# CS 453X: Class 8

Jacob Whitehill

# Softmax regression
# (aka multinomial logistic regression)

# Multi-class classification

- So far we have talked about classifying only 2 classes (e.g., smile versus non-smile).
  - This is sometimes called **binary classification**.
- But there are many settings in which multiple (>2) classes exist, e.g., emotion recognition, hand-written digit recognition:


6 classes (fear, anger, sadness, happiness, disgust, surprise)

10 classes (0-9)

# Classification versus regression

- Note that, even though the hand-written digit recognition ("MNIST") problem has classes called "0" , "1", …, "9", there is no sense of "distance" between the classes.
  - Misclassifying a 1 as a 2 is just as "bad" as misclassifying a 1 as a 9.

# Multi-class classification

- It turns out that logistic regression can easily be extended to support an arbitrary number (≥2) of classes.
    - The multi-class case is called **softmax regression** or sometimes **multinomial logistic regression.**
- How to represent the ground-truth $y$ and prediction $\hat{y}$?
    - Instead of just a scalar $y$, we will use a vector $\mathbf{y}$.

# Example: 2 classes

- Suppose we have a dataset of 3 examples, where the ground-truth class labels are 0, 1, 0.

- Then we would define our ground-truth vectors as:

$$\mathbf{y}^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\mathbf{y}^{(2)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\mathbf{y}^{(3)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

- Exactly 1 coordinate of each **y** is 1; the others are 0.

# Example: 2 classes

- Suppose we have a dataset of 3 examples, where the ground-truth class labels are 0, 1, 0.

- Then we would define our ground-truth vectors as:

$$\mathbf{y}^{(1)} = \begin{bmatrix} \boxed{1} \\ 0 \end{bmatrix}$$

This "slot" is for class 0.

$$\mathbf{y}^{(2)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\mathbf{y}^{(3)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

- This is called a **one-hot encoding** of the class label.

# Example: 2 classes

- Suppose we have a dataset of 3 examples, where the ground-truth class labels are 0, 1, 0.

- Then we would define our ground-truth vectors as:

$$\mathbf{y}^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

This "slot" is for class 1.

$$\mathbf{y}^{(2)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\mathbf{y}^{(3)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

- This is called a **one-hot encoding** of the class label.

# Example: 2 classes

- The machine's predictions $\hat{\mathbf{y}}$ about each example's label are also **probabilistic**.
- They could consist of:

$$\hat{\mathbf{y}}^{(1)} = \begin{bmatrix} \boxed{0.93} \\ 0.07 \end{bmatrix} \longleftarrow \text{Machine's "belief" that the label is 0.}$$

$$\hat{\mathbf{y}}^{(2)} = \begin{bmatrix} 0.4 \\ 0.6 \end{bmatrix}$$

$$\hat{\mathbf{y}}^{(3)} = \begin{bmatrix} 0.99 \\ 0.01 \end{bmatrix}$$

- Each coordinate of $\hat{\boldsymbol{y}}$ is a probability.

# Example: 2 classes

- The machine's predictions $\hat{\mathbf{y}}$ about each example's label are also **probabilistic**.

- They could consist of:

$$\hat{\mathbf{y}}^{(1)} = \begin{bmatrix} 0.93 \\ 0.07 \end{bmatrix}$$

Machine's "belief" that the label is 1.

$$\hat{\mathbf{y}}^{(2)} = \begin{bmatrix} 0.4 \\ 0.6 \end{bmatrix}$$

$$\hat{\mathbf{y}}^{(3)} = \begin{bmatrix} 0.99 \\ 0.01 \end{bmatrix}$$

- The sum of the coordinates in each $\hat{\mathbf{y}}$ is 1.

# Cross-entropy loss

- We need a loss function that can support $c \geq 2$ classes.

- We will use the **cross-entropy** loss (aka **negative log-likelihood**):

$$f_{\text{CE}} = -\sum_{i=1}^{n} \sum_{k=1}^{c} \mathbf{y}_k^{(i)} \log \hat{\mathbf{y}}_k^{(i)}$$

# Cross-entropy loss

- Note that the $f_{\text{log}}$ (for logistic regression) is a special case of $f_{\text{CE}}$ (for softmax regression) for $c$=2.

- To see how, consider just a simple example:

$$f_{\text{CE}} \quad = \quad -\sum_{k=0}^{1} \mathbf{y}_k \log \hat{\mathbf{y}}_k$$

# Cross-entropy loss

- Note that the $f_{\log}$ (for logistic regression) is a special case of $f_{\mathrm{CE}}$ (for softmax regression) for $c$=2.

- To see how, consider just a simple example:

$$f_{\mathrm{CE}} = -\sum_{k=0}^{1} \mathbf{y}_k \log \hat{\mathbf{y}}_k$$

**Note: the sum from *k*=1 to *c* is renumbered from 0 to c-1.**

# Cross-entropy loss

- Note that the $f_{\text{log}}$ (for logistic regression) is a special case of $f_{\text{CE}}$ (for softmax regression) for $c$=2.

- To see how, consider just a simple example:

$$\begin{aligned} f_{\text{CE}} &= -\sum_{k=0}^{1} \mathbf{y}_k \log \hat{\mathbf{y}}_k \\ &= -\mathbf{y}_1 \log \hat{\mathbf{y}}_1 - \mathbf{y}_0 \log \hat{\mathbf{y}}_0 \end{aligned}$$

# Cross-entropy loss

- Note that the $f_{\log}$ (for logistic regression) is a special case of $f_{\text{CE}}$ (for softmax regression) for $c$=2.

- To see how, consider just a simple example:

$$\begin{aligned} f_{\text{CE}} &= -\sum_{k=0}^{1} \mathbf{y}_k \log \hat{\mathbf{y}}_k \\ &= -\mathbf{y}_1 \log \hat{\mathbf{y}}_1 - \mathbf{y}_0 \log \hat{\mathbf{y}}_0 \\ &= -\mathbf{y}_1 \log \hat{\mathbf{y}}_1 - (1 - \mathbf{y}_1) \log(1 - \hat{\mathbf{y}}_1) \end{aligned}$$

$$\hat{\mathbf{y}}^{(1)} = \begin{bmatrix} 0.93 \\ 0.07 \end{bmatrix}$$

**Recall that the sum over all coordinates of each ŷ (and each y) must equal 1. Since there are only 2 classes, then $\hat{y}_0$ =1 - $\hat{y}_1$ (and $y_0$ = 1 - $y_1$).**

# Cross-entropy loss

- Note that the $f_{\text{log}}$ (for logistic regression) is a special case of $f_{\text{CE}}$ (for softmax regression) for $c$=2.

- To see how, consider just a simple example:

$$
\begin{aligned}
f_{\text{CE}} &= -\sum_{k=0}^{1} \mathbf{y}_k \log \hat{\mathbf{y}}_k \\
&= -\mathbf{y}_1 \log \hat{\mathbf{y}}_1 - \mathbf{y}_0 \log \hat{\mathbf{y}}_0 \\
&= -\mathbf{y}_1 \log \hat{\mathbf{y}}_1 - (1 - \mathbf{y}_1) \log(1 - \hat{\mathbf{y}}_1) \\
&= -\mathbf{y} \log \hat{\mathbf{y}} - (1 - \mathbf{y}) \log(1 - \hat{\mathbf{y}})
\end{aligned}
$$

**For $c$=2 classes, we can define ŷ (and y) simply as probability that the example is class 1.**

# Cross-entropy loss

- Note that the $f_{\log}$ (for logistic regression) is a special case of $f_{\mathrm{CE}}$ (for softmax regression) for $c$=2.

- To see how, consider just a simple example:

$$
\begin{aligned}
f_{\mathrm{CE}} &= -\sum_{k=0}^{1} \mathbf{y}_k \log \hat{\mathbf{y}}_k \\
&= -\mathbf{y}_1 \log \hat{\mathbf{y}}_1 - \mathbf{y}_0 \log \hat{\mathbf{y}}_0 \\
&= -\mathbf{y}_1 \log \hat{\mathbf{y}}_1 - (1 - \mathbf{y}_1) \log(1 - \hat{\mathbf{y}}_1) \\
&= -\mathbf{y} \log \hat{\mathbf{y}} - (1 - \mathbf{y}) \log(1 - \hat{\mathbf{y}}) \\
&= f_{\log}
\end{aligned}
$$

# Softmax activation function

- Logistic regression outputs a *scalar* probabilistic class label $\hat{y}$.
  - We needed just a single weight vector $\mathbf{w}$, so that $\hat{y} = \sigma(\mathbf{x}^\top \mathbf{w})$
- Softmax regression outputs a *vector* of probabilistic class labels $\hat{\mathbf{y}}$ containing $c$ components.
  - We need $c$ different vectors of weights $\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(c)}$.

# Softmax activation function

- With softmax regression, we first compute:

$$\mathbf{z}_1 = \mathbf{x}^\top \mathbf{w}^{(1)}$$
$$\mathbf{z}_2 = \mathbf{x}^\top \mathbf{w}^{(2)}$$
$$\ldots$$
$$\mathbf{z}_c = \mathbf{x}^\top \mathbf{w}^{(c)}$$

**I will refer to the z's as "pre-activation scores".**

# Softmax activation function

- With softmax regression, we first compute:

$$\mathbf{z}_1 = \mathbf{x}^\top \mathbf{w}^{(1)}$$
$$\mathbf{z}_2 = \mathbf{x}^\top \mathbf{w}^{(2)}$$
$$\ldots$$
$$\mathbf{z}_c = \mathbf{x}^\top \mathbf{w}^{(c)}$$

- We then **normalize** across all *c* classes so that:

1. Each output $\hat{\mathbf{y}}_k$ is non-negative.
2. The sum of $\hat{\mathbf{y}}_k$ over all *c* classes is 1.

# Normalization of the $\hat{y}_k$

1. To enforce non-negativity, we can exponential each $\mathbf{z}_k$:

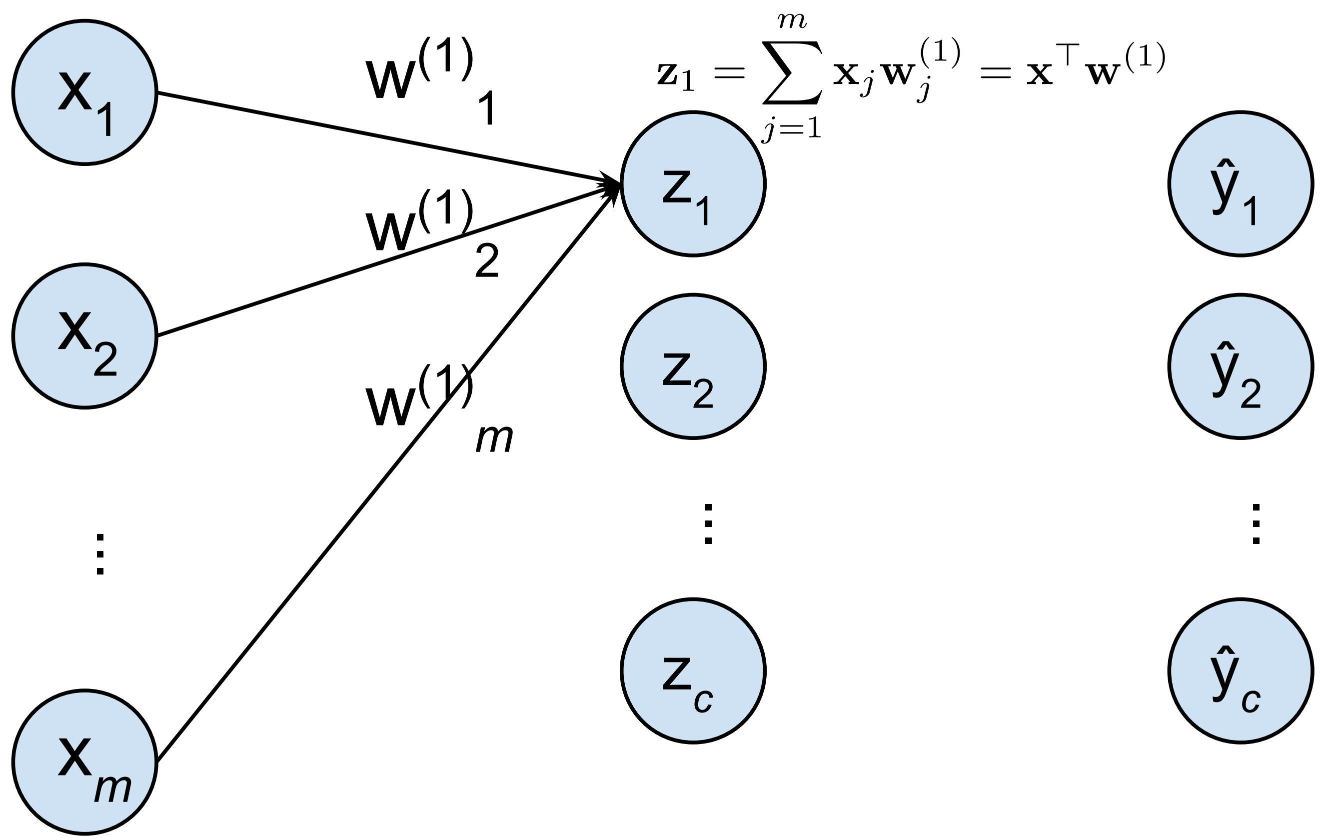
$$\hat{\mathbf{y}}_k = \quad \exp(\mathbf{z}_k)$$

# Normalization of the $\hat{y}_k$

2. To enforce that the $\hat{\mathbf{y}}_k$ sum to 1, we can divide each entry by the sum:

$$\hat{\mathbf{y}}_k = \frac{\exp(\mathbf{z}_k)}{\sum_{k'=1}^{c} \exp(\mathbf{z}_{k'})}$$

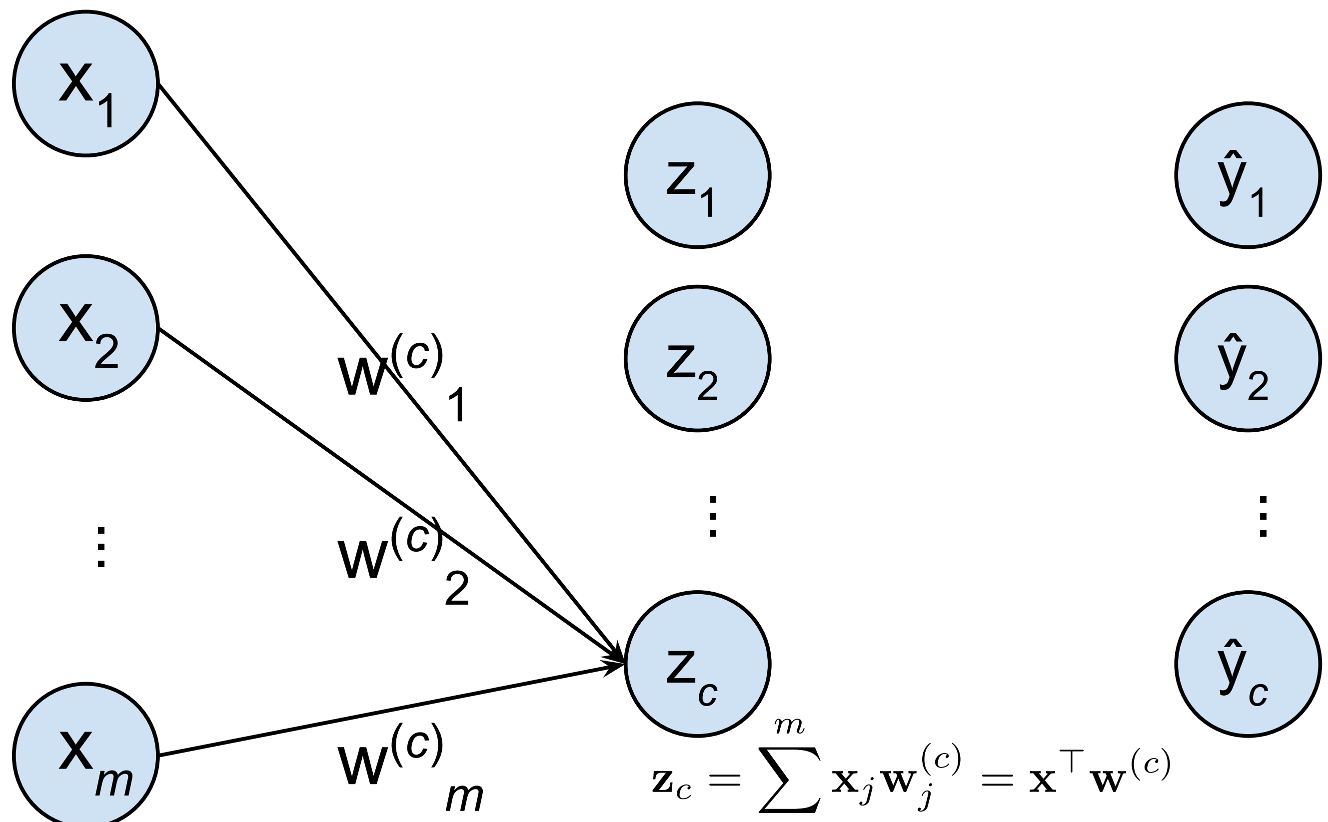
# Softmax regression diagram




- With softmax regression, we first compute:

$$\mathbf{z}_1 = \mathbf{x}^\top \mathbf{w}^{(1)}$$

# Softmax regression diagram




$$\mathbf{z}_c = \sum_{j=1}^{m} \mathbf{x}_j \mathbf{w}_j^{(c)} = \mathbf{x}^\top \mathbf{w}^{(c)}$$

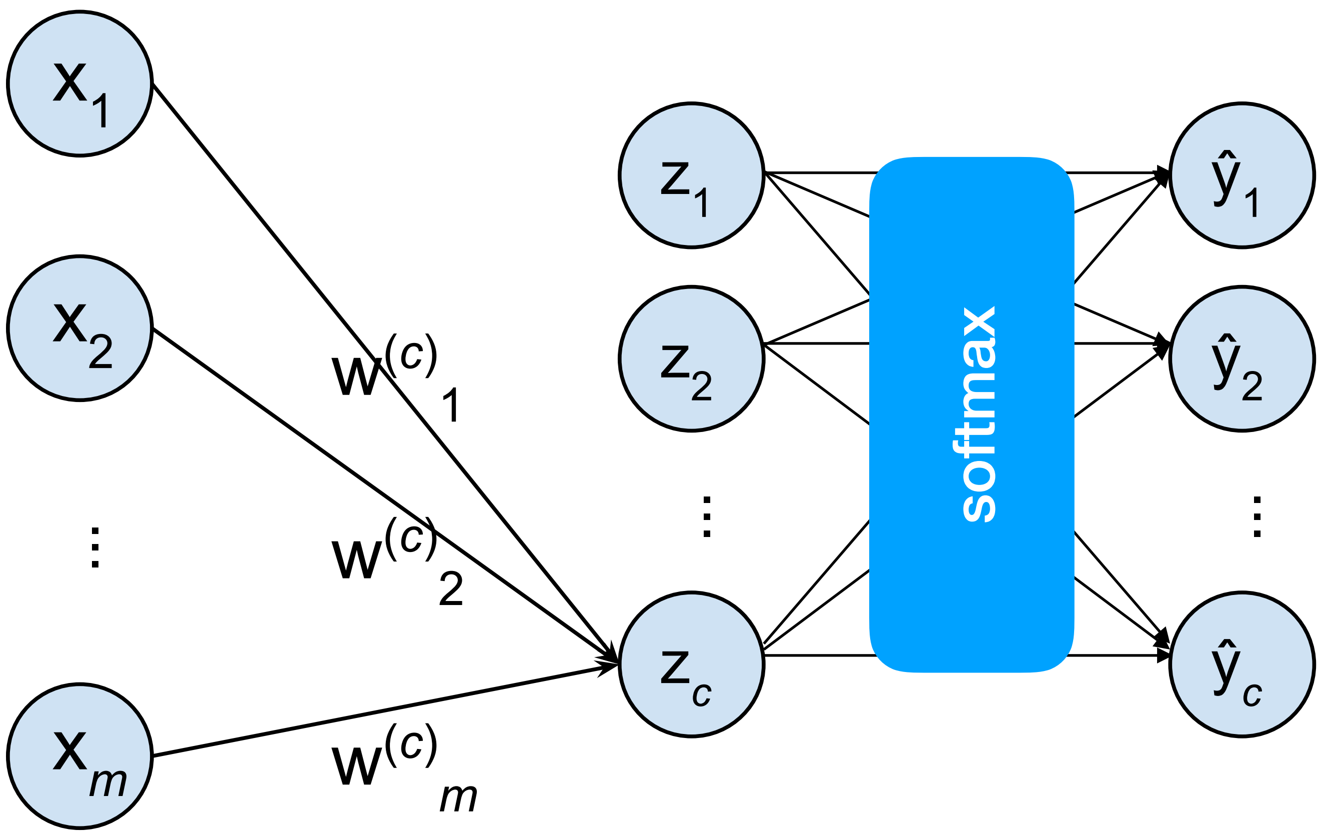
- With softmax regression, we first compute:

$$\mathbf{z}_1 = \mathbf{x}^\top \mathbf{w}^{(1)}$$

**...**

$$\mathbf{z}_c = \mathbf{x}^\top \mathbf{w}^{(c)}$$

# Softmax regression diagram




$$\hat{\mathbf{y}}_k = \frac{\exp(\mathbf{z}_k)}{\sum_{k'=1}^{c} \exp(\mathbf{z}_{k'})}$$

- We then **normalize** across all *c* classes.

# Illustration

- Let *m*=2, *c*=3.
- Let: $\mathbf{x} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$

$$\mathbf{w}^{(1)} = \begin{bmatrix} -2.5 \\ -1 \end{bmatrix} \qquad \mathbf{w}^{(2)} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad \mathbf{w}^{(3)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

- Which class will have highest estimated probability?

$$\mathbf{z} = \begin{bmatrix} \quad \\ \quad \\ \quad \end{bmatrix}$$

# Illustration

- Let $m$=2, $c$=3.
- Let: $$\mathbf{x} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$\mathbf{w}^{(1)} = \begin{bmatrix} -2.5 \\ -1 \end{bmatrix} \qquad \mathbf{w}^{(2)} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad \mathbf{w}^{(3)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

- Which class will have highest estimated probability?

$$\mathbf{z} = \begin{bmatrix} 1.5 \\ 1 \\ -1 \end{bmatrix}$$

# Illustration

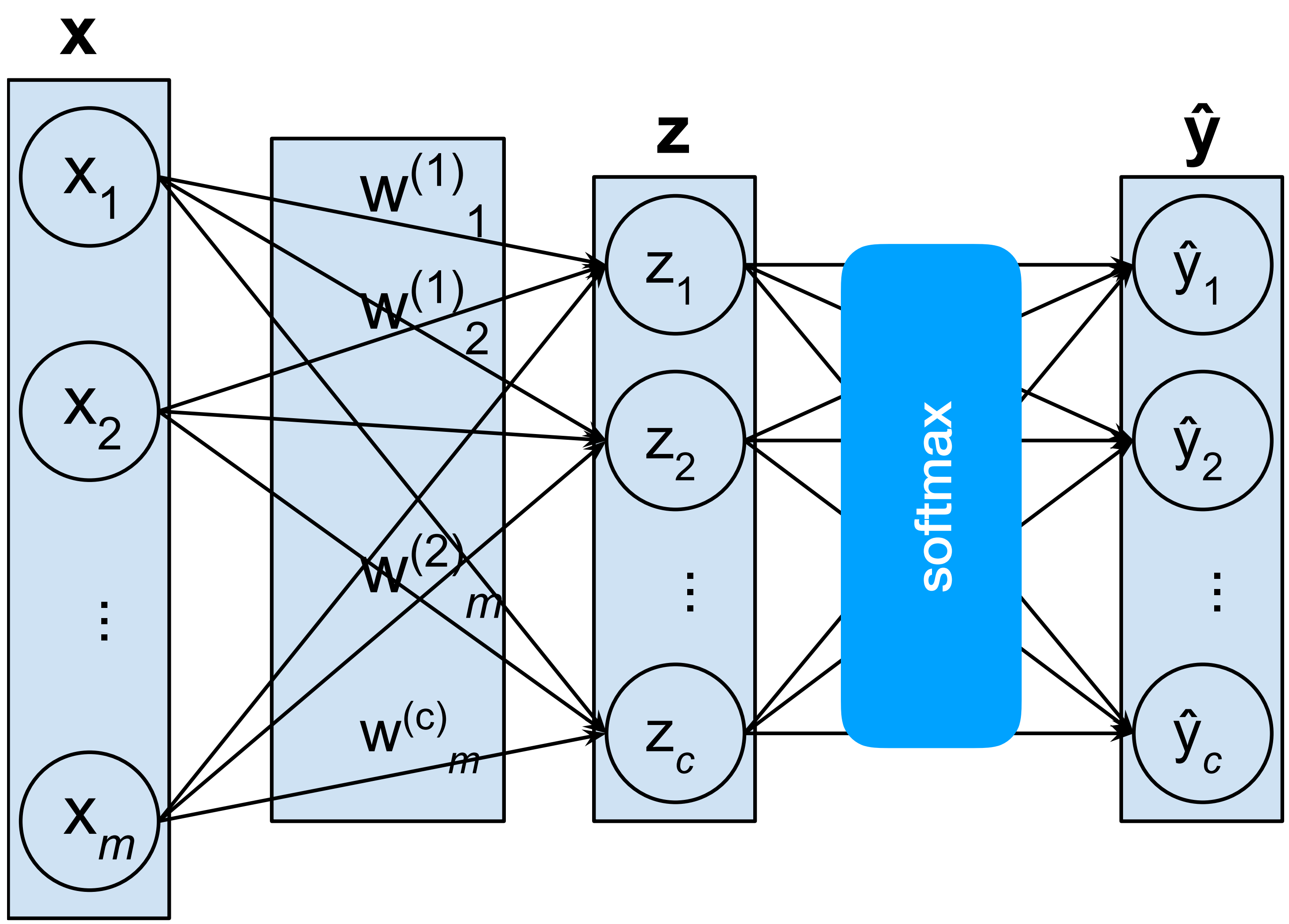
- Let $m$=2, $c$=3.
- Let: $\mathbf{x} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$

$$\mathbf{w}^{(1)} = \begin{bmatrix} -2.5 \\ -1 \end{bmatrix} \qquad \mathbf{w}^{(2)} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad \mathbf{w}^{(3)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

- Which class will have highest estimated probability?

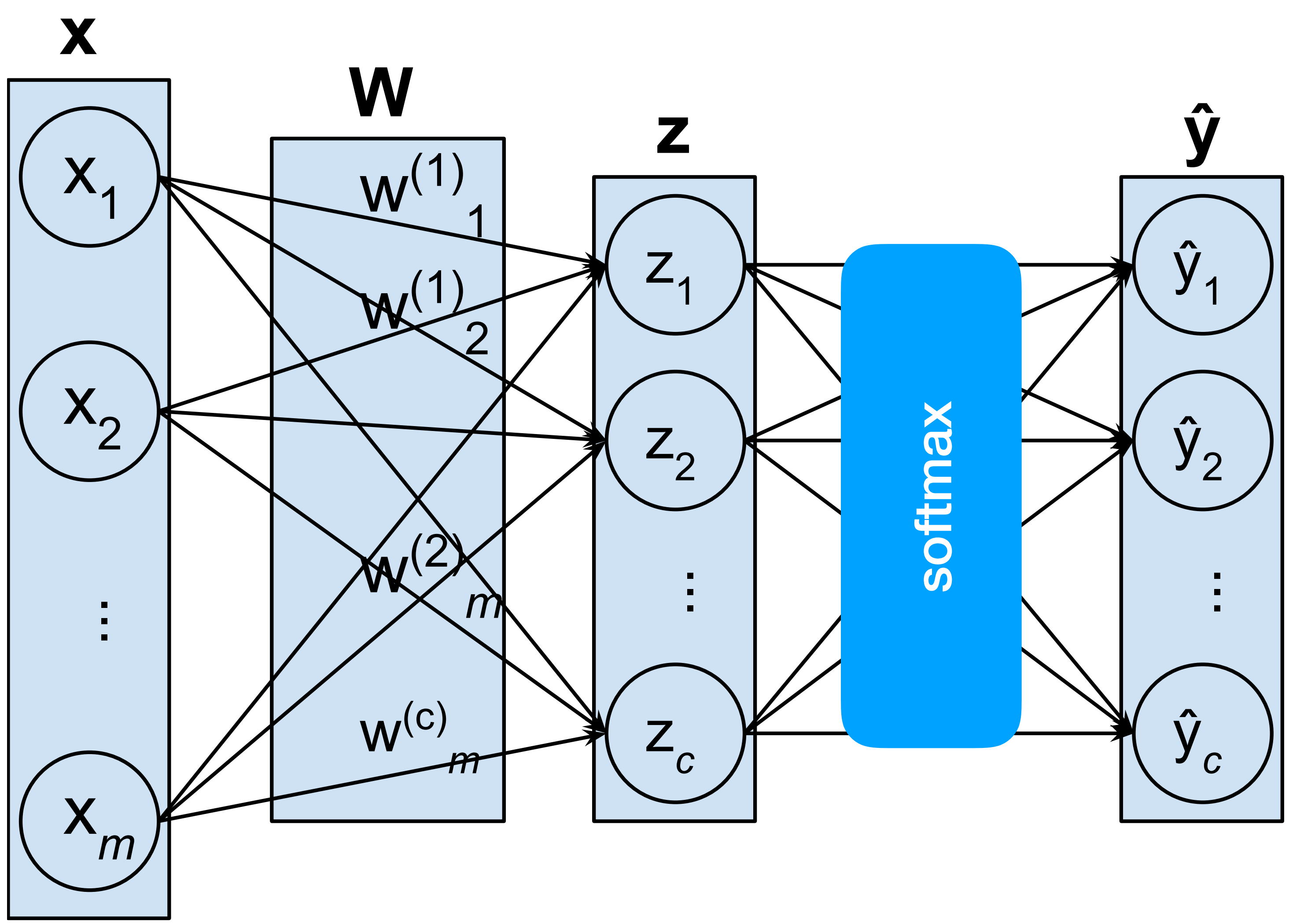
$$\mathbf{z} = \begin{bmatrix} 1.5 \\ 1 \\ -1 \end{bmatrix} \qquad \hat{\mathbf{y}} = \begin{bmatrix} .592 \\ .359 \\ .049 \end{bmatrix}$$

# Softmax regression: vectorization



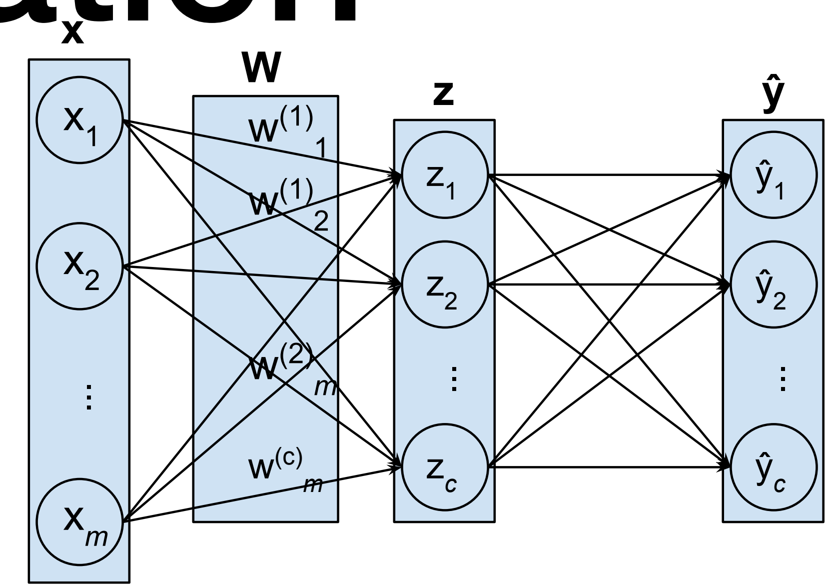

- We can represent each **layer** as a vector (**x**, **z**, **ŷ**).

# Softmax regression: vectorization




- We can represent the collection of all *c* weight vectors $\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(c)}$ as a matrix $\mathbf{W}$.

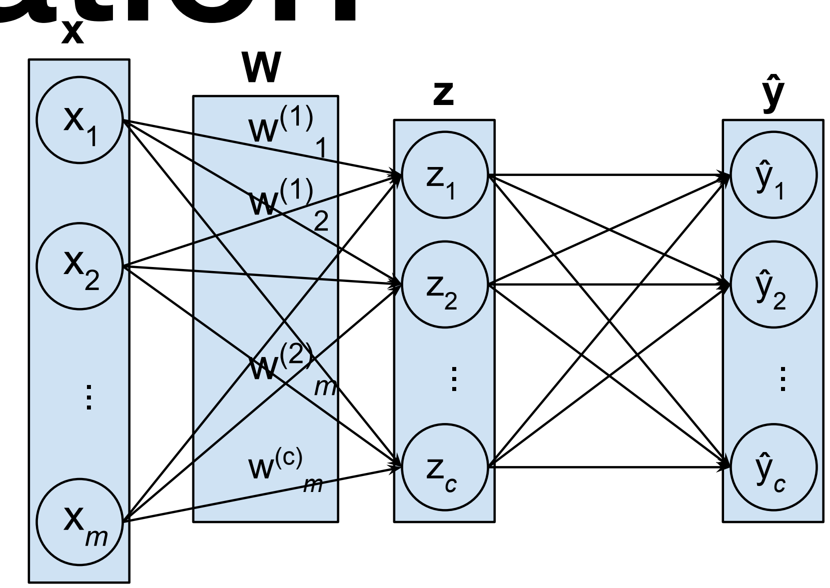
# Softmax regression: vectorization




- Let **x**, **z** be column vectors.
- Let $\mathbf{W} = \left[\begin{array}{ccc} | & & | \\ \mathbf{w}^{(1)} & \ldots & \mathbf{w}^{(c)} \\ | & & | \end{array}\right]$
- How can we compute the "pre-activation scores" **z** for all *c* classes in one-fell-swoop? Choose 0 or more of:

1. $\mathbf{z}^\top = \mathbf{x}^\top \mathbf{W}$
2. $\mathbf{z} = \mathbf{x}^\top \mathbf{W}$
3. $\mathbf{z} = \mathbf{W}\mathbf{x}$
4. $\mathbf{z} = \mathbf{W}^\top \mathbf{x}$

# Softmax regression: vectorization




- Let **x**, **z** be column vectors.
- Let $\mathbf{W} = \begin{bmatrix} | & & | \\ \mathbf{w}^{(1)} & \dots & \mathbf{w}^{(c)} \\ | & & | \end{bmatrix}$
- How can we compute the "pre-activation scores" **z** for all *c* classes in one-fell-swoop? Choose 0 or more of:

**1.** $\mathbf{z}^\top = \mathbf{x}^\top \mathbf{W}$

Both of these are correct.

**4.** $\mathbf{z} = \mathbf{W}^\top \mathbf{x}$

# Softmax regression: vectorization

- By vectorizing, we can compute the pre-activation scores for all *n* examples in one-fell-swoop as:

$$\mathbf{Z} = \mathbf{W}^\top \mathbf{X}$$

***c* x *n* matrix**

# Softmax regression: vectorization

- By vectorizing, we can compute the pre-activation scores for all *n* examples in one-fell-swoop as:

$$\mathbf{Z} = \mathbf{W}^\top \mathbf{X}$$ ***c* x *n* matrix**

- With numpy, we can call `np.exp` to exponentiate every element of **Z**.

- We can then use `np.sum` and `/` (element-wise division) to compute the softmax.

# Gradient descent for softmax regression

- With softmax regression, we need to conduct gradient descent on all $c$ of the weights vectors.
- As usual, let's just consider the gradient of the cross-entropy loss for a single example $\mathbf{x}$.
- We will compute the gradient w.r.t. each weight vector $\mathbf{w}_k$ separately (where $k = 1, \ldots, c$).

# Gradient descent for softmax regression

- Gradient for each weight vector $\mathbf{w}_k$:

$$\nabla_{\mathbf{w}_k} f_{\mathrm{CE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{W}) \quad = \quad \mathbf{x}(\hat{\mathbf{y}}_k - \mathbf{y}_k)$$

- This is the same expression (for each *k*) as for linear regression and logistic regression!

- We can vectorize this to compute all *c* gradients over all *n* examples…

# Gradient descent for softmax regression

- Let **Y** and **Ŷ** both be *n* x *c* matrices:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^{(1)} & \dots & \mathbf{y}_c^{(1)} \\ & \vdots & \\ \mathbf{y}_1^{(n)} & \dots & \mathbf{y}_c^{(n)} \end{bmatrix}$$

One-hot encoded vector of class labels for example 1.

# Gradient descent for softmax regression

- Let **Y** and **Ŷ** both be *n* x *c* matrices:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^{(1)} & \dots & \mathbf{y}_c^{(1)} \\ & \vdots & \\ \mathbf{y}_1^{(n)} & \dots & \mathbf{y}_c^{(n)} \end{bmatrix}$$

**One-hot encoded vector of class labels for example *n*.**

# Gradient descent for softmax regression

- Let **Y** and **Ŷ** both be *n* x *c* matrices:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^{(1)} & \dots & \mathbf{y}_c^{(1)} \\ & \vdots & \\ \mathbf{y}_1^{(n)} & \dots & \mathbf{y}_c^{(n)} \end{bmatrix} \qquad \hat{\mathbf{Y}} = \begin{bmatrix} \hat{\mathbf{y}}_1^{(1)} & \dots & \hat{\mathbf{y}}_c^{(1)} \\ & \vdots & \\ \hat{\mathbf{y}}_1^{(n)} & \dots & \hat{\mathbf{y}}_c^{(n)} \end{bmatrix}$$

**The machine's estimates of the *c* class probabilities for example *n*.**

# Gradient descent for softmax regression

- Let **Y** and **Ŷ** both be *n* x *c* matrices:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^{(1)} & \dots & \mathbf{y}_c^{(1)} \\ & \vdots & \\ \mathbf{y}_1^{(n)} & \dots & \mathbf{y}_c^{(n)} \end{bmatrix} \qquad \hat{\mathbf{Y}} = \begin{bmatrix} \hat{\mathbf{y}}_1^{(1)} & \dots & \hat{\mathbf{y}}_c^{(1)} \\ & \vdots & \\ \hat{\mathbf{y}}_1^{(n)} & \dots & \hat{\mathbf{y}}_c^{(n)} \end{bmatrix}$$

- Then we can compute all c gradient vectors as:

$$\nabla_{\mathbf{W}} f_{\mathrm{CE}}(\mathbf{Y}, \hat{\mathbf{Y}}; \mathbf{W}) = \frac{1}{n}\mathbf{X}(\hat{\mathbf{Y}} - \mathbf{Y})$$

# Gradient descent for softmax regression

- Let $\mathbf{Y}$ and $\hat{\mathbf{Y}}$ both be $n$ x $c$ matrices:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^{(1)} & \dots & \mathbf{y}_c^{(1)} \\ & \vdots & \\ \mathbf{y}_1^{(n)} & \dots & \mathbf{y}_c^{(n)} \end{bmatrix} \qquad \hat{\mathbf{Y}} = \begin{bmatrix} \hat{\mathbf{y}}_1^{(1)} & \dots & \hat{\mathbf{y}}_c^{(1)} \\ & \vdots & \\ \hat{\mathbf{y}}_1^{(n)} & \dots & \hat{\mathbf{y}}_c^{(n)} \end{bmatrix}$$

- Then we can compute all c gradient vectors as:

$$\nabla_{\mathbf{W}} f_{\mathrm{CE}}(\mathbf{Y}, \hat{\mathbf{Y}}; \mathbf{W}) = \frac{1}{n}\mathbf{X}(\hat{\mathbf{Y}} - \mathbf{Y})$$

How far the guesses are from ground-truth.

# Gradient descent for softmax regression

- Let **Y** and **Ŷ** both be *n* x *c* matrices:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^{(1)} & \dots & \mathbf{y}_c^{(1)} \\ & \vdots & \\ \mathbf{y}_1^{(n)} & \dots & \mathbf{y}_c^{(n)} \end{bmatrix} \qquad \hat{\mathbf{Y}} = \begin{bmatrix} \hat{\mathbf{y}}_1^{(1)} & \dots & \hat{\mathbf{y}}_c^{(1)} \\ & \vdots & \\ \hat{\mathbf{y}}_1^{(n)} & \dots & \hat{\mathbf{y}}_c^{(n)} \end{bmatrix}$$

- Then we can compute all c gradient vectors as:

$$\nabla_{\mathbf{W}} f_{\mathrm{CE}}(\mathbf{Y}, \hat{\mathbf{Y}}; \mathbf{W}) = \frac{1}{n} \mathbf{X}(\hat{\mathbf{Y}} - \mathbf{Y})$$

The input features (e.g., pixel values).

# Softmax regression demo

- Let's apply softmax regression to train a **handwriting recognition system** that can recognize all 10 digits (0-9).

- We will use the popular MNIST dataset consisting of 60K training examples and 10K testing examples: