Learning Linear Classifiers

$$S_n = \{\bar{x}^{(i)}, y^{(i)}\}_{i=1}^{n} \qquad \bar{x} \in \mathbb{R}^d \quad y \in \{-1, 1\}$$

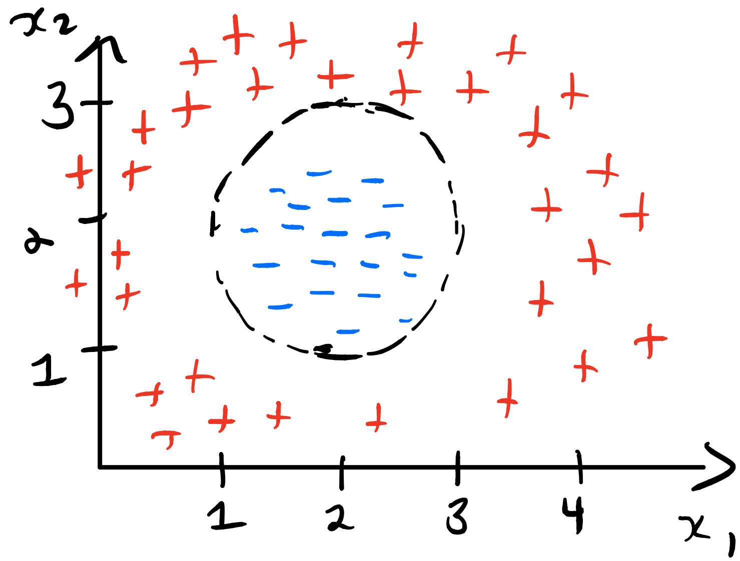
$$h(\bar{x}^{(i)}; \bar{\theta}) = sign(\bar{\theta} \cdot \bar{x})$$

Goal: learn model parameters $\bar{\theta}$ so as to minimize loss over the training examples

$$J(\bar{\theta}) = \frac{1}{n} \sum_{i=1}^{n} Loss(y^{(i)}(\bar{\theta} \cdot \bar{x}^{(i)}))$$

What if the data are not linearly separable?

E.g., $\bar{x} = [x_1, x_2]$,



Hint: eqn for a circle centered at h,k with radius r

$$(x-h)^2 + (y-k)^2 = r^2$$

Given the training data illustrated above and the feature mapping, find a corresponding set of model parameters that perfectly separates the data in the new feature space.

What if the data are not linearly separable?

Solution →

Idea: starting with the eqn. for a circle, with radius 1 and center at 2,2, work out the coefficient for each element in the new feature space

$$(x_1-2)^2 + (x_2-2)^2 = 1$$

$$x_1^2 - 4x_1 + 4 + x_2^2 - 4x_2 + 4 = 1$$

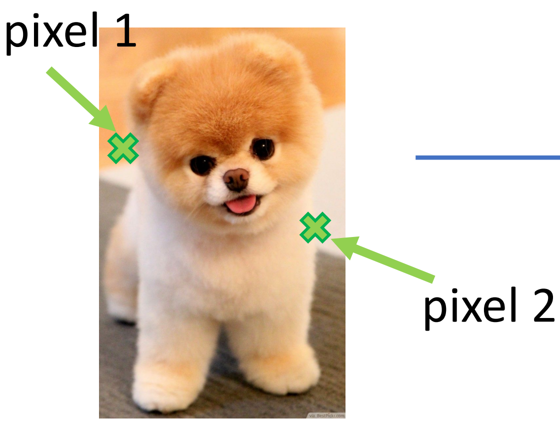
$$7 + x_1^2 + x_2^2 - 4x_1 - 4x_2 = 0$$

recall $\phi(\bar{x}) = [1, x_1^2, x_2^2, -2x_1, -2x_2]$ and $\bar{\theta} \cdot \phi(x) = 0$ decision boundary.

$\therefore \quad \bar{\theta} = [7, 1, 1, 2, 2]$

Problem: identifying the correct mapping can be difficult

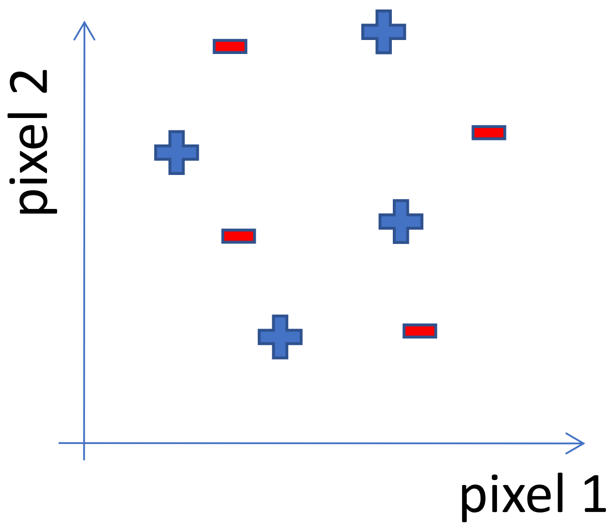
Feature Representations



Input

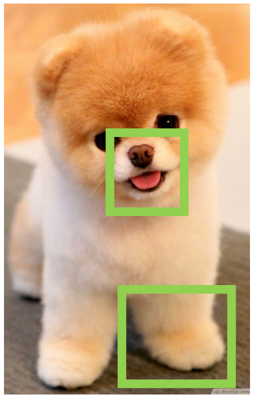
Learning algorithm

Input space



- ✚ Dog
- ▬ "Non"-Dogs

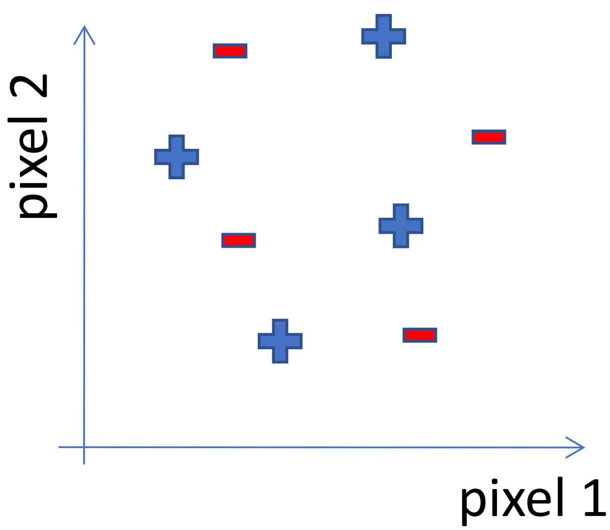
Feature Representations



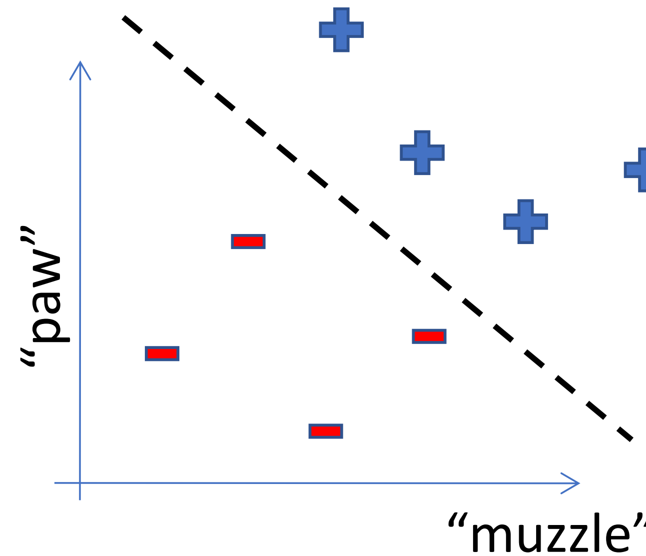
Input → Feature representation → Learning algorithm

- ✚ Dog
- ▬ “Non”-Dogs

Input space



Feature space



Representing Data

- The success of machine learning applications relies on having a good representation of the data.

- Machine learning practitioners put a lot of effort into “feature engineering”.

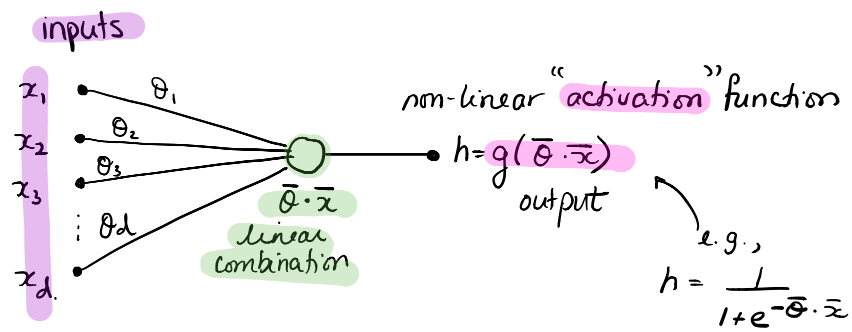
How can we develop good representations automatically?

Lecture ML #2: Introduction to Neural Networks

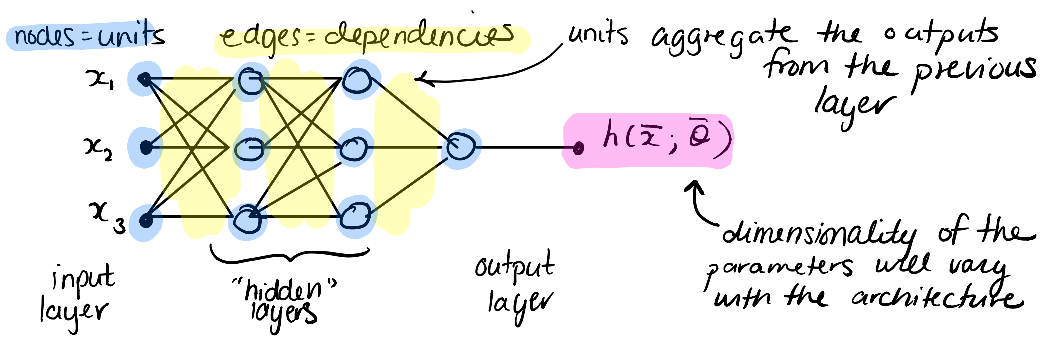
Feed-forward Neural Networks

- composed of computational units "neurons"

E.g.,



- neurons are arranged in a network composed of "layers"
- the specific arrangement corresponds to the "architecture"



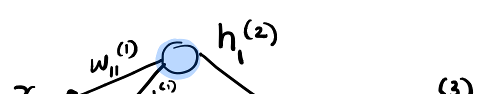
What did the "single layer" presented above represent? (w single unit)

* logistic regression, linear combination of inputs passed through a squashing function

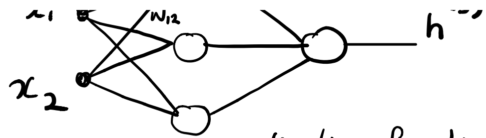
$$\hat{y} = h(\bar{x};\bar{\theta})$$

$$= \frac{1}{1+e^{-\bar{\theta}\cdot\bar{x}}}$$

Let's explore adding a hidden layer:



notation.



$W_{ki}^{(j)}$

j^{th} layer

k^{th} unit

i^{th} unit

activation function

$h_1^{(2)} = g(w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2)$

$h_2^{(2)} = g(w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2)$

$h_3^{(2)} = g(w_{31}^{(1)} x_1 + w_{32}^{(1)} x_2)$

forward propagation:

$h^{(3)} = f(w_{11}^{(2)} h_1^{(2)} + w_{12}^{(2)} h_2^{(2)} + w_{13} h_3^{(2)})$

not necessarily the same as the other activation functions

Common activation functions

$f(z) = \max\{0, z\}$ rectified linear unit "relu"

$f(z) = \text{sign}(z)$ thresholded.

$f(z) = e^{z} / \sum_k e^{z_k}$ softmax (common in multi-class settings)

$f(z) = 1/1+e^{-z}$ sigmoid.

$f(z) = \tanh(z)$ hyperbolic tangent similar to sigmoid but $(-1, 1)$

+ many more.

* linear activation is of little interest, might as well not have a hidden unit

What are the hidden layers doing?

The last layer is a linear combination of the output of all previous layers. In a way, the hidden layers essentially transform the input

i.e., maps the data to a new feature space in which we can learn a linear classifier.

Example:

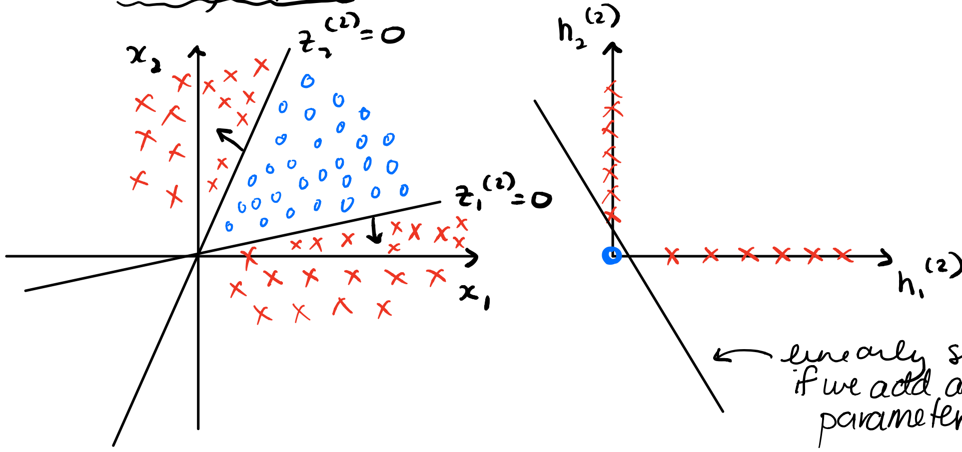
Given the following data you learn a NN

with two hidden units, described by the following equations:

$$z_1^{(2)} = w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 \qquad h_1^{(2)} = \max\{0, z_1^{(2)}\}$$

$$z_2^{(2)} = w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 \qquad h_2^{(2)} = \max\{0, z_2^{(2)}\}$$

Show that the points in the new feature space are linearly separable.



linearly separable if we add an offset parameter

Learning Neural Networks

$$S_n = \{\bar{x}^{(i)}, y^{(i)}\}_{i=1}^{n} \qquad \bar{x} \in \mathbb{R}^d \quad y \in \{-1, 1\}$$

Goal: learn model parameters $\bar{\theta}$ so as to minimize loss over the training examples

$$J(\bar{\theta}) = \frac{1}{n} \sum_{i=1}^{n} Loss(y^{(i)} h(\bar{x}^{(i)}; \bar{\theta}))$$

Overview of Optimization Proc.

Idea: sample a point at random, nudge parameters toward values that would improve classification on that particular example

Steps:
(0) Initialize parameters to small random values
(1) Select a point at random
(2) Update the parameters based on that point and the gradient:

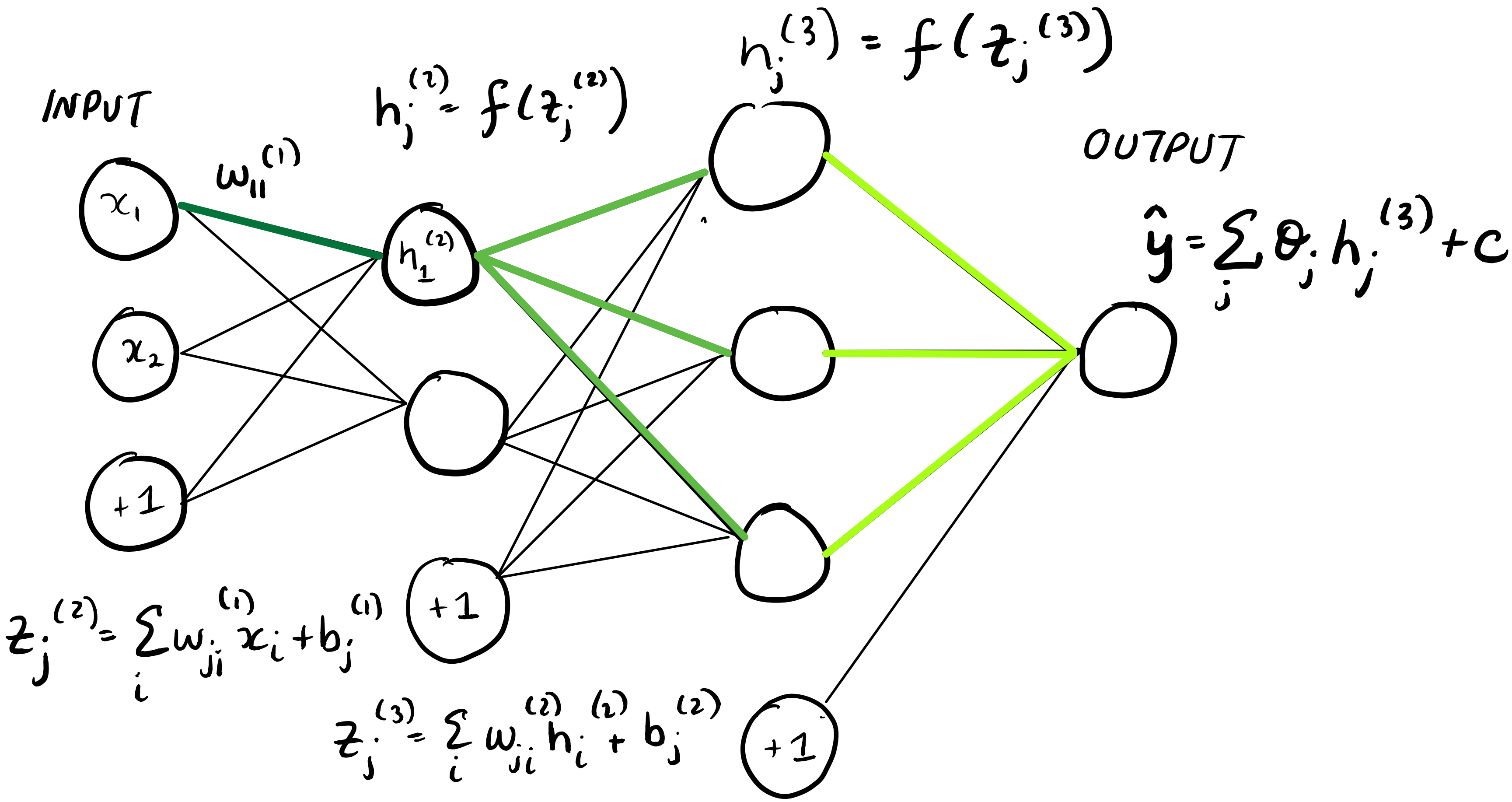
$$\bar{\theta}^{(k+1)} = \bar{\theta}^{(k)} - \eta_k \nabla_{\bar{\theta}} Loss(y^{(i)} h(\bar{x}^{(i)}; \bar{\theta}))$$

Optimization Details

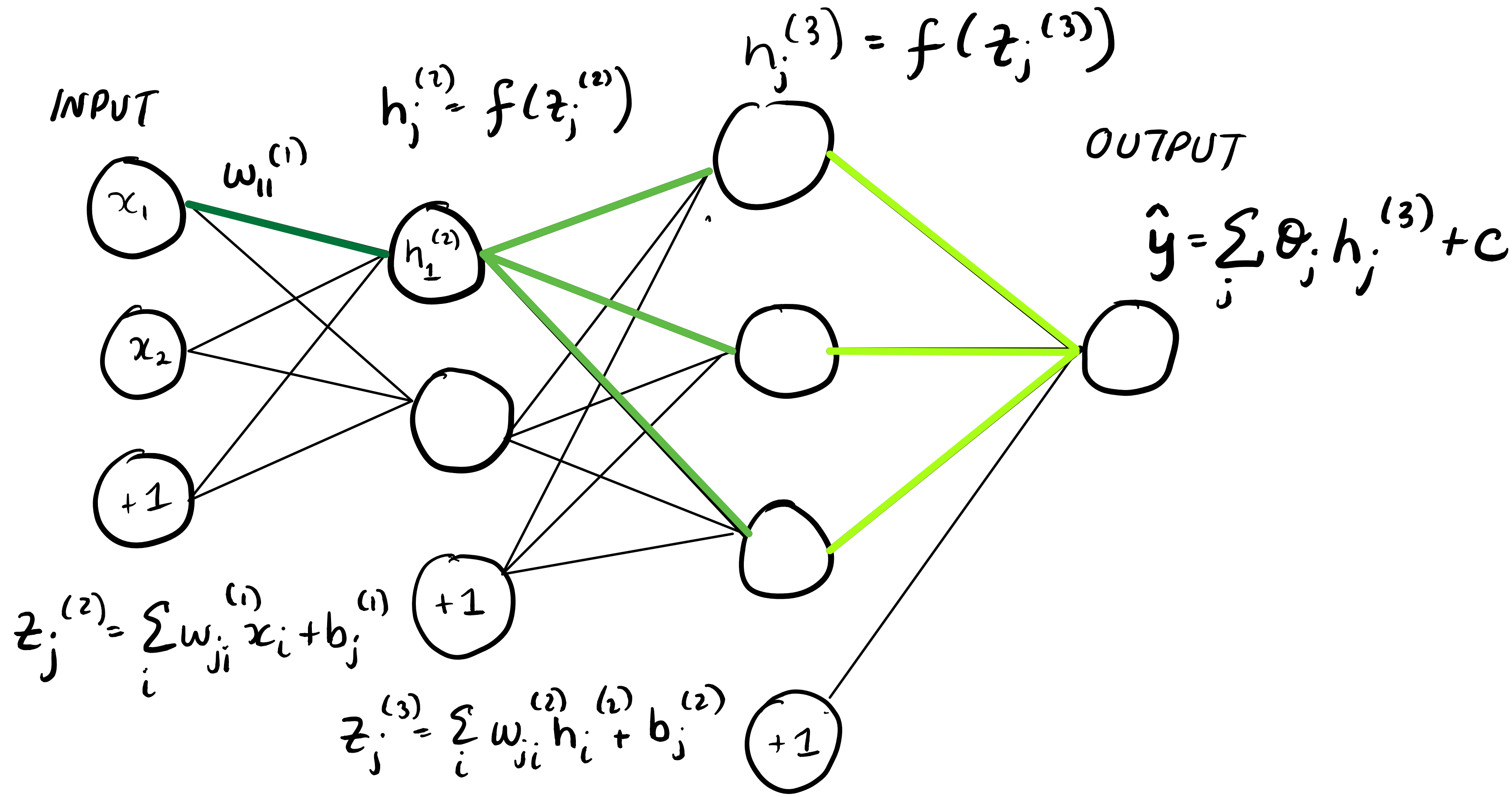
Details that need to be worked out:

1) How to evaluate the derivatives (when there is a hidden layer)
2) How to initialize the parameters
3) How to set learning rate
4) How to reduce likelihood of overfitting

SGD – two-hidden layer NN



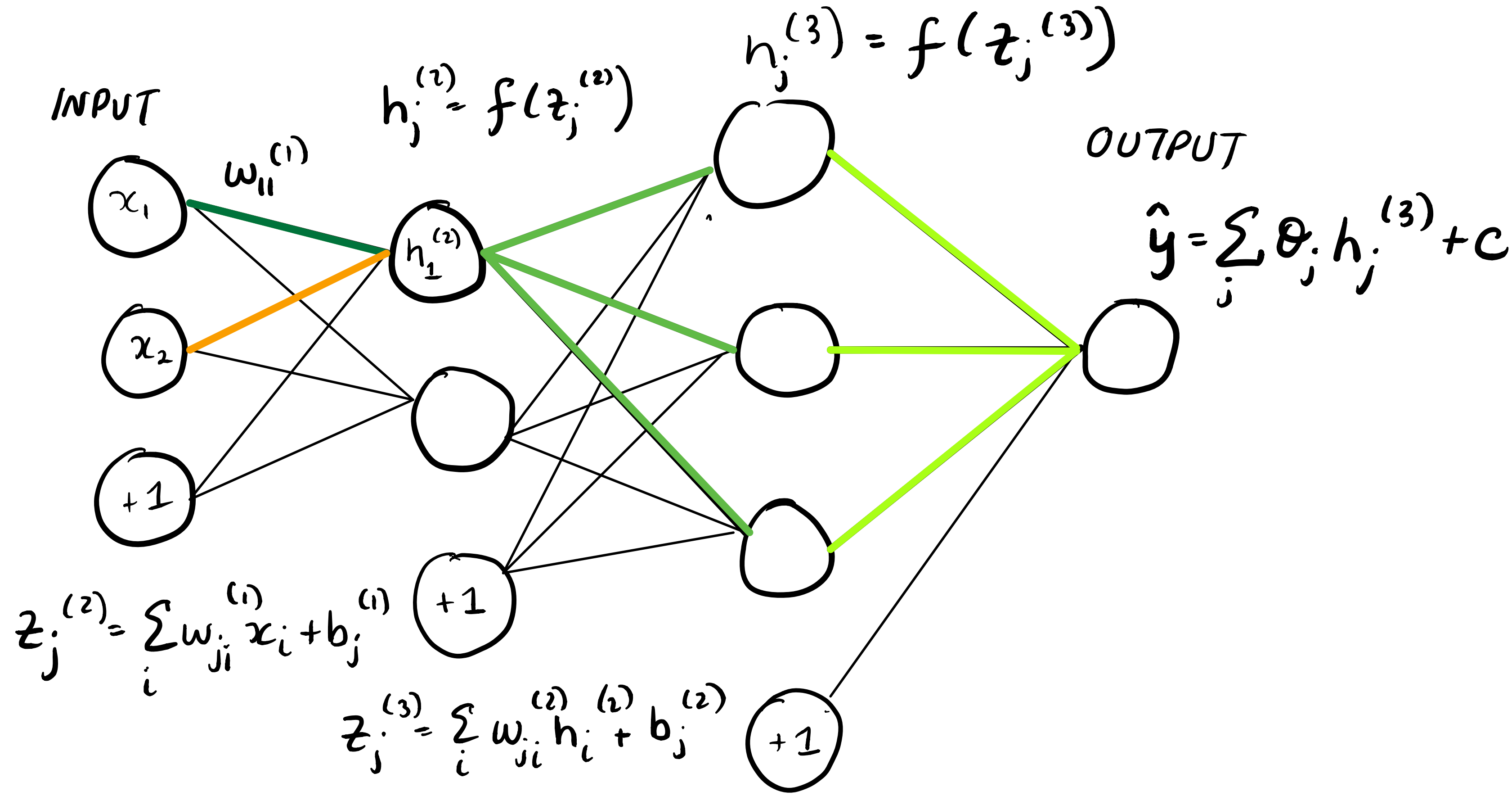
SGD – two-hidden layer NN



Chain Rule

$$\frac{\partial L}{\partial w_{11}^{(1)}} = \frac{\partial L}{\partial \hat{y}} \sum_{j=1}^{3} \frac{\partial \hat{y}}{\partial h_j^{(3)}} \cdot \frac{\partial h_j^{(3)}}{\partial z_j^{(3)}} \cdot \frac{\partial z_j^{(3)}}{\partial h_1^{(2)}} \cdot \frac{\partial h_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial w_{11}^{(1)}}$$

SGD – two-hidden layer NN



$$\frac{\partial L}{\partial w_{12}^{(1)}} = \frac{\partial L}{\partial \hat{y}} \sum_{j=1}^{3} \frac{\partial \hat{y}}{\partial h_j^{(3)}} \cdot \frac{\partial h_j^{(3)}}{\partial z_j^{(3)}} \cdot \frac{\partial z_j^{(3)}}{\partial h_1^{(2)}} \cdot \frac{\partial h_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial w_{12}^{(1)}}$$

local derivatives are shared!

A Generalized Approach

K

PYTORCH

TensorFlow

- Generalized approach to computing partial derivatives
- As long as your neural network fits the requirements, you do not need to derive the derivatives yourself!

Overview of Optimization Proc.

Idea: sample a point at random, nudge parameters toward values that would improve classification on that particular example

Steps:
(0) Initialize parameters to small random values
(1) Select a point at random
(2) Update the parameters based on that point and the gradient:

$$\bar{\theta}^{(k+1)} = \bar{\theta}^{(k)} - \eta_k \nabla_{\bar{\theta}} Loss(y^{(i)} h(\bar{x}^{(i)}; \bar{\theta}))$$

Note: there are faster optimizers (a lot of ongoing research in this area)
Popular techniques: SGD with Momentum, Nesterov Accelerated Gradient, AdaGrad, RMSProp, and **Adam Optimization** (pg. 293-300)