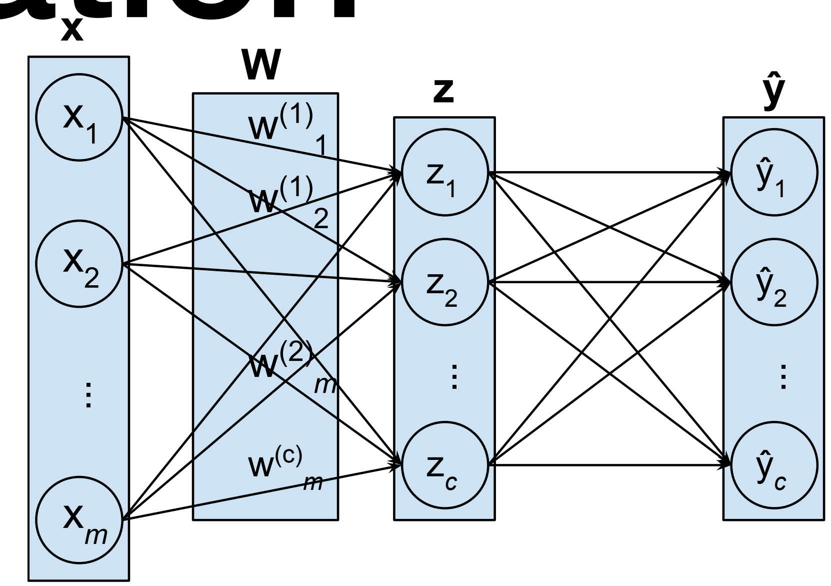
CS 4342: Class 9

Jacob Whitehill

Softmax regression (aka multinomial logistic regression)

Softmax regression: vectorization



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

- 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:

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

- 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 **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:

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

How far the guesses are from ground-truth.

Gradient descent for softmax regression

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

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

- Then we can compute all *c* gradient vectors as:

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

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:

Stochastic gradient descent (SGD)

Gradient descent

- With gradient descent, we only update the weights after scanning the *entire* training set.
  - This is slow.
- If the training set contains 60K examples (like in MNIST), then the gradient is an *average* over 60K images.
  - How much would the gradient really change if we just used, say, 30K images? 15K images? 128 images?

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

Average over entire training set.

Stochastic gradient descent

- This is the idea behind **stochastic gradient descent** (SGD):
  - Randomly sample a small ($\ll n$) **mini-batch** (or sometimes just **batch**) of training examples.
  - Estimate the gradient on just the mini-batch.
  - Update weights based on *mini-batch* gradient estimate.
  - Repeat.

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
    - An **epoch** is a single pass through the entire training set.
- Procedure:
    1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.
- Procedure:
  1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.
  2. Randomize the order of the examples in the training set.

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.
- Procedure:
  1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.
  2. Randomize the order of the examples in the training set.
  3. For *e* = 0 to numEpochs:

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.
- Procedure:
  1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.
  2. Randomize the order of the examples in the training set.
  3. For *e* = 0 to numEpochs:
     I. For *i* = 0 to $(\lceil n/\tilde{n} \rceil - 1)$ (one epoch):

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.

- Procedure:
  1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.
  2. Randomize the order of the examples in the training set.
  3. For *e* = 0 to numEpochs:
     I. For *i* = 0 to $(\lceil n/\tilde{n} \rceil - 1)$ (one epoch):
        A. Select a mini-batch $\mathcal{J}$ containing the next $\tilde{n}$ examples.

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.

- Procedure:
  1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.
  2. Randomize the order of the examples in the training set.
  3. For *e* = 0 to numEpochs:
     I. For *i* = 0 to $(\lceil n/\tilde{n} \rceil - 1)$ (one epoch):
        A. Select a mini-batch $\mathcal{J}$ containing the next $\tilde{n}$ examples.
        B. Compute the gradient on this mini-batch: $\frac{1}{\tilde{n}} \sum_{i \in \mathcal{J}} \nabla_{\mathbf{W}} f(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}; \mathbf{W})$

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.

- Procedure:
  1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.
  2. Randomize the order of the examples in the training set.
  3. For *e* = 0 to numEpochs:
     I. For *i* = 0 to $(\lceil n/\tilde{n} \rceil - 1)$ (one epoch):
        A. Select a mini-batch $\mathcal{J}$ containing the next $\tilde{n}$ examples.
        B. Compute the gradient on this mini-batch: $\frac{1}{\tilde{n}} \sum_{i \in \mathcal{J}} \nabla_{\mathbf{W}} f(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}; \mathbf{W})$
        C. Update the weights based on the current mini-batch gradient.

SGD versus GD: example

- Suppose our training set contains n=8 examples.
- Here is how *regular* gradient descent would proceed:
  - **Initialize weights $\mathbf{w}^{(0)}$ to random values.**

Training examples
1
2
3
4
5
6
7
8

SGD versus GD: example

- Suppose our training set contains n=8 examples.
- Here is how *regular* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - For each round:
    - **Compute gradient on all *n* examples.**

Training examples
1
2
3
4
5
6
7
8

SGD versus GD: example

- Suppose our training set contains n=8 examples.
- Here is how *regular* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - For each round:
    - Compute gradient on all n examples.
    - **Update weights**: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \nabla_{\mathbf{w}} f$

Training examples
1
2
3
4
5
6
7
8

SGD versus GD: example

- Suppose our training set contains n=8 examples.
- Here is how *regular* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - For each round:
    - **Compute gradient on all *n* examples.**
    - Update weights: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \nabla_{\mathbf{w}} f$

Training examples
1
2
3
4
5
6
7
8

Jacob Whitehill, WPI

SGD versus GD: example

- Suppose our training set contains n=8 examples.
- Here is how *regular* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - For each round:
    - Compute gradient on all n examples.
    - **Update weights**: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \nabla_{\mathbf{w}} f$

Training examples

1
2
3
4
5
6
7
8

SGD versus GD: example

- Suppose our training set contains *n*=8 examples with $\tilde{n} = 2$.
- Here is how *stochastic* gradient descent would proceed:
  - **Initialize weights $\mathbf{w}^{(0)}$ to random values.**

Training examples
1
2
3
4
5
6
7
8

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n} = 2$.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - **Randomize the order of the training data.**

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n} = 2$.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): e=1
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - **Compute gradient on next $\tilde{n}$ examples.**

Training examples
4
1
3
5
7
6
8
2

Jacob Whitehill, WPI

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n} = 2$.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): **e=1**
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - Compute gradient on next $\tilde{n}$ examples.
      - **Update weights**: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n}$= 2.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): e=1
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - **Compute gradient on next $\tilde{n}$ examples.**
      - Update weights: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n}$= 2.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): e=1
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - Compute gradient on next $\tilde{n}$ examples.
      - **Update weights**: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains *n*=8 examples with $\tilde{n}$= 2.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (*e*=1, …, *E*): e=1
    - For each round (*r*=1, …, $\lceil n/\tilde{n} \rceil$):
      - **Compute gradient on next $\tilde{n}$ examples.**
      - Update weights: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n}$= 2.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): e=1
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - Compute gradient on next $\tilde{n}$ examples.
      - **Update weights**: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n}$= 2.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): e=1
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - **Compute gradient on next $\tilde{n}$ examples.**
      - Update weights: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n}$= 2.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): **e=1**
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - Compute gradient on next $\tilde{n}$ examples.
      - **Update weights**: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n} = 2$.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): e=2
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - **Compute gradient on next $\tilde{n}$ examples.**
      - Update weights: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n}$= 2.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): e=2
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - Compute gradient on next $\tilde{n}$ examples.
      - Update weights: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

…

Training examples
4
1
3
5
7
6
8
2

Stochastic gradient descent

- Despite “noise” (statistical inaccuracy) in the mini-batch gradient estimates, we will still converge to local minimum.
- Training can be much faster than regular gradient descent because we adjust the weights *many times* per epoch.

Logistic regression is a special case of softmax regression

Logistic regression is a special case of softmax regression

$$\hat{\mathbf{y}}_1 = \frac{\exp \mathbf{z}_1}{\sum_{k'=1}^{2} \exp \mathbf{z}_{k'}}$$

- Suppose c=2. Then even though there are two weight vectors $\mathbf{w}^{(1)}$, $\mathbf{w}^{(2)}$ in softmax regression, they are equivalent to just a single weight vector **w**:

Logistic regression is a special case of softmax regression

- Suppose c=2. Then even though there are two weight vectors $\mathbf{w}^{(1)}$, $\mathbf{w}^{(2)}$ in softmax regression, they are equivalent to just a single weight vector $\mathbf{w}$:

$$\begin{aligned}\hat{\mathbf{y}}_1 &= \frac{\exp \mathbf{z}_1}{\sum_{k'=1}^{2} \exp \mathbf{z}_{k'}} \\ &= \frac{\exp \mathbf{z}_1}{\exp \mathbf{z}_1 + \exp \mathbf{z}_2}\end{aligned}$$

Logistic regression is a special case of softmax regression

- Suppose c=2. Then even though there are two weight vectors $\mathbf{w}^{(1)}$, $\mathbf{w}^{(2)}$ in softmax regression, they are equivalent to just a single weight vector $\mathbf{w}$:

$$
\begin{aligned}
\hat{\mathbf{y}}_1 &= \frac{\exp \mathbf{z}_1}{\sum_{k'=1}^{2} \exp \mathbf{z}_{k'}} \\
&= \frac{\exp \mathbf{z}_1}{\exp \mathbf{z}_1 + \exp \mathbf{z}_2} \\
&= \frac{1}{1 + \exp \mathbf{z}_2 / \exp \mathbf{z}_1}
\end{aligned}
$$

Logistic regression is a special case of softmax regression

- Suppose c=2. Then even though there are two weight vectors $\mathbf{w}^{(1)}$, $\mathbf{w}^{(2)}$ in softmax regression, they are equivalent to just a single weight vector $\mathbf{w}$:

$$
\begin{aligned}
\hat{\mathbf{y}}_1 &= \frac{\exp \mathbf{z}_1}{\sum_{k'=1}^{2} \exp \mathbf{z}_{k'}} \\
&= \frac{\exp \mathbf{z}_1}{\exp \mathbf{z}_1 + \exp \mathbf{z}_2} \\
&= \frac{1}{1 + \exp \mathbf{z}_2 / \exp \mathbf{z}_1} \\
&= \frac{1}{1 + \exp \left(\mathbf{z}_2 - \mathbf{z}_1 \right)}
\end{aligned}
$$

Logistic regression is a special case of softmax regression

- Suppose c=2. Then even though there are two weight vectors $\mathbf{w}^{(1)}$, $\mathbf{w}^{(2)}$ in softmax regression, they are equivalent to just a single weight vector $\mathbf{w}$:

$$\begin{aligned}
\hat{\mathbf{y}}_1 &= \frac{\exp \mathbf{z}_1}{\sum_{k'=1}^{2} \exp \mathbf{z}_{k'}} \\
&= \frac{\exp \mathbf{z}_1}{\exp \mathbf{z}_1 + \exp \mathbf{z}_2} \\
&= \frac{1}{1 + \exp \mathbf{z}_2 / \exp \mathbf{z}_1} \\
&= \frac{1}{1 + \exp\left(\mathbf{z}_2 - \mathbf{z}_1\right)} \\
&= \frac{1}{1 + \exp\left(-z\right)}
\end{aligned}$$

Logistic regression is a special case of softmax regression

- Suppose c=2. Then even though there are two weight vectors $\mathbf{w}^{(1)}$, $\mathbf{w}^{(2)}$ in softmax regression, they are equivalent to just a single weight vector $\mathbf{w}$:

$$
\begin{aligned}
\hat{\mathbf{y}}_1 &= \frac{\exp \mathbf{z}_1}{\sum_{k'=1}^{2} \exp \mathbf{z}_{k'}} \\
&= \frac{\exp \mathbf{z}_1}{\exp \mathbf{z}_1 + \exp \mathbf{z}_2} \\
&= \frac{1}{1 + \exp \mathbf{z}_2 / \exp \mathbf{z}_1} \\
&= \frac{1}{1 + \exp\left(\mathbf{z}_2 - \mathbf{z}_1\right)} \\
&= \frac{1}{1 + \exp\left(-z\right)} \\
&= \frac{1}{1 + \exp\left(-\mathbf{x}^\top \mathbf{w}\right)}
\end{aligned}
$$

Logistic regression is a special case of softmax regression

- Suppose *c*=2. Then even though there are two weight vectors $\mathbf{w}^{(1)}$, $\mathbf{w}^{(2)}$ in softmax regression, they are equivalent to just a single weight vector $\mathbf{w}$:

$$
\begin{aligned}
\hat{\mathbf{y}}_1 &= \frac{\exp \mathbf{z}_1}{\sum_{k'=1}^{2} \exp \mathbf{z}_{k'}} \\
&= \frac{\exp \mathbf{z}_1}{\exp \mathbf{z}_1 + \exp \mathbf{z}_2} \\
&= \frac{1}{1 + \exp \mathbf{z}_2 / \exp \mathbf{z}_1} \\
&= \frac{1}{1 + \exp\left(\mathbf{z}_2 - \mathbf{z}_1\right)} \\
&= \frac{1}{1 + \exp\left(-z\right)} \\
&= \frac{1}{1 + \exp\left(-\mathbf{x}^\top \mathbf{w}\right)} \\
\text{and } \hat{\mathbf{y}}_2 &= \frac{1}{1 + \exp\left(\mathbf{x}^\top \mathbf{w}\right)}
\end{aligned}
$$

XOR problem

XOR problem

- Suppose we use logistic regression to distinguish between the positive and negative classes of the following “image” dataset (where *m*=2):

+ **x=[0,1]** **+** **x=[1,0]** **-** **x=[0,0]** **-** **x=[1,1]**

XOR problem

- Suppose we use logistic regression to distinguish between the positive and negative classes of the following "image" dataset (where m=2):

+	+	-	-
x=[0,1]	**x=[1,0]**	**x=[0,0]**	**x=[1,1]**

- **Exercise**: find a vector of weights $\mathbf{w}=[w_1, w_2]$ such that, for each positive example $\mathbf{x}^+$ and each negative example $\mathbf{x}^-$, we have $\sigma(\mathbf{x}^{+\top}\mathbf{w}) > \sigma(\mathbf{x}^{-\top}\mathbf{w})$, where:

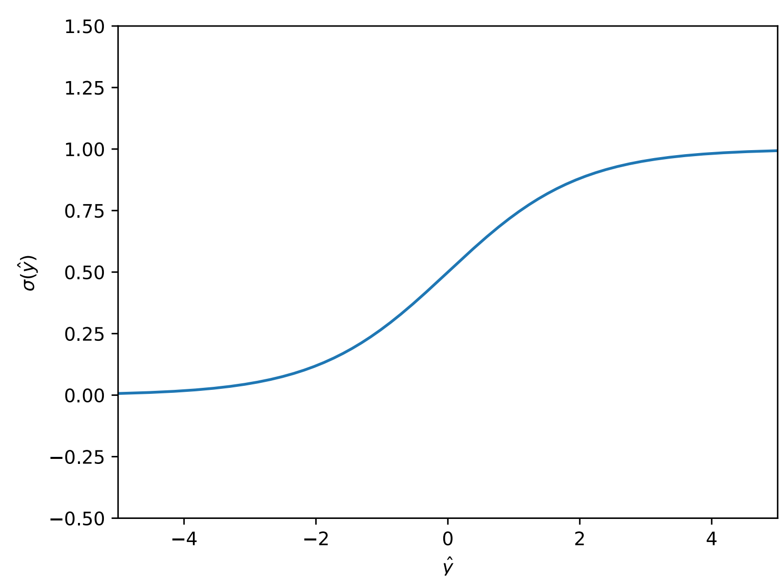
$$\sigma(\mathbf{x}^\top \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{x}^\top \mathbf{w})}$$

Jacob Whitehill, WPI

XOR problem

- Why is this task **impossible** with logistic regression?

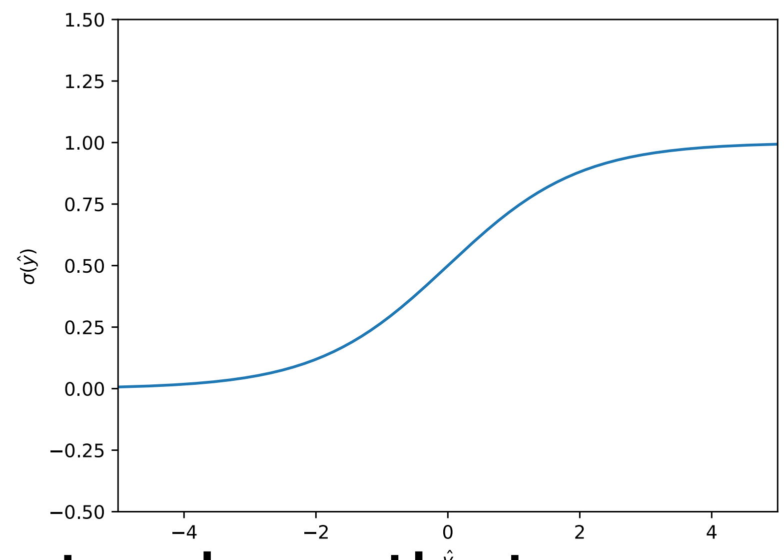
- First, note that $\sigma(z) > \sigma(z')$ if and only if $z > z'$.



XOR problem

- Why is this task **impossible** with logistic regression?

- First, note that $\sigma(z) > \sigma(z')$ if and only if $z > z'$.



- Hence, it suffices to show that no **w** satisfies:

$$(\mathbf{x}^{+})^{\top}\mathbf{w} > (\mathbf{x}^{-})^{\top}\mathbf{w}$$

for each positive example $\mathbf{x}^{+}$ and negative example $\mathbf{x}^{-}$.

XOR problem

- Suppose (by way of contradiction) such a **w** did exist.
- Then: $$\underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{\text{Positive example}} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} > \underbrace{\begin{bmatrix} 1 & 1 \end{bmatrix}}_{\text{Negative example}} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

XOR problem

- Suppose (by way of contradiction) such a **w** did exist.
- Then: $$\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} > \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

$$w_1 > w_1 + w_2$$

XOR problem

- Suppose (by way of contradiction) such a **w** did exist.
- Then: $\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} > \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$

$$w_1 > w_1 + w_2$$

$$w_2 < 0$$

XOR problem

- Suppose (by way of contradiction) such a **w** did exist.
- Then:

$$\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} > \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

$$w_1 > w_1 + w_2$$

$$w_2 < 0$$

$$\underbrace{\begin{bmatrix} 0 & 1 \end{bmatrix}}_{\text{Positive example}} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} > \underbrace{\begin{bmatrix} 0 & 0 \end{bmatrix}}_{\text{Negative example}} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

XOR problem

- Suppose (by way of contradiction) such a **w** did exist.
- Then:

$$\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} > \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

$$w_1 > w_1 + w_2$$

$$w_2 < 0$$

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} > \begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

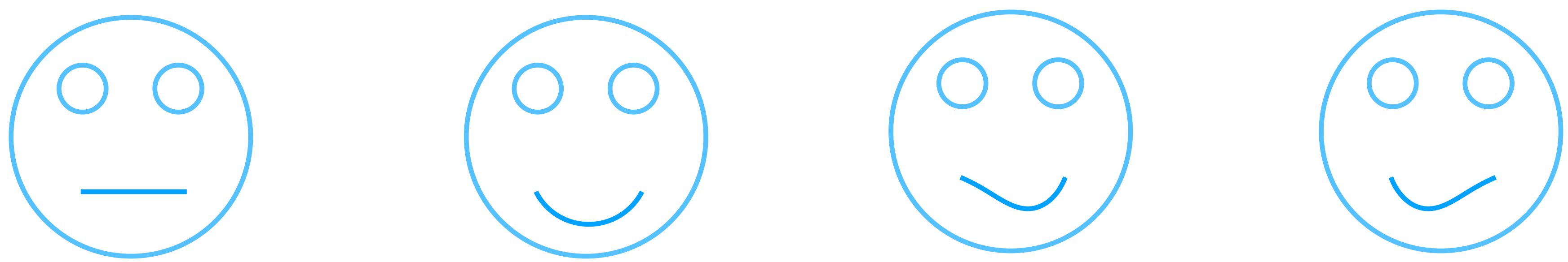
$$w_2 > 0$$

Contradiction

XOR problem

- This is an instance of the classic **XOR problem**:
    - 10 => 1
      01 => 1
      00 => 0
      11 => 0
- No linear or generalized linear model can solve this.
- Instead, we need non-linear models (more soon).

Smirk?



Neutral (0) **Smile (0)** **Smirk left (1)** **Smirk right (1)**