Artificial Intelligence for Tree Failure Identification and Risk Quantification
Introductory Meeting

NARS Lab

UMass Amherst
College of Engineering

November 4, 2020

Outline for this lecture

1. Introduction
2. Training overview
3. Convolutional NNs
4. AI for Tree Risk

Biological neuron

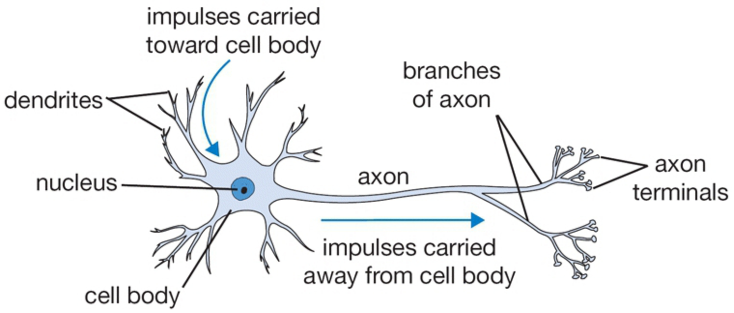
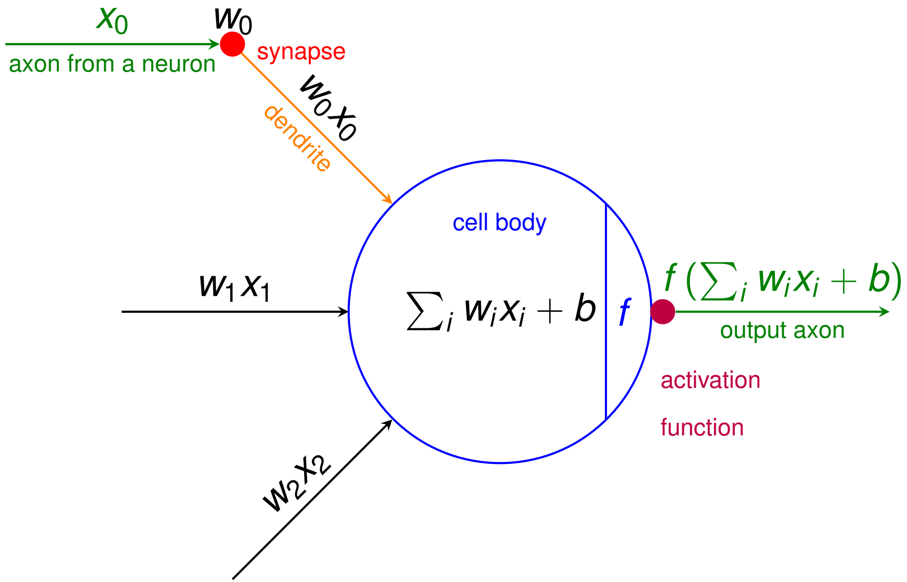

impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

Figure: Biological neuron (Source: `https://cs231n.github.io/neural-networks-1/`)

- $\sim$86 billion neurons are found in the human nervous system
- These neurons are connected by 10^{14} to 10^{15} synapses
- Each neuron receives input signals from its dendrites and outputs signals along a single axon
- The axon in turn connects to other neurons via synapses

Computational neuron model



Neural networks - basic ML terminology

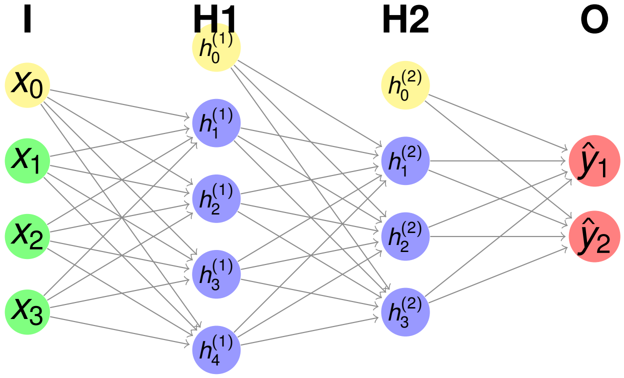
- x_i: signals traveling along axons (inputs)
- w_i: measure of synaptic strength, which is learned;
  - $w_i > 0 \rightarrow$ excitory influence
  - $w_i < 0 \rightarrow$ inhibitory influence
- Dendrites carry signals $w_i x_i$ to the cell body, where they are summed.
- If the final sum $w_i x_i + b > t$ where t is a threshold[1], the neuron sends a spike along its axon (i.e. fires)
- Computationally, the firing rate of a neuron is represented by an **activation function** f
- The output of a neuron is also called the *activation*
- The output neurons have no activation function. Instead, they perform a final transformation of outputs from the penultimate layer

Artificial Neural networks (ANNs) are typically modeled as connected layers of neuron in an acyclic graph (no loops).

- N-layer neural network (number of hidden layers + output layer)

[1] intercept b is referred to as the "bias" in ML literature

Three-layer neural network (with bias neurons)



- **Layers**: 3; **Hidden layers:** 2
- **Neurons**: 9
- **Learnable parameters:** $(4 \times 4) + (5 \times 3) + (4 \times 2) = 39$ weights

Matrix operations in neural networks

Given the activation vector ($m + 1$ neurons) in the zeroth (input) layer:

$$a^{(0)} = \begin{bmatrix} a_0^0 \\ a_1^0 \\ \vdots \\ a_m^0 \end{bmatrix} \tag{1}$$

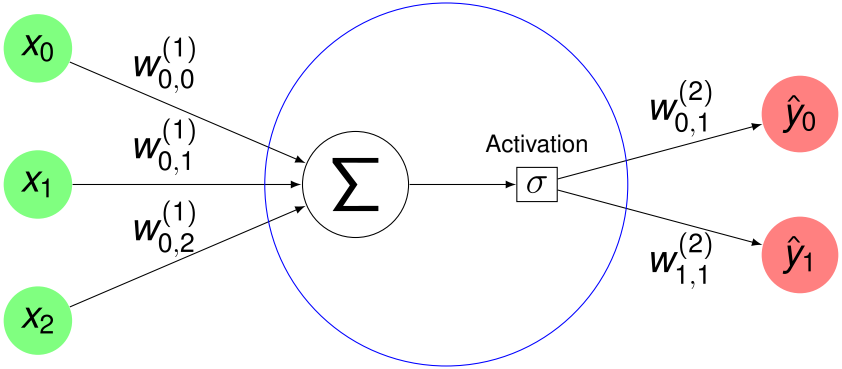
Then the activations in the next layer ($j + 1$ neurons) are given by:

$$a^{(1)} = \sigma\left(\boldsymbol{W} a^0 + b\right) = \sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,k} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,k} \\ \vdots & \vdots & \ddots & \vdots \\ w_{j,0} & w_{j,1} & \cdots & w_{j,k} \end{bmatrix} \begin{bmatrix} a_0^0 \\ a_1^0 \\ \vdots \\ a_n^0 \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}\right) \tag{2}$$

Example: If Layer 1 had only two neurons, then the weight matrix $\boldsymbol{W}$ would have only 2 rows.

Another look at the neuron

Given the notation formalization, we revisit the neuron.



Neural network notation

The **activation (output)** of a neuron is denoted:

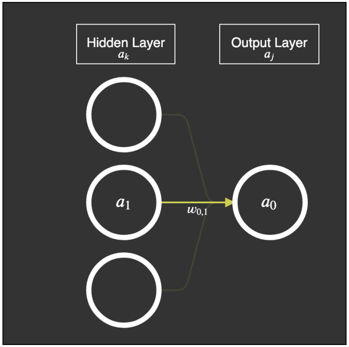
$$\sigma(w_1 a_1 + w_2 a_2 + \cdots + w_n a_n + b) = \sigma(\sum w_i a_i + b) = \text{new neuron} \quad (3)$$

Further, we denote each activation as $a_{neuron}^{(layer)}$, e.g.

- $a_3^{(1)}$: fourth neuron in first layer (layers are counted from first hidden layer)

Weights are denoted as $w_{to,from}$, e.g.

- $w_{1,2}^2$: from neuron 3 to neuron 2 in second layer
- The superscript is not often used, as it is clear from the context which layer we are dealing with



Training a neural network

- A neural network is trained or fitted by *learning* the optimal values of the weights.
- This learning is done via optimization (e.g. gradient descent)
- Gradient descent:

$$w^{\text{new}} = w^{\text{old}} - \eta \frac{\partial C}{\partial w^{\text{old}}} \tag{4}$$

  where:
  - η is the learning rate
  - C is the cost function (e.g. residual sum of squares):

$$C = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{5}$$

  - w the weight
- In neural networks, the gradients are computed via **backpropagation**

Neural network loss function

Given k output neurons and i observations (where θ represents the weights and f_k the output), we can compute the loss functions as follows.

For regression:

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{n} (y_{ik} - f_k(x_i))^2 \tag{6}$$

For classification, we use the cross-entropy (deviance):

$$R(\theta) = -\sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik} \log f_k(x_i)) \tag{7}$$

In modern ML terminology, this loss is usually referred to as a cost function C. Thus, we can write, where $k = 1$:

$$C = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{8}$$

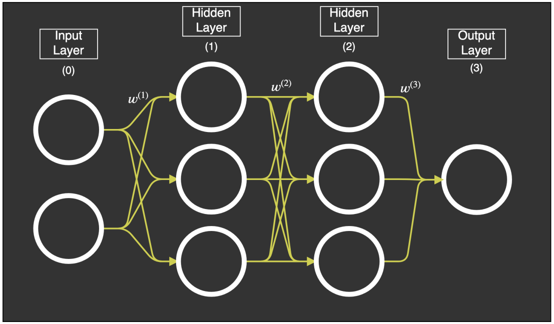
Summary of backpropagation

1. Fix initial weights $w^{(l),0}$, $b^{(l),0}$ and perform a forward sweep/pass through the network computing the activations a (outputs) of each layer l as:

$$a^{(l)} = \sigma(\boldsymbol{W}^{(l)} a^{(l-1)} + b^{(l)}) \tag{9}$$

2. At the output layer, we compute the cost function C (what we want to minimize)
3. Then, we *backpropagate* the errors through each layer in order to compute the gradients $\frac{\partial C}{\partial w^{(l)}}$, $\frac{\partial C}{\partial b^{(l)}}$ and weight updates $w^{(l),r+1}$ and $b^{(l),r+1}$
4. Repeat the forward and backward passes until cost is sufficiently minimized

Example: backpropagation for 3-layer network



$$\frac{\partial C}{\partial w^{(3)}} = \frac{\partial C}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial w^{(3)}} \tag{10}$$

$$\frac{\partial C}{\partial w^{(2)}} = \frac{\partial C}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial w^{(2)}} \tag{11}$$

$$\frac{\partial C}{\partial w^{(1)}} = \frac{\partial C}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial w^{(1)}} \tag{12}$$

Batch learning

The gradient descent method was described without referencing the observations.

In reality, the forward and backward passes are performed for each observation in the training set, with parameter updates obtained by **averaging** the gradients:

$$w^{(l),r+1} = w^{(l),r} - \eta \frac{1}{n} \sum_{i=1}^{n} \frac{\partial C_i}{\partial w^{(l)}} \tag{13}$$

$$b^{(l),r+1} = b^{(l),r} - \eta \frac{1}{n} \sum_{i=1}^{n} \frac{\partial C_i}{\partial b^{(l)}} \tag{14}$$

This approach is called **batch learning**.

- One sweep through all the training observations is called an *epoch*
- In batch learning, only one update results from a training epoch
- Thus, several epochs are required for convergence

Stochastic gradient descent

In standard gradient descent (batch learning), the updates are performed only after the gradient is computed for *all* training observations.

The stochastic gradient descent approach approximates the gradient using a randomly selected observation:

$$w^{(l),r+1} = w^{(l),r} - \eta \frac{\partial C_i}{\partial w^{(l)}} \tag{15}$$

$$b^{(l),r+1} = b^{(l),r} - \eta \frac{\partial C_i}{\partial b^{(l)}} \tag{16}$$

- Each iteration over observation i results in a weight/bias update
- Thus, one sweep through the entire training set (an epoch) produces n updates

This procedure is also known as **online learning**

Mini-batch learning

To introduce stability, we can compute weight updates over a *subset* of training observations

1. Randomly partition the training set into M mini-batches, each with m observations
2. Perform a forward and backward pass through each mini-batch to update the weights:

$$w^{(l),r+1} = w^{(l),r} - \eta \frac{1}{m} \sum_{i=1}^{m} \frac{\partial C_i}{\partial w^{(l)}} \tag{17}$$

$$b^{(l),r+1} = b^{(l),r} - \eta \frac{1}{m} \sum_{i=1}^{m} \frac{\partial C_i}{\partial b^{(l)}} \tag{18}$$

   Thus, for each iteration, average the gradient over a *mini-batch* of m observations
3. Repeat step 2 until convergence

Performance and robustness considerations

- Online learning is more efficient for very large datasets
- In practice, mini-batch learning is employed, as batch sizes (usually, $m = 16$ or $m = 32$) can be chosen to take advantage of parallel computing architectures
- An **adpative** learning rate η can guarantee convergence, e.g. $\eta_r = \frac{1}{r}$
- *Overfitting* can be mitigated via **weight decay** (regularization of weights):

$$C \leftarrow C + \lambda J = C + \lambda \left(\sum w^2 + \sum b^2 \right) \tag{19}$$

  where λ is tuning parameter estimated via cross-validation
  Early stopping and *dropout* are also used.
- Input **standardization** (mean zero, SD 1) is recommended for consistent weight initialization and regularization
- Weights/biases are initialized to *small* values near 0 for better performance
- Cost function C is nonlinear and nonconvex; other optimization approaches (e.g. conjugate gradient) can provide faster convergence compared to stochastic gradient descent

Other types of neural networks

The standard ANN architecture (MLP) we have studied is also called the feed-forward network.

Other architectures have been shown to give better performance for various applications:

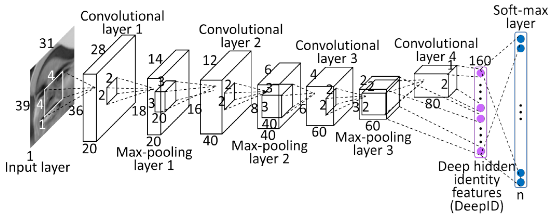
- Recurrent neural networks (RNNs): time-series forecasting
- Convolutional neural networks (CNNs): image classification
- Long short-term memory networks (LSTMs): time-series, pattern identification, etc.

The convolutional neural network (CNN)

- Motivated by the image recognition process of the brain's visual cortex.
- Groundbreaking study on cats revealed the importance of *local receptive fields* for activating neurons in the visual cortex. (Hubel & Wiesel, 1958; 1959)
- Earliest neural network for image recognitron introduced: *neocognitron* (Fukushima, 1980)
- Milestone: introduction of *LeNet-5* architecture for handwritten digit recognition (Yann LeCun et al., 1998)

Building blocks of a CNN

- **Input layer:** the image to be classified
- **Convolutional layer:** represents the action of a filter transmitting signals (features) from various portions (receptive fields) of the preceding layer. The size of the receptive field is specified by the *convolutional kernel*. Each layer can have multiple feature maps representing different filters.
- **Pooling layer:** subsamples signals from preceding layer to reduce dimensionality and extract dominant features (subsample space determined by kernel size)
- **Dense layer:** neuron outputs are flattened and fully connected
- **Output layer:** neurons equal to number of classes; with softmax activation



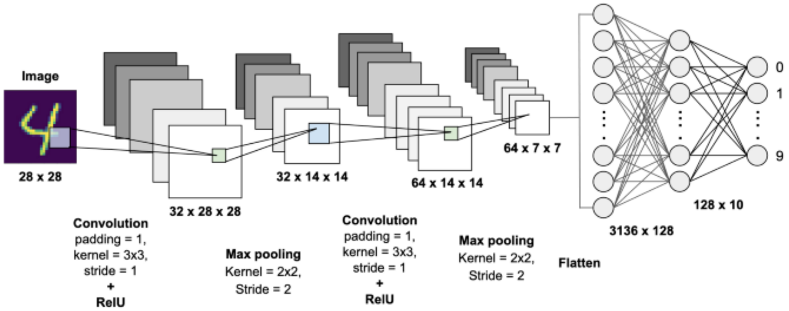
Training hyperparameters in a CNN

Several decisions must be made in selecting hyperparameters for training a CNN.

- Number of convolutional layers and feature maps in each layer
- Convolutional kernel size
- Stride length (spacing of filters)
- Choice of pooling function (max, average, etc)
- Number of dense layers
- Activation function in each layer (ReLU, tanh, etc)

Various high-performing architectures have been developed in recent years that can be adapted for other problems.
Training the network involves finding the weights for the various layers (using mini-batch gradient descent or other variants)

Another CNN example



Source: `https://towardsdatascience.com/mnist-handwritten-digits-classification-using-a-convolutional-neural-network-cnn-af5fafbc35e9`

Automating tree risk prediction using AI

- **Objective:** Automate tree risk classification using a convolutional neural network?
- **Question 1:** Can we elicit the visual features from a trained network that indicate various levels of risk?
- **Question 2:** Can we map estimated CNN weights to structural relationships governing tree stability?
- **Question 3:** How well can a trained CNN be transferred to another set of images and perform similarly well for risk prediction?
- **Question 4:** How can this process be adapted for Google Maps images (or drone photography)?

CNN Model 1

- Cropped and resized images to $108 \times 108 \times 1$ (grayscale)
- Binary classification; 103 probable/possible and 259 improbable
- Used a simple architecture (5 convolutional layers, 2 hidden dense layers, 1 output layer)
- Validation accuracy: 74% (10 epochs)
- Increasing input image size to 216×216 worsens outcome with accuracy of 68%
- If the stride in the first convolutional layer is increased to 2 (with the higher resolution), this does not improve the result.
- However, with a stride of 2 and 108×108 image resolution, we obtain a validation accuracy of 77%.

CNN Model 2

- Images are cropped but not converted to grayscale.
- Three classes: 40 probable, 63 possible , 259 improbable
- CNN architecture unchanged
- Validation accuracy: 74% (10 epochs)

Equation summary: outer layer

At the outer layer L (without indexing by neuron):

$$z^{(L)} = w^{(L)} \times a^{(L-1)} + b^{(L)} \tag{20}$$

$$a^{(L)} = \sigma(z^{(L)}) \tag{21}$$

$$C = (a^{(L)} - y)^2 \tag{22}$$

The gradient of the cost function with respect to $w^{(L)}$ is:

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}} = 2\left(a^{(L)} - y\right) \sigma'\left(z^{(L)}\right) a^{(L-1)} \tag{23}$$

Thus, we see that this gradient depends on the activation from the previous layer $a^{(L-1)}$. Also wrt to the bias:

$$\frac{\partial C}{\partial b^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial b^{(L)}} = 2\left(a^{(L)} - y\right) \sigma'(z^{(L)})(1) \tag{24}$$

Updating weights

We can then update the weights for the last layer for the next iteration $r + 1$:

$$w^{(L),r+1} = w^{(L),r} - \eta \frac{\partial C}{\partial w^{(L)}} \tag{25}$$

$$b^{(L),r+1} = b^{(L),r} - \eta \frac{\partial C}{\partial b^{(L)}} \tag{26}$$

To update the weights for layer $L-1$, we need to find the gradients $\frac{\partial C}{\partial w^{(L-1)}}$ and $\frac{\partial C}{\partial b^{(L-1)}}$.

Using the chain rule again, we write:

$$\frac{\partial C}{\partial w^{(L-1)}} = \frac{\partial C}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}} \tag{27}$$

$$\frac{\partial C}{\partial b^{(L-1)}} = \frac{\partial C}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial b^{(L-1)}} \tag{28}$$

Backward pass

But we recall that C is not *explicitly* dependent on $a^{(L-1)}$ as $C = (a^{(L)} - y)^2$. However, it is *implicitly* dependent, since

$$C \propto a^{(L)}, \tag{29}$$

$$a^{(L)} \propto z^{(L)} \tag{30}$$

and

$$z^{(L)} \propto a^{(L-1)} \tag{31}$$

So, we use the chain rule to expand $\frac{\partial C}{\partial a^{(L-1)}}$ as follows:

$$\frac{\partial C}{\partial a^{(L-1)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \tag{32}$$

Backward pass (cont.)

We can then expand the cost function gradient wrt to weights for layer $L-1$ as:

$$\frac{\partial C}{\partial w^{(L-1)}} = \frac{\partial C}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}} \tag{33}$$

$$\frac{\partial C}{\partial b^{(L-1)}} = \frac{\partial C}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial b^{(L-1)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial b^{(L-1)}} \tag{34}$$

Once these gradients are computed, we update the weights for the $r+1$th iteration using:

$$w^{(L-1),r+1} = w^{(L-1),r} - \eta \frac{\partial C}{\partial w^{(L-1)}} \tag{35}$$

$$b^{(L-1),r+1} = b^{(L-1),r} - \eta \frac{\partial C}{\partial b^{(L-1)}} \tag{36}$$

Summary: forward pass

1. Initialize weights and biases: $w^{(l),0}$, $b^{(l),0}$
2. Perform forward pass to compute activations:

$$z^{(l),0} = w^{(l),0} \times a^{(l-1),0} + b^{(l),0} \tag{37}$$

$$a^{(l)} = \sigma(z^{(l),0}) \tag{38}$$

At output layer:

$$z^{(L),0} = w^{(L),0} \times a^{(L-1),0} + b^{(L),0} \tag{39}$$

$$a^{(L),0} = \sigma(z^{(L),0}) \tag{40}$$

$$C = (a^{(L),0} - y)^2 \tag{41}$$

Summary: backward pass—outer layer

3. Backward pass, outer layer (L):
   1. Compute gradients:

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}} \tag{42}$$

$$\frac{\partial C}{\partial b^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial b^{(L)}} \tag{43}$$

   2. Update weights:

$$w^{(L),1} = w^{(L),0} - \eta \frac{\partial C^0}{\partial w^{(L)}} \tag{44}$$

$$b^{(L),1} = b^{(L),0} - \eta \frac{\partial C^0}{\partial b^{(L)}} \tag{45}$$

Summary: backward pass—last hidden layer

3. Backward pass, layer $(L-1)$:
   3. Compute gradients:

$$\frac{\partial C}{\partial w^{(L-1)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}} \tag{46}$$

$$\frac{\partial C}{\partial b^{(L-1)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial b^{(L-1)}} \tag{47}$$

   4. Update weights:

$$w^{(L-1),1} = w^{(L-1),0} - \eta \frac{\partial C}{\partial w^{(L-1)}} \tag{48}$$

$$b^{(L-1),1} = b^{(L-1),0} - \eta \frac{\partial C}{\partial b^{(L-1)}} \tag{49}$$

Summary: backward pass—second-to-last hidden layer

3. Backward pass, layer $(L-2)$:
   5. Compute gradients:

$$\frac{\partial C}{\partial w^{(L-2)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial a^{(L-2)}} \frac{\partial a^{(L-2)}}{\partial z^{(L-2)}} \frac{\partial z^{(L-2)}}{\partial w^{(L-2)}} \tag{50}$$

$$\frac{\partial C}{\partial b^{(L-2)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial a^{(L-2)}} \frac{\partial a^{(L-2)}}{\partial z^{(L-2)}} \frac{\partial z^{(L-2)}}{\partial b^{(L-2)}} \tag{51}$$

(52)

   6. Update weights:

$$w^{(L-2),1} = w^{(L-2),0} - \eta \frac{\partial C}{\partial w^{(L-2)}} \tag{53}$$

$$b^{(L-2),1} = b^{(L-2),0} - \eta \frac{\partial C}{\partial b^{(L-2)}} \tag{54}$$

Summary: backward pass—first hidden layer

3. Backward pass, layer (1):
    5. Compute gradients:

$$\frac{\partial C}{\partial w^{(1)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \cdots \frac{\partial z^{(2)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial w^{(1)}} \tag{55}$$

$$\frac{\partial C}{\partial b^{(1)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \cdots \frac{\partial z^{(2)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial b^{(1)}} \tag{56}$$

$$\tag{57}$$

    6. Update weights:

$$w^{(1),1} = w^{(1),0} - \eta \frac{\partial C}{\partial w^{(1)}} \tag{58}$$

$$b^{(1),1} = b^{(1),0} - \eta \frac{\partial C}{\partial b^{(1)}} \tag{59}$$