INF721

2024/2

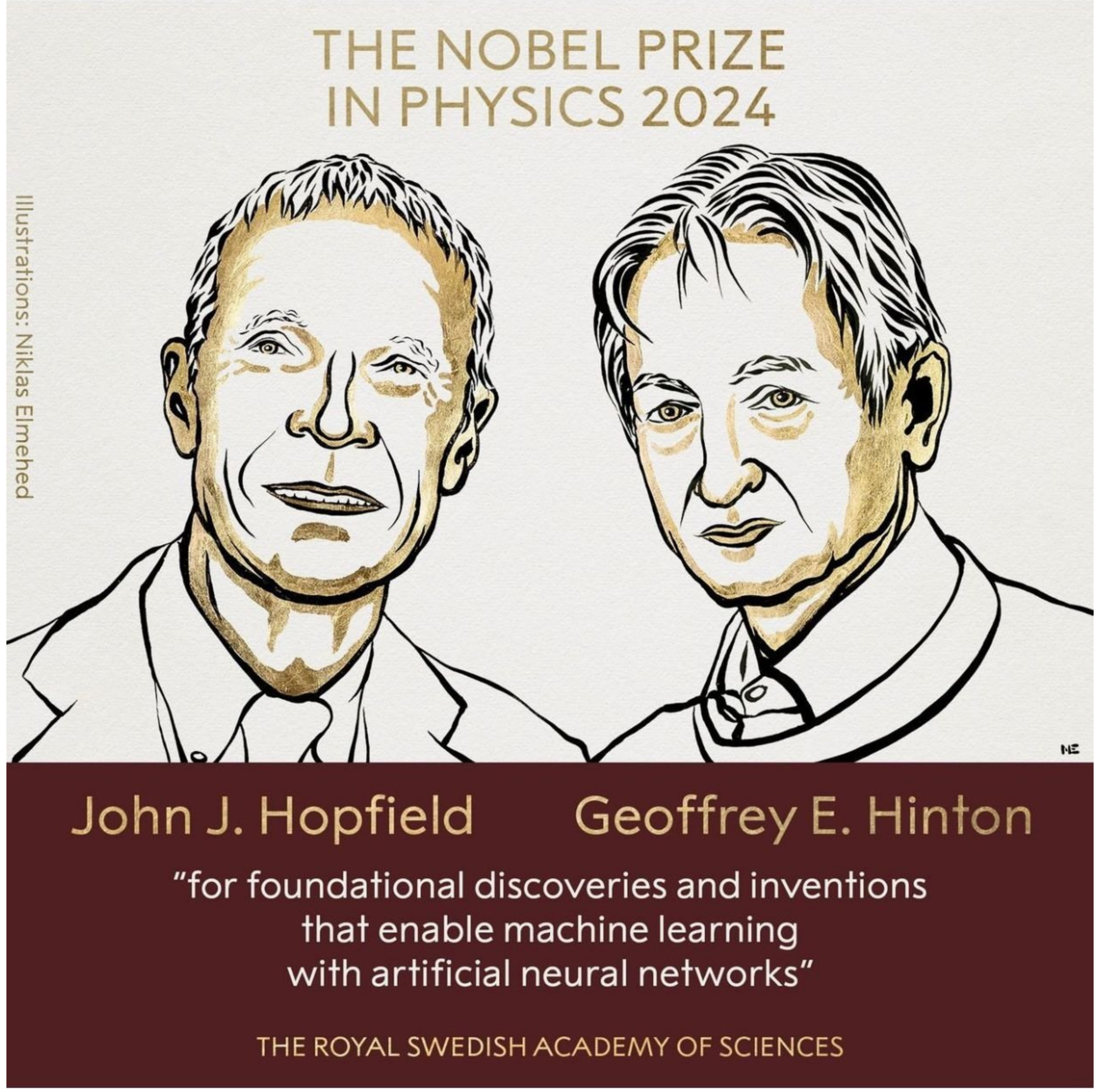
UFV

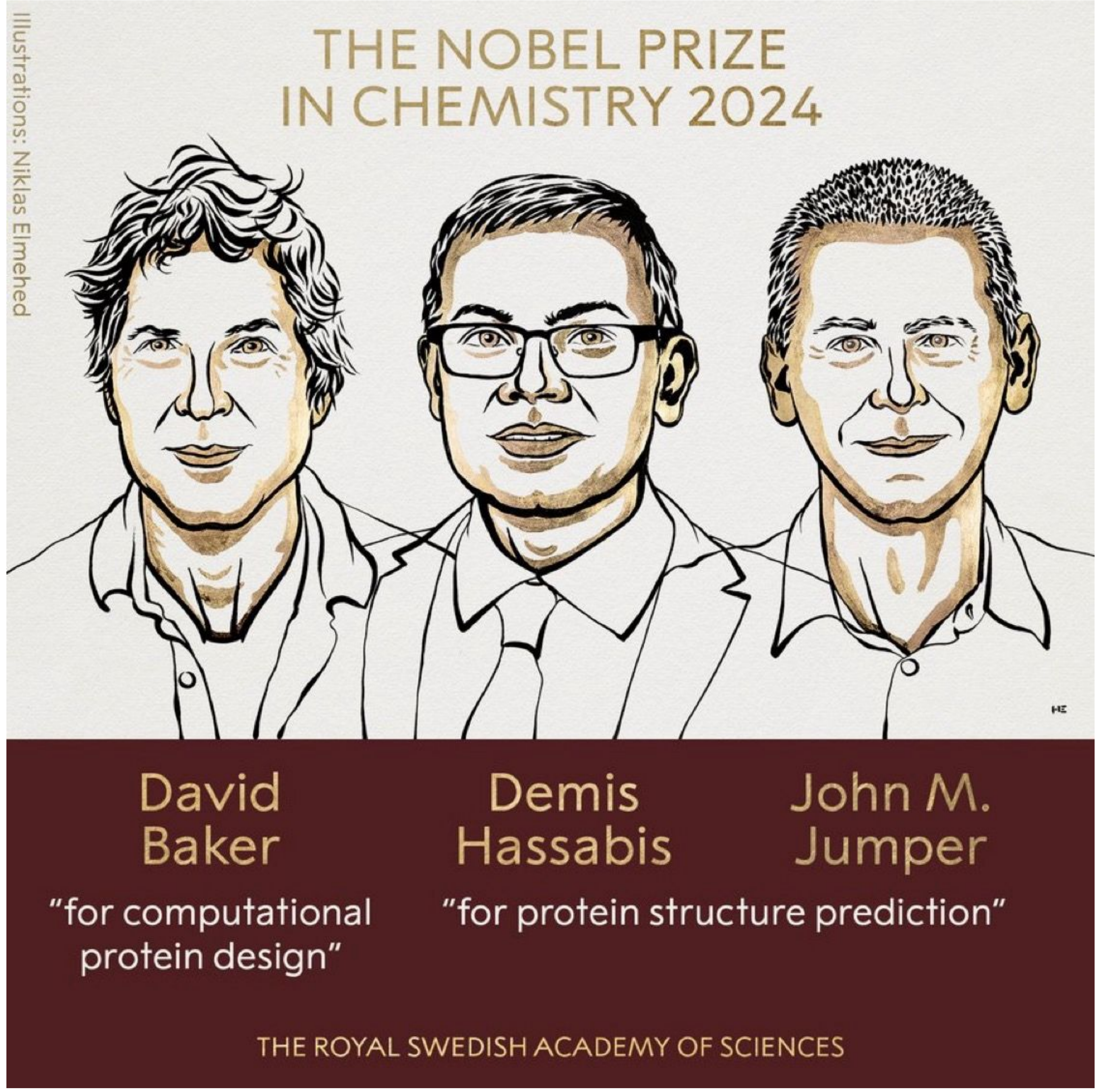
Deep Learning

L9: Advanced Optimization Algorithms

AI in the News

Computer Scientist were awared the Nobel Prize of Physics and Chemistry!





UFV

Logistics

Announcements

- Next class (Monday) we will have our Midterm I!
- PA2 - Multilayer Perceptron is due today (11:59pm)!

Last Lecture

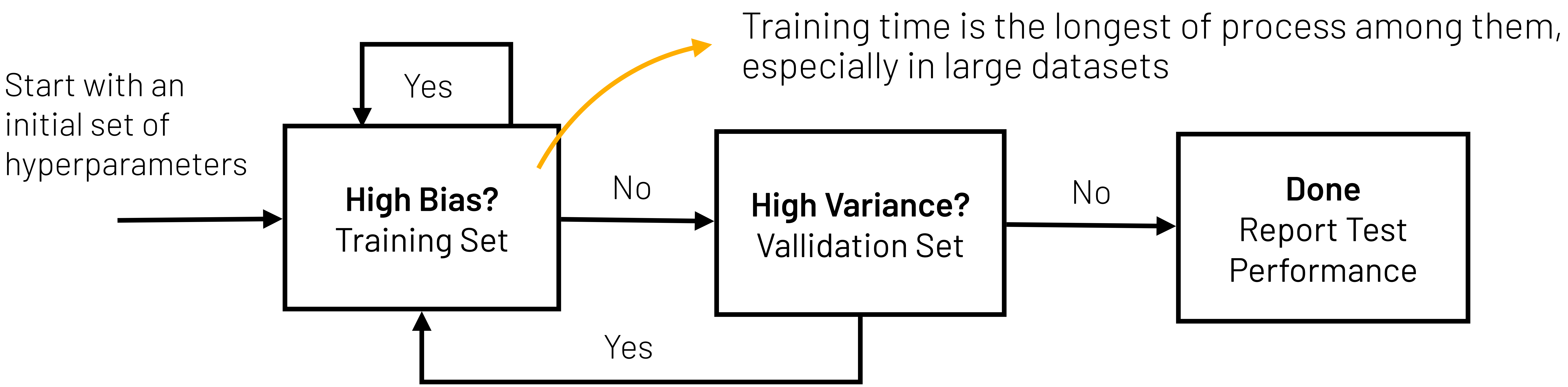
- L1/L2 Regularization
  - Vector/Matrix Norms
- Dropout
- Early Stopping
- Data Augmentation

Lecture Outline

- Mini-batch Gradient Descent
- Gradient Descent with Momentum
  - Exponential Moving Average
- Root Mean Squared Propagation (RMSProp)
- Adaptive Moment Estimation (Adam)

Deep Learning in Practice

Building good deep learning models involves a iterative process of training and validation:



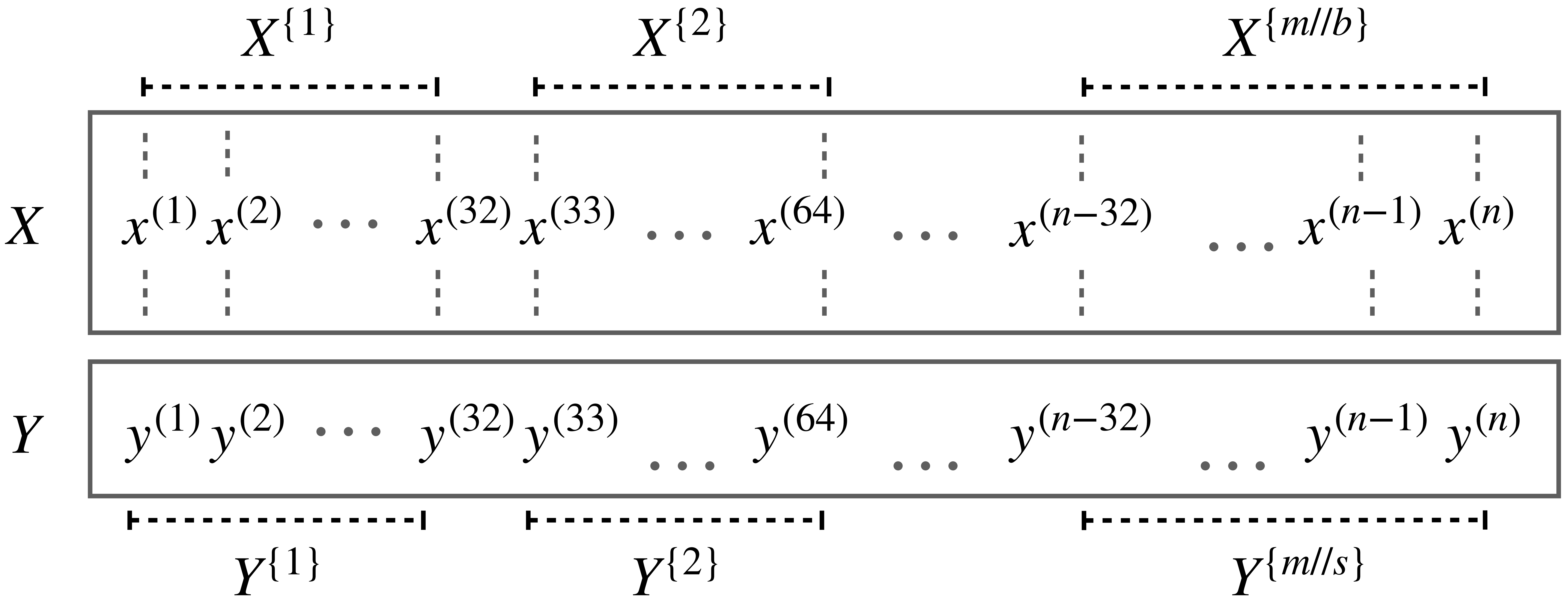
Reduced training time is a crucial factor in creating successful neural network models:

- Vectorization/GPUs
- **Faster Optimization Algorithms**

Mini-batch Gradient Descent

Large datasets (e.g., 5M) won't fit entirely in the GPU memory, so to vectorize model training:

1. Divide the training set in subsets of size s (e.g, 32) called mini-batches
2. Update the weights for each mini-batch $(X^{\{t\}}, Y^{\{t\}})$, instead of once for the entire dataset X



Mini-batch Gradient Descent

```
n_batches = m//s

for e in range(n_epochs):
  # For each minibatch X_t
  for t in range(n_batches):
    # Forward pass for X_t
    Yh_t = forward_pass(X_t)
    # Loss for Yh_t
    l_t = 1/s * np.sum(L(Yh_t, Y_t))
    # Backpropagation of l_t
    dW_t, db_t = backward_pass(X_t, Y_t)
    # Weight updates
    W[l] = W[l] - lr * dW_t
    b[l] = b[l] - lr * db_t
```

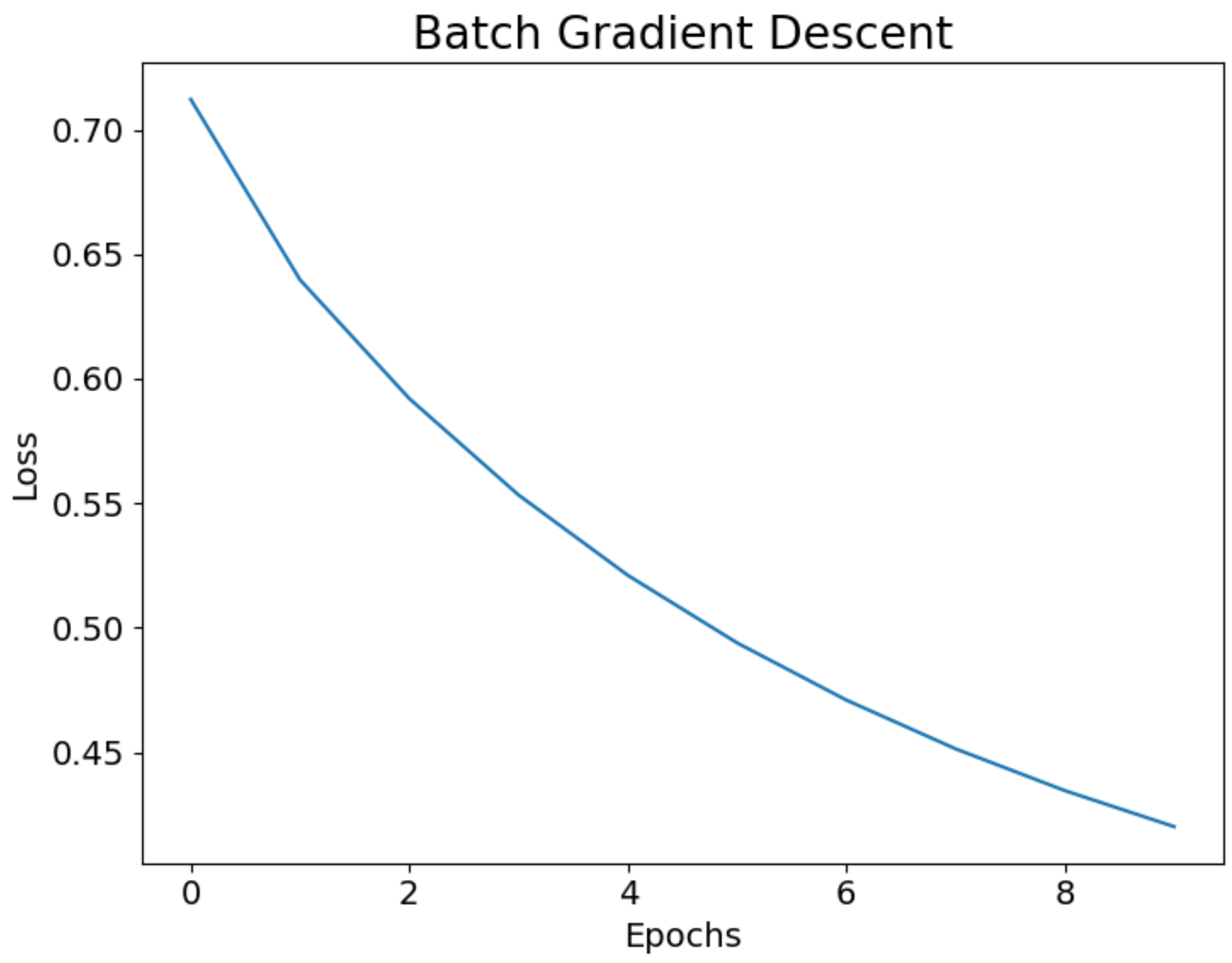
Update weights for each mini-batch $X^{\{t\}}$, instead of once for the entire dataset X

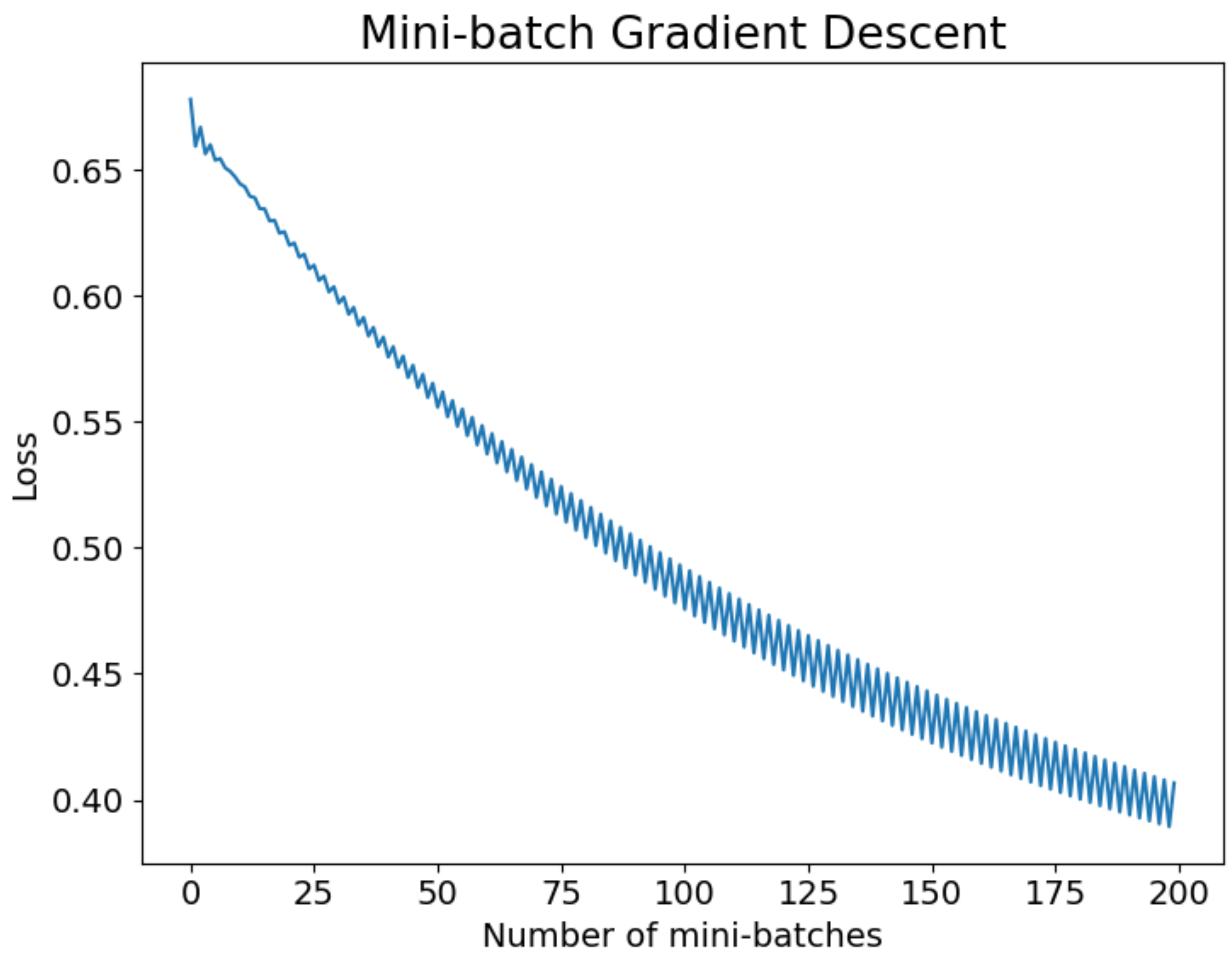
- Multiple weight updates per epoch

- Batch Gradient Descent: $s = m$
- Stochastic Gradient Descent: $s = 1$
- Mini-batch Gradient Descent: $1 > s < m$

UFV

Learning Curves





- BGD computes the exact gradient using the entire dataset in each iteration;
- The curve often shows a steady, monotonic decrease in loss.

- MBGD computes aproximate gradients using small subsets of the dataset;
- The curve often show a noisier/more jagged decrease in loss.

Training Time

Batch Gradient Descent

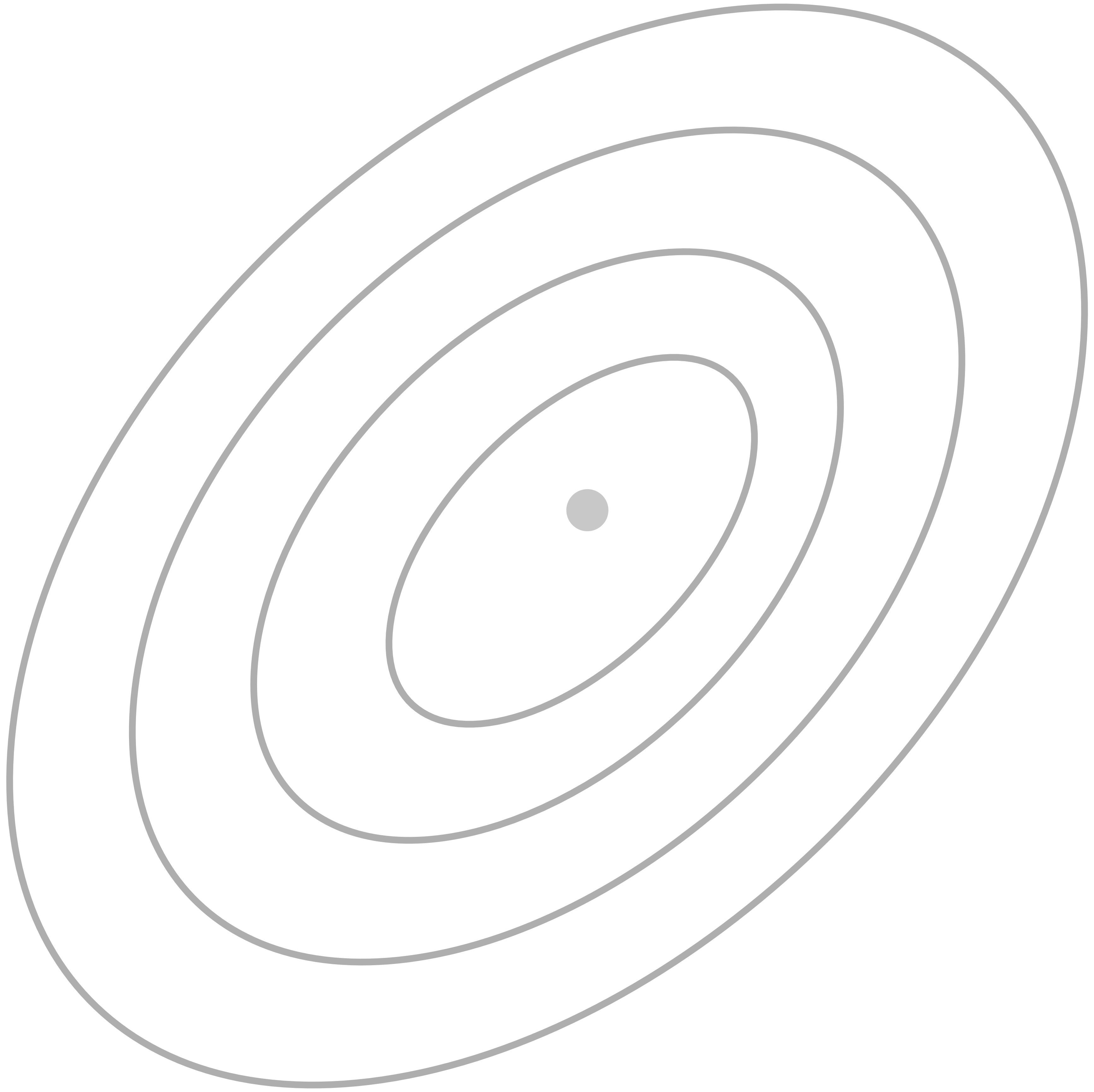
- A single weight update per epoch
- Exact gradient, but slow updates

Stochastic Gradient Descent

- m weight updates per epoch
- Very fast weight updates, but very noisy gradient
- Doesn't use vectorization!

Mini-batch Gradient Descent (most common!)

- One weight update for each mibi-bath X^t
- Fast update with approximate gradients



Training Time

Batch Gradient Descent

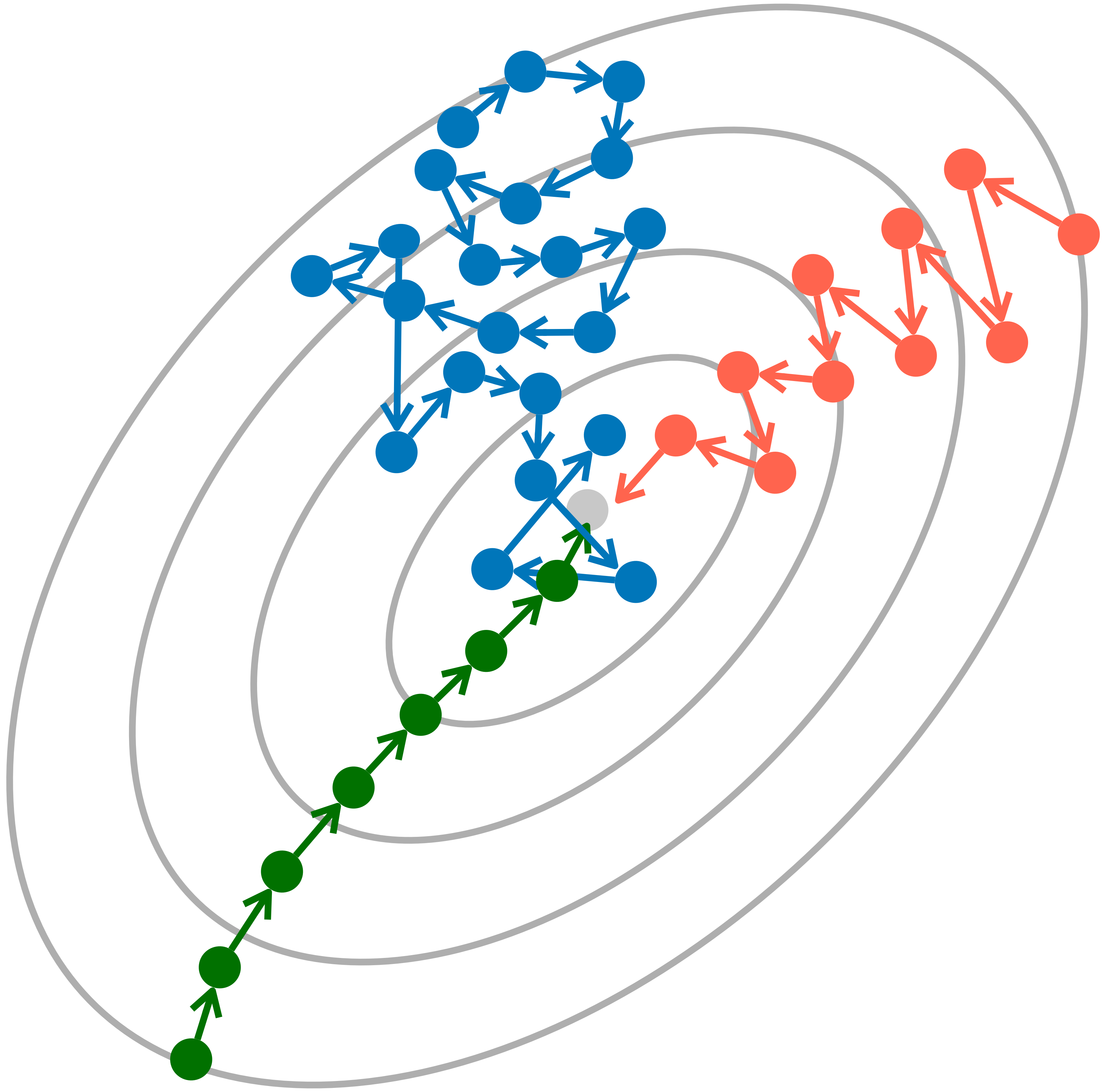
- 1 weight update per epoch for the entire dataset X
- Slow but precise updates, due to exact gradient

Stochastic Gradient Descent

- m updates per epoch, one for each example $x^{(i)}$
- Very fast but very imprecise weight updates, due to very noisy gradient
- Doesn't use vectorization!

Mini-batch Gradient Descent (most common!)

- Multiple updates per epoch, one for each mini-batch $X^{\{t\}}$
- Fast update with approximate gradients



UFV

Choosing batch size

For small datasets:

- Batch Gradient Descent

For large dataset:

- Mini-batch Gradient Descent
- Mini-batch size (hyperparmater):
  - Typically a power of two
  - Fits in your CPU/GPU memory
  - Examples: 32, 64, 128, 256, 512, 1024, ...

UFV

Gradient Descent with Momentum

Moving Average

Moving averages are averaging metrics for time series:

Simple Moving Average (SMA):

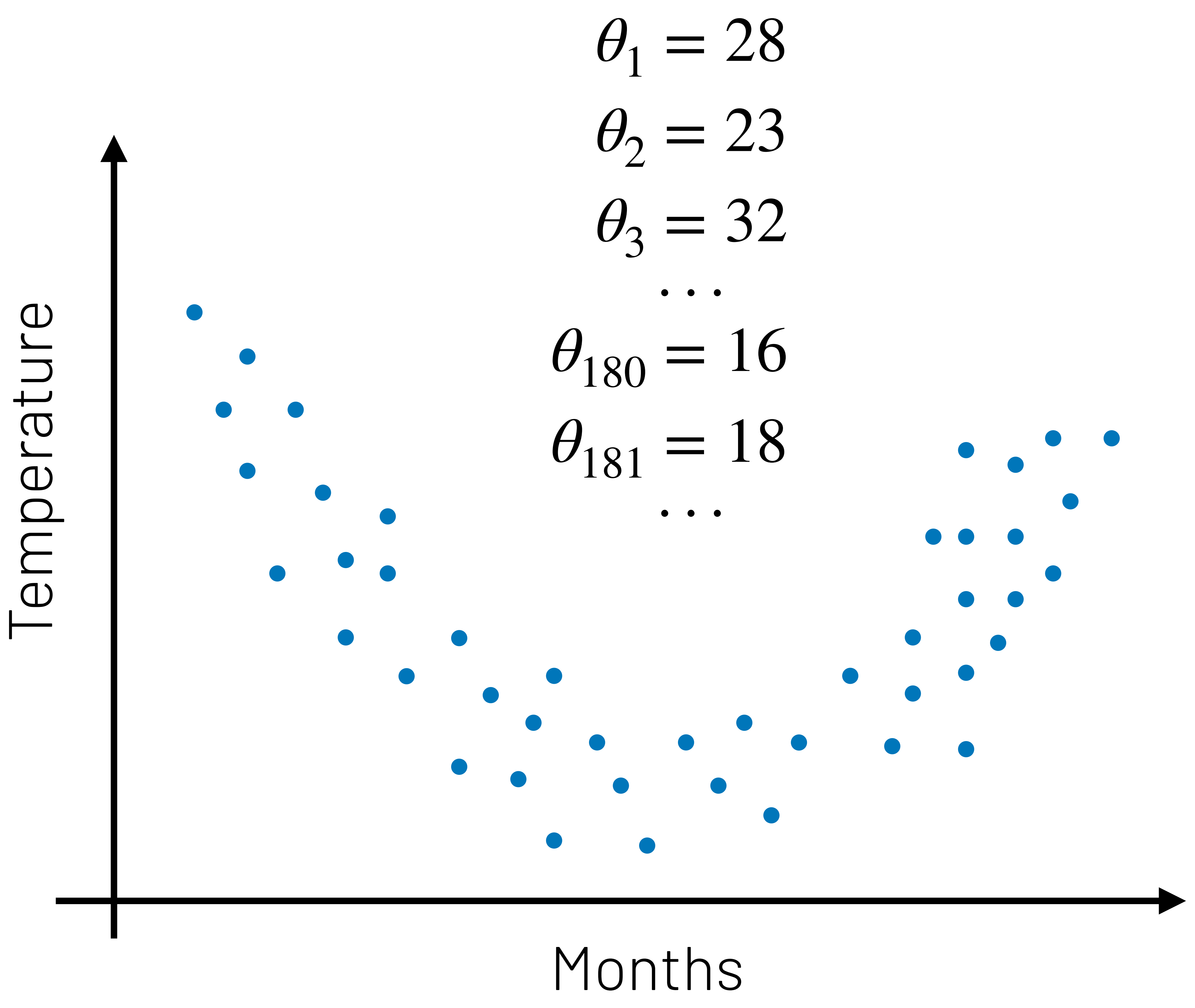
$$v_t = \frac{1}{T} \cdot \sum_{t=1}^{T} \theta_t$$

Weighted Moving Average (WMA):

$$v_t = \frac{1}{\sum_{t=1}^{T} w_t} \cdot \sum_{t=1}^{T} \theta_t \cdot w_t$$

Exponential Moving Average (EMA):

$$v_t = \beta v_{t-1} + (1 - \beta)\theta_t$$



Exponential Moving Average

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t \qquad \beta = 0.9$$

$$v_1 = 0.9 v_0 + 0.1\theta_1$$

$$v_2 = 0.9 v_1 + 0.1\theta_2$$

$$v_3 = 0.9 v_2 + 0.1\theta_3$$

$$\dots$$

v_t is approx. the average of the last $\dfrac{1}{1-\beta}$ samples!

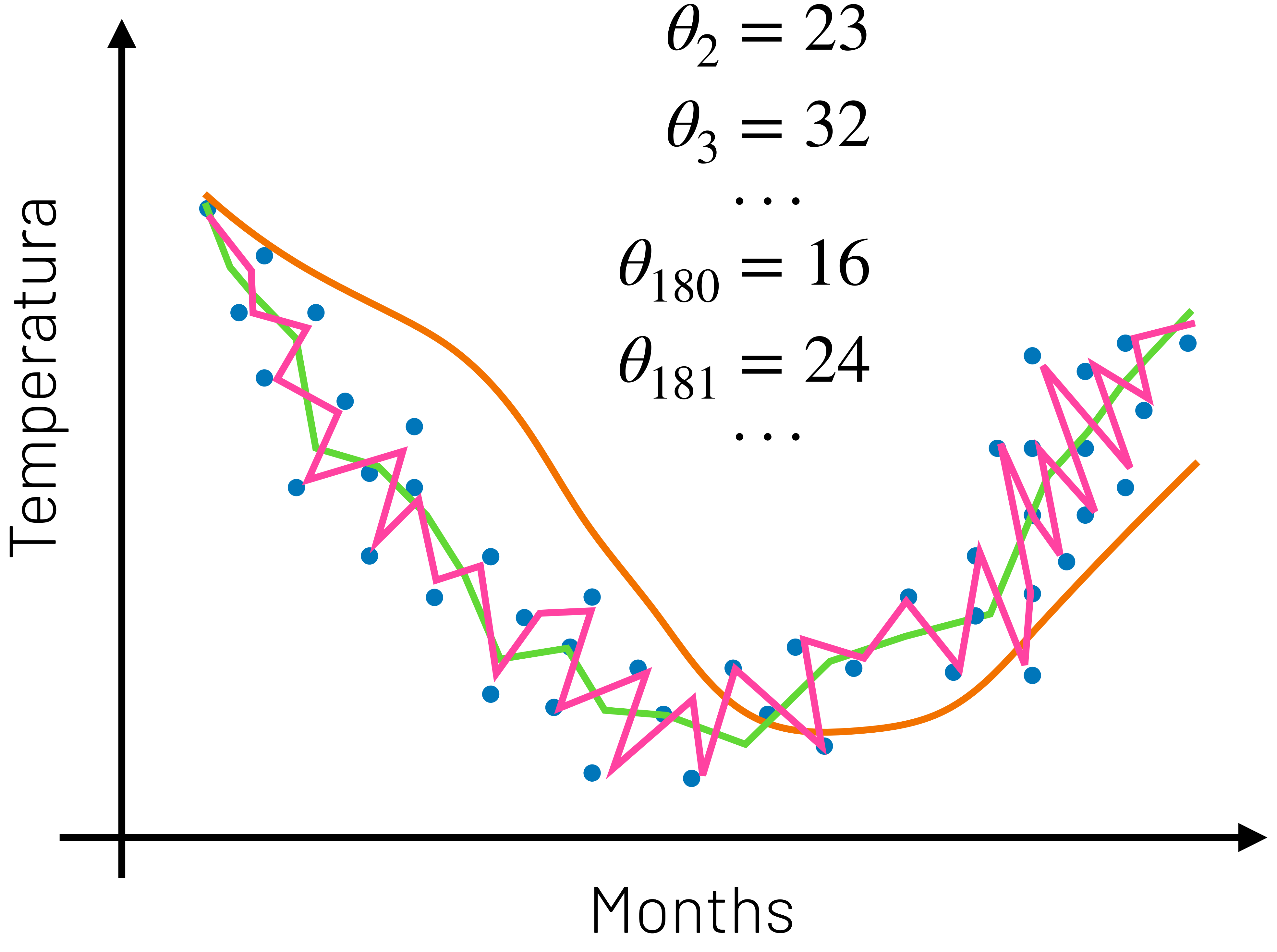
$$\beta = 0.9 = \frac{1}{1-0.9} \approx 10 \text{ days}$$

$$\beta = 0.98 = \frac{1}{1-0.98} \approx 50 \text{ days}$$

$$\beta = 0.5 = \frac{1}{1-0.5} \approx 2 \text{ days}$$

The higher the β, the slower the average adapts to the new samples θ_i

$$\theta_1 = 28$$
$$\theta_2 = 23$$
$$\theta_3 = 32$$
$$\dots$$
$$\theta_{180} = 16$$
$$\theta_{181} = 24$$
$$\dots$$



Exponential Moving Average

The **exponential moving average** is a weighted sum of exponentially decreasing weights!

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t \qquad \beta = 0.9$$

$$\begin{aligned} v_{100} &= 0.9 v_{99} + 0.1\theta_{100} \\ v_{99} &= 0.9 v_{98} + 0.1\theta_{99} \\ v_{98} &= 0.9 v_{97} + 0.1\theta_{98} \end{aligned}$$

$$\begin{aligned} v_{100} &= 0.1\theta_{100} + 0.9 v_{99} \\ &= 0.1\theta_{100} + 0.9(0.1\theta_{99} + 0.9 v_{98}) \\ &= 0.1\theta_{100} + 0.9(0.1\theta_{99} + 0.9(0.1\theta_{98} + 0.9 v_{97})) \\ &= 0.1\theta_{100} + 0.1\underline{(0.9)} \cdot \theta_{99} + 0.1\underline{(0.9)}^2 \cdot \theta_{98} + 0.1\underline{(0.9)}^3 \cdot \theta_{97} + \ldots \end{aligned}$$

UFV

Bias Correction

The initial average values are bad estimates!
This can be solved with bias correction (dividing by $1 - \beta^t$)

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t \longrightarrow v_t = \frac{\beta v_{t-1} + (1-\beta)\theta_t}{1-\beta^t}$$

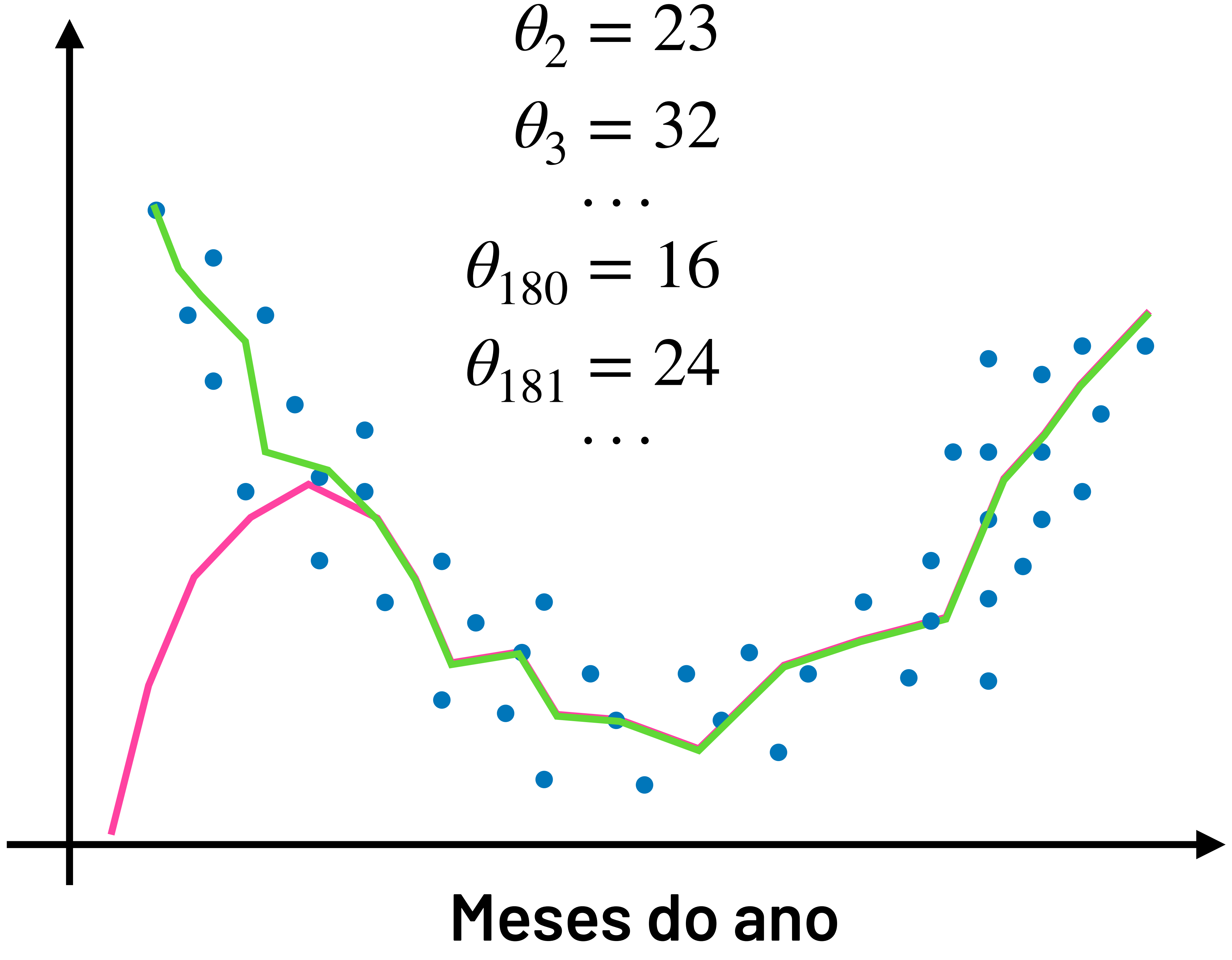
$$v_0 = 0$$
$$v_1 = 0.98 v_0 + 0.02\theta_1$$
$$v_2 = 0.98 v_1 + 0.02\theta_2$$
$$= 0.98 \cdot 0.02\theta_1 + 0.02\theta_2$$
$$= 0.00196\theta_1 + 0.02\theta_2$$

$$v_2 = \frac{0.00196\theta_1 + 0.02\theta_2}{1 - 0.98^2}$$
$$v_2 = \frac{0.00196\theta_1 + 0.02\theta_2}{0.0396}$$

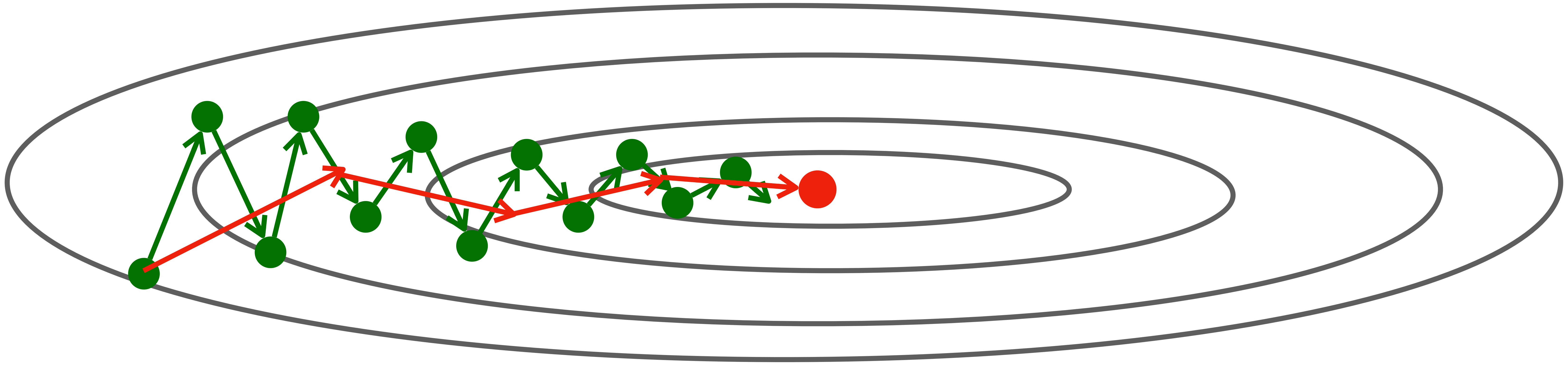
Weighted Average!

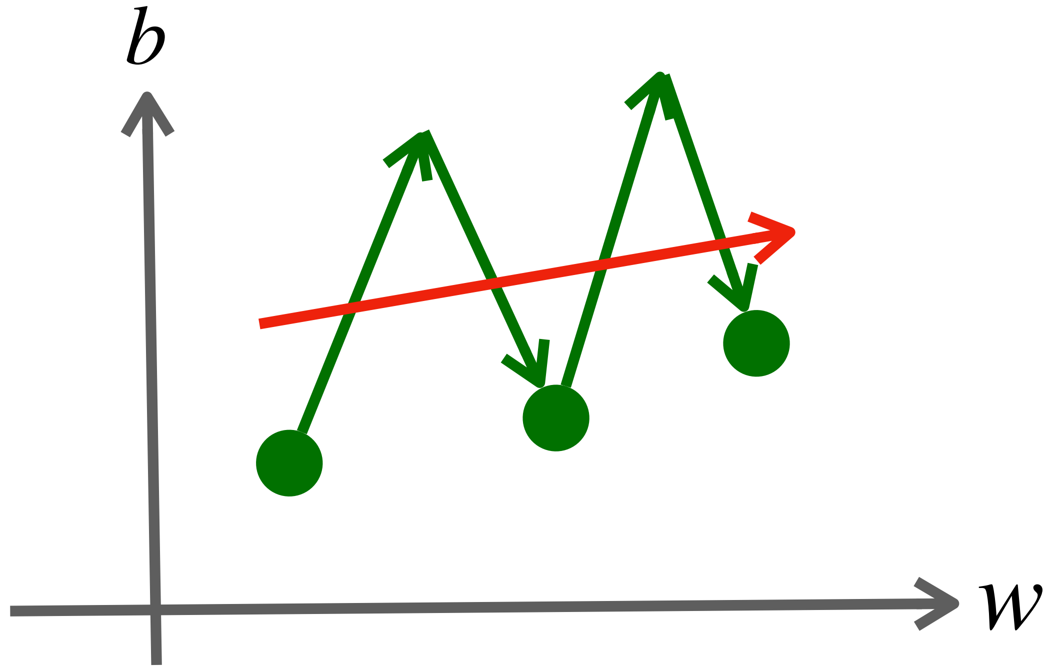
$$v_1 = 0.56 \qquad\qquad v_1 = 28$$
$$v_2 = 0.51136 \longrightarrow v_2 \approx 13$$

$$\theta_1 = 28$$
$$\theta_2 = 23$$
$$\theta_3 = 32$$
$$\cdots$$
$$\theta_{180} = 16$$
$$\theta_{181} = 24$$
$$\cdots$$



Gradient Descent with Momentum





Average close to zero in the vertical axis!

Mini-batch Gradient Descent
Low learning rate to avoid divergence.

Ideal

↕ Slow learning on the vertical axis

↔ Fast learning on the horizontal axis

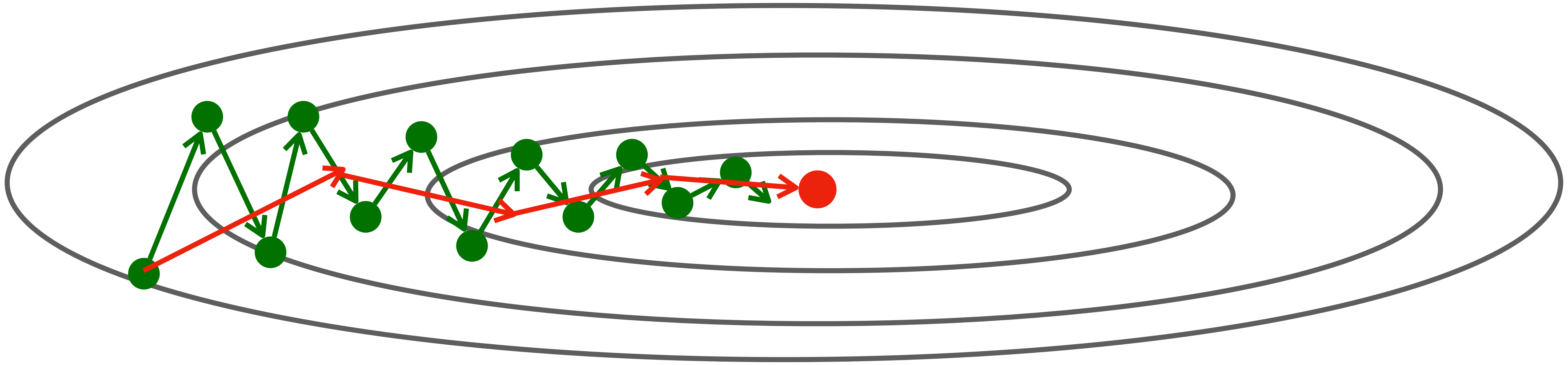
Gradient Descent with Momentum

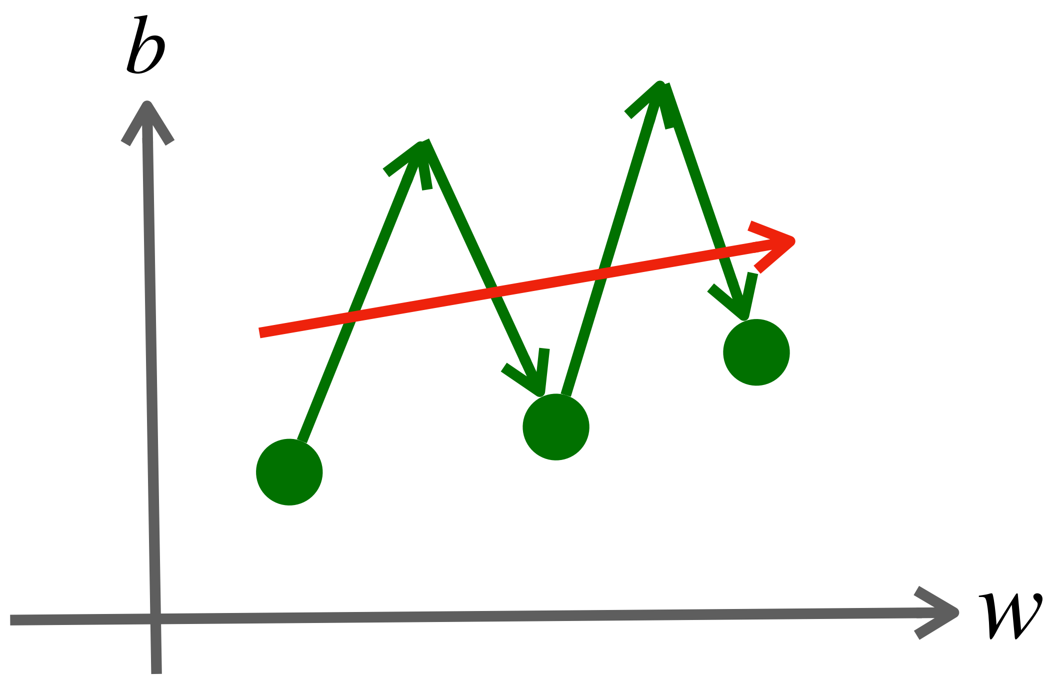
$$dw, db = backward(X^t)$$

$$Vdw = \beta \cdot Vdw + (1 - \beta)dw$$
$$Vdb = \beta \cdot Vdb + (1 - \beta)db$$

$$W^{[l]} = W^{[l]} - \alpha Vdw$$
$$b^{[l]} = b^{[l]} - \alpha Vdb$$

Root Mean Squared Propagation (RMSProp)





Average close to zero in the vertical axis!

Mini-batch Gradient Descent

Low learning rate to avoid divergence.

Ideal

↕ Slow learning on the vertical axis

↔ Fast learning on the horizontal axis

RMSProp

$$dw, db = backward(X^t)$$

$$Sdw = \beta \cdot Sdw + (1-\beta)dw^2$$ Small expected values

$$Sdb = \beta \cdot Sdb + (1-\beta)db^2$$ Large expected values

$$w = w - \alpha\frac{dw}{\sqrt{Sdw}}$$ Dividing by a small number

$$b = b - \alpha\frac{db}{\sqrt{Sdb}}$$ Dividing by a large number

UFV

Adaptive Moment Estimation (Adam)

Adam combines RMSProp and Momentum

$$dw, db = backward(X^t)$$

$$Vdw = \beta_1 \cdot Vdw + (1 - \beta_1)dw, \quad Vdb = \beta_1 \cdot Vdb + (1 - \beta_1)db$$

Momentum

$$Sdw = \beta_2 \cdot Sdw + (1 - \beta_2)dw^2, \quad Sdb = \beta_2 \cdot Sdb + (1 - \beta_2)db^2$$

RMSProp

$$Vdw = \frac{Vdw}{1 - \beta_1^t}, \quad Vdb = \frac{Vdb}{1 - \beta_1^t}$$

$$Sdw = \frac{Sdw}{1 - \beta_2^t}, \quad Sdb = \frac{Sdb}{1 - \beta_2^t}$$

$$w = w - \alpha \frac{Vdw}{\sqrt{Sdw}}$$

$$b = b - \alpha \frac{Vdb}{\sqrt{Sdb}}$$

Recomendations for the values of hyperparameters:

$$\beta_1 = 0.9$$

$$\beta_2 = 0.999$$

Next Lecture

L10: Convolutional Neural Networks

Convolutions, Filters, Padding, Strided Convolutions, Volume Convolutions