CS 331
Computer Vision

11 / 02 / 2026

Instructor: Michael Eckmann

Topics

- Questions / Comments?
- Neural Networks
  - Backprop
  - The entire learning process
  - Different ways to do gradient descent

How a Neural Network learns

- The steps are:

Initialize weights randomly

For all training examples

    Forward Propagation to make prediction

    Backward Propagation to compute the gradient

    Gradient Descent to alter the weights

- We covered forward propagation already.

Next Topics

- Introduce Backward propagation (aka backprop), what it does for us and how
- Go through an example
- Show the example with forward prop, backprop and gradient descent in a spreadsheet
- Outline of the overall learning process again including where backprop fits into it
- Computational Graphs
- Larger Network w/ activation functions and backprop

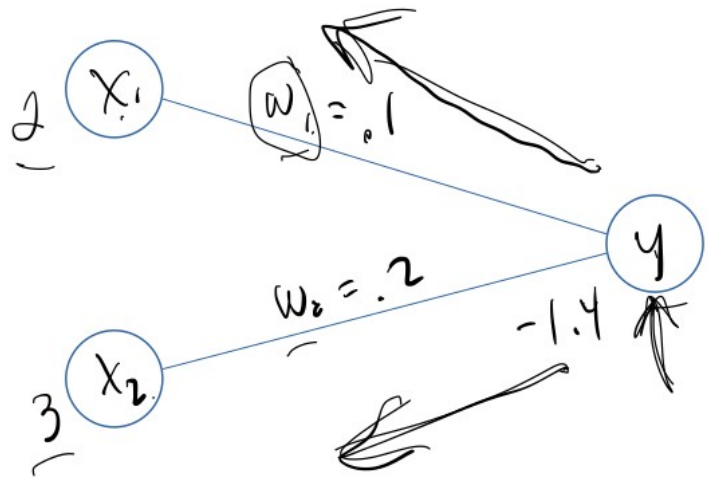
Backward Propogation

- The gradient could be computed analytically but would be inefficient for neural networks in general. **Backward Propogation**, simply backprop, is an efficient way to compute the gradient of a Neural Network. We will compute a partial derivative of the loss function with respect to each weight in the network and use these in gradient descent to update the weights.
- Backpropagation works by determining the partial derivatives at each layer starting at the last layer and going backwards.
- Think of each partial derivative with respect to a variable, say q, representing how much the loss function is affected by a change in variable q.

Backward Propogation

- Let’s go through an example of a simple network with 2 input units, 1 output unit, no activation function and only 1 training example.
- The loss function we’ll use is the familiar squared error (doesn’t have to be mean squared error b/c only 1 training example).

Backprop example



$3 \cdot -1.4 = -4.2$

$2 \cdot .1 + 3 \cdot .2 = .8$

x_1	x_2	Y
2	3	1.5

$y = w_1 x_1 + w_2 x_2$

$COST = (y - y_{train})^2$

$\frac{\partial COST}{\partial y} = 2(y - y_{train})$

$\frac{\partial COST}{\partial w_2} = \frac{\partial y}{\partial w_2} \cdot \frac{\partial COST}{\partial y} = 2 \cdot (.8 - 1.5) = -1.4$

$\frac{\partial COST}{\partial w_1} = \frac{\partial y}{\partial w_1} \cdot \frac{\partial COST}{\partial y} = -1.4 \cdot -1.4 = -2.8$

$\frac{\partial y}{\partial w_1} = x_1 \quad 2 \times -1.4$

$\alpha = 0.01$

Backprop example

$\frac{\partial \text{COST}}{\partial w_1} = -2.8$

$\frac{\partial \text{COST}}{\partial w_2} = -4.2$

$w_1 = w_1 - \alpha \frac{\partial \text{COST}}{\partial w_1}$

$w_2 = w_2 - \alpha \frac{\partial \text{COST}}{\partial w_2}$

$.1 - .01 \cdot (-2.8)$

$.2 - 0.01 \cdot (-4.2)$

$w_1 = .128$

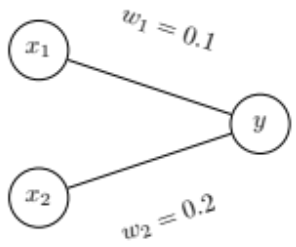
$w_2 = .242$

$.1 + .01 \cdot 2.8$

$.2 + .042$

$2.8 \times .01 = .028$

Gradient Descent, Part 1: Setup and Derivatives



x_1	x_2	y_{train}
2	3	1.5

Model and Cost

$$y = w_1 x_1 + w_2 x_2$$
$$\text{cost} = (y - y_{\text{train}})^2 = y^2 - 2y\, y_{\text{train}} + y_{\text{train}}^2$$

Derivatives

$$\frac{\partial\, \text{cost}}{\partial y} = 2y - 2y_{\text{train}} = 2(y - y_{\text{train}})$$
$$\frac{\partial y}{\partial w_1} = x_1, \qquad \frac{\partial y}{\partial w_2} = x_2$$

By the chain rule:

$$\frac{\partial\, \text{cost}}{\partial w_1} = \frac{\partial y}{\partial w_1} \cdot \frac{\partial\, \text{cost}}{\partial y} = x_1 \cdot \big(2(y - y_{\text{train}})\big)$$
$$\frac{\partial\, \text{cost}}{\partial w_2} = \frac{\partial y}{\partial w_2} \cdot \frac{\partial\, \text{cost}}{\partial y} = x_2 \cdot \big(2(y - y_{\text{train}})\big)$$

Gradient Descent, Part 2: Forward Pass and Update

Forward Propagation

$$y = 2(0.1) + 3(0.2) = 0.8$$

Backward Propagation

$$\frac{\partial \text{cost}}{\partial y} = 2(0.8 - 1.5) = -1.4$$

$$\frac{\partial \text{cost}}{\partial w_1} = 2 \cdot (-1.4) = -2.8$$

$$\frac{\partial \text{cost}}{\partial w_2} = 3 \cdot (-1.4) = -4.2$$

Weight Update

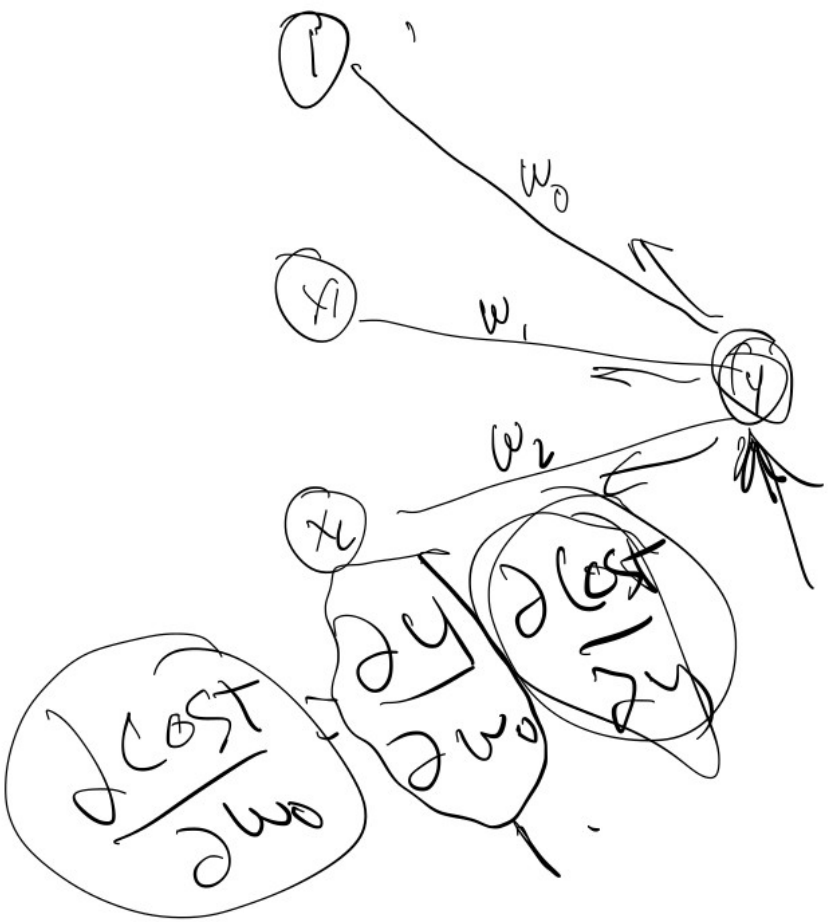
Learning rate: $\alpha = 0.01$

$$w_1 \leftarrow w_1 - \alpha \frac{\partial \text{cost}}{\partial w_1}, \qquad w_2 \leftarrow w_2 - \alpha \frac{\partial \text{cost}}{\partial w_2}$$

$$w_1 = 0.1 - 0.01 \cdot (-2.8) = 0.128$$

$$w_2 = 0.2 - 0.01 \cdot (-4.2) = 0.242$$

Backprop example



$$y = w_0 + w_1 x_1 + w_2 x_2$$

$$\frac{\partial COST}{\partial y} = 2(y - y_{train})$$

$$\frac{\partial COST}{\partial w_1} = \frac{\partial y}{\partial w_1} \cdot \frac{\partial COST}{\partial y}$$

$$\frac{\partial COST}{\partial w_2} = \frac{\partial y}{\partial w_2} \cdot \frac{\partial COST}{\partial y}$$

x_1

x_2

Complete Learning Example

- Let’s see a spreadsheet where we can perform many iterations of the learning of the two weights

How a Neural Network learns

- The steps are:

Initialize weights randomly

For many iterations

For each training example

Forward Propagation to make prediction

Backward Propagation to compute the gradient

Gradient Descent to alter the weights

Choices for Gradient Descent

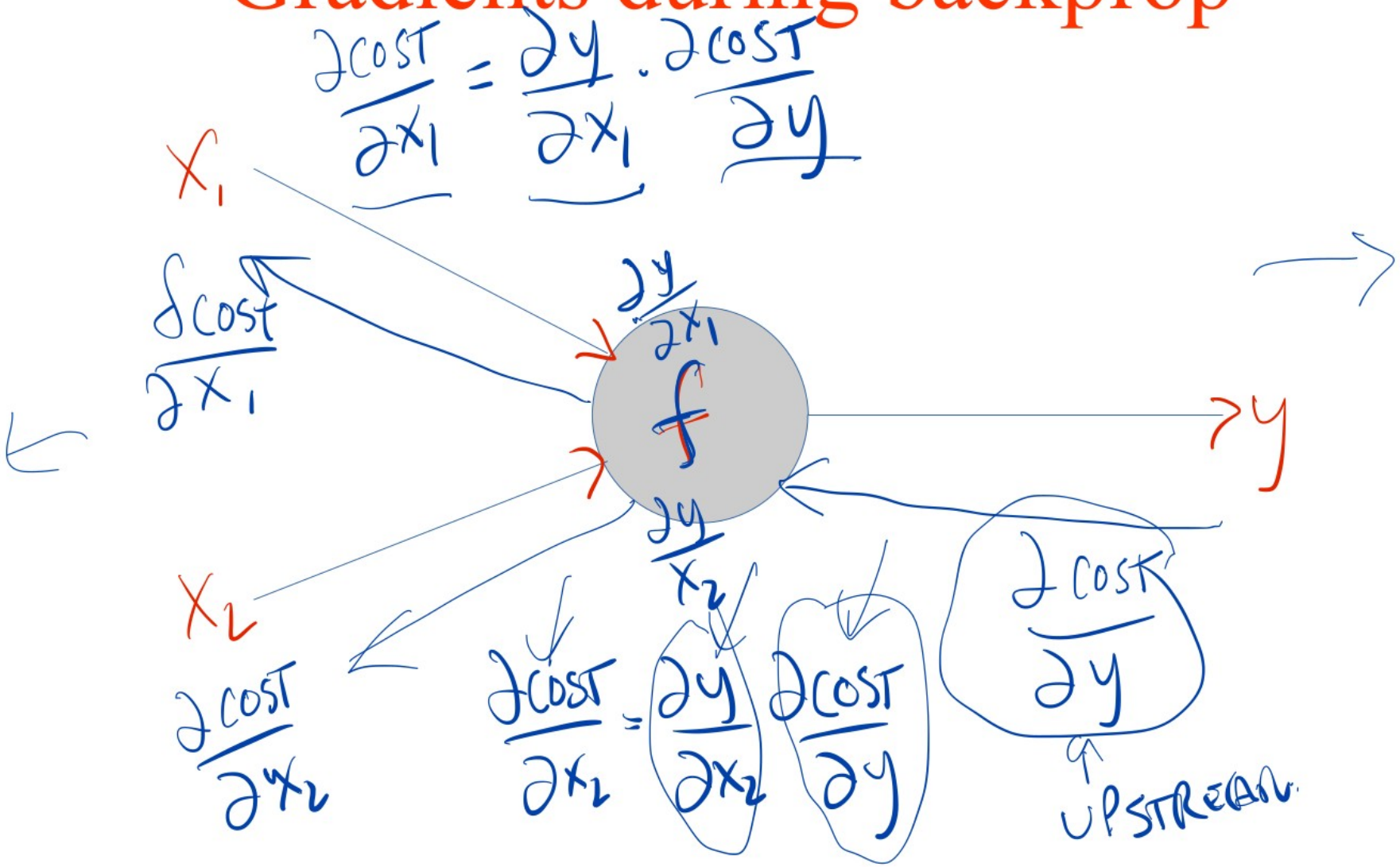
- Gradient descent can be done to alter the weights

a. once for each training example (Stochastic Gradient Descent (SGD)) – fast and low memory but noisy,

b. or once for the entire set of training examples (Batch Gradient Descent) – slow and high memory usage for large datasets

c. or once for each mini-batch of training examples (Mini-batch Gradient Descent) – practical middle-ground

Gradients during backprop



Backward Propogation

- Key ideas:
  - Gradients computed at the output travel backwards to compute other gradients
  - The chain rule was used to compute the gradients earlier in the network.
  - https://en.wikipedia.org/wiki/Chain_rule
  - Terminology: the gradient that travels backwards is the ***upstream gradient***, the gradient in the node itself is the ***local gradient*** and multiplying those two gets you the ***downstream gradient***.

Learning in a larger network w/activation functions

- What if we have activation functions – how do those affect the gradient?
- Show larger example