

# Lecture 08

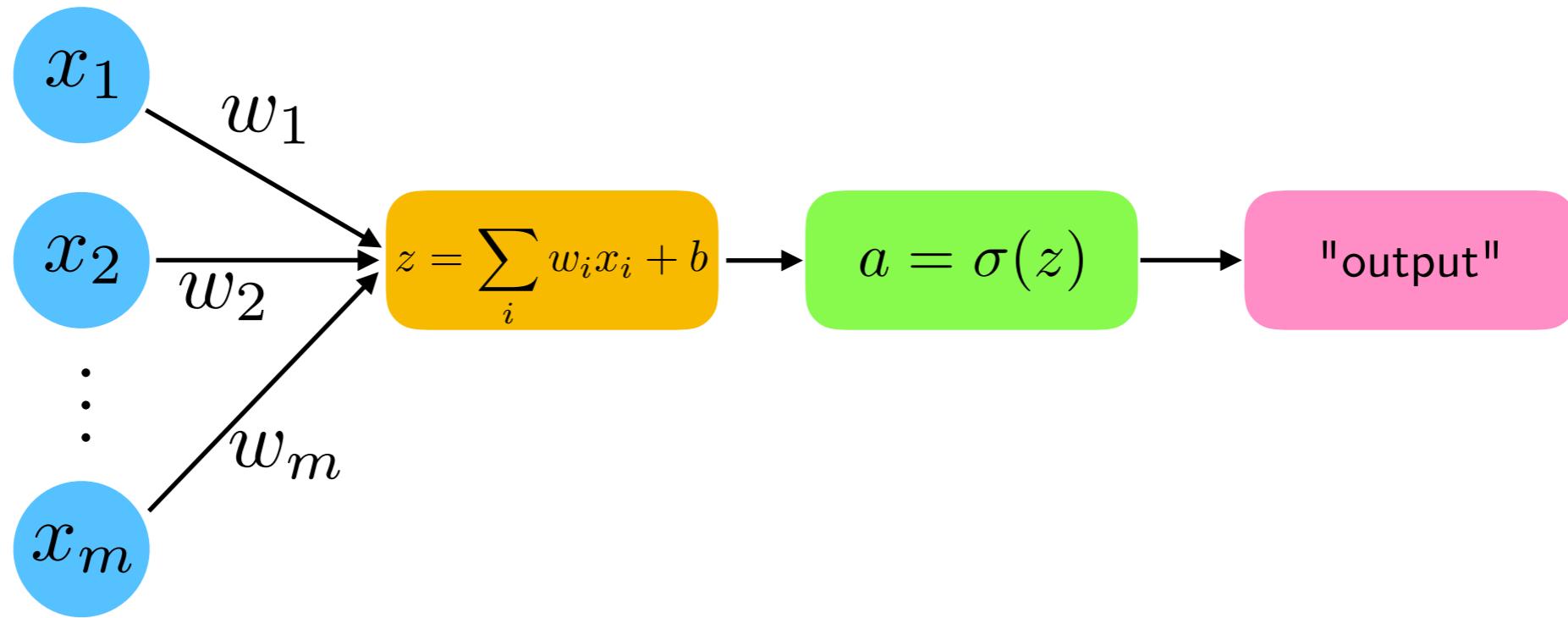
# Logistic Regression and Multi-class Classification

STAT 479: Deep Learning, Spring 2019

Sebastian Raschka

<http://stat.wisc.edu/~sraschka/teaching/stat479-ss2019/>

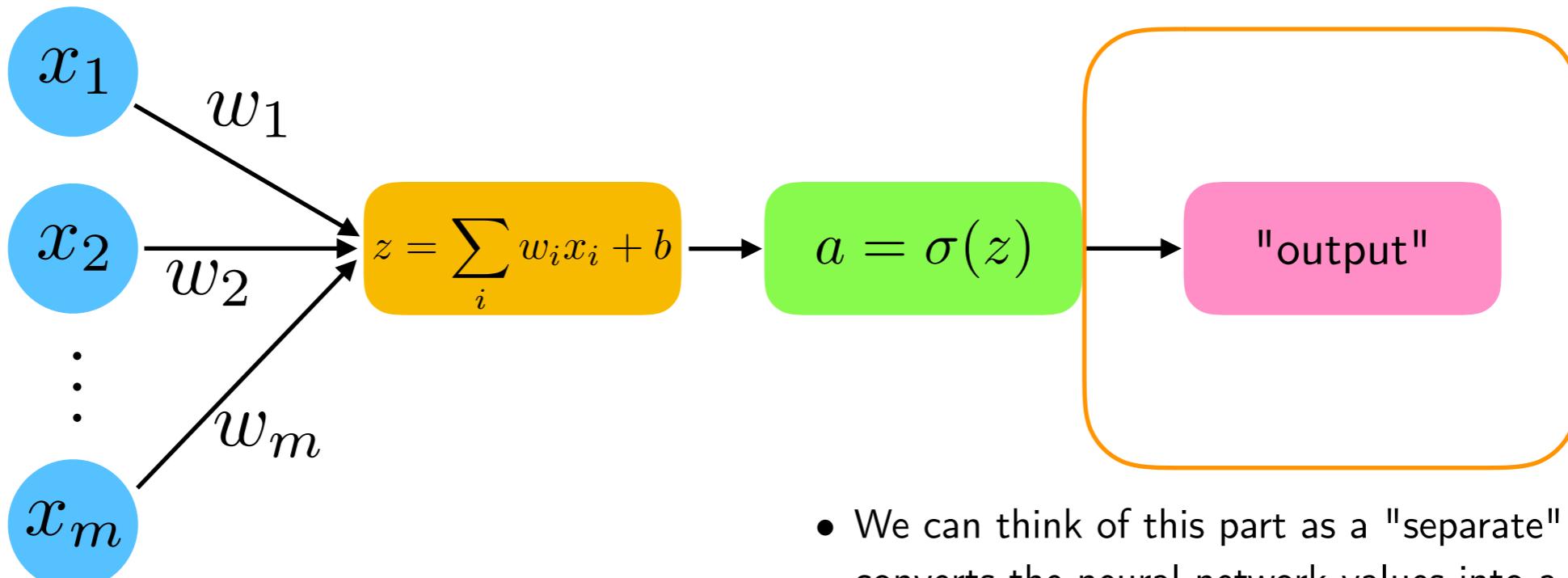
# Homework 2 Recap



Generic Neuron representation

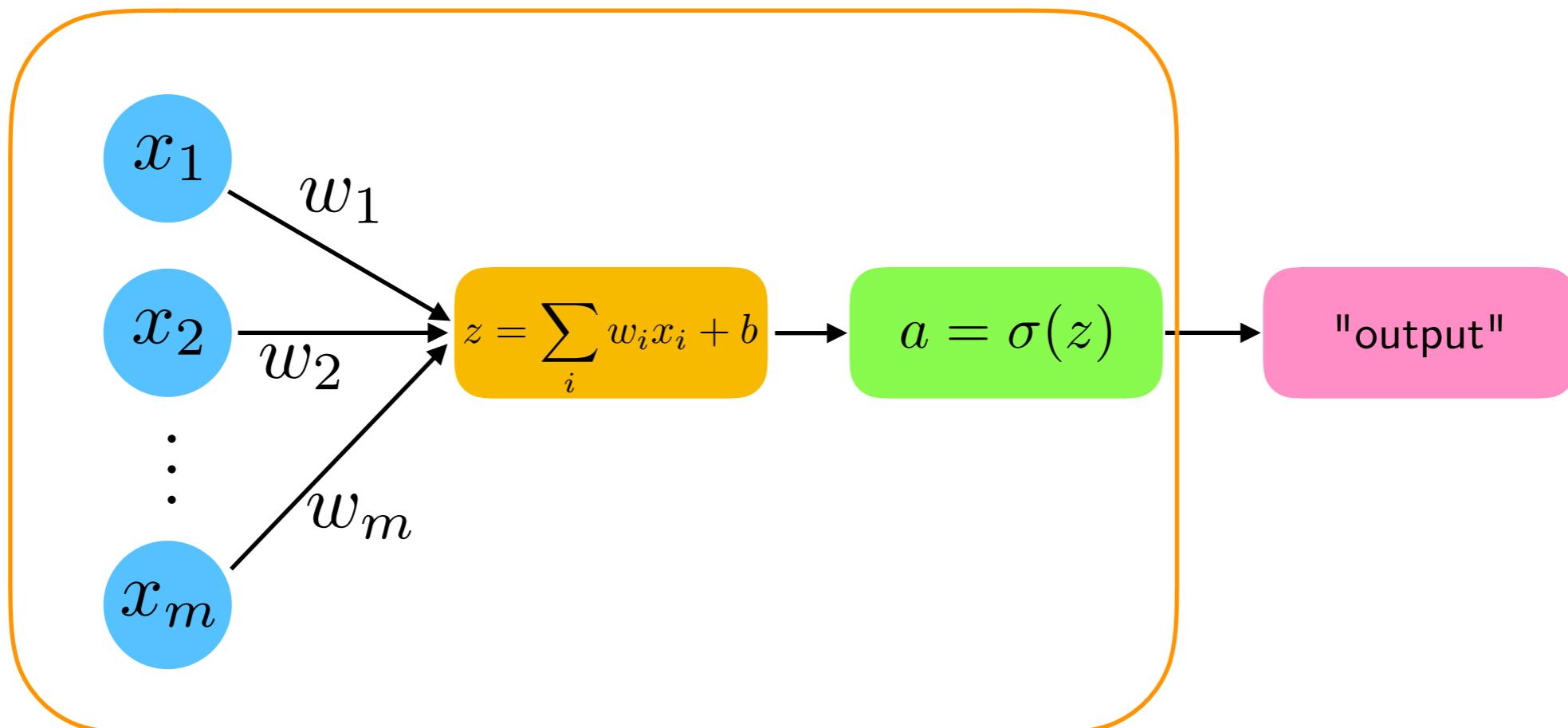
(Perceptron, ADALINE, Linear Regression, and later today: Logistic Regression)

# Homework 2 Recap



- We can think of this part as a "separate" part that converts the neural network values into a class label, for example; e.g., via a threshold function
- Predicted class labels are not used during training (except by the Perceptron)
- ADALINE, Logistic Regression (later today), and all common types of multi-layer neural networks don't use predicted class labels for optimization as a threshold function is not smooth

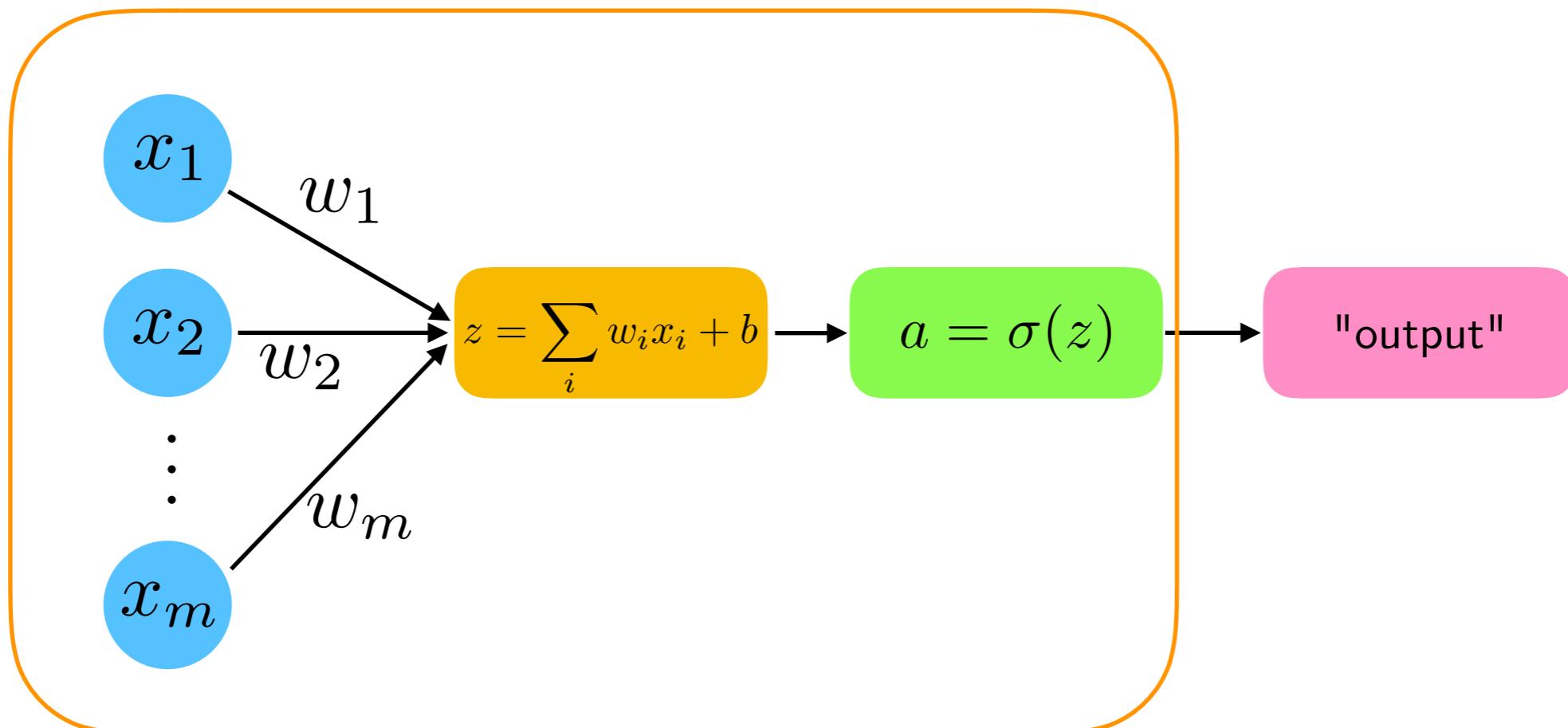
# Homework 2 Recap



- For training, only this part matters!

$$\mathcal{L} = \frac{1}{n} \sum_i (y^{[i]} - a^{[i]})^2$$

# Homework 2 Recap



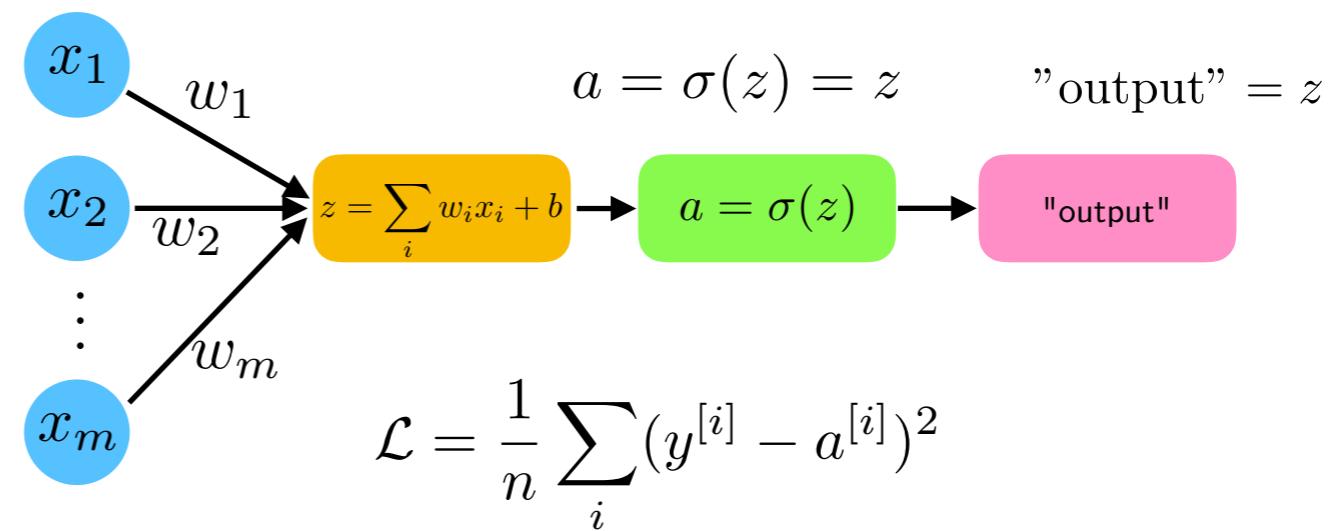
- For training, only this part matters!

$$\mathcal{L} = \frac{1}{n} \sum_i (y^{[i]} - a^{[i]})^2$$

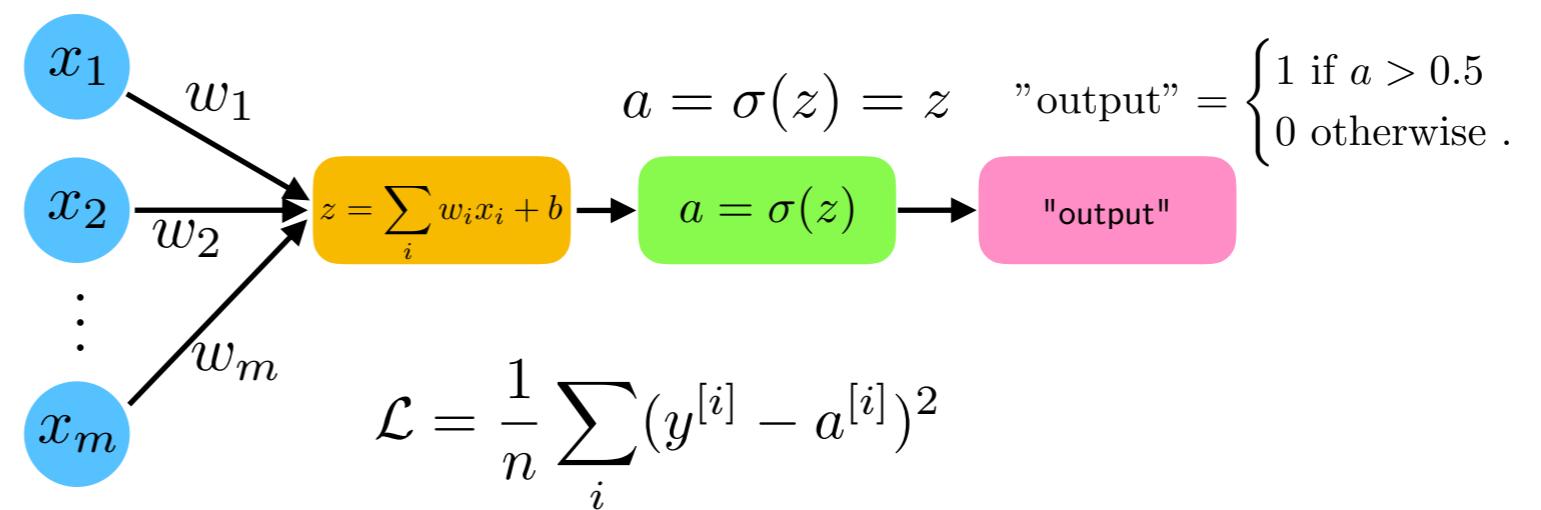
Want:  $\frac{\partial \mathcal{L}}{\partial w_i}$  and  $\frac{\partial \mathcal{L}}{\partial b}$

# Homework 2 Recap

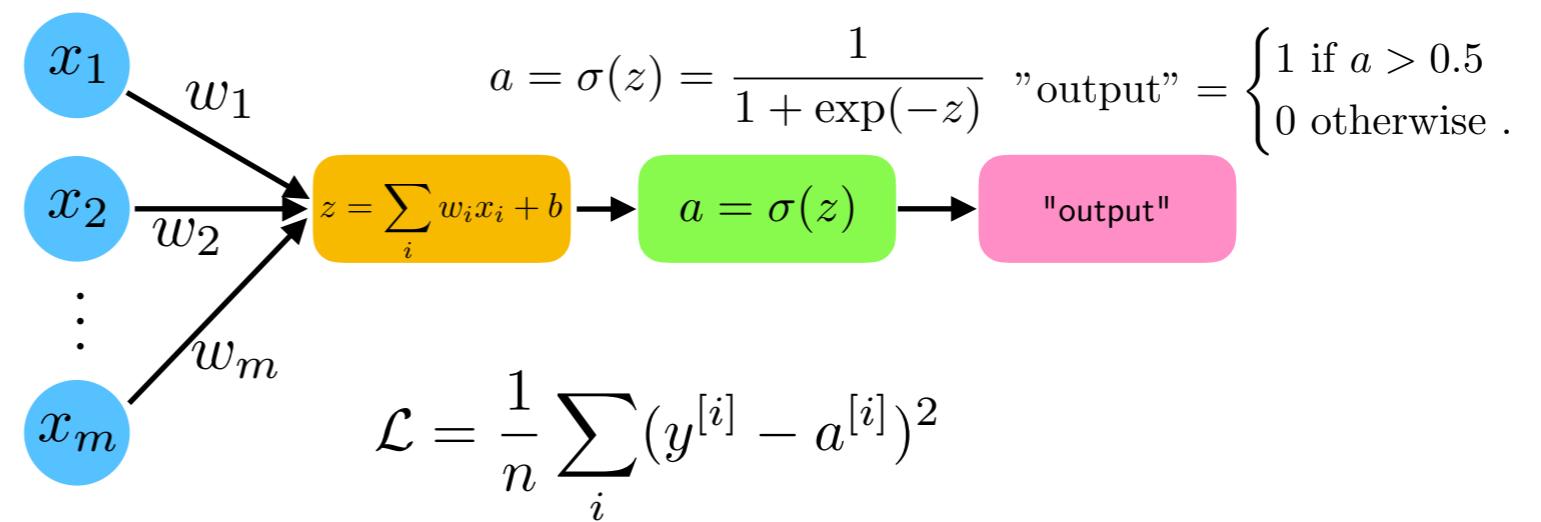
a) Linear regression



b) ADALINE

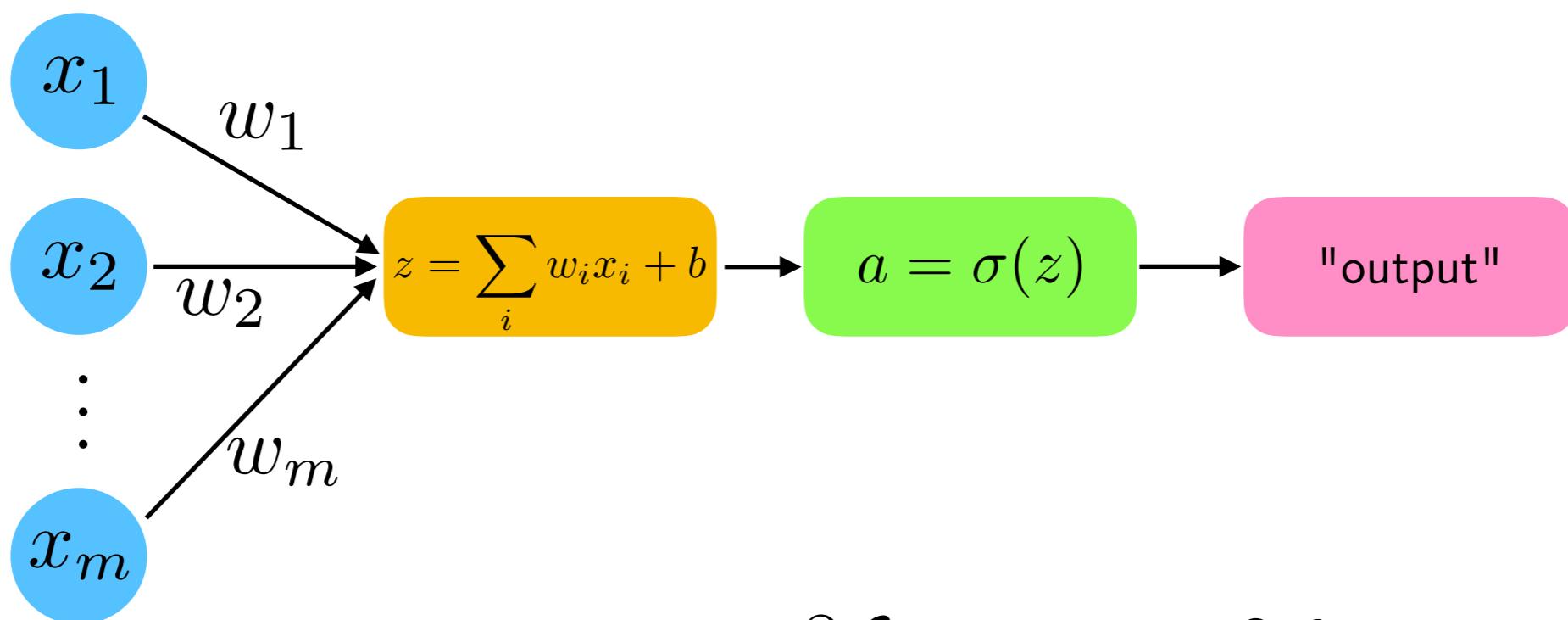


c) HW2 Neuron



# Homework 2 Recap

$$\mathcal{L} = (y^{[i]} - a^{[i]})^2$$



Want:  $\frac{\partial \mathcal{L}}{\partial w_i}$  and  $\frac{\partial \mathcal{L}}{\partial b}$

Why could

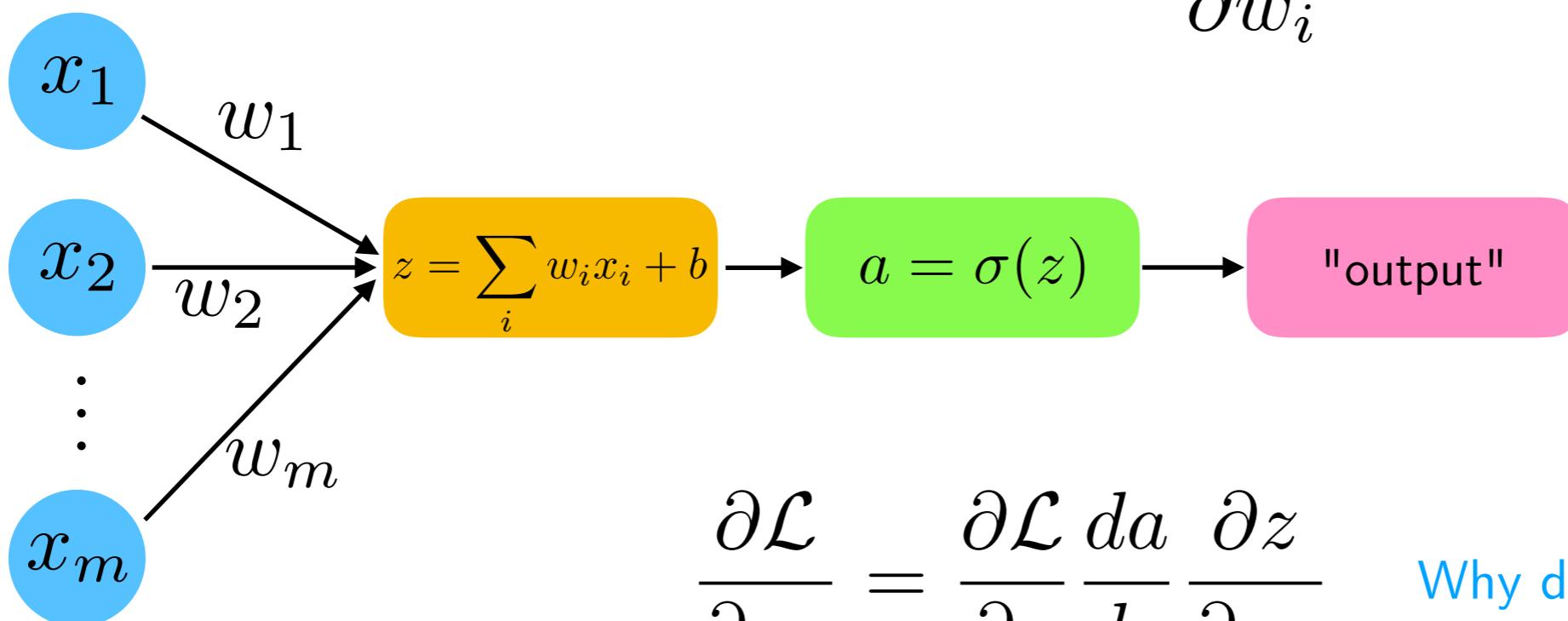
$$a = \sigma(z) = \frac{1}{1 + \exp(-z)}$$

be better than ADALINE's linear activation?

# Homework 2 Recap

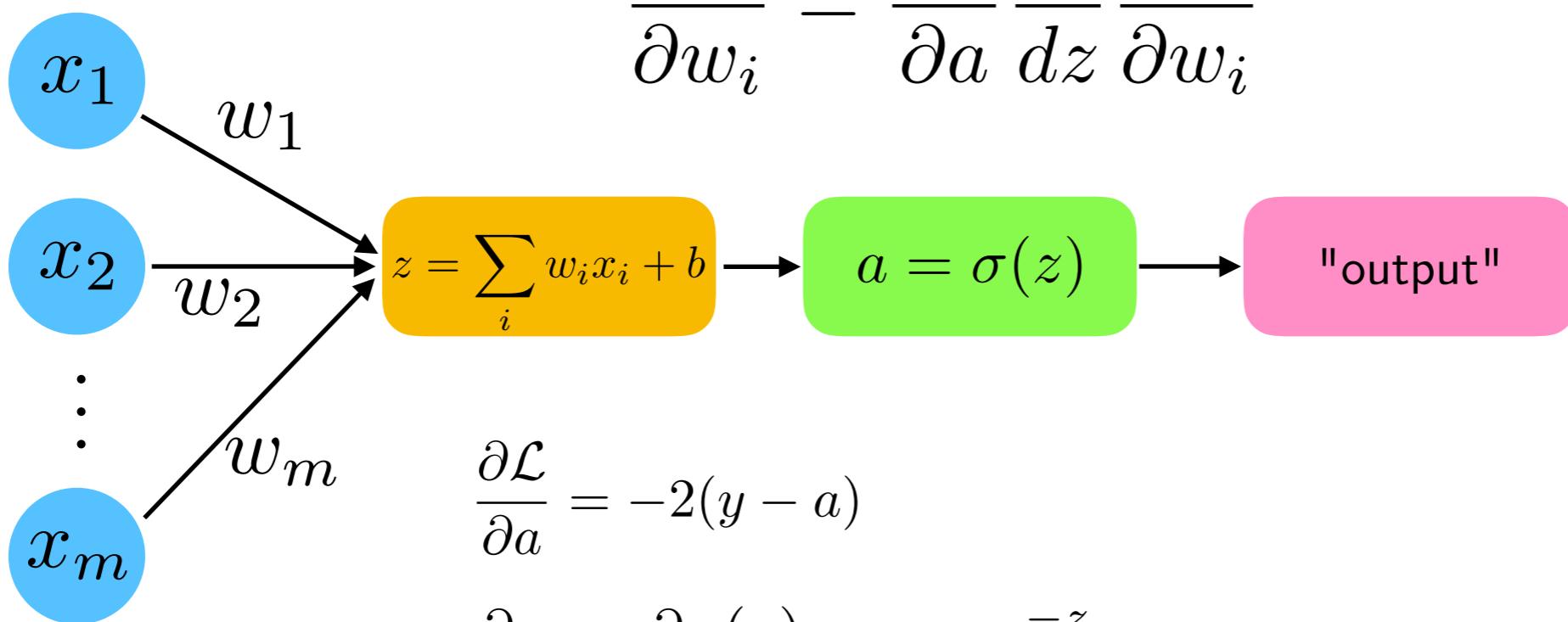
$\mathcal{L} = (y^{[i]} - a^{[i]})^2$  For simplicity, suppose we have only 1 training example

Want:  $\frac{\partial \mathcal{L}}{\partial w_i}$  and  $\frac{\partial \mathcal{L}}{\partial b}$



# Homework 2 Recap

$\mathcal{L} = (y^{[i]} - a^{[i]})^2$  For simplicity, suppose we have only 1 training example



$$\frac{\partial \mathcal{L}}{\partial a} = -2(y - a)$$

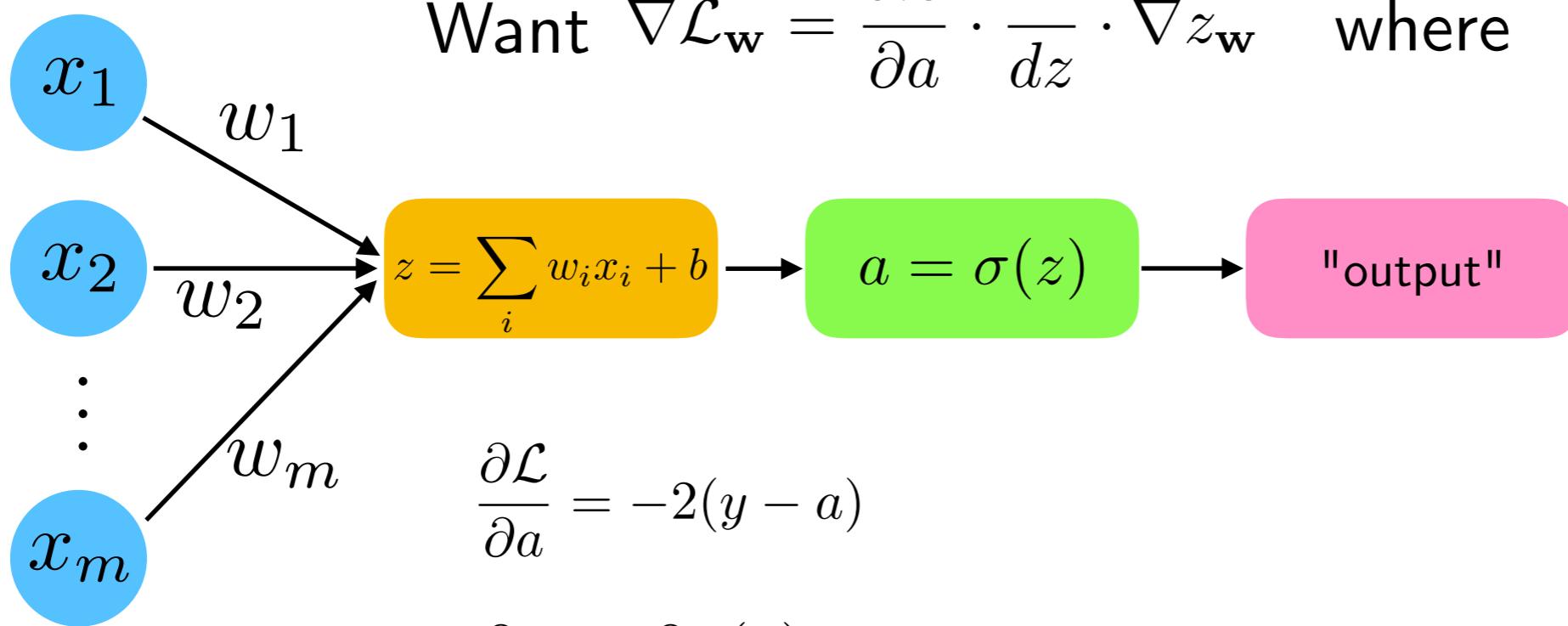
$$\frac{\partial a}{\partial z} = \frac{\partial \sigma(z)}{\partial z} = \frac{e^{-z}}{(1 + e^{-z})^2} = \sigma(z) \cdot (1 - \sigma(z))$$

$$\frac{\partial z}{\partial w_i} = x_i$$

# Homework 2 Recap

$\mathcal{L} = (y^{[i]} - a^{[i]})^2$  For simplicity, suppose we have only 1 training example

but multiple weights



Want  $\nabla \mathcal{L}_w = \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{da}{dz} \cdot \nabla z_w$  where  $\nabla \mathcal{L}_w =$

$$\begin{bmatrix} \frac{\partial \mathcal{L}}{w_1} \\ \frac{\partial \mathcal{L}}{w_2} \\ \vdots \\ \frac{\partial \mathcal{L}}{w_i} \end{bmatrix}$$

$$\frac{\partial \mathcal{L}}{\partial a} = -2(y - a)$$

$$\frac{\partial a}{\partial z} = \frac{\partial \sigma(z)}{\partial z} = \frac{e^{-z}}{(1 + e^{-z})^2} = \sigma(z) \cdot (1 - \sigma(z))$$

$$\nabla_w z = [x_1 \quad x_2 \quad \dots \quad x_m]^\top$$

# Homework 2 Recap

Multiple training examples:

$$\mathcal{L} = (y^{[i]} - a^{[i]})^2 \xrightarrow{\text{red arrow}} \mathcal{L} = \frac{1}{n} \sum_i (y^{[i]} - a^{[i]})^2$$

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial a} \frac{da}{dz} \frac{\partial z}{\partial w_i}$$

$$\frac{\partial \mathcal{L}}{\partial w_j} = \boxed{\frac{1}{n} \sum_{i=1}^n} - 2(y^{[i]} - a^{[i]}) \cdot \sigma(z^{[i]}) \cdot (1 - \sigma(z^{[i]})) \cdot x_j^{[i]}$$

Only this part is new (compared to 1 training example)!

# Homework 2 Recap

Multiple training examples:

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial a} \frac{da}{dz} \frac{\partial z}{\partial w_i}$$

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{1}{n} \sum_{i=1}^n -2(y^{[i]} - a^{[i]}) \cdot \sigma(z^{[i]}) \cdot (1 - \sigma(z^{[i]})) \cdot x_j^{[i]}$$

Vectorized:

$$\frac{\partial \mathcal{L}}{\partial w_j} = -\frac{2}{n} (\mathbf{y} - \mathbf{a}) \odot \sigma(\mathbf{z}) \odot (1 - \sigma(\mathbf{z})) \mathbf{x}_j^\top$$

# Homework 2 Recap

Multiple training examples:

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial a} \frac{da}{dz} \frac{\partial z}{\partial w_i}$$

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{1}{n} \sum_{i=1}^n -2(y^{[i]} - a^{[i]}) \cdot \sigma(z^{[i]}) \cdot (1 - \sigma(z^{[i]})) \cdot x_j^{[i]}$$

Vectorized:

$$\frac{\partial \mathcal{L}}{\partial w_j} = -\frac{2}{n} (\mathbf{y} - \mathbf{a}) \odot \sigma(\mathbf{z}) \odot (1 - \sigma(\mathbf{z})) \mathbf{x}_j^\top$$

Can you spot the problem?

# Homework 2 Recap

Multiple training examples:

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial a} \frac{da}{dz} \frac{\partial z}{\partial w_i}$$

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{1}{n} \sum_{i=1}^n -2(y^{[i]} - a^{[i]}) \cdot \sigma(z^{[i]}) \cdot (1 - \sigma(z^{[i]})) \cdot x_j^{[i]}$$

Vectorized:

$$\frac{\partial \mathcal{L}}{\partial w_j} = -\frac{2}{n} (\mathbf{y} - \mathbf{a}) \odot \sigma(\mathbf{z}) \odot (1 - \sigma(\mathbf{z})) \mathbf{x}_j^\top$$

In code, this is ok, in proper math, this is a vector of 1's now

# Homework 2 Recap

Multiple training examples:

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial a} \frac{da}{dz} \frac{\partial z}{\partial w_i}$$

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{1}{n} \sum_{i=1}^n -2(y^{[i]} - a^{[i]}) \cdot \sigma(z^{[i]}) \cdot (1 - \sigma(z^{[i]})) \cdot x_j^{[i]}$$

$$\frac{\partial \mathcal{L}}{\partial w_j} = -\frac{2}{n}(\mathbf{y} - \mathbf{a}) \odot \sigma(\mathbf{z}) \odot (1 - \sigma(\mathbf{z})) \mathbf{x}_j^\top \quad \text{where } \mathbf{x} \in \mathbb{R}^{m \times 1}$$

Multiple training examples and features:

$$\nabla_{\mathbf{w}} \mathcal{L} = \left( -\frac{2}{n}(\mathbf{y} - \mathbf{a}) \odot \sigma(\mathbf{z}) \odot (1 - \sigma(\mathbf{z}))^\top \mathbf{X} \right)^\top \quad \text{where } \mathbf{X} \in \mathbb{R}^{n \times m}$$
$$\nabla_{\mathbf{w}} \mathcal{L} = \mathbf{X}^\top \left( -\frac{2}{n}(\mathbf{y} - \mathbf{a}) \odot \sigma(\mathbf{z}) \odot (1 - \sigma(\mathbf{z})) \right)$$

# Homework 2 Recap

$$\mathcal{L} := 0, \quad w_1 := 0, \quad w_2 := 0, \quad b := 0$$

for  $i = 1$  to  $i = n$ :

$$z^{[i]} = w^T x^{[i]} + b$$

$$a^{[i]} = \sigma(z^{[i]})$$

$$\mathcal{L} \pm (y^{[i]} - a^{[i]})^2$$

$$\frac{\partial \mathcal{L}}{\partial z^{[i]}} = -2(y^{[i]} - a^{[i]}) \cdot \sigma(z^{[i]})(1 - \sigma(z^{[i]}))$$

$$\frac{\partial \mathcal{L}}{\partial w_1} \pm x_1^{[i]} \frac{\partial \mathcal{L}}{\partial z^{[i]}}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} \pm x_2^{[i]} \frac{\partial \mathcal{L}}{\partial z^{[i]}}$$

$$\frac{\partial \mathcal{L}}{\partial b} \pm \frac{\partial \mathcal{L}}{\partial z^{[i]}}$$

$$\mathcal{L} := \mathcal{L}/n$$

$$\frac{\partial \mathcal{L}}{\partial w_1} := \frac{\partial \mathcal{L}}{\partial w_1}/n, \quad \frac{\partial \mathcal{L}}{\partial w_2} := \frac{\partial \mathcal{L}}{\partial w_2}/n, \quad \frac{\partial \mathcal{L}}{\partial b} := \frac{\partial \mathcal{L}}{\partial b}/n$$

If still confused, think of the previous slide as a for-loop;  
we simply "vectorized" it :)

# Homework 2 Recap

Lastly, about the sigmoid derivative ...

$$\begin{aligned}\frac{d}{dz} \sigma(z) &= \frac{d}{dz} \frac{1}{1 + e^{-z}} \\&= \frac{d}{dz} (1 + e^{-z})^{-1} \quad [\text{apply chain rule}] \\&= -(1 + e^{-z})^{-2} \cdot \frac{d}{dz} (1 + e^{-z}) \quad [\text{apply sum rule}] \\&= -(1 + e^{-z})^{-2} \cdot \left( \frac{d}{dz} 1 + \frac{d}{dz} e^{-z} \right) \\&= -(1 + e^{-z})^{-2} \cdot \frac{d}{dz} e^{-z} \quad [\text{apply chain rule}] \\&= -(1 + e^{-z})^{-2} \cdot e^{-z} \frac{d}{dz} (-z) \\&= -(1 + e^{-z})^{-2} \cdot (-e^{-z}) \\&= \frac{1}{(1 + e^{-z})^2} \cdot e^{-z} \\&= \frac{e^{-z}}{(1 + e^{-z})^2}\end{aligned}$$

# Homework 2 Recap

Simplifying the sigmoid derivative ...

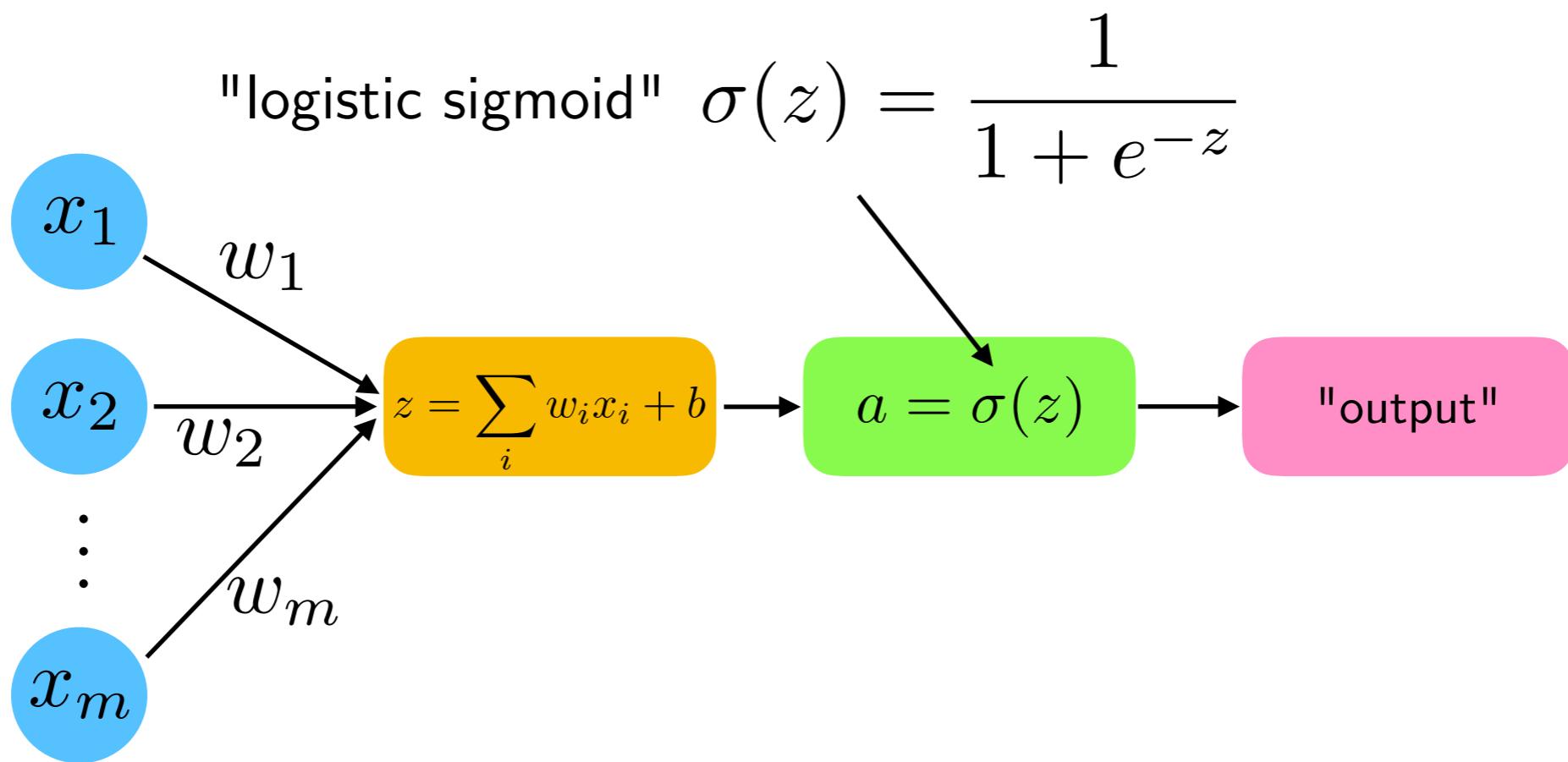
$$\begin{aligned}\frac{e^{-z}}{(1 + e^{-z})^2} &= \frac{e^{-z}}{1 + e^{-z}} \cdot \frac{1}{1 + e^{-z}} \\&= \frac{-1 + 1 + e^{-z}}{1 + e^{-z}} \cdot \frac{1}{1 + e^{-z}} \\&= \left( \frac{-1}{1 + e^{-z}} + \frac{1 + e^{-z}}{1 + e^{-z}} \right) \cdot \frac{1}{1 + e^{-z}} \\&= \left( \frac{-1}{1 + e^{-z}} + 1 \right) \cdot \frac{1}{1 + e^{-z}} \\&= \left( 1 - \frac{1}{1 + e^{-z}} \right) \cdot \frac{1}{1 + e^{-z}} \\&= (1 - \sigma(z)) \cdot \sigma(z)\end{aligned}$$

$$\frac{d\sigma(z)}{dz} = \frac{e^{-z}}{(1 + e^{-z})^2} = \sigma(z) \cdot (1 - \sigma(z))$$

# Logistic Regression for Binary Classification

# Logistic Regression Neuron

For binary classes  $y \in \{0, 1\}$

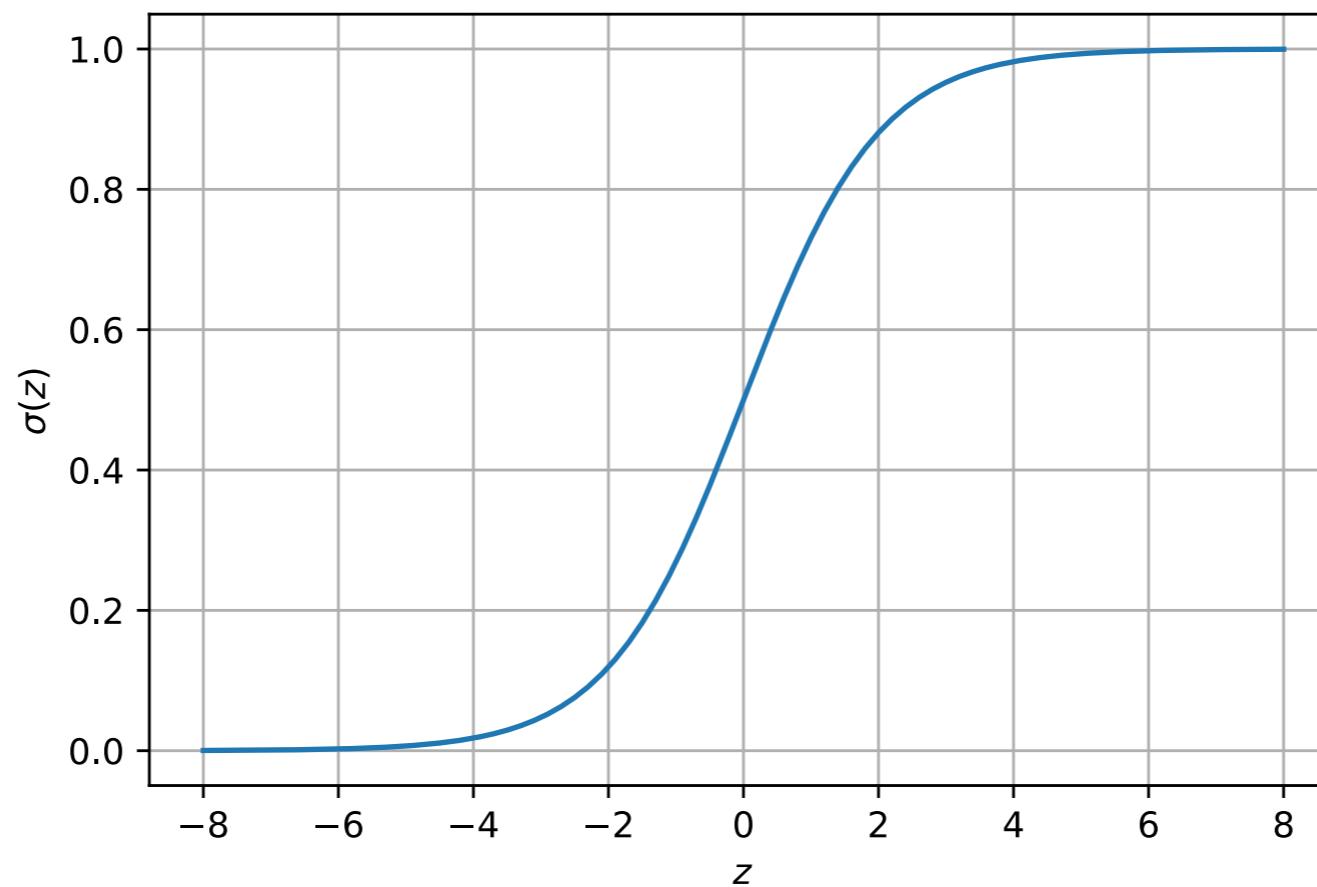


- Same as before, except different loss function
- in ADALINE and HW, we minimized the MSE loss:

$$\text{MSE} = \frac{1}{n} \sum_i (a^{[i]} - y^{[i]})^2$$

# Logistic Sigmoid Function

$$\sigma(z) = \frac{e^z}{1 + e^z} = \frac{1}{1 + e^{-z}}$$



# Logistic Regression

$$h(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$$

$$P(y|\mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{if } y = 1 \\ 1 - h(\mathbf{x}) & \text{if } y = 0 \end{cases}$$

want       $P(y = 0|\mathbf{x}) \approx 1 \quad \text{if } y = 0$

$$P(y = 1|\mathbf{x}) \approx 1 \quad \text{if } y = 1$$

# Logistic Regression

$$h(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$$

$$P(y|\mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{if } y = 1 \\ 1 - h(\mathbf{x}) & \text{if } y = 0 \end{cases}$$

here, hypothesis is just our activation function output

$$h(\mathbf{x}) = a$$

want  $P(y = 0|\mathbf{x}) \approx 1$  if  $y = 0$

$$P(y = 1|\mathbf{x}) \approx 1 \quad \text{if } y = 1$$

# Logistic Regression

$$h(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$$

$$P(y|\mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{if } y = 1 \\ 1 - h(\mathbf{x}) & \text{if } y = 0 \end{cases}$$

rewrite more compactly

want  $P(y = 0|\mathbf{x}) \approx 1 \quad \text{if } y = 0$

$$P(y = 1|\mathbf{x}) \approx 1 \quad \text{if } y = 1$$

$$P(y|\mathbf{x}) = a^y(1 - a)^{(1-y)}$$

# Logistic Regression

And for multiple training examples, we want to maximize:

$$P(y^{[i]}, \dots, y^{[n]} | \mathbf{x}^{[1]}, \dots, \mathbf{x}^{[n]}) = \prod_{i=1}^n P(y^{[i]} | \mathbf{x}^{[i]})$$

You may remember this as Maximum Likelihood Estimation from your other stats classes.

# Likelihood "Loss"

$$\begin{aligned} L(\mathbf{w}) &= P(\mathbf{y} \mid \mathbf{x}; \mathbf{w}) \\ &= \prod_{i=1}^n P(y^{(i)} \mid x^{(i)}; \mathbf{w}) \\ &= \prod_{i=1}^n \left( \sigma(z^{(i)}) \right)^{y^{(i)}} \left( 1 - \sigma(z^{(i)}) \right)^{1-y^{(i)}} \end{aligned}$$

# Log-Likelihood "Loss"

$$\begin{aligned}L(\mathbf{w}) &= P(\mathbf{y} \mid \mathbf{x}; \mathbf{w}) \\&= \prod_{i=1}^n P(y^{(i)} \mid x^{(i)}; \mathbf{w}) \\&= \prod_{i=1}^n \left( \sigma(z^{(i)}) \right)^{y^{(i)}} \left( 1 - \sigma(z^{(i)}) \right)^{1-y^{(i)}}\end{aligned}$$

In practice, it is easier to maximize the (natural) log of this equation, which is called the log-likelihood function:

$$\begin{aligned}l(\mathbf{w}) &= \log L(\mathbf{w}) \\&= \sum_{i=1}^n \left[ y^{(i)} \log (\sigma(z^{(i)})) + (1 - y^{(i)}) \log (1 - \sigma(z^{(i)})) \right]\end{aligned}$$

# Negative Log-Likelihood "Loss"

In practice, it is even more convenient to minimize negative log-likelihood instead of maximizing log-likelihood:

$$\mathcal{L}(\mathbf{w}) = -l(\mathbf{w})$$

$$= - \sum_{i=1}^n [y^{(i)} \log (\sigma(z^{(i)})) + (1 - y^{(i)}) \log (1 - \sigma(z^{(i)}))]$$

(in code, we also usually add a  $1/n$  scaling factor for further convenience, where  $n$  is the number of training examples or number of examples in a minibatch)

# Logistic Regression Loss

- So, "doing logistic regression" is similar to what we have done before
- in ADALINE and HW2, we minimized the MSE loss:

$$\text{MSE} = \frac{1}{n} \sum_i (a^{[i]} - y^{[i]})^2$$

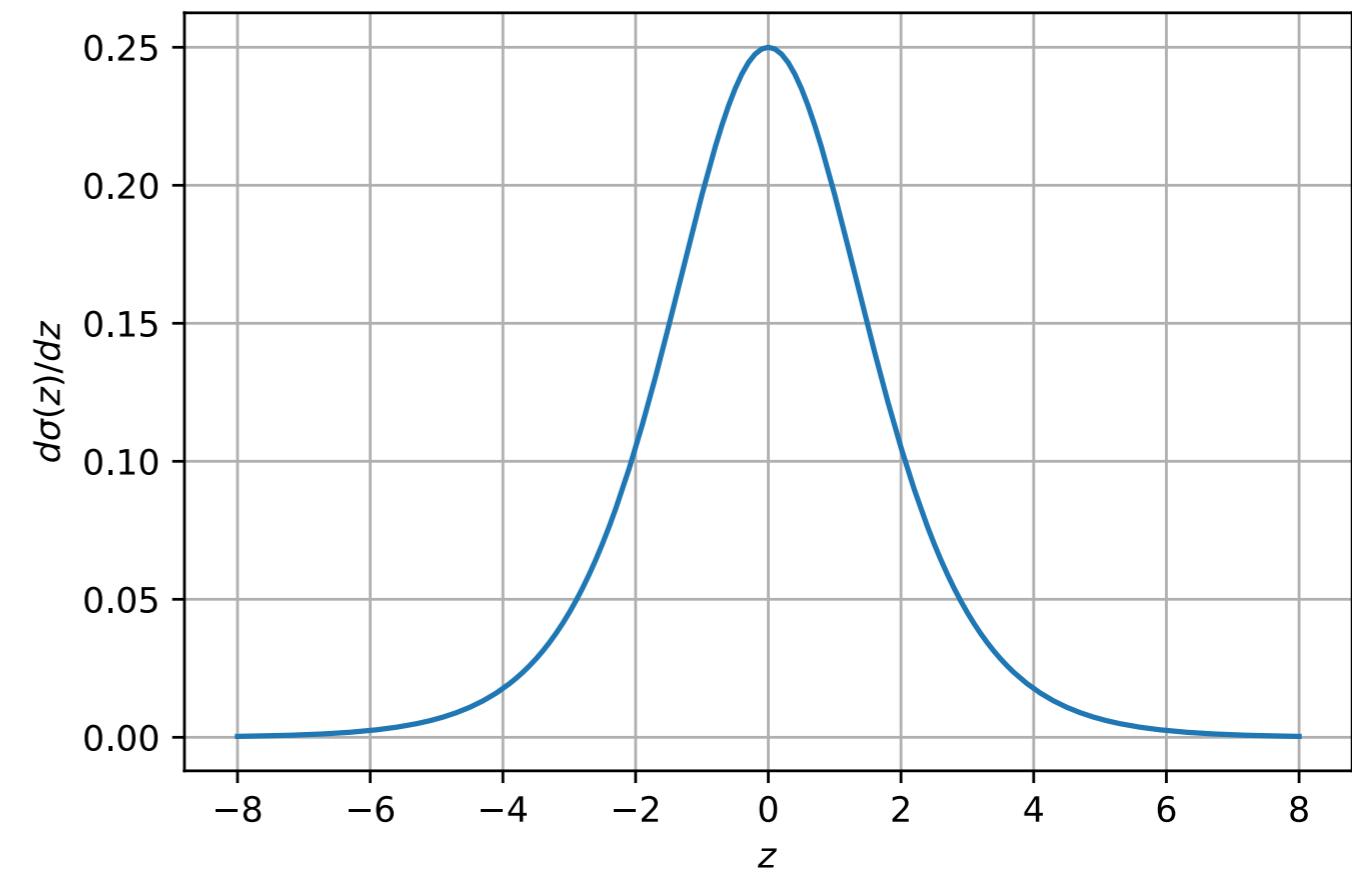
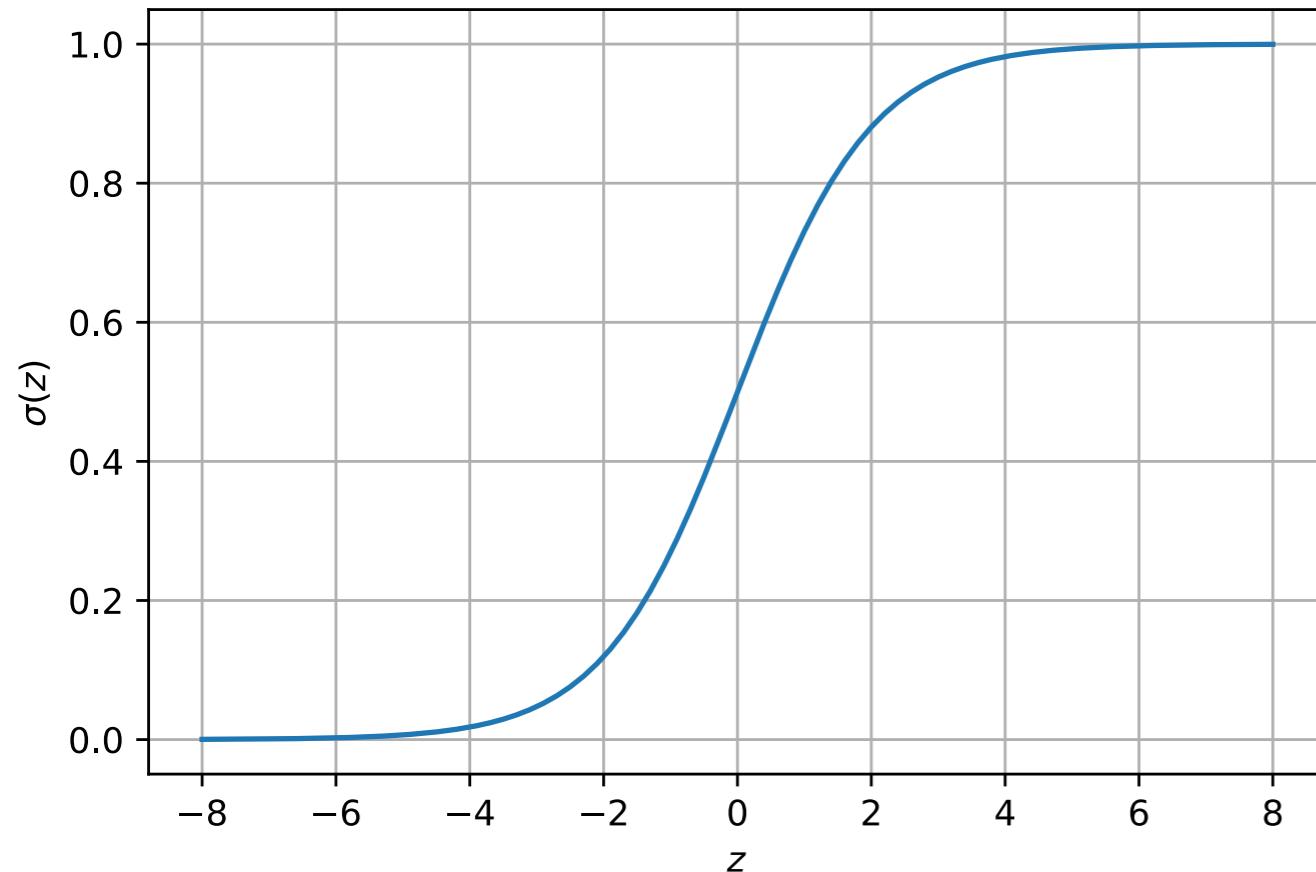
- However, the difference is that in Logistic Regression, we maximize the likelihood
- Maximizing likelihood is the same as maximizing the log-likelihood, but the latter is numerically more stable
- Maximizing the log-likelihood is the same as minimizing the negative log-likelihood, which is convenient, so we don't have to change our code and can still use gradient descent (instead of gradient ascent)

# Logistic Sigmoid Derivative

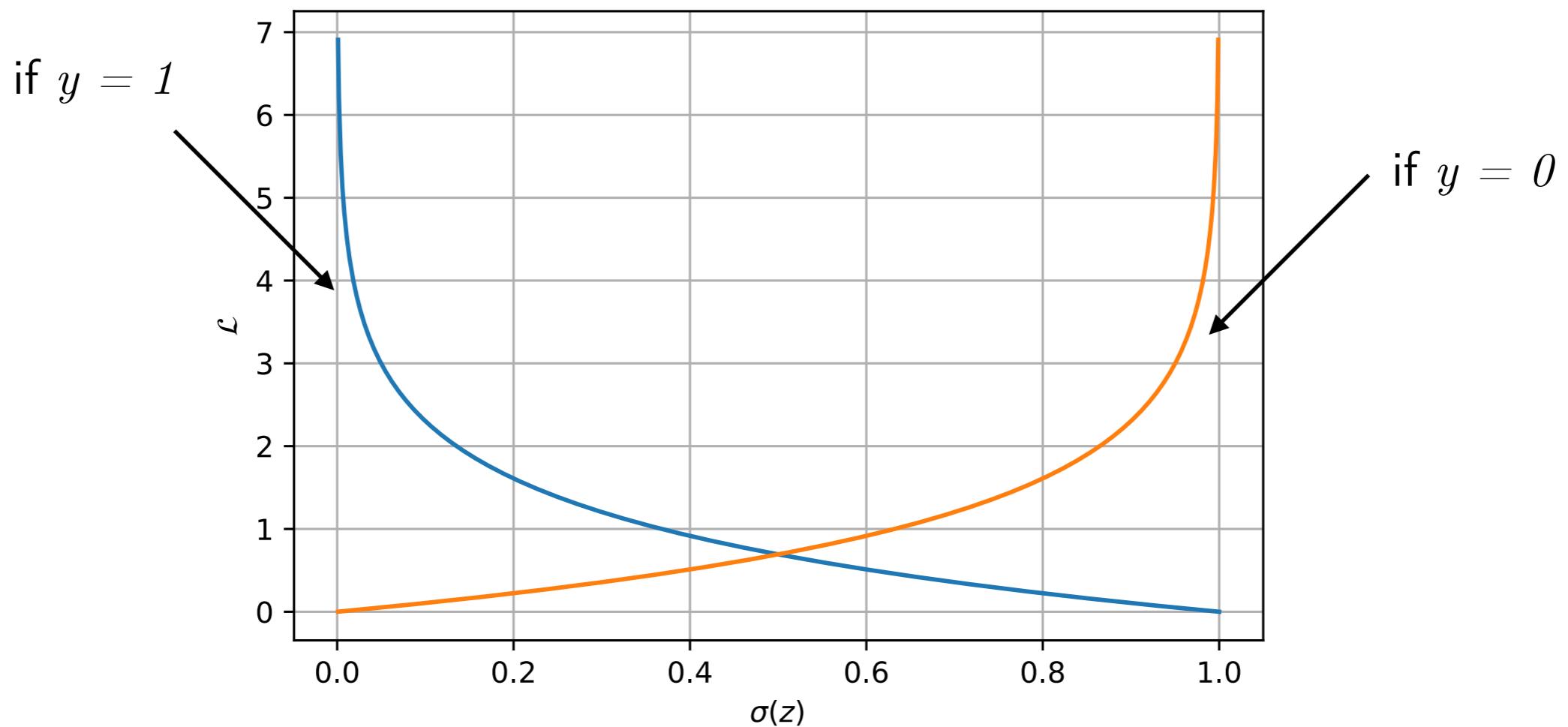
Generally, another nice property of the logistic sigmoid  
(in multi-layer nets) is that it has nice derivatives!

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\frac{d}{dz} \sigma(z) = \frac{e^{-z}}{(1 + e^{-z})^2} = \sigma(z)(1 - \sigma(z))$$



# Loss for a Single Training Example



$$\mathcal{L}(\mathbf{w}) = -y^{(i)} \log (\sigma(z^{(i)})) + (1 - y^{(i)}) \log (1 - \sigma(z^{(i)}))$$

# Learning Rule

Same gradient descent rule as before, for which we need

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{\partial \mathcal{L}}{\partial a} \frac{da}{dz} \frac{\partial z}{\partial w_j}$$

$$\frac{\partial \mathcal{L}}{\partial a} = \left( y \frac{1}{a} - (1-y) \frac{1}{1-a} \right)$$

$$\frac{da}{dz} = \frac{e^{-z}}{(1 + e^{-z})^2} = a \cdot (1 - a)$$

$$\frac{\partial z}{\partial w_j} = x_j$$

# Learning Rule

Same gradient descent rule as before, for which we need

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{\partial \mathcal{L}}{\partial a} \frac{da}{dz} \frac{\partial z}{\partial w_j}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial a} &= \left( y \frac{1}{a} - (1-y) \frac{1}{1-a} \right) \\ \frac{da}{dz} &= \frac{e^{-z}}{(1+e^{-z})^2} = a \cdot (1-a) \\ \frac{\partial z}{\partial w_j} &= x_j\end{aligned}\longrightarrow \frac{\partial \mathcal{L}}{\partial z} = y - a$$

# Learning Rule for Logistic Regression

Same gradient descent rule as for ADALINE & Linear Regression,  
for which we need

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{\partial \mathcal{L}}{\partial a} \frac{da}{dz} \frac{\partial z}{\partial w_j}$$

$$\frac{\partial \mathcal{L}}{\partial a} = - \left( y \frac{1}{a} - (1-y) \frac{1}{1-a} \right)$$

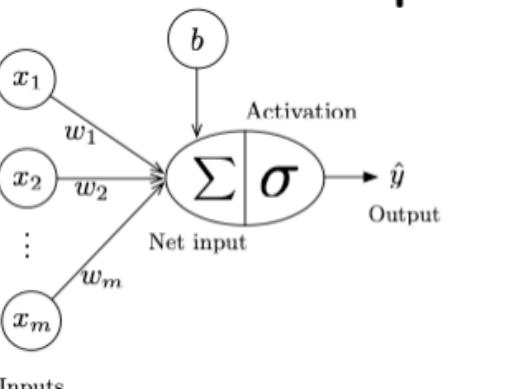
$$\frac{da}{dz} = \frac{e^{-z}}{(1+e^{-z})^2} = a \cdot (1-a) \longrightarrow \frac{\partial \mathcal{L}}{\partial z} = y - a \longrightarrow \frac{\partial \mathcal{L}}{\partial w_j} = (y - a)x_j$$

$$\frac{\partial z}{\partial w_j} = x_j$$

# Learning Rule for Logistic Regression

Remember the Perceptron & ADALINE Lectures?

**Perceptron Recap**


$$\sigma \left( \sum_{i=1}^m x_i w_i + b \right) = \sigma (\mathbf{x}^T \mathbf{w} + b) = \hat{y}$$
$$\sigma(z) = \begin{cases} 0, & z \leq 0 \\ 1, & z > 0 \end{cases}$$
$$b = -\theta$$

Inputs

Let  $\mathcal{D} = (\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, \langle \mathbf{x}^{[2]}, y^{[2]} \rangle, \dots, \langle \mathbf{x}^{[n]}, y^{[n]} \rangle) \in (\mathbb{R}^m \times \{0, 1\})^n$

1. Initialize  $\mathbf{w} := 0^{m-1}, \mathbf{b} := 0$
2. For every training epoch:
  - A. For every  $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$  :
    - (a)  $\hat{y}^{[i]} := \sigma(\mathbf{x}^{[i]T} \mathbf{w} + b)$  Compute output (prediction)
    - (b)  $\text{err} := (y^{[i]} - \hat{y}^{[i]})$  Calculate error
    - (c)  $\mathbf{w} := \mathbf{w} + \text{err} \times \mathbf{x}^{[i]}, b := b + \text{err}$  Update parameters

Sebastian Raschka      STAT 479: Deep Learning      SS 2019

$$\frac{\partial \mathcal{L}}{\partial w_j} = (y - a)x_j$$

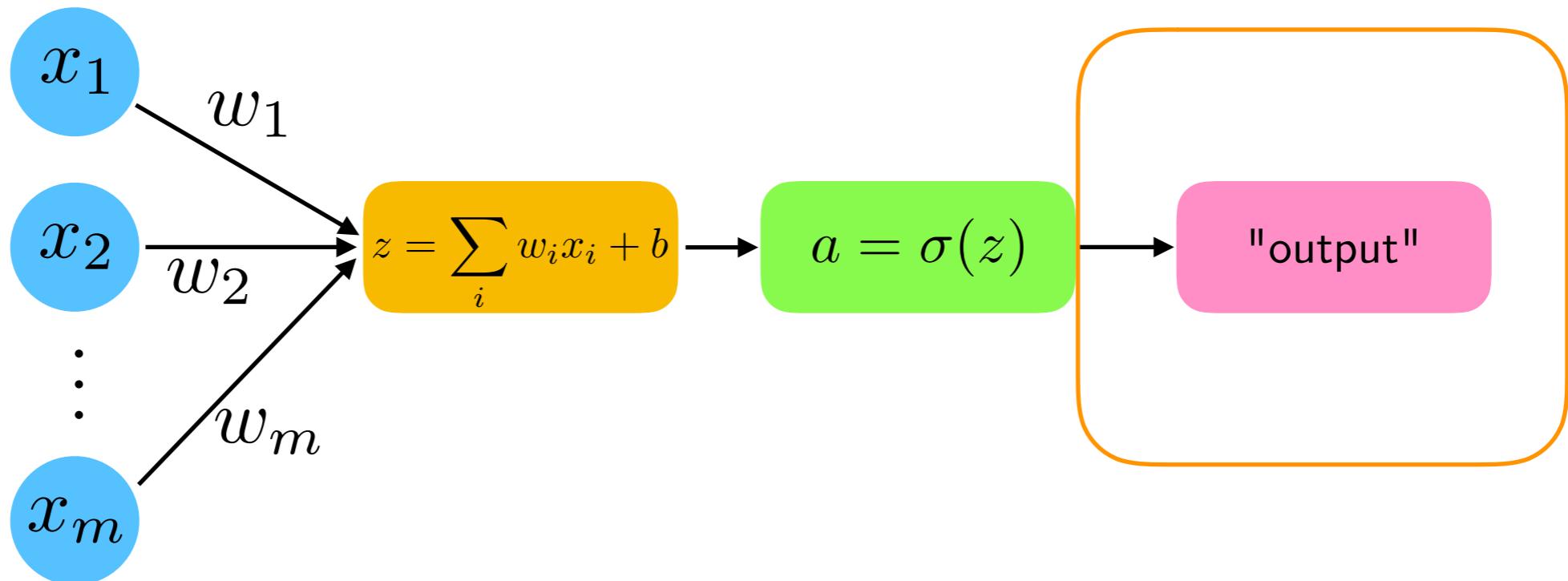
$\Delta w$  := "negative gradient"

$$= -\frac{\partial \mathcal{L}}{\partial w_j}$$

$$w := w + \eta \cdot \Delta w$$

Update for each batch/minibatch  
with learning rate

# Class Label Predictions with Logistic Regression



In logistic regression, we can use

$$\hat{y} := \begin{cases} 1 & \text{if } \sigma(z) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

which is the same as

$$\hat{y} := \begin{cases} 1 & \text{if } z > 0.0 \\ 0 & \text{otherwise} \end{cases}$$

- We can think of this part as a "separate" part that converts the neural network values into a class label, for example; e.g., via a threshold function
- Predicted class labels are not used during training (except by the Perceptron)
- ADALINE, Logistic Regression, and all common types of multi-layer neural networks don't use predicted class labels for optimization as a threshold function is not smooth

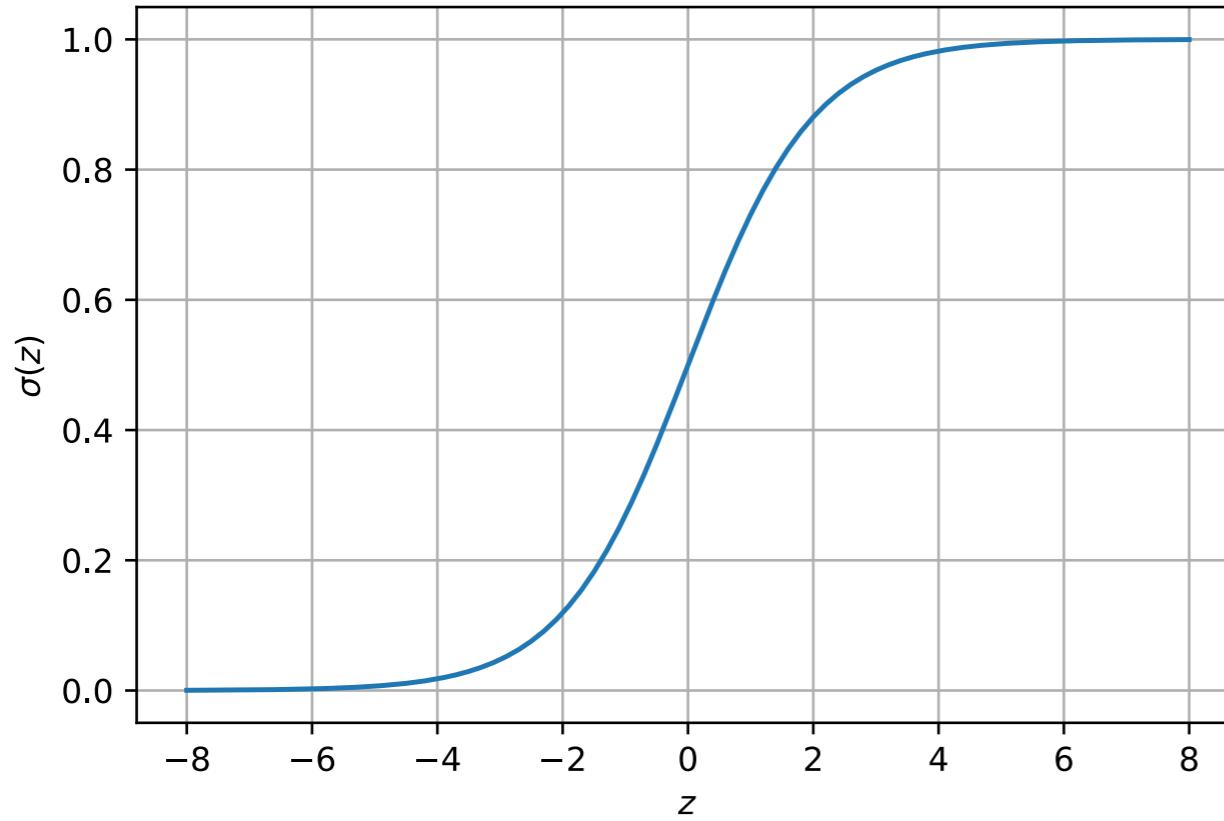
# About the Term "Logits"

"Logits" is a very commonly used Deep Learning jargon;  
probably inspired by the logits for logistic regression

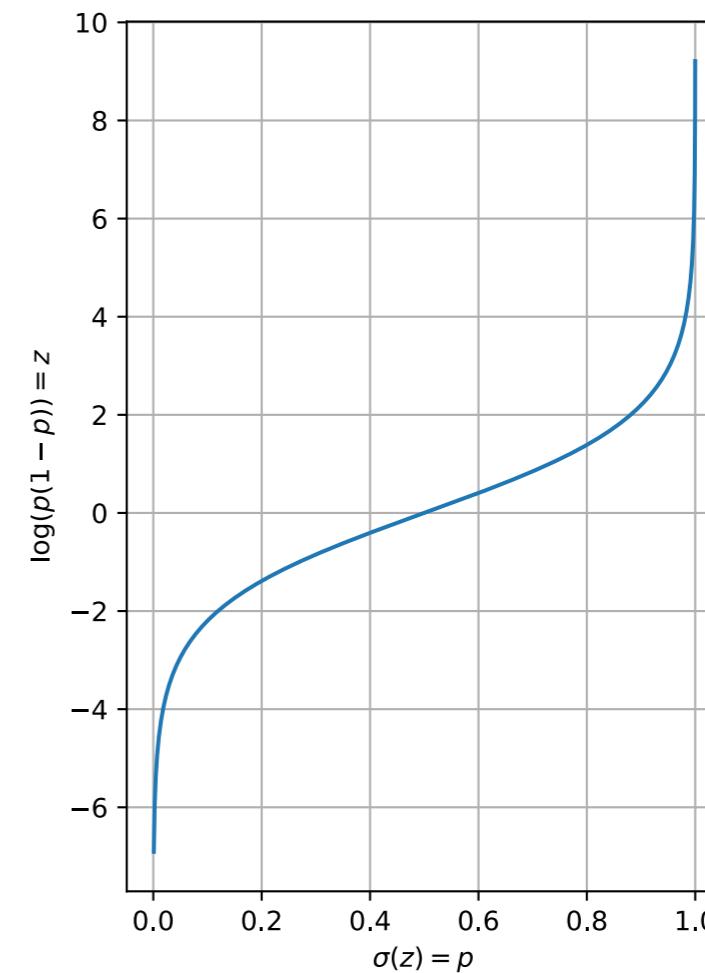
- In deep learning, the logits are the net inputs of the last neuron layer
- In statistics, we call the log-odds the logits
- In logistic regression, the logits are naturally  $\mathbf{w}^\top \mathbf{x}$  ...
- ... because log-odds is just short for "logarithm of the odds":  $\log(p/(1-p))$
- In other words, the logits are the inverse of the logistic sigmoid function

# About the Term "Logits"

"Logits" is a very commonly used Deep Learning jargon;  
probably inspired by the logits for logistic regression



$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



$$\text{logit}(\sigma(z)) = \log(\sigma(z)/(1 - \sigma(z))) = z$$

# About the Term "Binary Cross Entropy"

- Negative log-likelihood and binary cross entropy are equivalent
- They are just formulated in different contexts
- Cross entropy comes from the "information theory" (computer science) perspective
- For those who took my previous STAT479: Machine Learning class, we have seen regular entropy in the context of decision trees (but with  $\log_2$  instead of the natural log)

[https://github.com/rasbt/stat479-machine-learning-fs18/blob/master/06\\_trees/06\\_trees\\_notes.pdf](https://github.com/rasbt/stat479-machine-learning-fs18/blob/master/06_trees/06_trees_notes.pdf)

$$H_{\mathbf{a}}(\mathbf{y}) = - \sum_i \left( a^{[i]} \log(y^{[i]}) + (1 - a^{[i]}) \log(1 - y^{[i]}) \right) \quad \text{Binary Cross Entropy}$$

$$H_{\mathbf{a}}(\mathbf{y}) = \sum_{i=1}^n \sum_{k=1}^K -y_k^{[i]} \log \left( a_k^{[i]} \right) \quad \begin{array}{l} \text{(Multi-category) Cross Entropy} \\ \text{for K different class labels} \end{array}$$

# Logistic Regression Coding Example

[https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/  
L08\\_logistic/code/logistic-regression.ipynb](https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L08_logistic/code/logistic-regression.ipynb)

# Logistic Regression for Multi-class Classification:

Multinomial Logistic Regression/  
Softmax Regression

# Logistic Regression

Cross entropy

$$E(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n \log \left( 1 + \exp(-y^{[i]} \cdot \mathbf{w}^T \mathbf{x}) \right)$$

... Slides will be added before next lecture

Softmax activation: convert real values into "probabilities"

$$\sigma(z_i) = \frac{\exp(z_i)}{\sum_j^k \exp(z_j)}$$

Cross entropy code cross-entropy-pytorch.ipynb

## Softmax code

# Reading Assignments