ENGR 4399
Machine Learning

Lesson 13: Deep Learning - Regularization and Training

Instructor: Gonzalo D. Parra, PhD

Lesson 13

- Introduction to Deep Learning:
    - Optimization
    - Training
    - Regularization

Research Projects

Abstract

Once upon a time, there was a big mean dragon.

He threatened the kingdom and all of the people there with his fiery breath.

Fortunately, there were some brave knights who lived in the kingdom.

Their shields and swords were the best in the land.

These weapons were fire proof and their swords were even sharper than dragons teeth.

The brave knights have trained in a land far far away and are ready to take on the terrible beast.

With the dragon gone, the kingdom will live in harmony happily ever after.

Abstract

1. What is the overall problem you are trying to solve?

   Once upon a time, there was a big mean dragon.
2. Why is the problem important?

   He threatened the kingdom and all of the people there with his fiery breath.
3. How is your team qualified to do this work?

   Fortunately, there were some brave knights who lived in the kingdom.
4. What is your core technology?

   Their shields and swords were the best in the land.
5. How will your technology solve the problem?

   These weapons were fire proof and their swords were even sharper than dragons teeth.
6. Why is your idea feasible NOW?

   The brave knights have trained in a land far far away and are ready to take on the terrible beast.
7. What will success look like?

   With the dragon gone, the kingdom will live in harmony happily ever after.

Abstract

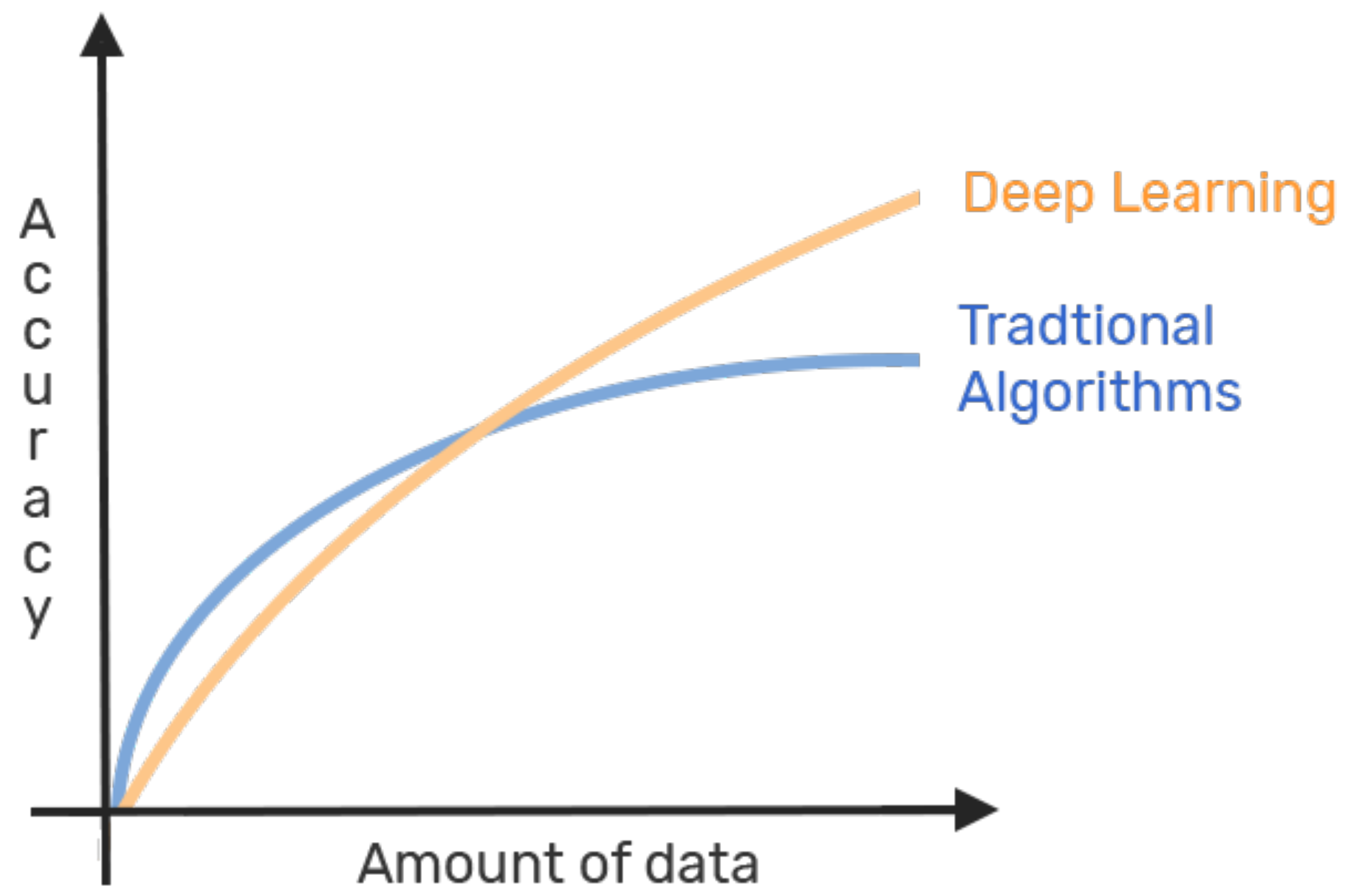
1. **What is the overall problem you are trying to solve?**
2. **Why is the problem important?**
3. **How is your team qualified to do this work?**
4. **What is your core technology?**
5. **How will your technology solve the problem?**
6. **Why is your idea feasible NOW?**
7. **What will success look like?**

Need good examples?
Refer to papers published at AI/ML/DL conferences & journals
(see course website for recommendations)

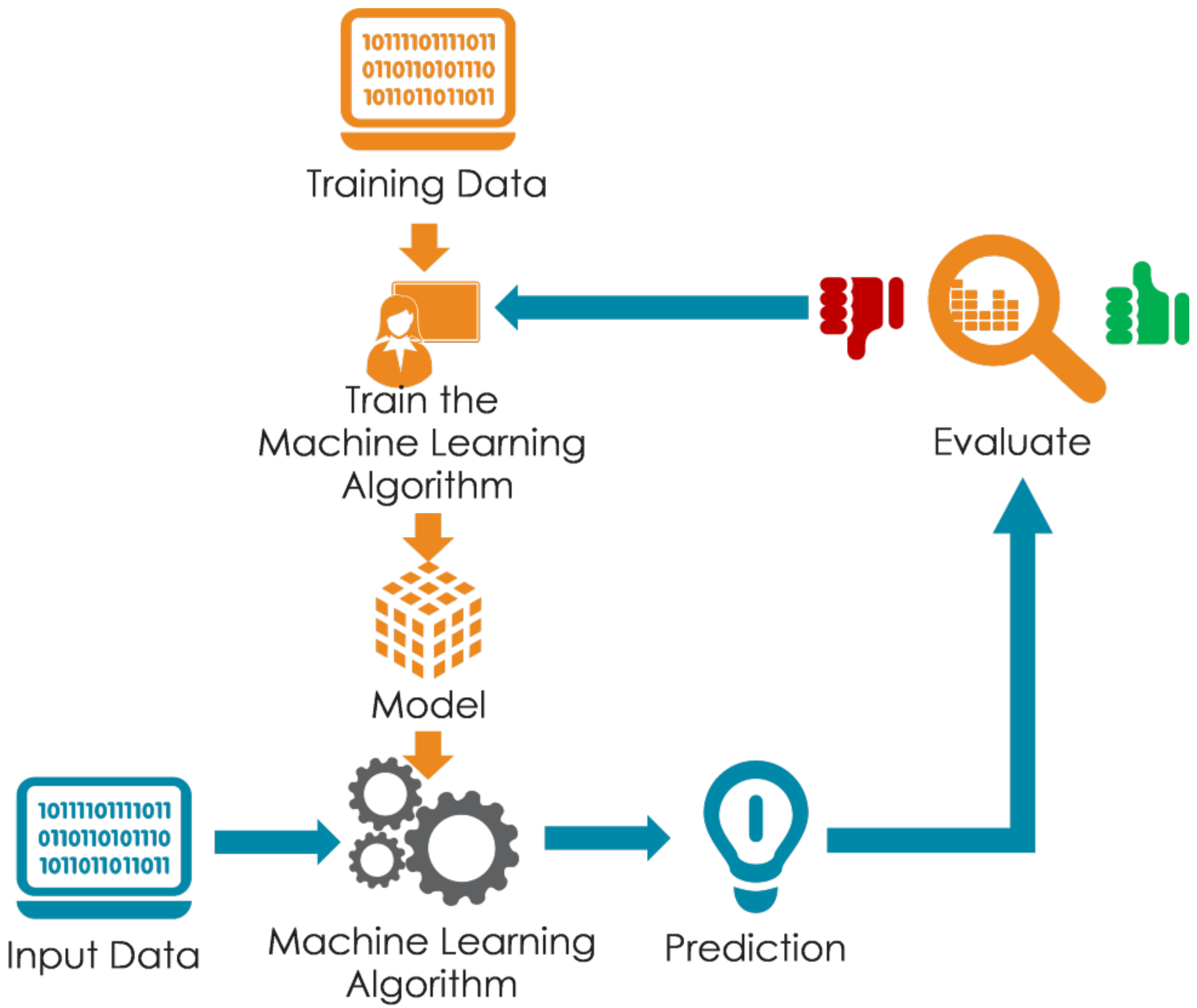
Introduction to Deep Learning

Introduction to Deep Learning

- Optimization
- Training
- Regularization
- Batching



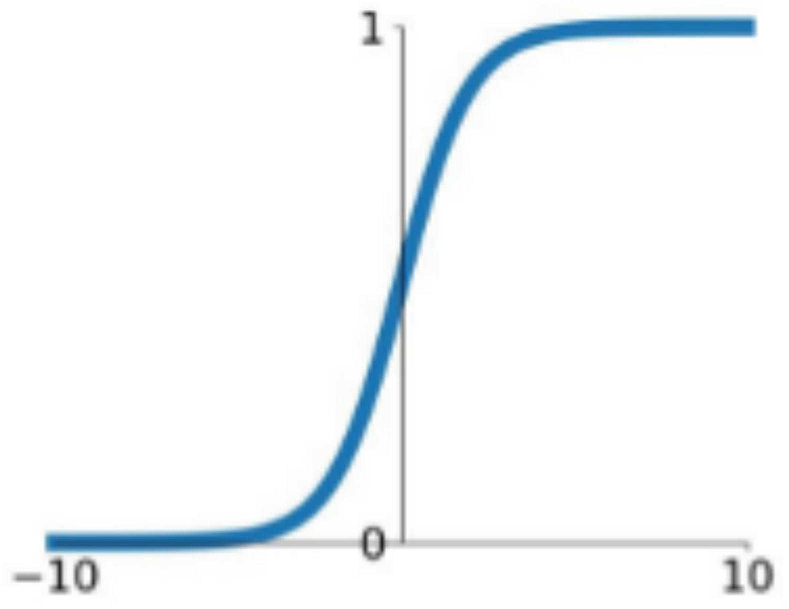
Training


101111011110111
0110110101110
1011011011011
Training Data
Train the Machine Learning Algorithm
Model
101111011110111
0110110101110
1011011011011
Input Data
Machine Learning Algorithm
Prediction
Evaluate

Activation Function

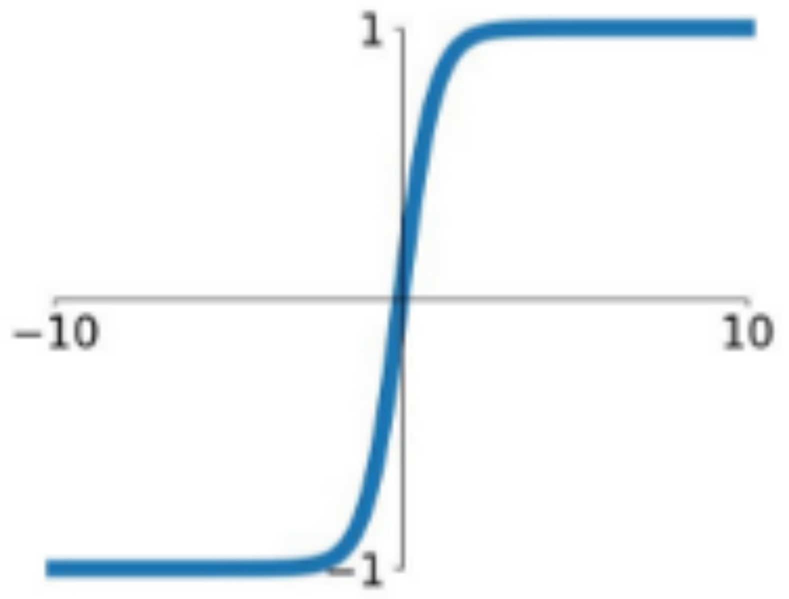
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



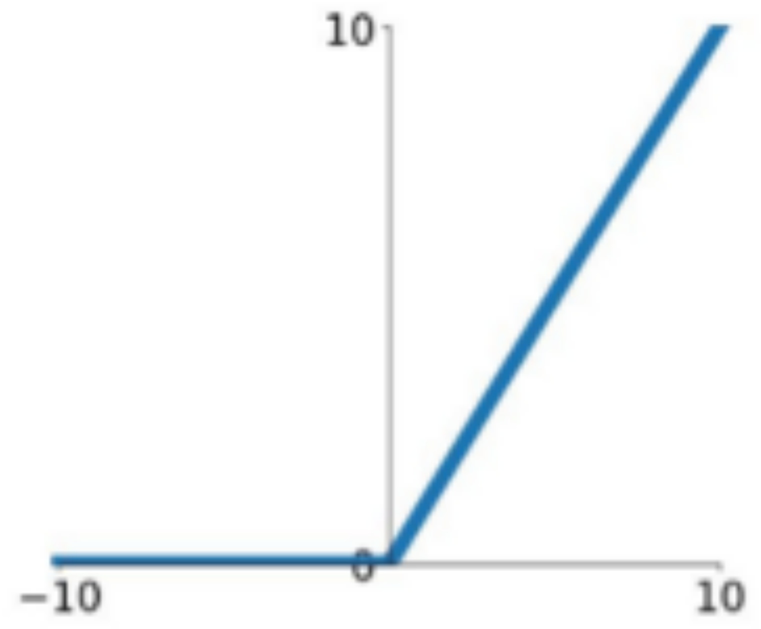
tanh

$\tanh(x)$



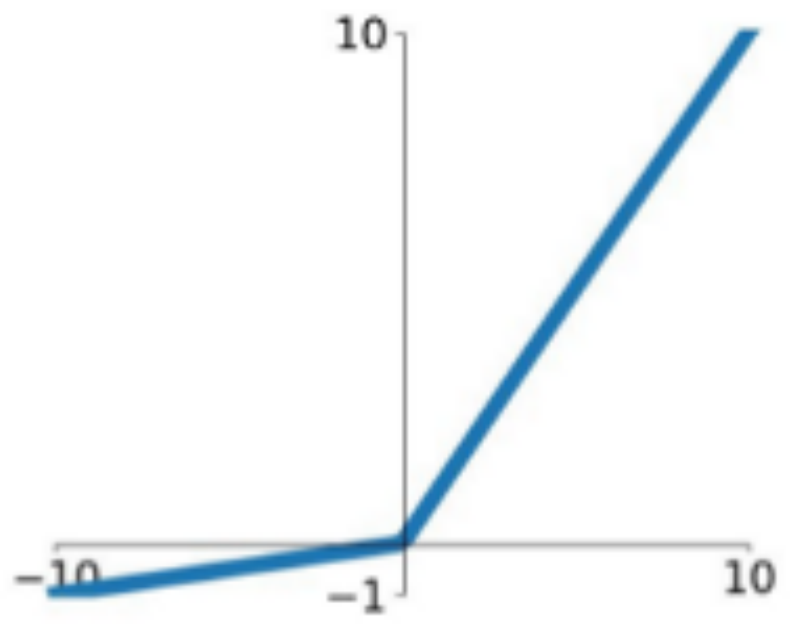
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

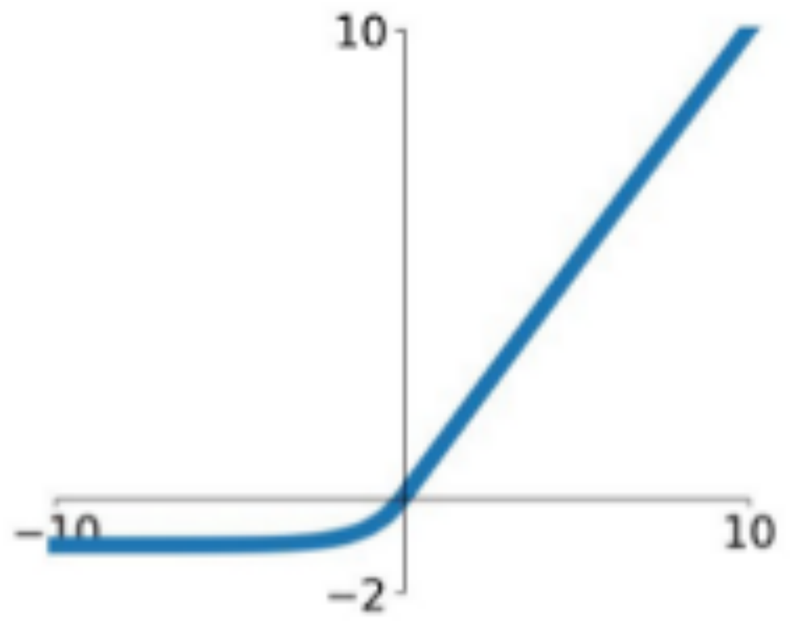


Maxout

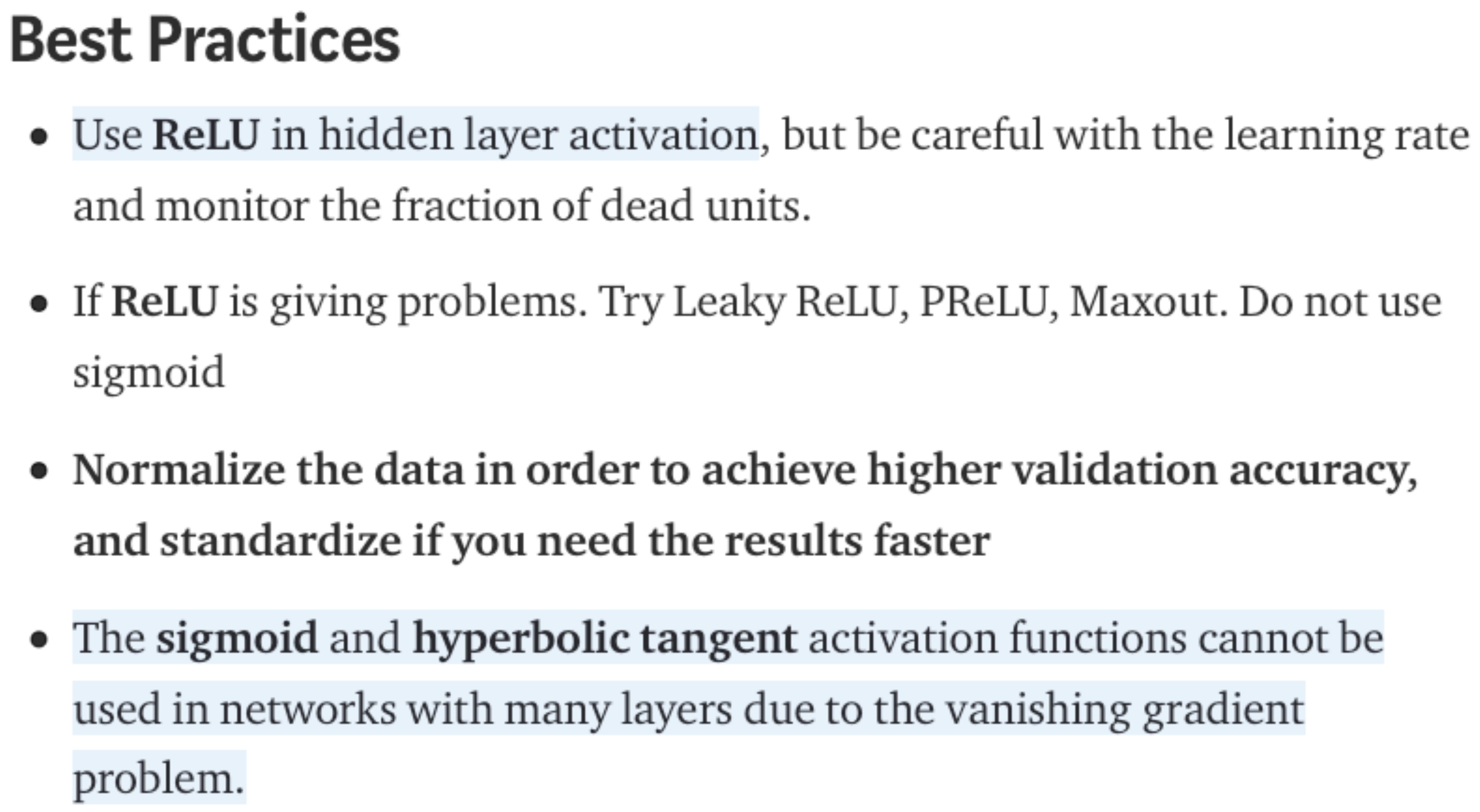
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation Function



Best Practices

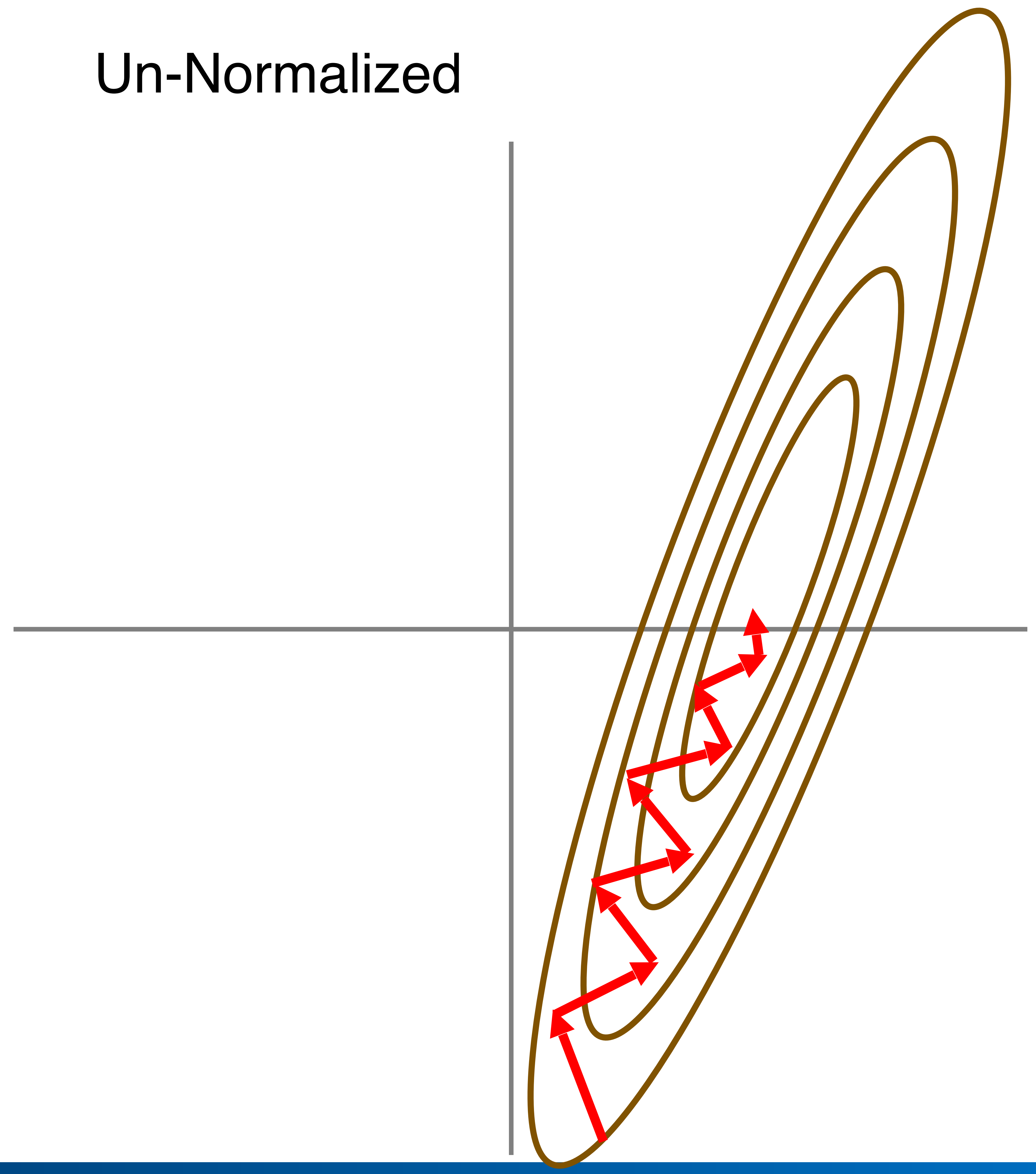
- Use **ReLU** in hidden layer activation, but be careful with the learning rate and monitor the fraction of dead units.
- If **ReLU** is giving problems. Try Leaky ReLU, PReLU, Maxout. Do not use sigmoid
- **Normalize the data in order to achieve higher validation accuracy, and standardize if you need the results faster**
- The **sigmoid** and **hyperbolic tangent** activation functions cannot be used in networks with many layers due to the vanishing gradient problem.

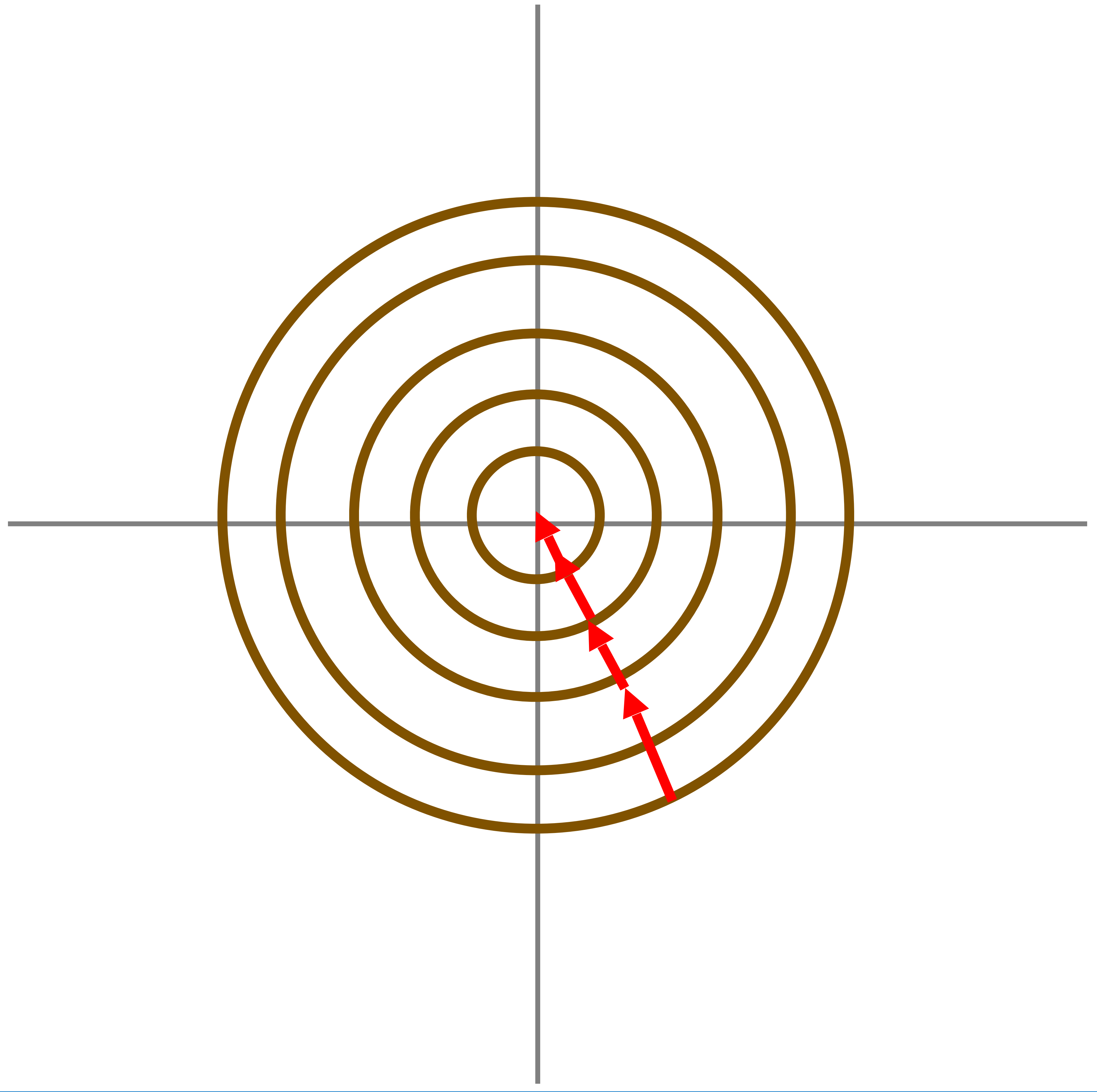
https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044

Optimization

- **AdaGrad** (Adaptive Gradient)
  - Adapt the learning rate to the parameters - large updates for infrequent params, small for frequent.
  - Problem: accumulates squared gradients in the denominator.
- **AdaDelta**
  - AdaGrad with gradient clipping - set a fixed number of accumulated past gradients.
- **ADAM** (ADAptive Moment Estimation)
  - For each parameter, calculate an adaptive learning rate.
  - Accumulate 2 exponentially decaying averages: past squared gradients and past gradients.

Normalization





Overhead view of cost curves

Shifting and rescaling

To have inputs range [0, 1]:

$$x_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

To have inputs range [-1, 1]:

$$x_i = 2\left(\frac{x_i - x_{min}}{x_{max} - x_{min}}\right) - 1$$

Normalizing

Mean zero, standard deviation one

$$x_i = \frac{x_i - \bar{x}}{\sigma}; \qquad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}$$

Regularization

- **Goal:** prevent overfitting
    - *Network should generalize beyond the data!*

- How?
    - One way: prevent large weights from dominating *(penalize them)*

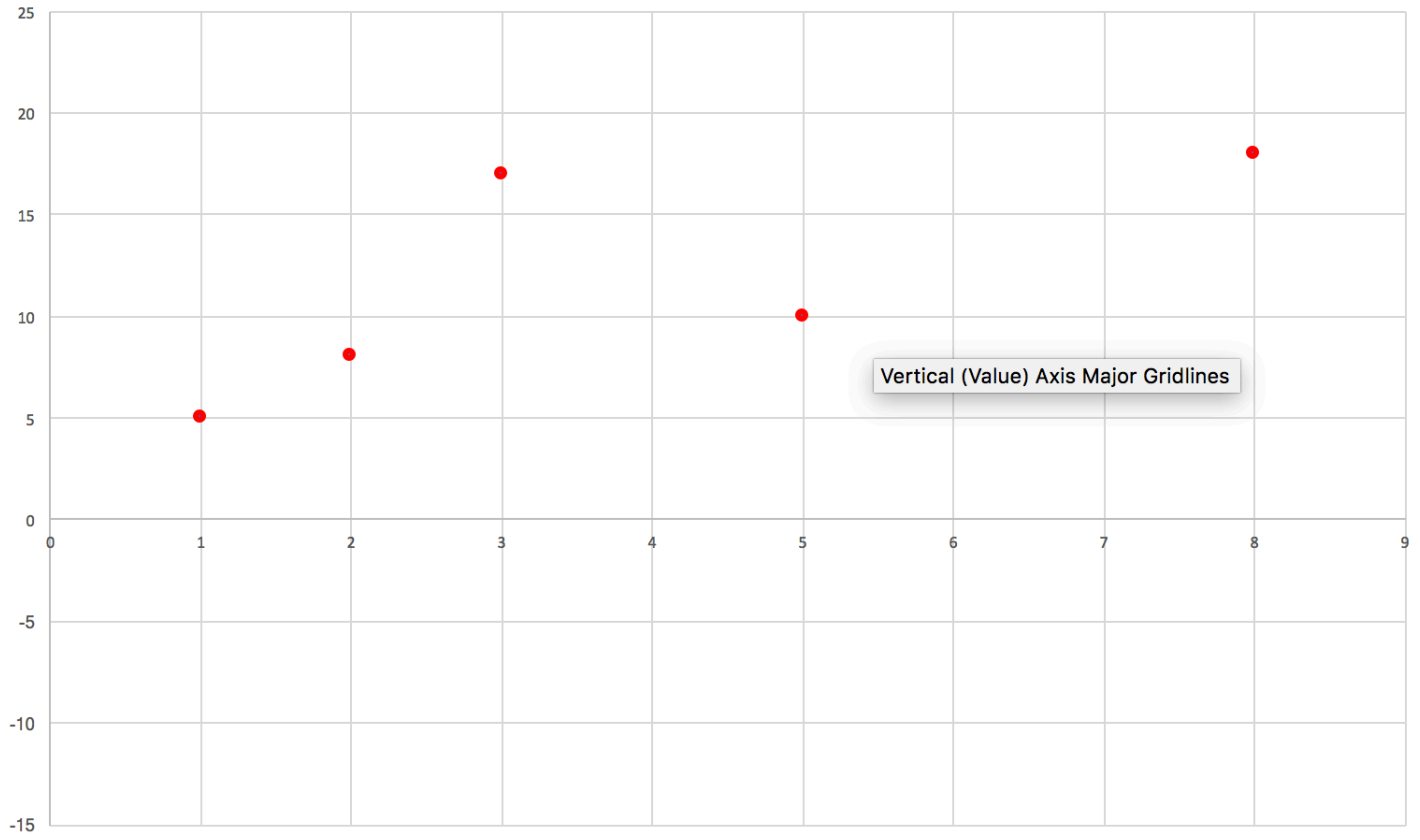
L2 Regularization

Adjust loss function to include a second term

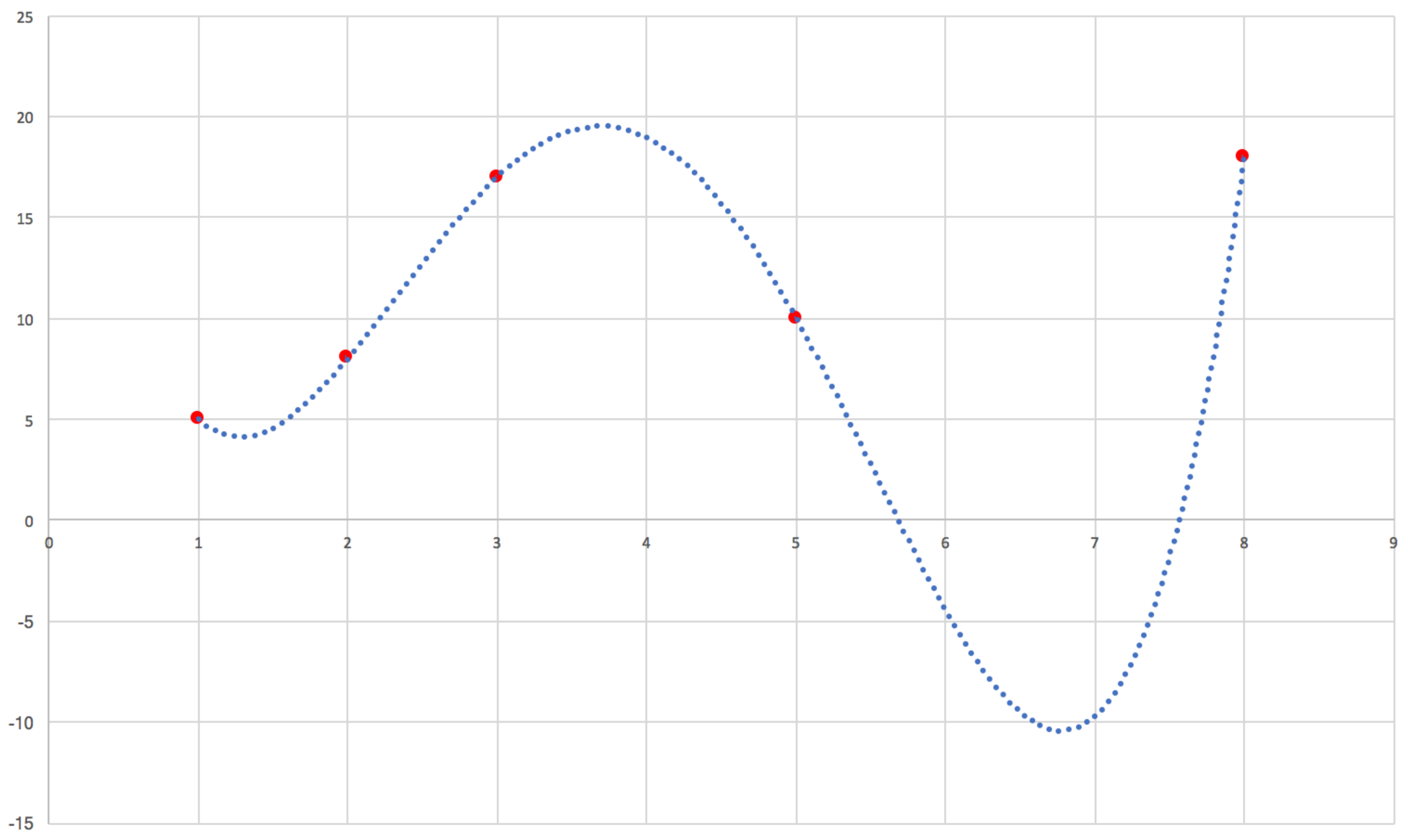
$$J = \frac{1}{2n} \sum_{i=1}^{n} \left(\hat{y}_i - y_i \right)^2 + \lambda \sum_{j=1}^{m} w_j^2$$

- Lambda is the **regularization *hyper-parameter***
  - *Bigger means more regularization*
  - *More penalty for big weights*
  - *Less attention to raw data fit*

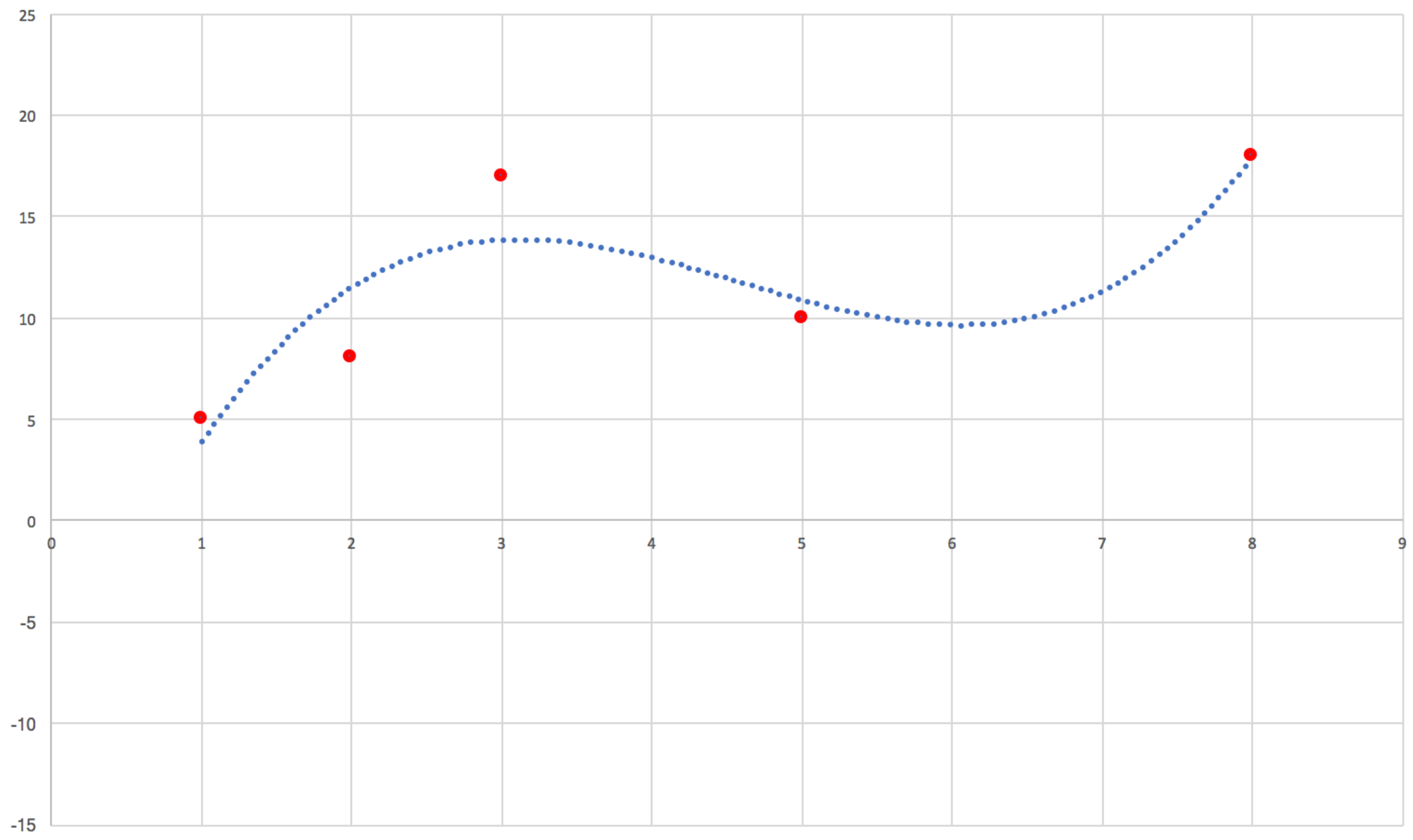
Scatter Plot



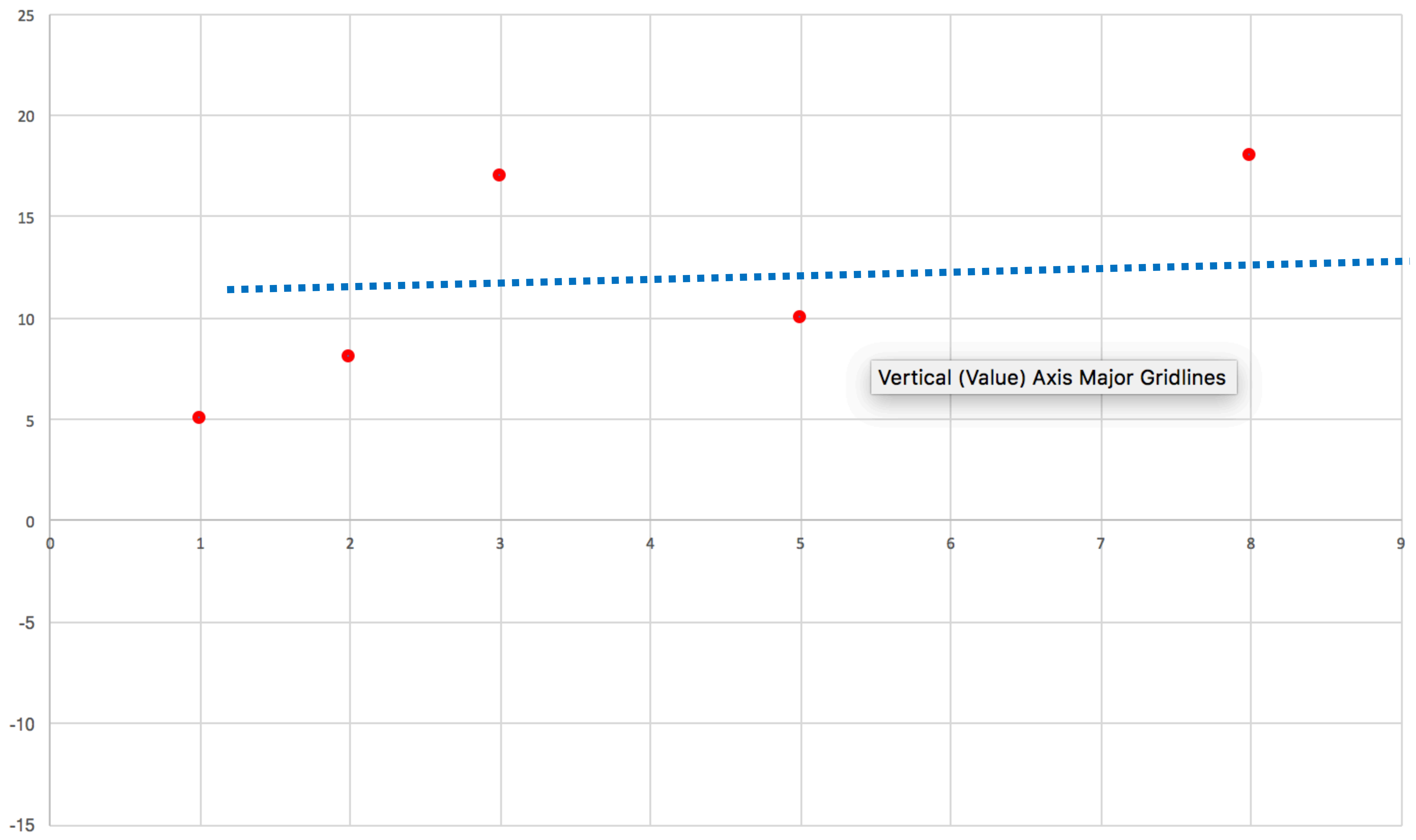
Overfitting



Possible model after regularization



Too much regularization!



Batching

Full batch gradient descent

So far in this class, we've passed the full training dataset into our model for each training step

Pro: we get an *exact* derivative for the training set

Con: Right now- it's *really* slow. Soon, it won't be computationally feasible.

What can we do?

Mini-batching

Idea: Estimate derivative with a sample of training data

Pro: *much* faster to compute

Con: don’t get the *exact* derivative

Side effect: can potentially get out of local minima, if loss function has them

Stochastic Gradient Descent (SGD)

At an extreme, use a single example to estimate derivative

- Mini-batch with a size of 1

Also known as online training

Movement along the error curve is much more sporadic

But also much easier to compute

Batching terminology cheat-sheet

Full-batch gradient descent

- Use all of training data per step

Mini-batch gradient descent

- Use small portion of training data per step

Stochastic gradient descent (SGD)

- Use single example per step
- Occasionally refers to mini-batch gradient descent

Epochs

An *epoch* refers to one entire pass through the dataset

- The number of times a model has seen each training example

Shuffling

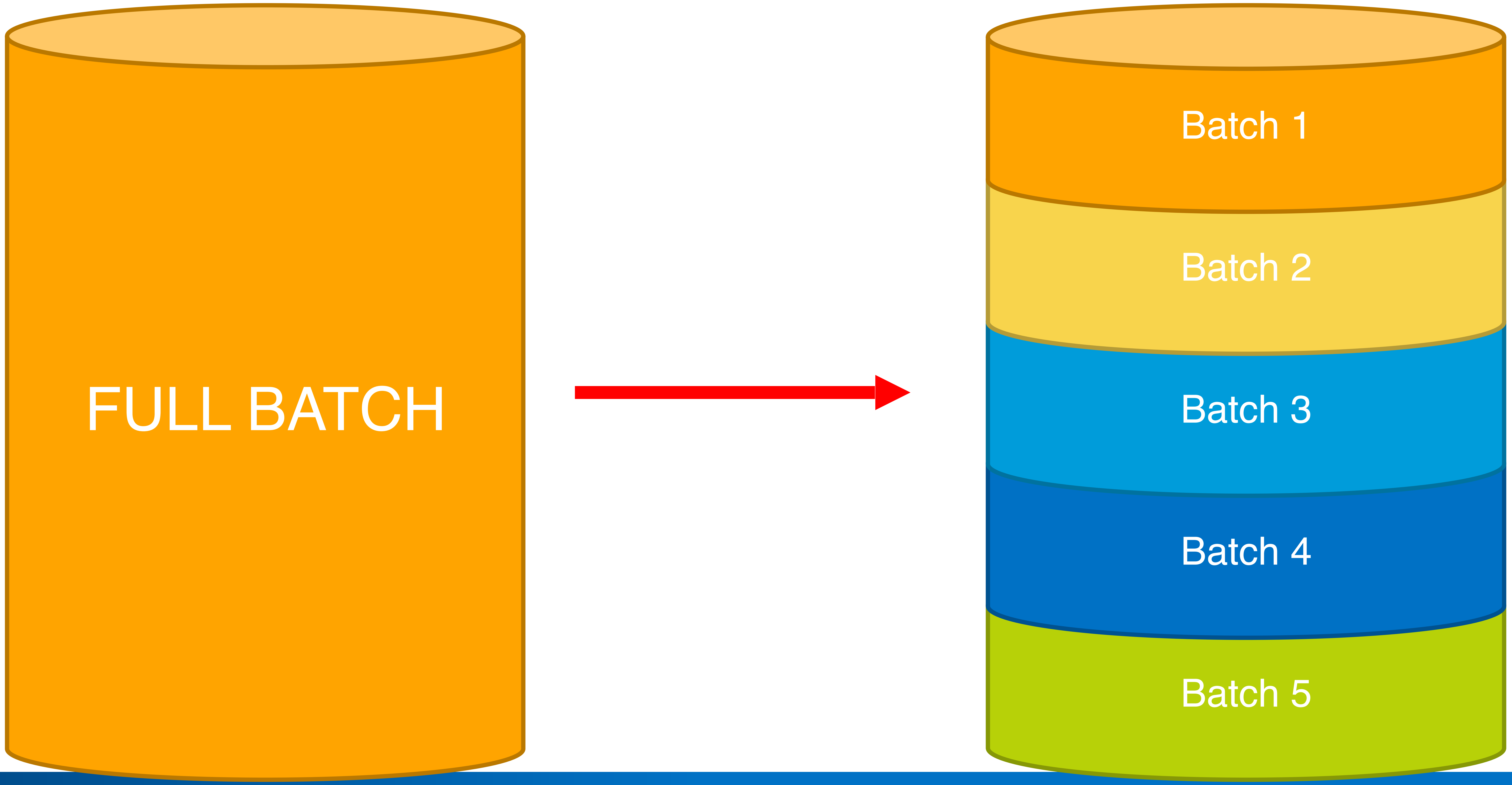
If we are using our data piecemeal, we must be careful:

- Don’t want to oversample
- Avoid reusing same mini-batch over and over

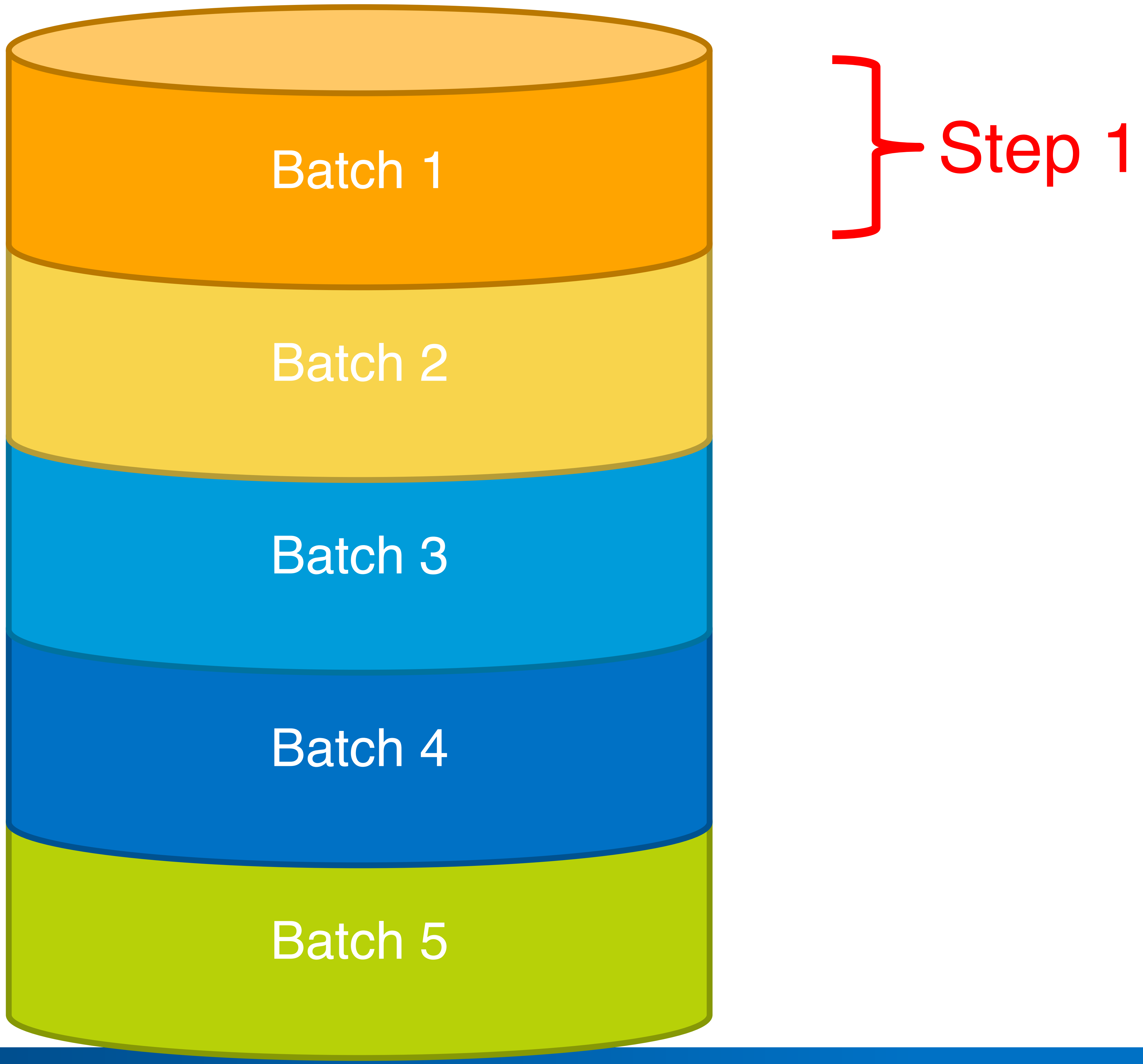
Solution: shuffle training data after each epoch

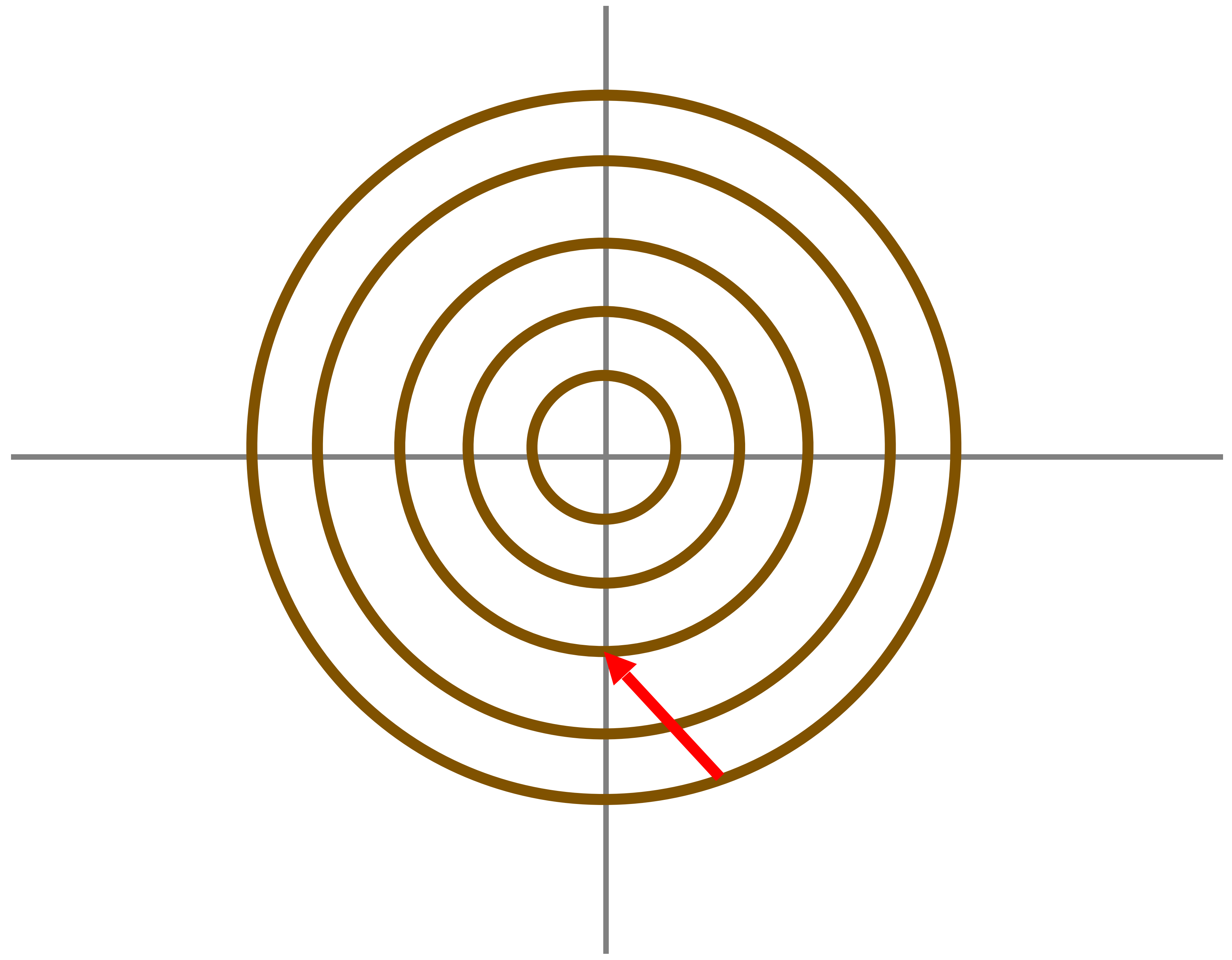
- Shuffle, make batches, repeat

Splitting data up into batches

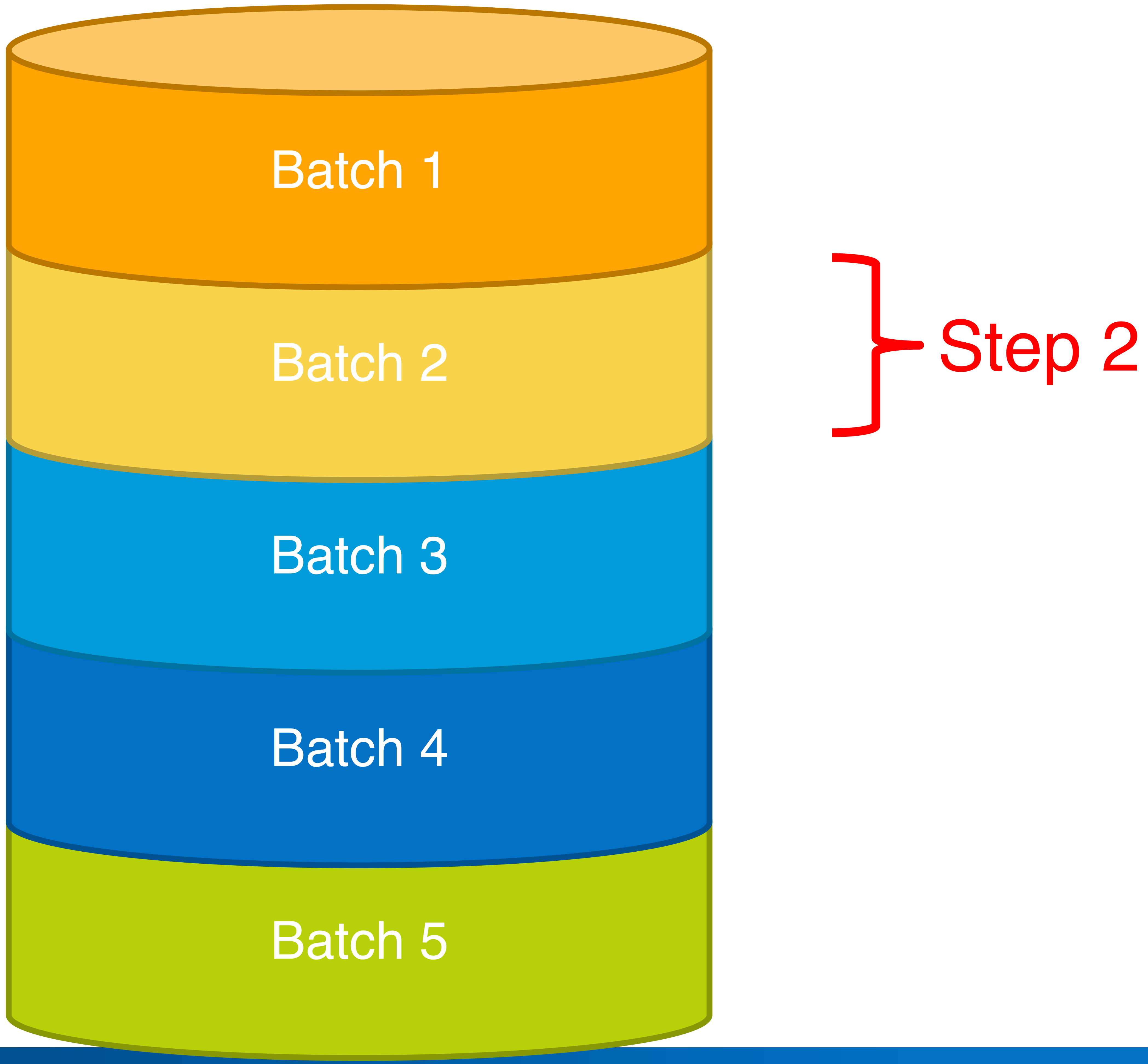


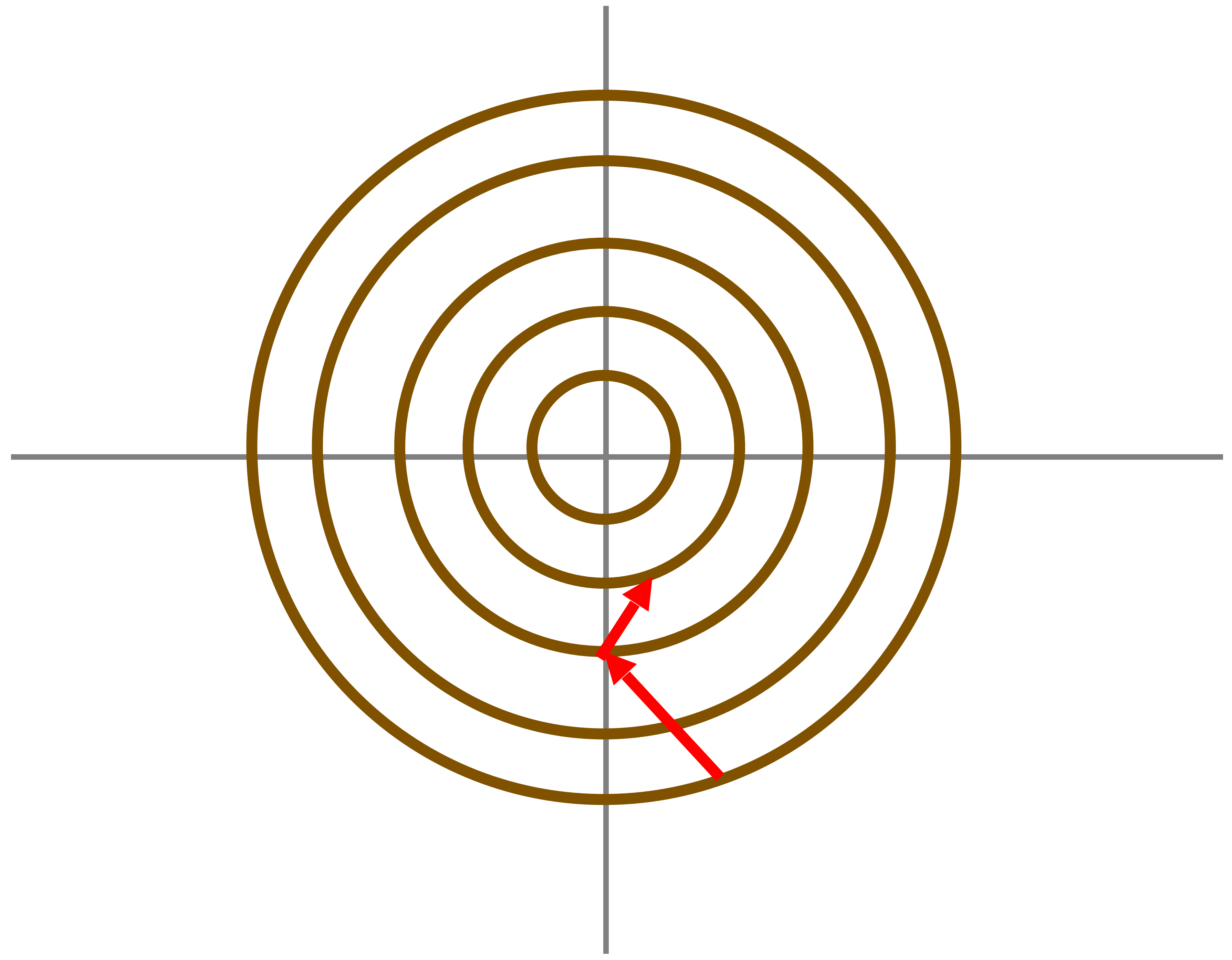
Splitting data up into batches





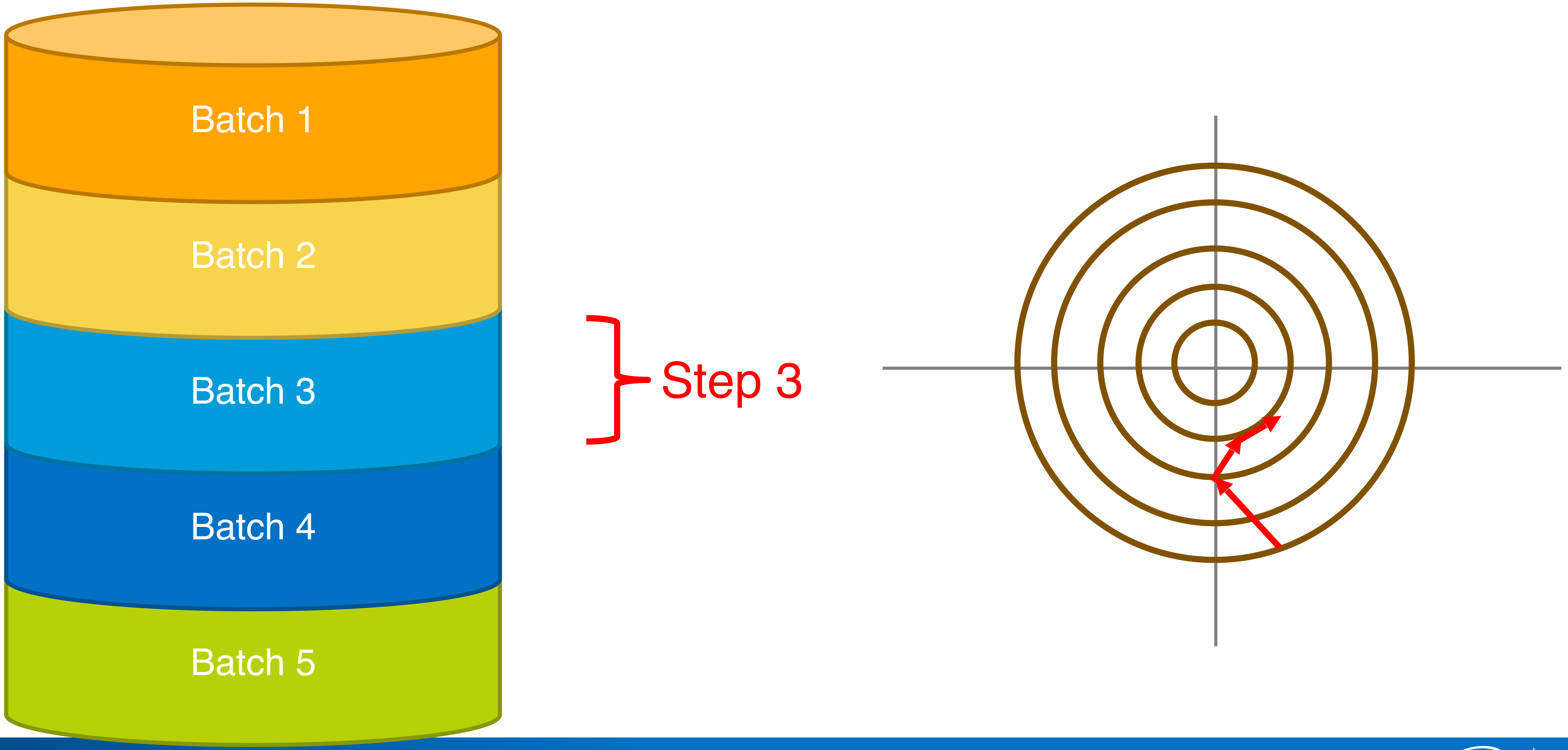
Splitting data up into batches



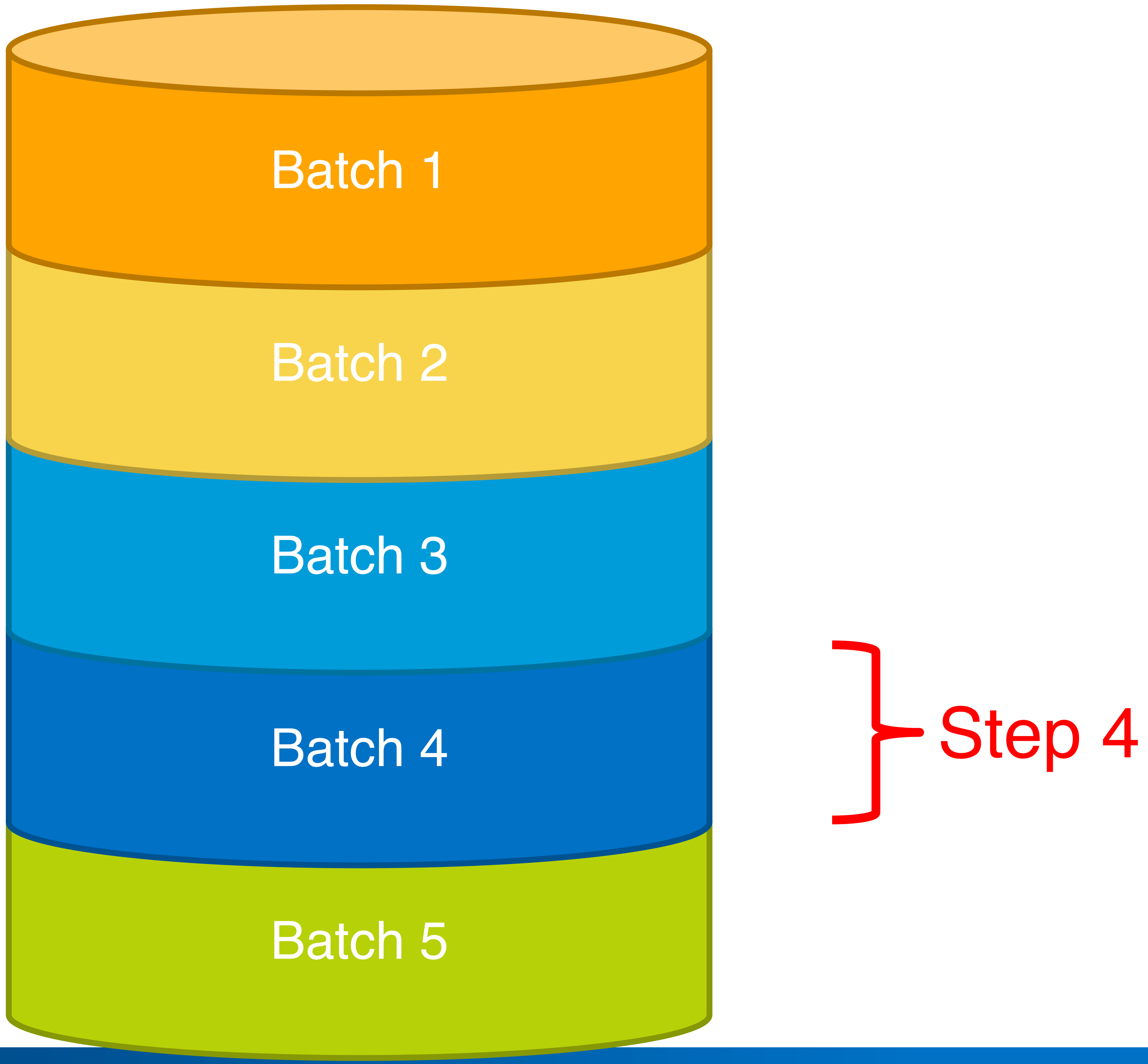


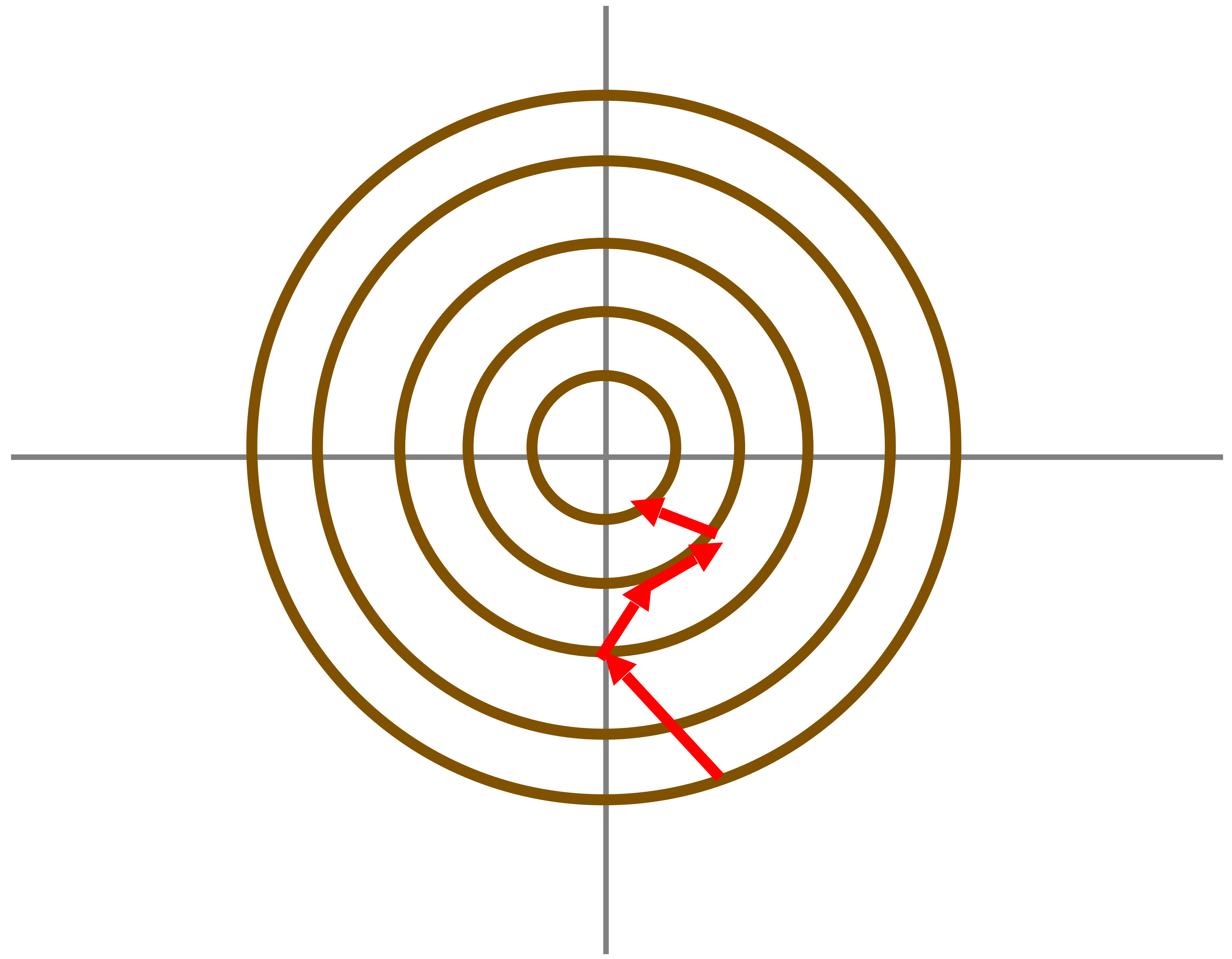
intel Software

Splitting data up into batches

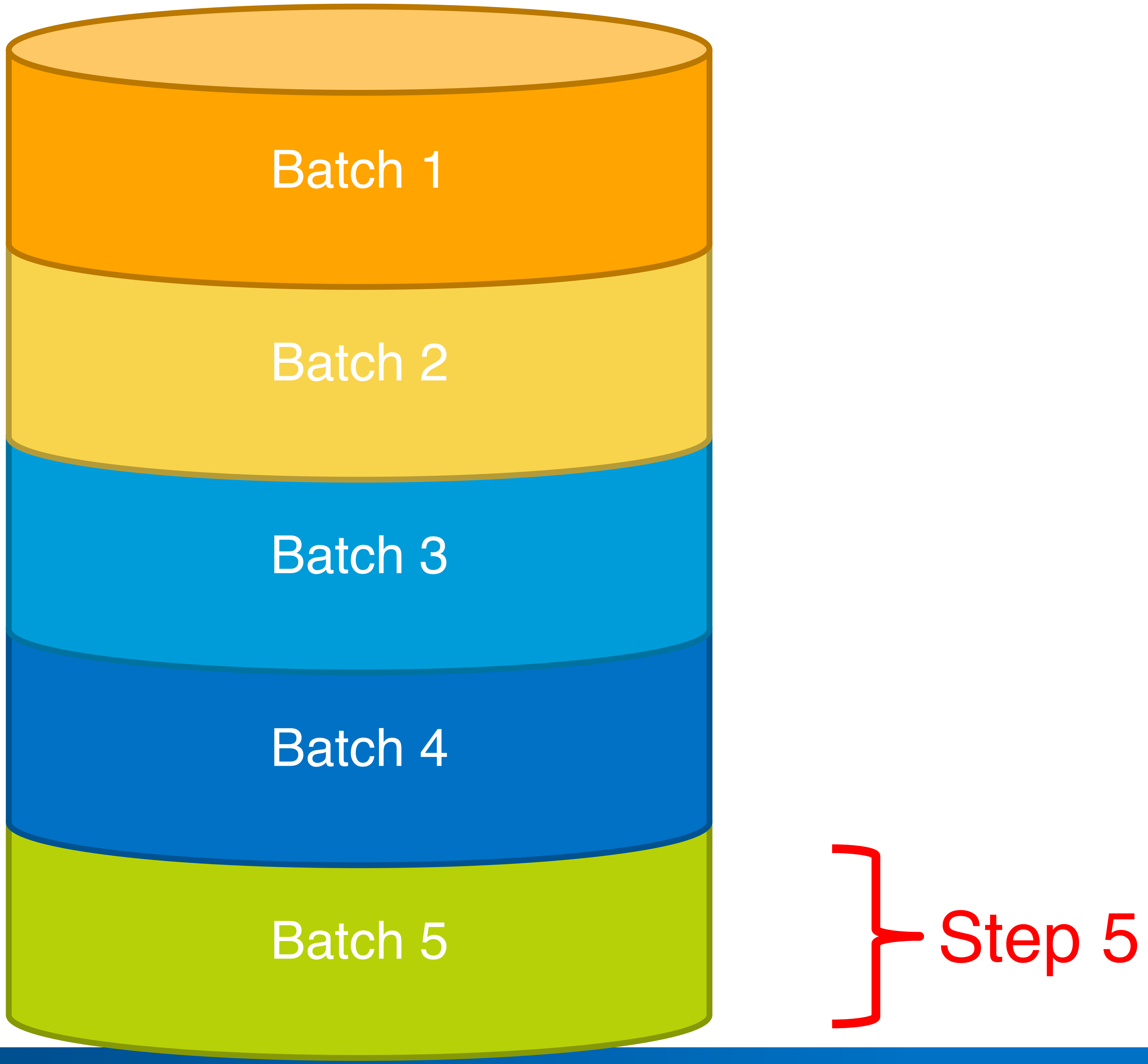


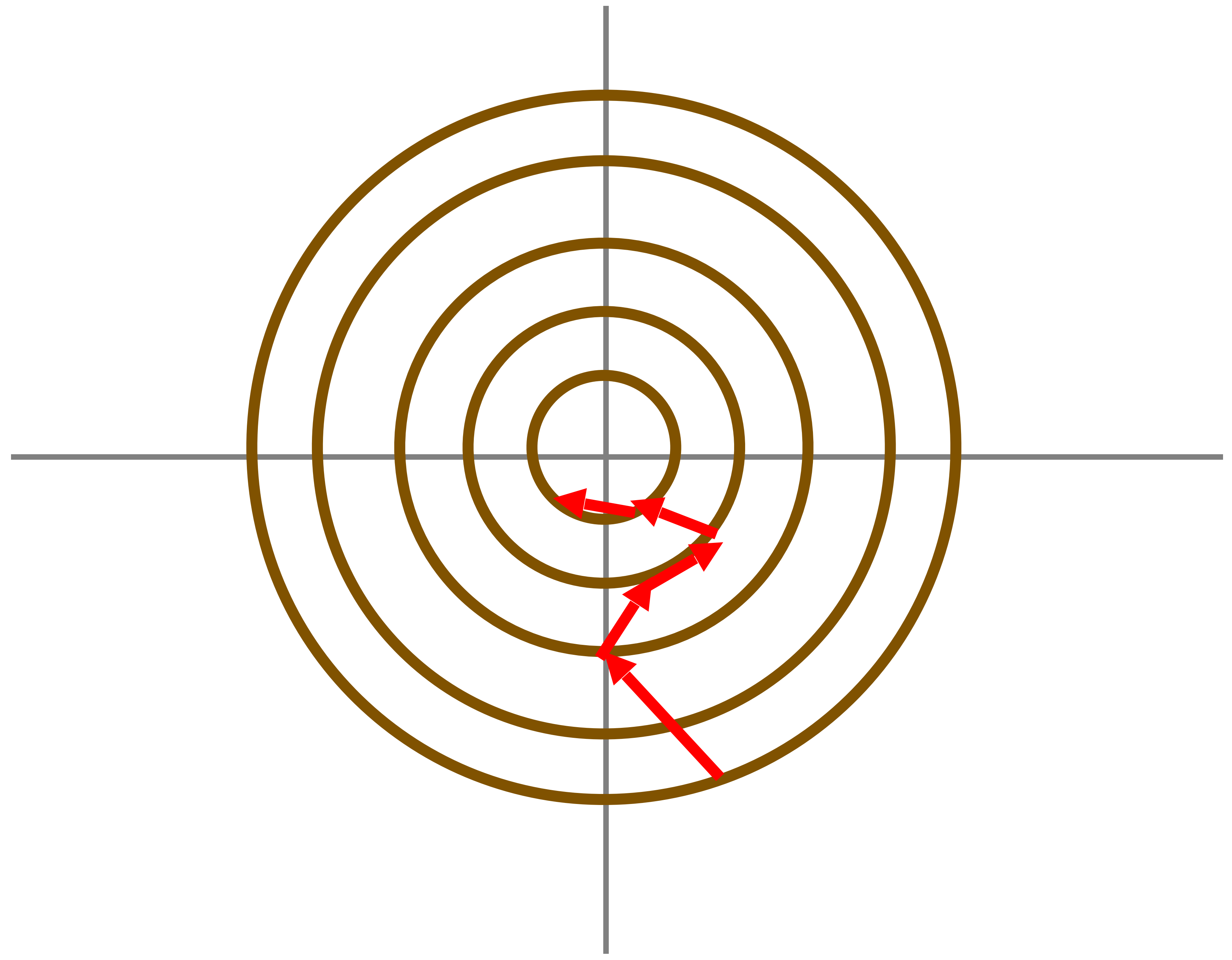
Splitting data up into batches



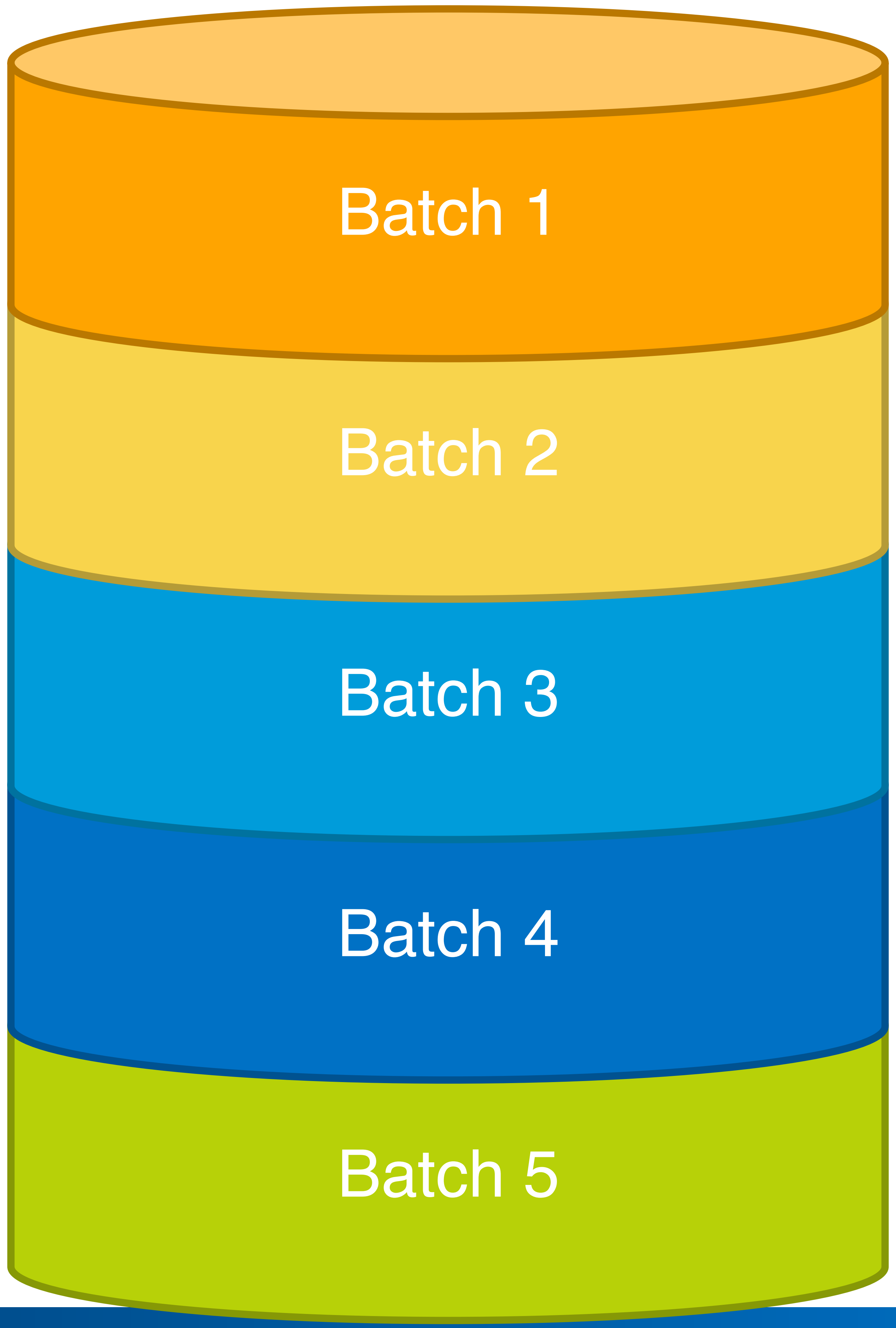


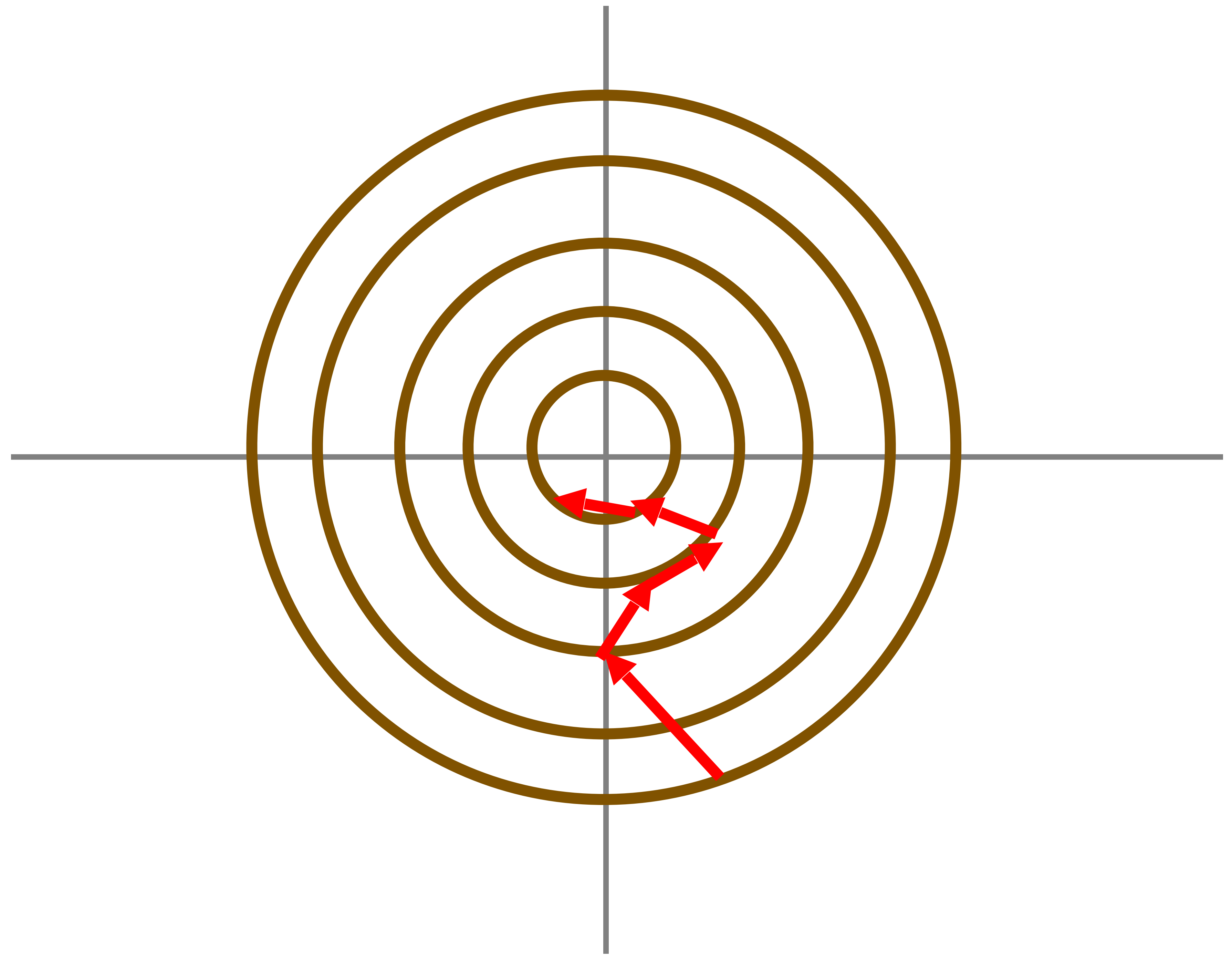
Splitting data up into batches



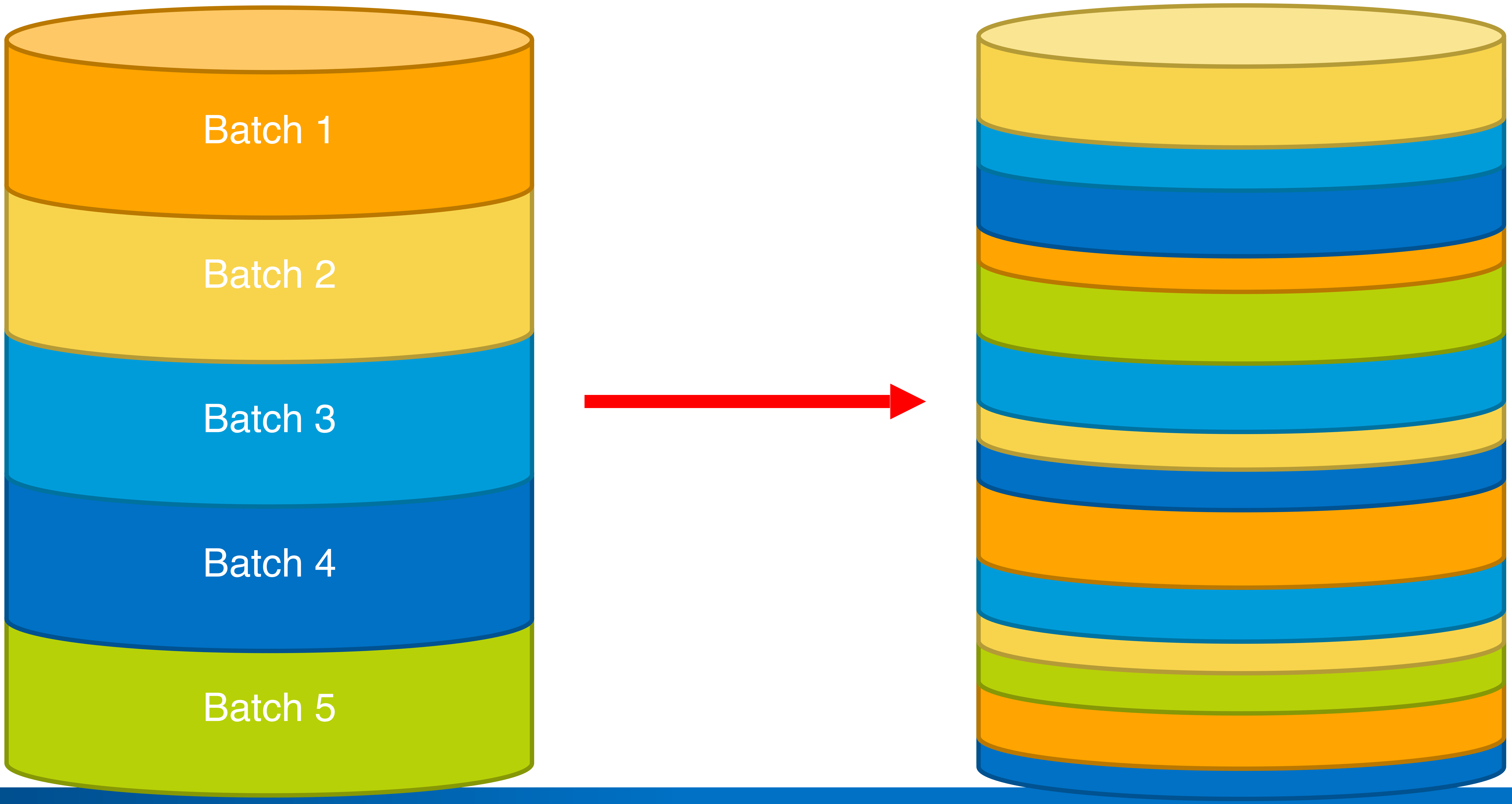


First epoch completed

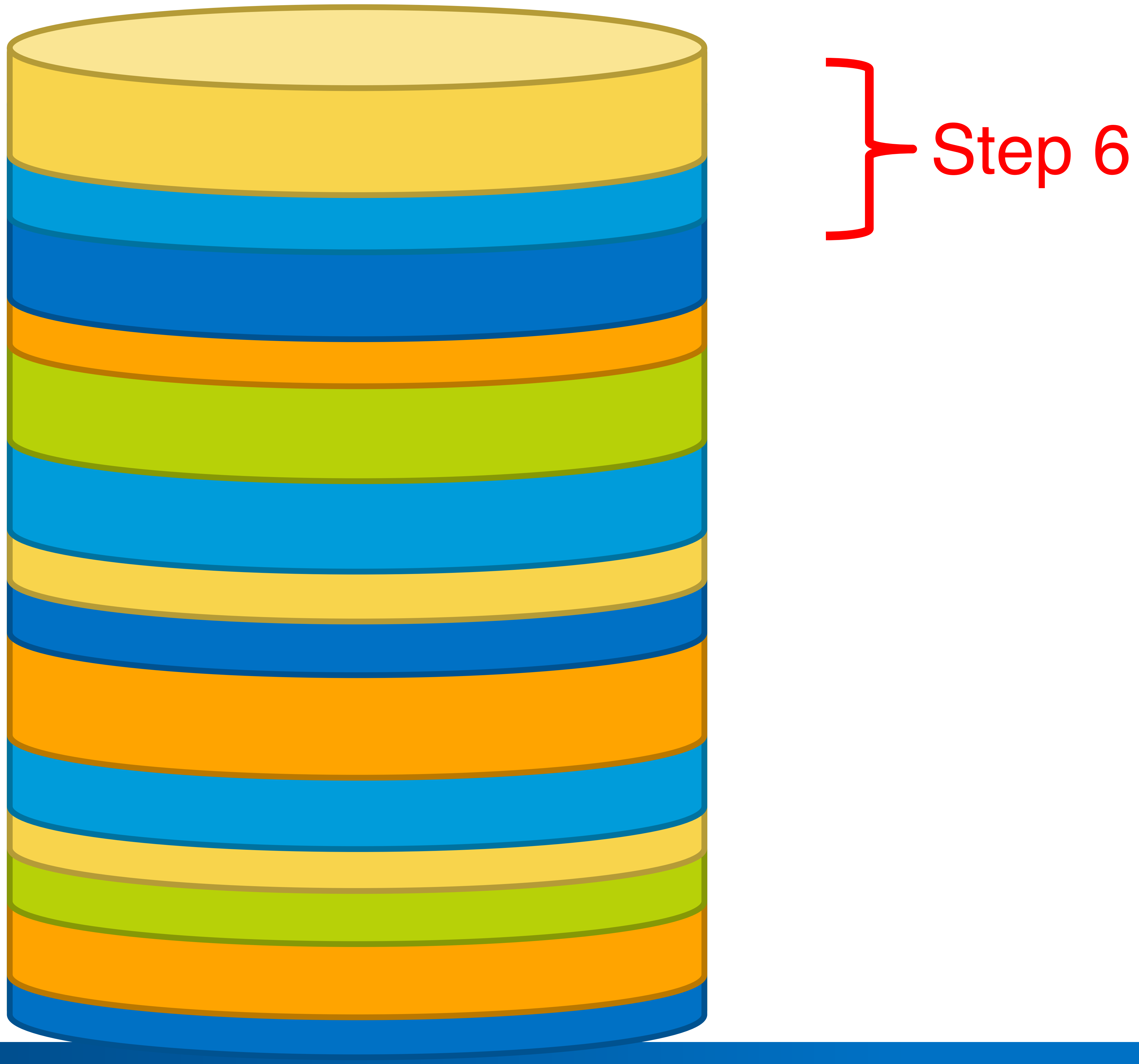


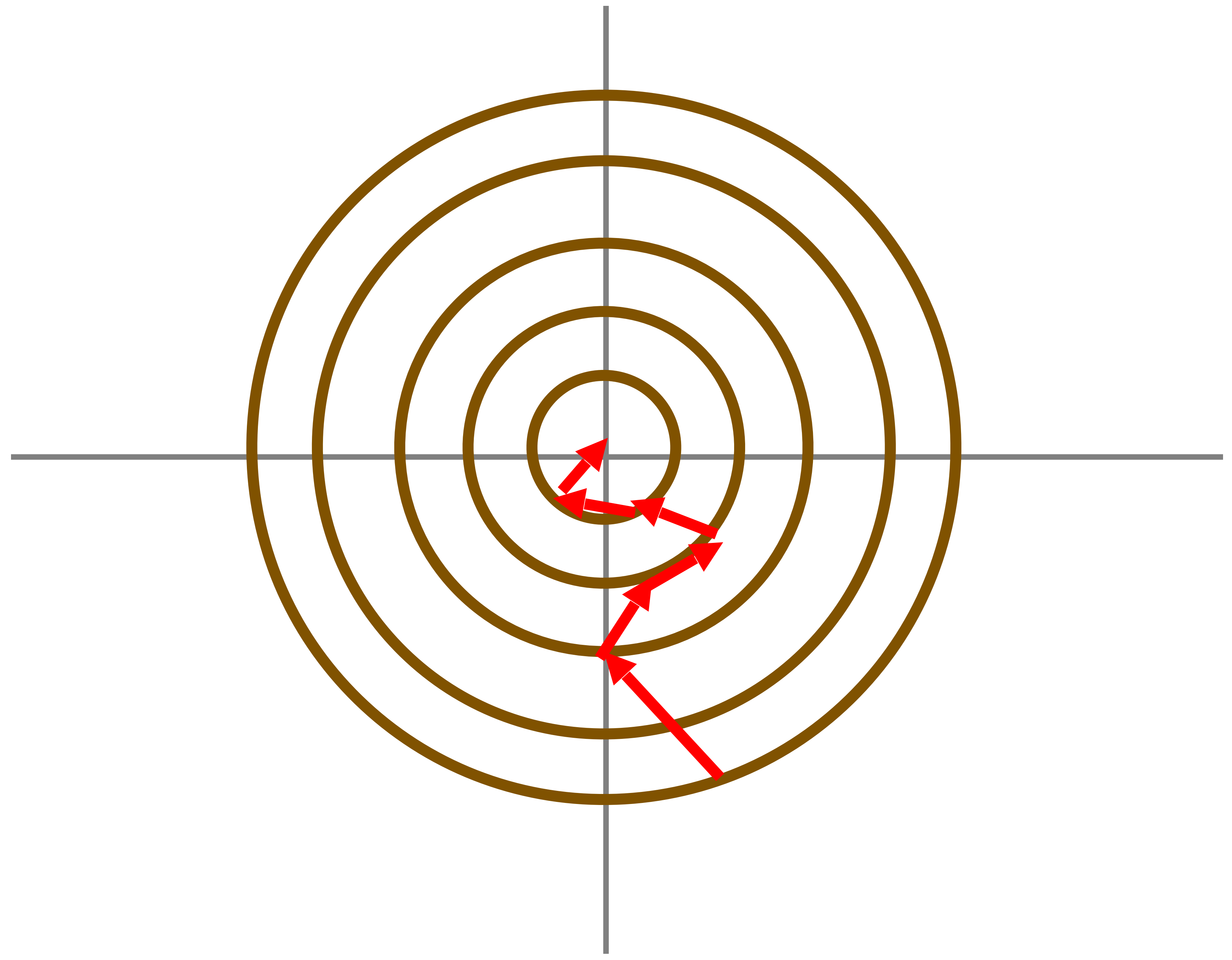


Shuffle the data!



Continue training





Summary

Lesson 13:
Introduction to Deep Learning

- Optimization
- Training
- Regularization

Next week:

- Convolutional neural networks