CS/DS 541: Class 5

Jacob Whitehill

Exercise

Suppose networks *N*1 and *N*2 are identical in design, are initialized with the same parameter values, and are trained on the same data. However, *N*1 is trained with an additional loss term ($\sum_{j=1}^{m} w_j^2$, where *m* is the total number of weights), whereas *N*2 is trained **without** this additional loss term. What would we expect to see (at the end of training) regarding the training and testing loss that we seek to minimize?

- ○ N1's training loss is higher than N2's, but N1's testing loss is lower than N2's.
- ○ N1's and N2's training loss values are the same, but N2's testing loss is higher than N1's.
- ○ N1's and N2's training loss values are the same, but N1's testing loss is higher than N2's.
- ○ N2's training loss is higher than N1's, but N2's testing loss is lower than N1's.

Logistic regression

Regression vs. classification

- Recall the two main supervised learning cases.
  - **Regression**: predict any real number.
  - **Classification**: choose from a finite set (e.g., {0, 1, 2}).
- So far, we have talked only about the first case.

Binary classification

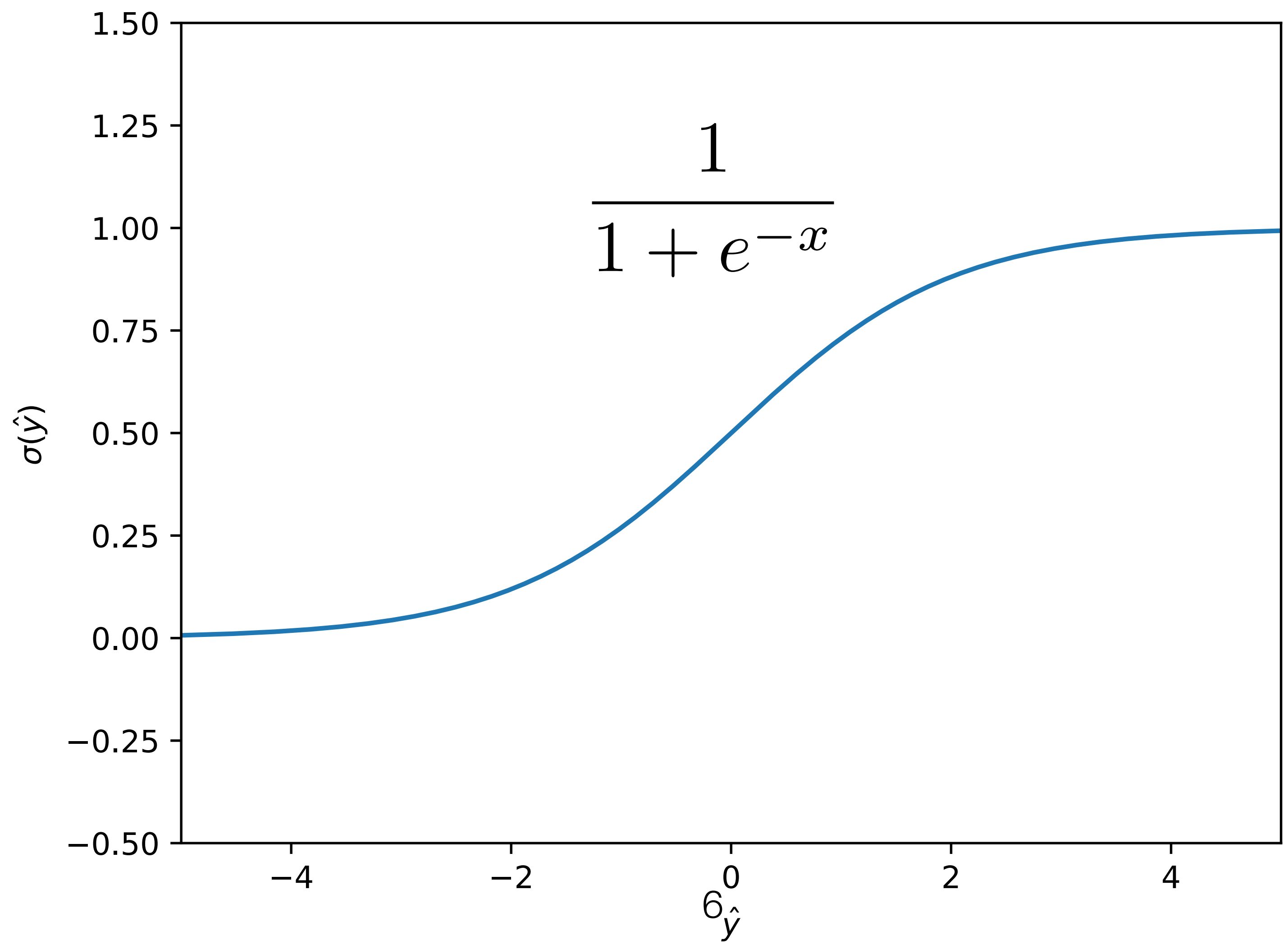
- The simplest classification problem consists of just 2 classes (binary classification), i.e., $y \epsilon \{ 0, 1 \}$.
- One of the simplest and most common classification techniques is **logistic regression**.
- Logistic regression is similar to linear regression but also uses a sigmoidal "squashing" function to ensure that $\hat{y} \epsilon (0, 1)$.

Jacob Whitehill, WPI

Sigmoid: a "squashing" function

- A sigmoid function is an "s"-shaped, monotonically increasing and bounded function.

- Here is the **logistic sigmoid** function σ:



Logistic sigmoid

- The logistic sigmoid function σ has some nice properties:
    - $\sigma(-z) = 1 - \sigma(z)$

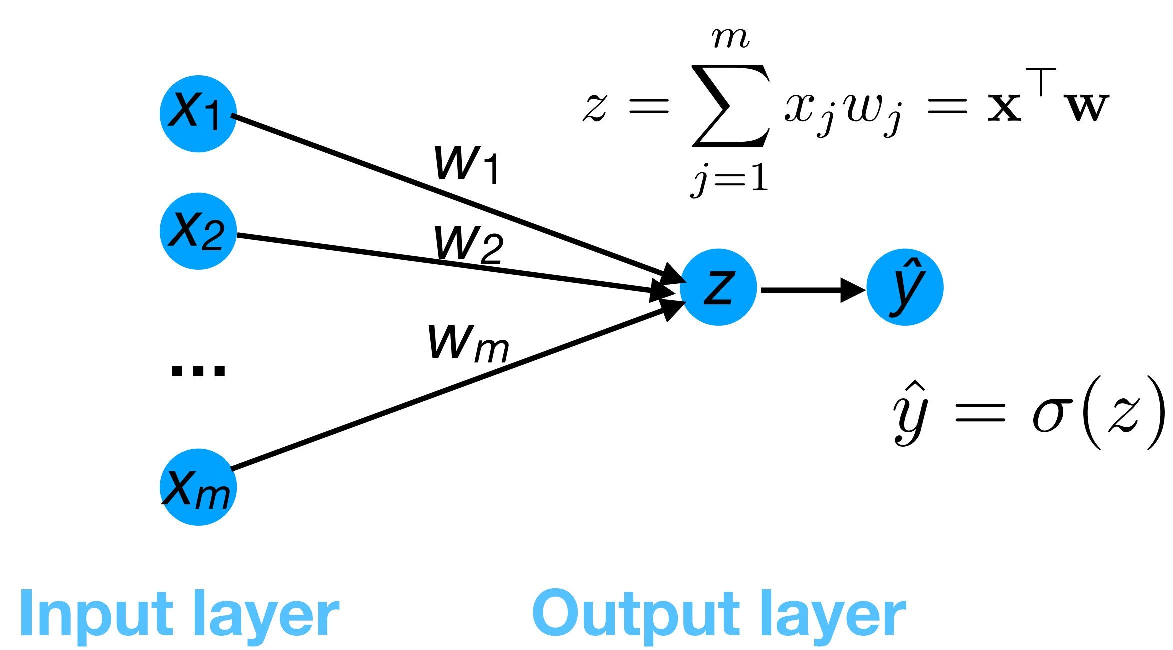
Logistic sigmoid

- The logistic sigmoid function σ has some nice properties:
  - $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

Logistic regression

- With logistic regression, our predictions are defined as:

$$\hat{y} = \sigma\left(\mathbf{x}^\top \mathbf{w}\right)$$

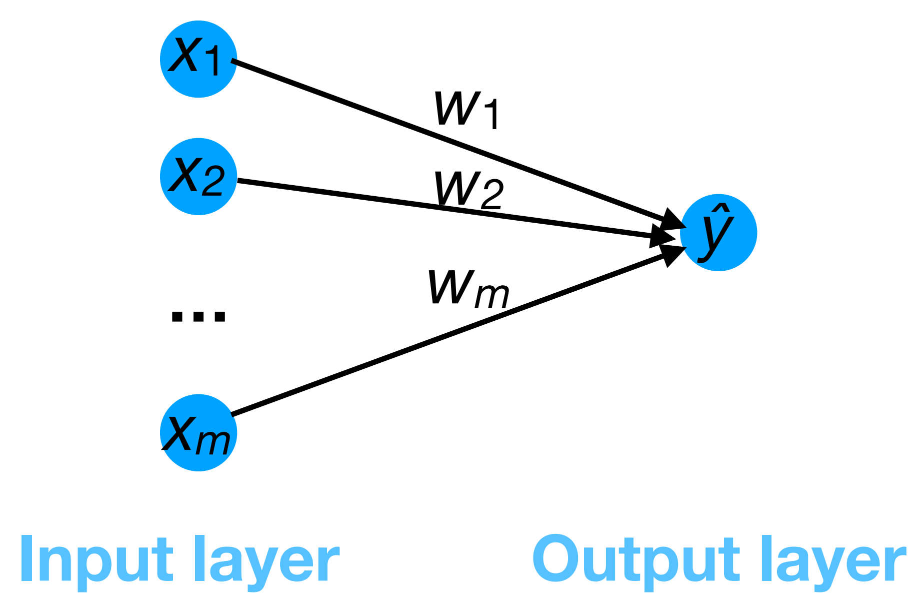


Jacob Whitehill, WPI

Logistic regression

- With logistic regression, our predictions are defined as:

$$\hat{y} = \sigma\left(\mathbf{x}^\top \mathbf{w}\right)$$



Logistic regression

- With logistic regression, our predictions are defined as:

$$\hat{y} = \sigma\left(\mathbf{x}^\top \mathbf{w}\right)$$

- Hence, they are forced to be in (0,1).

- For classification, we can interpret the real-valued outputs as probabilities that express how confident we are in a prediction, e.g.:

    - $\hat{y}$=0.95: very confident that a face contains a smile.

    - $\hat{y}$=0.58: not very confident that a face contains a smile.

Logistic regression

- How to train? Unlike linear regression, logistic regression has no analytical (closed-form) solution.
  - We can use (stochastic) gradient descent instead.
  - We have to apply the **chain-rule of differentiation** to handle the sigmoid function.

Gradient descent for logistic regression

- Let's compute the gradient of f_{MSE} for logistic regression.
- For simplicity, we'll consider just a single example:

$$
\begin{aligned}
f_{\mathrm{MSE}}(\mathbf{w}) &= \frac{1}{2}(\hat{y} - y)^2 \\
&= \frac{1}{2}\left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right)^2 \\
\nabla_{\mathbf{w}} f_{\mathrm{MSE}}(\mathbf{w}) &= \nabla_{\mathbf{w}} \left[\frac{1}{2}\left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right)^2\right] \\
&=
\end{aligned}
$$

Gradient descent for logistic regression

- Let's compute the gradient of f_{MSE} for logistic regression.
- For simplicity, we'll consider just a single example:

$$
\begin{aligned}
f_{\text{MSE}}(\mathbf{w}) &= \frac{1}{2}(\hat{y} - y)^2 \\
&= \frac{1}{2}\left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right)^2 \\
\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{w}) &= \nabla_{\mathbf{w}} \left[\frac{1}{2}\left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right)^2\right] \\
&= \left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right)
\end{aligned}
$$

Gradient descent for logistic regression

- Let's compute the gradient of f_{MSE} for logistic regression.
- For simplicity, we'll consider just a single example:

$$
\begin{aligned}
f_{\mathrm{MSE}}(\mathbf{w}) &= \frac{1}{2}(\hat{y} - y)^2 \\
&= \frac{1}{2}\left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right)^2 \\
\nabla_{\mathbf{w}} f_{\mathrm{MSE}}(\mathbf{w}) &= \nabla_{\mathbf{w}} \left[\frac{1}{2}\left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right)^2\right] \\
&= \left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right) \sigma(\mathbf{x}^\top \mathbf{w}) \left(1 - \sigma(\mathbf{x}^\top \mathbf{w})\right)
\end{aligned}
$$

Jacob Whitehill, WPI

Gradient descent for logistic regression

- Let's compute the gradient of f_{MSE} for logistic regression.
- For simplicity, we'll consider just a single example:

$$
\begin{aligned}
f_{\text{MSE}}(\mathbf{w}) &= \frac{1}{2}(\hat{y} - y)^2 \\
&= \frac{1}{2}\left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right)^2 \\
\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{w}) &= \nabla_{\mathbf{w}} \left[\frac{1}{2}\left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right)^2\right] \\
&= \mathbf{x}\left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right) \sigma(\mathbf{x}^\top \mathbf{w}) \left(1 - \sigma(\mathbf{x}^\top \mathbf{w})\right)
\end{aligned}
$$

Gradient descent for logistic regression

- Let's compute the gradient of f_{MSE} for logistic regression.
- For simplicity, we'll consider just a single example:

$$
\begin{aligned}
f_{\text{MSE}}(\mathbf{w}) &= \frac{1}{2}(\hat{y} - y)^2 \\
&= \frac{1}{2}\left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right)^2 \\
\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{w}) &= \nabla_{\mathbf{w}} \left[\frac{1}{2}\left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right)^2\right] \\
&= \mathbf{x}\left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right)\sigma(\mathbf{x}^\top \mathbf{w})\left(1 - \sigma(\mathbf{x}^\top \mathbf{w})\right) \\
&= \mathbf{x}\,(\hat{y} - y)\,\hat{y}\,(1 - \hat{y})
\end{aligned}
$$

Gradient descent for logistic regression

- Let's compute the gradient of f_{MSE} for logistic regression.
- For simplicity, we'll consider just a single example:

$$
\begin{aligned}
f_{\text{MSE}}(\mathbf{w}) &= \frac{1}{2}(\hat{y} - y)^2 \\
&= \frac{1}{2}\left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right)^2 \\
\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{w}) &= \nabla_{\mathbf{w}} \left[\frac{1}{2}\left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right)^2\right] \\
&= \mathbf{x}\left(\sigma(\mathbf{x}^\top \mathbf{w}) - y\right)\sigma(\mathbf{x}^\top \mathbf{w})\left(1 - \sigma(\mathbf{x}^\top \mathbf{w})\right) \\
&= \mathbf{x}\left(\hat{y} - y\right)\boxed{\hat{y}\left(1 - \hat{y}\right)}
\end{aligned}
$$

Notice the extra multiplicative terms compared to the gradient for *linear* regression: x($\hat{y}$ - y)

Attenuated gradient

- What if the weights **w** are initially chosen badly, so that $\hat{y}$ is very close to 1, even though $y = 0$ (or vice-versa)?
    - Then $\hat{y}(1 - \hat{y})$ is close to 0.
- In this case, the gradient:

$$\nabla_{\mathbf{w}} f_{\mathrm{MSE}}(\mathbf{w}) = \mathbf{x}\,(\hat{y} - y)\,\hat{y}\,(1 - \hat{y})$$

  will be very close to 0.

- If the gradient is 0, then no learning will occur!

Different cost function

- For this reason, logistic regression is typically trained using a different cost function from f_{MSE}.

- One particularly well-suited cost function uses logarithms.

- Logarithms and the logistic sigmoid interact well:

$$\frac{\partial}{\partial \mathbf{w}} \left[\log \sigma(\mathbf{x}^\top \mathbf{w}) \right] =$$

Different cost function

- For this reason, logistic regression is typically trained using a different cost function from f_{MSE}.

- One particularly well-suited cost function uses logarithms.

- Logarithms and the logistic sigmoid interact well:

$$\frac{\partial}{\partial \mathbf{w}} \left[\log \sigma(\mathbf{x}^\top \mathbf{w}) \right] = \mathbf{x} \frac{1}{\sigma(\mathbf{x}^\top \mathbf{w})} \sigma(\mathbf{x}^\top \mathbf{w}) \left(1 - \sigma(\mathbf{x}^\top \mathbf{w}) \right)$$

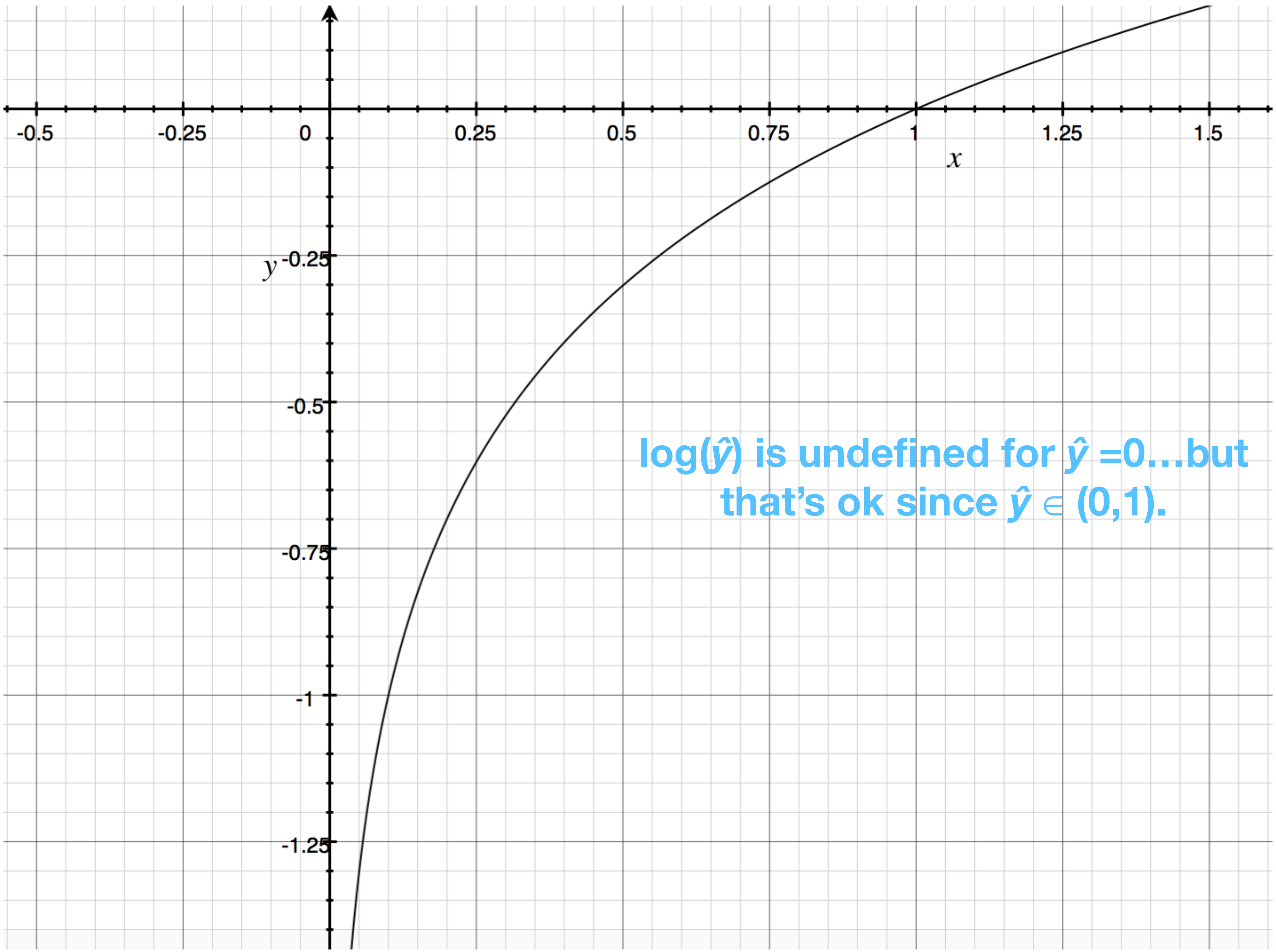
Different cost function

- For this reason, logistic regression is typically trained using a different cost function from f_{MSE}.
- One particularly well-suited cost function uses logarithms.
- Logarithms and the logistic sigmoid interact well:

$$\frac{\partial}{\partial \mathbf{w}} \left[\log \sigma(\mathbf{x}^\top \mathbf{w}) \right] = \mathbf{x} \frac{1}{\sigma(\mathbf{x}^\top \mathbf{w})} \sigma(\mathbf{x}^\top \mathbf{w}) \left(1 - \sigma(\mathbf{x}^\top \mathbf{w})\right)$$
$$= \mathbf{x} \left(1 - \sigma(\mathbf{x}^\top \mathbf{w})\right)$$

The gradient of log(σ) is surprisingly simple.

Logarithm function

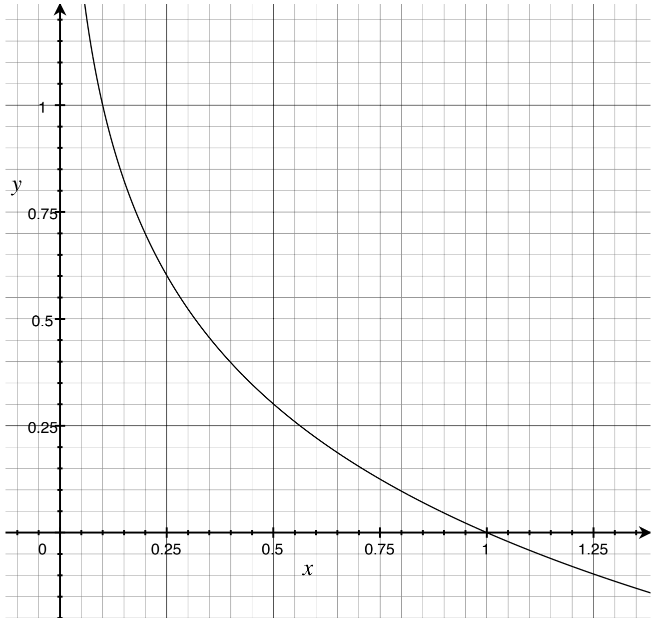


Log loss

- We want to assign a large loss when y=1 but $\hat{y}$=0

- We typically use the **log-loss** for logistic regression:

$$-y \log \hat{y}$$



-log($\hat{y}$)

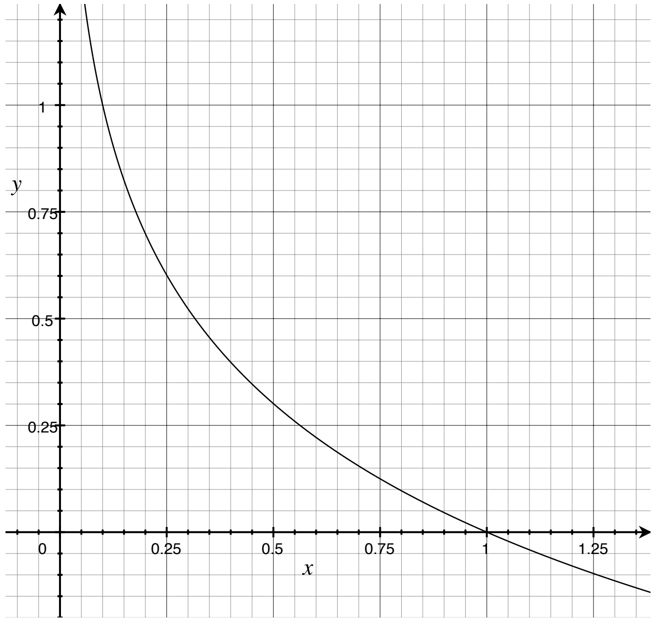
Log loss

- We want to assign a large loss when y=1 but $\hat{y}$=0, and for y=0 but $\hat{y}$=1.

- We typically use the **log-loss** for logistic regression:

$$-y \log \hat{y} - (1-y) \log(1-\hat{y})$$

The *y* or (1-*y*) "selects" which term in the expression is active, based on the ground-truth label.



-log($\hat{y}$)

Gradient descent for logistic regression with log-loss

$$\nabla_{\mathbf{w}} f_{\log}(\mathbf{w}) \quad = \quad \nabla_{\mathbf{w}} \left[- \left(y \log \hat{y} - (1-y) \log(1-\hat{y}) \right) \right]$$

Gradient descent for logistic regression with log-loss

$$
\begin{aligned}
\nabla_{\mathbf{w}} f_{\log}(\mathbf{w}) &= \nabla_{\mathbf{w}} \left[- \left(y \log \hat{y} - (1-y) \log(1-\hat{y}) \right) \right] \\
&= -\nabla_{\mathbf{w}} \left(y \log \sigma(\mathbf{x}^\top \mathbf{w}) + (1-y) \log(1 - \sigma(\mathbf{x}^\top \mathbf{w})) \right)
\end{aligned}
$$

Gradient descent for logistic regression with log-loss

$$
\begin{aligned}
\nabla_{\mathbf{w}} f_{\log}(\mathbf{w}) &= \nabla_{\mathbf{w}} \left[- \left(y \log \hat{y} - (1-y) \log(1-\hat{y}) \right) \right] \\
&= -\nabla_{\mathbf{w}} \left(y \log \sigma(\mathbf{x}^\top \mathbf{w}) + (1-y) \log(1-\sigma(\mathbf{x}^\top \mathbf{w})) \right) \\
&= -\left(y \frac{\mathbf{x}\sigma(\mathbf{x}^\top \mathbf{w})(1-\sigma(\mathbf{x}^\top \mathbf{w}))}{\sigma(\mathbf{x}^\top \mathbf{w})} - (1-y) \frac{\mathbf{x}\sigma(\mathbf{x}^\top \mathbf{w})(1-\sigma(\mathbf{x}^\top \mathbf{w}))}{1-\sigma(\mathbf{x}^\top \mathbf{w})} \right)
\end{aligned}
$$

Gradient descent for logistic regression with log-loss

$$
\begin{aligned}
\nabla_{\mathbf{w}} f_{\log}(\mathbf{w}) &= \nabla_{\mathbf{w}} \left[- \left(y \log \hat{y} - (1-y) \log(1-\hat{y}) \right) \right] \\
&= -\nabla_{\mathbf{w}} \left(y \log \sigma(\mathbf{x}^\top \mathbf{w}) + (1-y) \log(1 - \sigma(\mathbf{x}^\top \mathbf{w})) \right) \\
&= - \left(y \frac{\mathbf{x}\sigma(\mathbf{x}^\top \mathbf{w})(1-\sigma(\mathbf{x}^\top \mathbf{w}))}{\sigma(\mathbf{x}^\top \mathbf{w})} - (1-y) \frac{\mathbf{x}\sigma(\mathbf{x}^\top \mathbf{w})(1-\sigma(\mathbf{x}^\top \mathbf{w}))}{1-\sigma(\mathbf{x}^\top \mathbf{w})} \right) \\
&= - \left(y\mathbf{x}(1-\sigma(\mathbf{x}^\top \mathbf{w})) - (1-y)\mathbf{x}\sigma(\mathbf{x}^\top \mathbf{w}) \right) \\
&= -\mathbf{x} \left(y - y\sigma(\mathbf{x}^\top \mathbf{w}) - \sigma(\mathbf{x}^\top \mathbf{w}) + y\sigma(\mathbf{x}^\top \mathbf{w}) \right) \\
&= -\mathbf{x} \left(y - \sigma(\mathbf{x}^\top \mathbf{w}) \right) \\
&= \mathbf{x}(\hat{y} - y)
\end{aligned}
$$

Same as for linear regression!

Jacob Whitehill, WPI

Linear regression versus logistic regression

	Linear regression	Logistic regression
Primary use	Regression	Classification
Prediction ($\hat{y}$)	$\hat{y} = \mathbf{x}^\top \mathbf{w}$	$\hat{y} = \sigma(\mathbf{x}^\top \mathbf{w})$
Cost/Loss	f_{MSE}	f_{log}
Gradient	$\mathbf{x}(\hat{y} - y)$	$\mathbf{x}(\hat{y} - y)$

- Logistic regression is used primarily for *classification* even though it's called "regression".
- Logistic regression is an instance of a **generalized linear model** — a linear model combined with a **link function** (e.g., logistic sigmoid).
  - In DL, link functions are typically called **activation functions.**

Exercise

Consider a 2-layer neural network that computes the function

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

where **x** is an example, **w** is a vector of weights, *b* is a bias term, and *σ* is the logistic sigmoid function. Suppose **all** the training examples are **positive**, but the testing set can consist of both positive and negative examples. Which of the following claims are true when training this network to minimize the log-loss?

- [ ] The value for b might or might not converge, depending on the exact training examples.
- [ ] The values for w will definitely not converge.
- [ ] The value for b will definitely not converge.
- [ ] The testing loss will converge.
- [ ] The training loss will converge.
- [ ] The values for w might or might not converge, depending on the exact training examples.

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)

Jacob Whitehill, WPI

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.

Jacob Whitehill, WPI

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 **`a`**, **`b`**, **`a`**.

Example: 2 classes

- Suppose we have a dataset of 3 examples, where the ground-truth class labels are **`a`**, **`b`**, **`a`**.

- 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 $\mathbf{y}$ is 1; the others are 0.

Example: 2 classes

- Suppose we have a dataset of 3 examples, where the ground-truth class labels are **`a`**, **`b`**, **`a`**.

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

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

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

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

This "slot" is for class `a`.

- 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 **`a`**, **`b`**, **`a`**.

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

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

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

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

This "slot" is for class `b`.

- 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:

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

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

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

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

- 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 b.

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

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

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

Softmax activation function

- Logistic regression outputs a *scalar* label $\hat{y}$ representing the probability that the label is positive.
    - We needed just a single weight vector $\mathbf{w}$, so that $\hat{y} = \sigma(\mathbf{x}^\top \mathbf{w})$.

Softmax activation function

- Logistic regression outputs a *scalar* label $\hat{y}$ representing the probability that the label is positive.
    - We needed just a single weight vector $\mathbf{w}$, so that $\hat{y} = \sigma(\mathbf{x}^\top \mathbf{w})$.
- Softmax regression outputs a c-*vector* representing the probabilities that the label is k=1, …, c.
    - We need c different vectors of weights $\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(c)}$.
    - Weight vector $\mathbf{w}^{(k)}$ computes how much input $\mathbf{x}$ "agrees" with class k.

Softmax activation function

- With softmax regression, we first compute:

$$z_1 = \mathbf{x}^\top \mathbf{w}^{(1)}$$
$$z_2 = \mathbf{x}^\top \mathbf{w}^{(2)}$$
$$\vdots$$
$$z_c = \mathbf{x}^\top \mathbf{w}^{(c)}$$

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

Softmax activation function

- With softmax regression, we first compute:

$$z_1 = \mathbf{x}^\top \mathbf{w}^{(1)}$$

$$z_2 = \mathbf{x}^\top \mathbf{w}^{(2)}$$

$$\vdots$$

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

- We then **normalize** across all *c* classes so that:
   1. Each output $\hat{y}_k$ is non-negative.
   2. The sum of $\hat{y}_k$ over all *c* classes is 1.

Jacob Whitehill, WPI

Normalization of the $\hat{y}_k$

1. To enforce non-negativity, we can exponentiate each z_k:

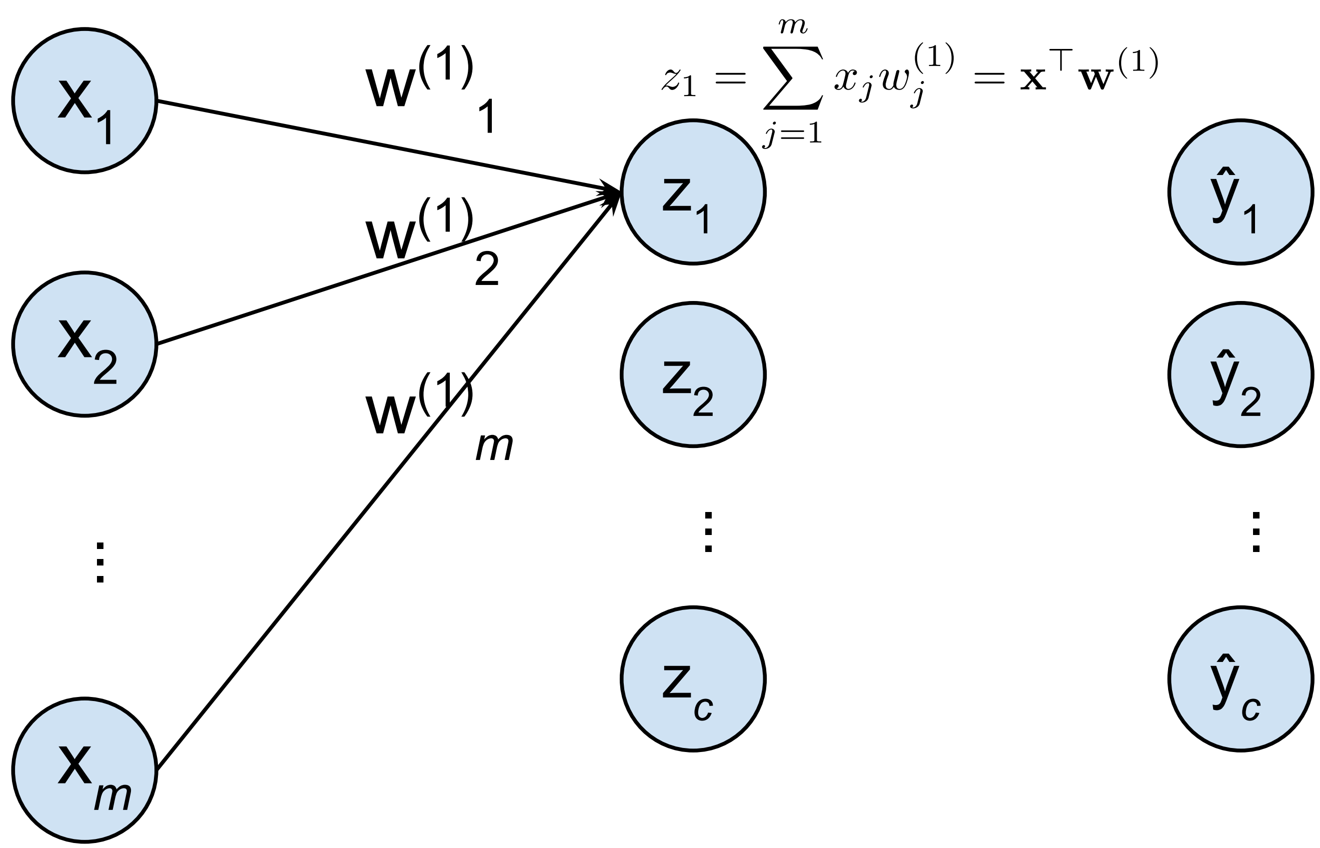
$$\hat{y}_k = \exp(z_k)$$

Normalization of the $\hat{y}_k$

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

$$\hat{y}_k = \frac{\exp(z_k)}{\sum_{k'=1}^{c} \exp(z_{k'})}$$

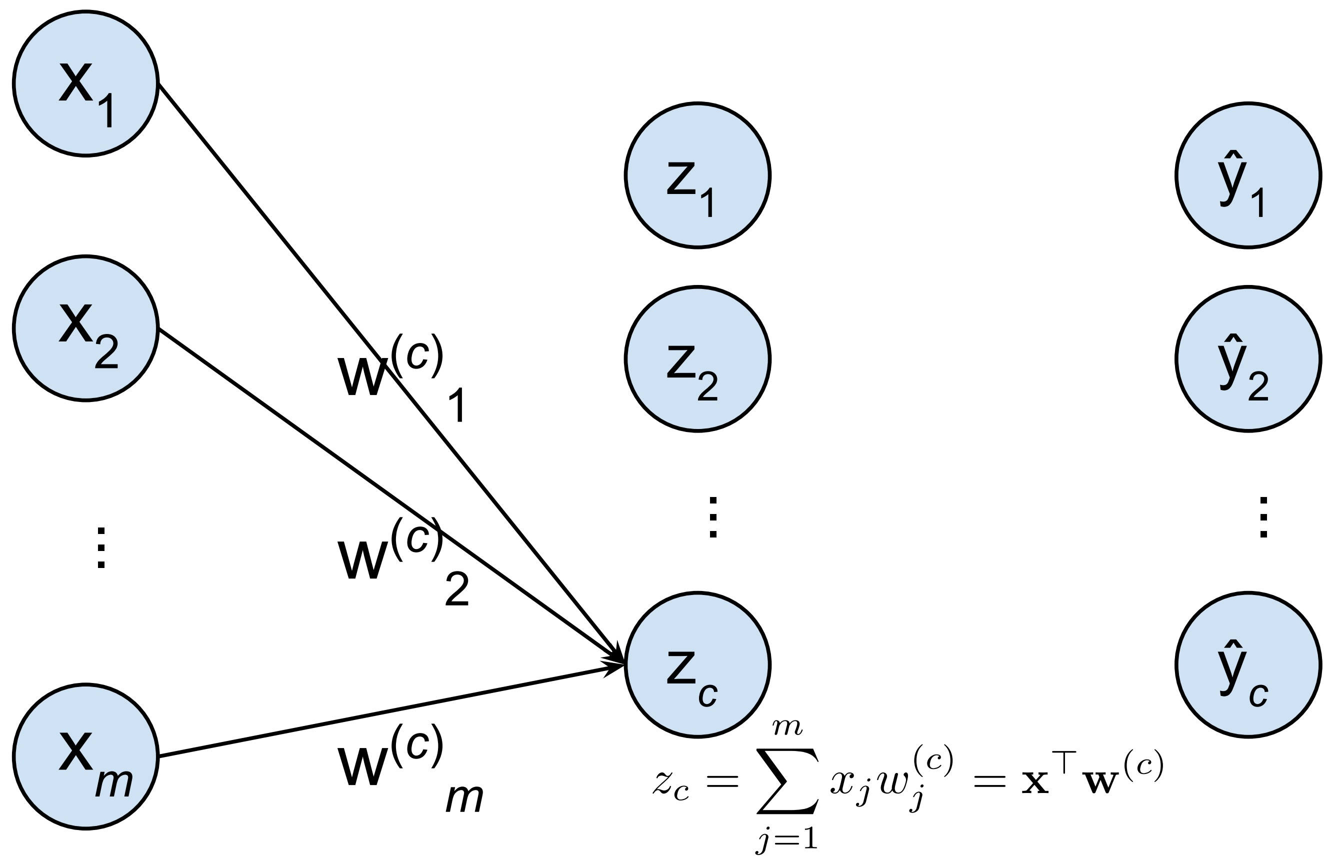
Softmax regression diagram



- With softmax regression, we first compute:

$$z_1 = \mathbf{x}^\top \mathbf{w}^{(1)}$$

Softmax regression diagram



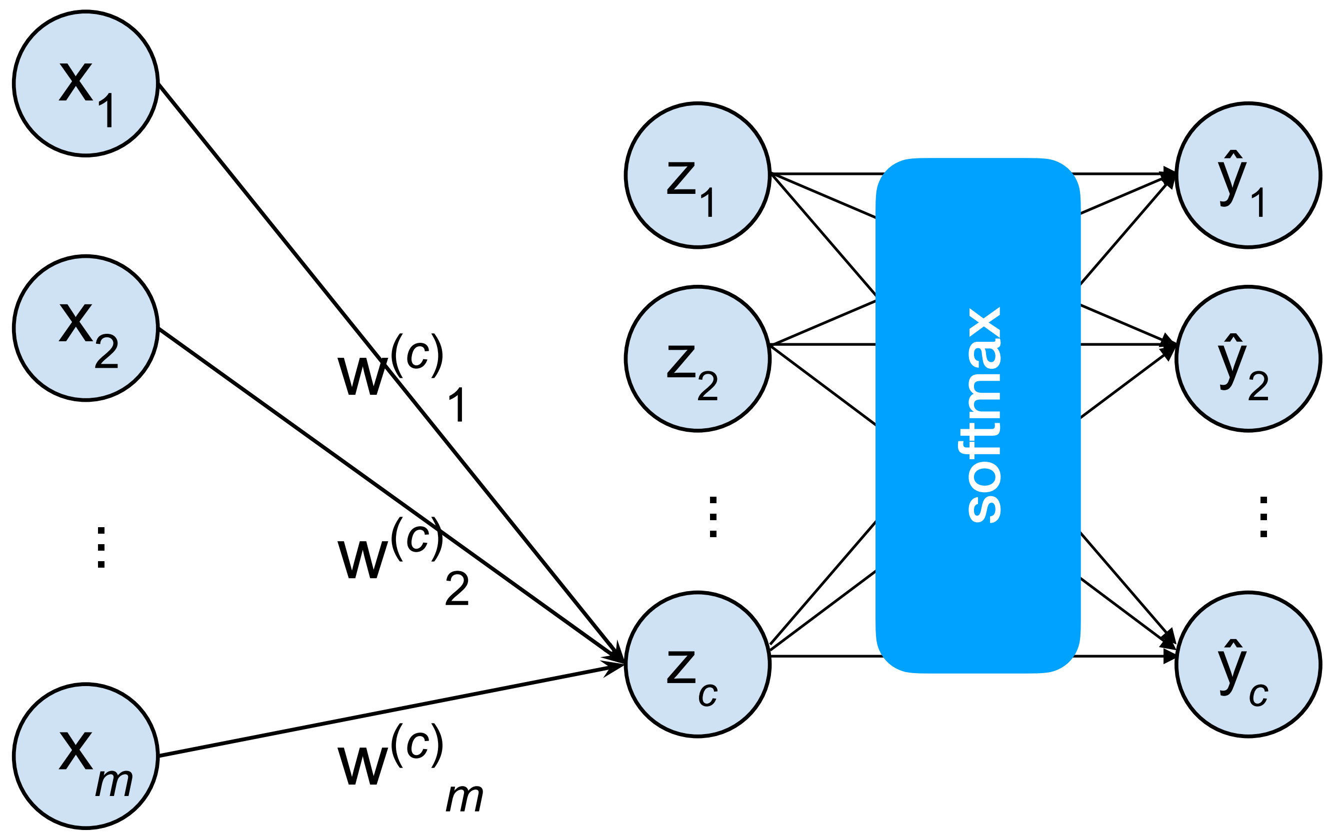
- With softmax regression, we first compute:

$$z_1 = \mathbf{x}^\top \mathbf{w}^{(1)}$$

$$\vdots$$

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

Softmax regression diagram



$$\hat{y}_k = \frac{\exp(z_k)}{\sum_{k'=1}^{c} \exp(z_{k'})}$$

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

$$\hat{y}_k = P(y = k \mid \mathbf{x}, \mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(c)}) = \frac{\exp(z_k)}{\sum_{k'=1}^{c} \exp(z_{k'})}$$

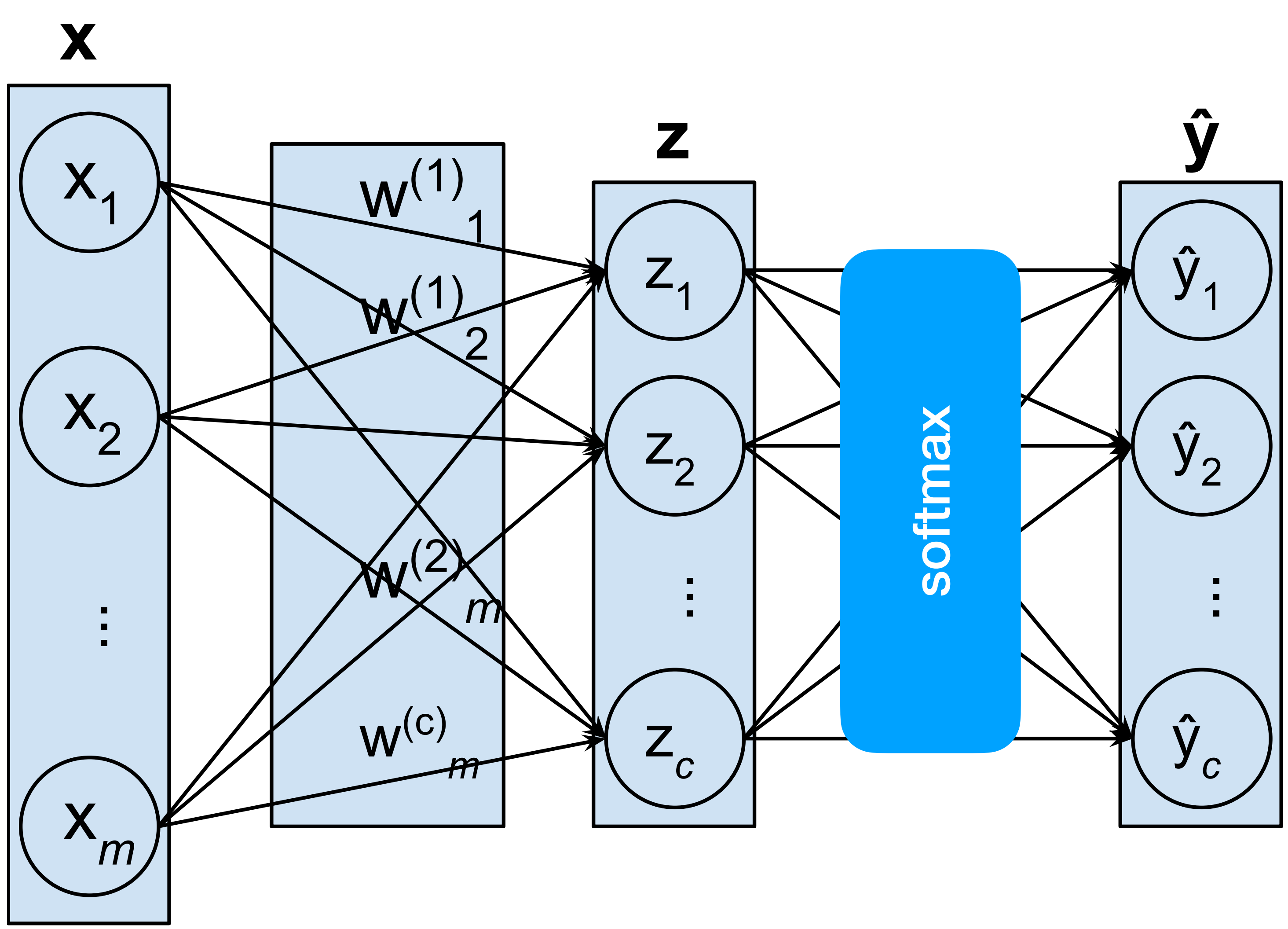
Cross-entropy loss

- We need a loss function that can support $c \geq 2$ classes.
- We will use the **cross-entropy** (CE) loss:

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

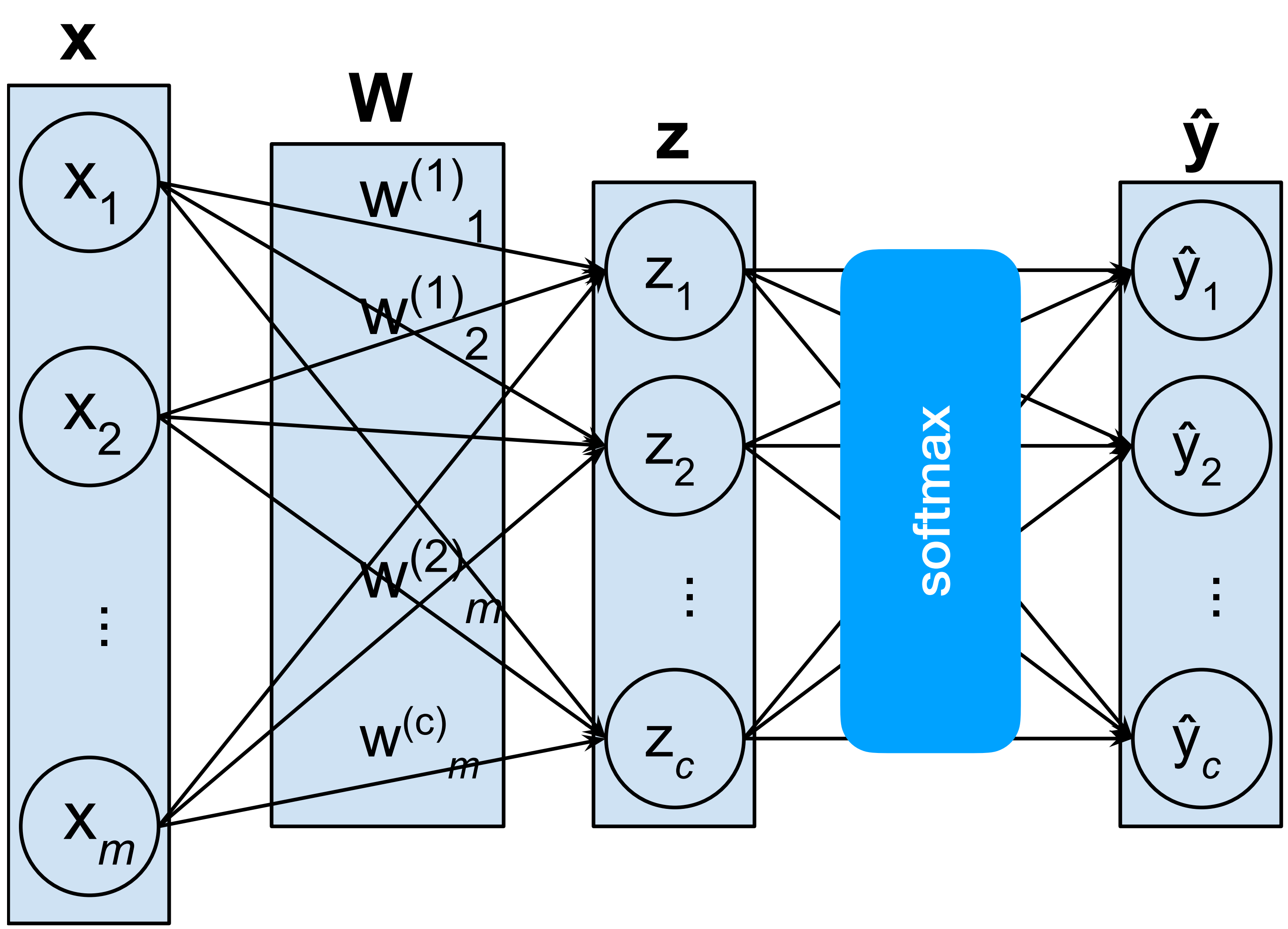
- Note that the CE loss subsumes the log-loss for $c=2$.

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 an (m x c) matrix $\mathbf{W}$.

Softmax regression: vectorization

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

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

Softmax regression: vectorization

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

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

- 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 $\mathbf{w}^{(k)}$:

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

- 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…

Jacob Whitehill, WPI

Gradient descent for softmax regression

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

$$\mathbf{Y} = \begin{bmatrix} y_1^{(1)} & \dots & y_c^{(1)} \\ & \vdots & \\ y_1^{(n)} & \dots & y_c^{(n)} \end{bmatrix}$$

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

Gradient descent for softmax regression

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

$$\mathbf{Y} = \begin{bmatrix} y_1^{(1)} & \dots & y_c^{(1)} \\ & \vdots & \\ y_1^{(n)} & \dots & y_c^{(n)} \end{bmatrix}$$

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

Jacob Whitehill, WPI

Gradient descent for softmax regression

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

$$\mathbf{Y} = \begin{bmatrix} y_1^{(1)} & \dots & y_c^{(1)} \\ & \vdots & \\ y_1^{(n)} & \dots & y_c^{(n)} \end{bmatrix} \qquad \hat{\mathbf{Y}} = \begin{bmatrix} \hat{y}_1^{(1)} & \dots & \hat{y}_c^{(1)} \\ & \vdots & \\ \hat{y}_1^{(n)} & \dots & \hat{y}_c^{(n)} \end{bmatrix}$$

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:

$$\mathbf{Y} = \begin{bmatrix} y_1^{(1)} & \dots & y_c^{(1)} \\ & \vdots & \\ y_1^{(n)} & \dots & y_c^{(n)} \end{bmatrix} \qquad \hat{\mathbf{Y}} = \begin{bmatrix} \hat{y}_1^{(1)} & \dots & \hat{y}_c^{(1)} \\ & \vdots & \\ \hat{y}_1^{(n)} & \dots & \hat{y}_c^{(n)} \end{bmatrix}$$

- 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:

$$\mathbf{Y} = \begin{bmatrix} y_1^{(1)} & \dots & y_c^{(1)} \\ & \vdots & \\ y_1^{(n)} & \dots & y_c^{(n)} \end{bmatrix} \qquad \hat{\mathbf{Y}} = \begin{bmatrix} \hat{y}_1^{(1)} & \dots & \hat{y}_c^{(1)} \\ & \vdots & \\ \hat{y}_1^{(n)} & \dots & \hat{y}_c^{(n)} \end{bmatrix}$$

- 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:

$$\mathbf{Y} = \begin{bmatrix} y_1^{(1)} & \dots & y_c^{(1)} \\ & \vdots & \\ y_1^{(n)} & \dots & y_c^{(n)} \end{bmatrix} \qquad \hat{\mathbf{Y}} = \begin{bmatrix} \hat{y}_1^{(1)} & \dots & \hat{y}_c^{(1)} \\ & \vdots & \\ \hat{y}_1^{(n)} & \dots & \hat{y}_c^{(n)} \end{bmatrix}$$

- 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).

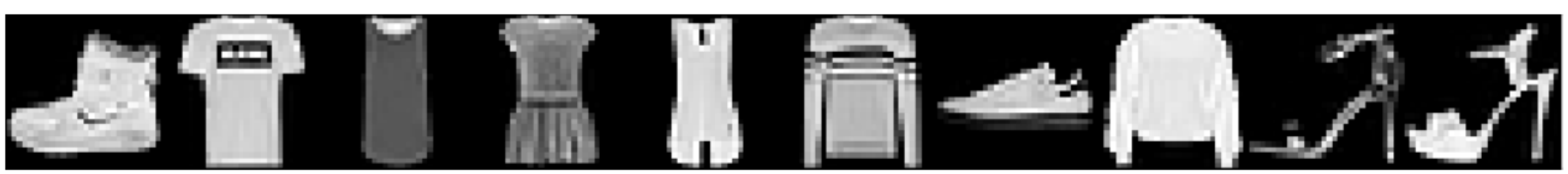
Bias term

- Like in linear regression, softmax regression also benefits from the use of a bias term.
- Instead of a scalar b, we have a bias vector $\mathbf{b}$ with c dimensions (one for each class).
- You will derive the gradient update for $\mathbf{b}$ as part of homework 3.

Softmax regression demo

- In HW3, you will apply softmax regression to train a **handwriting recognition system** that can recognize all 10 digits (0-9).

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



Exercise

- An important hyperparameter when performing (stochastic) gradient descent is the learning rate ϵ.

```
def SGD (X, y, eps):
  ...
  return finalParams, finalLoss
```

One simple approach to optimizing ϵ is to perform a grid-search, i.e., create a finite set of values to choose from, and select the best value from the set based on accuracy on a validation set. As an alternative approach, consider how ϵ itself might be optimized using gradient descent on the **`SGD`** function itself, i.e., computing the derivative of **`SGD`** with respect to ϵ and then adjusting ϵ to reduce the **`finalLoss`**. Either describe how this alternative approach would work, or describe why it would not work.