Learning Neural Networks - Part 2

Alice Gao

Lecture 9

Readings: RN 18.7, PM 7.5.

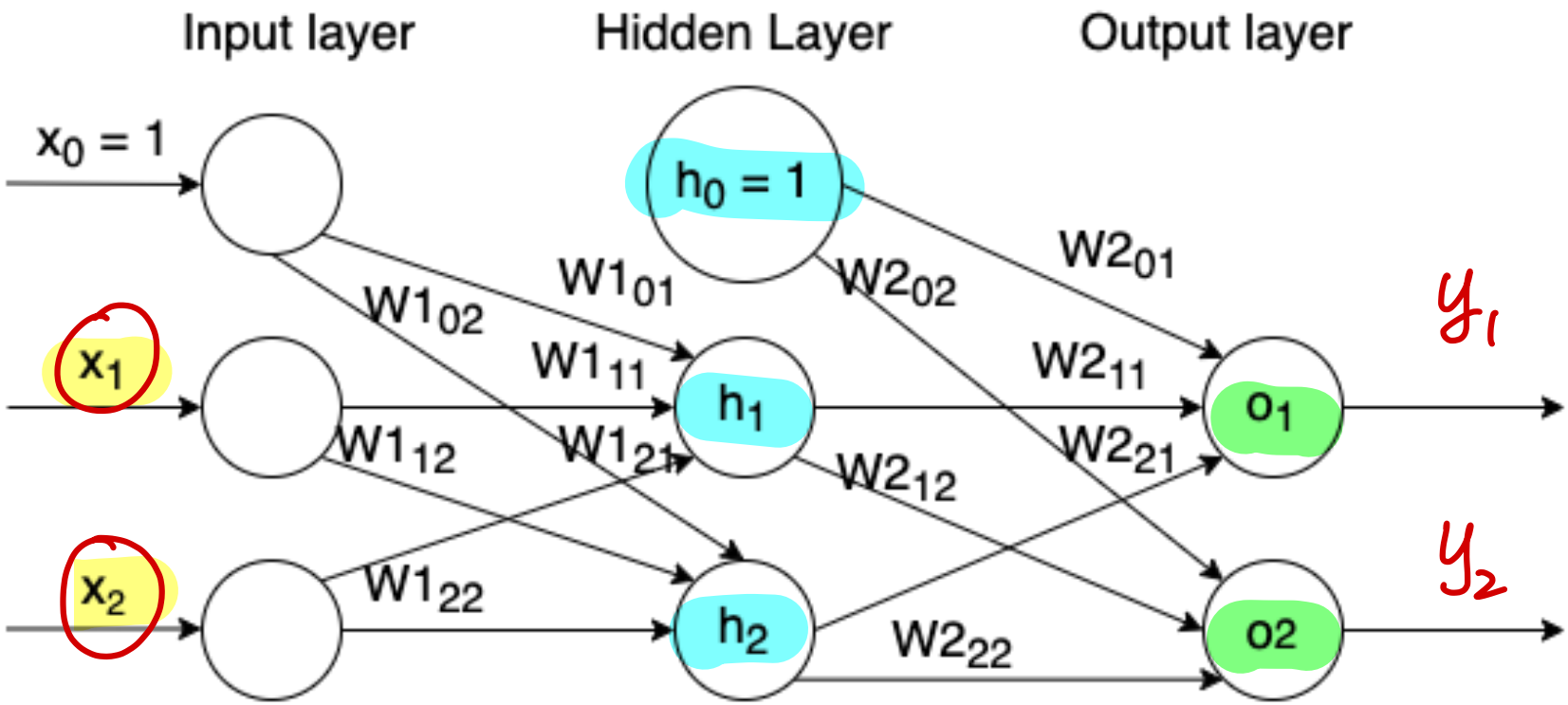
Outline

Learning Goals

By the end of the lecture, you should be able to

- Explain the steps of the gradient descent algorithm.
- Explain how we can modify gradient descent to speed up learning and ensure convergence.
- Describe the back-propagation algorithm including the forward and backward passes.
- Compute the gradient for a weight in a multi-layer feed-forward neural network.
- Describe situations in which it is appropriate to use a neural network or a decision tree.

A 3-Layer Neural Network



Gradient Descent

expected outputs based on training data
actual outputs based on training data & network.
error

"Walking downhill and always taking a step in the direction that goes down the most."

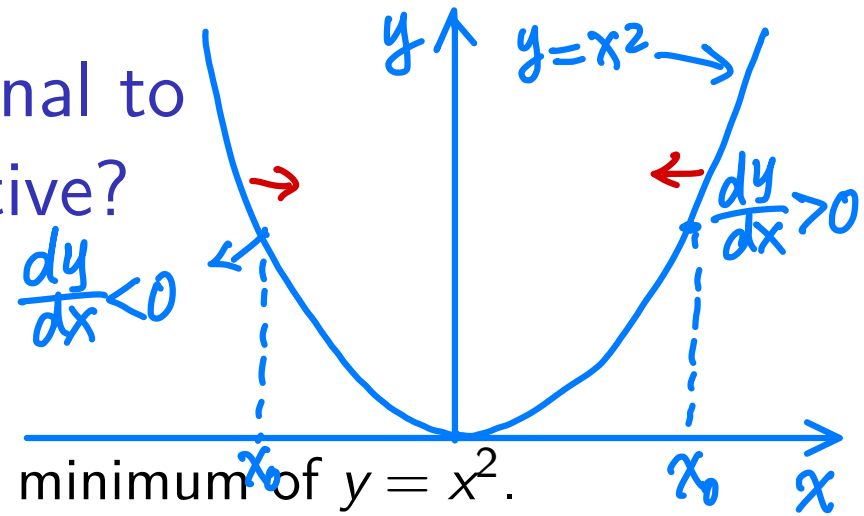
- A local search algorithm to find the minimum of a function.
- Steps of the algorithm:
  - Initialize weights randomly.
  - Change each weight in proportion to the negative of the partial derivative of the error with respect to the weight.

$$w := w - \sum \eta \frac{\partial error}{\partial w}$$

sum over all training examples.

  - η is the learning rate.
  - Terminate after some number of steps when the error is small or when the changes get small.

Why update the weight proportional to the negative of the partial derivative?



- Suppose that we want to find the minimum of $y = x^2$.
- Start with $x = x_0$.
- In what direction should we change the value of x?

  change x in the direction of the negative of the partial derivative.

- By what amount should we change the value of x? What is the step size?

  Take a step in proportion to the magnitude of the partial derivative.

How do we update the weights based on the data points?

cheaper steps, weights become more accurate more quickly.
not guaranteed to converge.

- Gradient descent updates the weights after sweeping through all the examples.
- To speed up learning, update weights after each example.
  - Incremental gradient descent
    update weights after each example.
  - Stochastic gradient descent
    same as incremental gradient descent except that each example is chosen randomly.
- Trade off learning speed and convergence.
  - Batched gradient descent
    update weights after a batch of examples.
    start w/ small batches to learn quickly.
    then, increase batch size so it converges.

The Back-propagation Algorithm

- Learn the weights in a neural network by using gradient descent
- For each training example (x_1, x_2, y_1, y_2), perform 2 passes.
  - Forward pass: compute o_1, o_2 given x_1, x_2 and the weights. For example

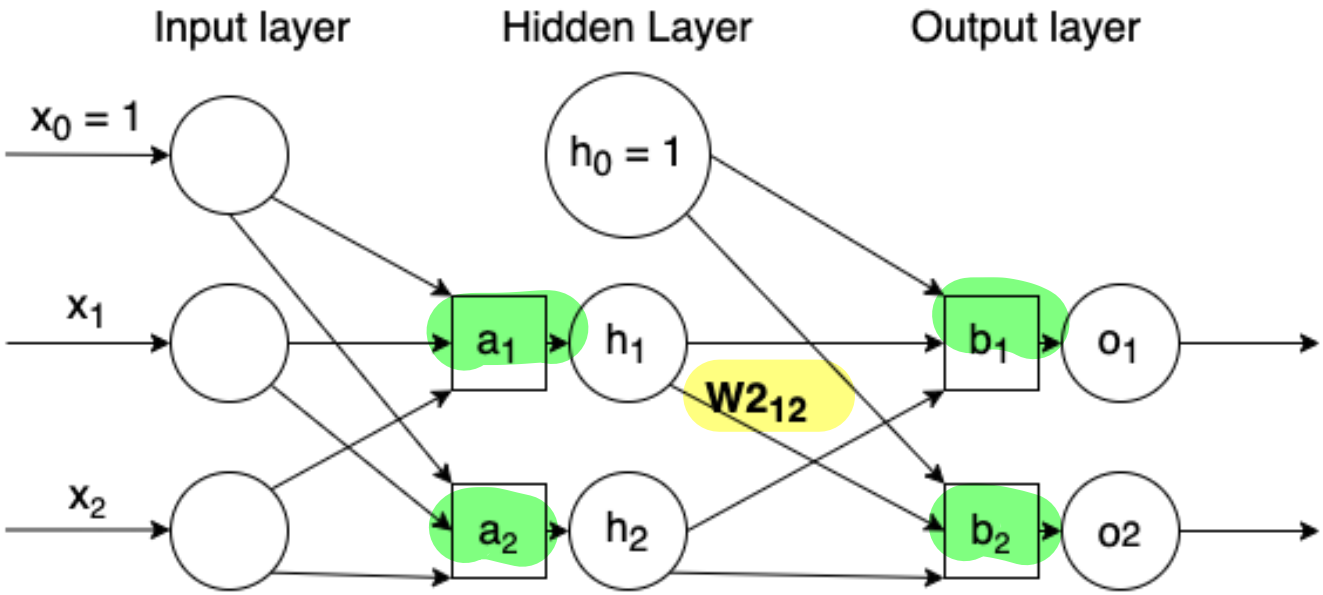
$$h_j = g\left(\sum_{i=0}^{2} w1_{ij} x_i\right), \text{ for } j = 1, 2$$

$E(o_1, o_2, y_1, y_2)$

$$o_j = g\left(\sum_{i=0}^{2} w2_{ij} h_i\right), \text{ for } j = 1, 2$$

  - Backward pass: calculate the partial derivative of the error with respect to each weight.
- Update each weight by the sum of changes for all the training examples.

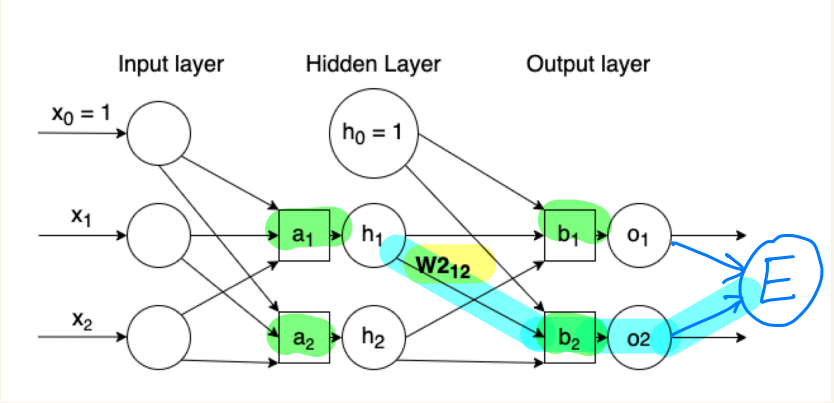
Calculating a gradient for $W2_{ij}$



$\frac{\partial error}{\partial W2_{12}} = ?$

$$a_j = \sum_i W1_{ij} x_i \text{ , for } j = 1, 2.$$

$$b_j = \sum_i W2_{ij} x_i \text{ , for } j = 1, 2.$$



$W2_{12} \rightarrow b_2 \rightarrow O_2 \rightarrow E$

$\frac{df(g(x))}{dx} = f'(g(x)) * \frac{dg(x)}{dx}$ chain rule.

$$\frac{\partial E}{\partial W2_{12}} = \frac{\partial E}{\partial O_2} * \frac{\partial O_2}{\partial b_2} * \frac{\partial b_2}{\partial W2_{12}}$$

$$E = (O_2 - y_2)^2 + (O_1 - y_1)^2$$

$$\frac{\partial E}{\partial O_2} = 2(O_2 - y_2)$$

$$O_2 = g(b_2) \qquad g(x) = \frac{1}{1 + e^{-x}}$$

$$g'(x) = g(x)(1 - g(x))$$

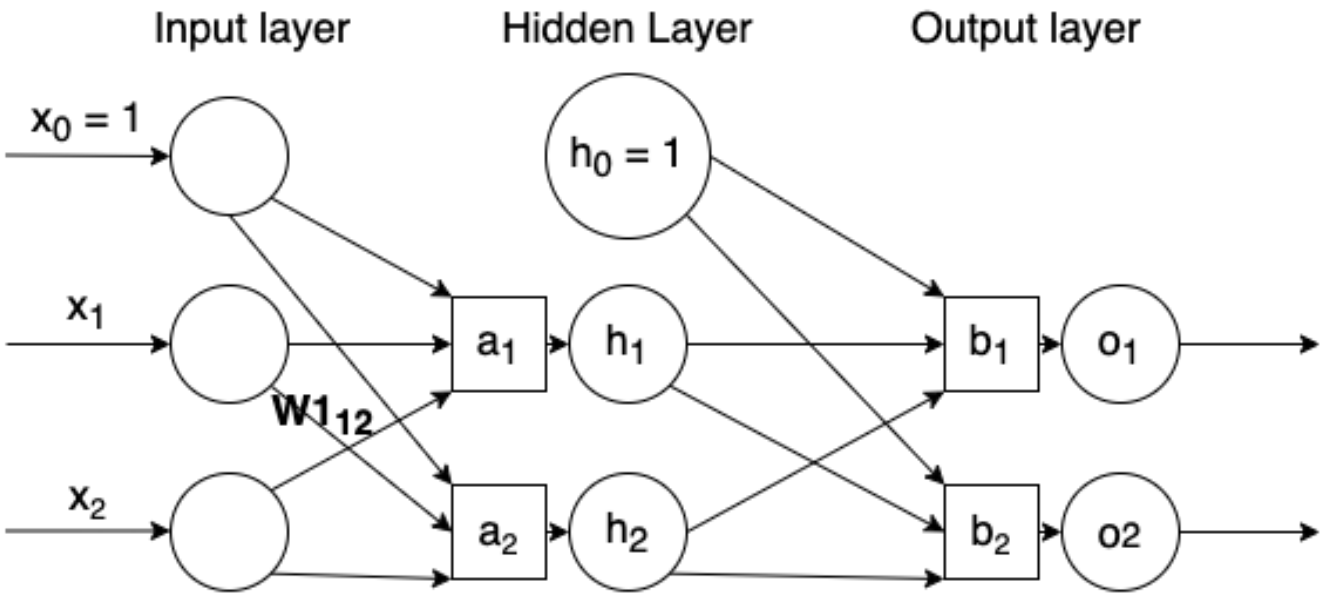
$$\frac{\partial O_2}{\partial b_2} = O_2(1 - O_2) = g(b_2)(1 - g(b_2))$$

$$b_2 = W2_{02} h_0 + W2_{12} h_1 + W2_{22} h_2$$

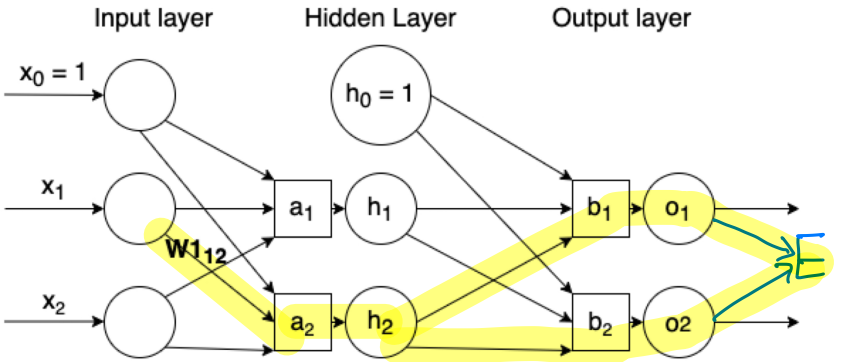
$$\frac{\partial b_2}{\partial W2_{12}} = h_1$$

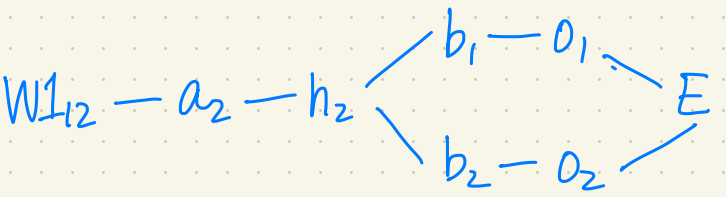
$$\frac{\partial E}{\partial W2_{12}} = 2(O_2 - y_2)\, O_2(1 - O_2)\, h_1$$

Calculating a gradient for $W1_{ij}$



$\frac{\partial error}{\partial W1_{12}} = ?$





chain rule:

$$\frac{d f(g(x))}{dx} = f'(g(x)) * \frac{dg(x)}{dx}$$

$$\frac{\partial E}{\partial W1_{12}} = \frac{\partial E}{\partial o_1}\frac{\partial o_1}{\partial b_1}\frac{\partial b_1}{\partial h_2}\frac{\partial h_2}{\partial a_2}\frac{\partial a_2}{\partial W1_{12}} + \frac{\partial E}{\partial o_2}\frac{\partial o_2}{\partial b_2}\frac{\partial b_2}{\partial h_2}\frac{\partial h_2}{\partial a_2}\frac{\partial a_2}{\partial W1_{12}}$$

$$= \left(\frac{\partial E}{\partial o_1}\frac{\partial o_1}{\partial b_1}\frac{\partial b_1}{\partial h_2} + \frac{\partial E}{\partial o_2}\frac{\partial o_2}{\partial b_2}\frac{\partial b_2}{\partial h_2}\right)\frac{\partial h_2}{\partial a_2}\frac{\partial a_2}{\partial W1_{12}}$$

When should we use Neural Network?

- High dimensional or real-valued inputs, noisy (sensor) data.
- Form of target function is unknown (no model).
- Not important for humans to explain the learned function.

When should we NOT use Neural Network?

- Difficult to determine the network structure (number of layers, number of neurons).
- Difficult to interpret weights, especially in multi-layered networks.
- Tendency to over-fit in practice (poor predictions outside of the range of values it was trained on).

Decision Tree v.s. Neural Network

- Data types.
  DT: tabular data. NN: images, audio, text.
- Size of data set.
  DT: works well w/ little data. NN: requires lots of data. ^ overfits easily.
- Form of target function.
  DT: nested if-then-else statement. NN: arbitrary function.
- The architecture.
  DT: few params NN: lots of params, all are critical.
- Interpret the learned function.
  DT: easily interpretable. NN: black box, difficult to ^ interpret.
- Time available for training and classification.
  DT: fast NN: slow to train and test.

Revisiting the Learning Goals

By the end of the lecture, you should be able to

- Explain the steps of the gradient descent algorithm.
- Explain how we can modify gradient descent to speed up learning and ensure convergence.
- Describe the back-propagation algorithm including the forward and backward passes.
- Compute the gradient for a weight in a multi-layer feed-forward neural network.
- Describe situations in which it is appropriate to use a neural network or a decision tree.