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 $y$. 
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:

\[
\begin{align*}
\mathbf{y}^{(1)} &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\
\mathbf{y}^{(2)} &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\
\mathbf{y}^{(3)} &= \begin{bmatrix} 1 \\ 0 \end{bmatrix}
\end{align*}
\]

• 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 0, 1, 0.

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

\[
\begin{align*}
y^{(1)} &= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} & \text{(This “slot” is for class 0.)} \\
y^{(2)} &= \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \\
y^{(3)} &= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\end{align*}
\]

• 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:

\[ y^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \]
\[ y^{(2)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \]
\[ y^{(3)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \]

• This is called a one-hot encoding of the class label.
Example: 2 classes

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

- They could consist of:

  \[
  \hat{y}^{(1)} = \begin{bmatrix} 0.93 \\ 0.07 \end{bmatrix} \quad \text{Machine’s “belief” that the label is 0.}
  \]

  \[
  \hat{y}^{(2)} = \begin{bmatrix} 0.4 \\ 0.6 \end{bmatrix}
  \]

  \[
  \hat{y}^{(3)} = \begin{bmatrix} 0.99 \\ 0.01 \end{bmatrix}
  \]

- Each coordinate of $\hat{y}$ is a probability.
Example: 2 classes

• The machine’s predictions $\hat{y}$ about each example’s label are also **probabilistic**.

• They could consist of:

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

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

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

• The sum of the coordinates in each $\hat{y}$ 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**):

\[
f_{CE} = - \sum_{i=1}^{n} \sum_{k=1}^{c} y_k^{(i)} \log \hat{y}_k^{(i)}
\]
Cross-entropy loss

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

• To see how, consider just a simple example:

$$f_{CE} = - \sum_{k=0}^{1} y_k \log \hat{y}_k$$
Cross-entropy loss

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

- To see how, consider just a simple example:

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

Note: the sum from $k=1$ to $c$ is renumbered from 0 to $c-1$. 
Cross-entropy loss

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

• To see how, consider just a simple example:

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

$$= -y_1 \log \hat{y}_1 - y_0 \log \hat{y}_0$$
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:

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

$$= -y_1 \log \hat{y}_1 - y_0 \log \hat{y}_0$$

$$= -y_1 \log \hat{y}_1 - (1 - y_1) \log (1 - \hat{y}_1)$$

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

Recall that the sum over all coordinates of each $\hat{y}$ (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 $f_{\log}$ (for logistic regression) is a special case of $f_{CE}$ (for softmax regression) for $c=2$.

• To see how, consider just a simple example:

\[
\begin{align*}
    f_{CE} &= - \sum_{k=0}^{1} y_k \log \hat{y}_k \\
    &= - y_1 \log \hat{y}_1 - y_0 \log \hat{y}_0 \\
    &= - y_1 \log \hat{y}_1 - (1 - y_1) \log(1 - \hat{y}_1) \\
    &= - y \log \hat{y} - (1 - y) \log(1 - \hat{y})
\end{align*}
\]

For $c=2$ classes, we can define $\hat{y}$ (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 $f_{CE}$ (for softmax regression) for $c=2$.

• To see how, consider just a simple example:

\[
f_{CE} = - \sum_{k=0}^{1} y_k \log \hat{y}_k
\]

\[
= -y_1 \log \hat{y}_1 - y_0 \log \hat{y}_0
\]

\[
= -y_1 \log \hat{y}_1 - (1 - y_1) \log (1 - \hat{y}_1)
\]

\[
= -y \log \hat{y} - (1 - y) \log (1 - \hat{y})
\]

\[
= f_{\log}
\]
Softmax activation function

- Logistic regression outputs a *scalar* probabilistic class label $\hat{y}$.
  - We needed just a single weight vector $w$, so that $\hat{y} = \sigma(x^Tw)$

- Softmax regression outputs a *vector* of probabilistic class labels $\hat{y}$ containing $c$ components.
  - We need $c$ different vectors of weights $w^{(1)}, \ldots, w^{(c)}$. 
Softmax activation function

- With softmax regression, we first compute:
  \[ z_1 = x^\top w^{(1)} \]
  \[ z_2 = x^\top w^{(2)} \]
  \[ \vdots \]
  \[ z_c = x^\top w^{(c)} \]

I will refer to the z's as “pre-activation scores”.
Softmax activation function

- With softmax regression, we first compute:
  \[ z_1 = x^\top w^{(1)} \]
  \[ z_2 = x^\top w^{(2)} \]
  \[ \vdots \]
  \[ z_c = x^\top w^{(c)} \]

- 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.
Normalization of the $\hat{y}_k$

1. To enforce non-negativity, we can exponential 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:

$$
\hat{y}_k = \frac{\exp(z_k)}{\sum_{k'=1}^{c} \exp(z_{k'})}
$$
• With softmax regression, we first compute:

\[ z_1 = \sum_{j=1}^{m} x_j w_{j}^{(1)} = x^\top w^{(1)} \]
With softmax regression, we first compute:

\[ z_1 = x^\top w^{(1)} \]

\[ \vdots \]

\[ z_c = \sum_{j=1}^{m} x_j w_j^{(c)} = x^\top w^{(c)} \]
• We then **normalize** across all $c$ classes.
Illustration

• Let $m=2$, $c=3$.

• Let: $\mathbf{x} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$

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

• Which class will have highest estimated probability?

\[
\mathbf{z} = \begin{bmatrix}
\end{bmatrix}
\]
Illustration

• Let $m=2$, $c=3$.

• Let: $x = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$

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

• Which class will have highest estimated probability?

  $z = \begin{bmatrix} 1.5 \\ 1 \\ -1 \end{bmatrix}$
• Let $m=2$, $c=3$.

• Let:

\[ x = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \]

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

• Which class will have highest estimated probability?

\[ z = \begin{bmatrix} 1.5 \\ 1 \\ -1 \end{bmatrix} \quad \hat{y} = \begin{bmatrix} 0.592 \\ 0.359 \\ 0.049 \end{bmatrix} \]
Softmax regression: vectorization

- We can represent each layer as a vector \((x, z, \hat{y})\).
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 \( W = \begin{bmatrix} w^{(1)} & \ldots & w^{(c)} \end{bmatrix} \)

- How can we compute the “pre-activation scores” \( z \) for all \( c \) classes in one-fell-swoop? Choose 0 or more of:

1. \( z^\top = x^\top W \)
2. \( z = x^\top W \)
3. \( z = Wx \)
4. \( z = W^\top x \)
Softmax regression: vectorization

- Let \( \mathbf{x}, \mathbf{z} \) be column vectors.
- Let \( \mathbf{W} = \begin{bmatrix} \mathbf{w}^{(1)} & \cdots & \mathbf{w}^{(c)} \end{bmatrix} \)
- How can we compute the “pre-activation scores” \( \mathbf{z} \) 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:

\[
Z = W^\top X \quad \text{\( c \times n \) matrix}
\]
Softmax regression: vectorization

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

\[
Z = W^\top X \quad c \times n \text{ matrix}
\]

- 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 $x$.

- We will compute the gradient w.r.t. each weight vector $w_k$ separately (where $k = 1, \ldots, c$).
Gradient descent for softmax regression

• Gradient for each weight vector $w_k$:

$$\nabla_{w_k} f_{CE}(y, \hat{y}; W) = x(\hat{y}_k - y_k)$$

• 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...
Gradient descent for softmax regression

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

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

One-hot encoded vector of class labels for example 1.
Gradient descent for softmax regression

• Let $\mathbf{Y}$ and $\mathbf{\hat{Y}}$ both be $n \times c$ matrices:

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

One-hot encoded vector of class labels for example $n$. 
Gradient descent for softmax regression

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

$$
Y = \begin{bmatrix}
  y_{1}^{(1)} & \cdots & y_{c}^{(1)} \\
  \vdots & & \vdots \\
  y_{1}^{(n)} & \cdots & y_{c}^{(n)}
\end{bmatrix}
$$

$$
\hat{Y} = \begin{bmatrix}
  \hat{y}_{1}^{(1)} & \cdots & \hat{y}_{c}^{(1)} \\
  \vdots & \vdots & \vdots \\
  \hat{y}_{1}^{(n)} & \cdots & \hat{y}_{c}^{(n)}
\end{bmatrix}
$$

The machine’s estimates of the $c$ class probabilities for example $n$. 


Gradient descent for softmax regression

• Let $Y$ and $\hat{Y}$ both be $n \times c$ matrices:

\[
Y = \begin{bmatrix}
y_{1}^{(1)} & \cdots & y_{c}^{(1)} \\
\vdots & \ddots & \vdots \\
y_{1}^{(n)} & \cdots & y_{c}^{(n)}
\end{bmatrix}
\quad \hat{Y} = \begin{bmatrix}
\hat{y}_{1}^{(1)} & \cdots & \hat{y}_{c}^{(1)} \\
\vdots & \ddots & \vdots \\
\hat{y}_{1}^{(n)} & \cdots & \hat{y}_{c}^{(n)}
\end{bmatrix}
\]

• Then we can compute all $c$ gradient vectors as:

\[
\nabla_{W} f_{CE}(Y, \hat{Y}; W) = \frac{1}{n} X(\hat{Y} - Y)
\]
Gradient descent for softmax regression

• Let $Y$ and $\hat{Y}$ both be $n \times c$ matrices:

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

• Then we can compute all $c$ gradient vectors as:

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

How far the guesses are from ground-truth.
Gradient descent for softmax regression

• Let $Y$ and $\hat{Y}$ both be $n \times c$ matrices:

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

• Then we can compute all $c$ gradient vectors as:

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

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: