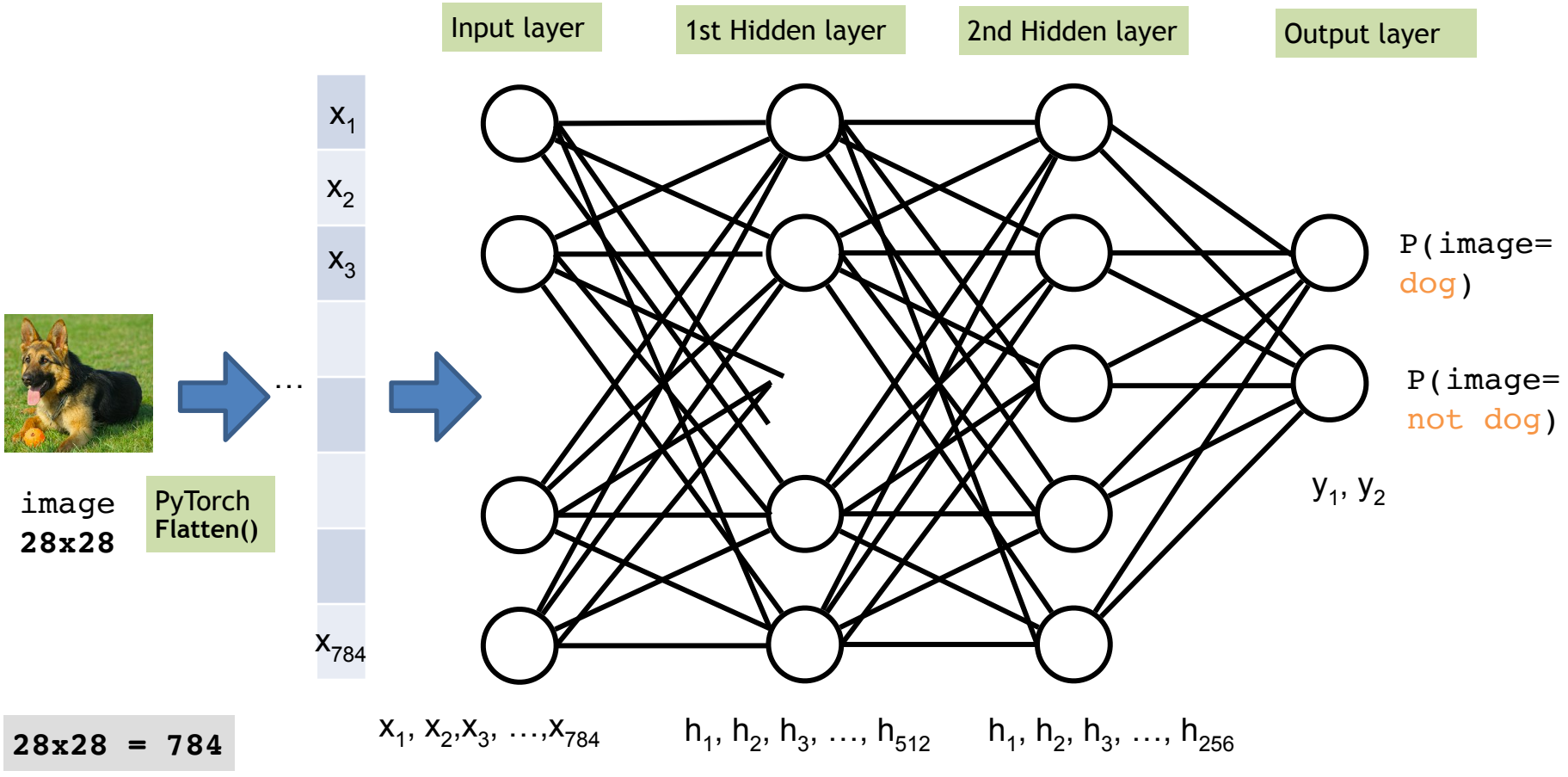
CS167: Machine Learning

MLP Training using PyTorch

Wednesday, April 15th, 2026

Recap: Multilayer Perceptron (MLP)

- A **multilayer perceptron** is the simplest type of neural network. It consists of perceptrons (aka nodes, neurons) arranged in layers



Recap: List of PyTorch Functions We Need

- nn.Linear()
  creates the dense connections between two adjacent layers (*left layer* and *right layer*)
  just provide **#neurons_left_layer** and **#neurons_right_layer**
- nn.ReLU()
- nn.Softmax()
- nn.Sequential()

- nn.flatten()
- nn.CrossEntropyLoss()
- torch.optim.SGD

Today's Agenda

- Review: Gradient Descent and Stochastic Gradient Descent (SGD)
- Modular code for MLP training
- Activity Poll on MLP

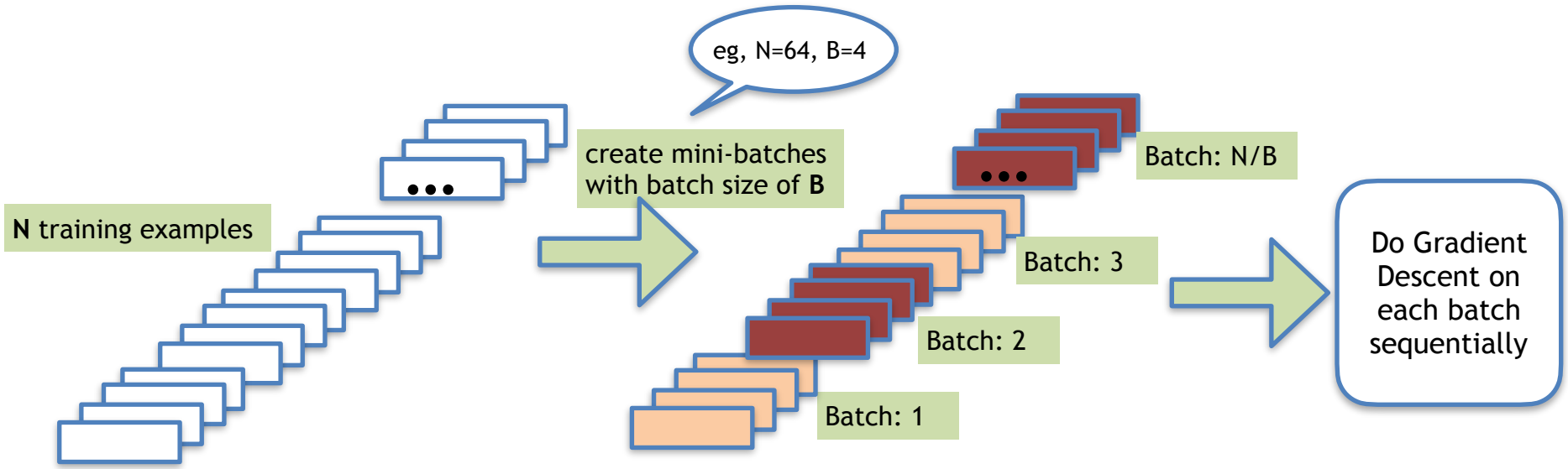
Stochastic Gradient Descent (SGD)

- Keep doing the Gradient Descent, but instead of using all the training samples, *use small subset of training samples* picked randomly when computing the gradient vector

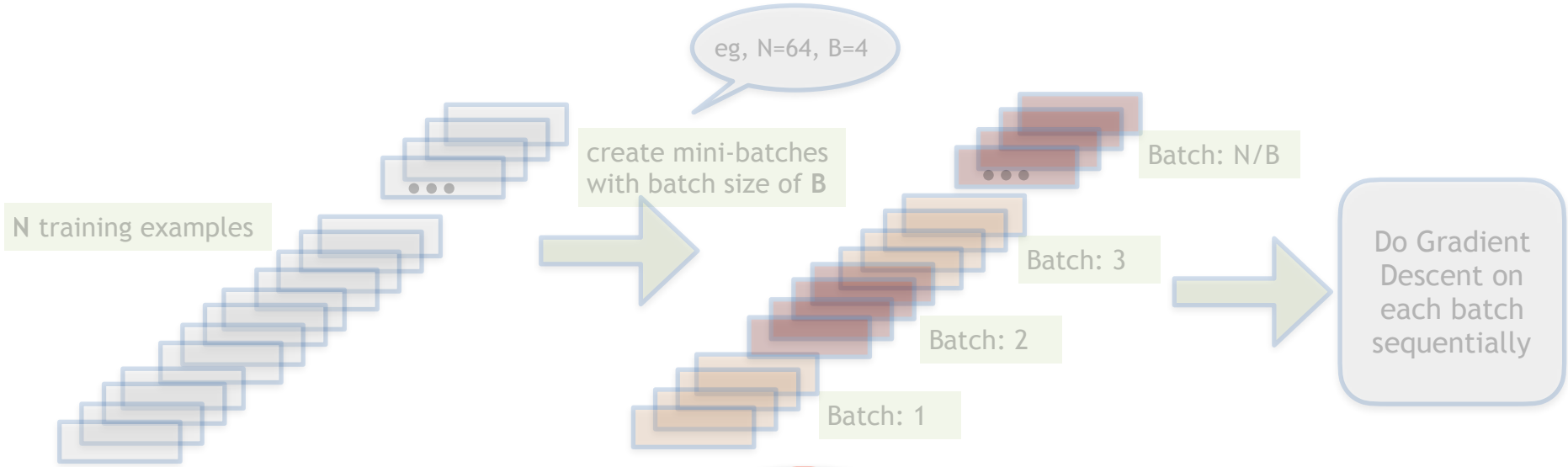
  - divide the entire training data into mini batches
  - calculate the gradient vector based on that batch $\nabla E(\mathbf{w})$
  - adjust (or update) the values of the weights based on the gradient vector to that batch

$$\mathbf{w}^{\mathbf{new}} = \mathbf{w}^{\mathbf{old}} - \eta \nabla \mathbf{E}(\mathbf{w})$$

Stochastic Gradient Descent (SGD)



Hyperparameters: things you could pick and choose but you don't learn those during training

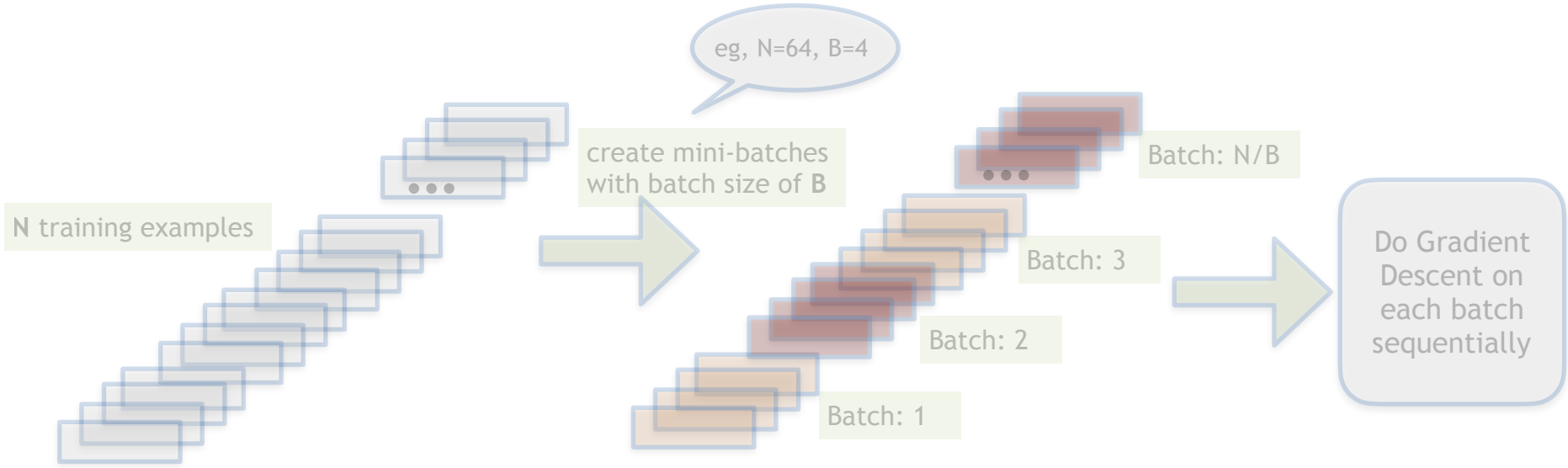


- **Optimization algorithm**
  - SGD
  - Adam
  - RMSProp
  - AdaGrad

- **Optimization algorithm related hyperparameters**
  - Batch size
  - Epoch
  - Learning rate
  - Learning rate schedule
  - Momentum

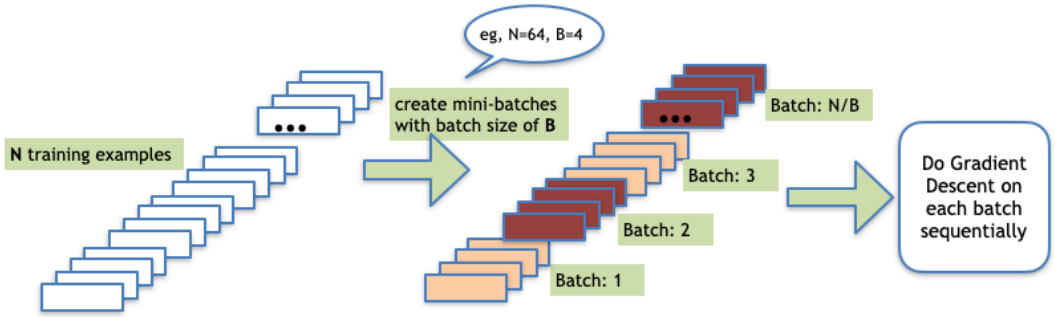
- **Regularization related hyperparameters**
  - Weight decay
  - Dropout
  - Batchnorm/groupnorm

Hyperparameters: things you could pick and choose but not learned



- **Optimization algorithm matters**
  - **SGD**: slow but gives great results eventually
  - **Adam**: fast but tends to over-fit
  - **RMSProp**: sometimes works best

In PyTorch: Stochastic Gradient Descent (SGD)



PyTorch Implementation

```
learning_rate = 1e-3
batch_size    = 4 # If the total sample size is 64, setting batch_size=4 will divide the data into 64÷4=16 mini-batches of tensors
epochs        = 10
# let's use SGD optimization algorithm for training our model
optimizer     = torch.optim.SGD(mlp_model.parameters(), lr=learning_rate)
```

Optimiztaion, as we have discussed earlier, is process of adjusting model parameters to reduce model error in each training step. PyTorch provides a selection of optimization algorithms in the torch.optim package. Some of them are as follows:

- torch.optim.SGD
- torch.optim..Adam
- torch.optim.RMSprop

In addition to selecting the optimizer, we can also select the hyperparameters which are refered to as adjustable parameters crucial for controlling the model optimization process. You can influence the training and convergence of the model by tweaking these hyperparameters:

- **epochs:** denotes the number of iterations over the dataset
- **batch size:** represents the quantity of data samples in each iteration propagated through the network before updating the parameters
- **learning rate:** determines the extent of parameter updates made at each batch/epoch

Today's Agenda

- Review: Gradient Descent and Stochastic Gradient Descent (SGD)
- MLP training
- Activity Poll on MLP

Training modular MLP using PyTorch

A multilayer perceptron is the simplest type of neural network. It consists of perceptrons (aka nodes, neurons) arranged in layers. Create a network class with two methods:

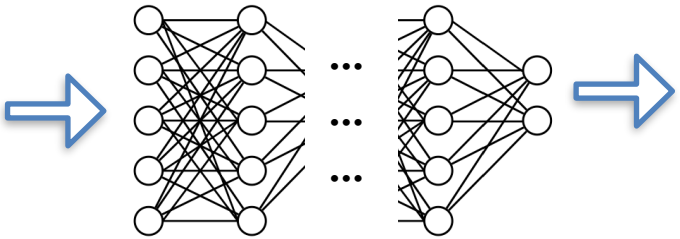
- *init()*
- *forward()*

```python
import torch
from torch import nn

# You can give any name to your new network, e.g., SimpleMLP.
# However, you have to mandatorily inherit from nn.Module to
# create your own network class. That way, you can access a lot of
# useful methods and attributes from the parent class nn.Module

class SimpleMLP(nn.Module):
  def __init__(self):
    super().__init__()
    # your network layer construction should take place here
    # ...
    # ...

  def forward(self, x):
    # your code for MLP forward pass should take place here
    # ...
    # ...
    return x
```

Dive into the Colab demo!

What's Next?

- A **multilayer perceptron** (MLP) is the simplest type of neural network. It consists of perceptrons (aka nodes, neurons) arranged in layers



- A **multilayer perceptron** (MLP) is just the tip of the iceberg; plenty of other neural network variants exist.

- Convolutional Neural Network (CNN): another type of neural network

Poll: MLP Summary

- Finish the MLP poll below:

https://forms.gle/jEDrK5z75BKmfCzu6