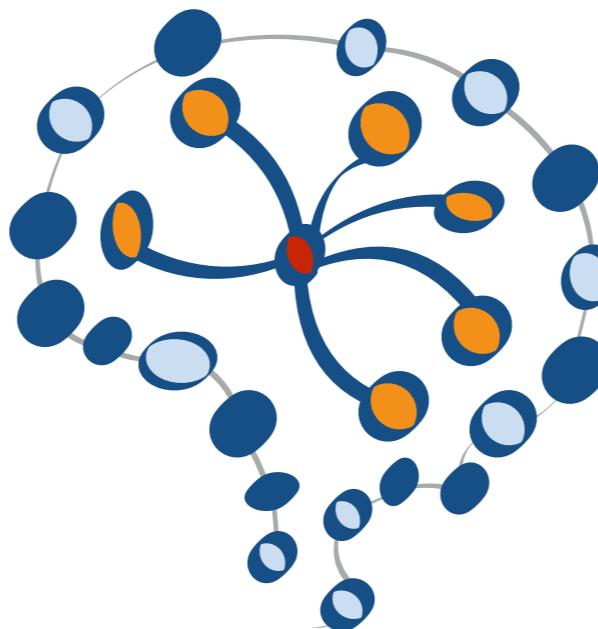


STAT 453: Introduction to Deep Learning and Generative Models

Sebastian Raschka

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

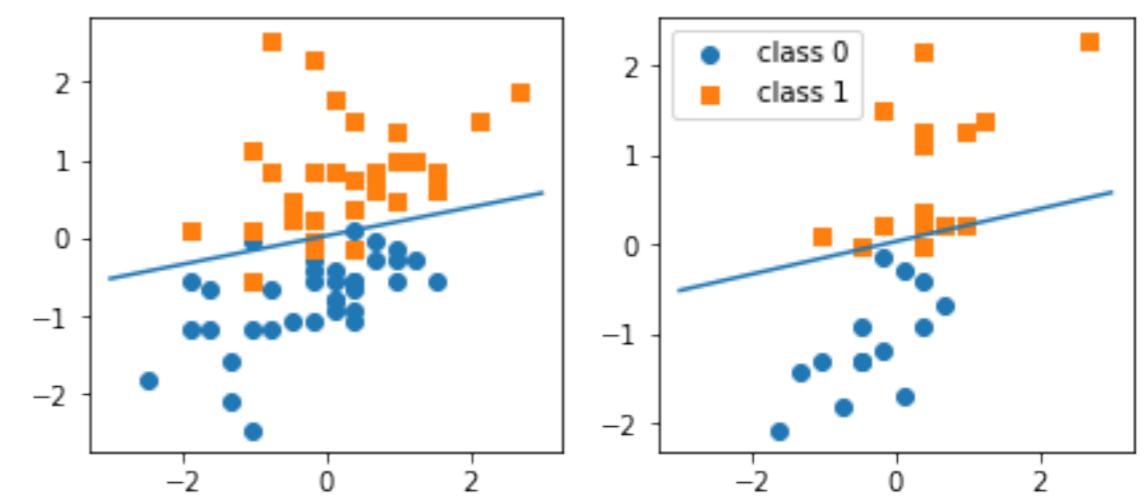
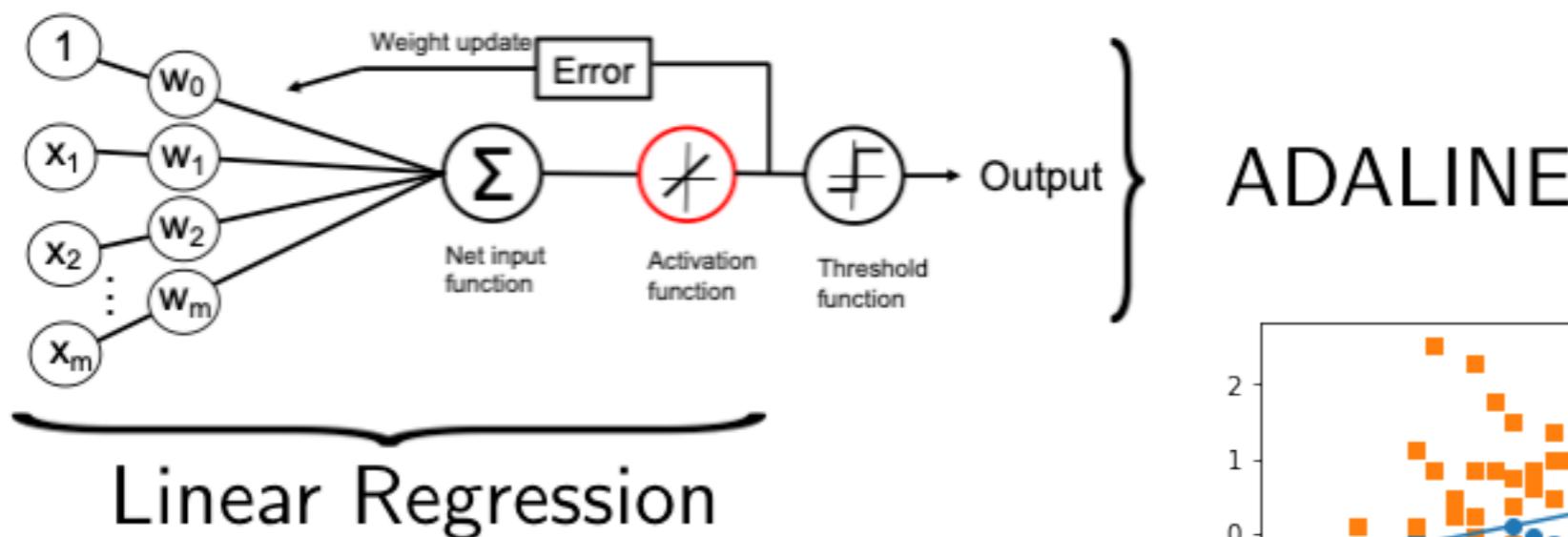


Lecture 08

Logistic Regression and Multi-class classification

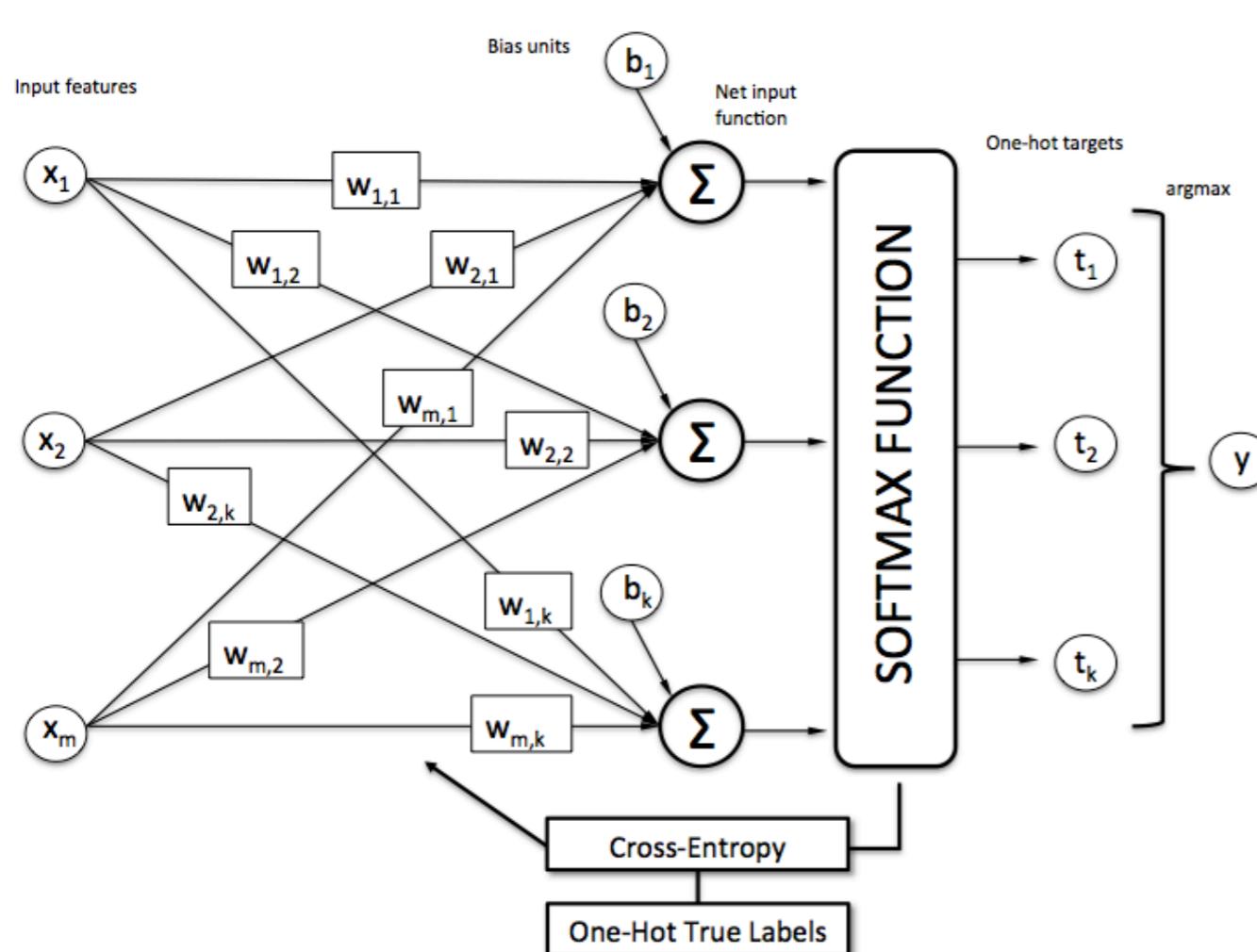
Previous Lecture(s):

- Solved convergence issue of Perceptron via Adaline
- Conceptualized gradient descent via computation graphs

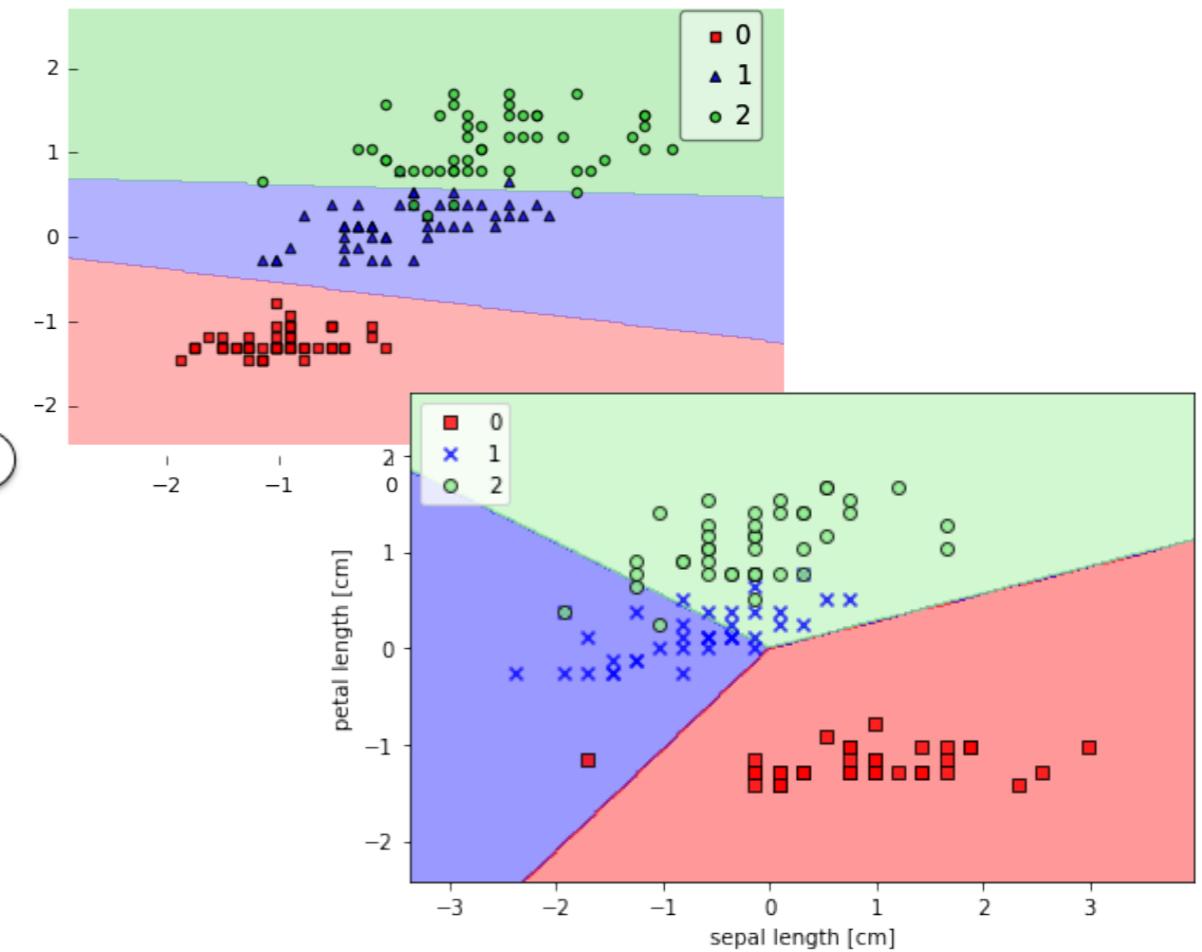


This Lecture:

- A better loss function for classification (cross entropy instead of MSE)
- Extending neurons to multi-class classification (multiple output nodes + softmax)



Softmax Regression - Gradient Descent



Topics

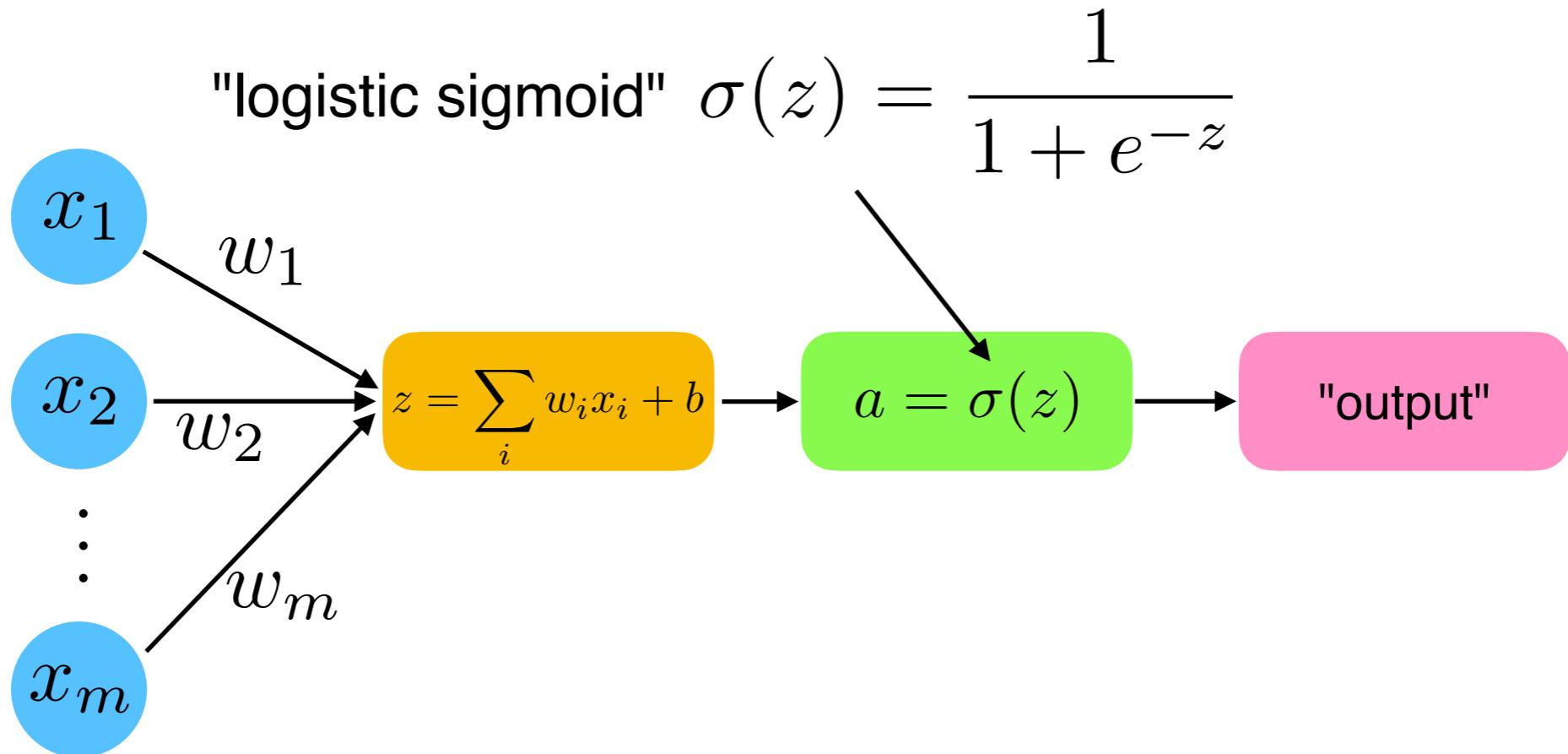
1. Logistic Regression as an Artificial Neuron
2. Negative Log-Likelihood Loss
3. Logistic Regression Learning Rule
4. Logits and Cross Entropy
5. Logistic Regression Code Example
6. Generalizing to Multiple Classes: Softmax Regression
7. OneHot Encoding and Multi-category Cross Entropy
8. Softmax Regression Learning Rule
9. Softmax Regression Code Example

Viewing Logistic Regression as a Single Layer Neural Network

- 1. Logistic Regression as an Artificial Neuron**
2. Negative Log-Likelihood Loss
3. Logistic Regression Learning Rule
4. Logits and Cross Entropy
5. Logistic Regression Code Example
6. Generalizing to Multiple Classes: Softmax Regression
7. OneHot Encoding and Multi-category Cross Entropy
8. Softmax Regression Learning Rule
9. Softmax Regression Code Example

Logistic Regression Neuron

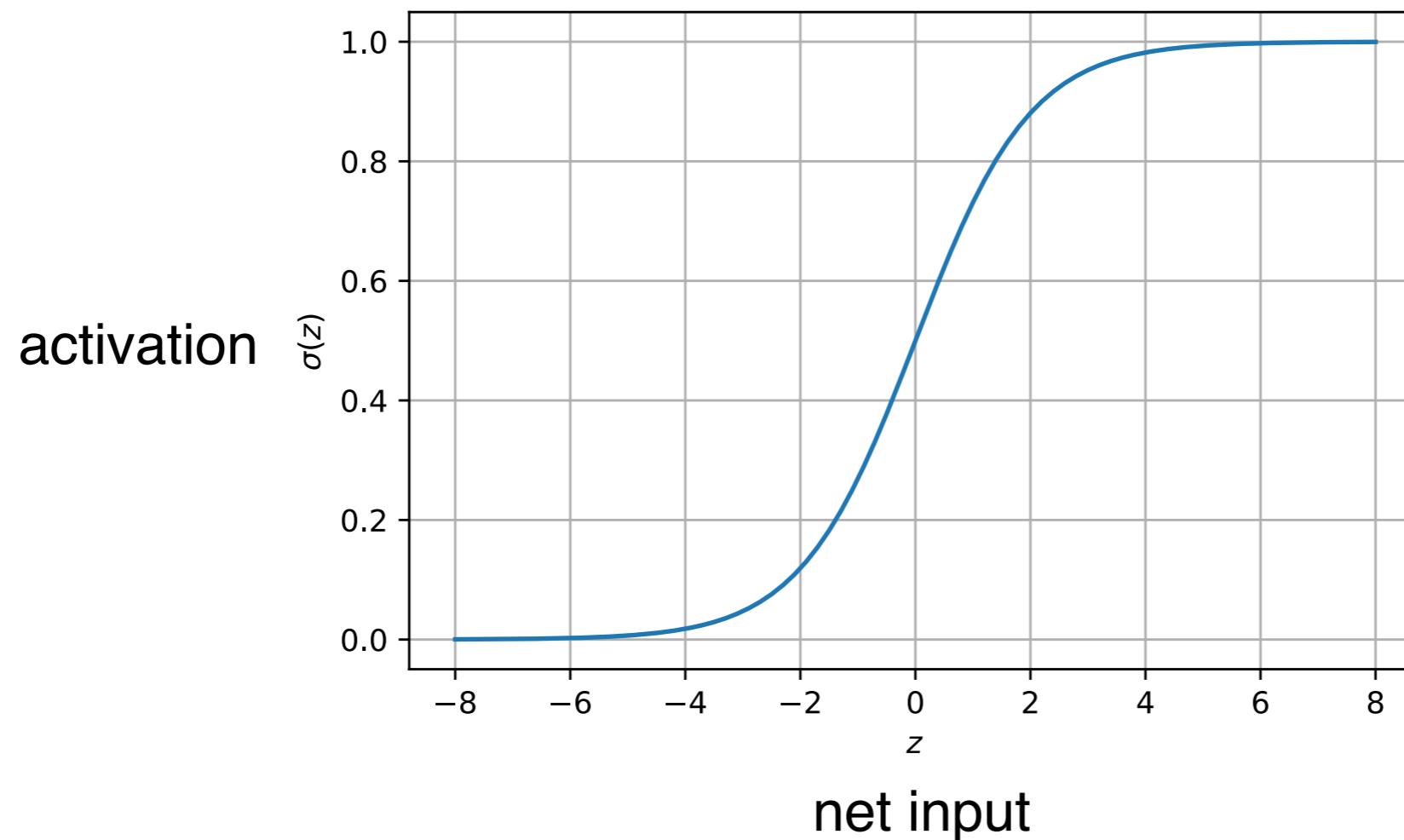
For binary classes $y \in \{0, 1\}$



- In ADALINE, the activation function was an identity function, $\sigma(z) = z$
- In ADALINE we used MSE as loss function: $MSE = \frac{1}{n} \sum_i (a^{[i]} - y^{[i]})^2$
- We will use a different loss function for logistic regression

Logistic Sigmoid Function

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



Logistic Regression

Given the output:

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

here, the "hypothesis" is just our activation function output

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

We compute the posterior as:

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

For a binary class problem (0 and 1), we want these probabilities to be:

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

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

Logistic Regression's Loss Function

1. Logistic Regression as an Artificial Neuron
- 2. Negative Log-Likelihood Loss**
3. Logistic Regression Learning Rule
4. Logits and Cross Entropy
5. Logistic Regression Code Example
6. Generalizing to Multiple Classes: Softmax Regression
7. OneHot Encoding and Multi-category Cross Entropy
8. Softmax Regression Learning Rule
9. Softmax Regression Code Example

Logistic Regression

Given the output:

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

here, the "hypothesis" is just our activation function output

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

We compute the posterior as:

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

For a binary class problem (0 and 1), we want these probabilities to be:

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

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

Logistic Regression

Given the output:

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

here, the "hypothesis" is just our activation function output

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

We compute the posterior as:

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

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

rewrite more compactly

Logistic Regression

And for multiple training examples, we want to maximize:

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

by finding *good/optimal* parameters

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, 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)

Training a Logistic Regression Model

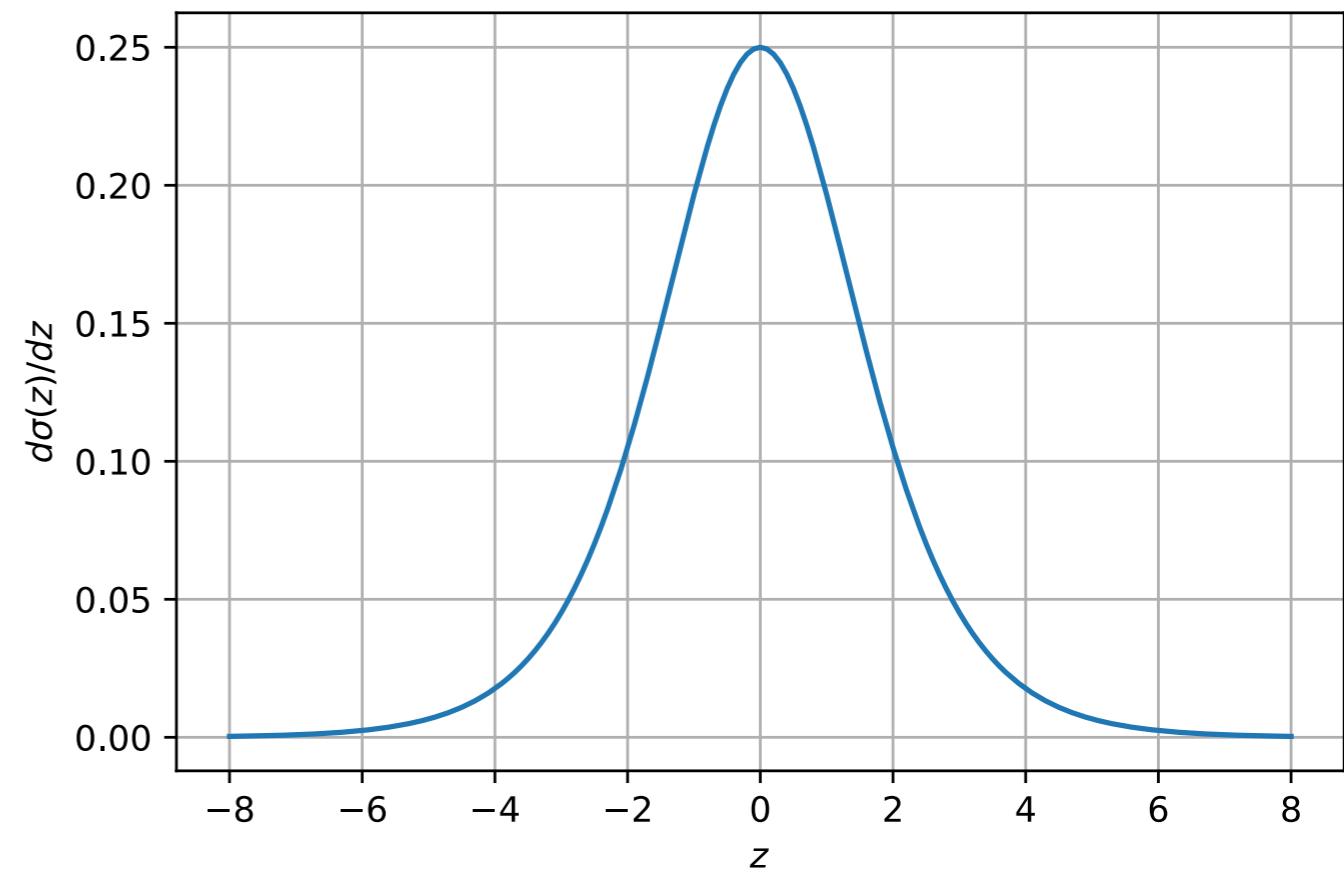
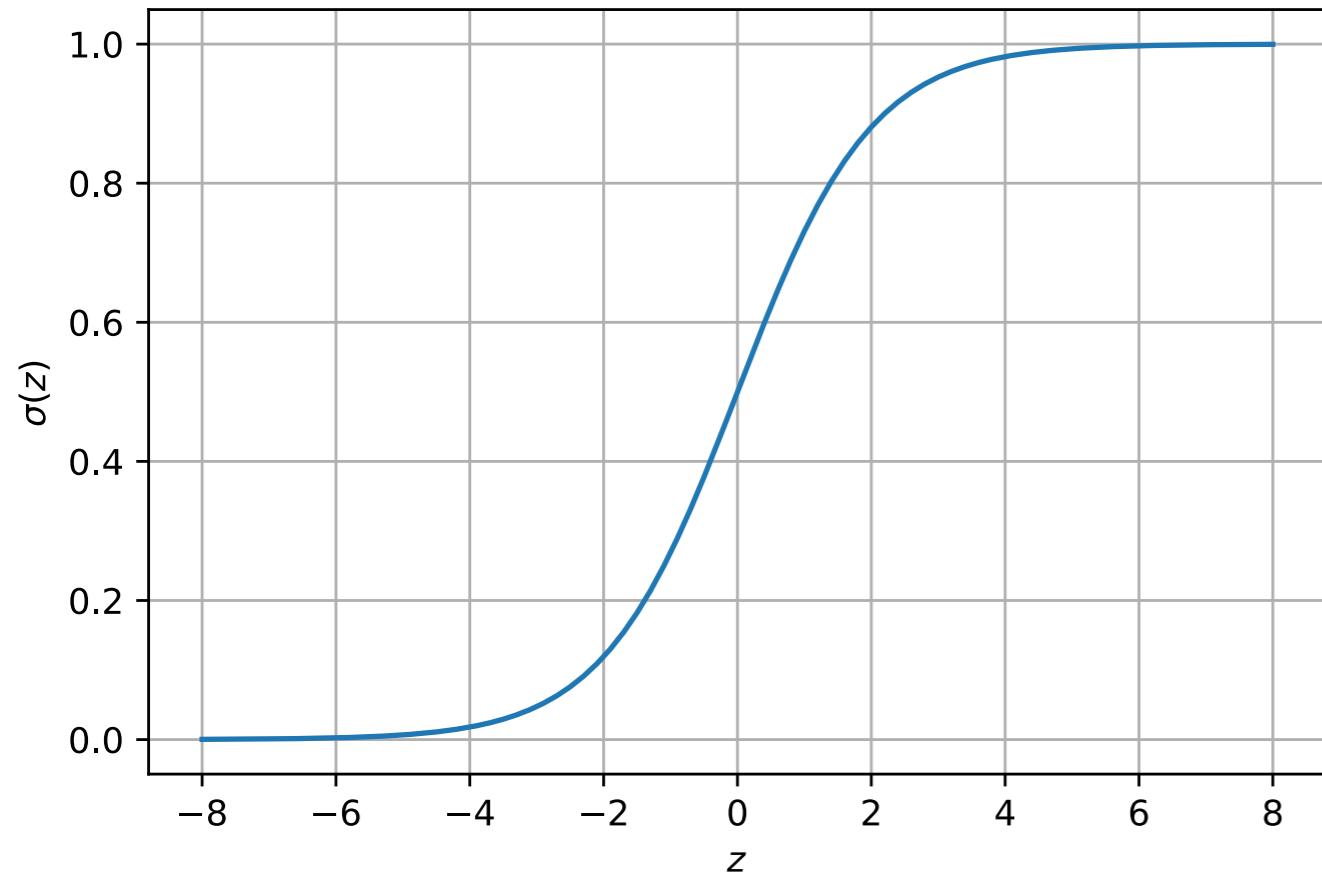
1. Logistic Regression as an Artificial Neuron
2. Negative Log-Likelihood Loss
- 3. Logistic Regression Learning Rule**
4. Logits and Cross Entropy
5. Logistic Regression Code Example
6. Generalizing to Multiple Classes: Softmax Regression
7. OneHot Encoding and Multi-category Cross Entropy
8. Softmax Regression Learning Rule
9. Softmax Regression Code Example

Logistic Sigmoid Derivative

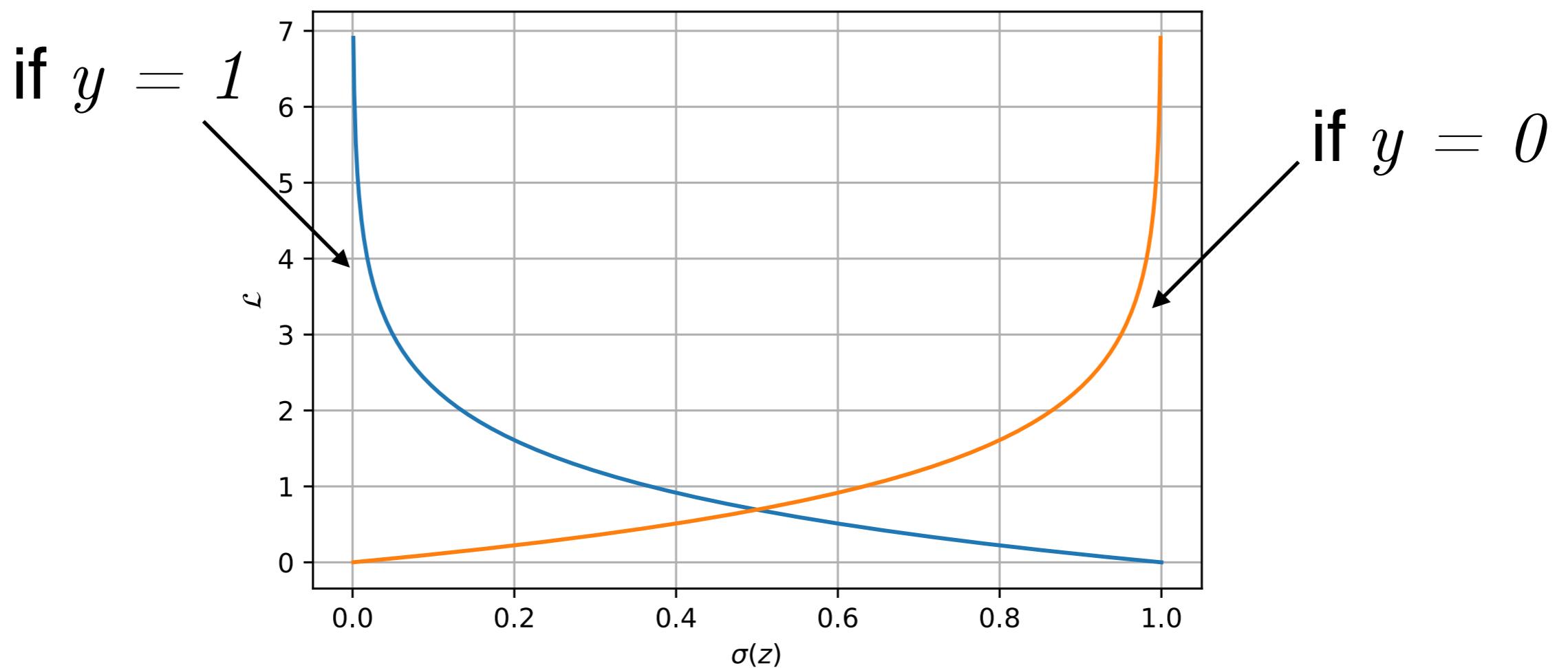
Generally, another nice property of the logistic sigmoid (in multi-layer nets) is that it has nice derivatives!

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

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



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

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

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

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

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

Learning Rule

Same gradient descent rule as before, for which we need

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

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

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

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

$$\frac{\partial \mathcal{L}}{\partial z} = a - y$$



Learning Rule for Logistic Regression

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

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

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial a} &= \frac{a - y}{a - a^2} \\ \frac{da}{dz} &= \frac{e^{-z}}{(1 + e^{-z})^2} = a \cdot (1 - a) \\ \frac{\partial z}{\partial w_j} &= x_j\end{aligned}\quad \begin{array}{c}\xrightarrow{\hspace{1cm}} \\ \xrightarrow{\hspace{1cm}} \\ \xrightarrow{\hspace{1cm}}\end{array}\quad \begin{aligned}\frac{\partial \mathcal{L}}{\partial z} &= a - y \\ \frac{\partial \mathcal{L}}{\partial w_j} &= (a - y)x_j\end{aligned}$$

Learning Rule for Logistic Regression

Remember the Linear Regression & ADALINE Lectures?

Stochastic gradient descent

1. Initialize $\mathbf{w} := \mathbf{0} \in \mathbb{R}^m$, $b := 0$

2. For every training epoch:

A. For every $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$

$$(a) \hat{y}^{[i]} := \sigma(\mathbf{x}^{[i]T} \mathbf{w} + b)$$

$$(b) \nabla_{\mathbf{w}} \mathcal{L} = -(y^{[i]} - \hat{y}^{[i]}) \mathbf{x}^{[i]}$$

$$\nabla_b \mathcal{L} = -(y^{[i]} - \hat{y}^{[i]})$$

$$(c) \mathbf{w} := \mathbf{w} + \eta \times (-\nabla_{\mathbf{w}} \mathcal{L})$$

$$b := b + \eta \times (-\nabla_b \mathcal{L})$$

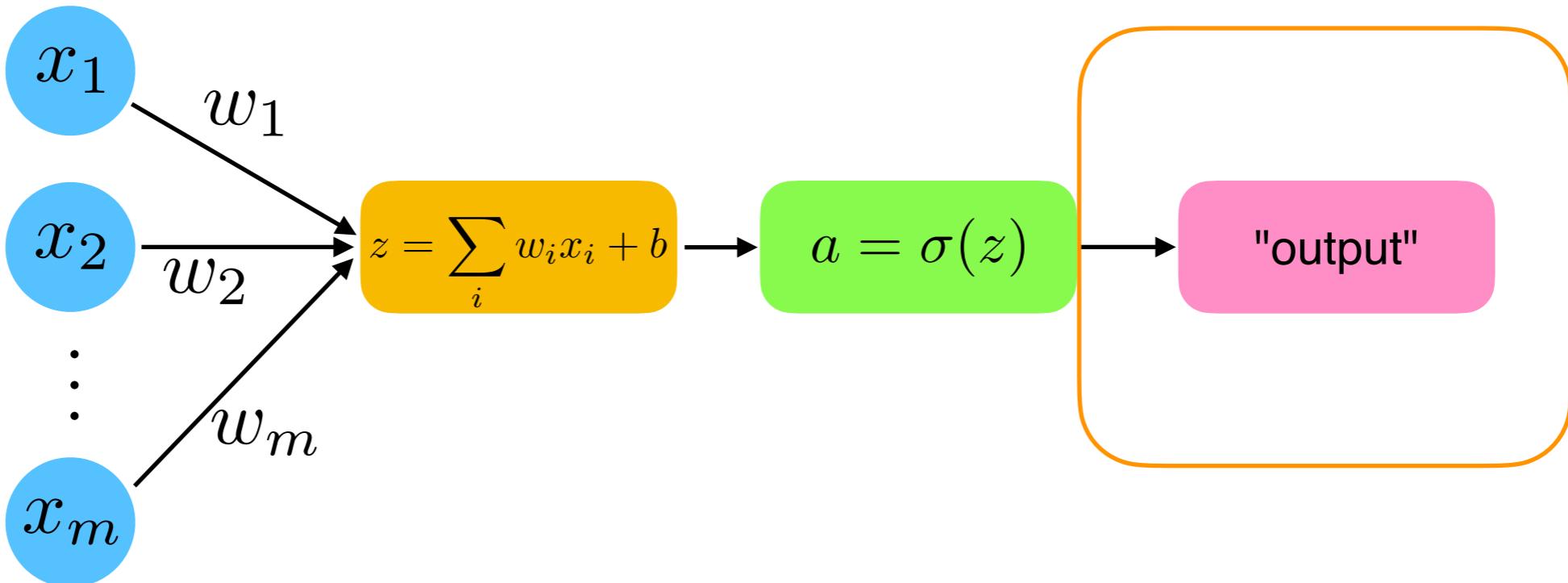
learning rate

Note

$$a - y \Leftrightarrow -(y^{[i]} - \hat{y}^{[i]})$$

negative gradient

Class Label Predictions with Logistic Regression



In logistic regression, we can use

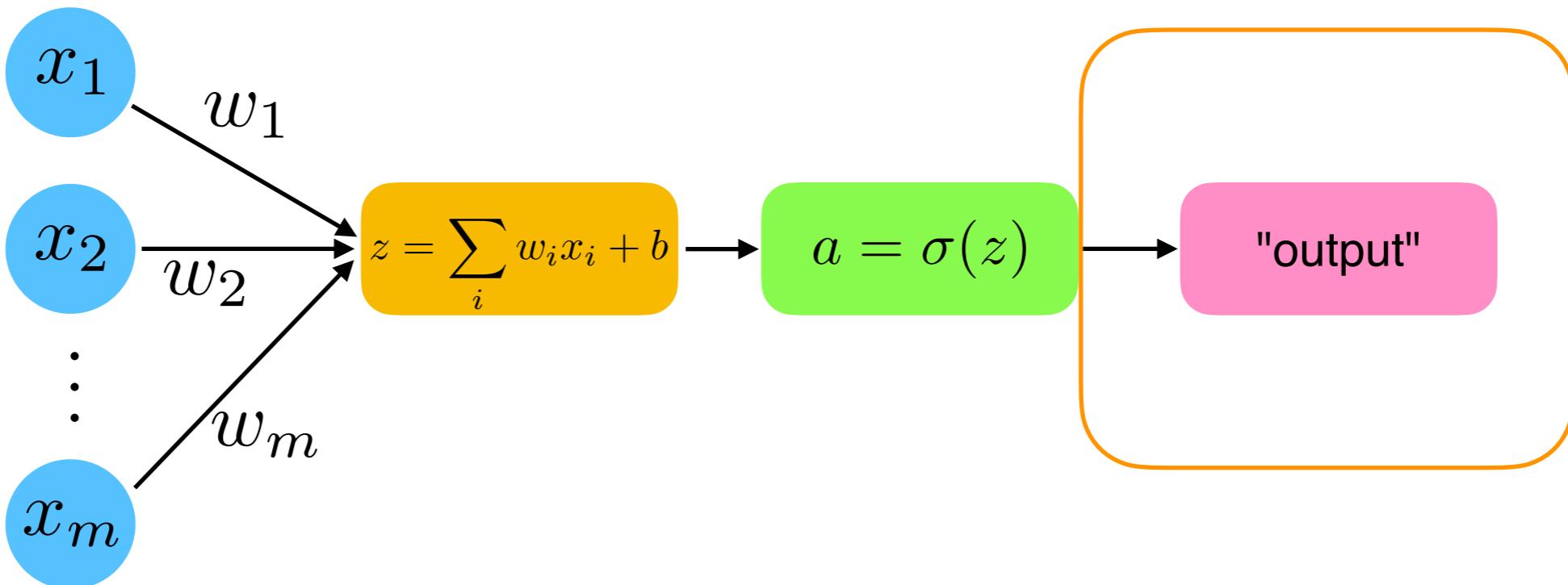
$$\hat{y} := \begin{cases} 1 & \text{if } \sigma(z) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

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

Class Label Predictions with Logistic Regression

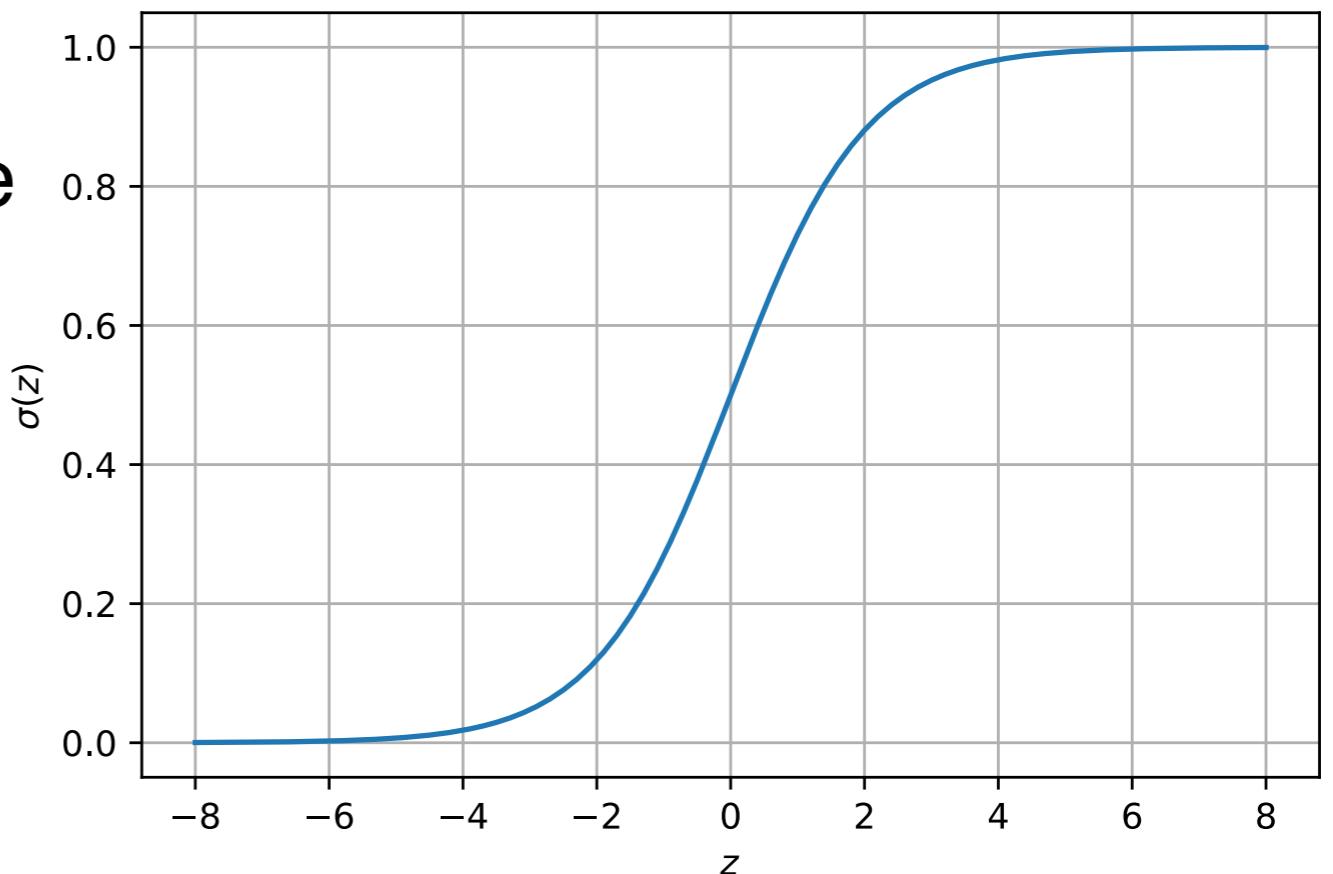


In logistic regression, we can use

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

which is the same as

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



Logits and Cross Entropy

1. Logistic Regression as an Artificial Neuron
2. Negative Log-Likelihood Loss
3. Logistic Regression Learning Rule
- 4. Logits and Cross Entropy**
5. Logistic Regression Code Example
6. Generalizing to Multiple Classes: Softmax Regression
7. OneHot Encoding and Multi-category Cross Entropy
8. Softmax Regression Learning Rule
9. Softmax Regression Code Example

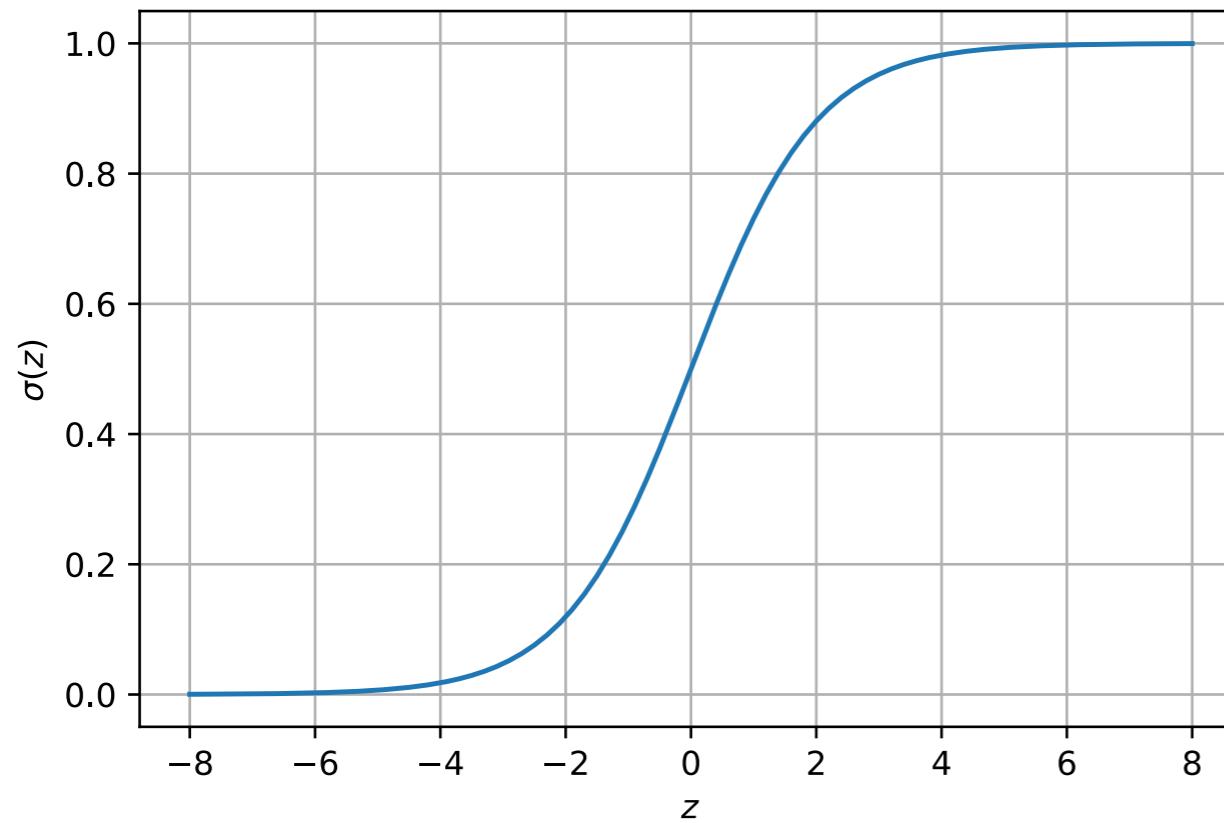
About the Term "Logits"

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

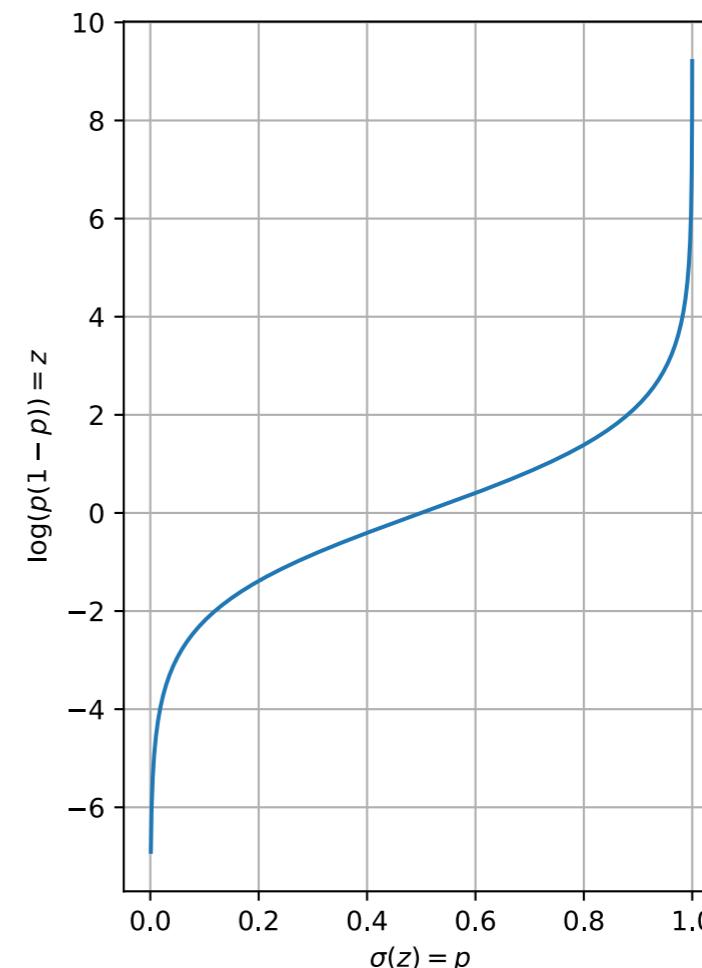
- In deep learning, the logits are the net inputs of the last neuron layer
- In statistics, we call the log-odds the logits
- In logistic regression, the logits are naturally $\mathbf{w}^\top \mathbf{x}$...
... because log-odds is just short for "logarithm of the odds": $\log(p/(1-p))$
- In other words, 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 STAT451: Machine Learning class, we have seen entropy (not-cross entropy though) in the context of decision trees (but with \log_2 instead of the natural log)

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

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

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 STAT451: Machine Learning class, we have seen entropy (not-cross entropy though) in the context of decision trees (but with \log_2 instead of the natural log)

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

This assumes one-hot encoding where the y 's are either 0 or 1

$$H_a(\mathbf{y}) = \sum_{i=1}^n \sum_{k=1}^K -y_k^{[i]} \log(a_k^{[i]}) \quad \begin{aligned} & \text{(Multi-category) Cross Entropy} \\ & \text{for } K \text{ different class labels} \end{aligned}$$


Logistic Regression Code Example

Logits and Cross Entropy

1. Logistic Regression as an Artificial Neuron
2. Negative Log-Likelihood Loss
3. Logistic Regression Learning Rule
4. Logits and Cross Entropy
- 5. Logistic Regression Code Example**
6. Generalizing to Multiple Classes: Softmax Regression
7. OneHot Encoding and Multi-category Cross Entropy
8. Softmax Regression Learning Rule
9. Softmax Regression Code Example

Logistic Regression Coding Example

[https://github.com/rasbt/stat453-deep-learning-ss21/blob/master/L08/code/
logistic-regression.ipynb](https://github.com/rasbt/stat453-deep-learning-ss21/blob/master/L08/code/logistic-regression.ipynb)

Multinomial Logistic Regression / Softmax Regression

1. Logistic Regression as an Artificial Neuron
2. Negative Log-Likelihood Loss
3. Logistic Regression Learning Rule
4. Logits and Cross Entropy
5. Logistic Regression Code Example
- 6. Generalizing to Multiple Classes: Softmax Regression**
7. OneHot Encoding and Multi-category Cross Entropy
8. Softmax Regression Learning Rule
9. Softmax Regression Code Example

MNIST - 60k Handwritten Digits

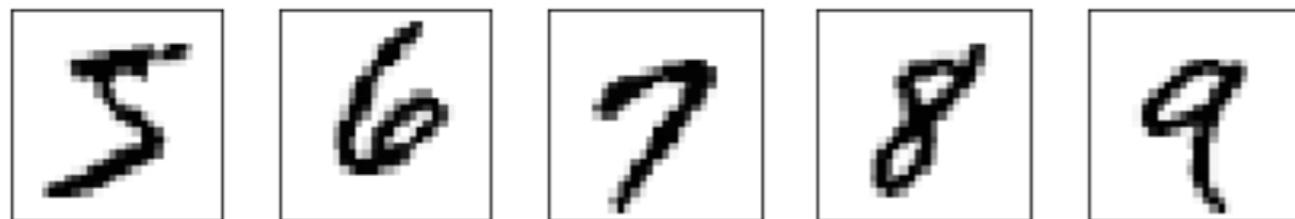
<http://yann.lecun.com/exdb/mnist/>



Balanced dataset:

- 10 classes (digits 0-9)
- 6k digits per class

Image dimensions: 28x28x1



In NCHW, an image batch of 128 examples would be a tensor with dimensions (128, 1, 28, 28)

- **Training set images:** train-images-idx3-ubyte.gz (9.9 MB, 47 MB unzipped, and 60,000 examples)
- **Training set labels:** train-labels-idx1-ubyte.gz (29 KB, 60 KB unzipped, and 60,000 labels)
- **Test set images:** t10k-images-idx3-ubyte.gz (1.6 MB, 7.8 MB, unzipped and 10,000 examples)
- **Test set labels:** t10k-labels-idx1-ubyte.gz (5 KB, 10 KB unzipped, and 10,000 labels)

MNIST - 60k Handwritten Digits

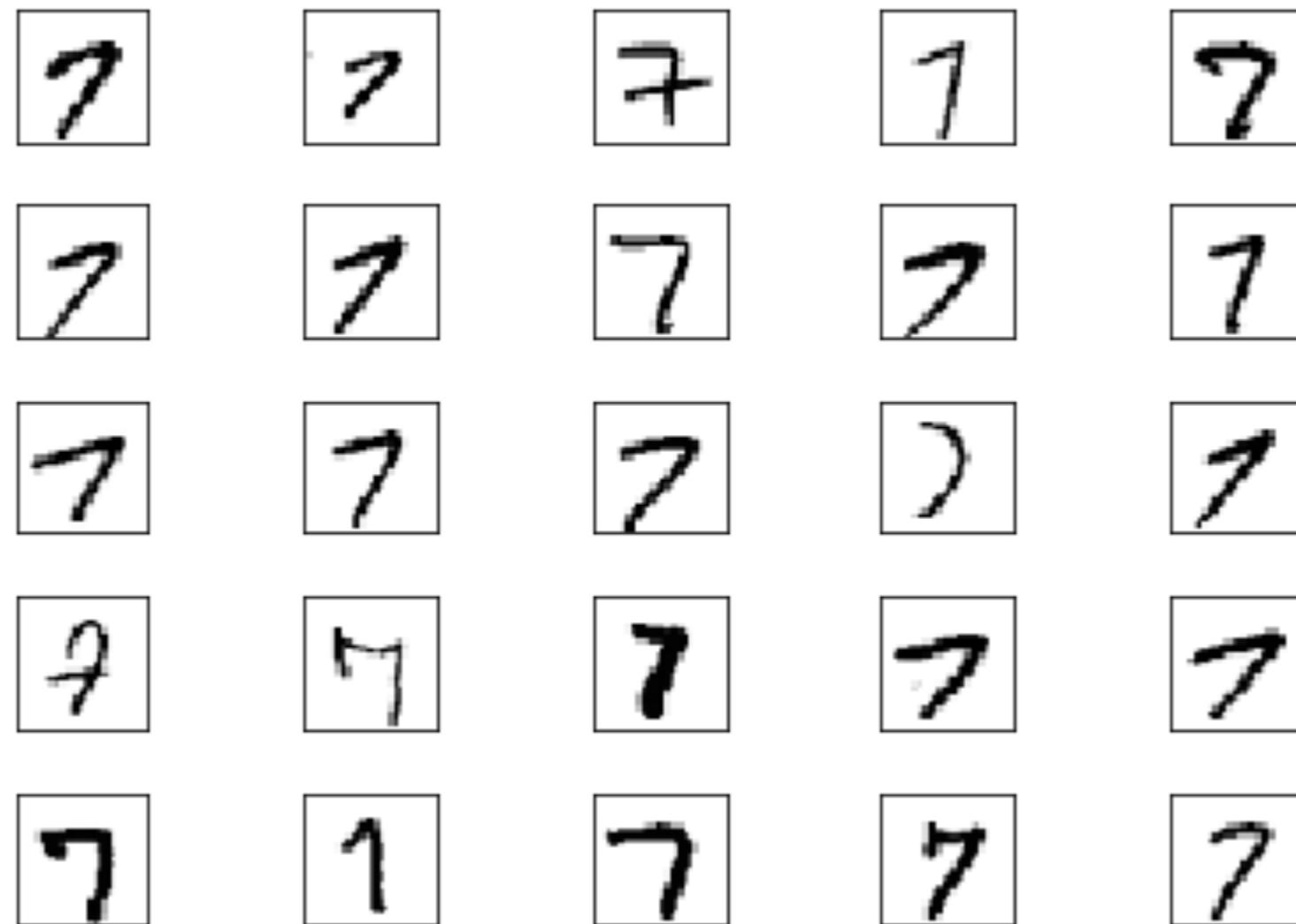


Illustration of different "7"s

Data Representation (unstructured data; images)

Convolutional Neural Networks (later)

Image batch dimensions: torch.Size([128, 1, 28, 28]) ← "NCHW" representation

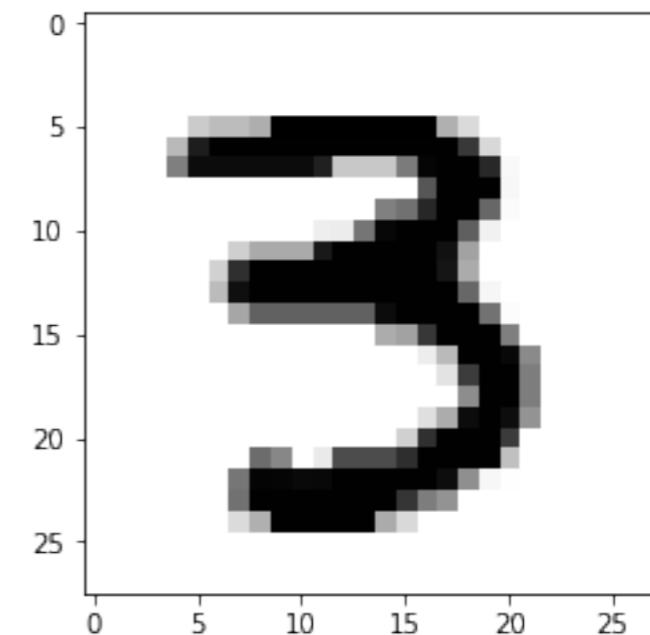
Image label dimensions: torch.Size([128])

```
print(images[0].size())
```

```
torch.Size([1, 28, 28])
```

```
images[0]
```

```
tensor([[[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
       [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000],
       [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000],
       [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000],
       [0.0000, 0.0000, 0.0000, 0.0000, 0.5020, 0.9529, 0.9529, 0.9529,
        0.9529, 0.9529, 0.9529, 0.8706, 0.2157, 0.2157, 0.2157, 0.5176,
        0.9804, 0.9922, 0.9922, 0.8392, 0.0235, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000],
       [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.6627, 0.9922, 0.9922, 0.9922, 0.0314, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000],
       [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.8471, 0.9922, 0.9922, 0.9922, 0.5961, 0.0157, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000],
       [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0667, 0.0745, 0.5412, 0.9725, 0.9922,
        0.9922, 0.9922, 0.6275, 0.0548, 0.0000, 0.0000, 0.0000, 0.0000,
```

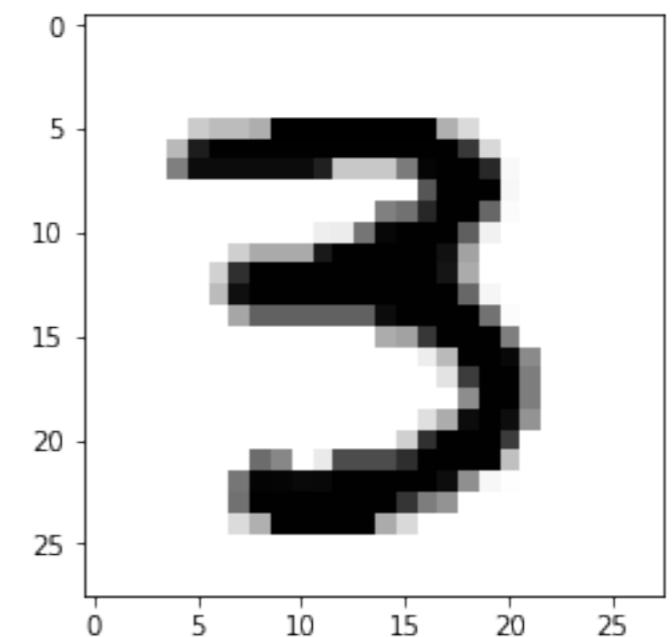


Note that I normalized pixels by factor 1/255 here

Data Representation (unstructured data; images)

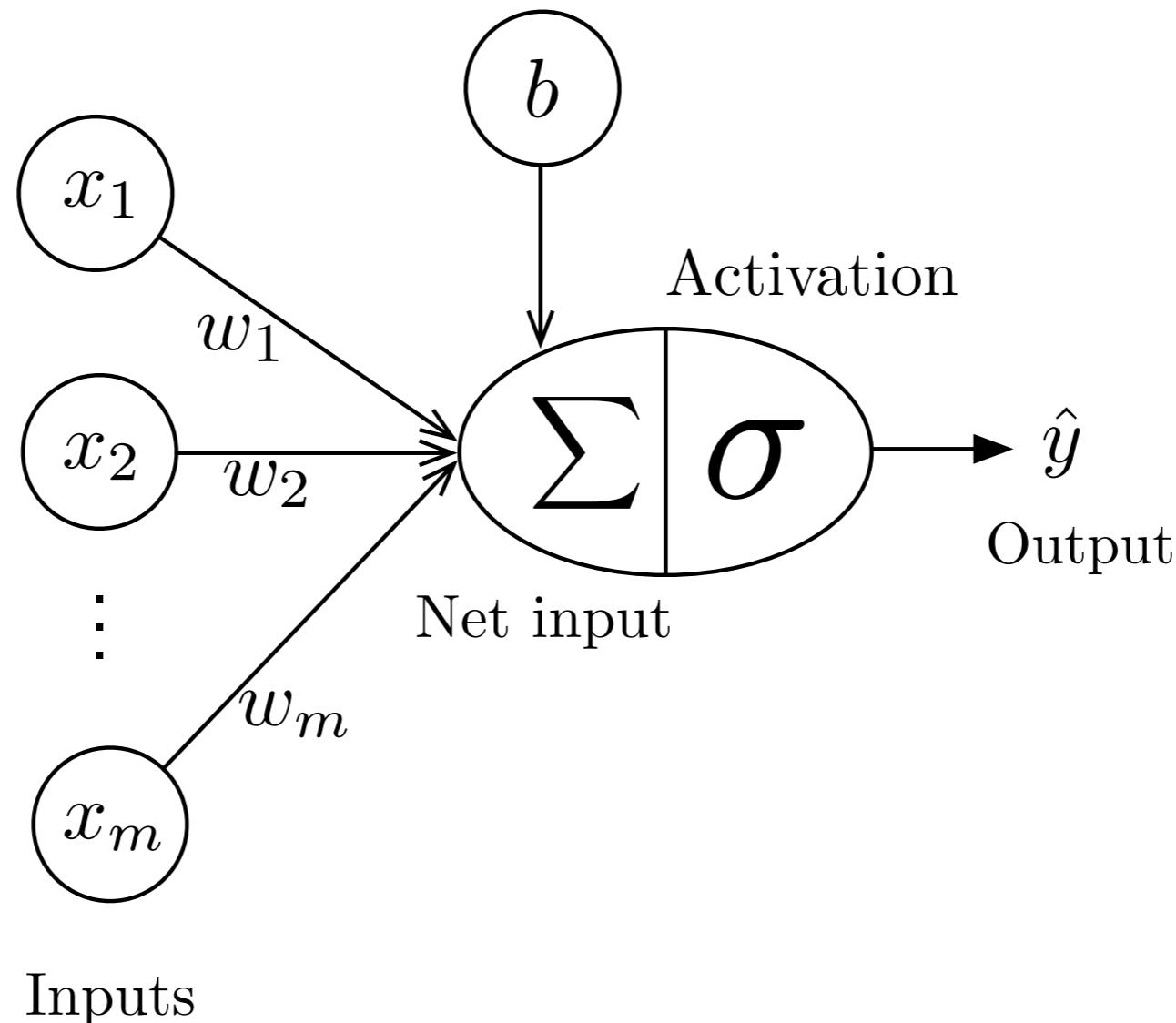
Softmax regression: "traditional method"

Represent digit as a long vector of pixels



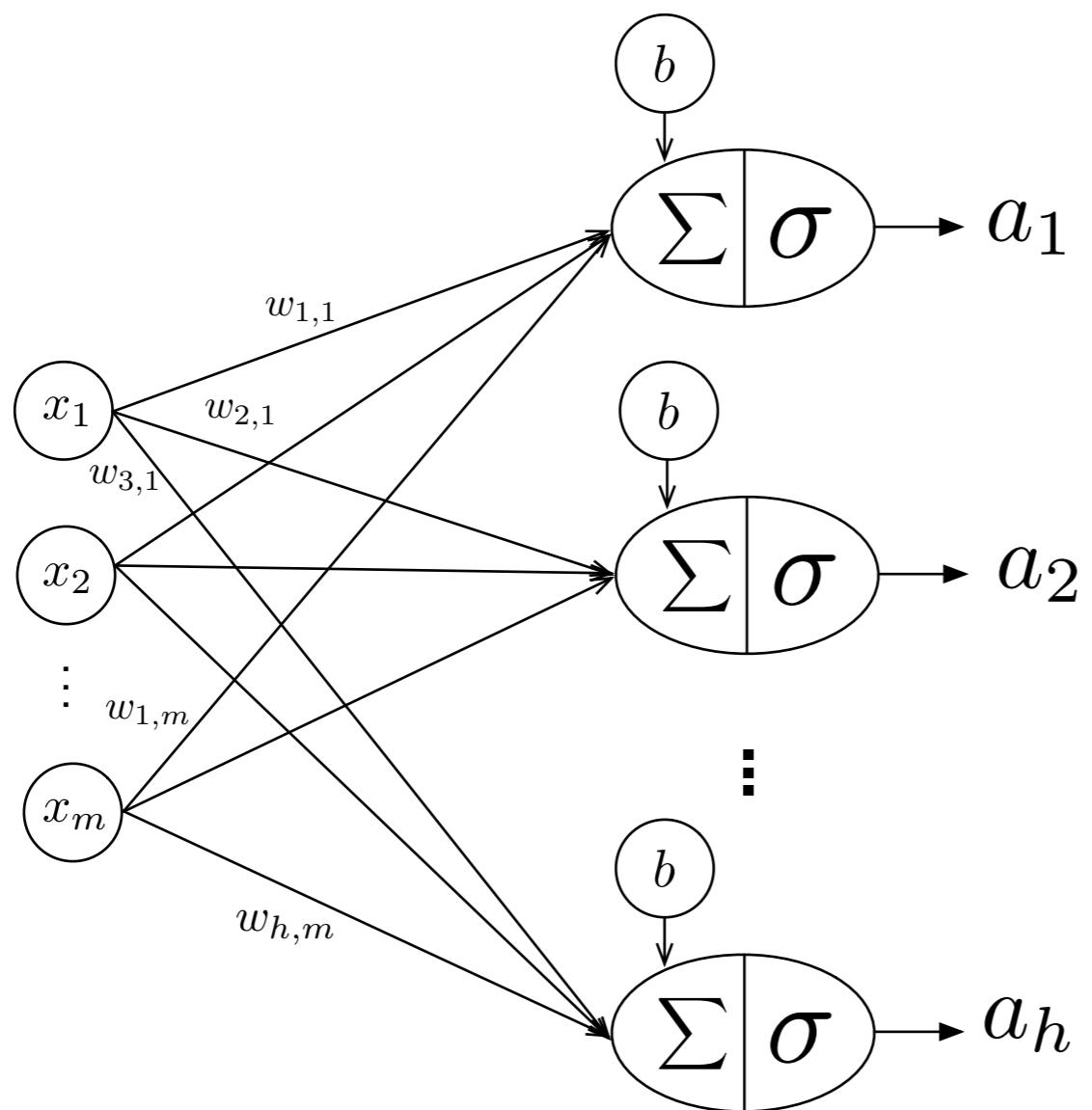
Note that I normalized pixels by factor 1/255 here

Previous Approach for Binary Classes



In logistic regression, the activation can be interpreted as $p(y=1/x)$

Another Approach Could be ...



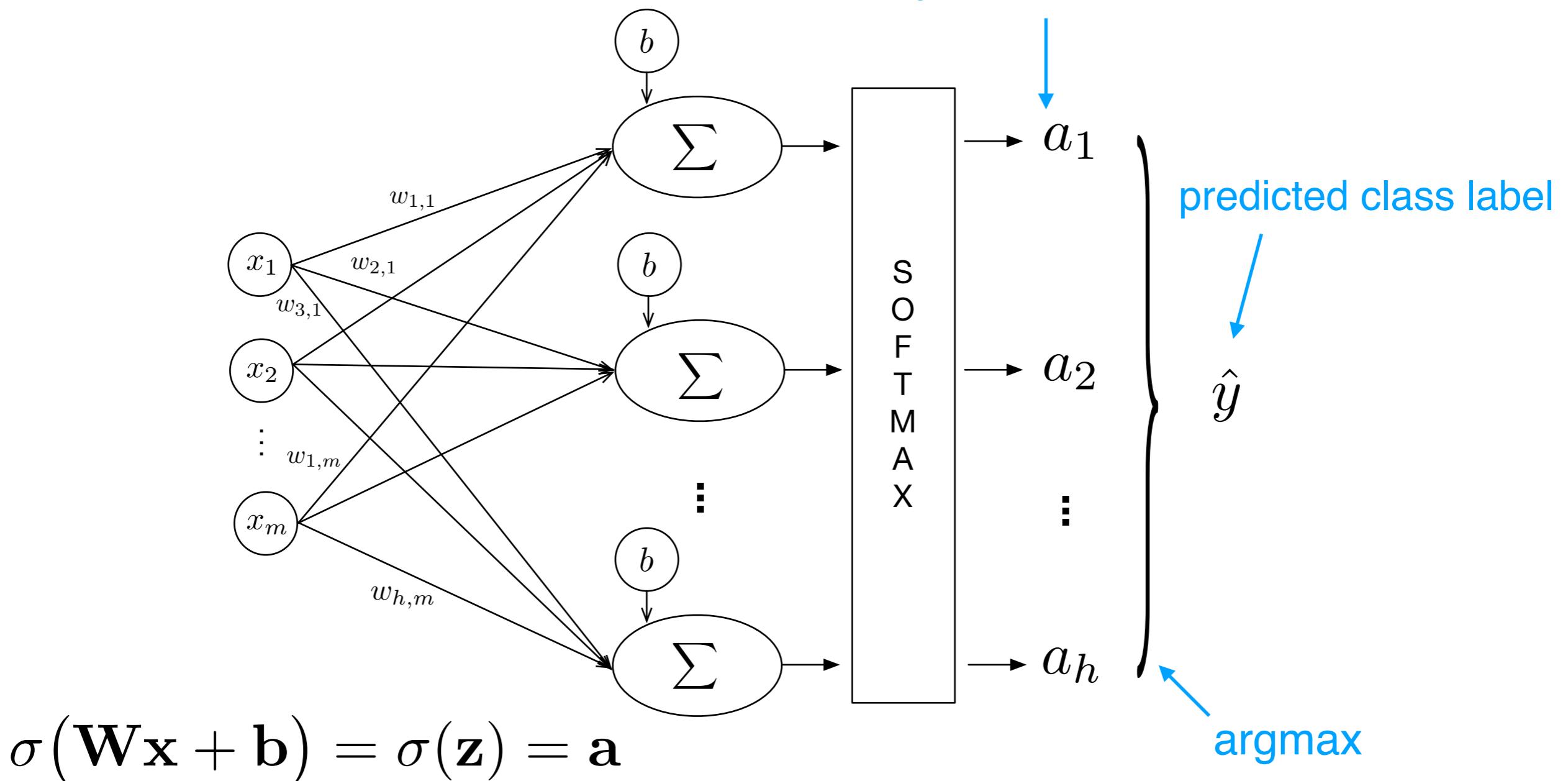
activations are
class-membership
probabilities
(NOT mutually
exclusive classes)

$$\sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) = \sigma(\mathbf{z}) = \mathbf{a}$$

$\mathbf{W} \in \mathbb{R}^{h \times m}$
where h is the number of classes

Multinomial Logistic Regression / Softmax Regression

activations are
class-membership probabilities
(mutually exclusive classes)

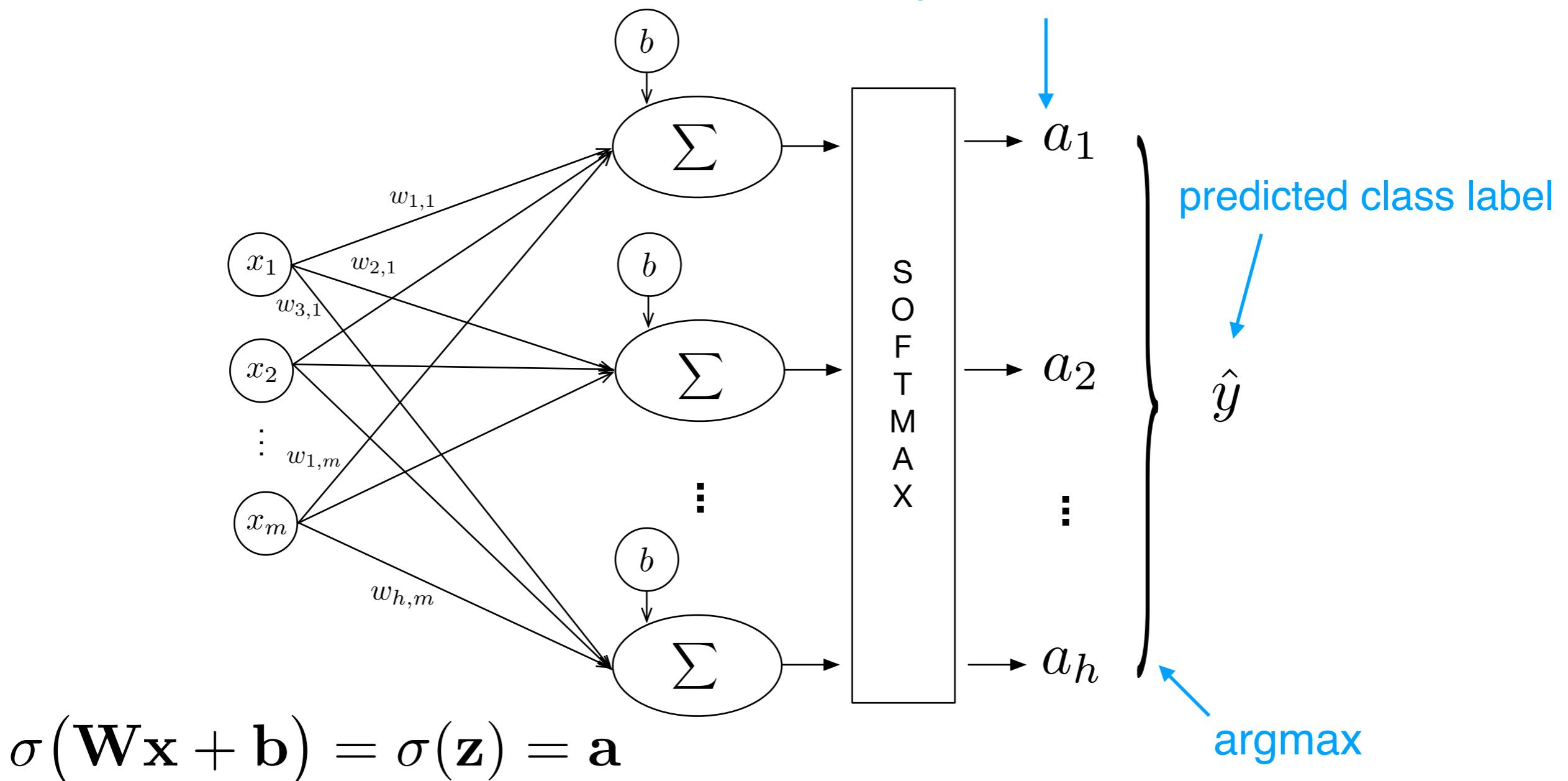


Representing Class Labels via Onehot Encoding for the Multi-category Cross Entropy Loss

1. Logistic Regression as an Artificial Neuron
2. Negative Log-Likelihood Loss
3. Logistic Regression Learning Rule
4. Logits and Cross Entropy
5. Logistic Regression Code Example
6. Generalizing to Multiple Classes: Softmax Regression
7. **OneHot Encoding and Multi-category Cross Entropy**
8. Softmax Regression Learning Rule
9. Softmax Regression Code Example

Multinomial Logistic Regression / Softmax Regression

activations are
class-membership probabilities
(mutually exclusive classes)



Softmax Activation

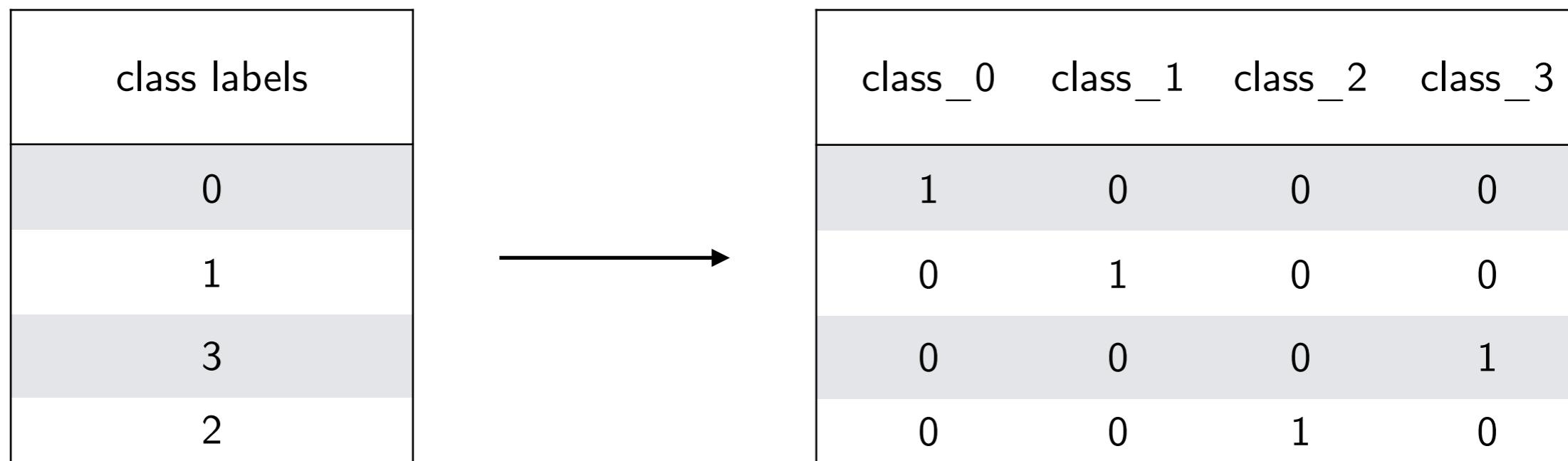
$$P(y = t \mid z_t^{[i]}) = \sigma_{\text{softmax}}(z_t^{[i]}) = \frac{e^{z_t^{[i]}}}{\sum_{j=1}^h e^{z_j^{[i]}}}$$

$t \in \{j \dots h\}$

h is the number of class labels

(Essentially, softmax is just an exponential function that normalizes the activations so that they sum up to 1)

Onehot Encoding Needed



Loss Function

(Multi-category) Cross Entropy
for h different class labels

$$\mathcal{L} = \sum_{i=1}^n \sum_{j=1}^h -y_j^{[i]} \log(a_j^{[i]})$$

This assumes one-hot encoded labels!

Loss Function

$$\mathcal{L}_{\text{binary}} = - \sum_{i=1}^n \left(y^{[i]} \log(a^{[i]}) + (1 - y^{[i]}) \log(1 - a^{[i]}) \right)$$

This assumes one-hot encoded labels!

$$\mathcal{L} = \sum_{i=1}^n \sum_{j=1}^h -y_j^{[i]} \log \left(a_j^{[i]} \right)$$

for h different class labels
(Multi-category) Cross Entropy

Cross Entropy Loss Function Example

$$\mathbf{Y}_{\text{onehot}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{A}_{\text{softmax outputs}} = \begin{bmatrix} 0.3792 & 0.3104 & 0.3104 \\ 0.3072 & 0.4147 & 0.2780 \\ 0.4263 & 0.2248 & 0.3490 \\ 0.2668 & 0.2978 & 0.4354 \end{bmatrix}$$

(4 training examples, 3 classes)

Cross Entropy Loss Function Example

1 training example

$$\mathbf{Y}_{\text{onehot}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{A}_{\text{softmax outputs}} = \begin{bmatrix} 0.3792 & 0.3104 & 0.3104 \\ 0.3072 & 0.4147 & 0.2780 \\ 0.4263 & 0.2248 & 0.3490 \\ 0.2668 & 0.2978 & 0.4354 \end{bmatrix}$$

(4 training examples, 3 classes)

$$\mathcal{L} = \sum_{i=1}^n \sum_{j=1}^h -y_j^{[i]} \log(a_j^{[i]})$$

$$\begin{aligned} \mathcal{L}^{[1]} &= [(-1) \cdot \log(0.3792)] \\ &\quad + [(-0) \cdot \log(0.3104)] \\ &\quad + [(-0) \cdot \log(0.3104)] \\ &= 0.969692... \end{aligned}$$

Cross Entropy Loss Function Example

$$\mathbf{Y}_{\text{onehot}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{A}_{\text{softmax outputs}} = \begin{bmatrix} 0.3792 & 0.3104 & 0.3104 \\ 0.3072 & 0.4147 & 0.2780 \\ 0.4263 & 0.2248 & 0.3490 \\ 0.2668 & 0.2978 & 0.4354 \end{bmatrix}$$

$$\begin{aligned}\mathcal{L}^{[1]} &= [(-1) \cdot \log(0.3792)] \\ &+ [(-0) \cdot \log(0.3104)] \\ &+ [(-0) \cdot \log(0.3104)] \\ &= 0.969692...\end{aligned}$$

$$\begin{aligned}\mathcal{L}^{[2]} &= [(-0) \cdot \log(0.3072)] \\ &+ [(-1) \cdot \log(0.4147)] \\ &+ [(-0) \cdot \log(0.2780)] \\ &= 0.880200...\end{aligned}$$

$$\begin{aligned}\mathcal{L}^{[3]} &= [(-0) \cdot \log(0.4263)] \\ &+ [(-0) \cdot \log(0.2248)] \\ &+ [(-1) \cdot \log(0.3490)] \\ &= 1.05268...\end{aligned}$$

$$\begin{aligned}\mathcal{L}^{[4]} &= [(-0) \cdot \log(0.2668)] \\ &+ [(-0) \cdot \log(0.2978)] \\ &+ [(-1) \cdot \log(0.4354)] \\ &= 0.831490...\end{aligned}$$

Cross Entropy Loss Function Example

$$\mathbf{Y}_{\text{onehot}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{A}_{\text{softmax outputs}} = \begin{bmatrix} 0.3792 & 0.3104 & 0.3104 \\ 0.3072 & 0.4147 & 0.2780 \\ 0.4263 & 0.2248 & 0.3490 \\ 0.2668 & 0.2978 & 0.4354 \end{bmatrix}$$

$$\begin{aligned}\mathcal{L}^{[1]} &= [(-1) \cdot \log(0.3792)] \\ &+ [(-0) \cdot \log(0.3104)] \\ &+ [(-0) \cdot \log(0.3104)] \\ &= 0.969692...\end{aligned}$$

$$\begin{aligned}\mathcal{L}^{[3]} &= [(-0) \cdot \log(0.4263)] \\ &+ [(-0) \cdot \log(0.2248)] \\ &+ [(-1) \cdot \log(0.3490)] \\ &= 1.05268...\end{aligned}$$

$$\begin{aligned}\mathcal{L}^{[2]} &= [(-0) \cdot \log(0.3072)] \\ &+ [(-1) \cdot \log(0.4147)] \\ &+ [(-0) \cdot \log(0.2780)] \\ &= 0.880200...\end{aligned}$$

$$\begin{aligned}\mathcal{L}^{[4]} &= [(-0) \cdot \log(0.2668)] \\ &+ [(-0) \cdot \log(0.2978)] \\ &+ [(-1) \cdot \log(0.4354)] \\ &= 0.831490...\end{aligned}$$

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^h -y_j^{[i]} \log(a_j^{[i]})$$

$$\approx 0.9335$$

Cross Entropy Hands-On Example

Understanding Onehot Encoding and Cross Entropy in PyTorch

```
import torch
```

Onehot Encoding

```
def to_onehot(y, num_classes):
    y_onehot = torch.zeros(y.size(0), num_classes)
    y_onehot.scatter_(1, y.view(-1, 1).long(), 1).float()
    return y_onehot

y = torch.tensor([0, 1, 2, 2])

y_enc = to_onehot(y, 3)

print('one-hot encoding:\n', y_enc)
```

one-hot encoding:
tensor([[1., 0., 0.],
 [0., 1., 0.],
 [0., 0., 1.],
 [0., 0., 1.]])

<https://github.com/rasbt/stat453-deep-learning-ss21/blob/master/L07/code/cross-entropy-pytorch.ipynb>

Softmax Regression Derivatives for Gradient Descent

1. Logistic Regression as an Artificial Neuron
2. Negative Log-Likelihood Loss
3. Logistic Regression Learning Rule
4. Logits and Cross Entropy
5. Logistic Regression Code Example
6. Generalizing to Multiple Classes: Softmax Regression
7. OneHot Encoding and Multi-category Cross Entropy
- 8. Softmax Regression Learning Rule**
9. Softmax Regression Code Example

The Same Overall Concept Applies ...

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

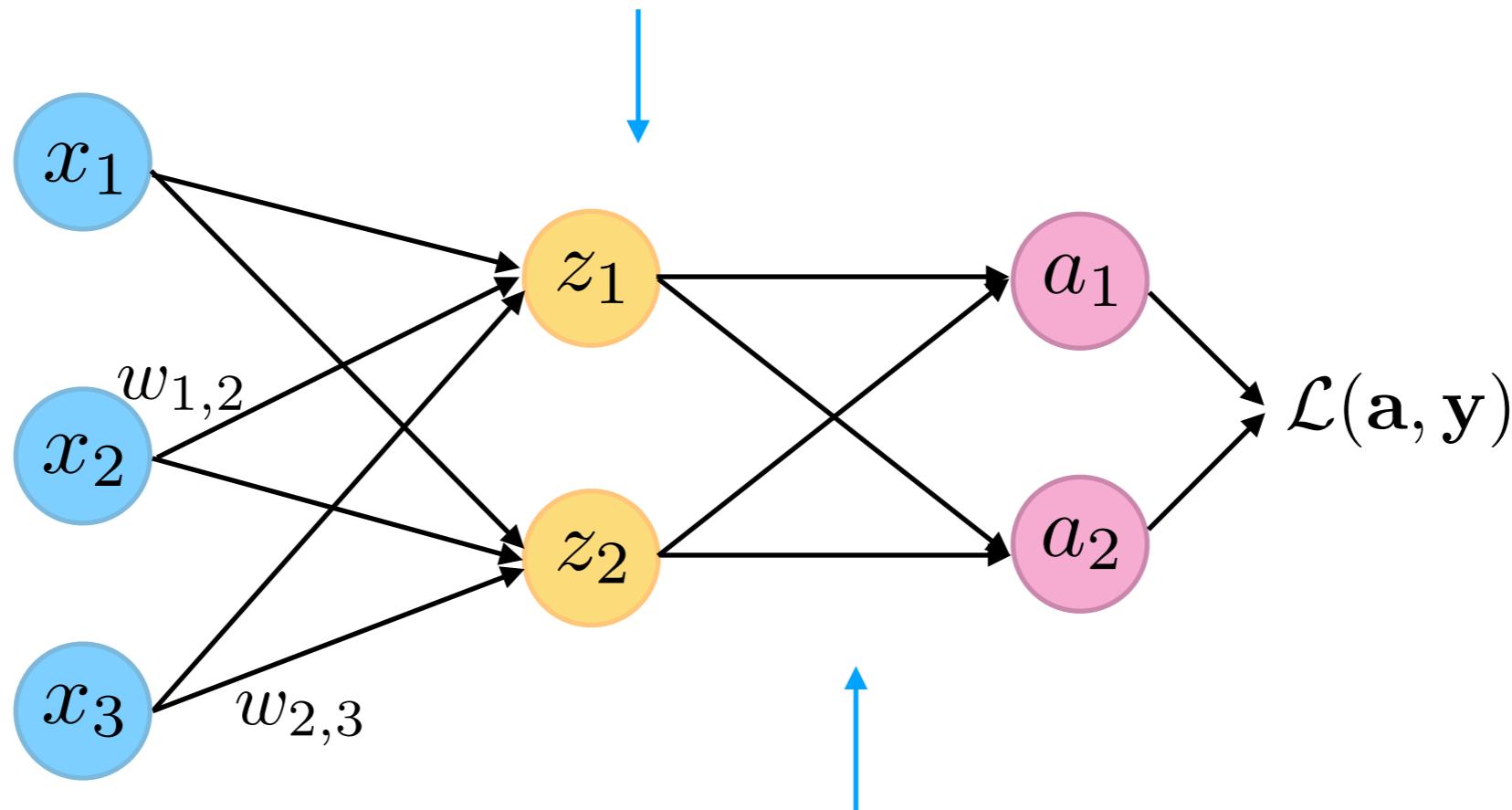
$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial a} \frac{da}{dz} \frac{\partial z}{\partial w_i}$$

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

Softmax Regression Sketch

(This is NOT a multi-layer neural network; still 1-layer, no hidden layer)

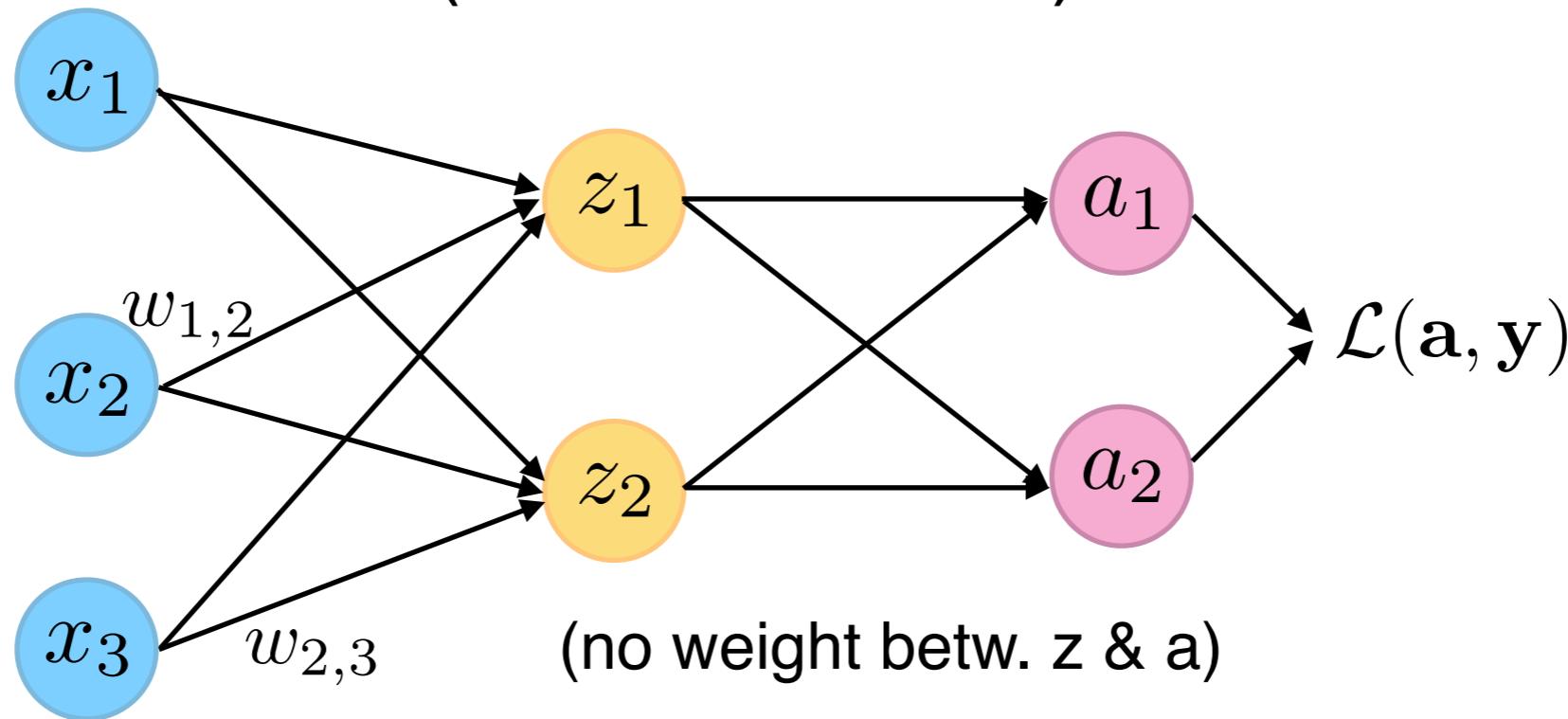
Note that I put the logits (net inputs) as the intermediate step



The activation function
(softmax) would be applied
here to obtain the activations

Softmax Regression Sketch

(bias not shown)



Multivariable
chain rule

$$\frac{\partial L}{\partial w_{1,2}} = \frac{\partial L}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial w_{1,2}} + \frac{\partial L}{\partial a_2} \frac{\partial a_2}{\partial z_1} \frac{\partial z_1}{\partial w_{1,2}}$$

\downarrow \downarrow \downarrow
 $-\frac{y_1}{a_1}$ $a_1(1 - a_1)$ x_2
 \downarrow \downarrow \downarrow
 $-\frac{y_2}{a_2}$ $-a_2 a_1$ x_2

$$\frac{\partial L}{\partial w_{1,2}} = \boxed{\frac{\partial L}{\partial a_1}} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial w_{1,2}} + \frac{\partial L}{\partial a_2} \frac{\partial a_2}{\partial z_1} \frac{\partial z_1}{\partial w_{1,2}}$$

\downarrow

$$-\frac{y_1}{a_1}$$

$$\frac{\partial L}{\partial a_1} = \frac{\partial}{\partial a_1} \left[\sum_{j=1}^h -y_j \log(a_j) \right]$$

$$= \frac{\partial}{\partial a_1} [-y_1 \log(a_1)]$$

$$= -\frac{y_1}{a_1}$$

$$\frac{\partial L}{\partial w_{1,2}} = \frac{\partial L}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial w_{1,2}} + \frac{\partial L}{\partial a_2} \frac{\partial a_2}{\partial z_1} \frac{\partial z_1}{\partial w_{1,2}}$$

↓

$$a_1(1 - a_1)$$

$$\begin{aligned}\frac{\partial a_1}{\partial z_1} &= \frac{\partial}{\partial z_1} \left[\frac{e^{z_1}}{\sum_{j=1}^h e^{z_j}} \right] \\ &= \frac{\left[\sum_{j=1}^h e^{z_j} \right] \frac{\partial}{\partial z_1} e^{z_1} - e^{z_1} \frac{\partial}{\partial z_1} \left[\sum_{j=1}^h e^{z_j} \right]}{\left[\sum_{j=1}^h e^{z_j} \right]^2}\end{aligned}$$

$$= \frac{\left[\sum_{j=1}^h e^{z_j} \right] e^{z_1} - e^{z_1} e^{z_1}}{\left[\sum_{j=1}^h e^{z_j} \right]^2}$$

$$= \frac{e^{z_1} \left(\left[\sum_{j=1}^h e^{z_j} \right] - e^{z_1} \right)}{\left[\sum_{j=1}^h e^{z_j} \right]^2} = \frac{e^{z_1}}{\left[\sum_{j=1}^h e^{z_j} \right]} \cdot \frac{\left[\sum_{j=1}^h e^{z_j} \right] - e^{z_1}}{\left[\sum_{j=1}^h e^{z_j} \right]} = a_1(1 - a_1)$$

	Function	Derivative
Sum Rule	$f(x) + g(x)$	$f'(x) + g'(x)$
Difference Rule	$f(x) - g(x)$	$f'(x) - g'(x)$
Product Rule	$f(x)g(x)$	$f'(x)g(x) + f(x)g'(x)$
Quotient Rule	$f(x)/g(x)$	$[g(x)f'(x) - f(x)g'(x)]/[g(x)]^2$
Reciprocal Rule	$1/f(x)$	$-[f'(x)]/[f(x)]^2$
Chain Rule	$f(g(x))$	$f'(g(x))g'(x)$

$$\frac{\partial L}{\partial w_{1,2}} = \frac{\partial L}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial w_{1,2}} + \frac{\partial L}{\partial a_2} \frac{\partial a_2}{\partial z_1} \frac{\partial z_1}{\partial w_{1,2}}$$

↓

$-a_2 a_1$

$$\frac{\partial a_2}{\partial z_1} = \frac{\partial}{\partial z_1} \left[\frac{e^{z_2}}{\sum_{j=1}^h e^{z_j}} \right]$$

	Function	Derivative
Sum Rule	$f(x) + g(x)$	$f'(x) + g'(x)$
Difference Rule	$f(x) - g(x)$	$f'(x) - g'(x)$
Product Rule	$f(x)g(x)$	$f'(x)g(x) + f(x)g'(x)$
Quotient Rule	$f(x)/g(x)$	$[g(x)f'(x) - f(x)g'(x)]/[g(x)]^2$
Reciprocal Rule	$1/f(x)$	$-\frac{f'(x)}{[f(x)]^2}$
Chain Rule	$f(g(x))$	$f'(g(x))g'(x)$

$$= \frac{\left[\sum_{j=1}^h e^{z_j} \right] \frac{\partial}{\partial z_1} e^{z_2} - e^{z_2} \frac{\partial}{\partial z_1} \left[\sum_{j=1}^h e^{z_j} \right]}{\left[\sum_{j=1}^h e^{z_j} \right]^2}$$

$$= \frac{0 - e^{z_2} e^{z_1}}{\left[\sum_{j=1}^h e^{z_j} \right]^2}$$

$$= \frac{-e^{z_2}}{\left[\sum_{j=1}^h e^{z_j} \right]} \cdot \frac{e^{z_1}}{\left[\sum_{j=1}^h e^{z_j} \right]} = -a_2 a_1$$

$$\frac{\partial L}{\partial w_{1,2}} = \frac{\partial L}{\partial a_1} \frac{\partial a_1}{\partial z_1} \boxed{\frac{\partial z_1}{\partial w_{1,2}}} + \frac{\partial L}{\partial a_2} \frac{\partial a_2}{\partial z_1} \frac{\partial z_1}{\partial w_{1,2}}$$

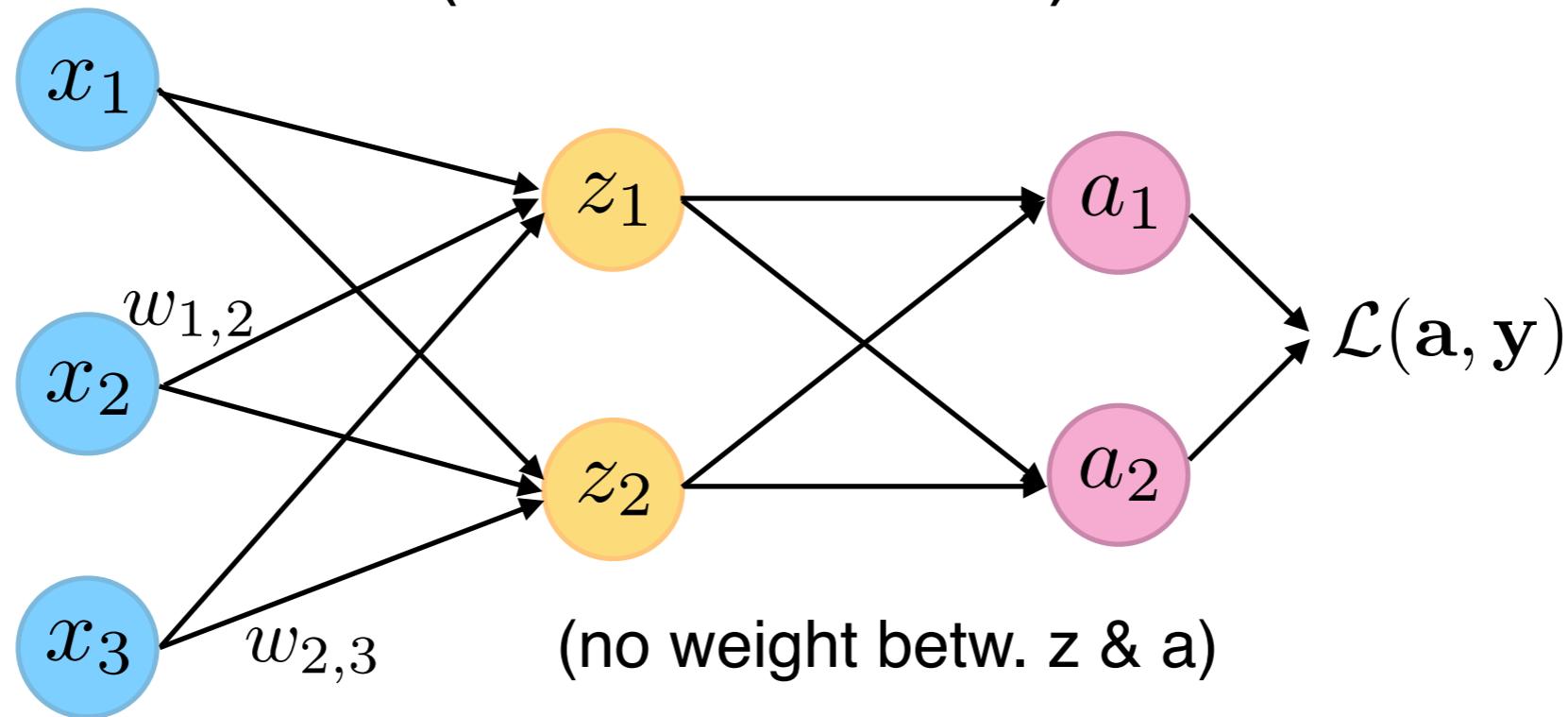
\downarrow

x_2

$$= \frac{\partial}{\partial w_{1,2}} [w_{1,2} \cdot x_2 + b]$$

$$= x_2$$

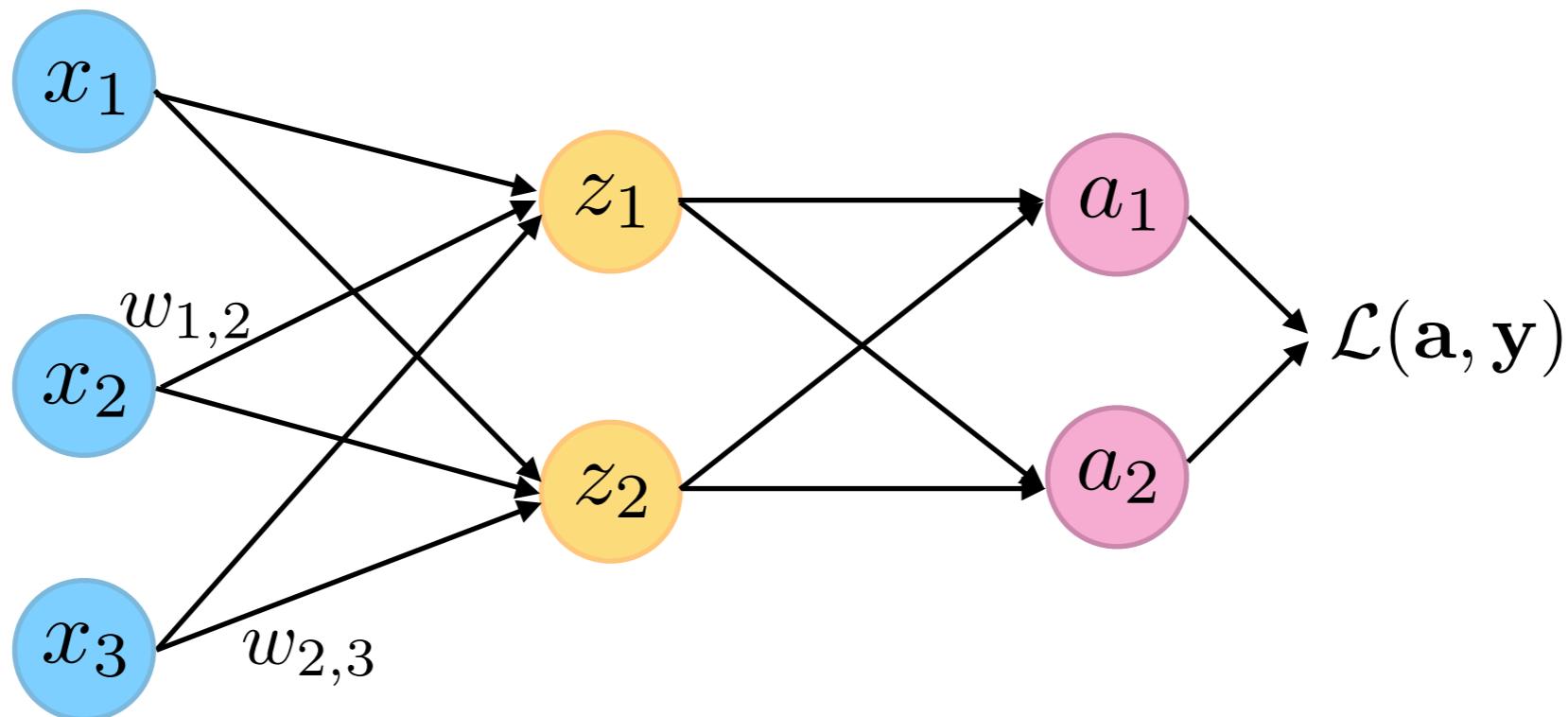
Softmax Regression Sketch (bias not shown)



Multivariable
chain rule

$$\begin{aligned}\frac{\partial L}{\partial w_{1,2}} &= \frac{\partial L}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial w_{1,2}} + \frac{\partial L}{\partial a_2} \frac{\partial a_2}{\partial z_1} \frac{\partial z_1}{\partial w_{1,2}} \\ &= \frac{-y_1}{a_1} [a_1(1 - a_1)]x_2 + \frac{-y_2}{a_2} (-a_2 a_1)x_2 \\ &= (y_2 a_1 - y_1 + y_1 a_1)x_2 \\ &= (a_1(y_1 + y_2) - y_1)x_2 \\ &= -(y_1 - a_1)x_2\end{aligned}$$

Softmax Regression Sketch



Vectorized Form:

$$\begin{aligned}\frac{\partial L}{\partial w_{1,2}} &= \frac{\partial L}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial w_{1,2}} + \frac{\partial L}{\partial a_2} \frac{\partial a_2}{\partial z_1} \frac{\partial z_1}{\partial w_{1,2}} \\ &= \frac{-y_1}{a_1} [a_1(1 - a_1)]x_2 + \frac{-y_2}{a_2} (-a_2 a_1)x_2 \\ &= (y_2 a_1 - y_1 + y_1 a_1)x_2 \\ &= (a_1(y_1 + y_2) - y_1)x_2 \\ &= -(y_1 - a_1)x_2\end{aligned}\qquad \nabla_{\mathbf{W}} \mathcal{L} = -(\mathbf{X}^T (\mathbf{Y} - \mathbf{A}))^\top$$

where $\mathbf{W} \in \mathbb{R}^{k \times m}$

$\mathbf{X} \in \mathbb{R}^{n \times m}$

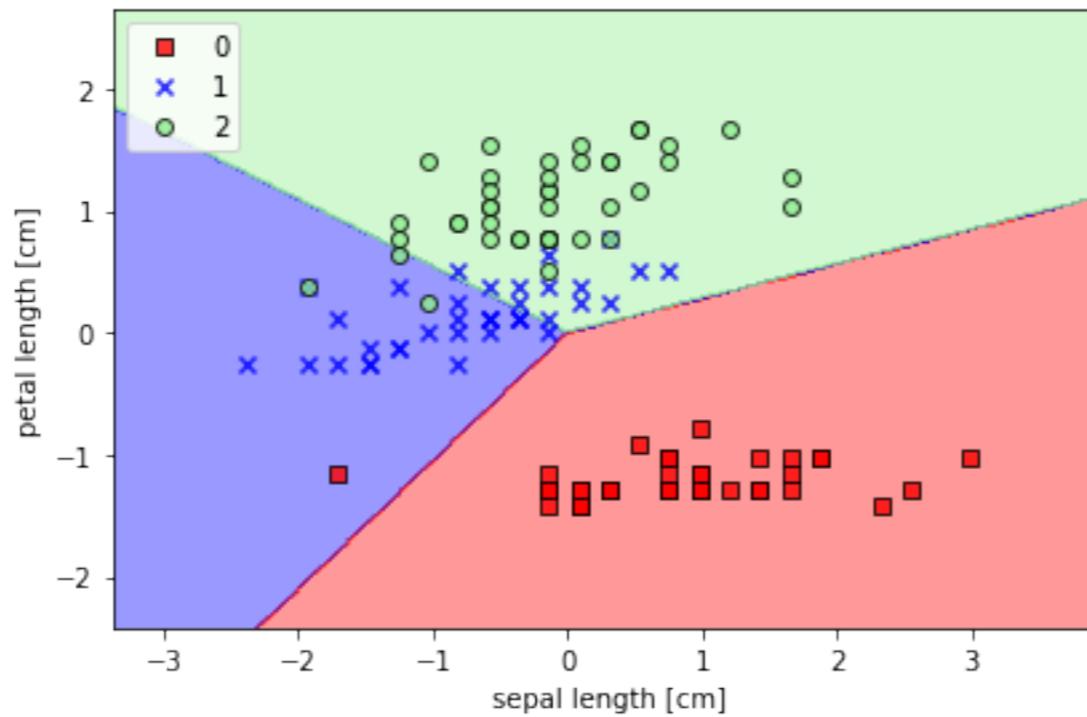
$\mathbf{A} \in \mathbb{R}^{n \times k}$

$\mathbf{Y} \in \mathbb{R}^{n \times k}$

Softmax Regression Code Example

1. Logistic Regression as an Artificial Neuron
2. Negative Log-Likelihood Loss
3. Logistic Regression Learning Rule
4. Logits and Cross Entropy
5. Logistic Regression Code Example
6. Generalizing to Multiple Classes: Softmax Regression
7. OneHot Encoding and Multi-category Cross Entropy
8. Softmax Regression Learning Rule
- 9. Softmax Regression Code Example**

Softmax Regression Hands-On Example (Iris)

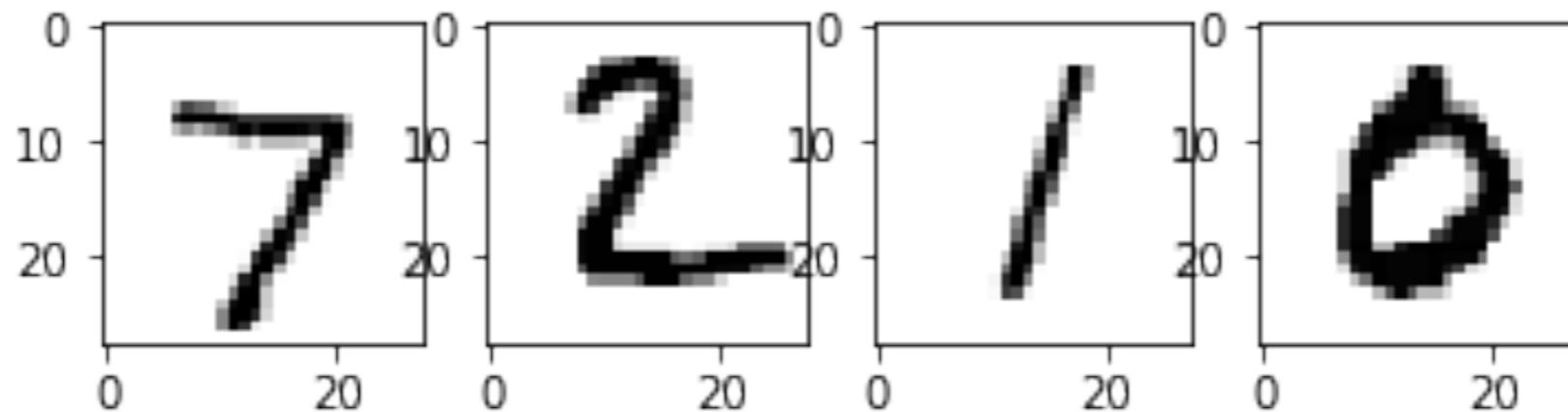


"From scratch" implementation & PyTorch autodiff implementation

[https://github.com/rasbt/stat453-deep-learning-ss21/blob/
master/L08/code/softmax-regression_scratch.ipynb](https://github.com/rasbt/stat453-deep-learning-ss21/blob/master/L08/code/softmax-regression_scratch.ipynb)

Softmax Regression Hands-On (MNIST)

Using PyTorch (with autodiff an nn.Module)



<https://github.com/rasbt/stat453-deep-learning-ss21/blob/master/L08/code/softmax-regression-mnist.ipynb>

PyTorch Loss-Input Confusion (Cheatsheet)

- `torch.nn.functional.binary_cross_entropy` takes logistic sigmoid values as inputs
- `torch.nn.functional.binary_cross_entropy_with_logits` takes logits as inputs
- `torch.nn.functional.cross_entropy` takes logits as inputs (performs `log_softmax` internally)
- `torch.nn.functional.nll_loss` is like `cross_entropy` but takes log-probabilities (log-softmax) values as inputs