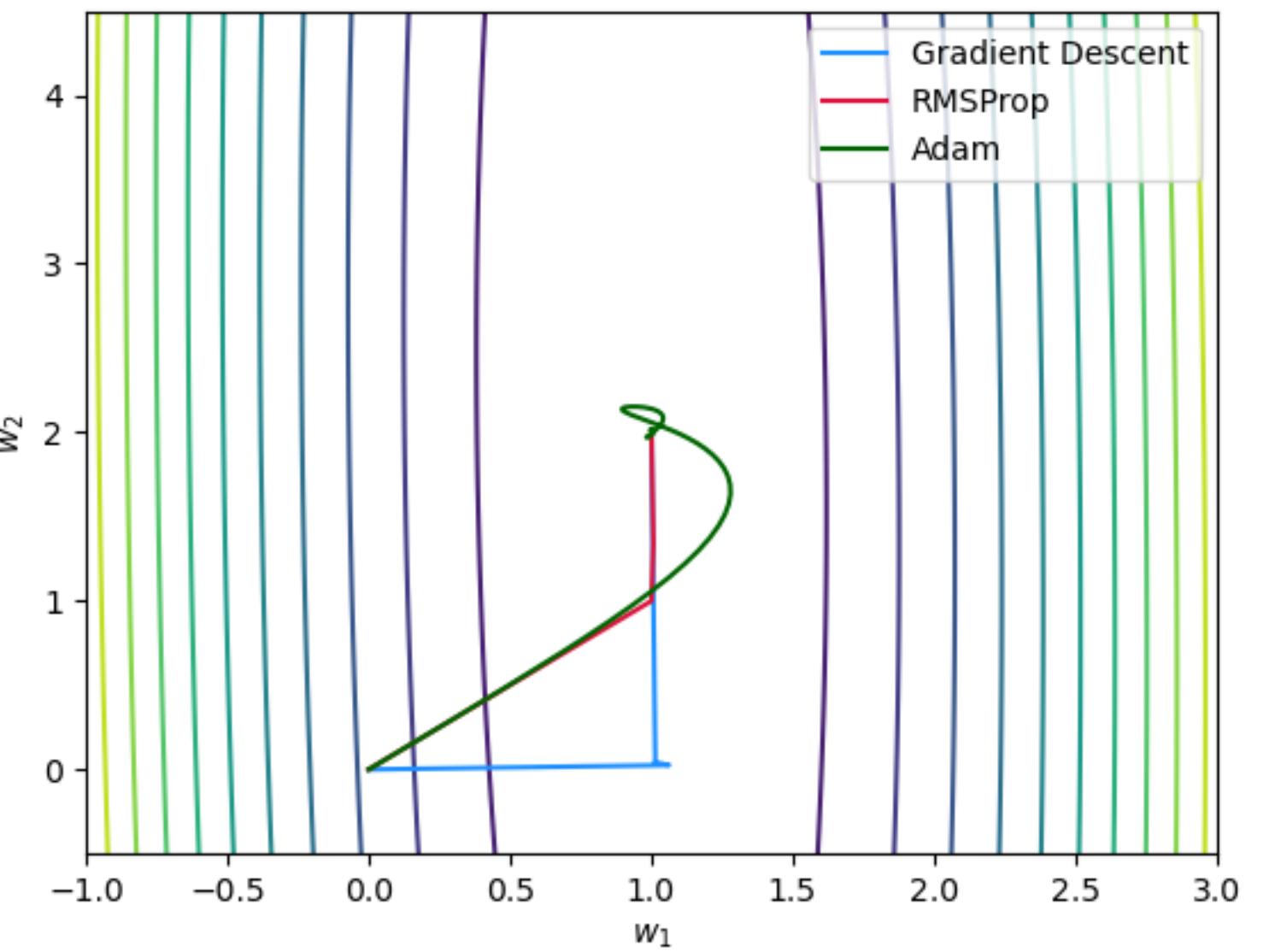
$$\eta \in [10^{-5}, 10^{-3}]$$

# COMPARISON



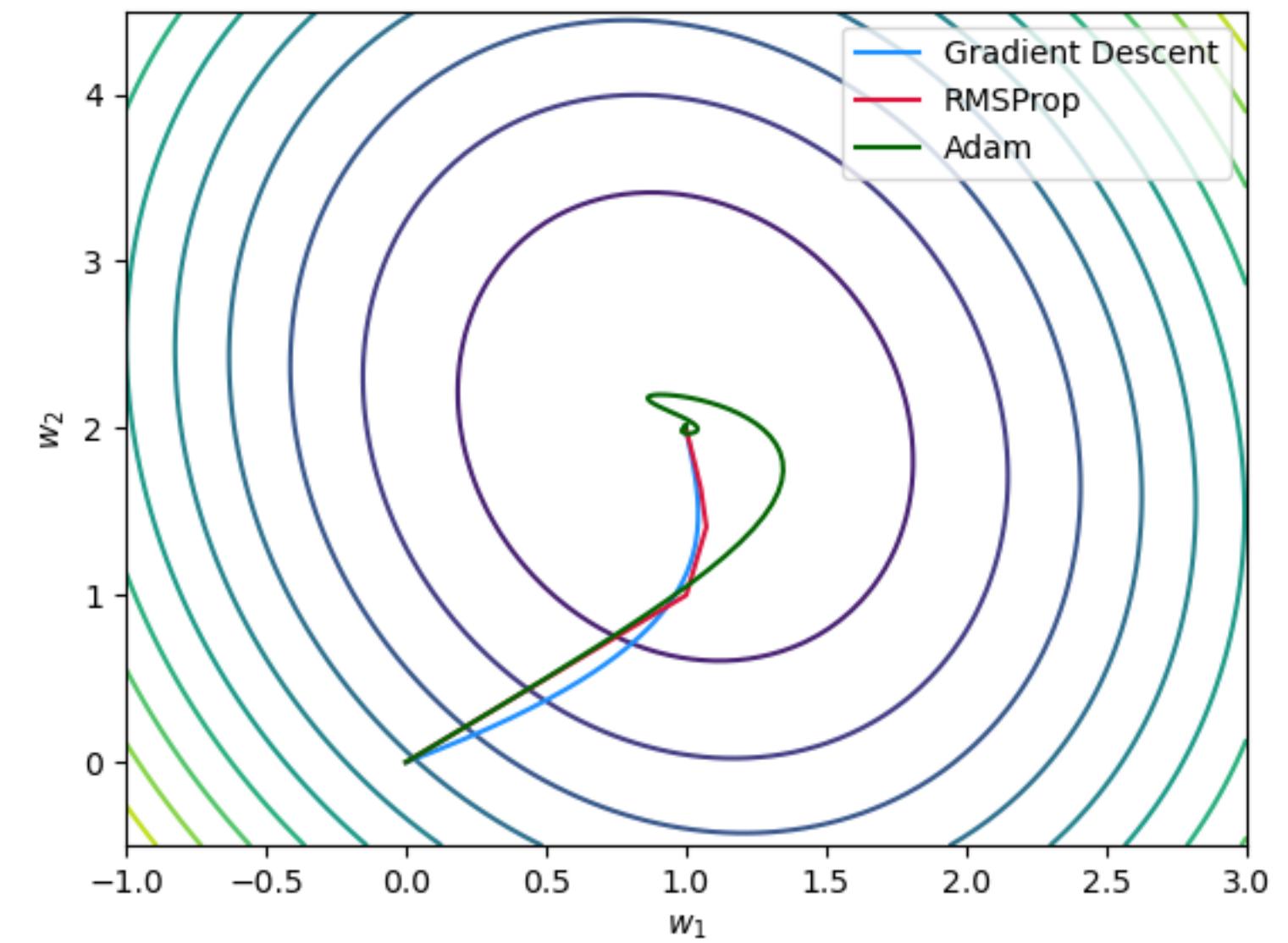
$$x_1 \sim \mathcal{N}(0, 1)$$

$$x_2 \sim \mathcal{N}(0, 1)$$



$$x_1 \sim \mathcal{N}(0, 10^2)$$

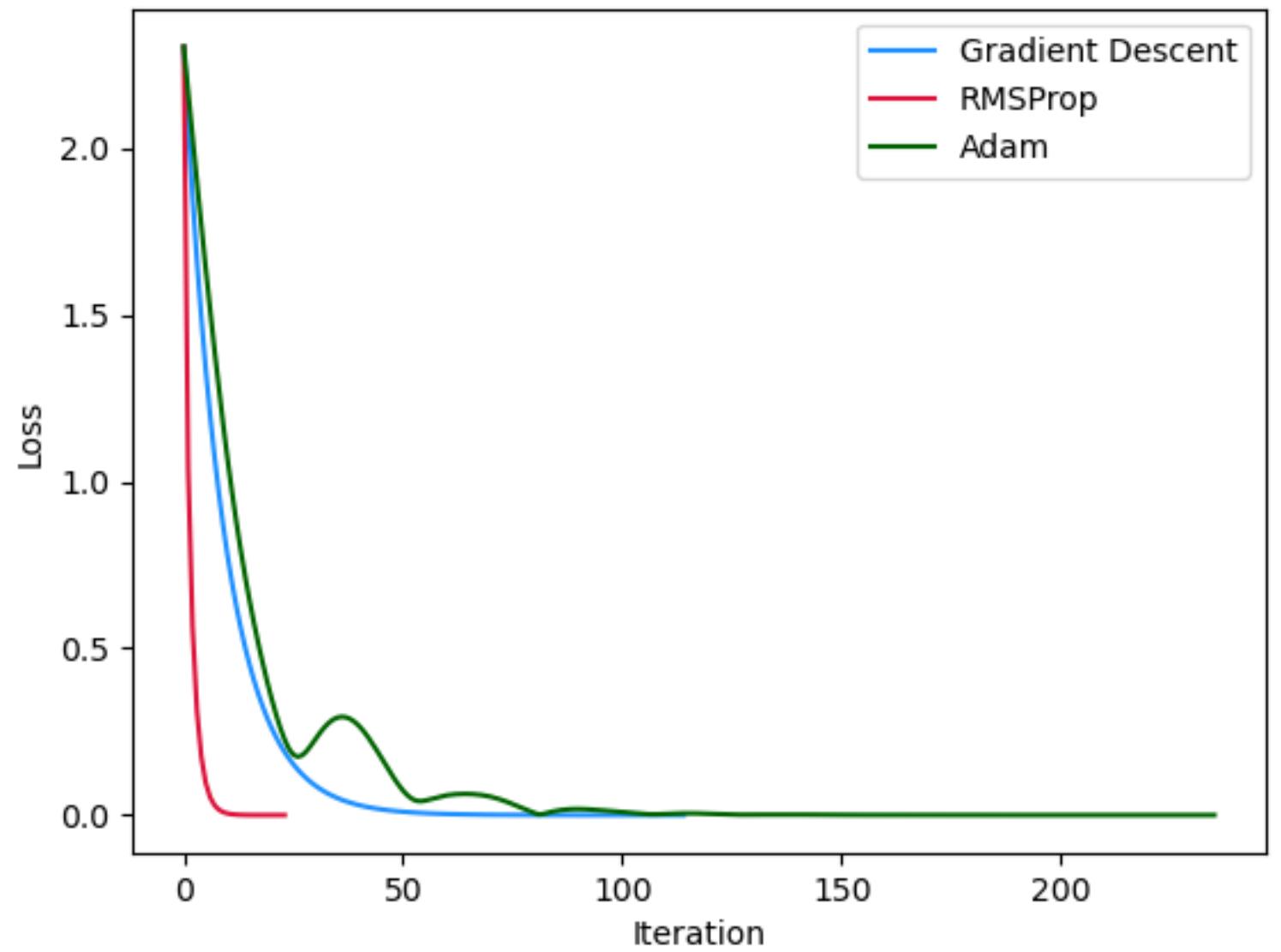
$$x_2 \sim \mathcal{N}(0, 1)$$



$$x_1 \sim \mathcal{N}(3, 1)$$

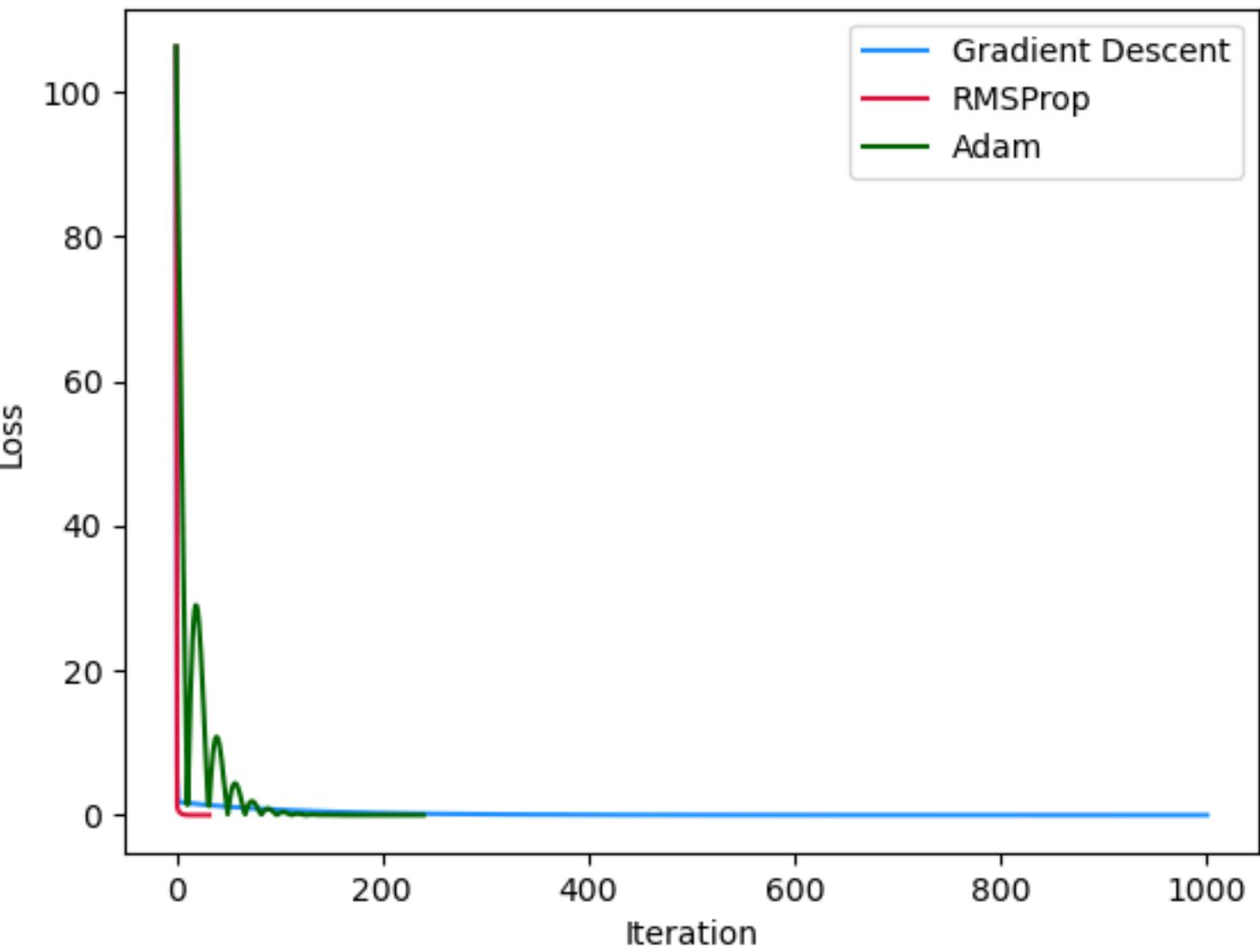
$$x_2 \sim \mathcal{N}(-2, 0.3^2)$$

# COMPARISON



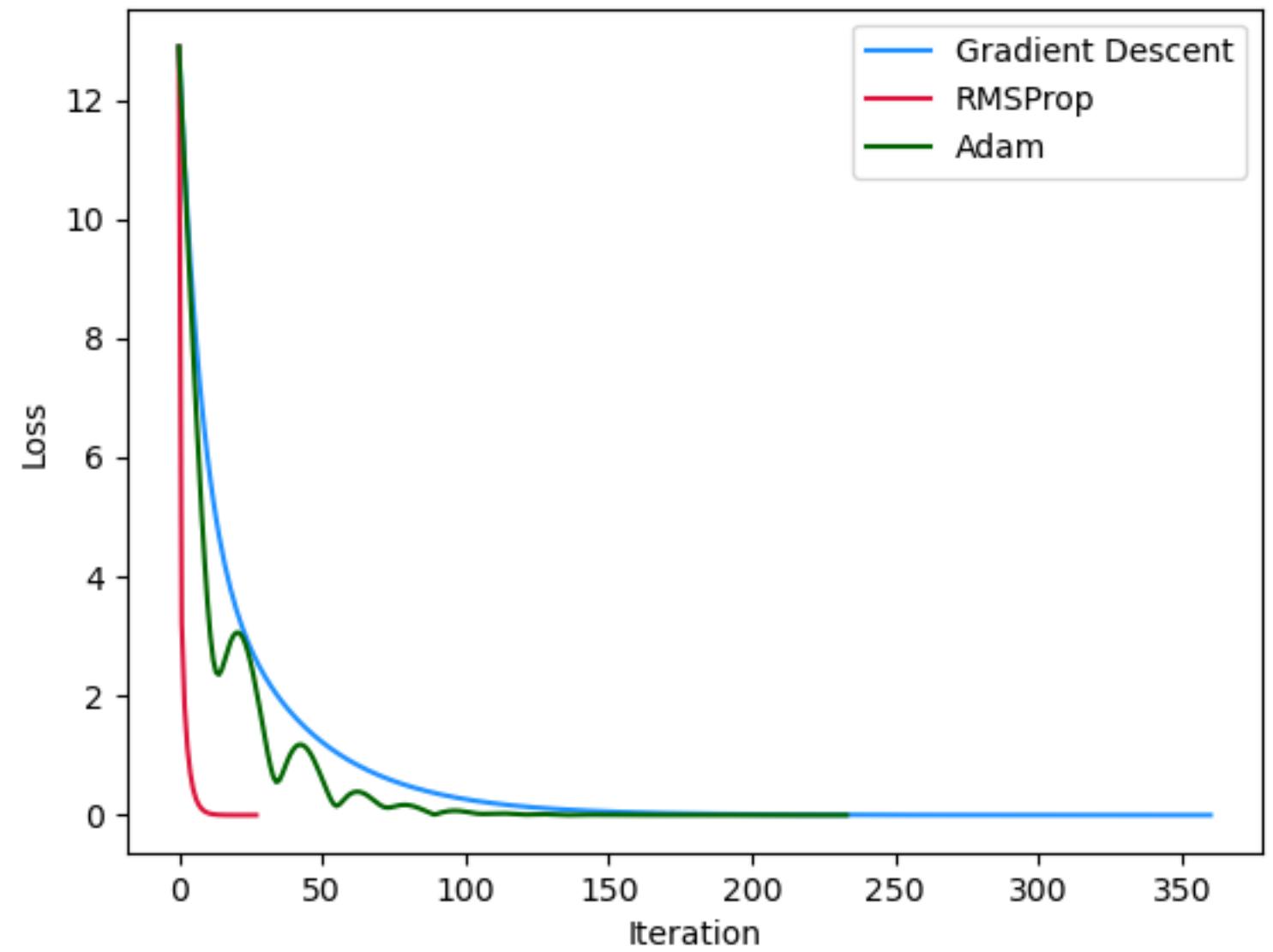
$$x_1 \sim \mathcal{N}(0,1)$$

$$x_2 \sim \mathcal{N}(0,1)$$



$$x_1 \sim \mathcal{N}(0,10^2)$$

$$x_2 \sim \mathcal{N}(0,1)$$



$$x_1 \sim \mathcal{N}(3,1)$$

$$x_2 \sim \mathcal{N}(-2,0.3^2)$$

# TRAINING NNS

## REVIEW

1. Uses randomized mini-batches for fast gradients computation
2. Initialize the network smartly so activations and gradients are not too big or too small
3. Use adaptive step sizes to rescale step sizes for each parameter