Details of Training

Neural Nets

What next?

- Given an example (or group of examples), we know how to compute the derivative for each weight.
- How exactly do we update the weights?
- How often? (after each training data point? after all the training data points?)

What next? – Gradient Descent

- W_new = W_old - lr * derivative

- Classical approach – get derivative for entire data set, then take a step in that direction

- Pros: Each step is informed by all the data
- Cons: Very slow, especially as data gets big

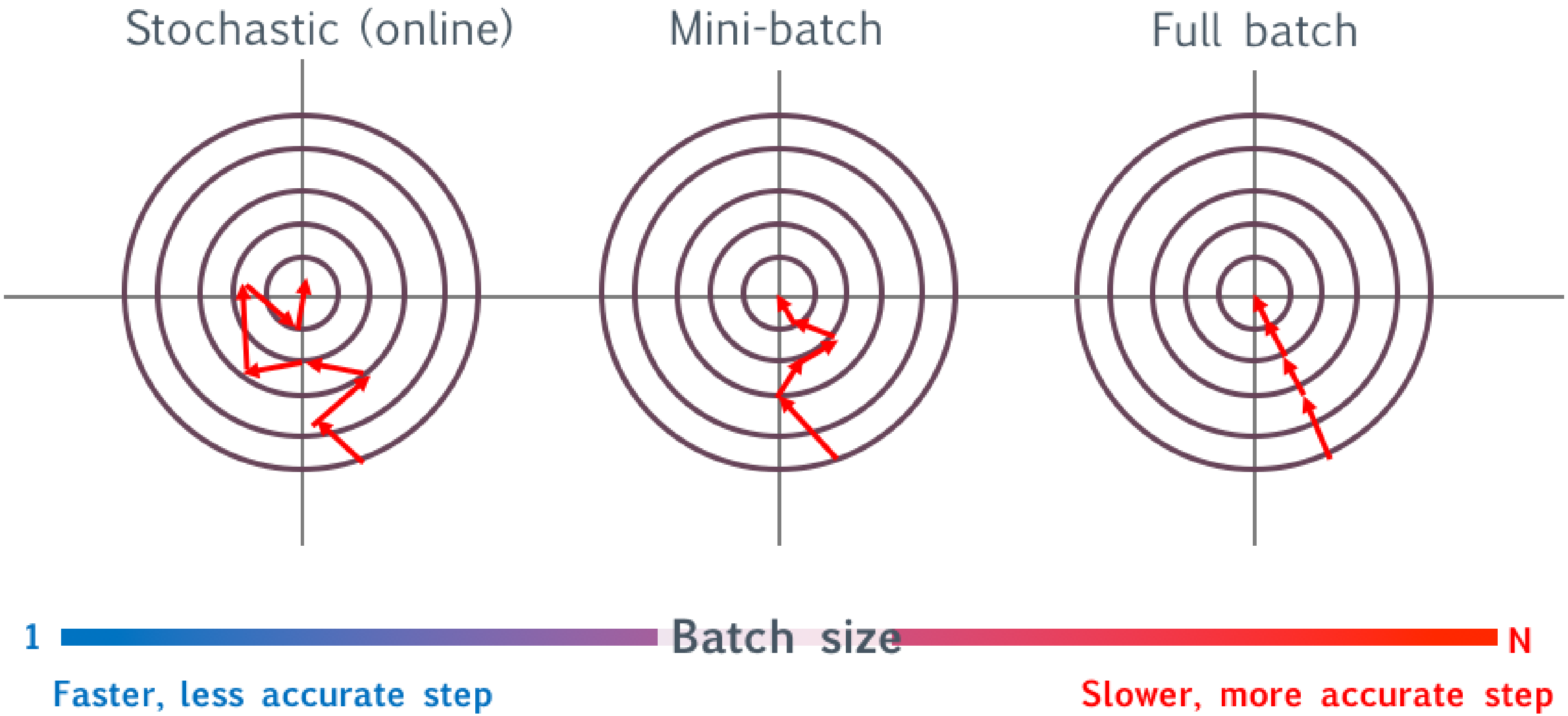
Another approach: Stochastic Gradient Descent

- Get derivative for just one point, and take a step in that direction
- Steps are “less informed” but you take more of them
- Should “balance out”
- Probably want a smaller step size
- Also helps “regularize”

Compromise approach: Mini-batch

- Get derivative for a ”small” set of points, then take a step in that direction
- Typical mini batch sizes are 16, 32
- Strikes a balance between two extremes

Comparison of Batching Approaches



Batching Terminology

- Full-batch: Use entire data set to compute gradient before updating
- Mini-batch: Use a smaller portion of data (but more than single example) to compute gradient before updating
- Stochastic Gradient Descent (SGD): Use a single example to compute gradient before updating (though sometimes people use SGD to refer to minibatch, also)

Batching Terminology

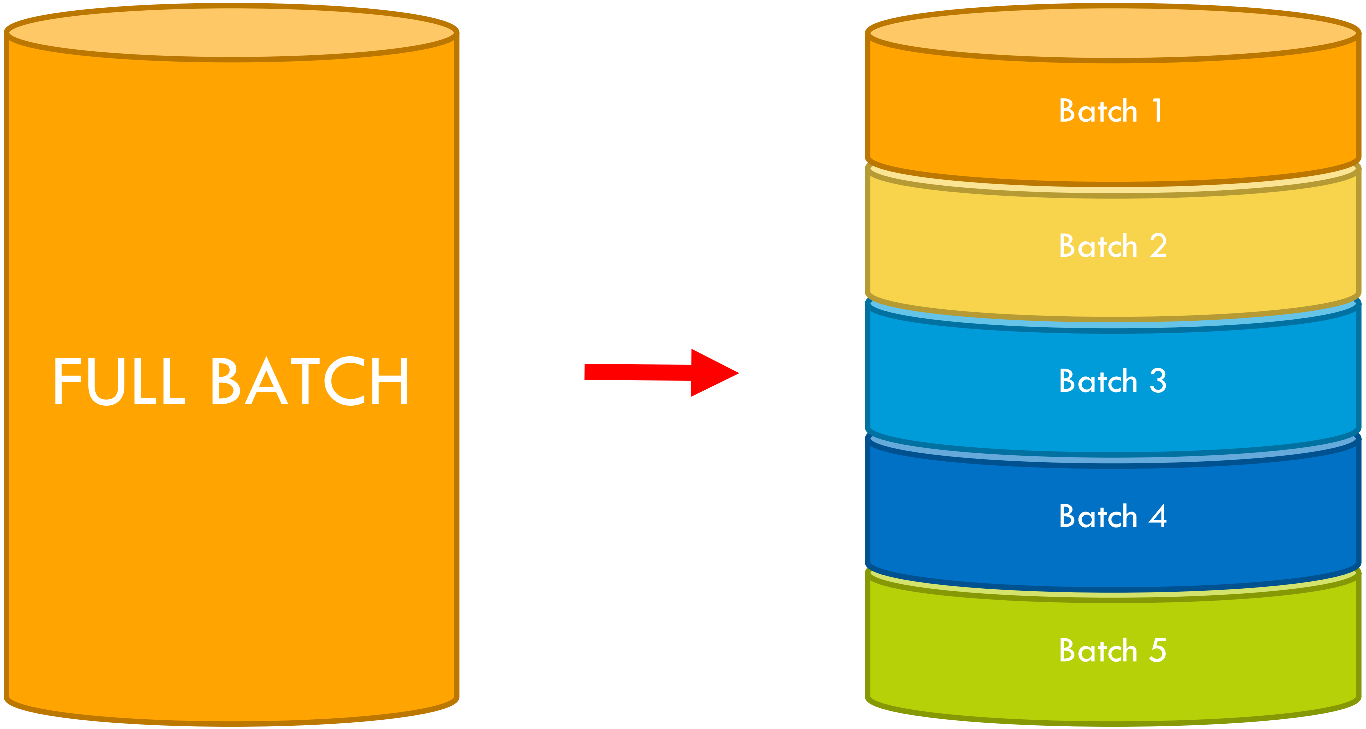
- An Epoch refers to a single pass through all of the training data.
- In full batch gradient descent, there would be one step taken per epoch.
- In SGD / Online learning, there would be n steps taken per epoch (n = training set size)
- In Minibatch there would be (n/batch size) steps taken per epoch
- When training, it is common to refer to the number of epochs needed for the model to be “trained”.

Note on Data Shuffling

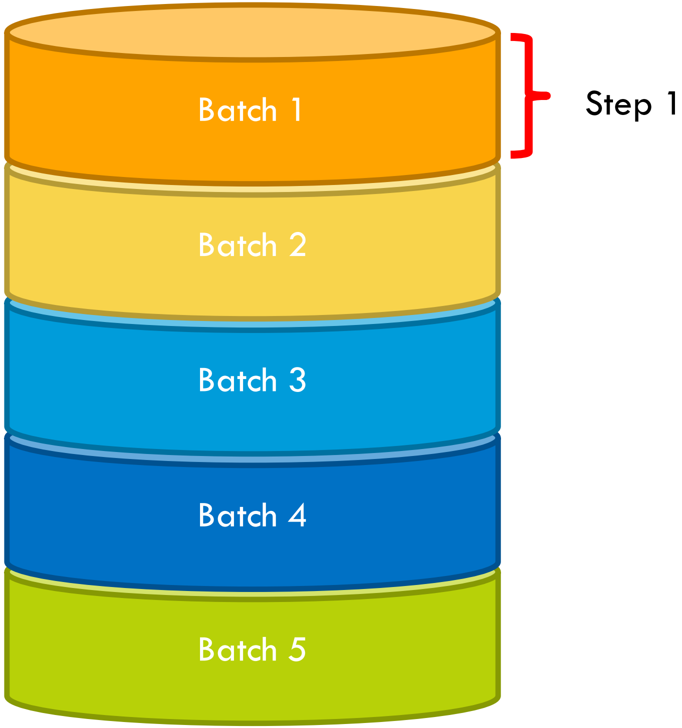
- To avoid any cyclical movement and aid convergence, it is recommended to shuffle the data after each epoch.

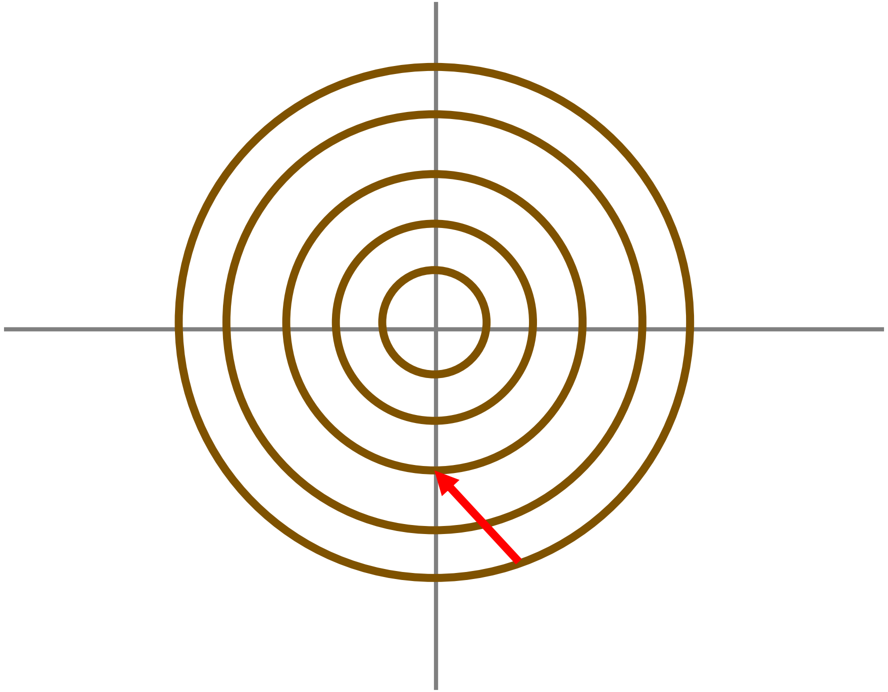
- This way, the data is not seen in the same order every time, and the batches are not the exact same ones.

Feedforward Neural Network

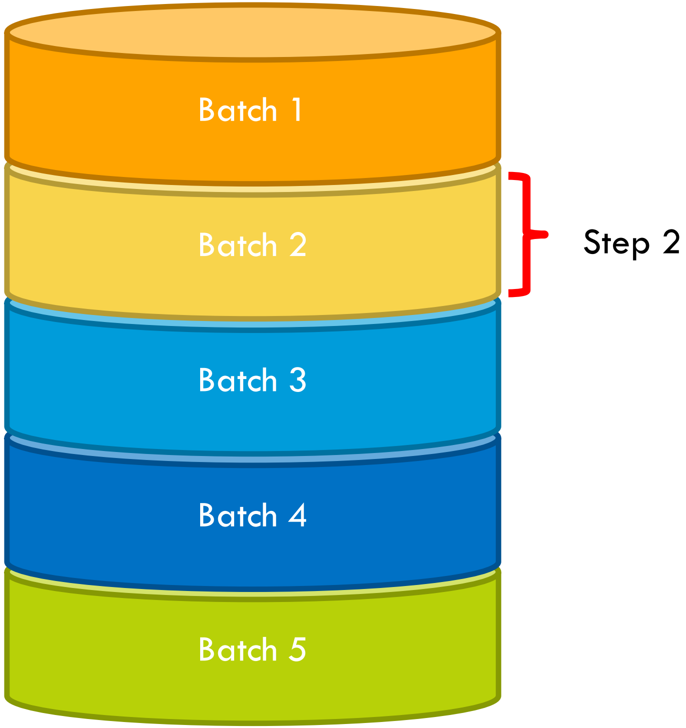


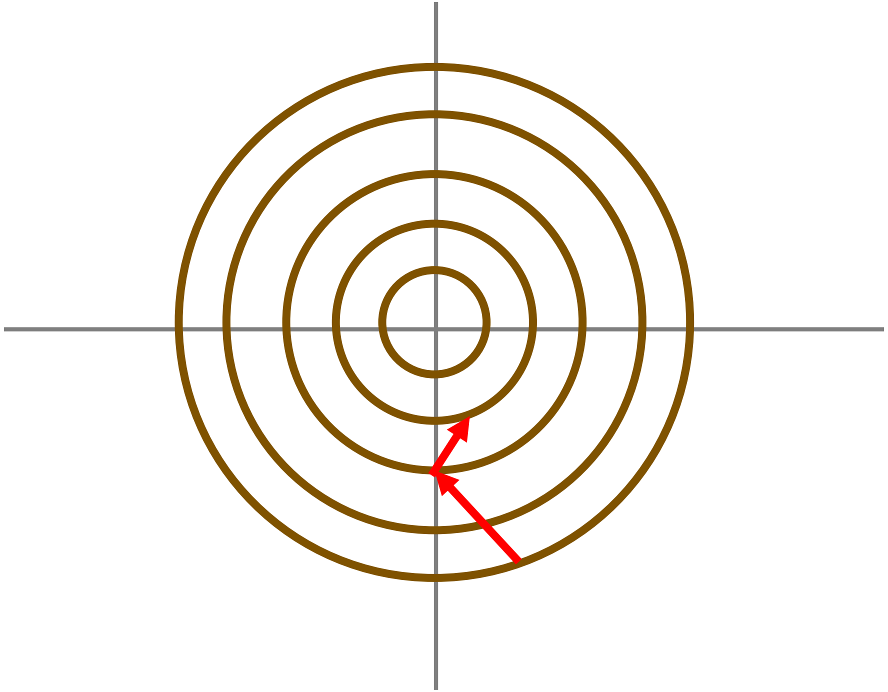
Training in Action



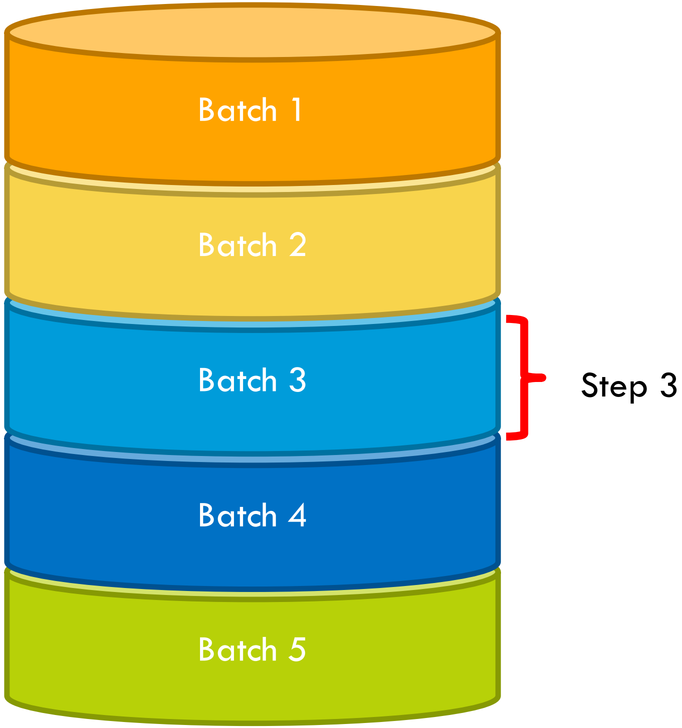


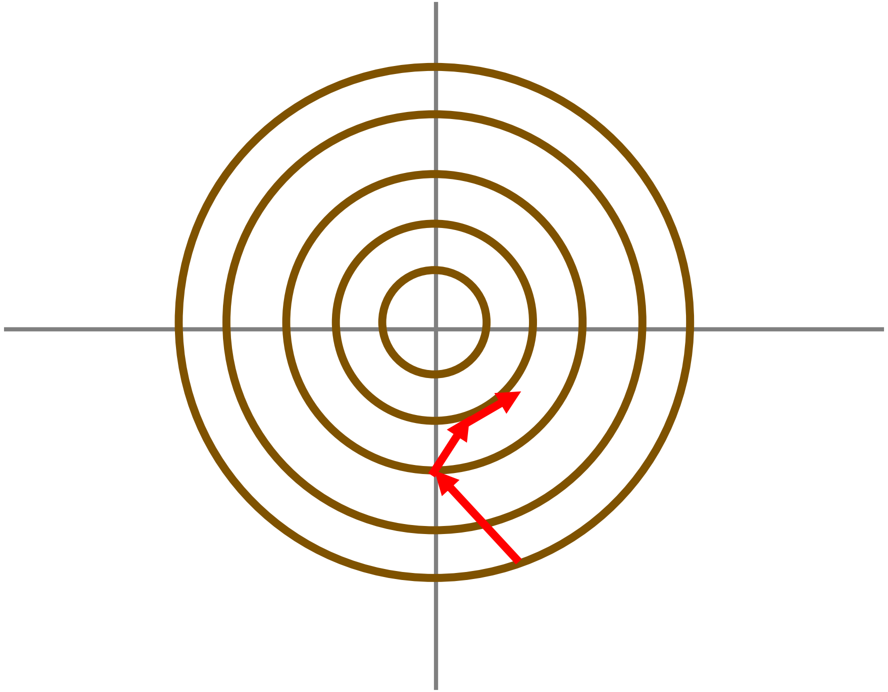
Training in Action



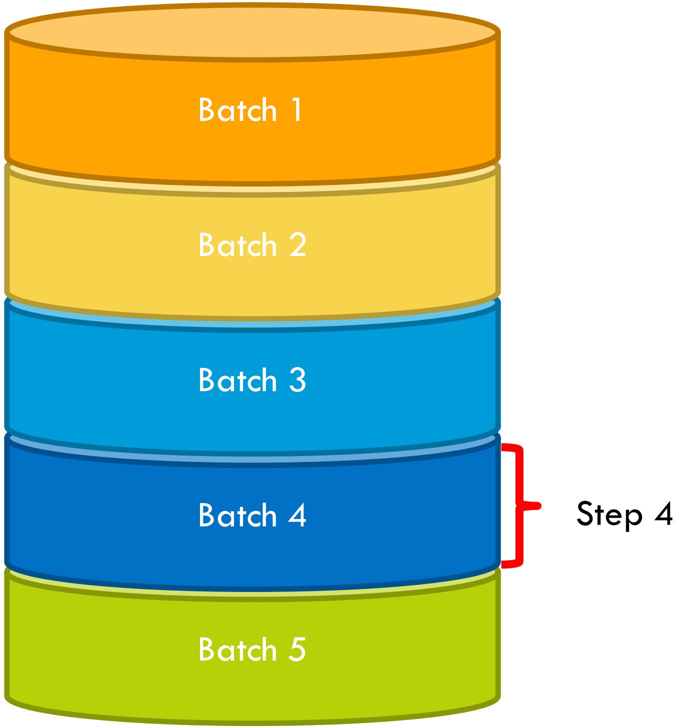


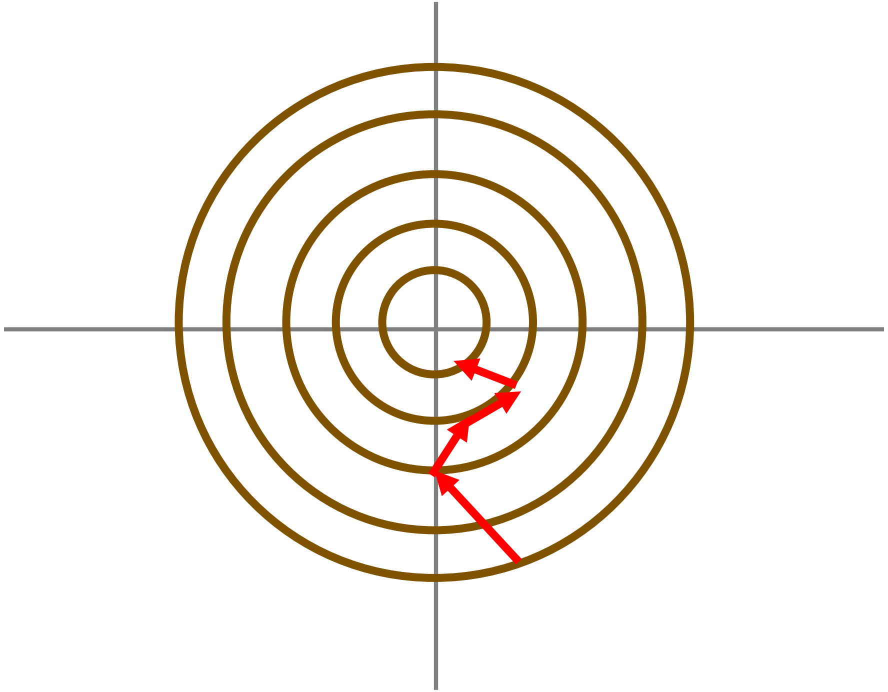
Training in Action



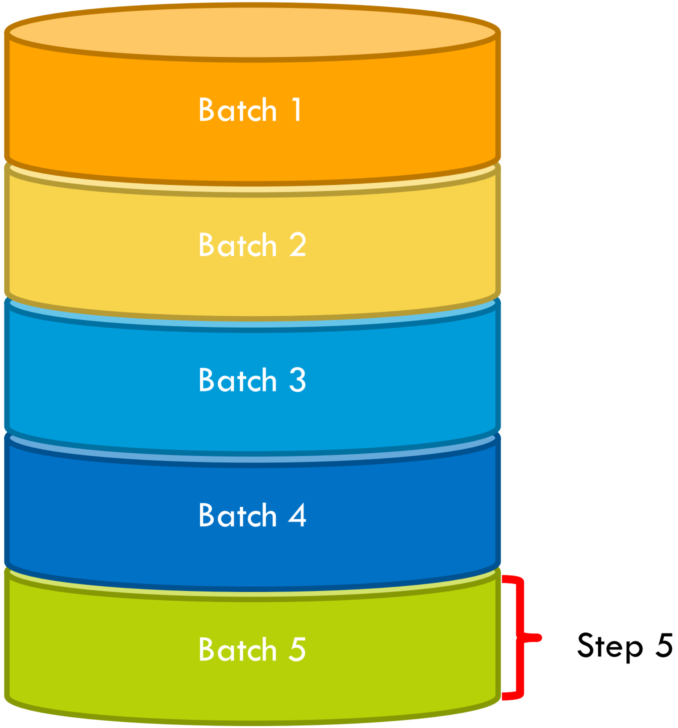


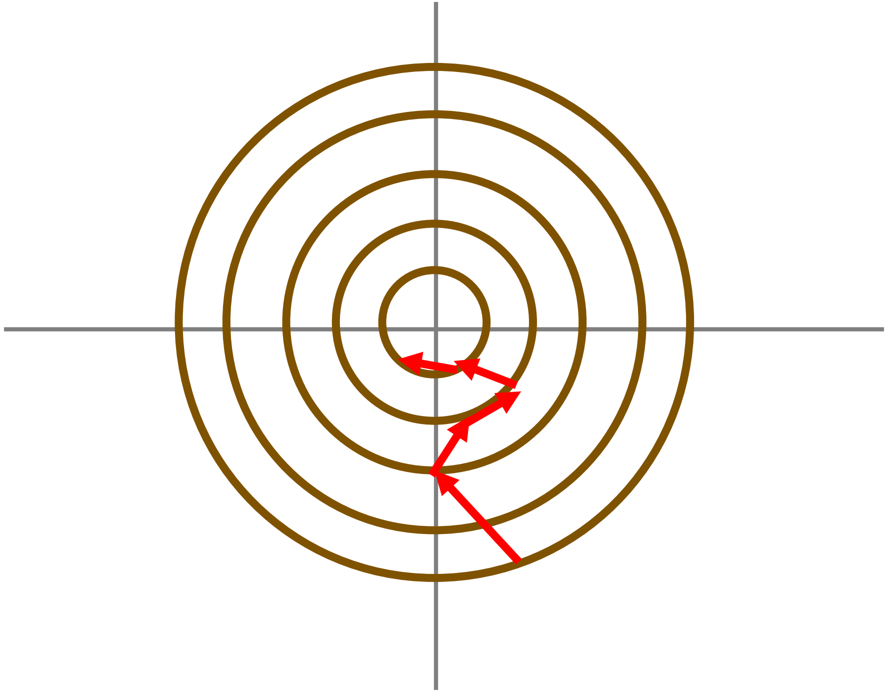
Training in Action



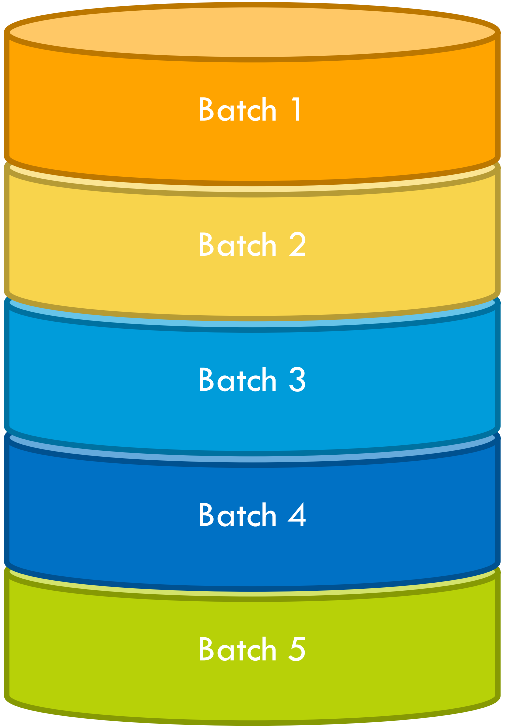


Training in Action

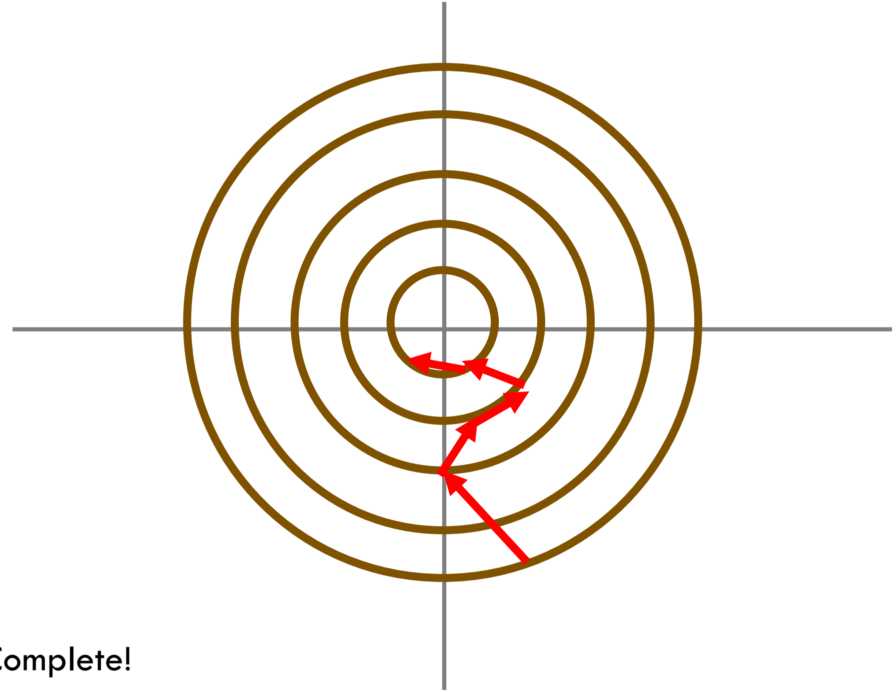




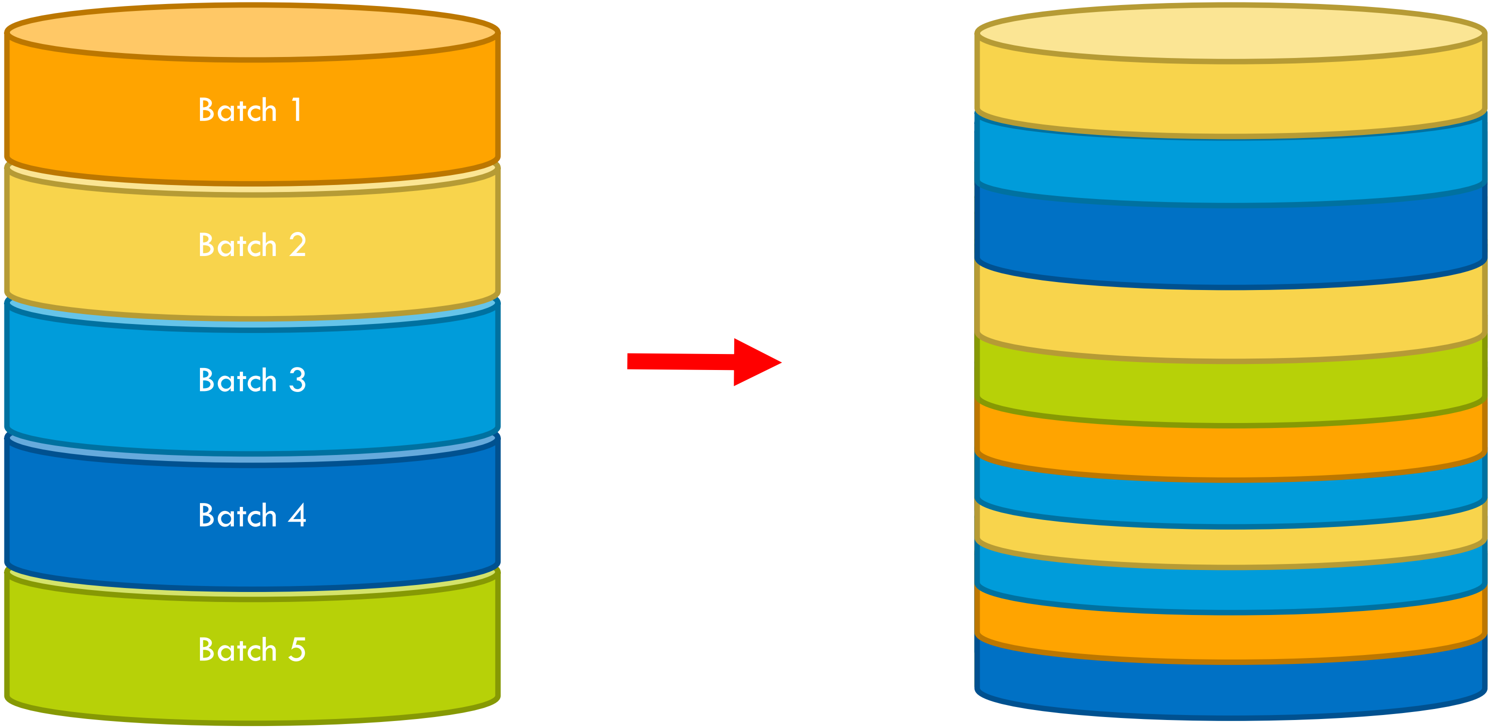
Training in Action



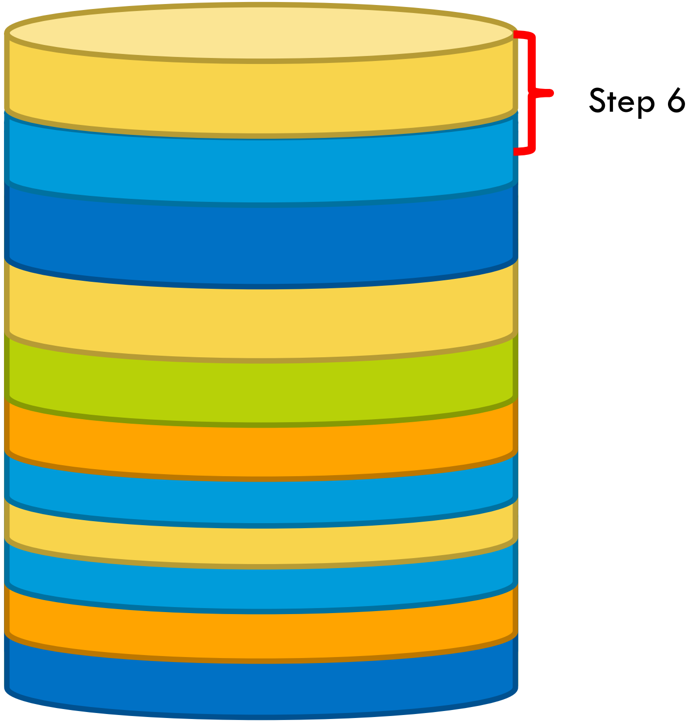
First Epoch Complete!

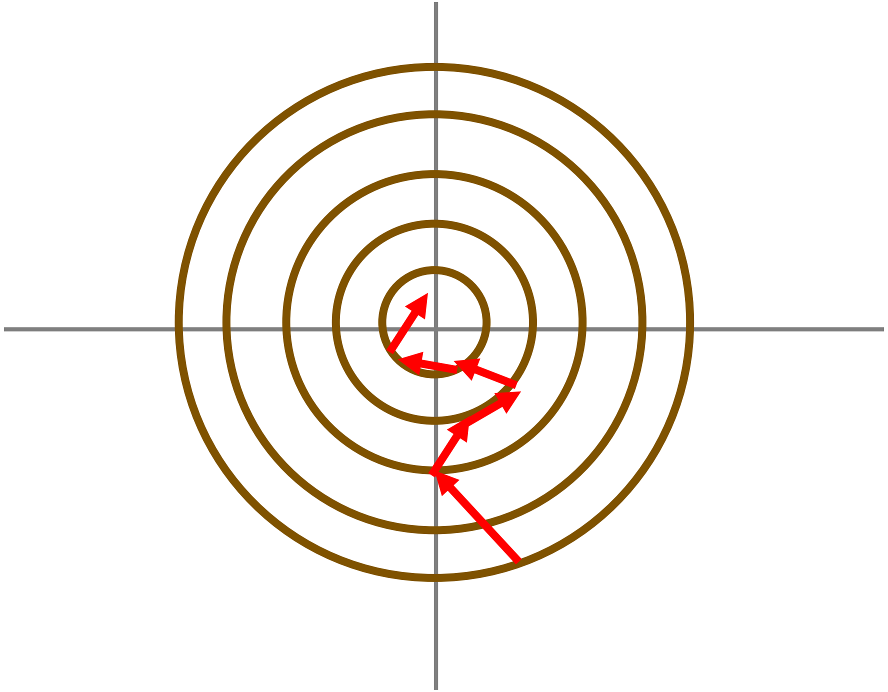


Shuffle the Data!



Shuffle the Data!





The Keras Package

- Keras allows easy construction, training, and execution of Deep Neural Networks
- Written in Python, and allows users to configure complicated models directly in Python
- Uses either Tensorflow or Theano “under the hood”
- Uses either CPU or GPU for computation
- Uses numpy data structures, and a similar command structure to scikit-learn (model.fit , model.predict, etc.)

Typical Command Structure in Keras

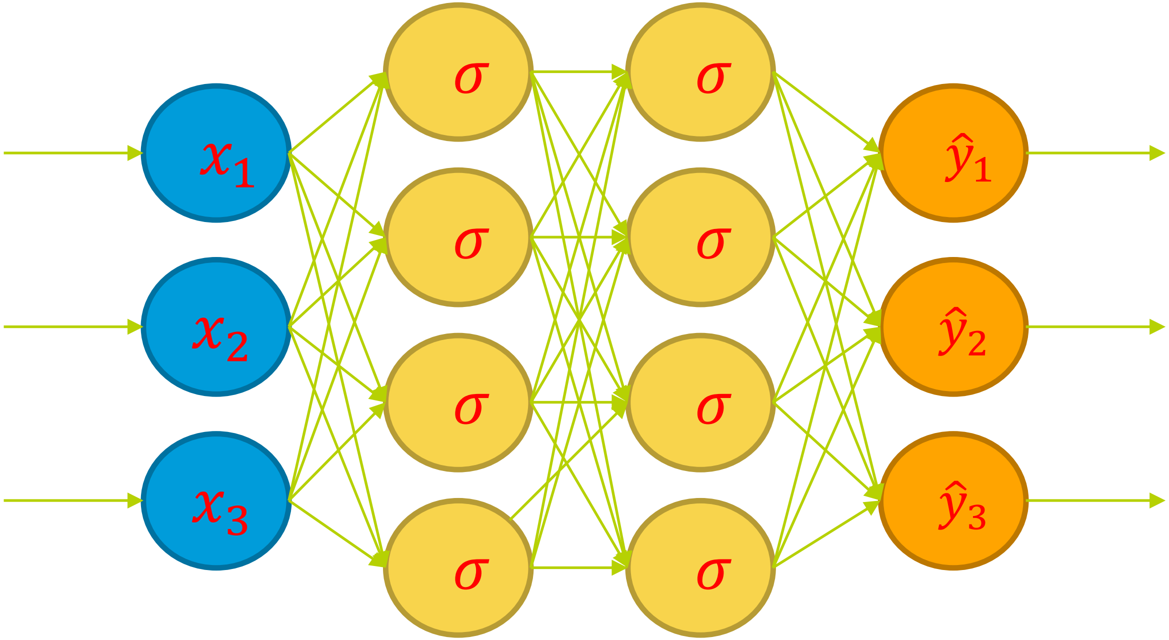
- Build the structure of your network.
- Compile the model, specifying your loss function, metrics, and optimizer (which includes the learning rate).
- Fit the model on your training data (specifying batch size, number of epochs)
- Predict on new data
- Evaluate your results

Building the model

- Keras provides two approaches to building the structure of your model:
- Sequential Model: allows a linear stack of layers – simpler and more convenient if model has this form
- Functional API: more detailed and complex, but allows more complicated architectures
- We will focus on the Sequential Model.

Running Example, this time in Keras

Let's build this Neural Network structure shown below in Keras:



Keras - Sequential Model

First, import the Sequential function and initialize your model object:

```
from keras.models import Sequential
model = Sequential()
```

Keras - Sequential Model

Then we add layers to the model one by one.

```
from keras.layers import Dense, Activation

# For the first layer, specify the input dimension
model.add(Dense(units=4, input_dim=3))

# Specify an activation function
model.add(Activation(sigmoid'))

# For subsequent layers, the input dimension is presumed from
# the previous layer
model.add(Dense(units=4))
model.add(Activation(sigmoid'))
model.add(Dense(units=3))
model.add(Activation('softmax'))
```

Multiclass Classification with Neural Networks

- For binary classification problems, we have a final layer with a single node and a sigmoid activation.
- This has many desirable properties
  - Gives an output strictly between 0 and 1
  - Can be interpreted as a probability
  - Derivative is “nice”
  - Analogous to logistic regression
- Is there a natural extension of this to a multiclass setting?

Multiclass Classification with Neural Networks

- Reminder: one hot encoding for categories
- Take a vector with length equal to the number of categories
- Represent each category with one at a particular position (and zero everywhere else)

$$\begin{bmatrix}1\\0\\0\end{bmatrix} \qquad \begin{bmatrix}0\\1\\0\end{bmatrix} \qquad \begin{bmatrix}0\\0\\1\end{bmatrix}$$

Cat Dog Toaster

Multiclass Classification with Neural Networks

- For multiclass classification problems, let the final layer be a vector with length equal to the number of possible classes.
- Extension of sigmoid to multiclass is the softmax function.
- $softmax(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$
- Yields a vector with entries that are between 0 and 1, and sum to 1

Multiclass Classification with Neural Networks

- For loss function use “categorical cross entropy”
- This is just the log-loss function in disguise

$$C.E. = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

- Derivative has a nice property when used with softmax

$$\frac{\partial C.E.}{\partial softmax} \cdot \frac{\partial softmax}{\partial z_i} = \hat{y}_i - y_i$$

Ways to scale inputs

- Linear scaling to the interval [0,1]

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

- Linear scaling to the interval [-1,1]

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

Ways to scale inputs

- Standardization (making variable approx. std. normal)

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