Automatic Differentiation
SpeakUp

Join by URL: https://web.speakup.info/room/join/80740

Join by key: web.speakup.info → use key 80740

Join by saved session: web.speakup.info

Join by QR:
Outline

- Recap neural networks and backpropagation
- Simple minimization examples
- Example with partial derivatives and a compute graph
- Partial derivatives and compute graphs for a one-layer network
- Partial derivatives and compute graphs for a two-layer network
- Automatic differentiation in code
Recap: Neural Networks and Backpropagation

• 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
\[ f(a) = 4(a - 2)^2 - 3 \]

Plot | Closed-form solution

https://www.desmos.com/calculator/xileztfrht
\[ f(a) = 4a^2 - 3, \text{ where } a = 1.8 \text{ and } a = 2.2 \]

Plot and Gradient Descent
\[ f(a, b, c, d) = b(a - c)^2 - d \]
One-Layer Network (MSE and Sigmoid)

- $Z^{[1]} = XW^{[1]T} + b^{[1]T}$
- $A^{[1]} = \sigma(Z^{[1]})$
- $\mathcal{L} = \frac{1}{2}(\hat{Y} - Y)^2$
Two-Layer Network (MSE and Sigmoid)

- $Z^{[1]} = XW^{[1]T} + b^{[1]T}$
- $A^{[1]} = \sigma(Z^{[1]})$
- $Z^{[2]} = A^{[1]}W^{[2]T} + b^{[2]T}$
- $A^{[2]} = \sigma(Z^{[2]})$
- $\mathcal{L} = \frac{1}{2} (\hat{Y} - Y)^2$
Manual Differentiation

• How does the derivation change if we add hidden layers?

• How does the derivation change if we change the activation function?

• How does the derivation change if we change the loss function?
Automatic Differentiation (Autodiff, Autograd)

# Forward propagation
A₀ = X, Yhat = A₂
Z₁ = linear(A₀, W₁, b₁)
A₁ = sigmoid(Z₁)
Z₂ = linear(A₁, W₂, b₂)
A₂ = sigmoid(Z₂)

# Compute loss as the mean-square-error
bce_loss = torch.mean(Y * torch.log(A₂) + (1 - Y) * torch.log(1 - Yhat))

learning_rate = 0.01 # aka alpha or α

# Compute gradients for W^[2] and b^[2]
dL_dY = (Y / Yhat - (1 - Y) / (1 - Yhat)) / 2
dY_dZ2 = Yhat * (1 - Yhat)
dZ2 = dL_dY * dY_dZ2
dW2 = (1 / N) * dZ2.T @ A1
db2 = dZ2.mean(dim=0)

# Compute gradients for W^[1] and b^[1]
dZ1 = dZ2 @ W2 * ((A1 * (1 - A1)))
dW1 = (1 / N) * dZ1.T @ X
db1 = dZ1.mean(dim=0)

W₁ -= learning_rate * dW₁
b₁ -= learning_rate * db₁
W₂ -= learning_rate * dW₂
b₂ -= learning_rate * db₂

bce_loss.backward()
W₁ -= learning_rate * W₁.grad
b₁ -= learning_rate * b₁.grad
W₂ -= learning_rate * W₂.grad
b₂ -= learning_rate * b₂.grad
Creating the Compute Graph

```python
# Sigmoid compute node from matrix.py

def sigmoid(self) -> Matrix:
    """Element-wise sigmoid.""
    result = Matrix(self.data.sigmoid(), children=(self,))

    def _gradient() -> None:
        self.grad += result.data * (1 - result.data) * result.grad

    result._gradient = _gradient
    return result

# Sigmoid math from list2d.py

def sigmoid(self) -> List2D:
    vals = [
        [sigmoid(self.vals[i][j]) for j in range(self.ncol)]
        for i in range(self.nrow)
    ]
    return List2D(*self.shape, vals)
```
# Sigmoid compute node from matrix.py

def sigmoid(self) -> Matrix:
    """Element-wise sigmoid."""
    result = Matrix(self.data.sigmoid(), children=(self,))

    def _gradient() -> None:
        self.grad += result.data * (1 - result.data) * result.grad

    result._gradient = _gradient
    return result
Backpropagation (.backward)

```python
def backward(self) -> None:
    """Compute all gradients using backpropagation."""

    sorted_nodes: list[Matrix] = []
    visited: set[Matrix] = set()

    # Sort all elements in the compute graph using a topological ordering (DFS)
    # (Creating a closure here for convenience; capturing sorted_nodes and visited)
    def topological_sort(node: Matrix) -> None:
        if node not in visited:
            visited.add(node)
            for child in node._children:
                topological_sort(child)
            sorted_nodes.append(node)

    # Perform the topological sort
    topological_sort(self)

    # Initialize all gradients with ones
    self.grad.ones_()

    # Update gradients from output to input (backwards)
    for node in reversed(sorted_nodes):
        node._gradient()
```