

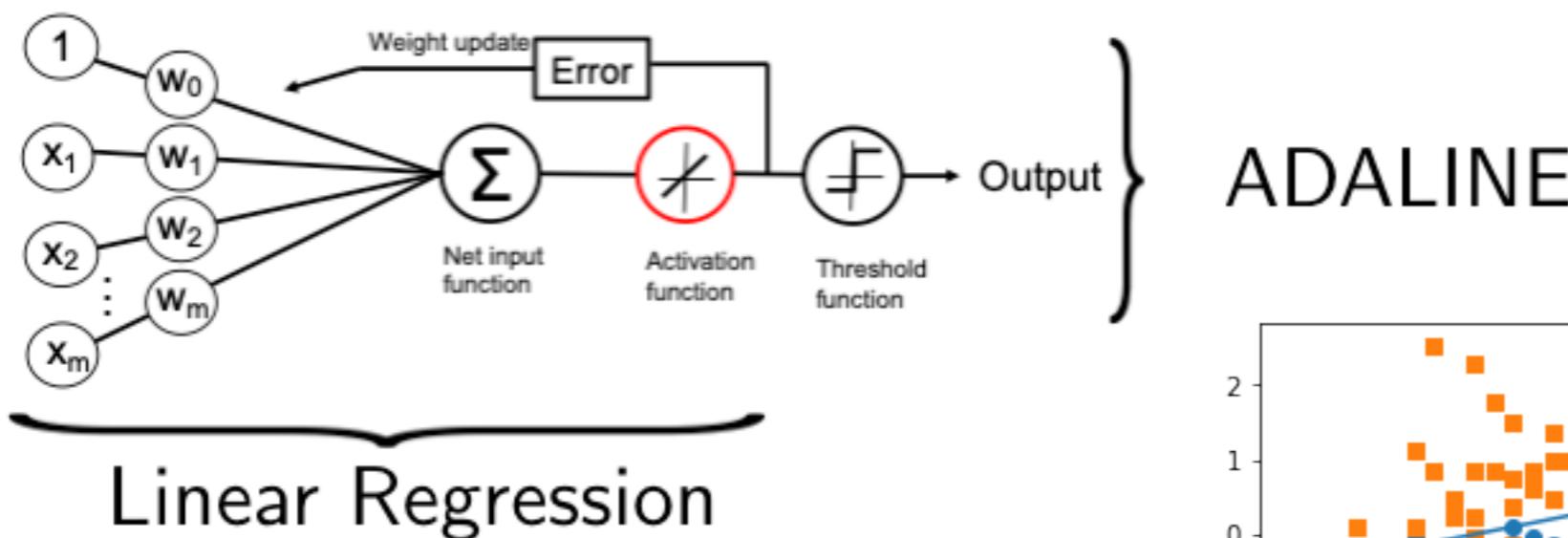
Lecture 07

Logistic Regression and Multi-class Classification

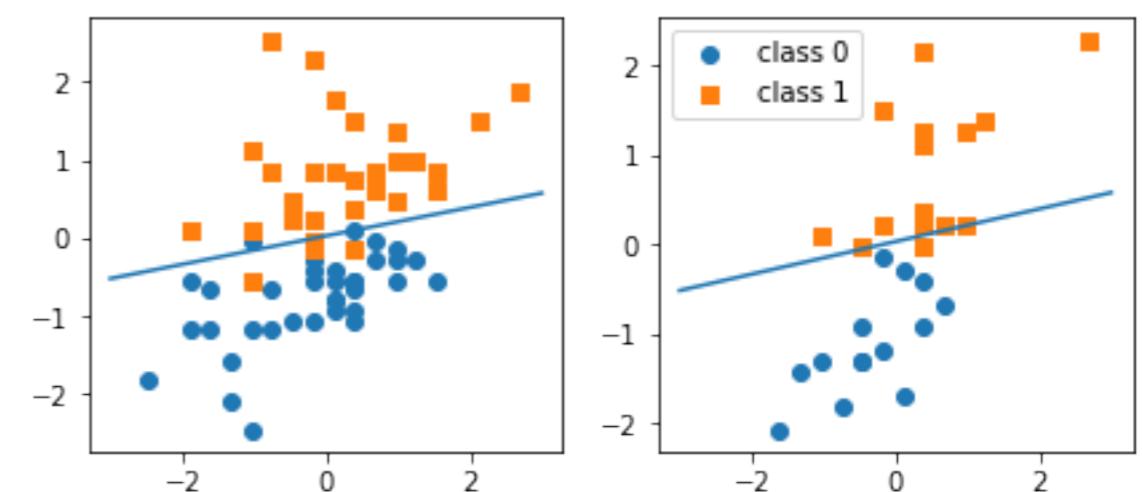
STAT 453: Deep Learning, Spring 2020
Sebastian Raschka
pages.stat.wisc.edu/~sraschka/teaching/stat453-ss2020/

Last Lecture(s):

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

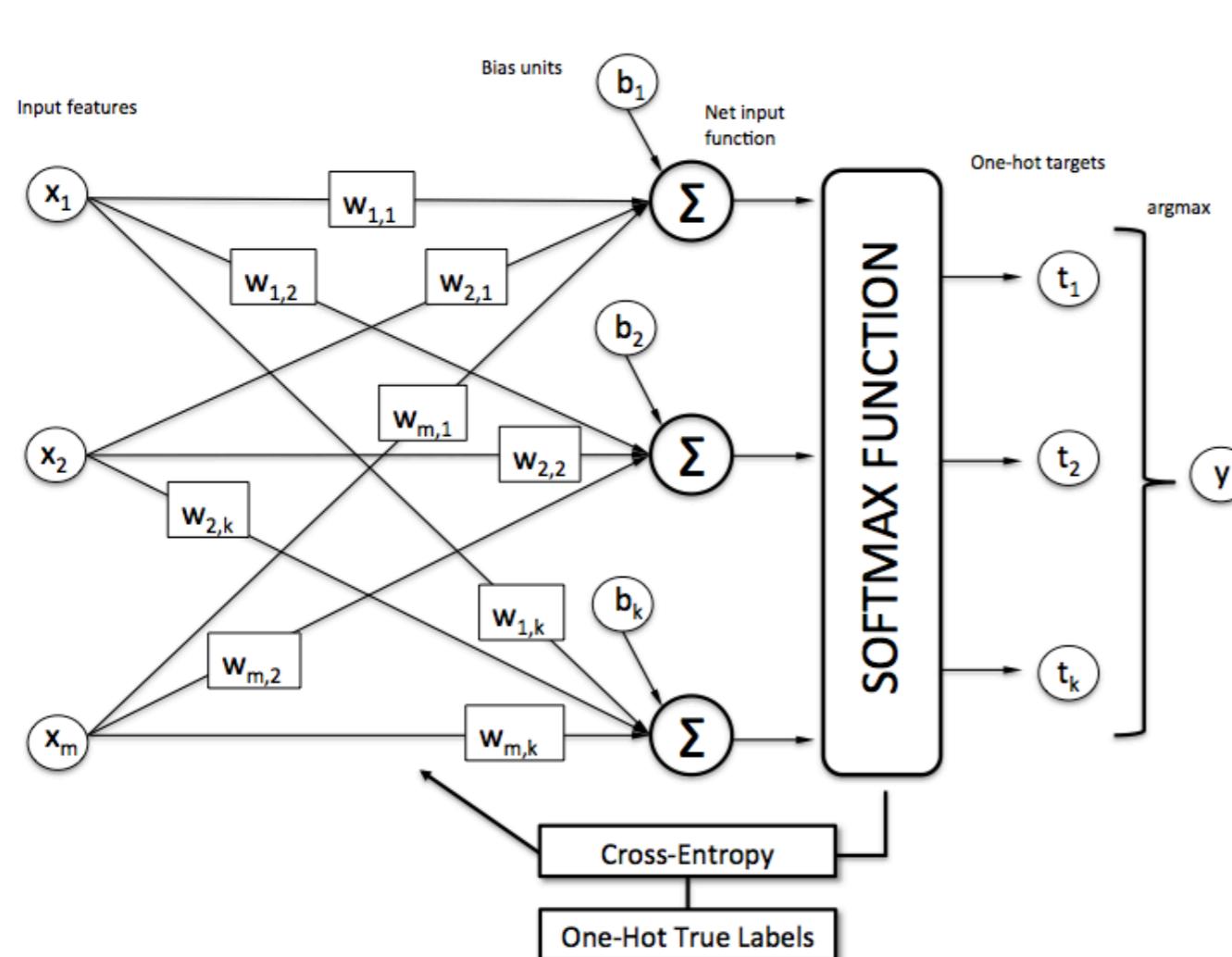


ADALINE

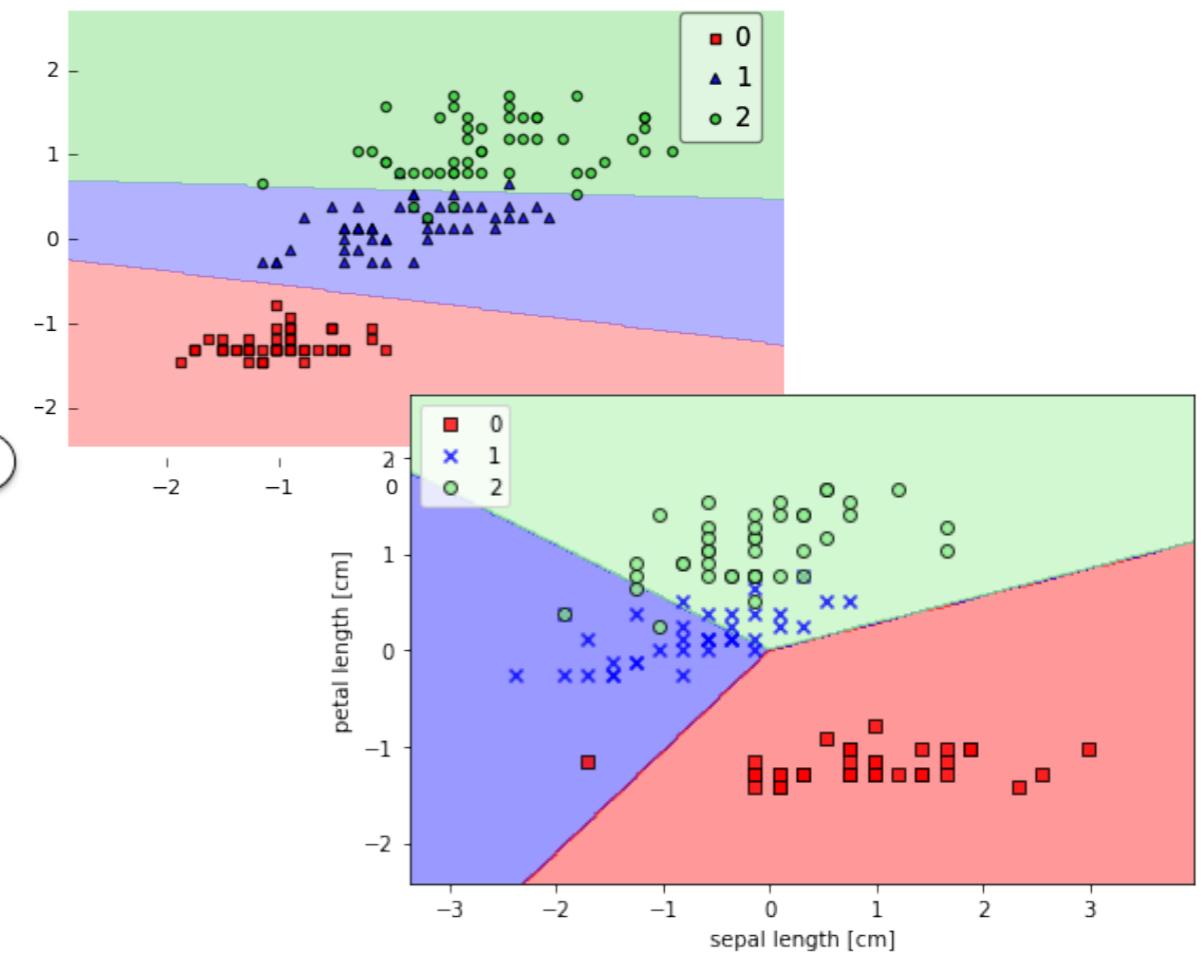


This Lecture:

- 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

Logistic Regression

Negative Log-Likelihood Loss

Logistic Regression Learning Rule

Softmax Regression

Softmax Loss Gradients

Logistic Regression for Binary Classification

Logistic Regression

Negative Log-Likelihood Loss

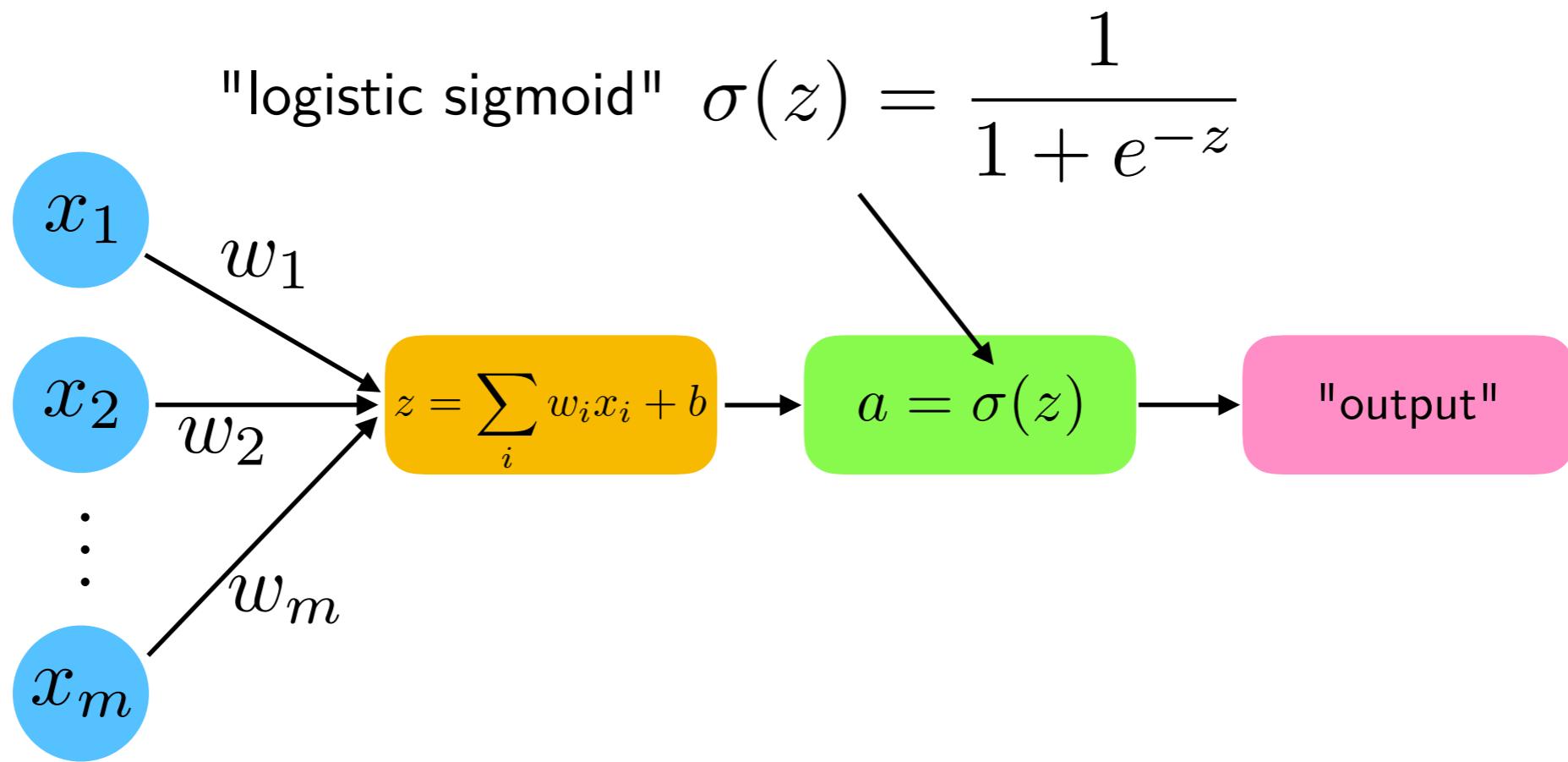
Logistic Regression Learning Rule

Softmax Regression

Softmax Loss Gradients

Logistic Regression Neuron

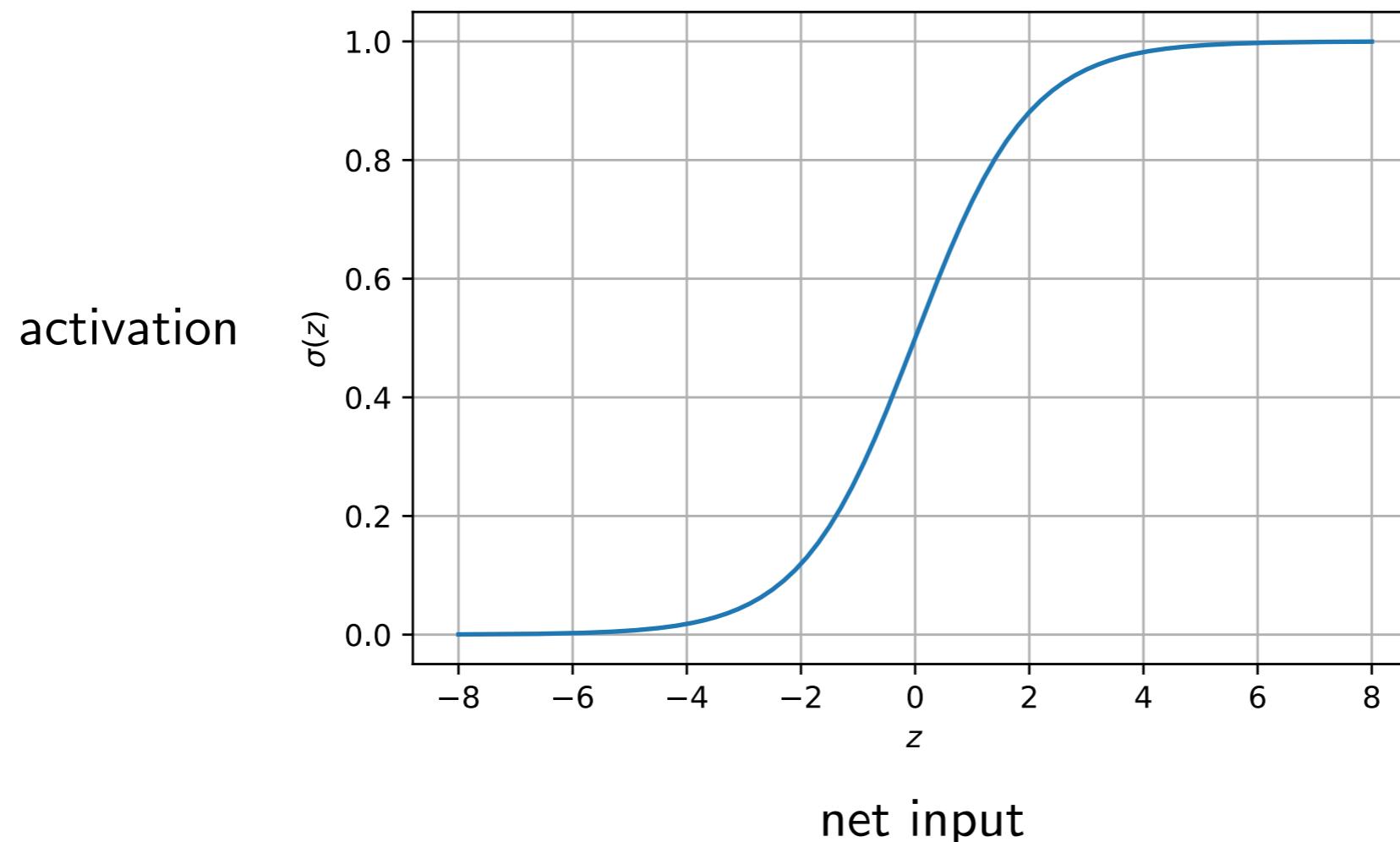
For binary classes $y \in \{0, 1\}$



- In ADALINE, the activation function was an identity function, $\sigma(z) = z$
- In HW2, the activation function was different sigmoid function, $\sigma(z) = (e^z - e^{-z})/(e^z + e^{-z})$
- In both cases, ADALINE & HW2, we used MSE as loss function: $\text{MSE} = \frac{1}{n} \sum_i (a^{[i]} - y^{[i]})^2$

Logistic Sigmoid Function

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



Logistic Regression

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:

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

rewrite more compactly

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

Logistic Regression

And for multiple training examples, we want to maximize:

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

by finding *good/optimal* parameters

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

Logistic Regression Loss Function

Logistic Regression

Negative Log-Likelihood Loss

Logistic Regression Learning Rule

Softmax Regression

Softmax Loss Gradients

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)

Training Logistic Regression

Logistic Regression

Negative Log-Likelihood Loss

Logistic Regression Learning Rule

Softmax Regression

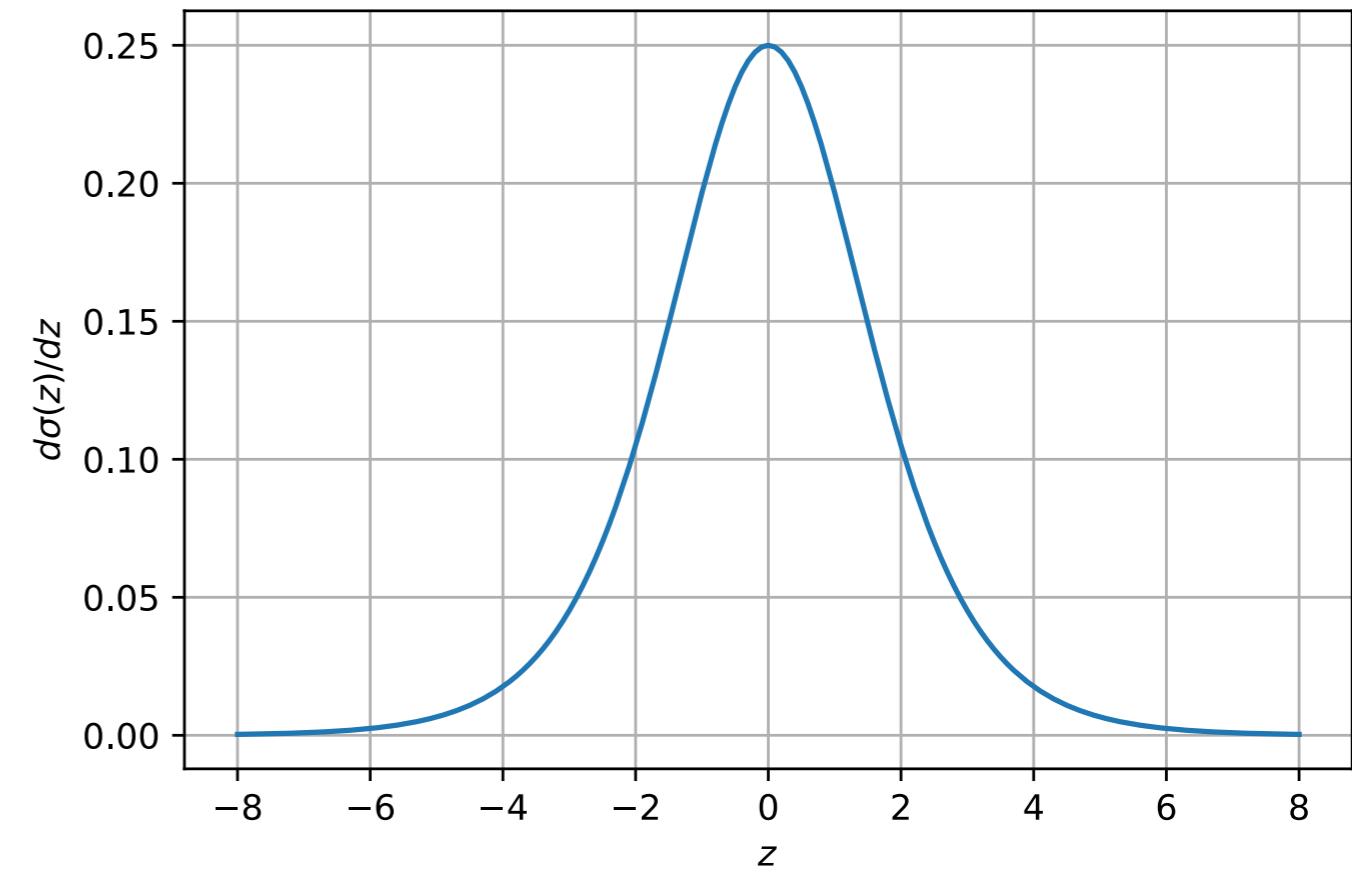
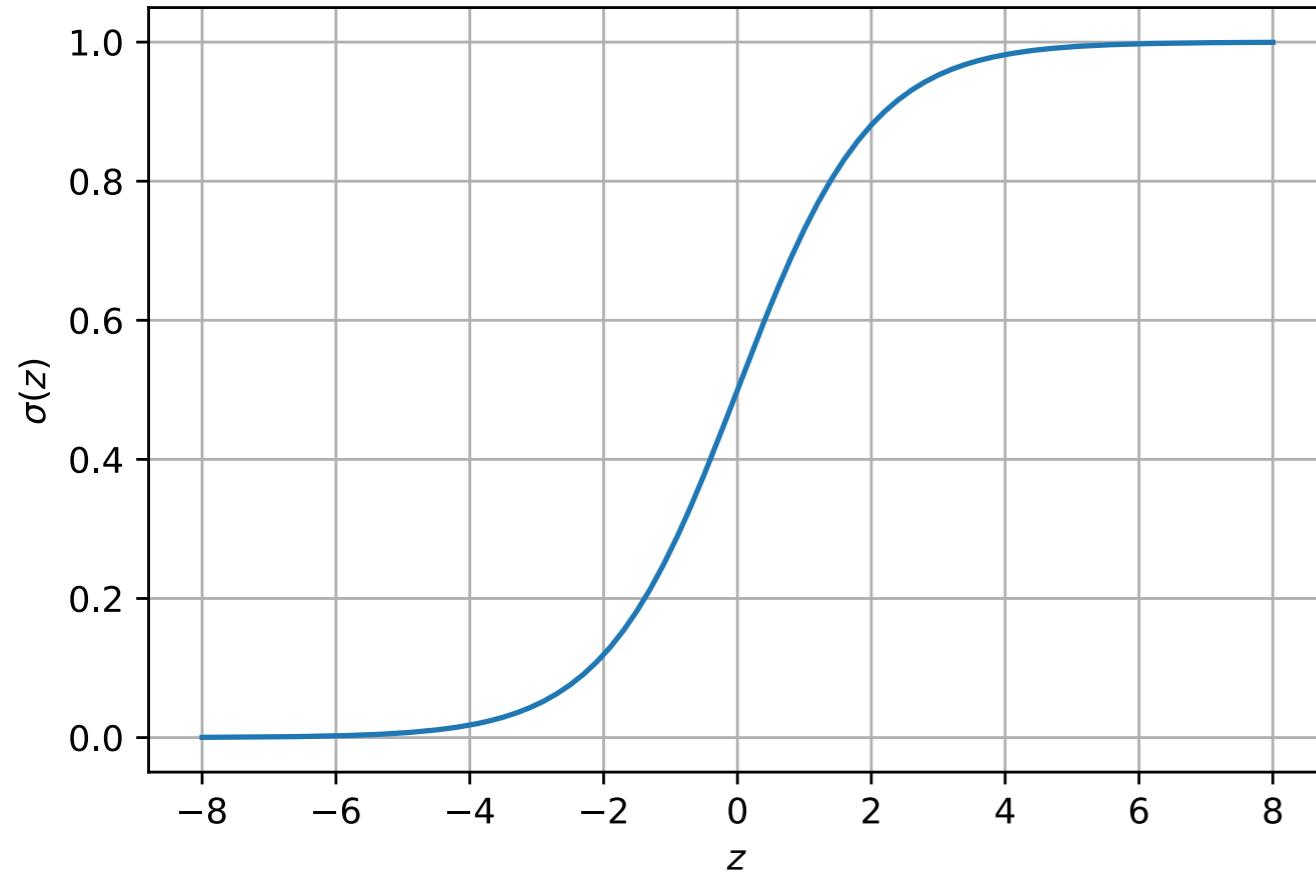
Softmax Loss Gradients

Logistic Sigmoid Derivative

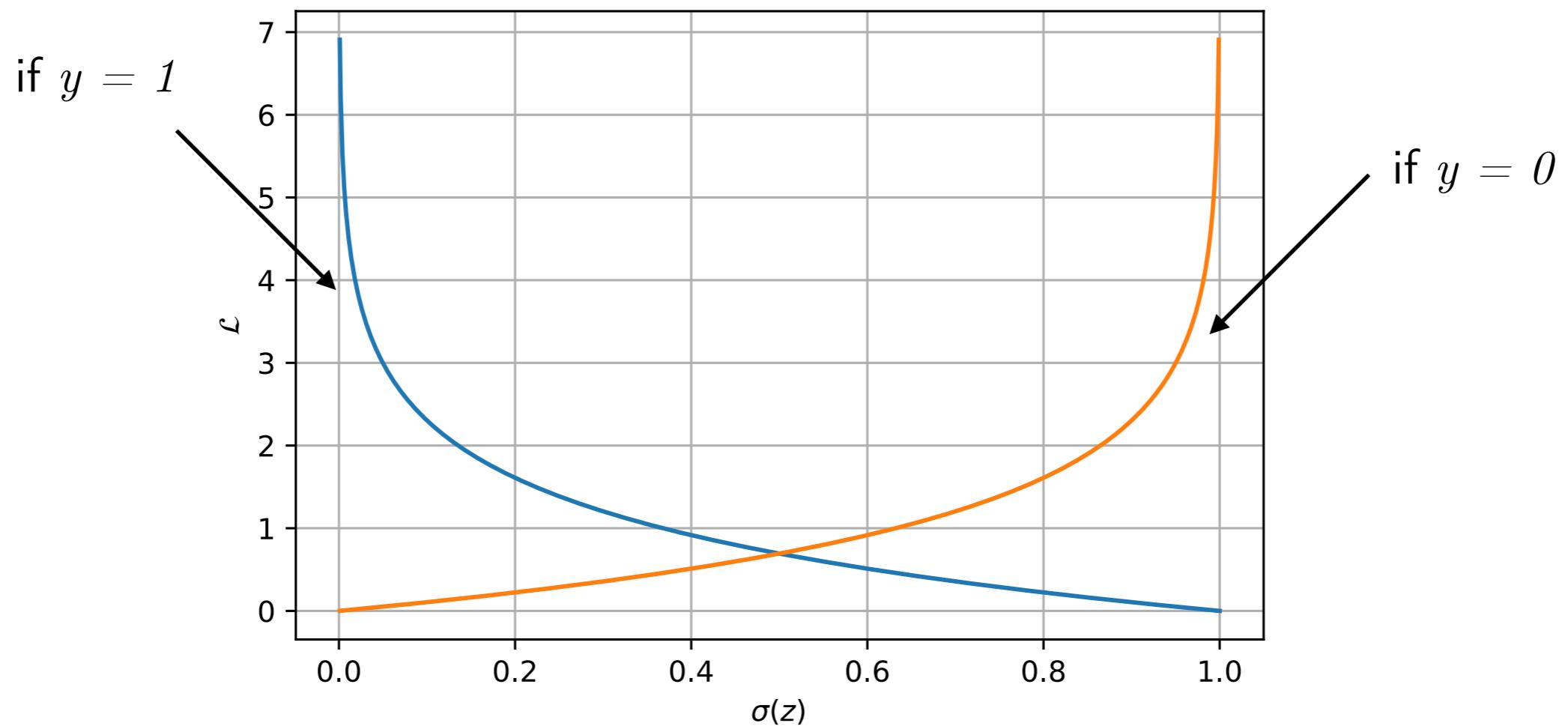
Generally, another nice property of the logistic sigmoid (in multi-layer nets) is that it has nice derivatives!

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

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



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

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

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

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

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

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

Learning Rule for Logistic Regression

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

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

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

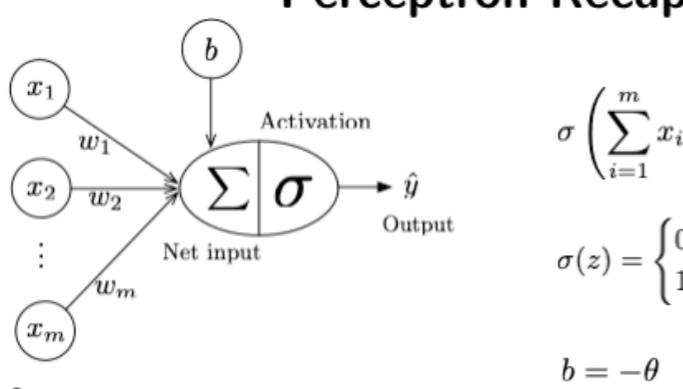
$$\frac{da}{dz} = \frac{e^{-z}}{(1+e^{-z})^2} = a \cdot (1-a) \longrightarrow \frac{\partial \mathcal{L}}{\partial z} = y - a \longrightarrow \frac{\partial \mathcal{L}}{\partial w_j} = (y - a)x_j$$

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

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

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

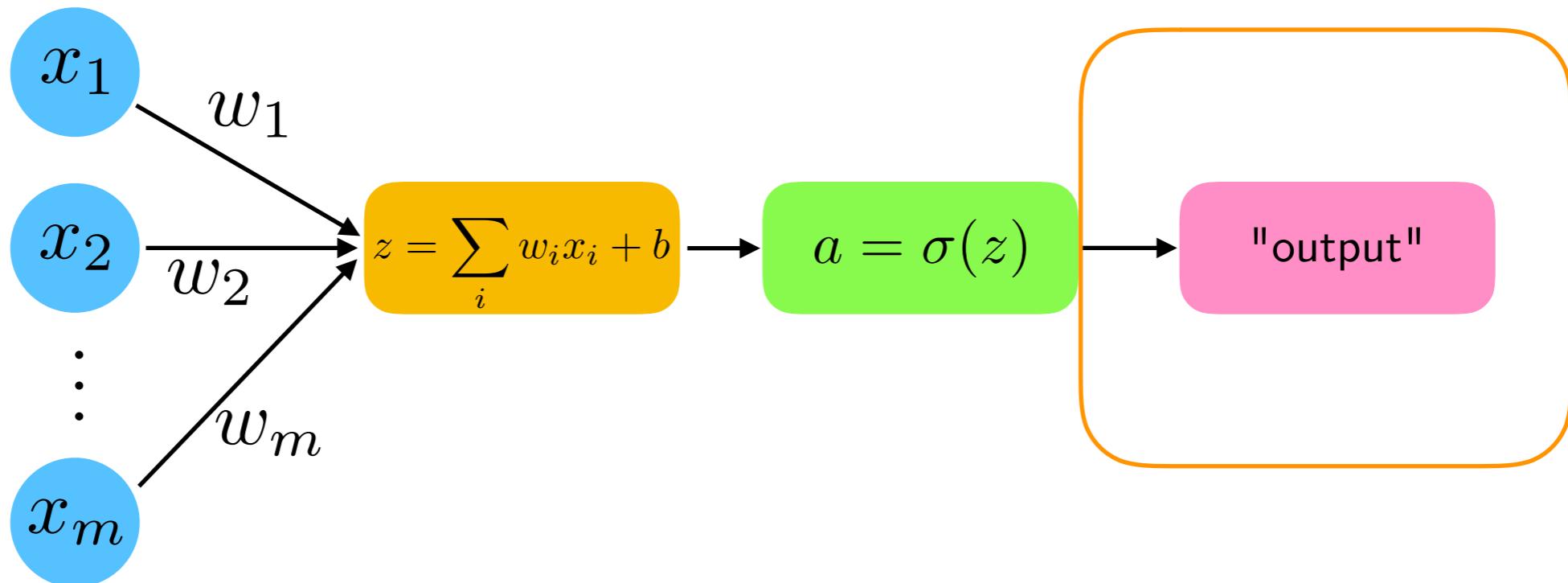
Δw := "negative gradient"

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

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

Update for each batch/minibatch
with learning rate

Class Label Predictions with Logistic Regression



In logistic regression, we can use

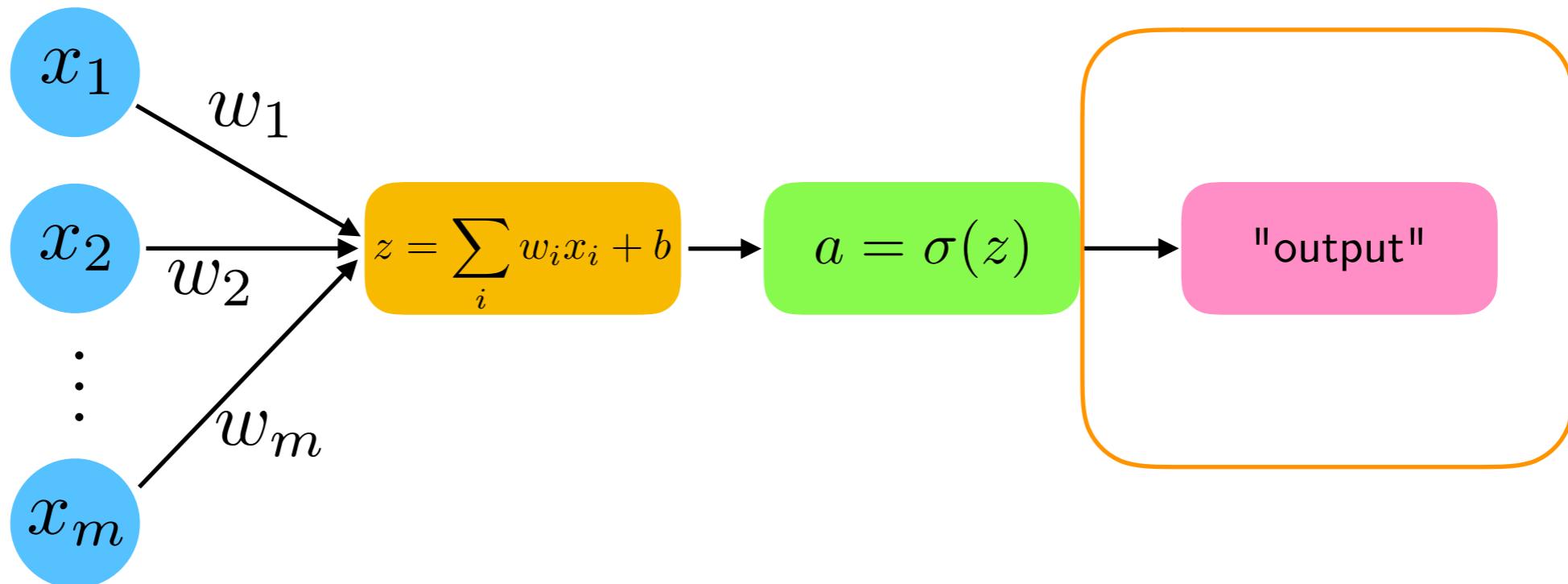
$$\hat{y} := \begin{cases} 1 & \text{if } \sigma(z) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

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

Class Label Predictions with Logistic Regression

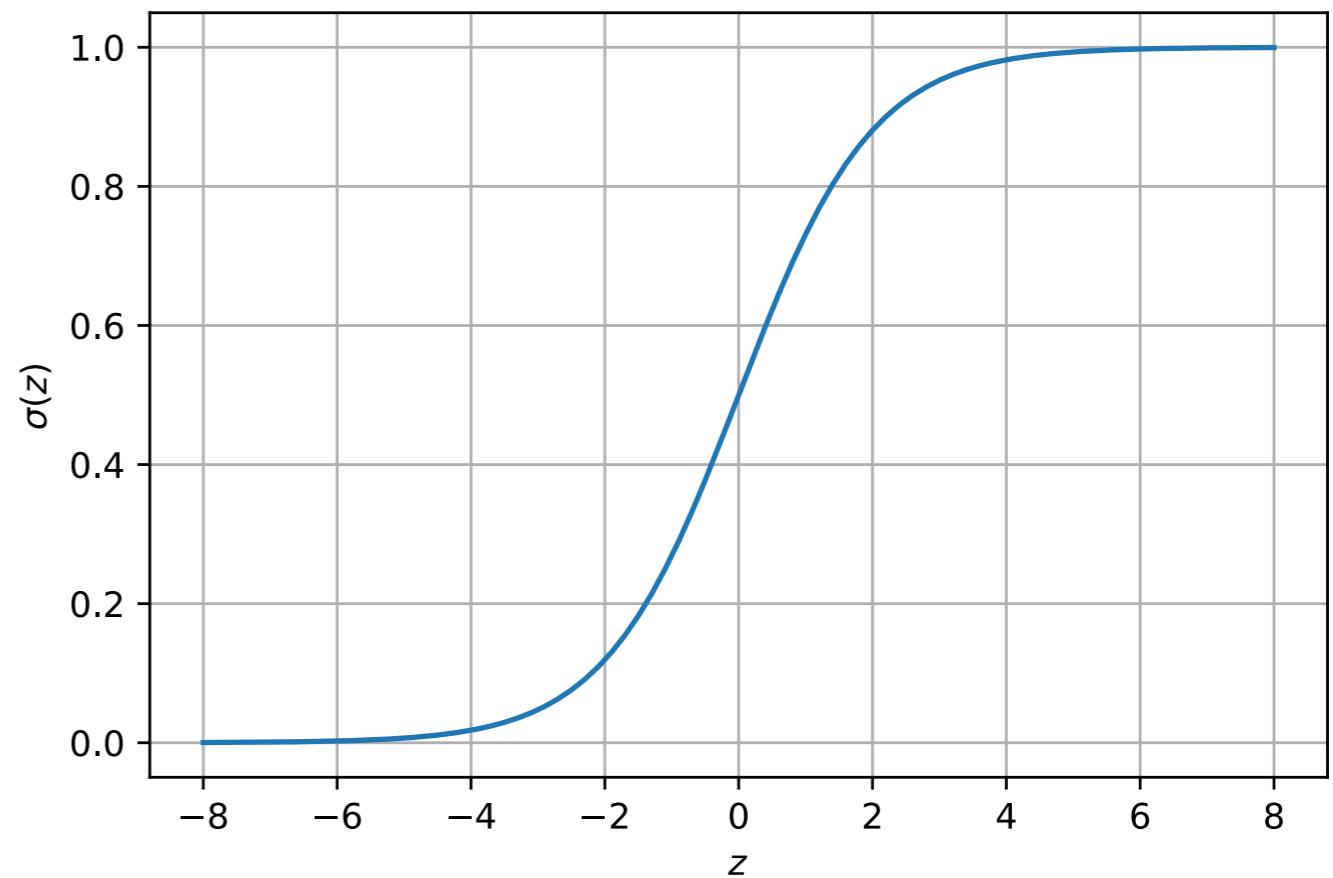


In logistic regression, we can use

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

which is the same as

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



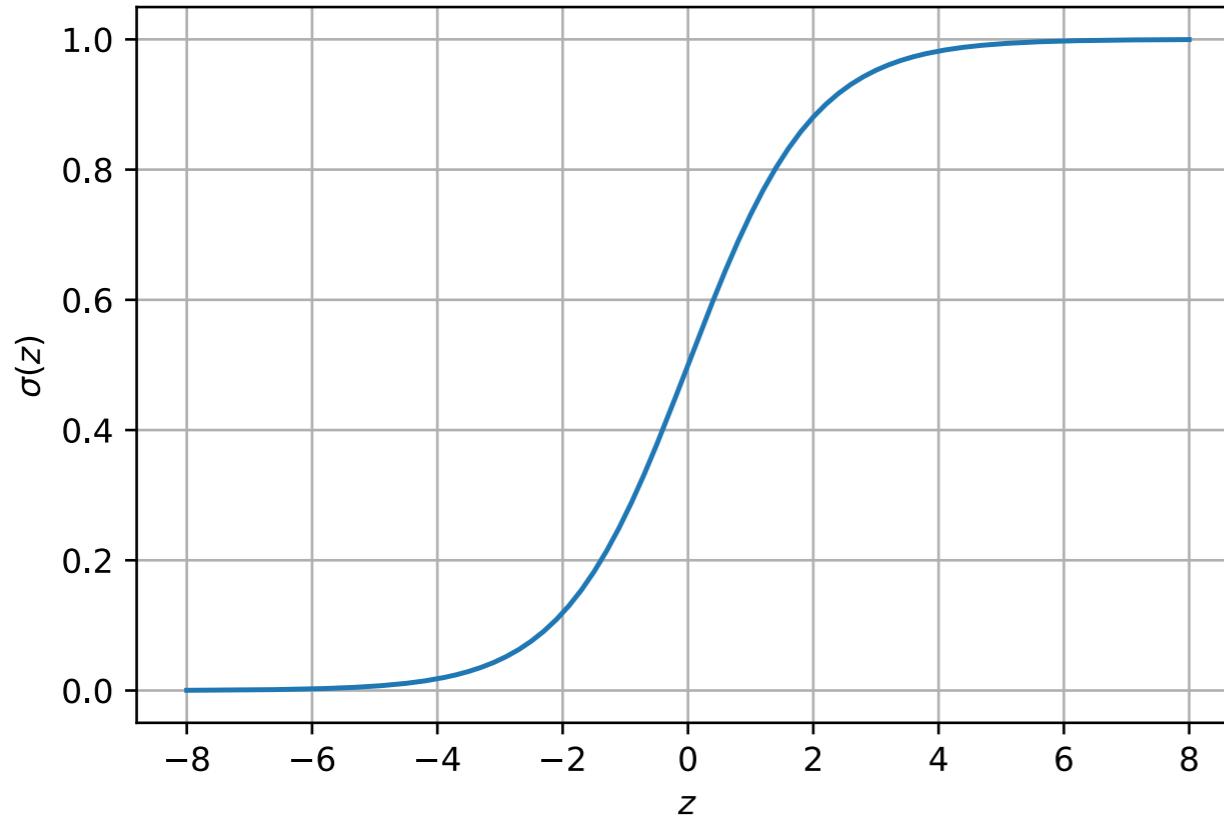
About the Term "Logits"

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

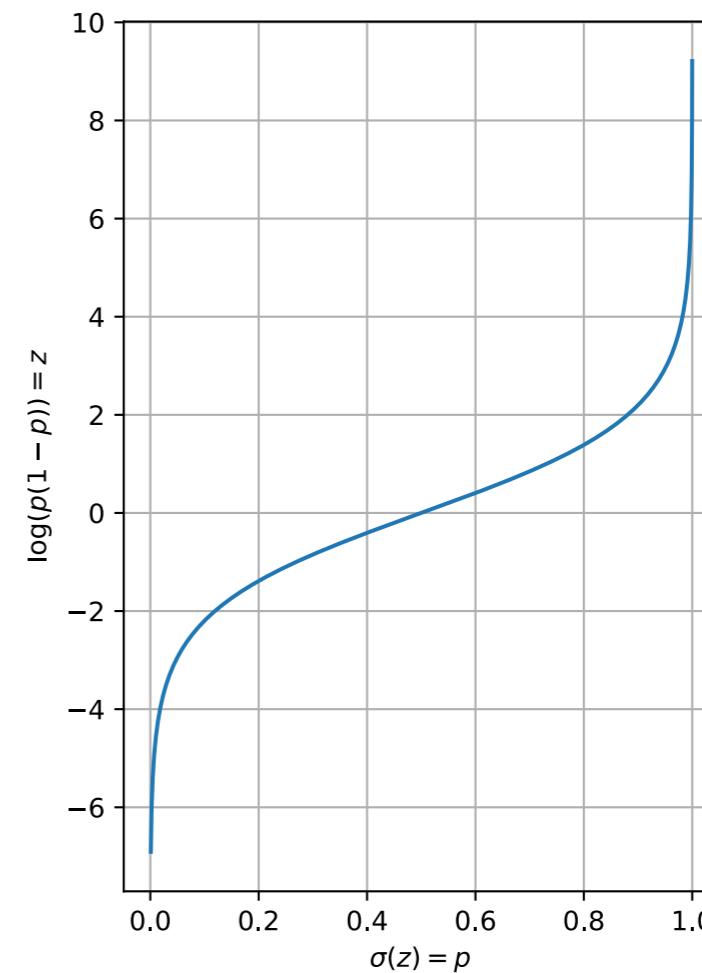
- In deep learning, the logits are the net inputs of the last neuron layer
- In statistics, we call the log-odds the logits
- In logistic regression, the logits are naturally $\mathbf{w}^\top \mathbf{x}$...
- ... because log-odds is just short for "logarithm of the odds": $\log(p/(1-p))$
- In other words,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

https://github.com/rasbt/stat479-machine-learning-fs19/blob/master/06_trees/06-trees__notes.pdf

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

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

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 entropy (not-cross entropy though) in the context of decision trees (but with \log_2 instead of the natural log)

https://github.com/rasbt/stat479-machine-learning-fs19/blob/master/06_trees/06-trees__notes.pdf

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

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

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

Logistic Regression Coding Example

[https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L07-logistic/
code/logistic-regression.ipynb](https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L07-logistic/code/logistic-regression.ipynb)

Logistic Regression for Multi-class Classification:

Multinomial Logistic Regression / Softmax Regression

Logistic Regression

Negative Log-Likelihood Loss

Logistic Regression Learning Rule

Softmax Regression

Softmax Loss Gradients

MNIST - 60k Handwritten Digits

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



Balanced dataset:

- 10 classes (digits 0-9)
- 10k 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

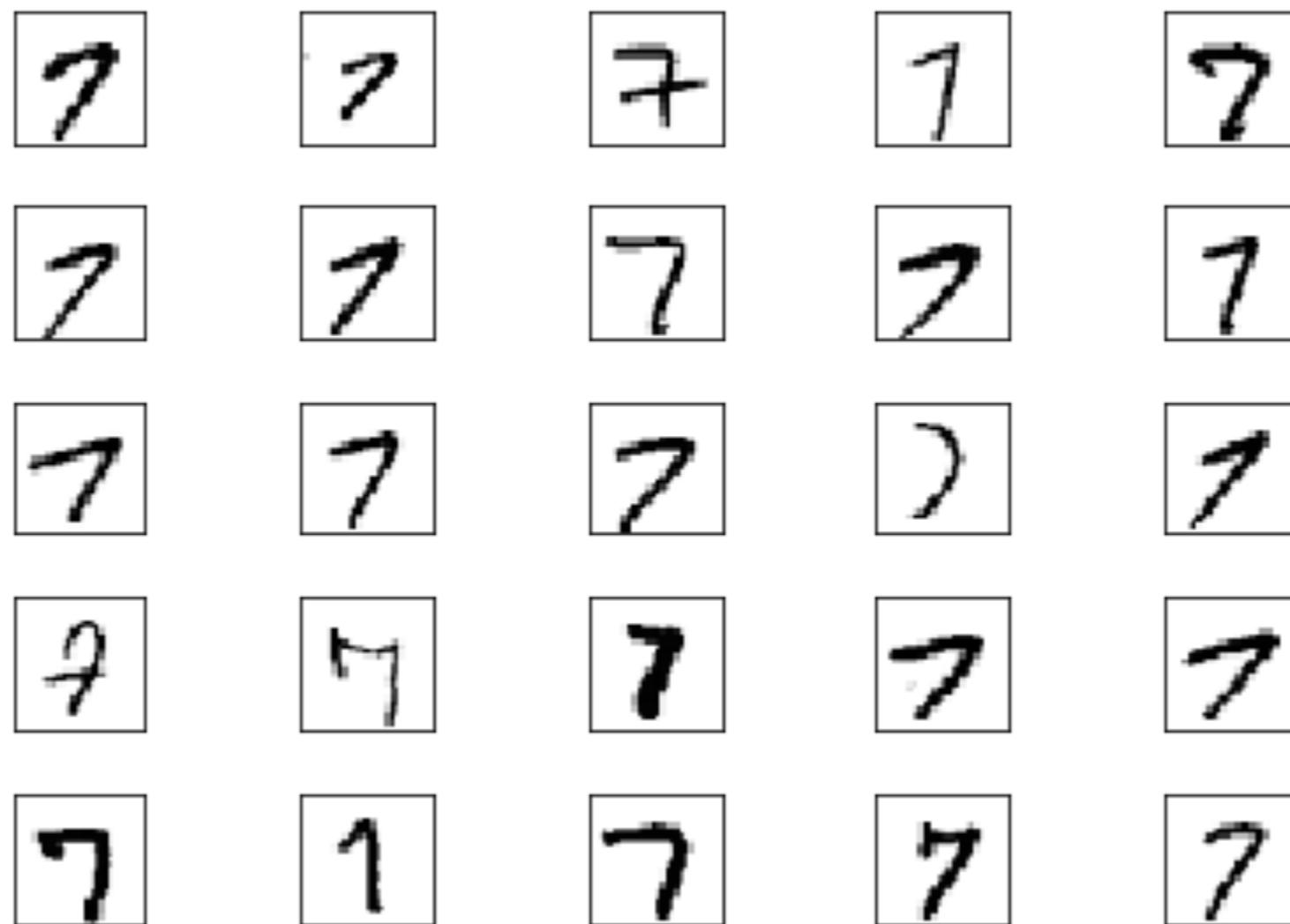


Illustration of different "7"s

Data Representation (unstructured data; images)

Convolutional Neural Networks (later)

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

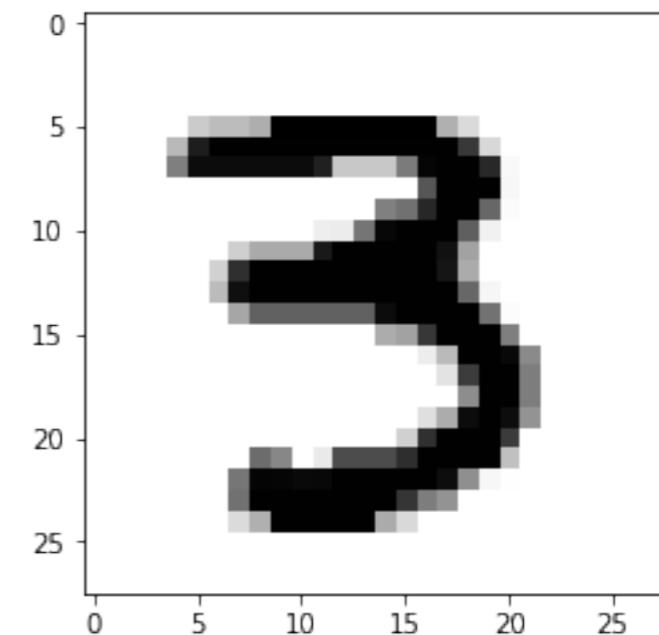
Image label dimensions: torch.Size([128])

```
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, 0.6375, 0.0549, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000]]]
```

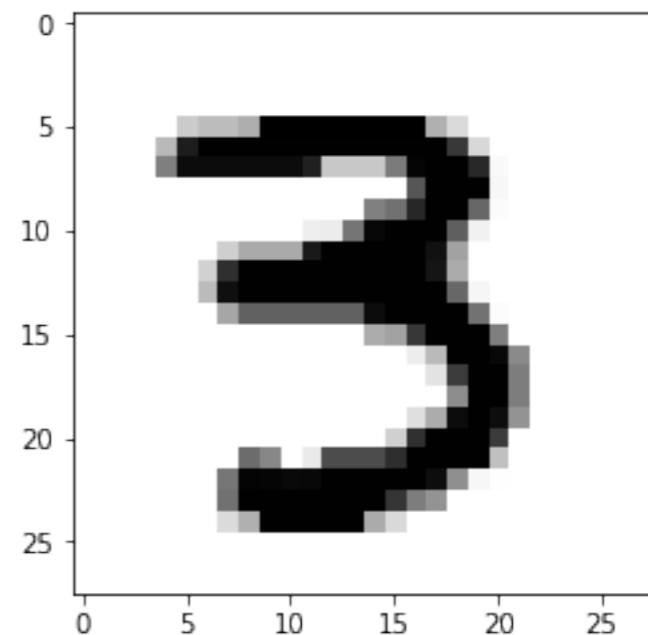


Note that I normalized pixels by factor 1/255 here

Data Representation (unstructured data; images)

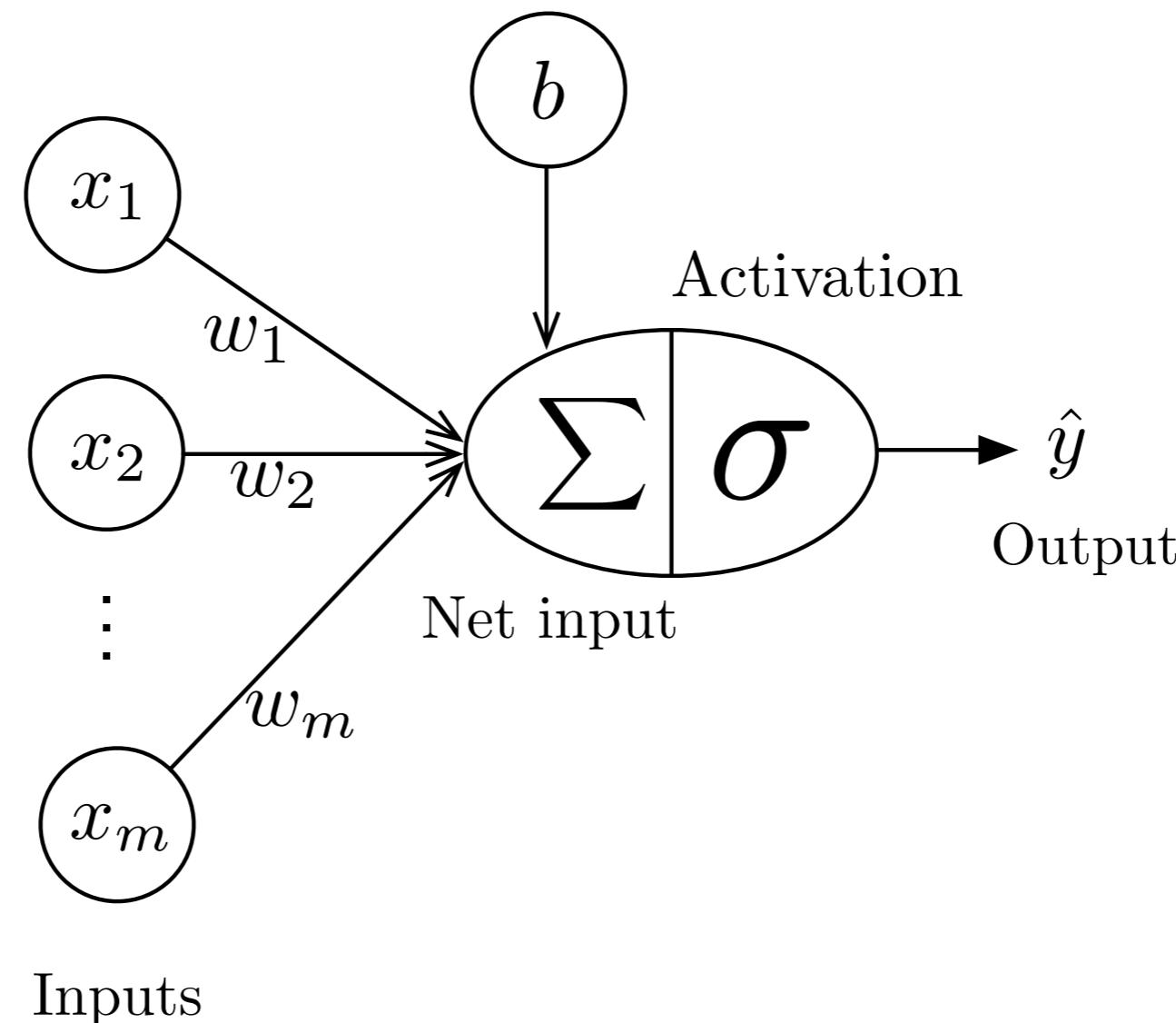
Softmax regression: "traditional method"

Represent digit as a long vector
of pixels



Note that I normalized pixels by factor 1/255 here

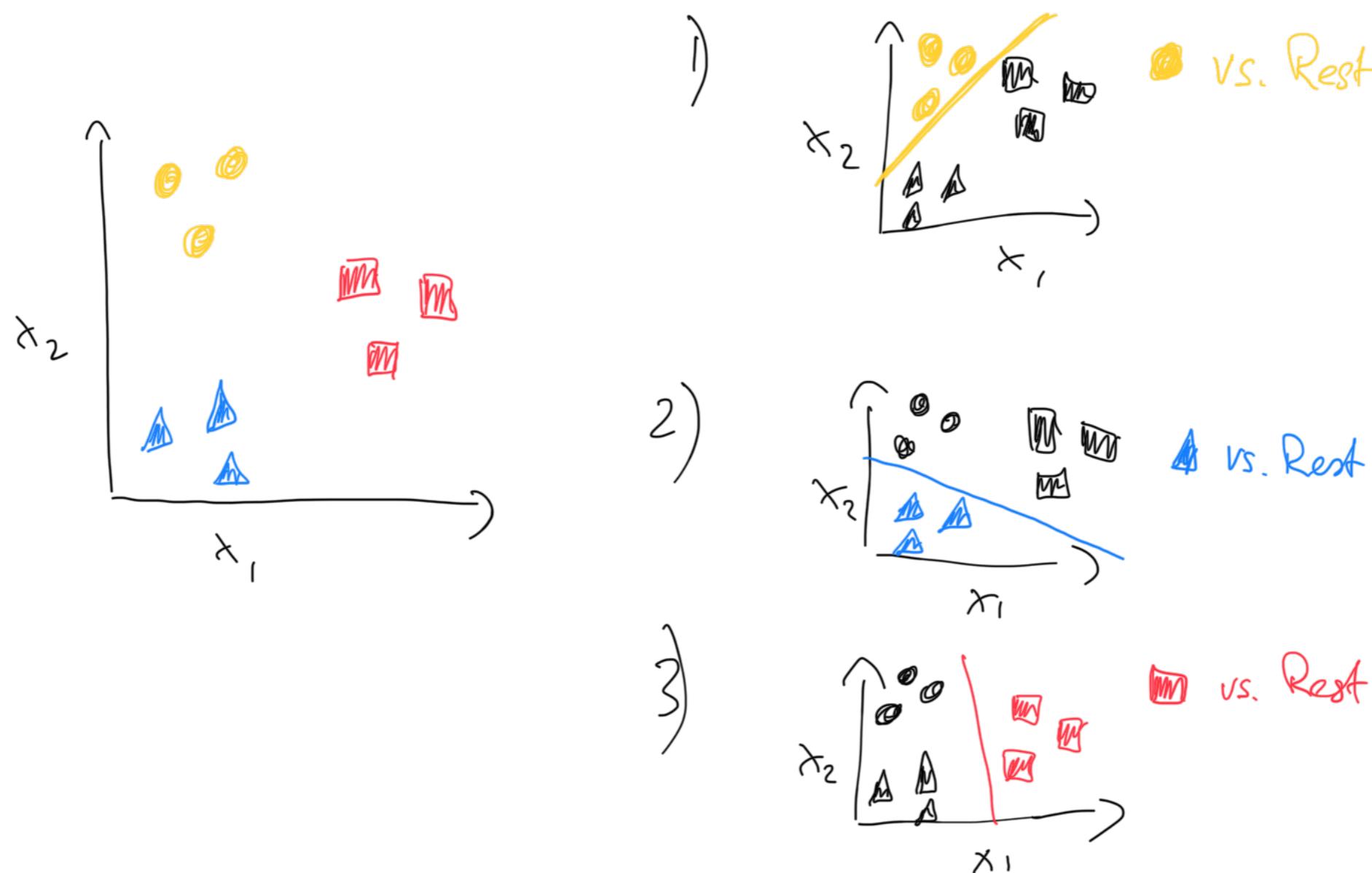
Previous Approach for Binary Classes



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

Multi-Class Classification with Multiple Binary Classifiers

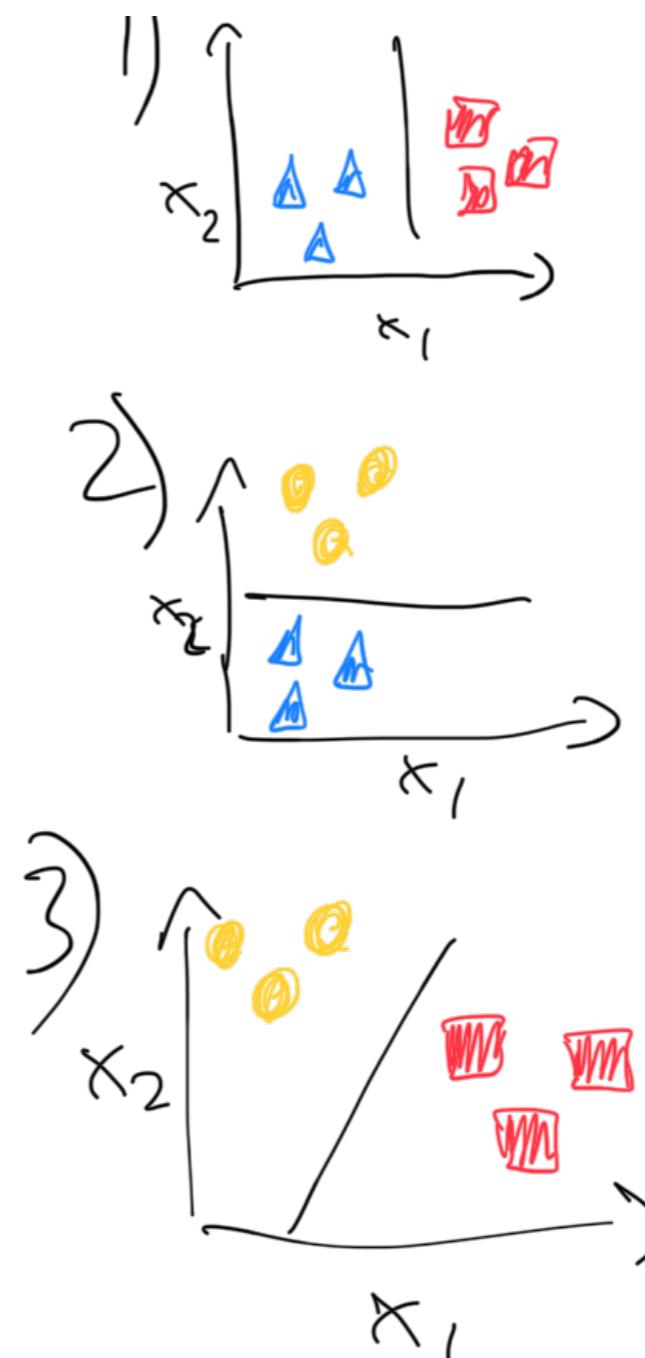
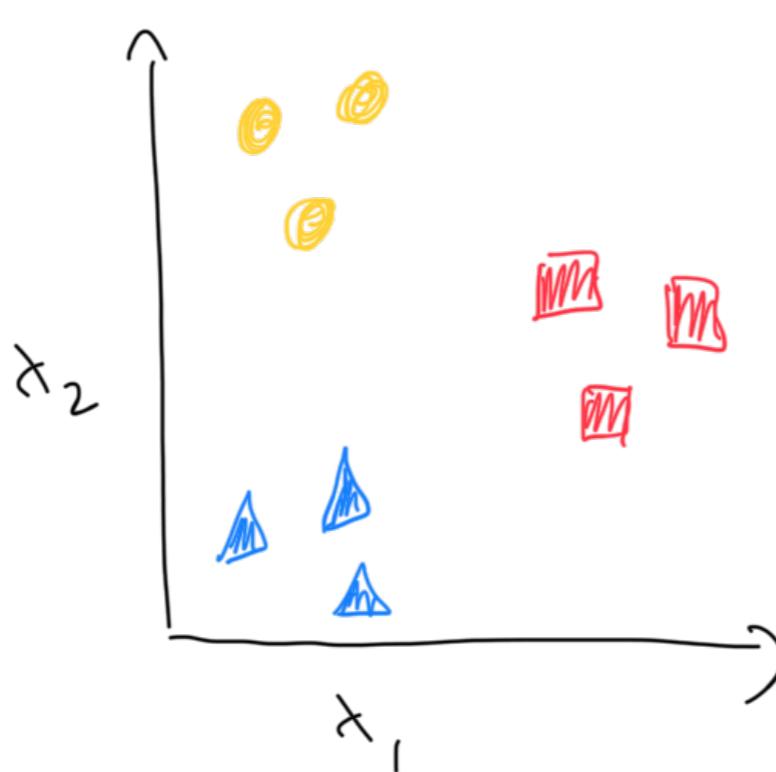
One-vs-Rest (OvR) / One-vs-All (OvA)



Then, choose the class with the highest confidence score

Multi-Class Classification with Multiple Binary Classifiers

One-vs-One (OvO) / All-vs-All (AvA)

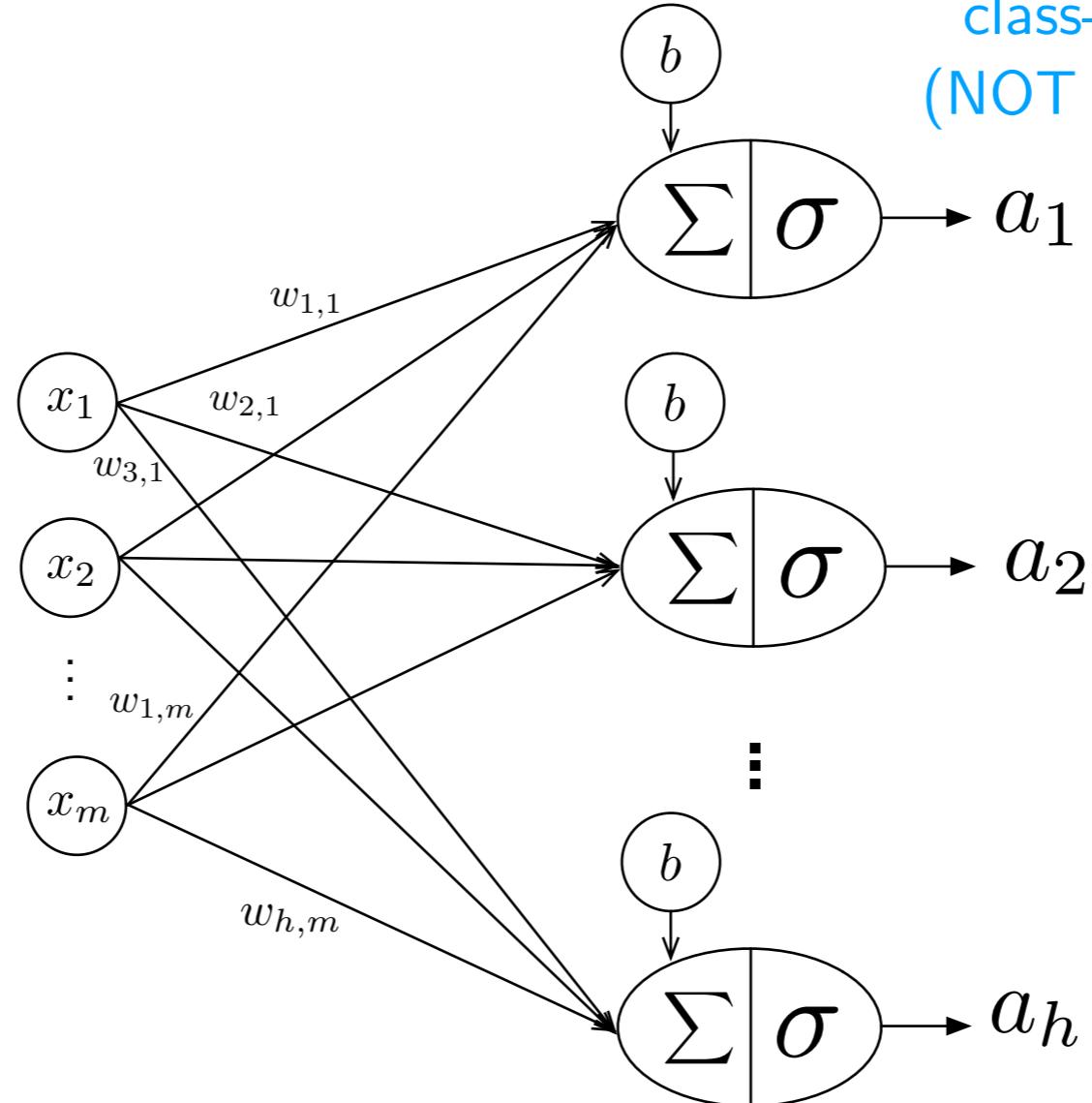


Number of classifiers needed:

$$\text{num_classes} \times (\text{num_classes} - 1) / 2$$

Then, select the class by majority vote (and use confidence score in case of ties)

Another Approach Could be ...



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

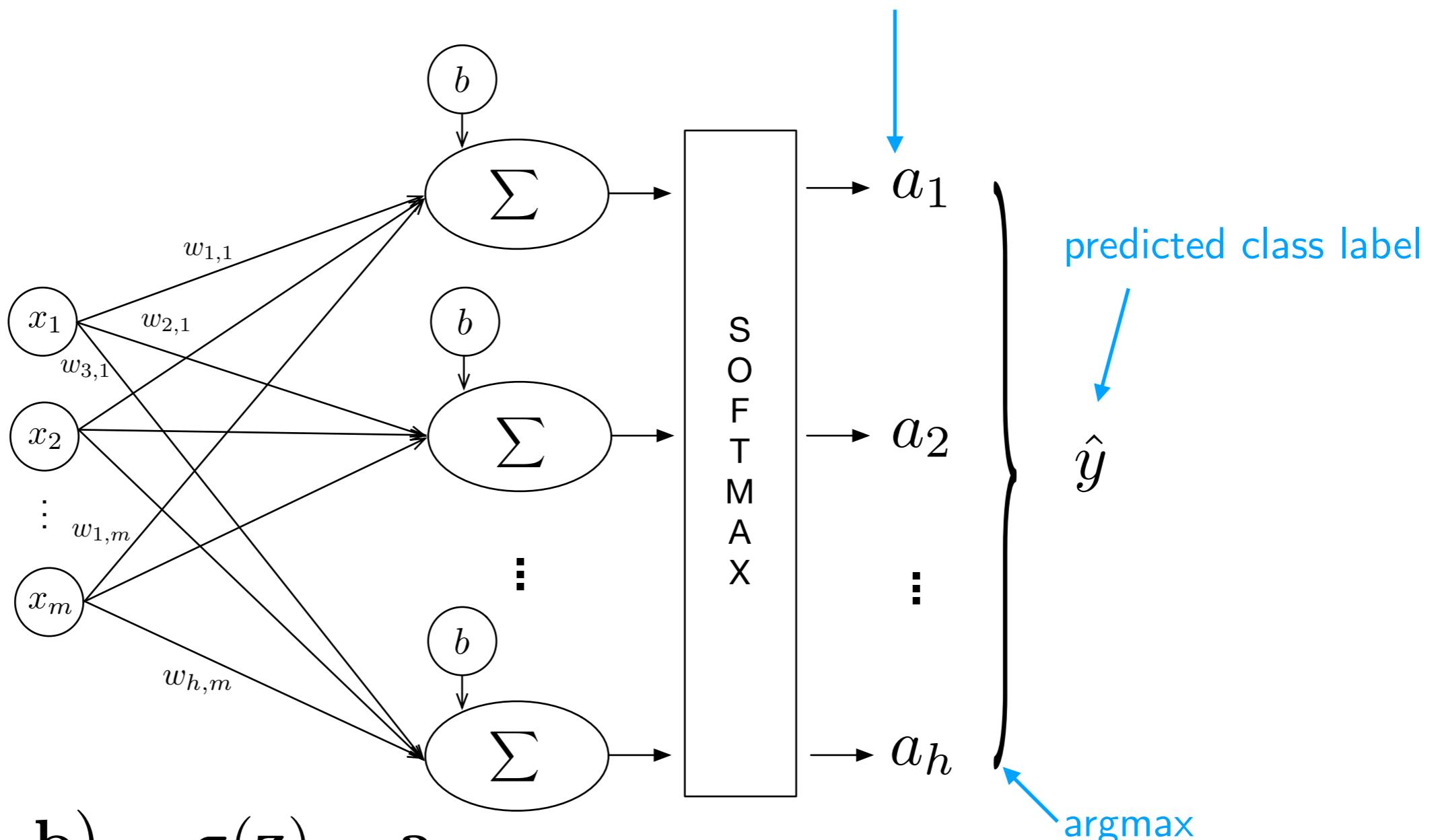
$$\sigma(\mathbf{Wx} + \mathbf{b}) = \sigma(\mathbf{z}) = \mathbf{a}$$

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

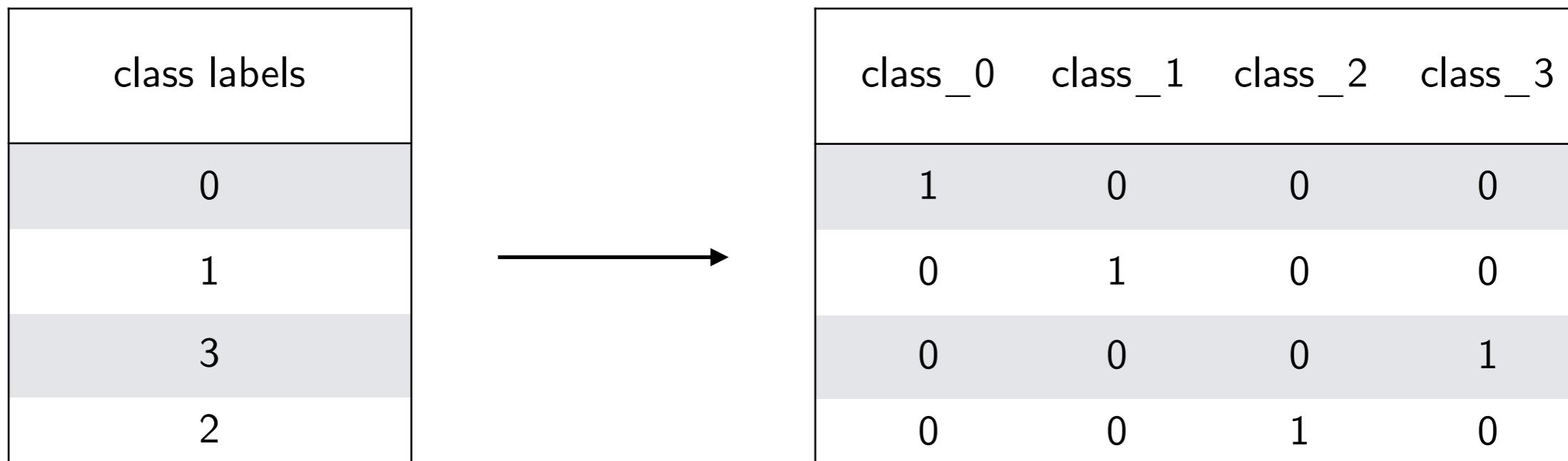
where k is the number of classes

Multinomial Logistic Regression / Softmax Regression

activations are
class-membership probabilities
(mutually exclusive classes)



Onehot Encoding Needed



Softmax Activation

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

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

k is the number of class labels



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

Loss Function

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

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

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!

$$\mathcal{L}_{\text{multiclass}} = \sum_{i=1}^n \sum_{j=1}^k -y_j^{[i]} \log \left(a_k^{[i]} \right)$$

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

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)

$$\mathcal{L}_{\text{multiclass}} = \sum_{i=1}^n \sum_{j=1}^k -y_j^{[i]} \log(a_k^{[i]})$$

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

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

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

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

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

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

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

$$\mathcal{L}_{\text{multiclass}} = \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^k -y_j^{[i]} \log(a_k^{[i]})$$

$$\approx 0.9335$$

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-ss20/blob/master/L07-logistic/code/cross-entropy-pytorch.ipynb>

Training Softmax Regression

Logistic Regression

Negative Log-Likelihood Loss

Logistic Regression Learning Rule

Softmax Regression

Softmax Loss Gradients

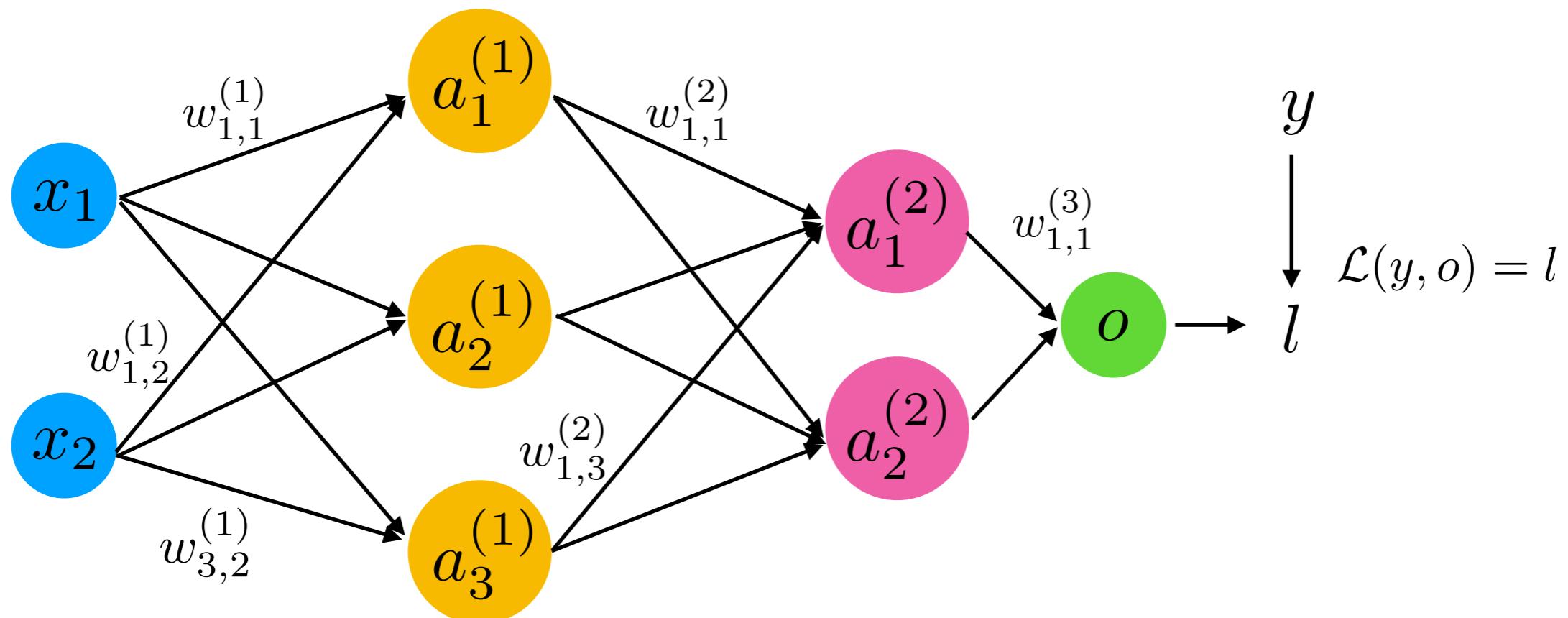
The Same Overall Concept Applies ...

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

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

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

Graph with Fully-Connected Layers (later in this course)

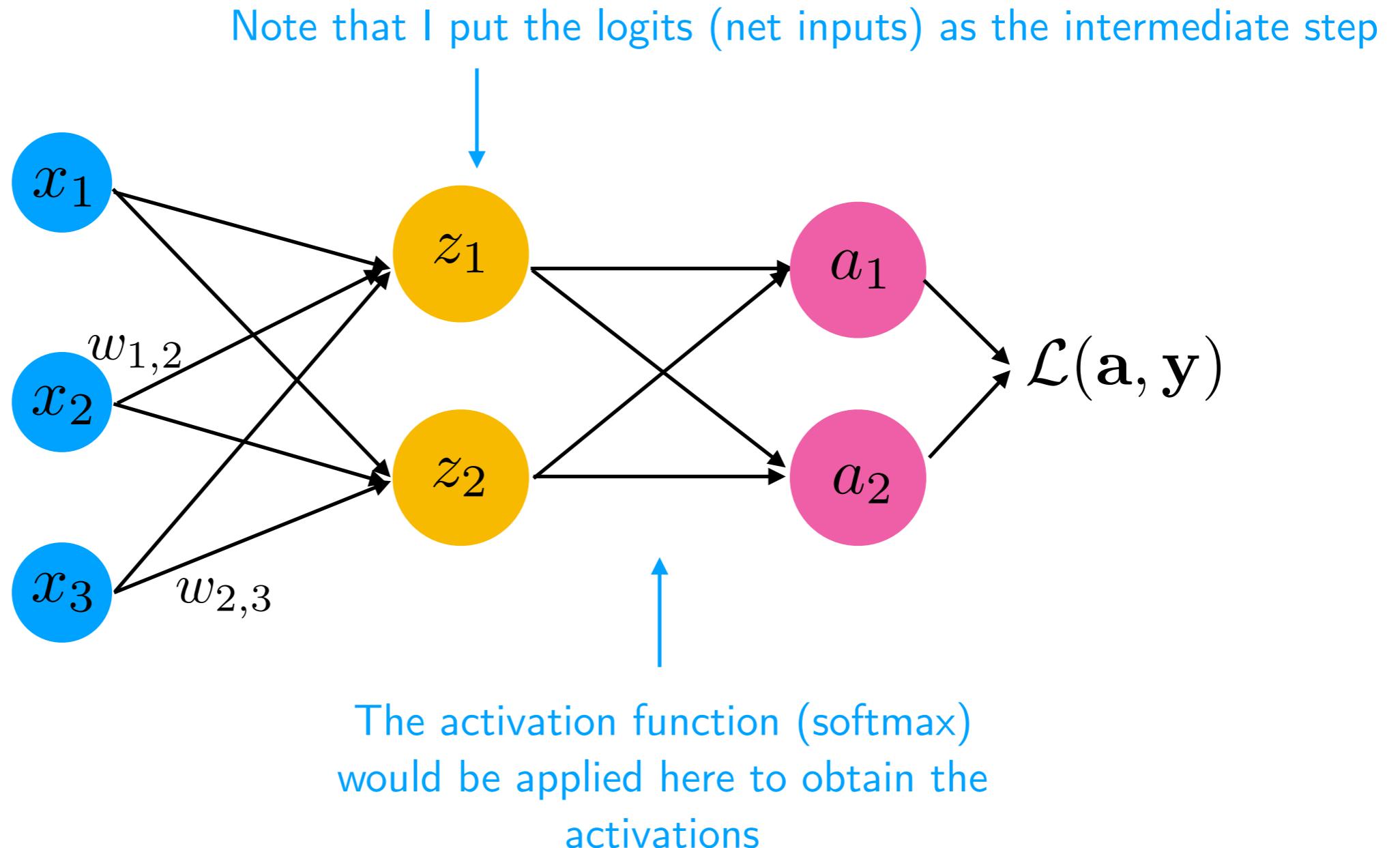


$$\begin{aligned}\frac{\partial l}{\partial w_{1,1}^{(1)}} &= \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial w_{1,1}^{(1)}} \\ &\quad + \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_2^{(2)}} \cdot \frac{\partial a_2^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial w_{1,1}^{(1)}}\end{aligned}$$

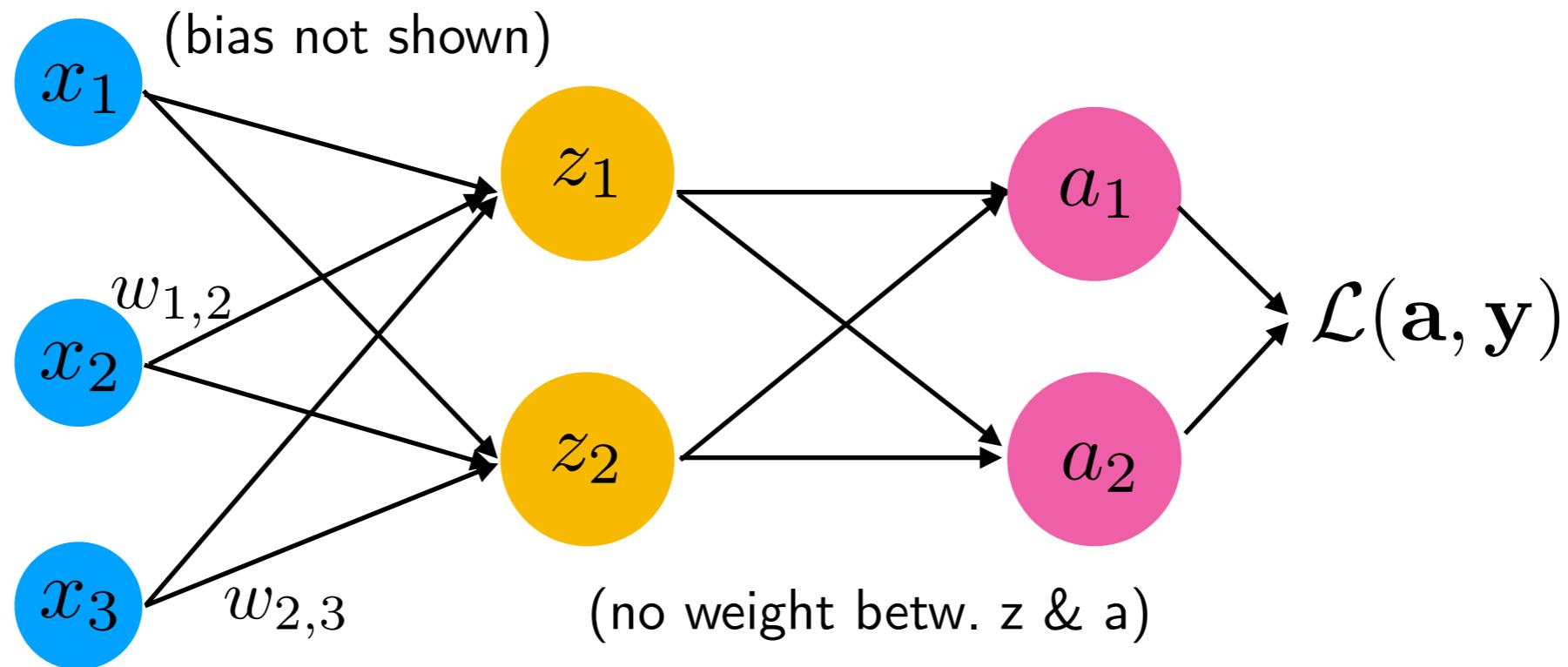
Remember when I introduced the multivariable chain rule earlier in this course? Now, we need it!

Softmax Regression Sketch

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



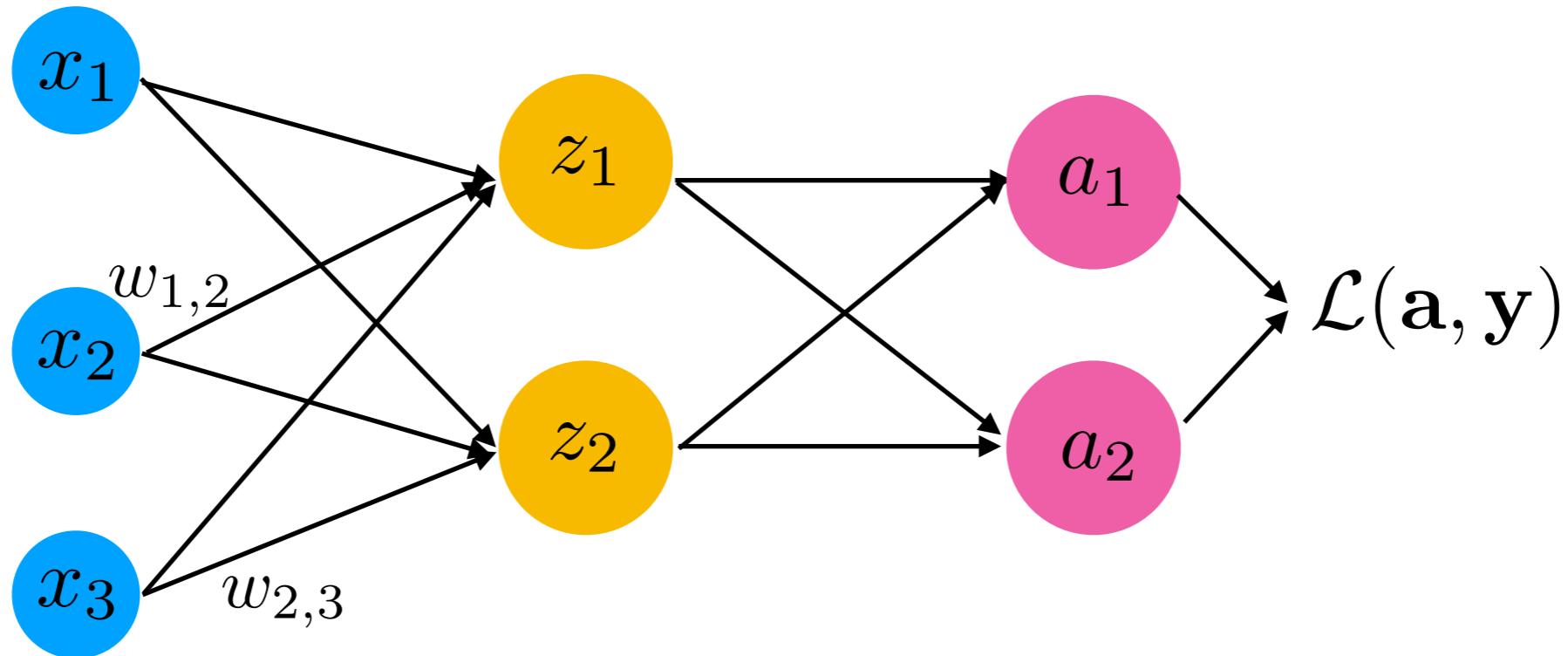
Softmax Regression Sketch



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



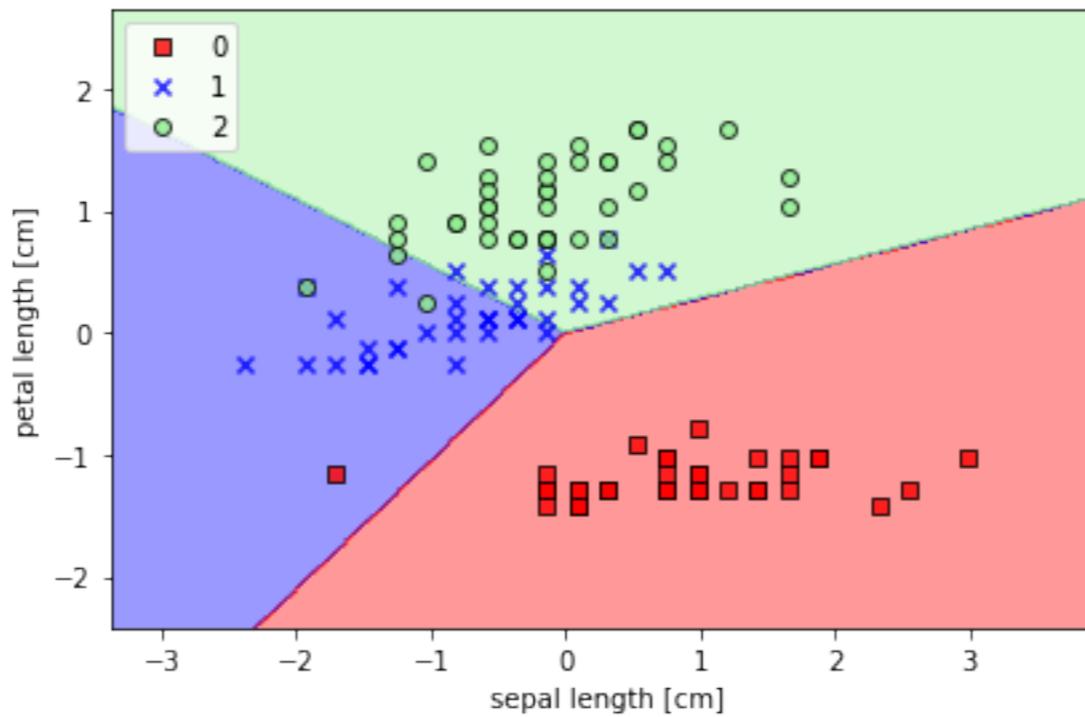
Vectorized Form:

$$\begin{aligned}
 \frac{\partial L}{\partial w_{1,2}} &= \frac{\partial L}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial w_{1,2}} + \frac{\partial L}{\partial a_2} \frac{\partial a_2}{\partial z_1} \frac{\partial z_1}{\partial w_{1,2}} \\
 &= \frac{-y_1}{a_1} [a_1(1 - a_1)]x_2 + \frac{-y_2}{a_2} (-a_2a_1)x_2 \\
 &= (y_2a_1 - y_1 + y_1a_1)x_2 \\
 &= (a_1(y_1 + y_2) - y_1)x_2 \\
 &= -(y_1 - a_1)x_2
 \end{aligned}$$

$$\nabla_{\mathbf{W}} \mathcal{L} = -(\mathbf{X}^\top (\mathbf{Y} - \mathbf{A}))^\top$$

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 Hands-On Example (Iris)

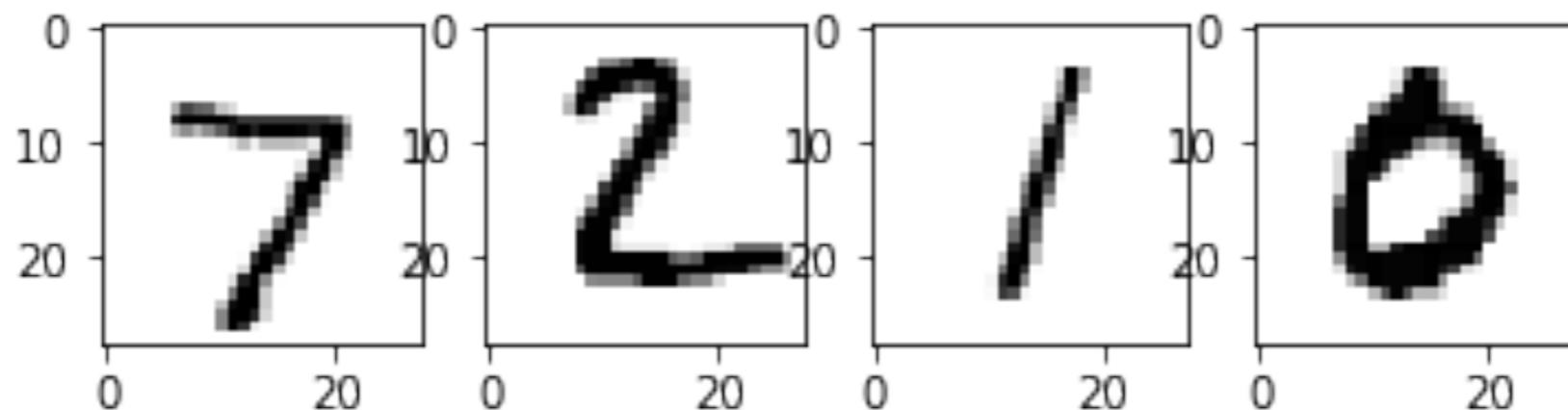


"From scratch" implementation & PyTorch autodiff implementation

https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L07-logistic/code/softmax-regression_scratch.ipynb

Softmax Regression Hands-On (MNIST)

Using PyTorch (with autodiff an nn.Module)



<https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L07-logistic/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

Additional Reading Material

Softmax regression tutorial:

<https://mccormickml.com/2014/06/13/deep-learning-tutorial-softmax-regression/>