Minibatch Stochastic Gradient Descent

Outline

- General neural network process
- (Batch) gradient descent
- Stochastic gradient descent (SGD)
- Minibatch stochastic gradient descent (Minibatch SGD)
- Discuss tradeoffs

Recap: Automatic Differentiation

- Take five minutes to draw
  - Whatever will help you remember (no correct or incorrect drawings)
  - You’ll keep a running drawing log the rest of the semester

What do MSE and MAE look like in code?

Mean

Mean

```
def mean(self) -> Matrix:
    """Return the mean of all values across both dimensions."""
    result = Matrix(List2D(1, 1, self.data.mean()), children=(self,))

    def _gradient() -> None:
        info(f"Gradient of mean. Shape: {self.shape}")
        n = self.nrow * self.ncol
        self.grad += List2D(self.nrow, self.ncol, result.grad.vals[0][0] / n)

    result._gradient = _gradient
    return result
```

```
def mean(self) -> float:
    """Compute the mean of all values in the matrix."""
    return self.sum() / (self.nrow * self.ncol)
```

Absolute Value

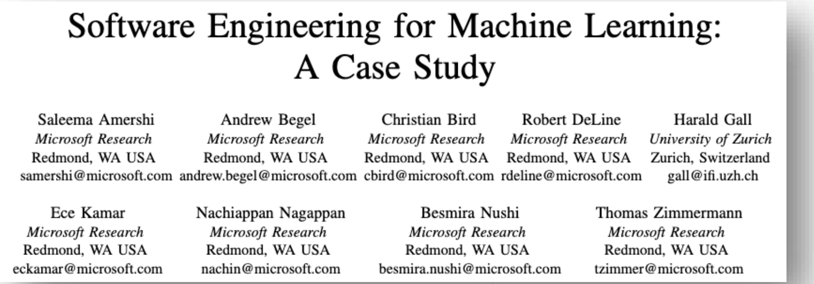
Absolute Value

```python
def abs(self) -> Matrix:
    """Return the element-wise absolute matrix values."""
    result = Matrix(self.data.abs(), children=(self,))

    def _gradient() -> None:
        self.grad += (self.data > 0) * result.grad
        self.grad += -(self.data < 0) * result.grad

    result._gradient = _gradient
    return result

def __abs__(self) -> Matrix:
    """Return the element-wise absolute matrix values."""
    return self.abs()
```

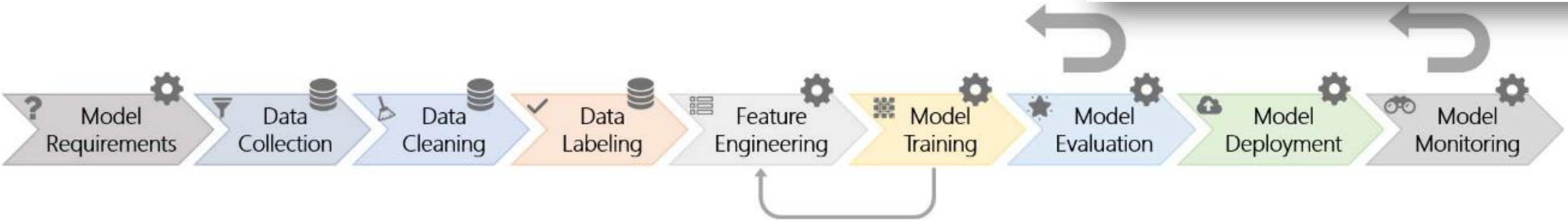
```python
def abs(self) -> List2D:
    data = [
        [abs(self.data[i][j]) for j in range(self.ncol)]
        for i in range(self.nrow)
    ]
    return List2D(*self.shape, data)

def __abs__(self) -> List2D:
    return self.abs()
```

General Neural Network Process

1. Prepare your dataset
   - Normalize inputs
   - Create a proxy (smaller) dataset for debugging and testing workflows
   - Split the dataset into training, validation, and evaluation
2. Design a neural network (architecture)
3. Set initial hyperparameters (learning rate, number of epochs, etc.)
4. Train model parameters (includes validation)
5. Evaluate the trained model
6. Deploy the trained model


Software Engineering for Machine Learning: A Case Study

Saleema Amershi, Microsoft Research, Redmond, WA USA, samershi@microsoft.com
Andrew Begel, Microsoft Research, Redmond, WA USA, andrew.begel@microsoft.com
Christian Bird, Microsoft Research, Redmond, WA USA, cbird@microsoft.com
Robert DeLine, Microsoft Research, Redmond, WA USA, rdeline@microsoft.com
Harald Gall, University of Zurich, Zurich, Switzerland, gall@ifi.uzh.ch
Ece Kamar, Microsoft Research, Redmond, WA USA, eckamar@microsoft.com
Nachiappan Nagappan, Microsoft Research, Redmond, WA USA, nachin@microsoft.com
Besmira Nushi, Microsoft Research, Redmond, WA USA, besmira.nushi@microsoft.com
Thomas Zimmermann, Microsoft Research, Redmond, WA USA, tzimmer@microsoft.com



(Batch) Gradient Descent

(Batch) Gradient Descent

```python
for _ in range(num_epochs):

    # Grab the entire dataset as a single batch
    X, y = next(iter(train_loader))

    yhat = model(X)
    loss = criterion(yhat, y)
    train_losses.append(loss.detach().item())

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Compare results with validation data
    X, y = next(iter(valid_loader))
    yhat = model(X)
    loss = criterion(yhat, y)
    valid_losses.append(loss.detach().item())
```

Stochastic Gradient Descent

Stochastic Gradient Descent

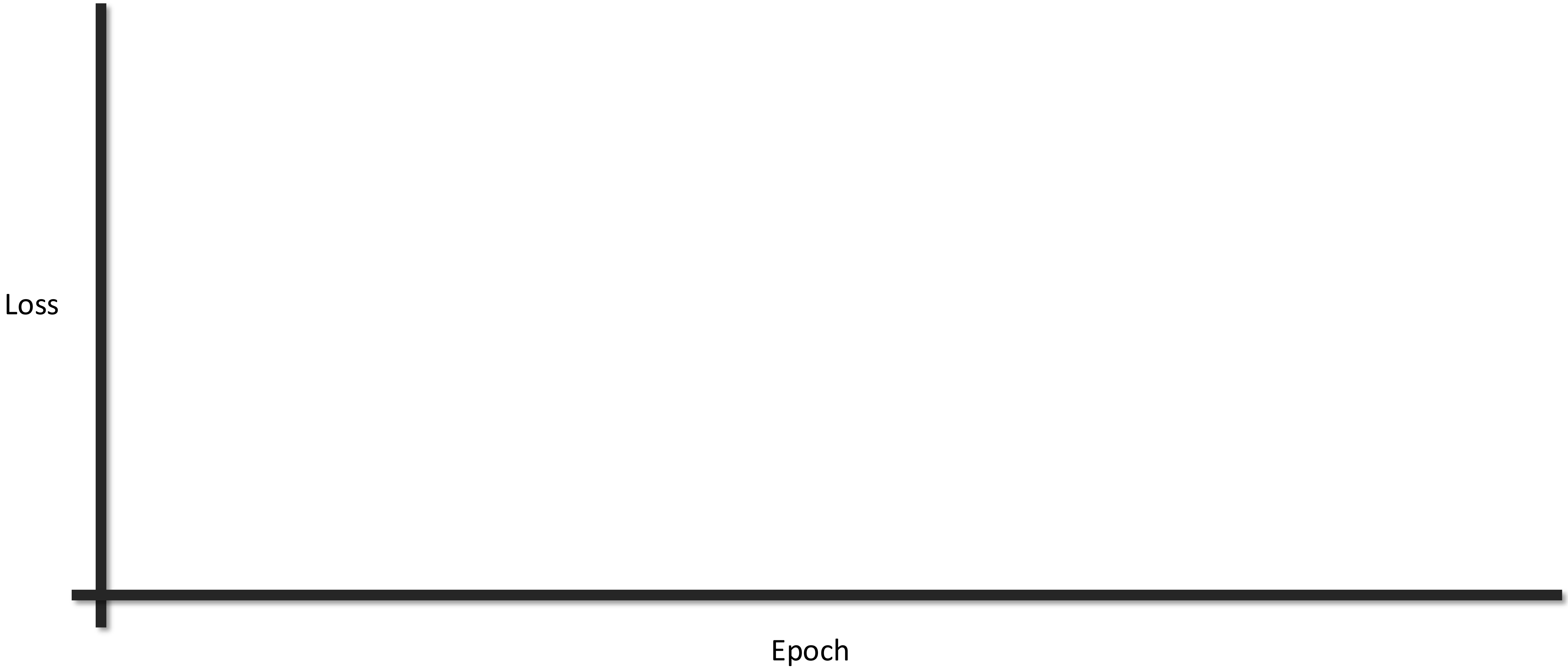
```
for _ in range(num_epochs):

    # Grab the entire dataset as a single batch
    for X, y in train_loader:
        yhat = model(X)
        loss = criterion(yhat, y)
        train_losses.append(loss.detach().item())

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    # Compare results with validation data
    X, y = next(iter(valid_loader))
    yhat = model(X)
    loss = criterion(yhat, y)
    valid_losses.append(loss.detach().item())
```

Minibatch Stochastic Gradient Descent

Minibatch Stochastic Gradient Descent

```
for _ in range(num_epochs):

    # Grab the entire dataset as a single batch
    for X, y in train_loader:
        yhat = model(X)
        loss = criterion(yhat, y)
        train_losses.append(loss.detach().item())

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    # Compare results with validation data
    X, y = next(iter(valid_loader))
    yhat = model(X)
    loss = criterion(yhat, y)
    valid_losses.append(loss.detach().item())
```

Typical Behavior



Tradeoffs

- (Batch) gradient descent
  - Fewer updates means slower progress in terms of real time
  - Maximally leverages parallel computations (faster)

Tradeoffs

- (Batch) gradient descent
  - Fewer updates means slower progress in terms of real time
  - Maximally leverages parallel computations (faster)

- Stochastic gradient descent
  - Many updates can sometimes lead to faster convergence
  - Noisy data means that some updates are bad

Tradeoffs

- (Batch) gradient descent
  - Fewer updates means slower progress in terms of real time
  - Maximally leverages parallel computations (faster)

- Stochastic gradient descent
  - Many updates can sometimes lead to faster convergence
  - Noisy data means that some updates are bad

- Minibatch stochastic gradient descent
  - Noisy but less than SGD
  - Pretty fast but slower than batch gradient descent
  - Should always be your default choice
  - Add another hyperparameter called `batch_size`

Code Demo

https://github.com/anthonyjclark/cs152spring25/blob/main/Lectures/06-MinibatchSgd/SgdDemo.ipynb