STAT 453: Introduction to Deep Learning and Generative Models

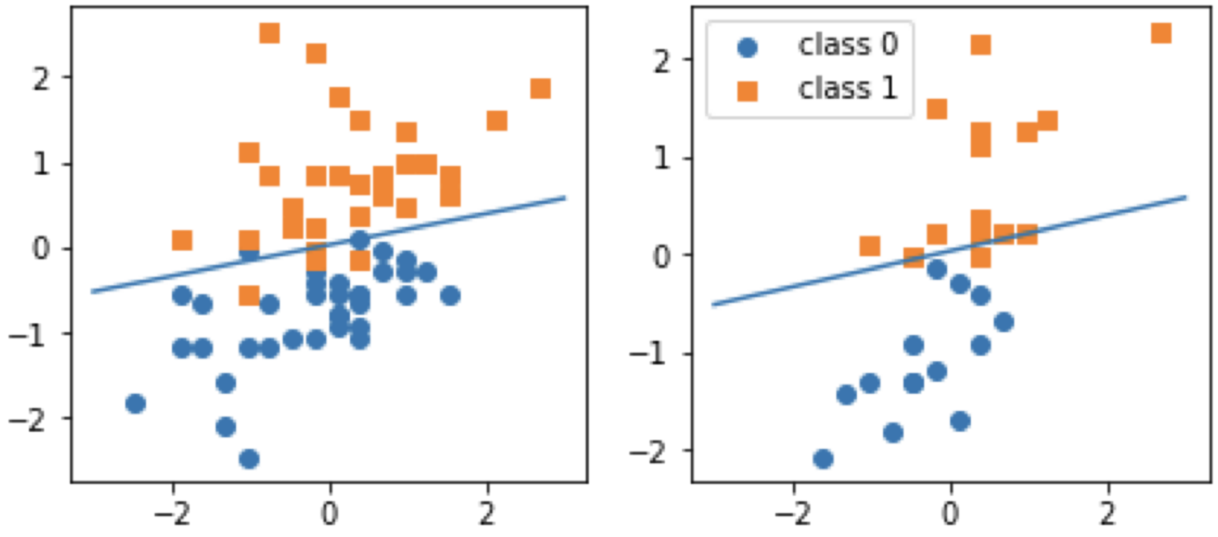
Ben Lengerich

Lecture 08: (Multinomial) Logistic Regression

September 29, 2025

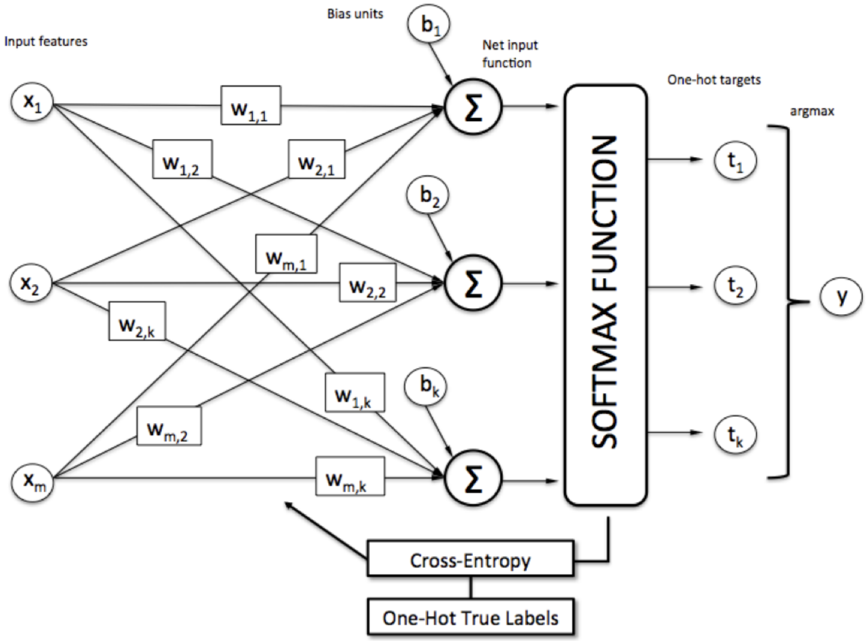
Recall

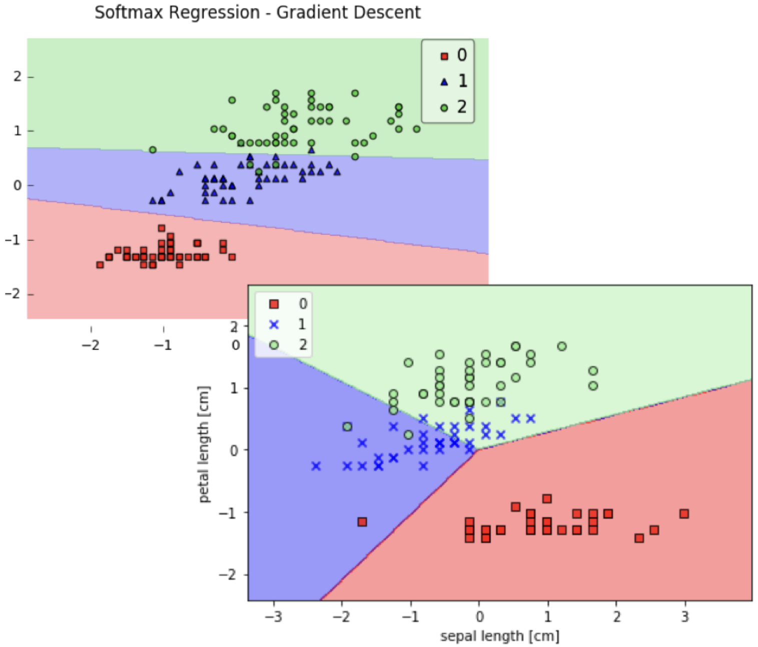
1. Perceptron learning algorithm → gradient descent as a general algorithm
2. Conceptualized gradient descent via computation graphs
3. How to write code in PyTorch to train basic neural nets



Today: Our old friend logistic regression…

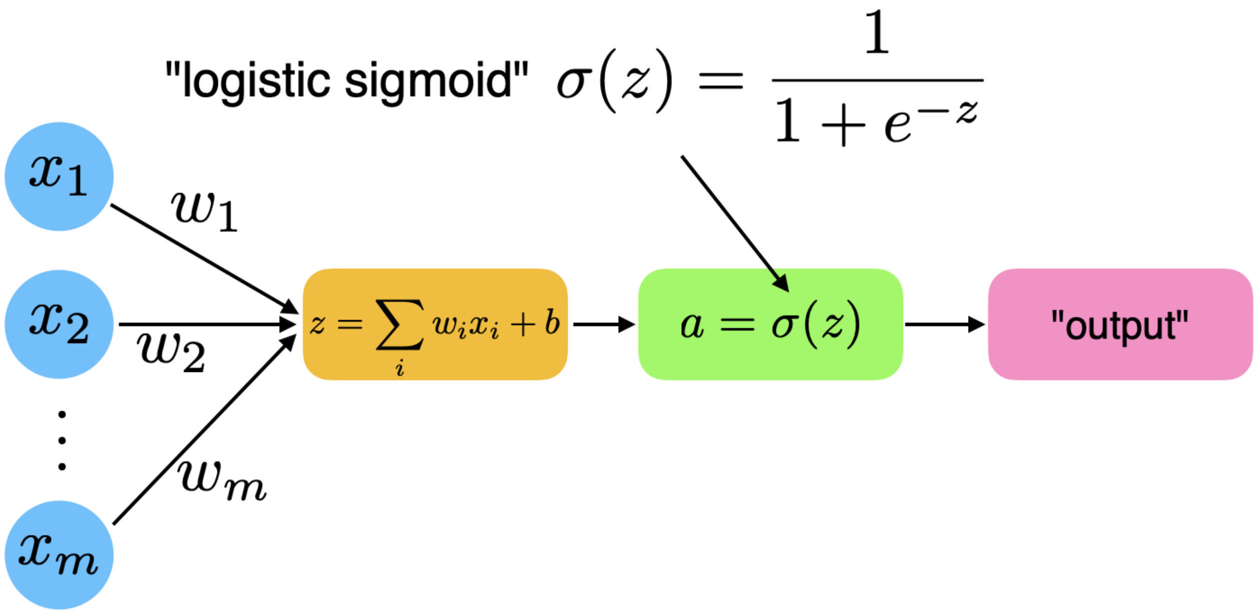
1. A better loss function for classification (cross entropy instead of MSE)
2. Extending neurons to multi-classification (multiple output nodes + softmax)





Logistic Regression Neuron

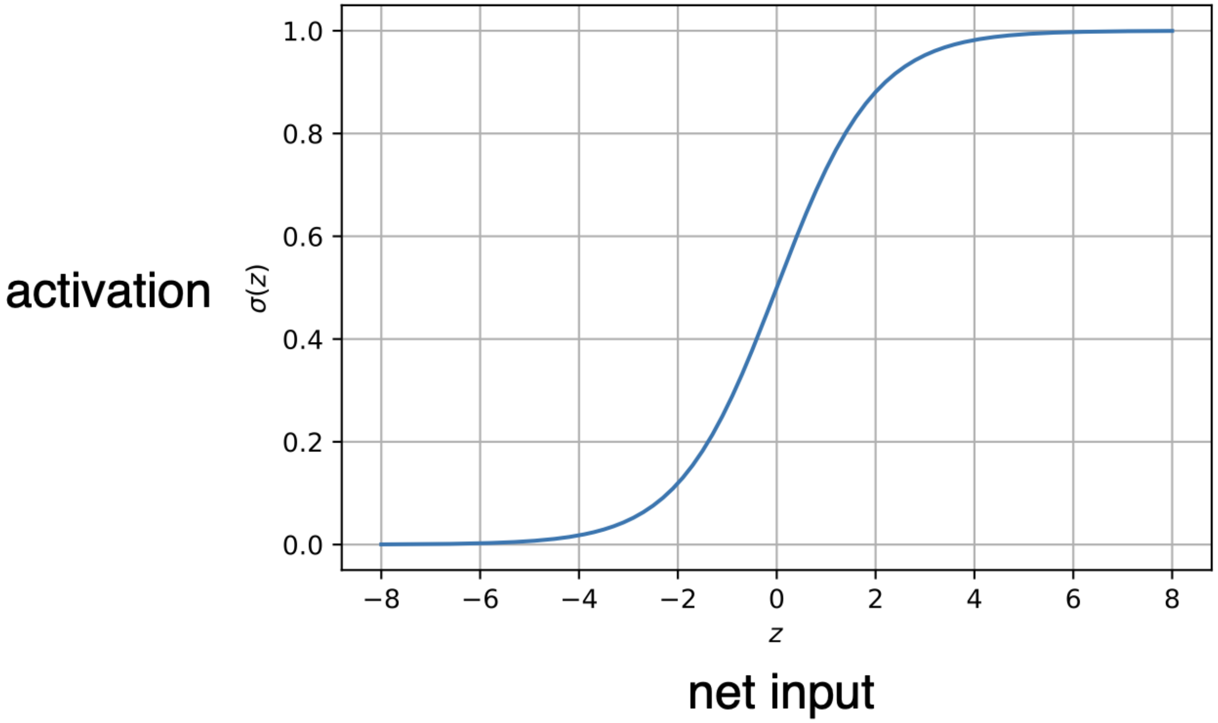
- For binary classes $y \in \{0, 1\}$



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

The building block: Logistic Sigmoid Function

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



Logistic Regression

- Given the output:

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

- We compute the probability as

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

Can we write this more compactly?

Today: Our old friend logistic 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. One-Hot Encoding and Multi-category Cross-Entropy
8. Softmax Regression Learning Rule
9. Softmax Regression Code Example

Logistic Regression

- Given the output:

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

- We compute the probability as

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

$$\downarrow$$

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

Recall Bernoulli distribution…

Logistic Regression: Estimation

- Given the probability:

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

- Under MLE estimation, we would like to maximize the multi-sample likelihood:

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

Logistic Regression: Estimation

- Given the probability:

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

- Under MLE estimation, we would like to maximize the multi-sample likelihood:

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

Suppose this were linear regression: $h(x) = \boldsymbol{w}^T\boldsymbol{x} + b$

$$L(\boldsymbol{w}, b; \boldsymbol{X}, \boldsymbol{y}) = \prod_i N(\, y^{[i]} \mid h(x^{[i]})\,)$$

$$L(\boldsymbol{w}, b; \boldsymbol{X}, \boldsymbol{y}) \propto -\prod_i \left(y^{[i]} - h(x^{[i]})\right)^2$$

$$\ell(\boldsymbol{w}, b; \boldsymbol{X}, \boldsymbol{y}) \propto -\sum_i \left(y^{[i]} - h(x^{[i]})\right)^2$$

$$\Rightarrow \quad \hat{\boldsymbol{w}}_{MLE}, \hat{b}_{MLE} = \operatorname{argmin} \sum_i \left(y^{[i]} - h(x^{[i]})\right)^2$$

Logistic Regression: Estimation

- Given the probability:

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

- Under MLE estimation, we would like to maximize the multi-sample likelihood:

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

$$= \prod_{i=1}^{n} \left(\sigma\left(z^{(i)}\right)\right)^{y^{(i)}} \left(1 - \sigma\left(z^{(i)}\right)\right)^{1-y^{(i)}}$$

Likelihood

Logistic Regression: Estimation

$$P\left(y^{[i]}, \ldots, y^{[n]} \middle| \mathbf{x}^{[1]}, \ldots, \mathbf{x}^{[n]}\right) = \underbrace{\prod_{i=1}^{n} \left(\sigma\left(z^{(i)}\right)\right)^{y^{(i)}} \left(1 - \sigma\left(z^{(i)}\right)\right)^{1-y^{(i)}}}_{\text{Likelihood}}$$

- We are going to optimize via gradient descent, so let's apply the logarithm to separate components:

$$l(\mathbf{w}) = \log L(\mathbf{w})$$

$$= \underbrace{\sum_{i=1}^{n} \left[y^{(i)} \log\left(\sigma\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - \sigma\left(z^{(i)}\right)\right)\right]}_{\text{Log-Likelihood}}$$

Negative Log-Likelihood (NLL) Loss

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

Log-Likelihood

- In practice, we often **minimize negative log-likelihood** instead of **maximizing log-likelihood**:

$$\widehat{\boldsymbol{w}} = \text{argmin} - l(\boldsymbol{w})$$

Log-Likelihood

Today: Our old friend logistic 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. One-Hot Encoding and Multi-category Cross-Entropy
8. Softmax Regression Learning Rule
9. Softmax Regression Code Example

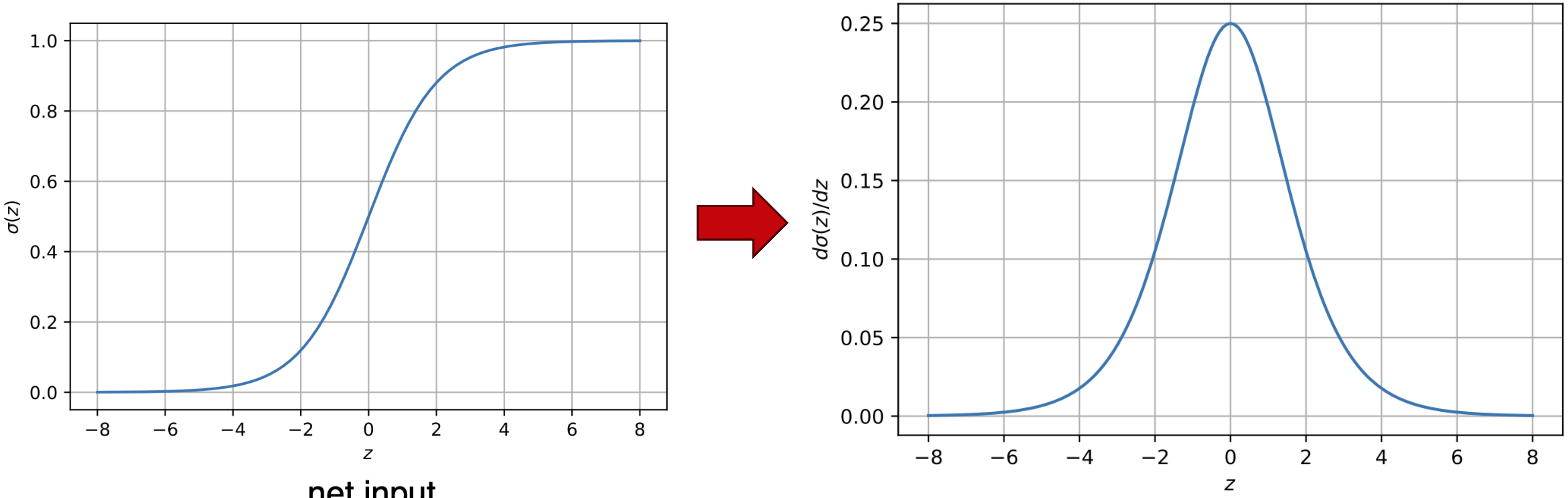
The building block: Logistic Sigmoid Function

A nice property: Derivatives of the sigmoid function are nice to us

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

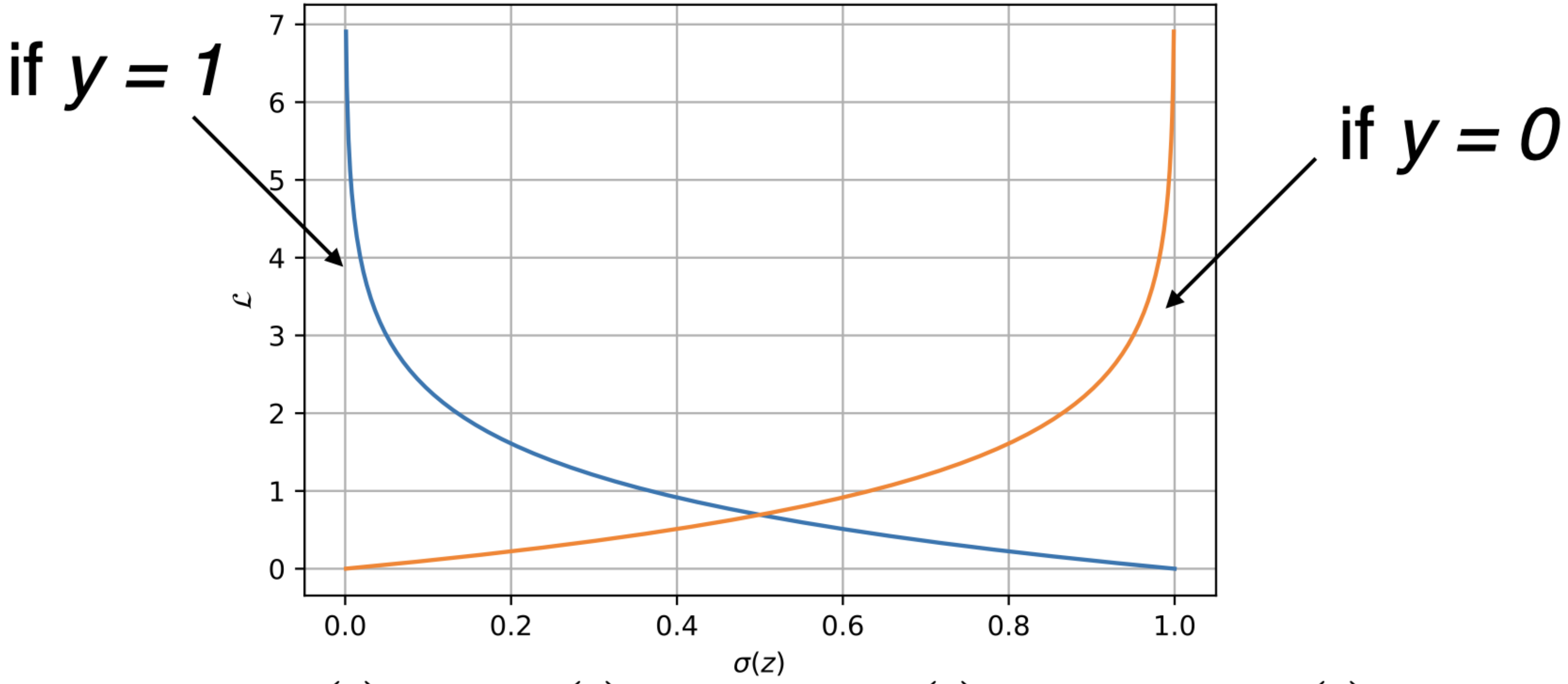
$$\frac{d}{dz}\sigma(z) = \frac{e^{-z}}{(1+e^{-z})^2} = \sigma(z)(1-\sigma(z))$$

activation



net input

Logistic Regression: Loss for a Single Training Example



$$\mathcal{L}(\mathbf{w}) = -\left(y^{(i)} \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)\right)$$

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

Logistic Regression: Learning Rule

Same gradient descent rule as before:

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

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

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

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

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

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

Logistic Regression: Learning Rule

Stochastic gradient descent:

1. Initialize $\mathbf{w} := \mathbf{0} \in \mathbb{R}^m,\ \mathbf{b} := 0$
2. For every training epoch:
   A. For every $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$

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

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

      $\nabla_{b}\mathcal{L} = -\left(y^{[i]} - \hat{y}^{[i]}\right)$

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

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

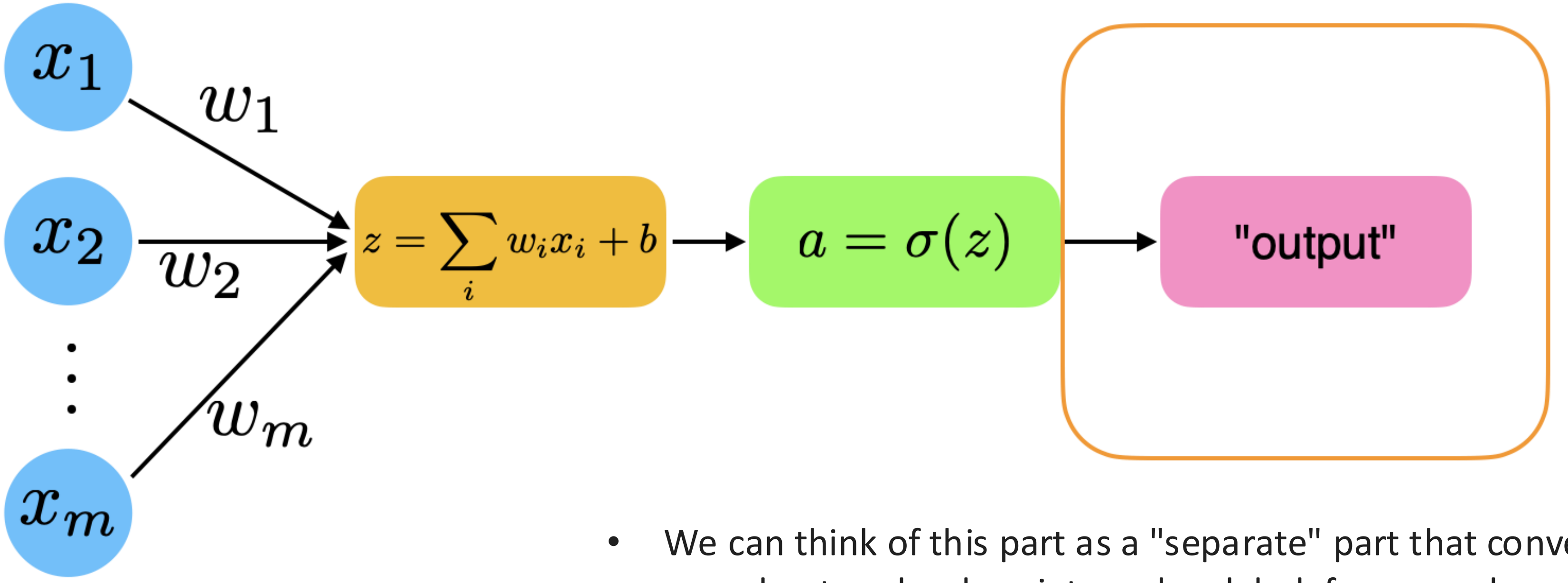
η: learning rate

$(-\nabla_{b}\mathcal{L})$: negative gradient

Note

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

Logistic Regression: Predicting Labels vs Probabilities



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 directly for optimization as a threshold function is not smooth

Today: Our old friend logistic 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. One-Hot Encoding and Multi-category Cross-Entropy
8. Softmax Regression Learning Rule
9. Softmax Regression Code Example

About the term "Logits"

- "Logits" = "log-odds unit": $\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$
- "Logits" is very common DL jargon
  - Typically means the <u>net input of the last neuron layer</u>
- In logistic regression, the "logits" are: $\boldsymbol{w}^T\boldsymbol{x}$

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" perspective

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

Binary Cross Entropy

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

$$H_{\mathbf{a}}(\mathbf{y}) = \sum_{i=1}^{n} \sum_{k=1}^{K} -y_k^{[i]} \log \left(a_k^{[i]} \right)$$

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

Today: Our old friend logistic 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. One-Hot Encoding and Multi-category Cross-Entropy
8. Softmax Regression Learning Rule
9. Softmax Regression Code Example

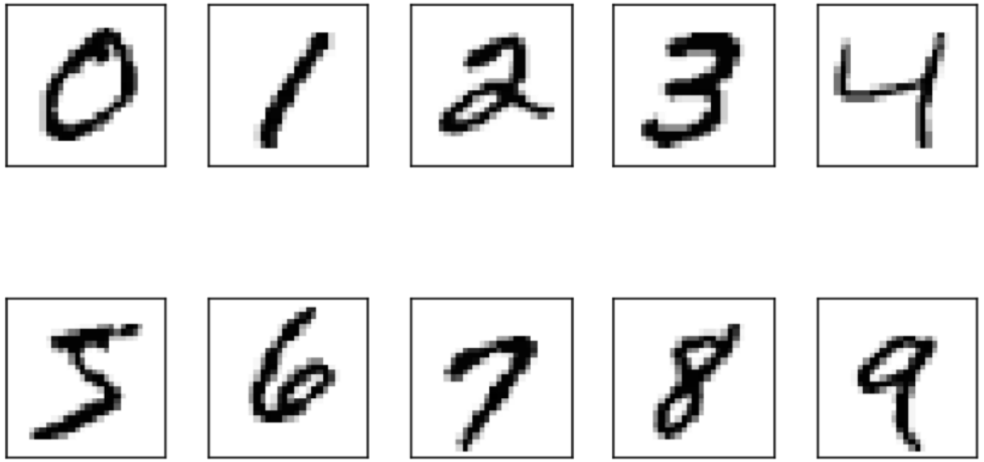
Logistic regression coding example

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

Today: Our old friend logistic 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. One-Hot Encoding and Multi-category Cross-Entropy
8. Softmax Regression Learning Rule
9. Softmax Regression Code Example

Example: MNIST Image Dataset



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)

Example: MNIST Image Dataset

```
Image batch dimensions: torch.Size([128, 1, 28, 28])
Image label dimensions: torch.Size([128])
```

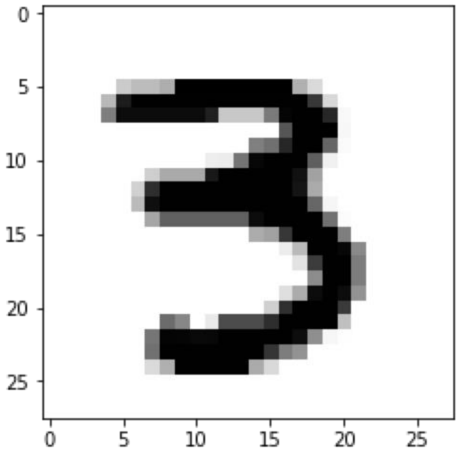
"NCHW" representation

```
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, 0.5020, 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, 0.4980, 0.5529,
          0.8471, 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, 0.0000, 0.0667, 0.0745, 0.5412, 0.9725, 0.9922,
```

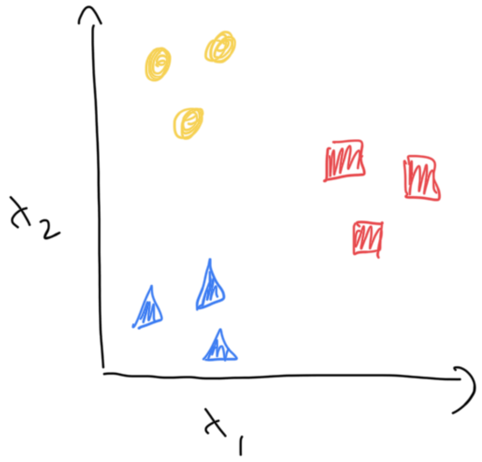


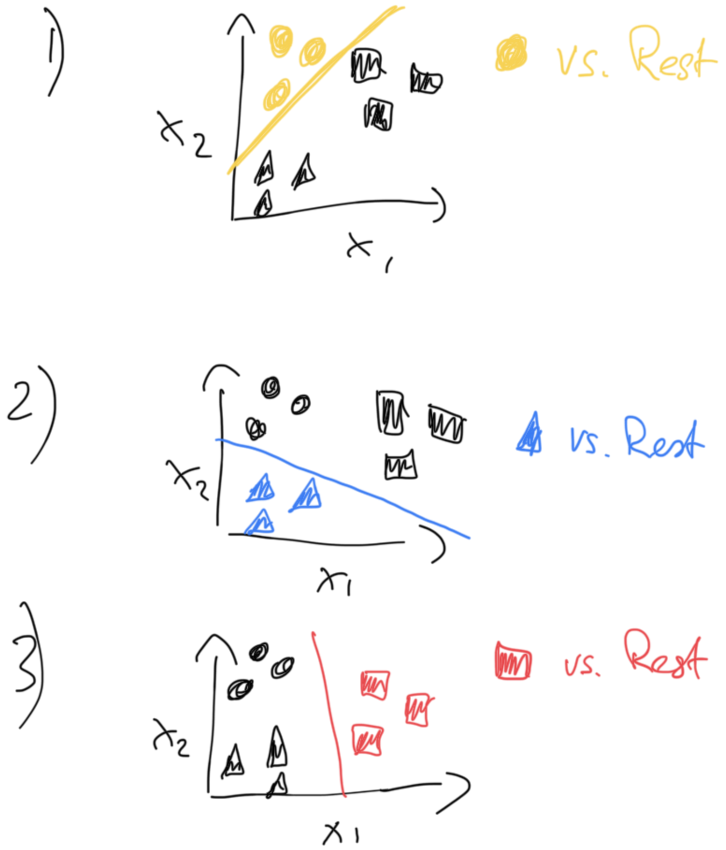
Note that I normalized pixels by factor 1/255 here

One approach to multi-class classification

One-vs-all

Predict each class label independently…



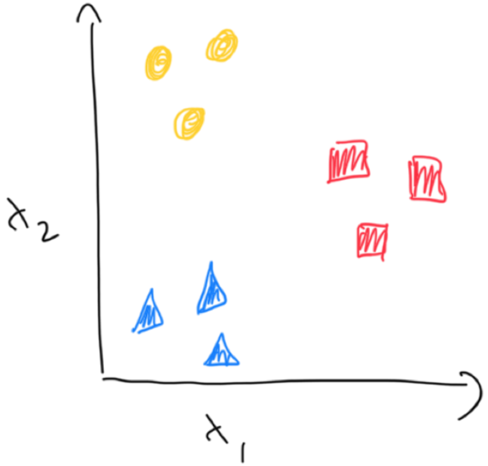


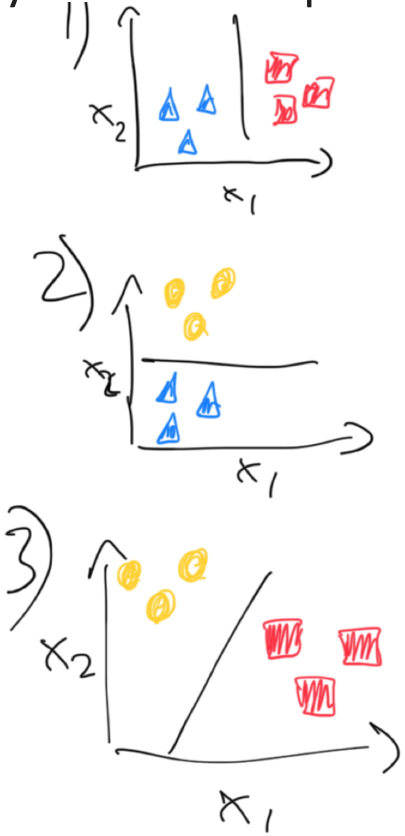
…Then choose the class with the highest confidence score

One approach to multi-class classification

All-vs-all

Explicitly predict the probability of each competing outcome…

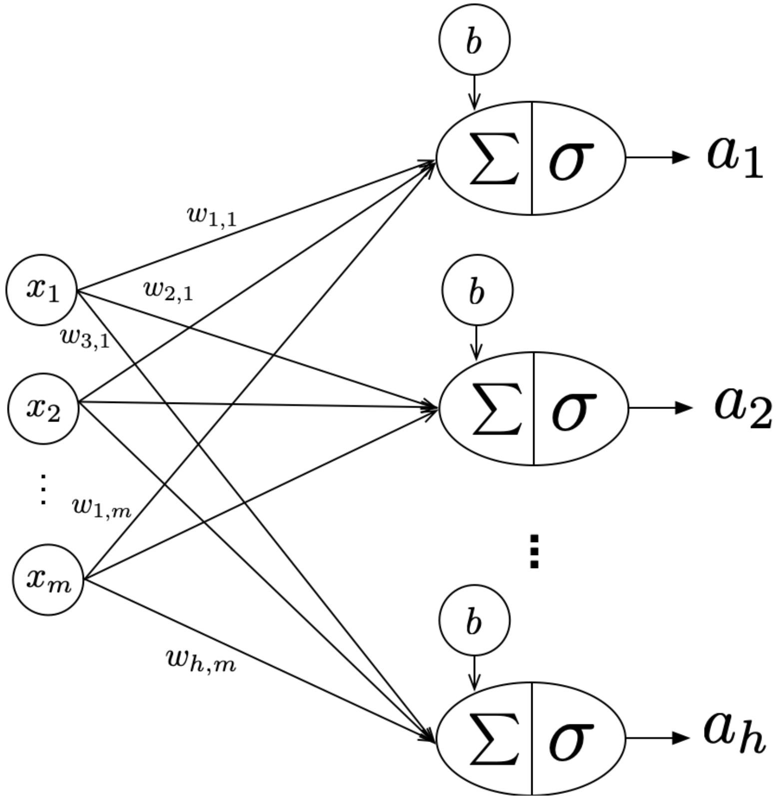




…Then choose the class with the highest confidence score

Another approach

Predict probabilities of class membership simultaneously



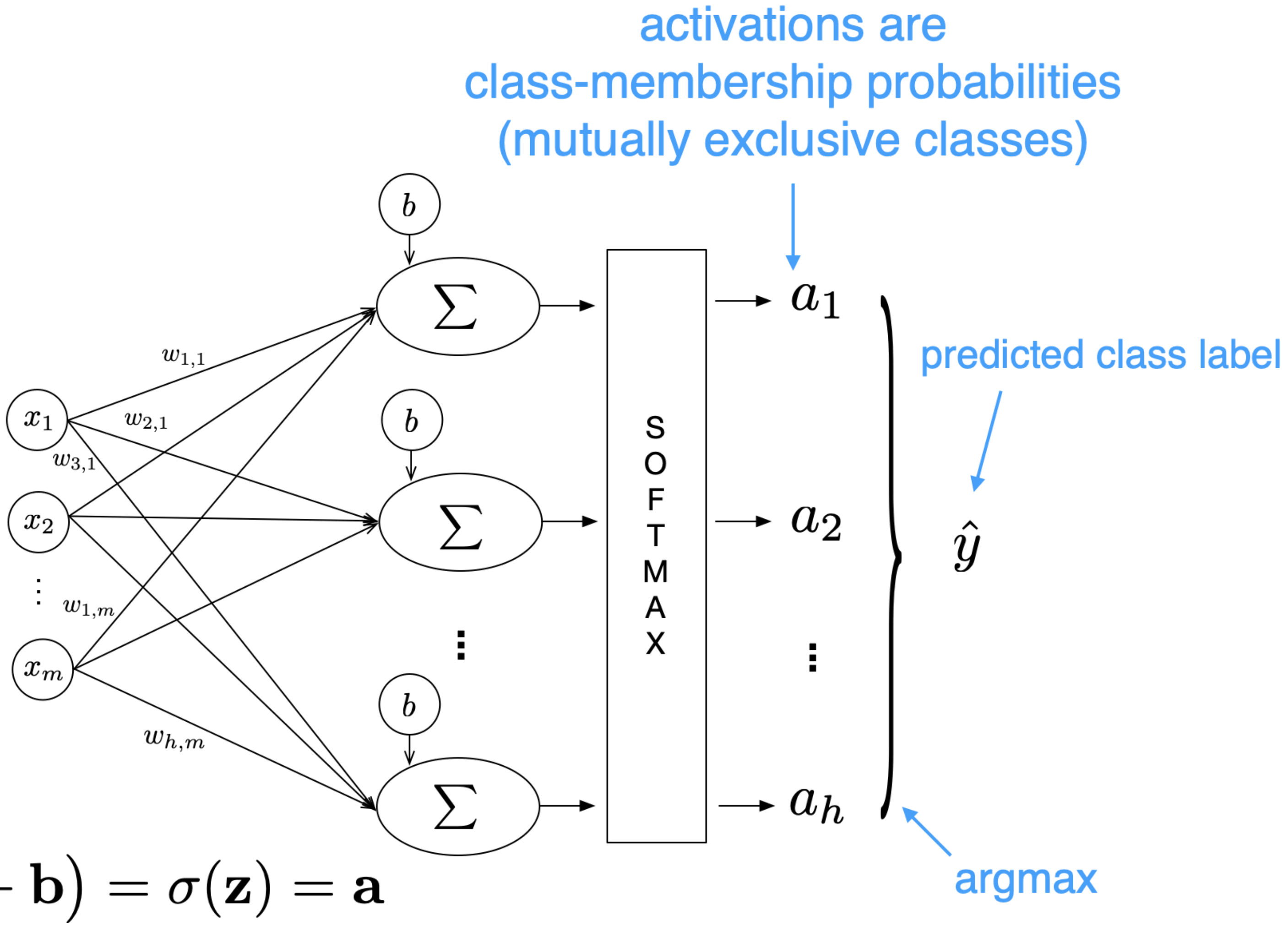
activations are class-membership probabilities (NOT mutually exclusive classes)

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

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

where *h* is the number of classes

Multinomial ("Softmax") Logistic Regression

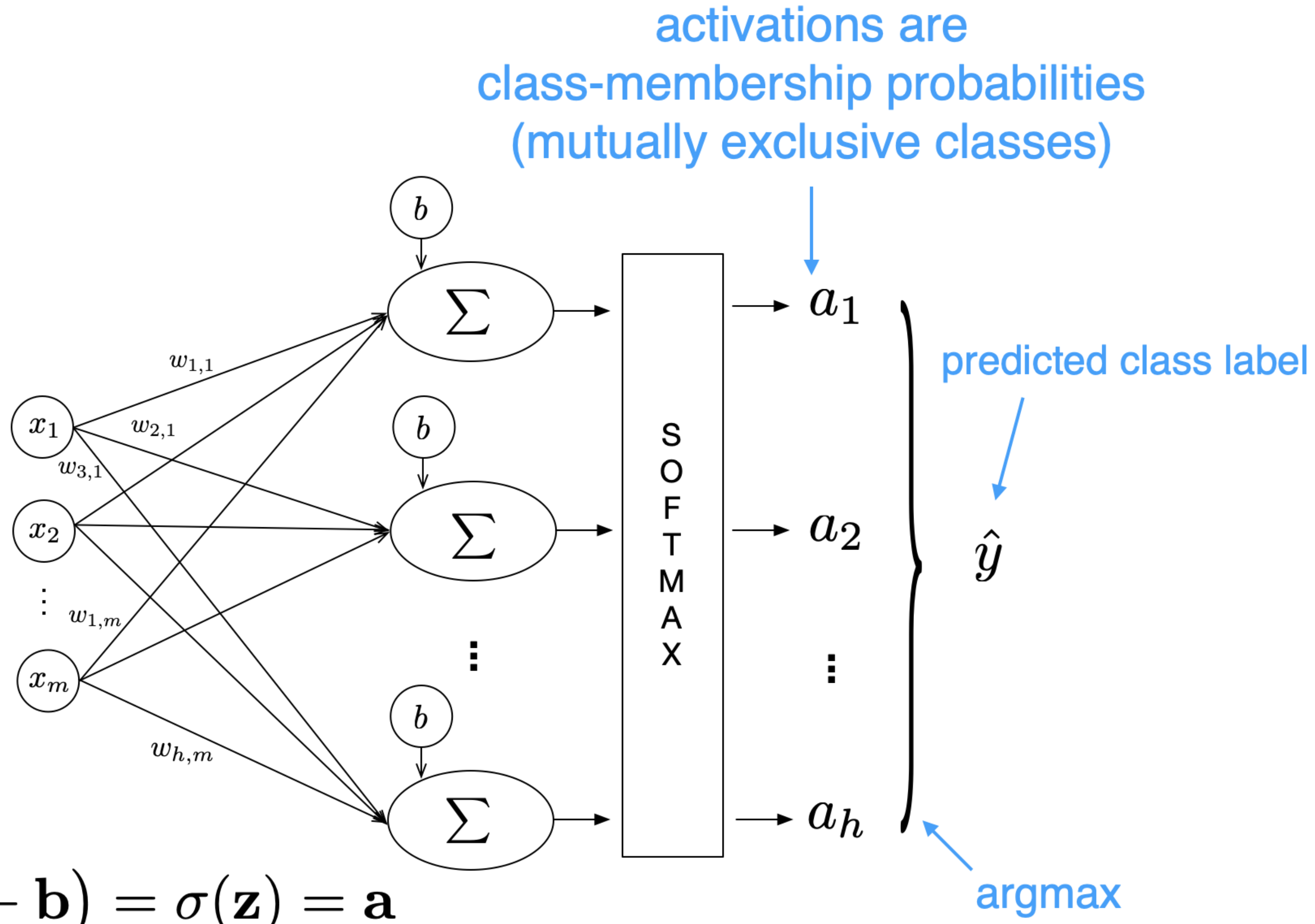


$$\sigma(\mathbf{Wx} + \mathbf{b}) = \sigma(\mathbf{z}) = \mathbf{a}$$

Today: Our old friend logistic 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. **One-Hot Encoding and Multi-category Cross-Entropy**
8. Softmax Regression Learning Rule
9. Softmax Regression Code Example

Multinomial ("Softmax") Logistic Regression



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

"Softmax"

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

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

h is the number of class labels

A "soft" (differentiable) version of "max"

Requires one-hot encoding

class labels
0
1
3
2

→

class_0	class_1	class_2	class_3
1	0	0	0
0	1	0	0
0	0	0	1
0	0	1	0

Loss Function (assuming one-hot encoding)

(Multi-category) Cross Entropy

for *h* different class labels

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

Loss Function (assuming one-hot encoding)

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

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 Example

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

(4 training examples, 3 classes)

Cross Entropy Example

1 training example

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

(4 training examples, 3 classes)

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

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

Cross Entropy Example

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

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

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

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

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

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

$$\approx 0.9335$$

Today: Our old friend logistic regression…

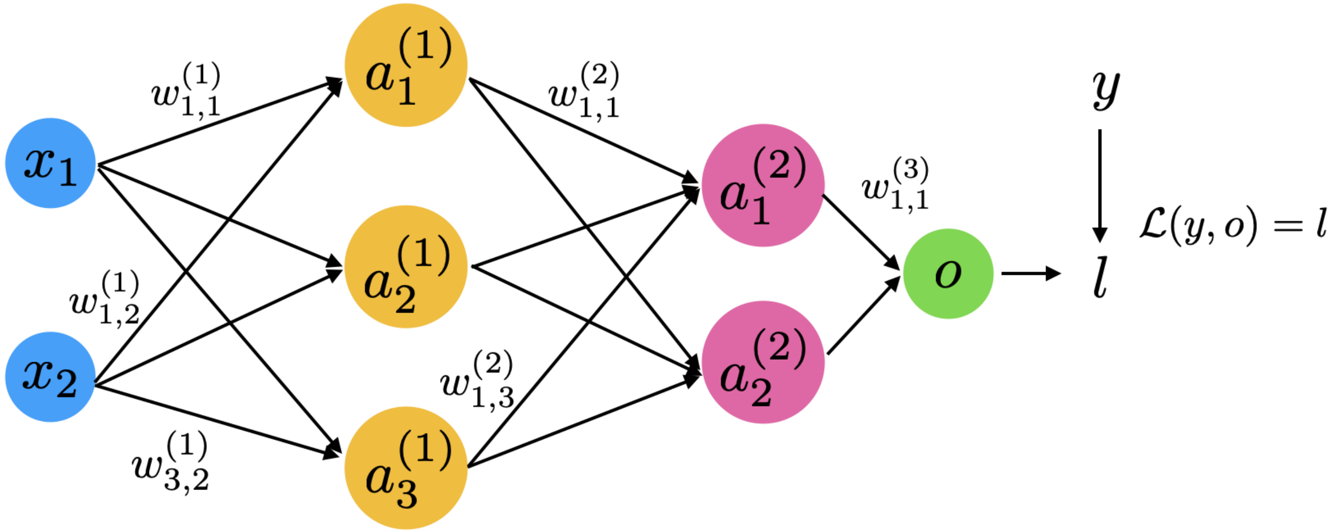
1. Logistic regression as an artificial neuron
2. Negative log-likelihood loss
3. Logistic Regression Learning Rule
4. Logits and Cross-Entropy
5. Logistic Regression Code Example
6. Generalizing to Multiple Classes: Softmax Regression
7. One-Hot Encoding and Multi-category Cross-Entropy
8. **Softmax Regression Learning Rule**
9. Softmax Regression Code Example

The Same Overall Concept Applies…

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

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

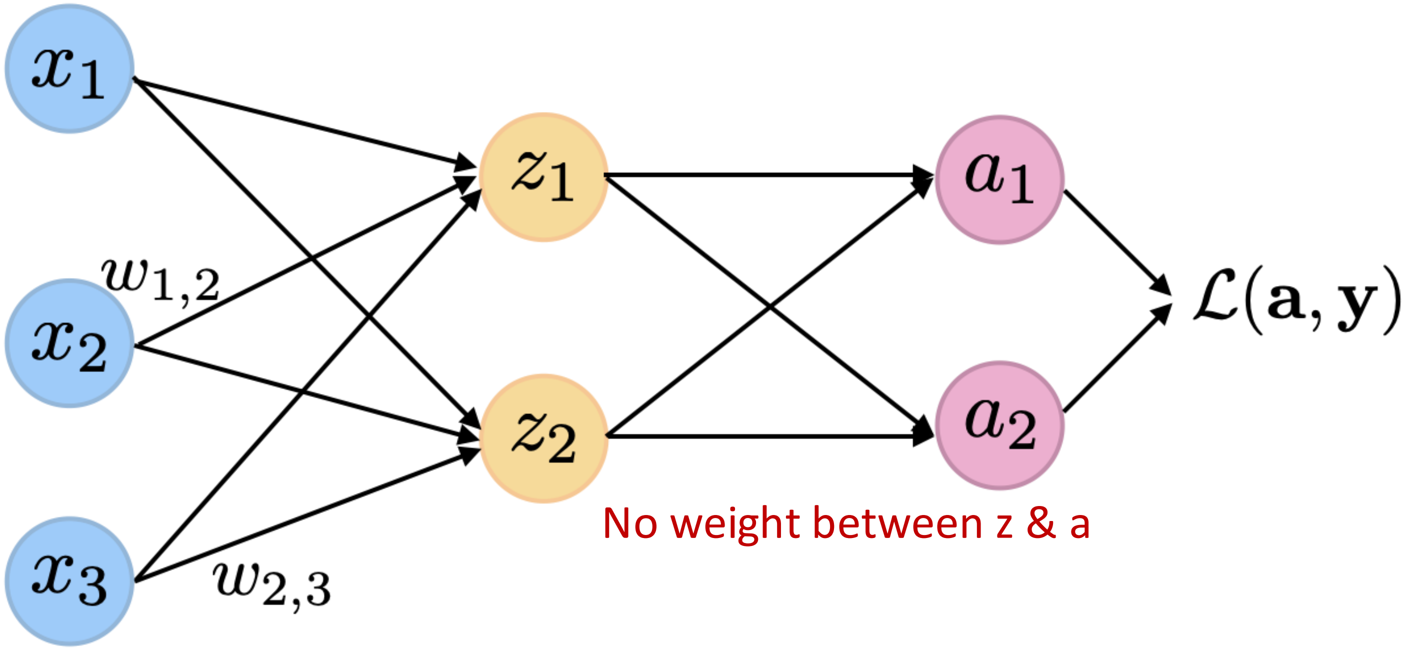
$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial a}\frac{da}{dz}\frac{\partial z}{\partial b}$$



$$\frac{\partial l}{\partial w^{(1)}_{1,1}} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a^{(2)}_1} \cdot \frac{\partial a^{(2)}_1}{\partial a^{(1)}_1} \cdot \frac{\partial a^{(1)}_1}{\partial w^{(1)}_{1,1}} + \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a^{(2)}_2} \cdot \frac{\partial a^{(2)}_2}{\partial a^{(1)}_1} \cdot \frac{\partial a^{(1)}_1}{\partial w^{(1)}_{1,1}}$$

Multivariable chain rule :)

Softmax Regression Sketch



No weight between z & a

Multivariable chain rule

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

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

Softmax Regression Derivation

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

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

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

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

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

Softmax Regression Derivation

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

$\frac{\partial a_1}{\partial z_1} \rightarrow a_1(1 - a_1)$

$$\frac{\partial a_1}{\partial z_1} = \frac{\partial}{\partial z_1}\left[\frac{e^{z_1}}{\sum_{j=1}^{h} e^{z_j}}\right]$$

	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	$f(g(x))$	$f'(g(x))g'(x)$

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

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

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

Softmax Regression Derivation

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

$\frac{\partial a_2}{\partial z_1} \rightarrow -a_2 a_1$

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

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

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

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

	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	$f(g(x))$	$f'(g(x))g'(x)$

Softmax Regression Derivation

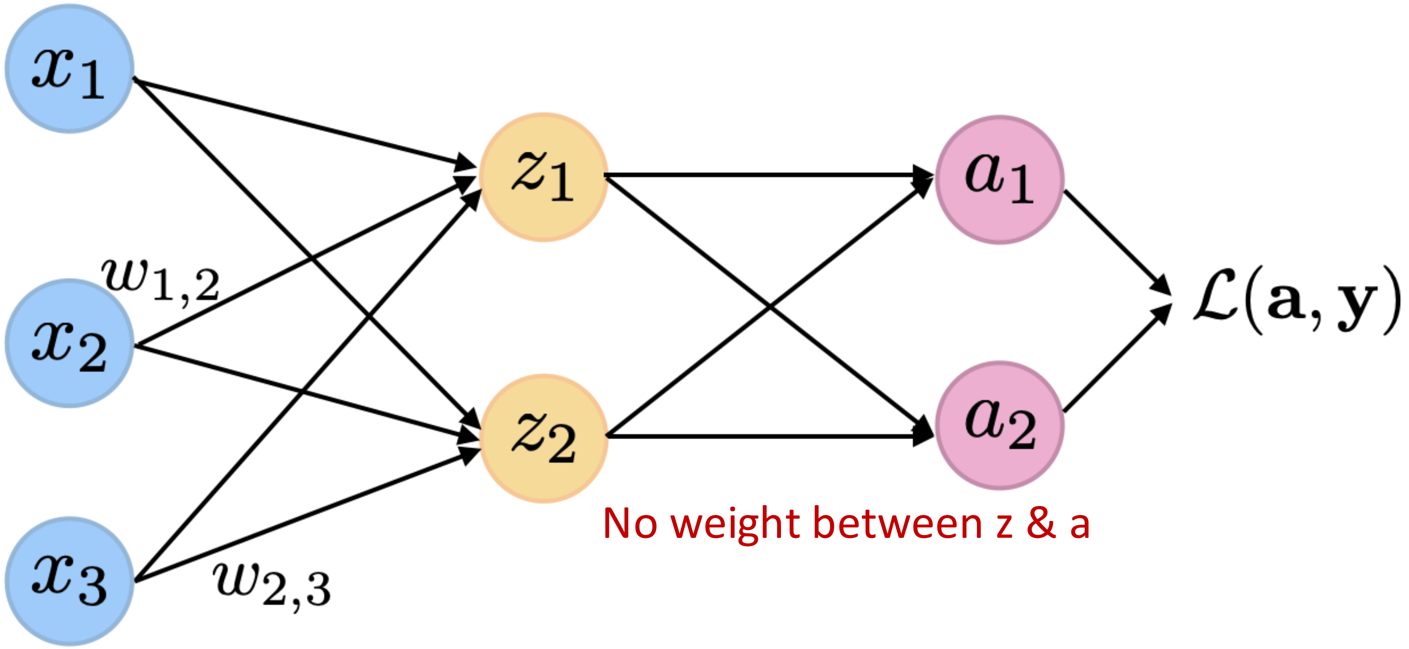
$$\frac{\partial L}{\partial w_{1,2}} = \frac{\partial L}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial w_{1,2}} + \frac{\partial L}{\partial a_2}\frac{\partial a_2}{\partial z_1}\frac{\partial z_1}{\partial w_{1,2}}$$

x_2

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

$$= x_2$$

Softmax Regression Sketch



No weight between z & a

Multivariable chain rule

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

Vectorized Form:

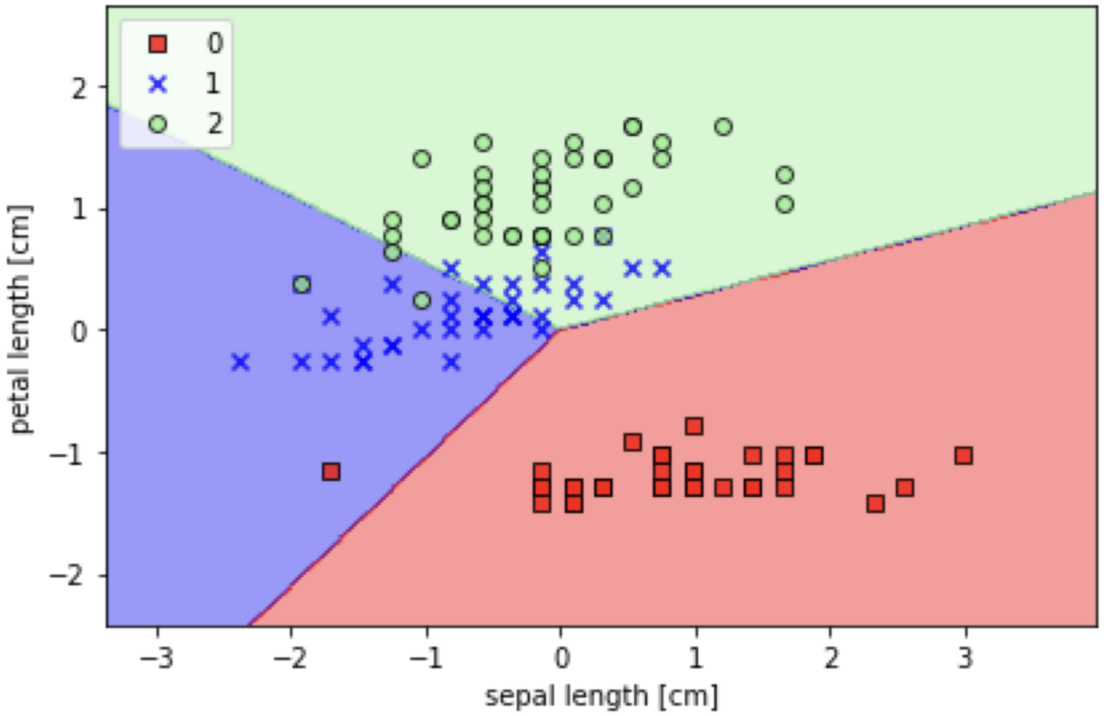
$$\nabla_{\mathbf{W}}\mathcal{L} = -(\mathbf{X}^\top(\mathbf{Y} - \mathbf{A}))^\top$$

where $\mathbf{W} \in \mathbb{R}^{k\times m}$
$\mathbf{X} \in \mathbb{R}^{n\times m}$
$\mathbf{A} \in \mathbb{R}^{n\times h}$
$\mathbf{Y} \in \mathbb{R}^{n\times h}$

Today: Our old friend logistic 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. One-Hot Encoding and Multi-category Cross-Entropy
8. Softmax Regression Learning Rule
9. **Softmax Regression Code Example**

Softmax Regression Hands-On Example



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

Questions?

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON