CS 4342: Class 8

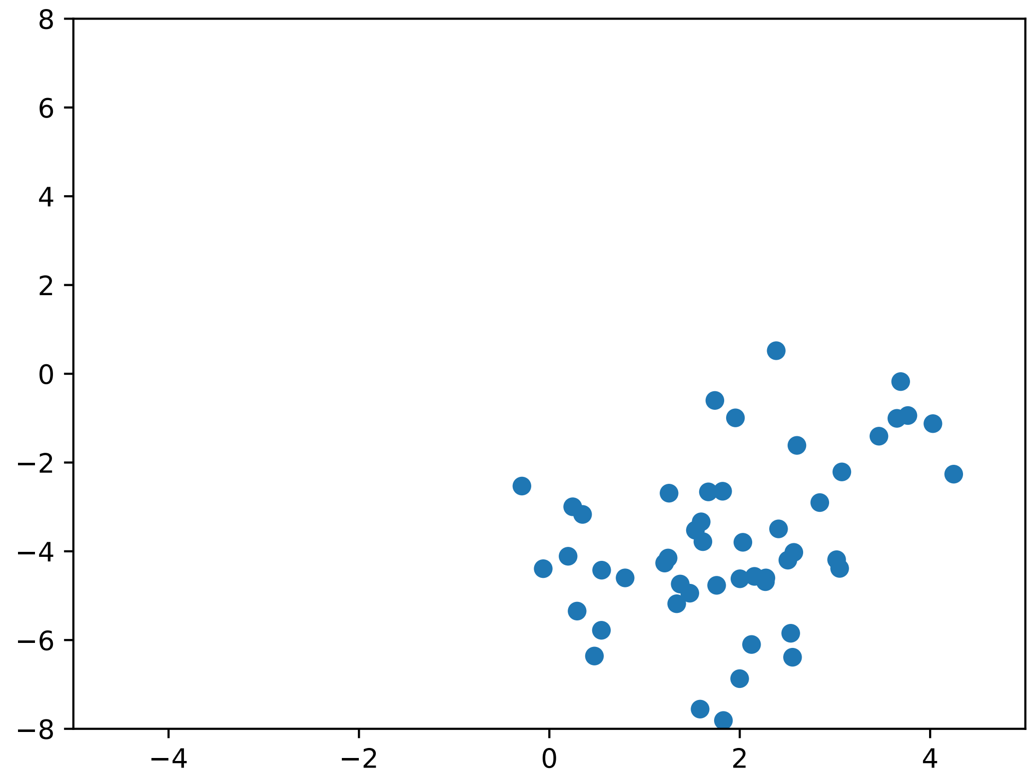
Jacob Whitehill

Exercises for homework 2

- Recall the loss term for L2-regularized regression:

$$f_{\mathrm{MSE}}(\mathbf{w}) = \frac{1}{2n}(\hat{\mathbf{y}} - \mathbf{y})^\top(\hat{\mathbf{y}} - \mathbf{y}) + \frac{\alpha}{2n}\mathbf{w}^\top\mathbf{w}$$

- Suppose we want to predict $\hat{y}$ based on a scalar feature x. How will the line-of-best-fit change as α increases if: (a) our model includes a bias term that we do *not* regularize? (b) our model includes a bias term that we *do* regularize? or (c) our model does *not* include a bias term?



Jacob Whitehill, WPI

Softmax regression (aka multinomial logistic regression)

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 and 2 classes, where the ground-truth class labels are 0, 1, 0.

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

$$\mathbf{y}^{(1)} = \left[\begin{array}{c} 1 \\ 0 \end{array} \right]$$

$$\mathbf{y}^{(2)} = \left[\begin{array}{c} 0 \\ 1 \end{array} \right]$$

$$\mathbf{y}^{(3)} = \left[\begin{array}{c} 1 \\ 0 \end{array} \right]$$

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

Example: 2 classes

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

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

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

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

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

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**):

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

Softmax activation function

- Softmax regression outputs a *vector* of probabilistic class labels $\hat{\mathbf{y}}$ containing c components.
  - We need c different vectors of weights $\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(c)}$.
  - Each weight vector $\mathbf{w}^{(i)}$ measures how "compatible" $\mathbf{x}$ is with class i.

Softmax activation function

- With softmax regression, we first compute:

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

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

Softmax activation function

- With softmax regression, we first compute:

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

- Since we want to output probabilities, 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 exponentiate each $\mathbf{z}_k$:

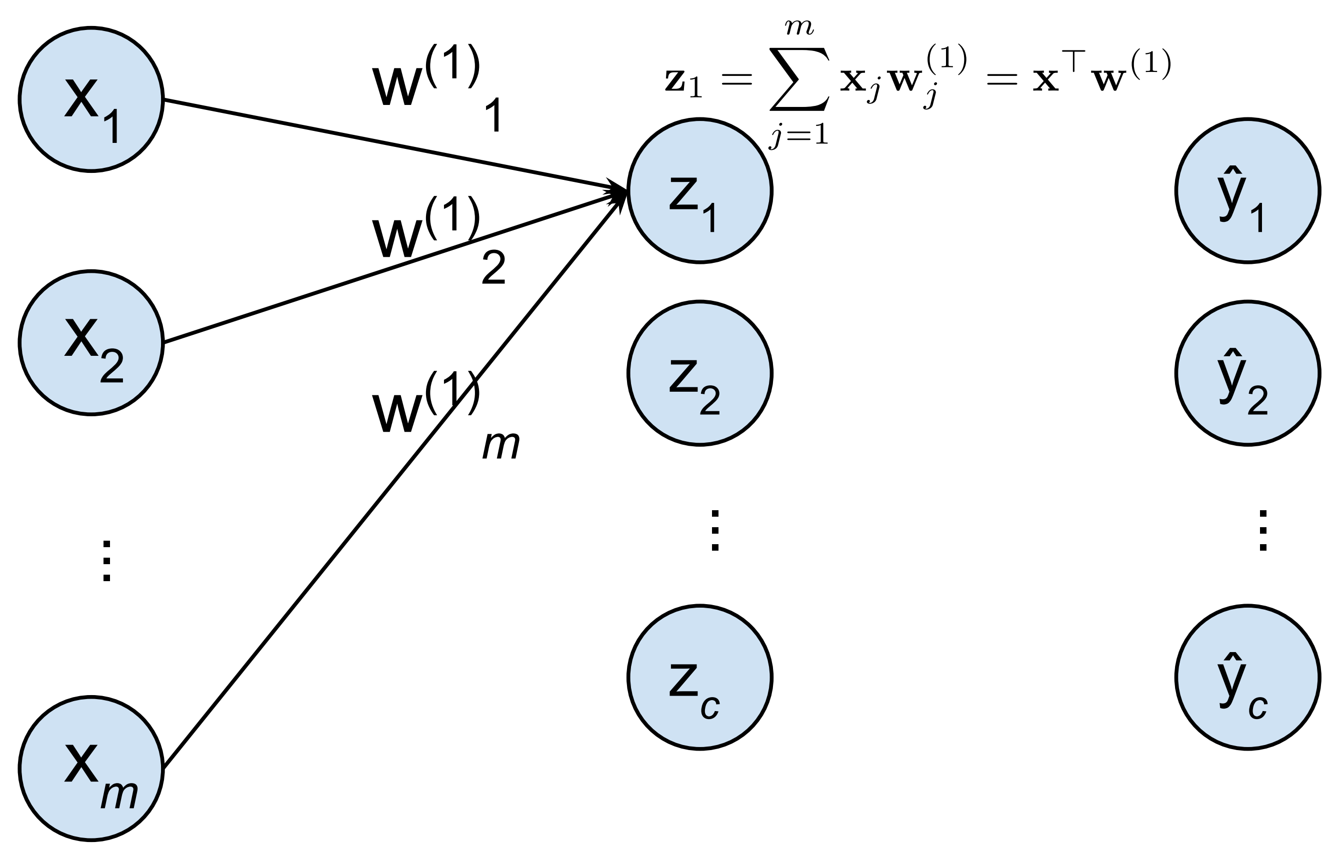
$$\hat{\mathbf{y}}_k = \exp(\mathbf{z}_k)$$

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:

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

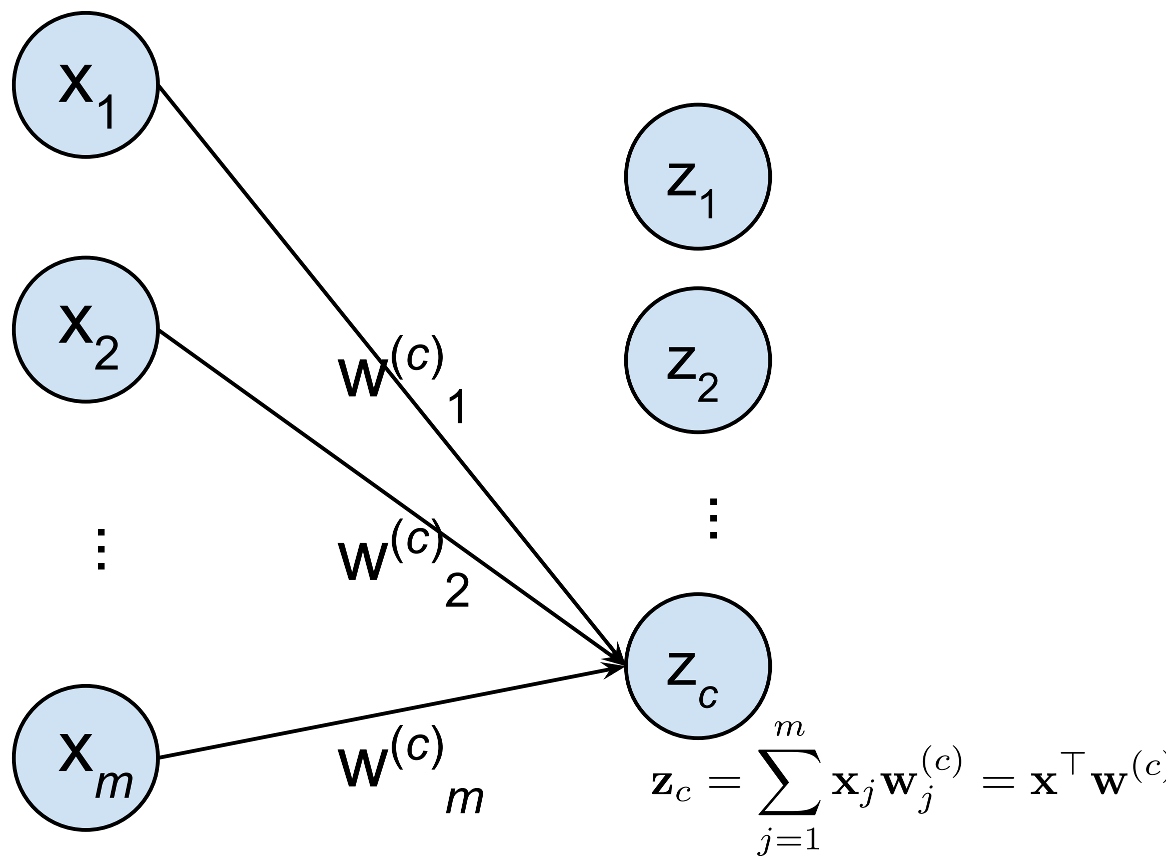
Softmax regression diagram



- With softmax regression, we first compute:

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

Softmax regression diagram



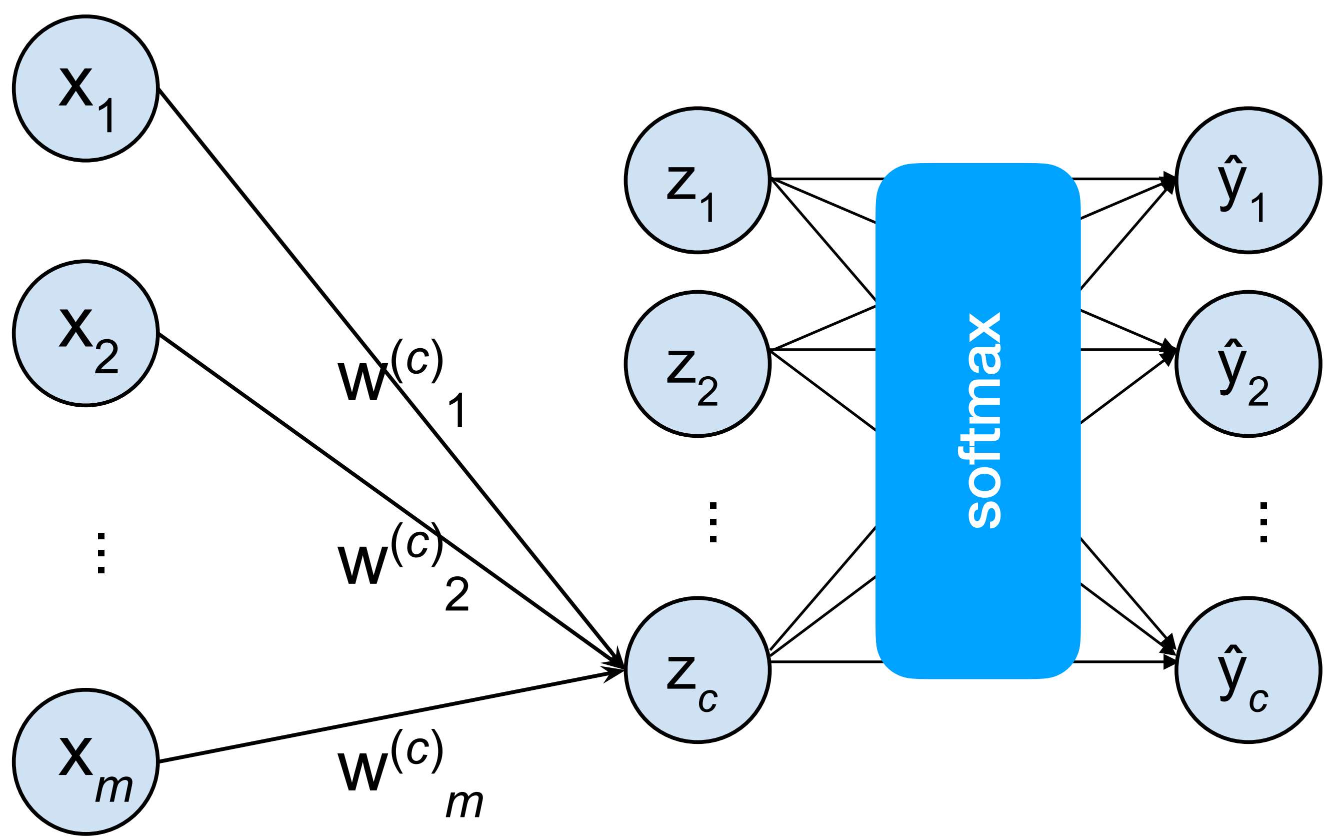
- With softmax regression, we first compute:

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

...

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

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?

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

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

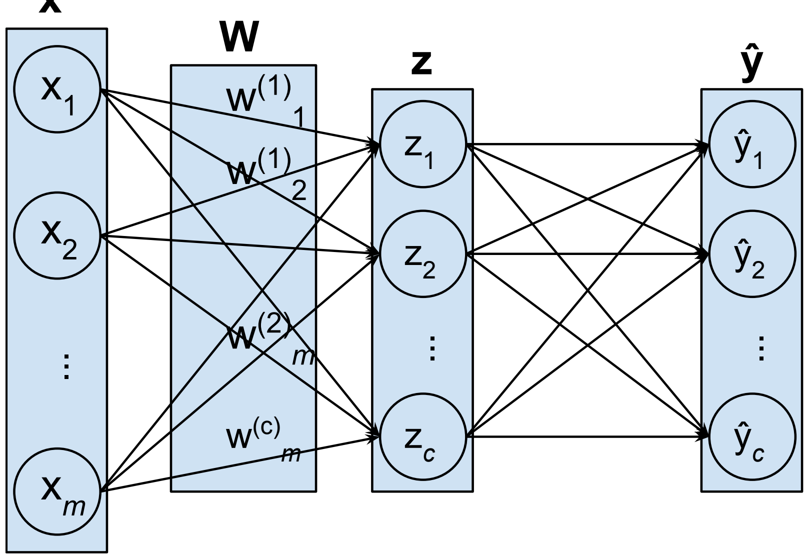
- Let *m*=2, *c*=3.
- Let: $\mathbf{x} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$

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

- Which class will have highest estimated probability?

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

Softmax regression: vectorization



- **x**: column vector, **z**: row vector
- Let $\mathbf{W} = \left[\begin{array}{ccc} | & & | \\ \mathbf{w}^{(1)} & \dots & \mathbf{w}^{(c)} \\ | & & | \end{array} \right]$

- We can compute the "pre-activation scores" **z** for all *c* classes in one-fell-swoop with the equation:

$$\mathbf{z} = \mathbf{x}^\top \mathbf{W}$$

Note: z is a row vector.

Softmax regression: vectorization

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

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

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

Softmax regression: vectorization

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

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

***n* x *c* 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:

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

- 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 **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:

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

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: