Teaching Evaluations

https://heliocampusac.wisc.edu/

Multiclass Logistic Regression and Neural Networks

- cross entropy
- logistic regression
- neural network basics

Entropy, Cross-Entropy

- entropy is a fundamental quantity that measures the number of bits required to encode a sequences of i.i.d. random variables
  - we can encode/compress n i.i.d. random variables with $nH(X)$ bits
  - to build a code, we needed to know $p(x)$.

$$X_i \overset{i.i.d.}{\sim} \text{bernoulli}(\theta) \qquad H_2(\theta) = \theta \log \tfrac{1}{\theta} + (1-\theta) \log \tfrac{1}{1-\theta}$$

$$H(X) = \sum_i p_i \log_2 \left(\frac{1}{p_i} \right)$$

  - approach: only encode/compress vectors that had about θn ones.
  - code is optimized for $p(x)$

- field of study on constructing codes with desirable properties

Entropy, Cross-Entropy

```
model.compile(optimizer='sgd', loss=tf.keras.losses.CategoricalCrossentropy())
```

- entropy:

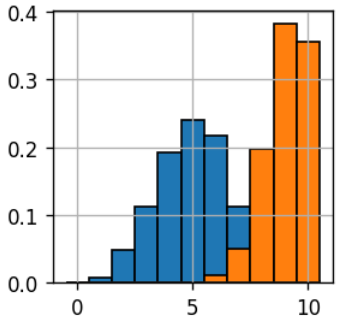
$$H(\boldsymbol{p}) = \sum_i p_i \log \frac{1}{p_i}$$

- number of bits required to encode a sequences of i.i.d. random variables

- cross-entropy:

$$H(\boldsymbol{p}, \boldsymbol{q}) = \sum_i p_i \log \frac{1}{q_i}$$

- number of bits required to encode a sequences of i.i.d. random variables from $\boldsymbol{p}$ using a code optimized for $\boldsymbol{q}$.
- $H(\boldsymbol{p}, \boldsymbol{q})$ is big if $q_i \ll p_i$ for some i



- Kullback-Leibler Divergence:

$$D\left(\boldsymbol{p}||\boldsymbol{q}\right) = \sum_i p_i \log \frac{p_i}{q_i}$$

- distance between $\boldsymbol{p}$ and $\boldsymbol{q}$
- important in binary hypothesis testing

- relationship:

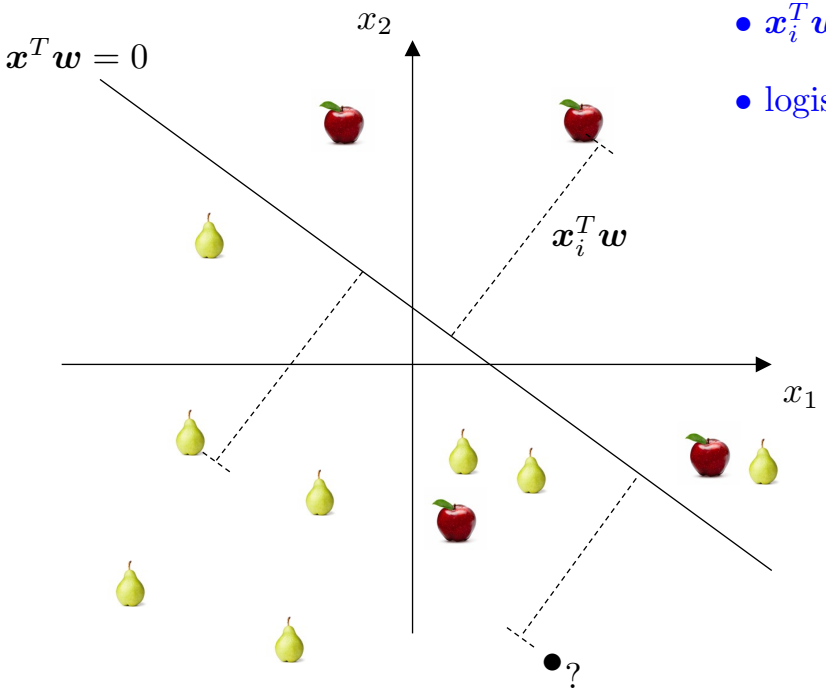
$$H(\boldsymbol{p}, \boldsymbol{q}) = H(\boldsymbol{p}) + D(\boldsymbol{p}||\boldsymbol{q})$$

Two Class Logistic Regression

- logistic regression is a *discriminative* classifier

$$p(y = 1|\boldsymbol{x}_i) = \frac{1}{1 + e^{-\boldsymbol{x}_i^T \boldsymbol{w}}}$$

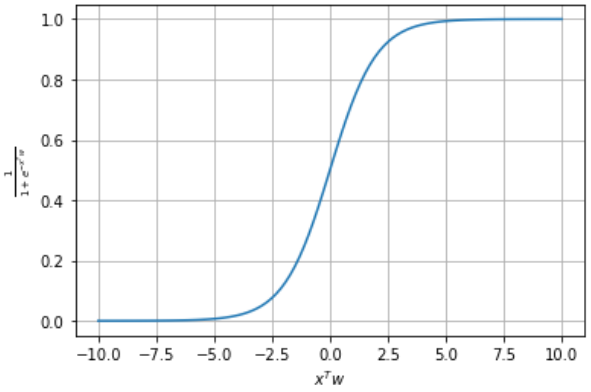
directly fit $p(y|\boldsymbol{x})$ to data by treating $\boldsymbol{x}$ as non-random



$d = 2$

- decision boundary is $\boldsymbol{x}^T\boldsymbol{w} = 0$
- $\boldsymbol{x}_i^T\boldsymbol{w}$ is the signed distance from the decision boundary
- logistic function takes $\boldsymbol{x}_i^T\boldsymbol{w}$ and squishes it to make it a probability

$\sigma(\cdot)$

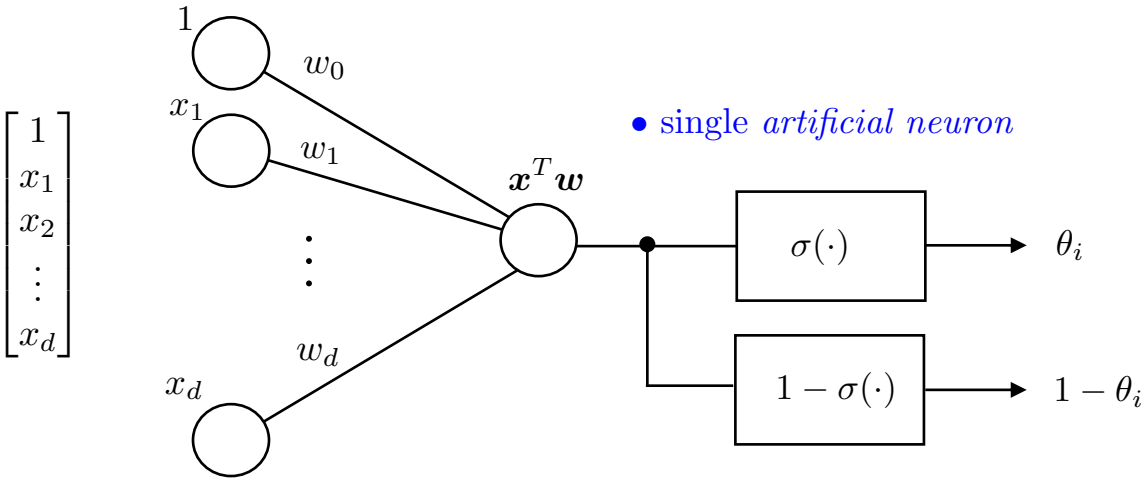


- probability that ? is a pear/apple depends distance from boundary

Logistic Regression

- testing

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



- single *artificial neuron*

MAP estimate

$$\widehat{y}_i = \arg\max_y p(y|\boldsymbol{x}_i)$$

- training

$$\mathcal{D} = \{(\boldsymbol{x}_i, y)\}_{i=1}^n$$

$$p_{\boldsymbol{w}}(y_1, y_2, \ldots, y_n) = \prod_i \theta_i^{\mathbb{I}\{y_i=1\}}(1-\theta_i)^{\mathbb{I}\{y_i=-1\}}$$

$$\theta_i = \frac{1}{1+e^{-\boldsymbol{x}_i^T\boldsymbol{w}}}$$

- notation

$$y_i = 1 \qquad \boldsymbol{p}_i = \begin{bmatrix}1\\0\end{bmatrix}$$

$$y_i = -1 \qquad \boldsymbol{p}_i = \begin{bmatrix}0\\1\end{bmatrix}$$

$$\boldsymbol{q}_i = \begin{bmatrix}\theta_i\\1-\theta_i\end{bmatrix}$$

Training

$$\theta_i = \frac{1}{1 + e^{-\boldsymbol{x}_i^T \boldsymbol{w}}}$$

- maximum likelihood principle

$$p_{\boldsymbol{w}}(y_1, y_2, \ldots, y_n) = \prod_i \theta_i^{\mathbb{I}\{y_i=1\}} (1-\theta_i)^{\mathbb{I}\{y_i=-1\}}$$

$$\log L(\boldsymbol{w}) = \sum_i \mathbb{I}_{\{y_i=1\}} \log \theta_i + \sum_i \mathbb{I}_{\{y_i=-1\}} \log(1-\theta_i)$$

- notation

$$y_i = 1 \qquad \boldsymbol{p}_i = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$y_i = -1 \qquad \boldsymbol{p}_i = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\boldsymbol{q}_i = \begin{bmatrix} \theta_i \\ 1-\theta_i \end{bmatrix}$$

$$= -\sum_i H(\boldsymbol{p}_i, \boldsymbol{q}_i)$$

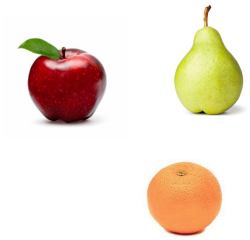
```
model.compile(optimizer='sgd', loss=tf.keras.losses.CategoricalCrossentropy())
```

$$\text{nll}(\boldsymbol{w}) = \sum_i H(\boldsymbol{p}_i, \boldsymbol{q}_i)$$

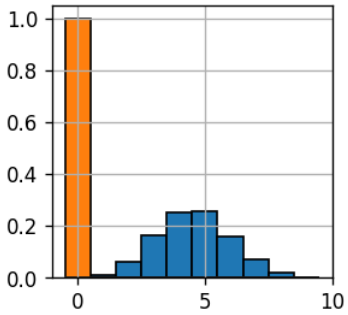
$$= \sum_i (H(\boldsymbol{p}_i) + D(\boldsymbol{p}_i || \boldsymbol{q}_i))$$

$$= \sum_i D(\boldsymbol{p}_i || \boldsymbol{q}_i)$$

Multiclass Classification



$$\boldsymbol{x} \longrightarrow \boxed{f(\cdot)} \longrightarrow \widehat{y}_i \in \{1, 2, \ldots, C\}$$



$$\mathcal{D} = \{(\boldsymbol{x}_i, y)\}_{i=1}^n \longrightarrow \mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{p}_i)\}_{i=1}^n$$

$\boldsymbol{p}_i$ = vector of zeros with a 1 in the y_i location

- example:

$$\boldsymbol{x}_i = \boxed{0} \longrightarrow \boxed{f(\cdot)} \longrightarrow \boldsymbol{q}_i = \begin{bmatrix} 0.92 \\ 0.01 \\ \vdots \\ 0.03 \end{bmatrix}$$

$$\boldsymbol{p}_i = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

- loss of ith training example:

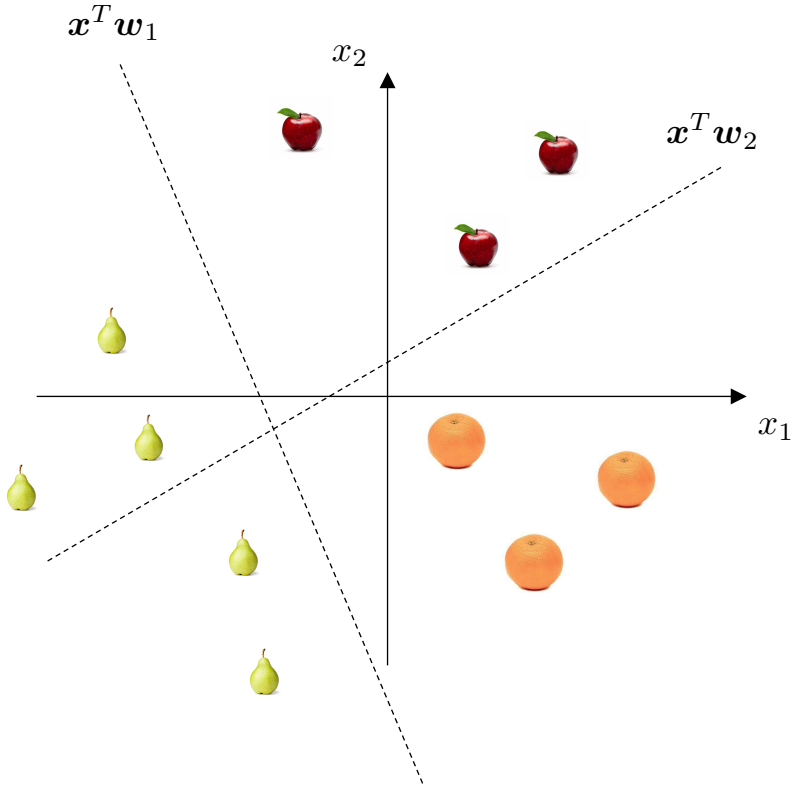
$$\ell(\widehat{y}, y) = D(\boldsymbol{p}_i || \boldsymbol{q}_i)$$

Multinomial Logistic Regression

- logistic regression is a *discriminative* classifier

directly fit $p(y|\boldsymbol{x})$ to data by treating $\boldsymbol{x}$ as non-random

$$p(y = c|\boldsymbol{x}_i) = \frac{e^{\boldsymbol{x}_i^T \boldsymbol{w}_c}}{\sum_{c'=1}^{C} e^{\boldsymbol{x}_i^T w_{c'}}}$$



- $\boldsymbol{x}_i^T \boldsymbol{w}_c$ is a **signed and scaled** distance from $\{\boldsymbol{x} : \boldsymbol{x}_c^T \boldsymbol{w} = 0\}$

- softmax function takes $\boldsymbol{x}^T \boldsymbol{w}_c, c = 1, \ldots, C$ and squishes distance to make a probability

- classification rule:

$$\widehat{y} = \arg\max_y p(y|\boldsymbol{x})$$

- boundaries are linear

$$\frac{p(y = c|\boldsymbol{x}_i)}{p(y = c'|\boldsymbol{x}_i)} = \frac{e^{\boldsymbol{x}_i^T \boldsymbol{w}_c}}{e^{\boldsymbol{x}_i^T \boldsymbol{w}_{c'}}} \gtrless 1$$

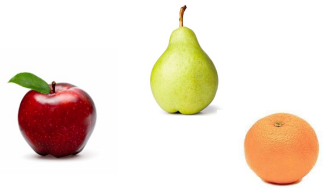
$$\boldsymbol{x}_i^T (\boldsymbol{w}_c - \boldsymbol{w}_{c'}) \gtrless 0$$

- convention: set $\boldsymbol{w}_C = \mathbf{0}$ to ensure identifiability

Multinomial Logistic Regression

- logistic regression is a *discriminative* classifier

$$y_i = \begin{cases} 1 & \text{with probability } \theta_i \\ -1 & \text{with probability } 1 - \theta_i \end{cases} \qquad \theta_i = \frac{1}{1 + e^{-\boldsymbol{x}_i^T \boldsymbol{w}}} = \frac{e^{\boldsymbol{x}_i^T \boldsymbol{w}}}{e^{-\boldsymbol{x}_i^T \boldsymbol{w}} + 1}$$



- multiclass (multinomial) logistic regression:

$$y_i = \begin{cases} 1 & \text{with probability } \theta_{i,1} \\ 2 & \text{with probability } \theta_{i,2} \\ & \vdots \\ c & \text{with probability } \theta_{i,C} \end{cases} \qquad \theta_{i,c} = p(y = c|\boldsymbol{x}_i) = \frac{e^{\boldsymbol{x}_i^T \boldsymbol{w}_c}}{\sum_{c'=1}^{C} e^{\boldsymbol{x}_i^T w_{c'}}}$$

- softmax:

$$[\boldsymbol{\sigma}(\boldsymbol{z})]_i = \frac{e_i^z}{\sum_j e^{z_j}}$$

$$\boldsymbol{\sigma}(\boldsymbol{z}) = \frac{e^{\boldsymbol{z}}}{\sum_j e^{z_j}}$$

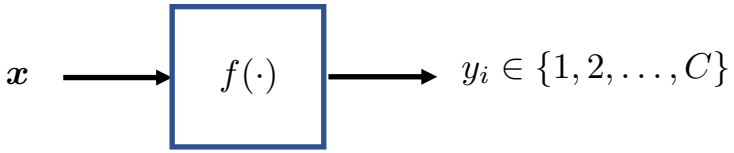
- softmax example:

```
[[-18.77687567]
 [39.26910416]
 [50.56209096]
 [47.72731802]
 [34.95432764]]
```

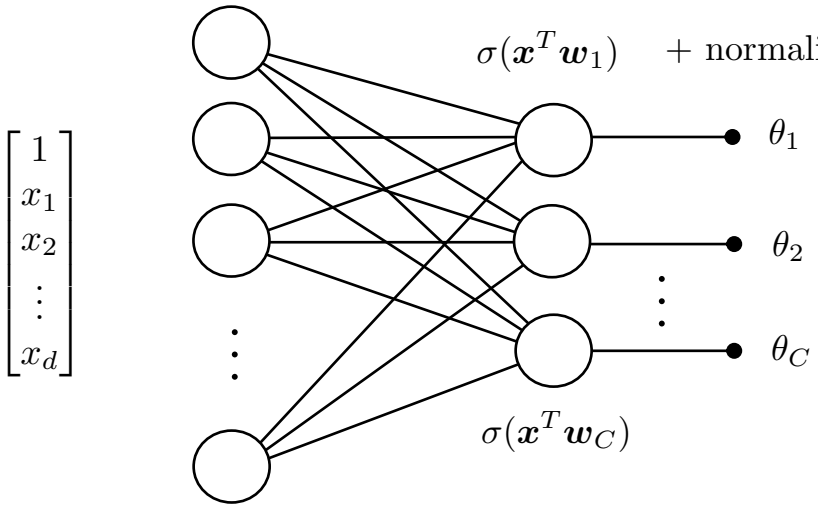
→

```
[[0.    ]
 [0.    ]
 [0.9445]
 [0.0555]
 [0.    ]]
```

Multinomial Logistic Regression



- testing
- single *single layer of artificial neural network*



$$\widehat{y}_i = \arg\max_y p(y|\boldsymbol{x}_i)$$

- loss

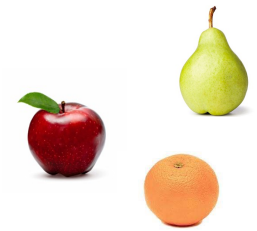
$$\ell(\widehat{y}, y) = D(\boldsymbol{p}_i || \boldsymbol{q}_i)$$

- training

$$\mathcal{D} = \{(\boldsymbol{x}_i, y)\}_{i=1}^n$$

$$p_{\boldsymbol{w}}(y_1, y_2, \ldots, y_n) = \prod_i \theta_{i,1}^{\mathbb{I}\{y_i=1\}} \theta_{i,2}^{\mathbb{I}\{y_i=2\}} \ldots \theta_{i,C}^{\mathbb{I}\{y_i=C\}}$$

Training



- training: how do we find a good $\boldsymbol{w}$?
- maximum likelihood principle

$$p(y = c|\boldsymbol{x}_i) = \theta_{i,c} = \frac{e^{\boldsymbol{w}_c^T \boldsymbol{x}_i}}{\sum_{c'=1}^{C} e^{\boldsymbol{w}_{c'}^T \boldsymbol{x}_i}}$$

- given data (and our model) what's the most likely $\boldsymbol{w}$?

$$p_{\boldsymbol{W}}(y_1, y_2, \ldots, y_n) = \prod_{i=1}^{n} \prod_{c=1}^{C} \theta_{i,c}^{\mathbb{I}\{y_i = c\}}$$

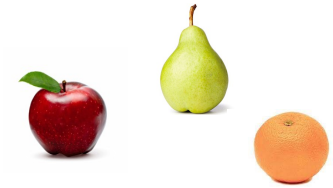
$$= \prod_{i=1}^{n} \prod_{c=1}^{C} \left(\frac{e^{\boldsymbol{w}_c^T \boldsymbol{x}_i}}{\sum_{c'=1}^{C} e^{\boldsymbol{w}_{c'}^T \boldsymbol{x}_i}} \right)^{\mathbb{I}\{y_i = c\}}$$

$$\log p_{\boldsymbol{W}} = \sum_{i=1}^{n} \sum_{c=1}^{C} \mathbb{I}\{y_i = c\} \log \left(\frac{e^{\boldsymbol{w}_c^T \boldsymbol{x}_i}}{\sum_{c'=1}^{C} e^{\boldsymbol{w}_{c'}^T \boldsymbol{x}_i}} \right)$$

$$= -\sum_i H(\boldsymbol{p}_i, \boldsymbol{q}_i)$$

$$\boldsymbol{p}_i = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad \boldsymbol{q}_i = \begin{bmatrix} e^{\boldsymbol{w}_1^T \boldsymbol{x}_i} \\ e^{\boldsymbol{w}_2^T \boldsymbol{x}_i} \\ \vdots \\ e^{\boldsymbol{w}_C^T \boldsymbol{x}_i} \end{bmatrix} \times \frac{1}{\sum_{c'} e^{\boldsymbol{w}_{c'}^T \boldsymbol{x}_i}}$$

Training



- training: how do we find a good $\boldsymbol{w}$?
- maximum likelihood principle

$$p(y = c|\boldsymbol{x}_i) = \frac{e^{\boldsymbol{w}_c^T \boldsymbol{x}_i}}{\sum_{c'=1}^{C} e^{\boldsymbol{w}_{c'}^T \boldsymbol{x}_i}}$$

- given data (and our model) what's the most likely $\boldsymbol{w}$?

$$p_{\boldsymbol{W}}(y_1, y_2, \ldots, y_n) = \prod_{i=1}^{n} \prod_{c=1}^{C} \theta_{i,c}^{\mathbb{I}\{y_i = c\}}$$

$$= \prod_{i=1}^{n} \prod_{c=1}^{C} \left(\frac{e^{\boldsymbol{w}_c^T \boldsymbol{x}_i}}{\sum_{c'=1}^{C} e^{\boldsymbol{w}_{c'}^T \boldsymbol{x}_i}} \right)^{\mathbb{I}\{y_i = c\}}$$

$$\log p_{\boldsymbol{W}} = \sum_{i=1}^{n} \sum_{c=1}^{C} \left(\mathbb{I}\{y_i = c\} \boldsymbol{w}_c^T \boldsymbol{x}_i - \mathbb{I}\{y_i = c\} \log \left(\sum_{c'=1}^{C} e^{\boldsymbol{w}_{c'}^T \boldsymbol{x}_i} \right) \right)$$

$$= \sum_{i=1}^{n} \left(\left(\sum_{c=1}^{C} \mathbb{I}_{\{y_i = c\}} \boldsymbol{w}_c^T \boldsymbol{x}_i \right) - \log \left(\sum_{c'=1}^{C} e^{\boldsymbol{w}_{c'}^T \boldsymbol{x}_i} \right) \right)$$

$$= \sum_{i=1}^{n} \left(\boldsymbol{w}_{y_i}^T \boldsymbol{x}_i - \log \left(\sum_{c'=1}^{C} e^{\boldsymbol{w}_{c'}^T \boldsymbol{x}_i} \right) \right)$$

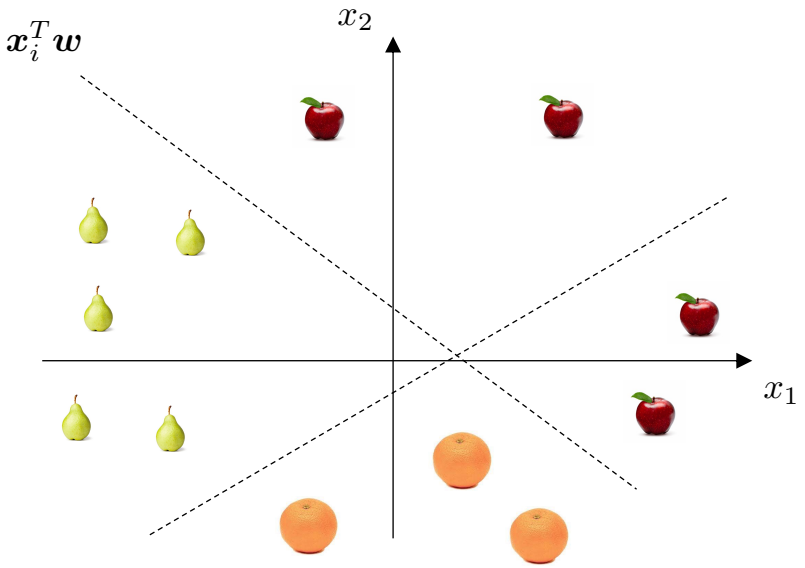
Optimizing

- training reduces to solving the following optimization

$$\arg \max_{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_c} \sum_{i=1}^{n} \left(\boldsymbol{w}_{y_i}^T \boldsymbol{x}_i - \log \left(\sum_{c'=1}^{C} e^{\boldsymbol{w}_{c'}^T \boldsymbol{x}_i} \right) \right)$$

- changing $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_c$ changes decision boundaries and steepness of logistic function

- no closed form solution



- iterative methods for finding $\boldsymbol{w}$

- concave and differentiable

Computing the Gradient

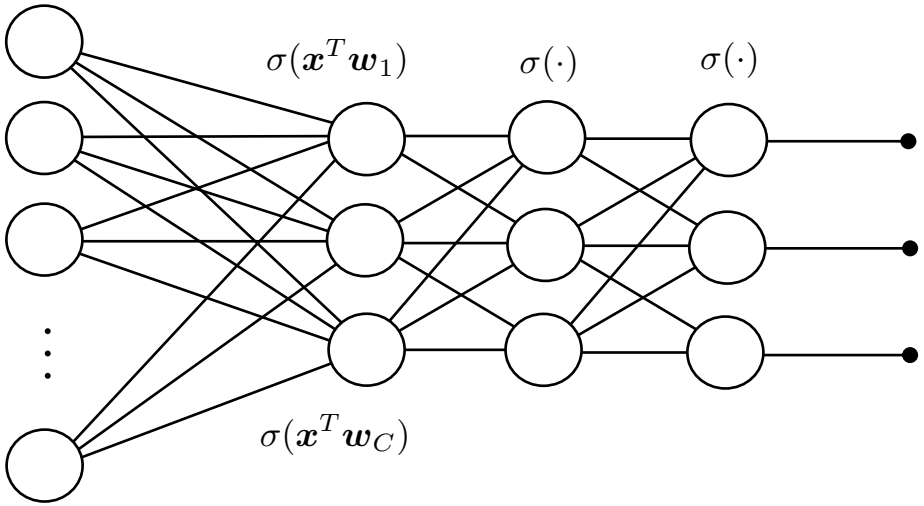
$$\arg \max_{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_c} \sum_{i=1}^{n} \left(\boldsymbol{w}_{y_i}^T \boldsymbol{x}_i - \log \left(\sum_{c'=1}^{C} e^{\boldsymbol{w}_{c'}^T \boldsymbol{x}_i} \right) \right)$$

$$\nabla_{\boldsymbol{w}_c} = \sum_i \left(\mathbb{I}_{\{y_i=c\}} - \frac{e^{\boldsymbol{w}_c^T \boldsymbol{x}_i}}{\sum_{c'} e^{\boldsymbol{w}_{c'}^T \boldsymbol{x}_i}} \right) \boldsymbol{x}_i$$

Artificial Neural Networks and Logistic Regression

- binary logistic regression:
  - single *artificial neuron*
- multiclass logistic regression:
  - single layer *artificial neural network*
- neural network:

- important notes:
  - with stacked layers, objective not convex
  - decision boundaries are no longer linear
  - can be used for regression



Differential Entropy

- definition - entropy:

$$H(X) = E\left[\log_2\left(\frac{1}{p(x)}\right)\right]$$

- discrete $x \in \mathcal{X}$:

$$H(X) = \sum_x p(x)\log_2\left(\frac{1}{p(x)}\right)$$

- amount of uncertainty in a random variable

- *differential entropy* is the equivalent definition of entropy, for continuous random variables

- continuous $x \in \mathbb{R}$:

$$H(X) = \int_x p(x)\log_2\left(\frac{1}{p(x)}\right)dx$$

- example: $X \sim \mathcal{N}(\mu, \sigma^2)$

$$H(X) = \frac{1}{2}\log_2(2\pi e\sigma^2) \text{ bits}$$