165B
Machine Learning
Logistic Regression

Lei Li (leili@cs)
UCSB

Acknowledgement: Slides borrowed from Bhiksha Raj's 11485 and Mu Li & Alex Smola's 157 courses on Deep Learning, with modification

Reminder

- Homework 1 due 11am Jan 12,
  - Please prepare your solution PDF using LaTeX (to make it clear and rigorous)
  - Handwritten and scanned image will not be accepted.
  - Submit to Gradescope. (please let me know immediately if you do not have access)
- Everyone enrolled should submit answer for in-class quiz.
  - Class participation counts 10%.
  - Only DSP students are allowed extra time for quiz.

Recap

- Machine learning is the study of machines that can improve their performance with more experience
- Linear Regression Model
  – Output is linearly dependent on the input variables
  – Minimize squared loss

Linear Regression

- Add bias into weights by

$$\mathbf{X} \leftarrow [\mathbf{X}, \mathbf{1}] \quad \mathbf{w} \leftarrow \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}$$

$$\ell(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \frac{1}{n} \| \mathbf{y} - \mathbf{X}\mathbf{w} \|^2$$

- Loss is convex, so the optimal solutions satisfies

$$\frac{\partial}{\partial \mathbf{w}} \ell(\mathbf{X}, \mathbf{y}, \mathbf{w}) = 0$$

$$\Leftrightarrow \mathbf{w}^* = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}\mathbf{y}$$

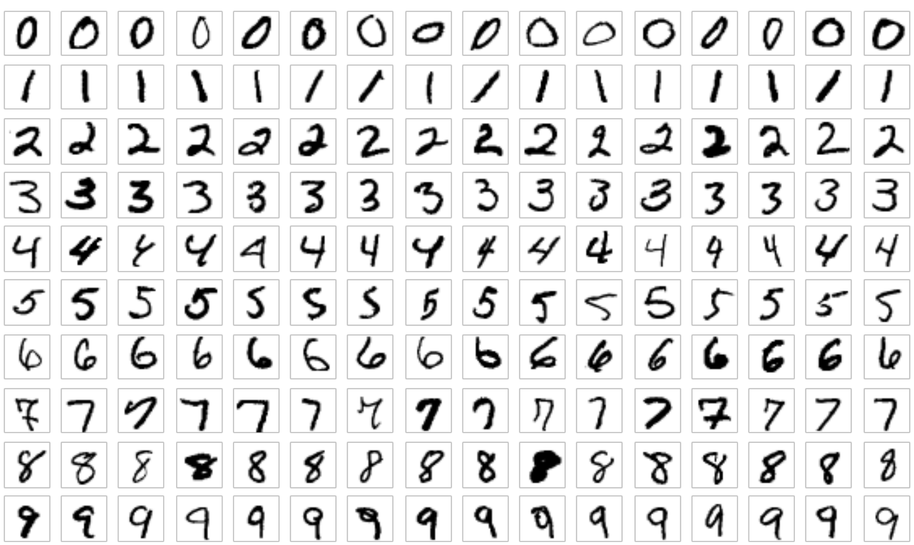
Quiz-3.1

- https://edstem.org/us/courses/16390/lessons/27551/slides/158899

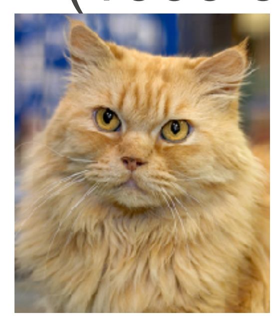
Regression vs. Classification

- Regression estimates a continuous value
- Classification predicts a discrete category

MNIST: classify hand-written digits (10 classes)



ImageNet: classify nature objects (1000 classes)



Cat

Dog

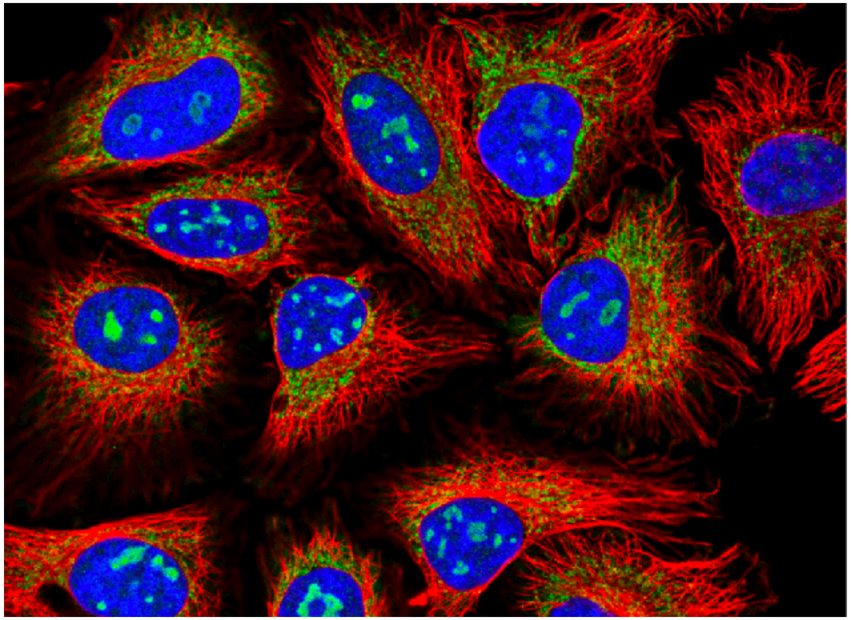
Handwriting Recognition

Optical Character Recognition (OCR)

0
1
2
3
4
5
6
7
8
9

Classifying Protein

Classify human protein microscope images into 28 categories



0. Nucleoplasm
1. Nuclear membrane
2. Nucleoli
3. Nucleoli fibrillar
4. Nuclear speckles
5. Nuclear bodies
6. Endoplasmic reticu
7. Golgi apparatus
8. Peroxisomes
9. Endosomes
10. Lysosomes
11. Intermediate fila
12. Actin filaments
13. Focal adhesion si
14. Microtubules
15. Microtubule ends
16. Cytokinetic bridg

https://www.kaggle.com/c/human-protein-atlas-image-classification

Text Classification

Classifying the sentiment of online movie reviews. (Positive, negative, neutral)

Spider-Man is an almost-perfect extension of the experience of reading comic-book adventures. 👍

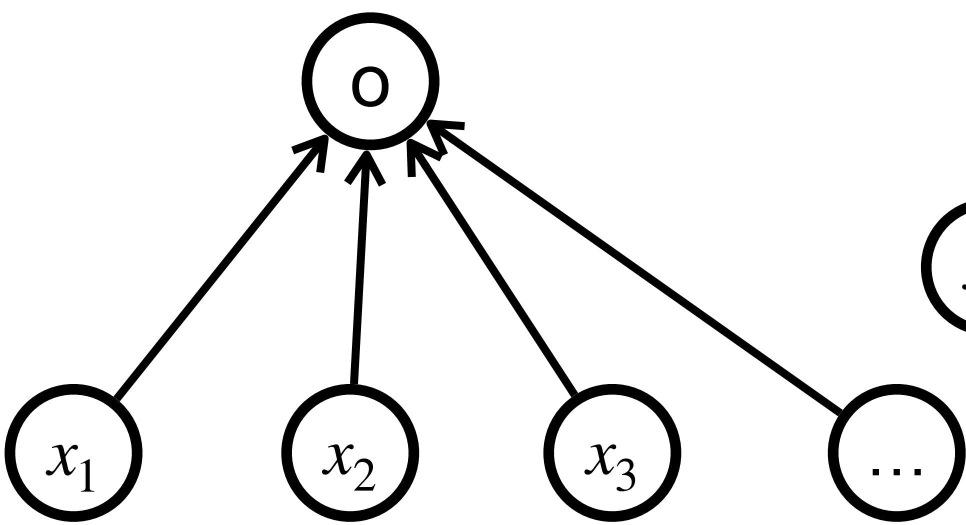
The acting is decent, casting is good. 👍

It was a boring! It was a waste of a movie to even be made. It should have been called a family reunion. 👎

From Regression to Multi-class Classification

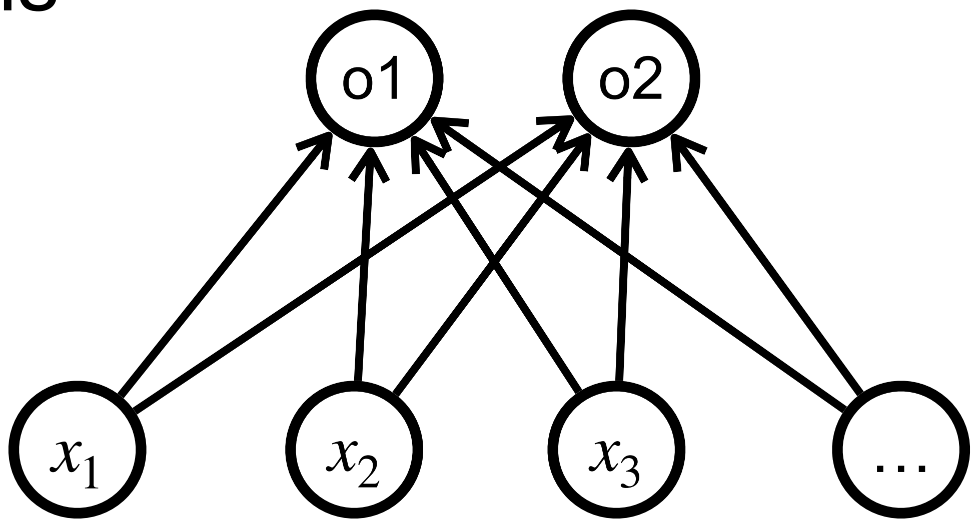
Regression

- Single continuous output
- Natural scale in
- Loss given e.g. in terms of difference



Classification

- Discrete output
- Score *should* reflect confidence/uncertainty …



From Regression to Multi-class Classification

Square Loss

- One hot encoding per class

$$\mathbf{y} = [y_1, y_2, \ldots, y_n]^\top$$

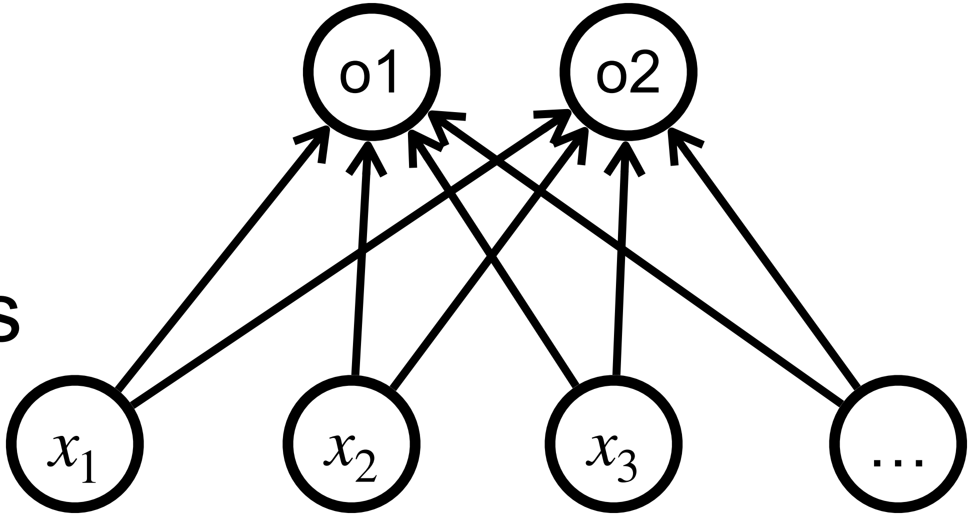
$$y_i = \begin{cases} 1 \text{ if } i = y \\ 0 \text{ otherwise} \end{cases}$$

- Train with squared loss
- Largest output wins

$$\hat{y} = \underset{i}{\operatorname{argmax}}\, o_i$$

Classification

- Discrete output
- Score *should* reflect confidence/uncertainty .



But, is there better way to model?

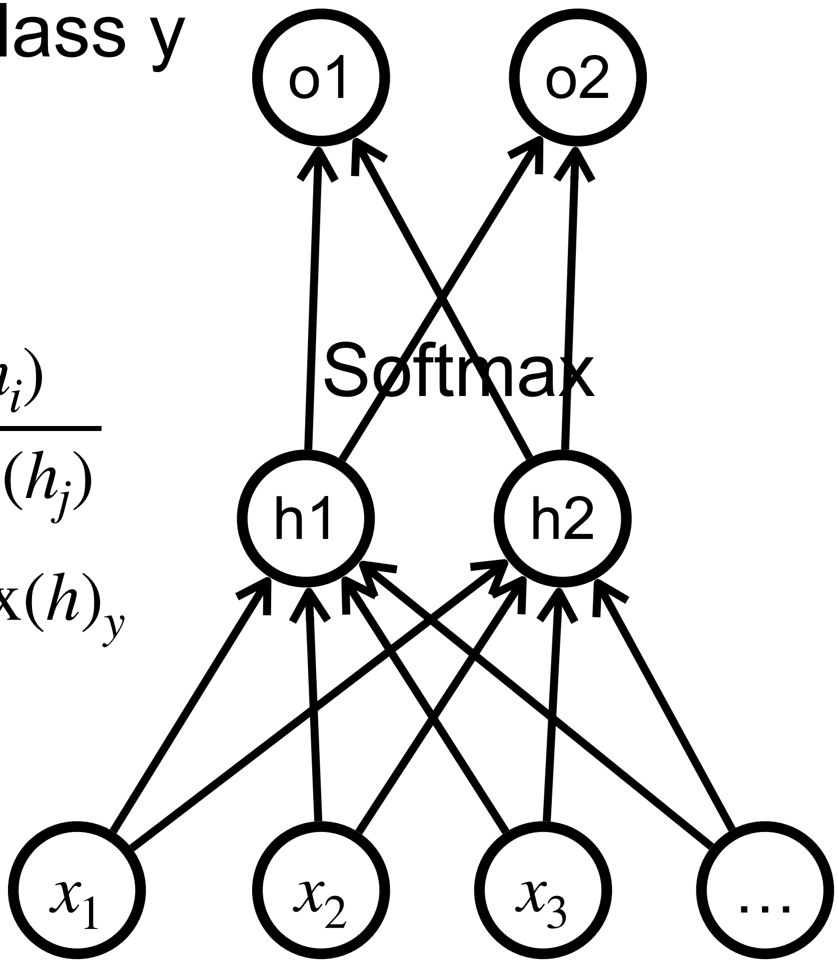
Logistic Regression

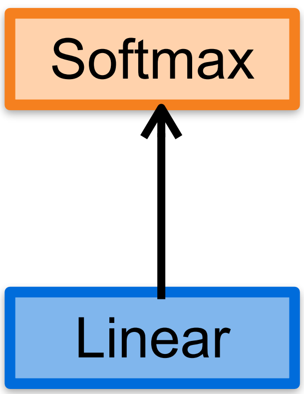
output: prob. of class y

$$h = \mathbf{W} \cdot \mathbf{x}$$

$$\text{softmax}(h)_i = \frac{\exp(h_i)}{\sum_j \exp(h_j)}$$

$$p(y \mid h) = \text{softmax}(h)_y$$





Logistic Regression in Pytorch

```
class LogisticRegression(torch.nn.Module):
    def __init__(self, input_dim, output_dim):
        super(LogisticRegression, self).__init__()
        self.linear = torch.nn.Linear(input_dim, output_dim)

    def forward(self, x):
        outputs = torch.sigmoid(self.linear(x))
        return outputs
```

Maximum Likelihood Estimation

$$\hat{\theta} = \arg\max \mathscr{L}(\theta; D)$$

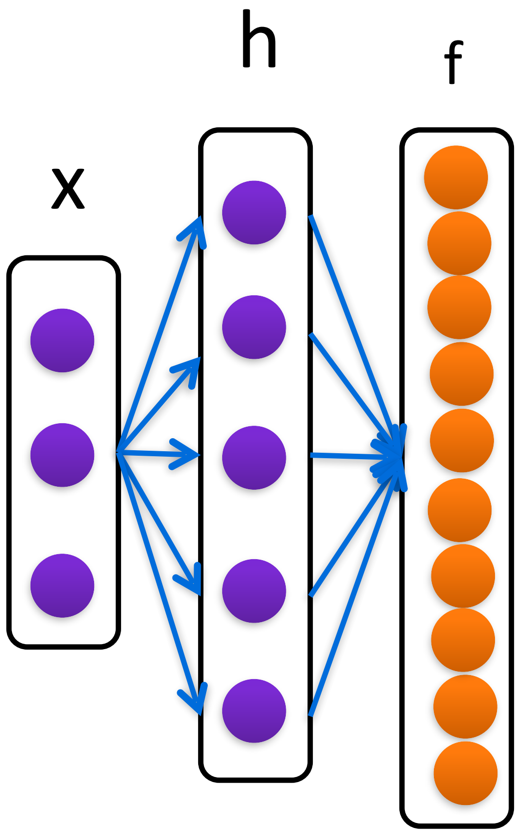
$\mathscr{L}$ is the log-likelihood function

$$\mathscr{L}(\theta; D) = \frac{1}{N}\sum_{n=1}^{N} \log p(y_n \mid x_n; \theta)$$

Or. equivalent to minimize negative log-likelihood

$$\hat{\theta} = \arg\min \ell(\theta; D) = -\frac{1}{N}\sum_{n=1}^{N} \log p(y_n \mid x_n; \theta)$$

Loss for Classification: Cross-Entropy



$$\mathscr{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} \ell(y_n, f(x_n); \theta)$$

$$\ell(y_n, f(x_n)) = H(y_n, f(x_n)) = -\log f(x_n)_{y_n}$$

$f(x_n)$ is a vector (e.g. $\in R^{10}$), representing predicted distribution

y_n is the ground-truth label, can be represented as an one-hot "distribution" [0,…,0, 1, 0,…,0]

Cross-entropy
$$H(p, q) = -\sum_k p_k \log q_k$$

Maximum Likelihood and Cross-Entropy

MLE

$$\boxed{\max \frac{1}{N}\sum_{n=1}^{N} \log p(y_n \,|\, x_n; \theta) = \frac{1}{N}\sum_{n=1}^{N}\sum_{k} y_{n,k} f(x_n)_k}$$

Or equivalently, minimize CE loss

$$\boxed{\min \mathscr{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} H(y_n, f(x_n)) = \frac{1}{N}\sum_{n=1}^{N} -\log f(x_n)_{y_n}}$$

Cross-Entropy Loss with Softmax

- Negative log-likelihood (for given label y)

$$-\log p(y \mid h) = \log \sum_i \exp(h_i) - h_y$$

- Cross-Entropy Loss (the true label y is an one-hot vector)

$$\ell(y, h) = \log \sum_i \exp(h_i) - y^\top h$$

- **Gradient**

$$\partial_h \ell(y, h) = \frac{\exp(h)}{\sum_i \exp(h_i)} - y$$

Difference between true and estimated

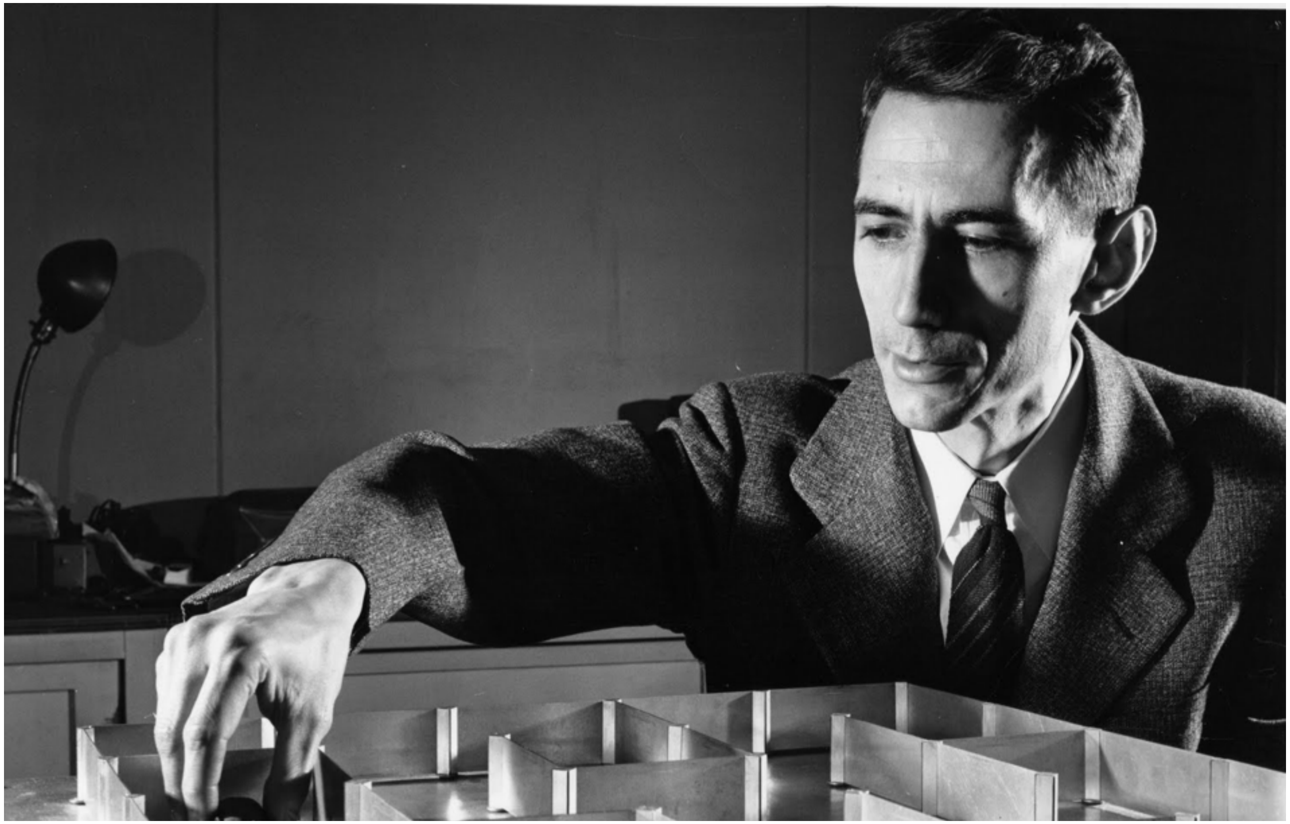
Quiz-3

https://edstem.org/us/courses/16390/lessons/27551/slides/156087

Compute the cross-entropy loss for the prediction prob.

Cat	0.6	0.2	0.4
Dog	0.1	0.8	0.05
Tiger	0.3	0	0.55

Information Theory



Claude Shannon

Entropy

- Data source producing observations $x_1 \ldots x_n$
- **How much 'information' is in this source?**
  - Tossing a fair coin - at each step the surprise is whether it's heads or tails
  - Rolling a fair dice - we have 1 out of 6 outcomes. This should be *more* surprising than the coin
  - Picture of a white wall vs. picture of a football stadium
    (the football stadium should have more information)
- **Measure is minimum number of bits needed**

Entropy

- Data source producing data $x_1 \ldots x_n$ with probability $p(x)$
- **Definition** $H[p] = -\sum_j p_j \log p_j$
- **Coding theorem**
  Entropy is lower bound on bits (or rather nats - base e) $2^a = e^b$ hence $a \log 2 = b$ hence $\text{bits} = \frac{H[p]}{\log 2}$

$$H[\lambda p + (1-\lambda)q] \geq \lambda H[p] + (1-\lambda)H[q]$$

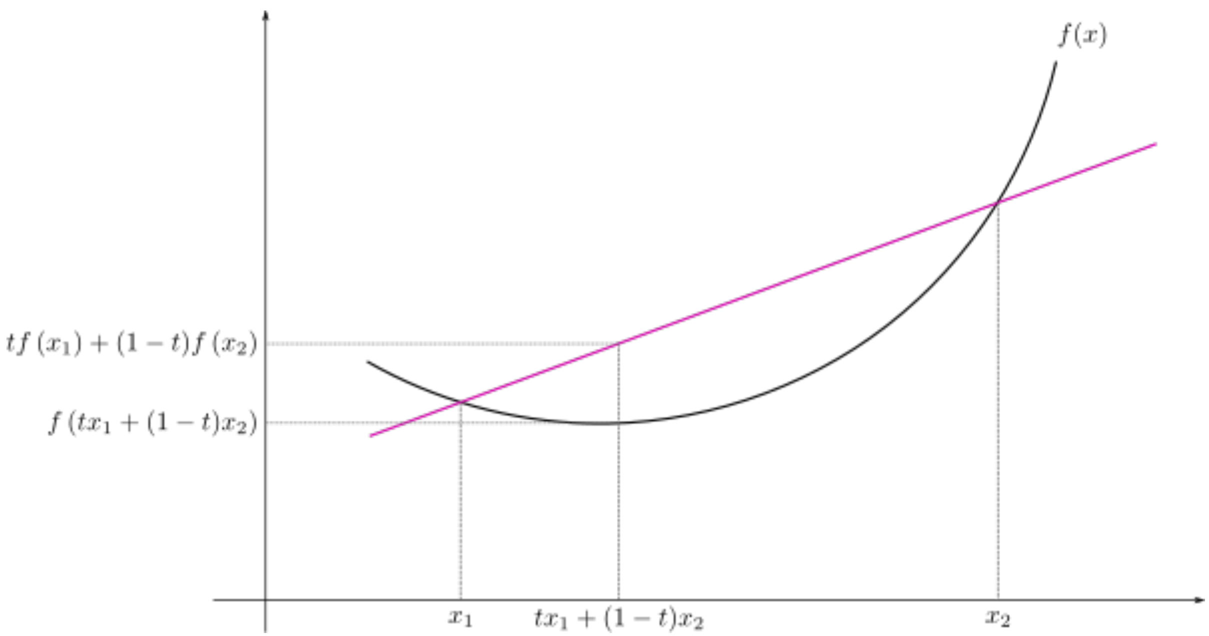
- Entropy is concave

Convex Function

f is convex iff

for all $0 < t < 1$, and all $x_1 \neq x_2$

$$tf(x_1) + (1-t)f(x_2) \geq f\left(tx_1 + (1-t)x_2\right)$$



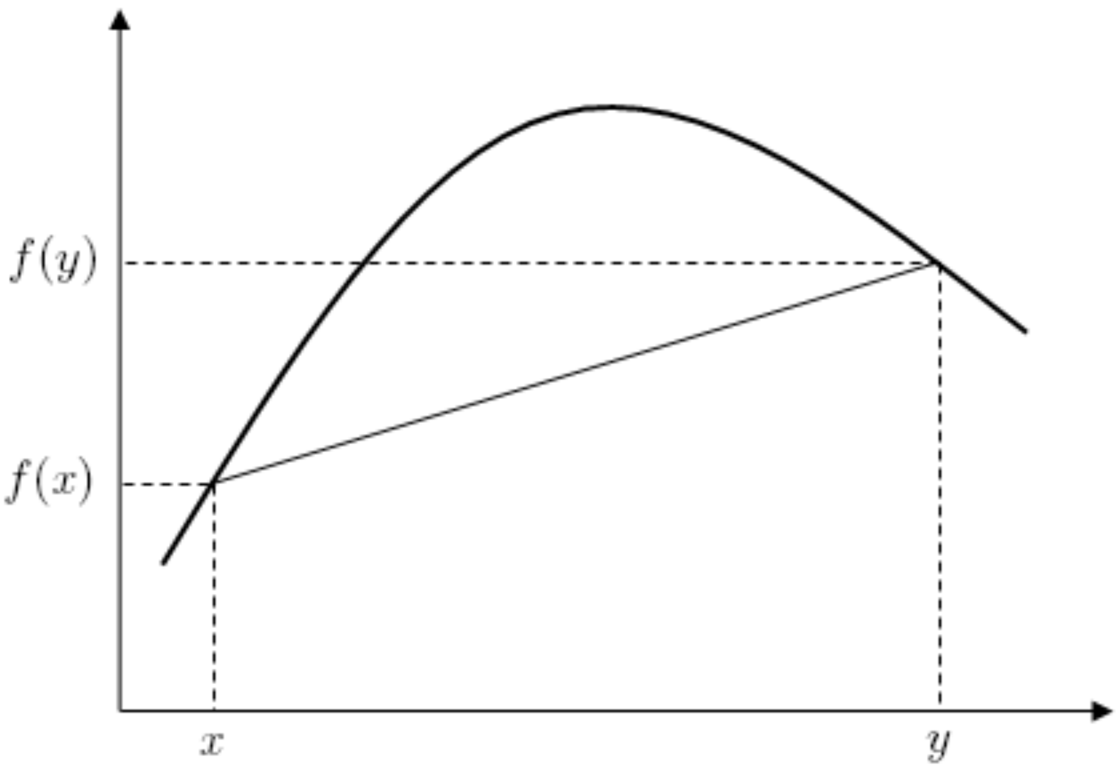
Convex function is very useful in optimization.

Concave Function

f is concave iff

for all $0 < t < 1$, and all $x_1 \neq x_2$

$$tf(x_1) + (1-t)f(x_2) \leq f\left(tx_1 + (1-t)x_2\right)$$



Entropy (binary form)

- Fair coin (p = 0.5)

$$H[p] = -0.5 \cdot \log_2 0.5 - 0.5 \cdot \log_2 0.5 = 1 \text{ bit}$$

- Biased coin (p = 0.9)

$$H[p] = -0.9 \cdot \log_2 0.9 - 0.1 \cdot \log_2 0.1 = 0.47 \text{ bit}$$

- Dungeons and Dragons (20-sided dice)

$$H[p] = -\log_2 \frac{1}{20} = 4.32 \text{ bit}$$

Kraft Inequality

- **Prefix Code**
  - m codes with length $l_1, l_2, \ldots, l_m$
  - No code c(x) is the prefix for any c(x'),

$$\cancel{\begin{cases} a \to 0 \\ b \to 01 \\ c \to 011 \\ d \to 0111 \end{cases}} \qquad \begin{cases} a \to 0 \\ b \to 10 \\ c \to 110 \\ d \to 111 \end{cases}$$

- The code length of all prefix code <==> Kraft inequality

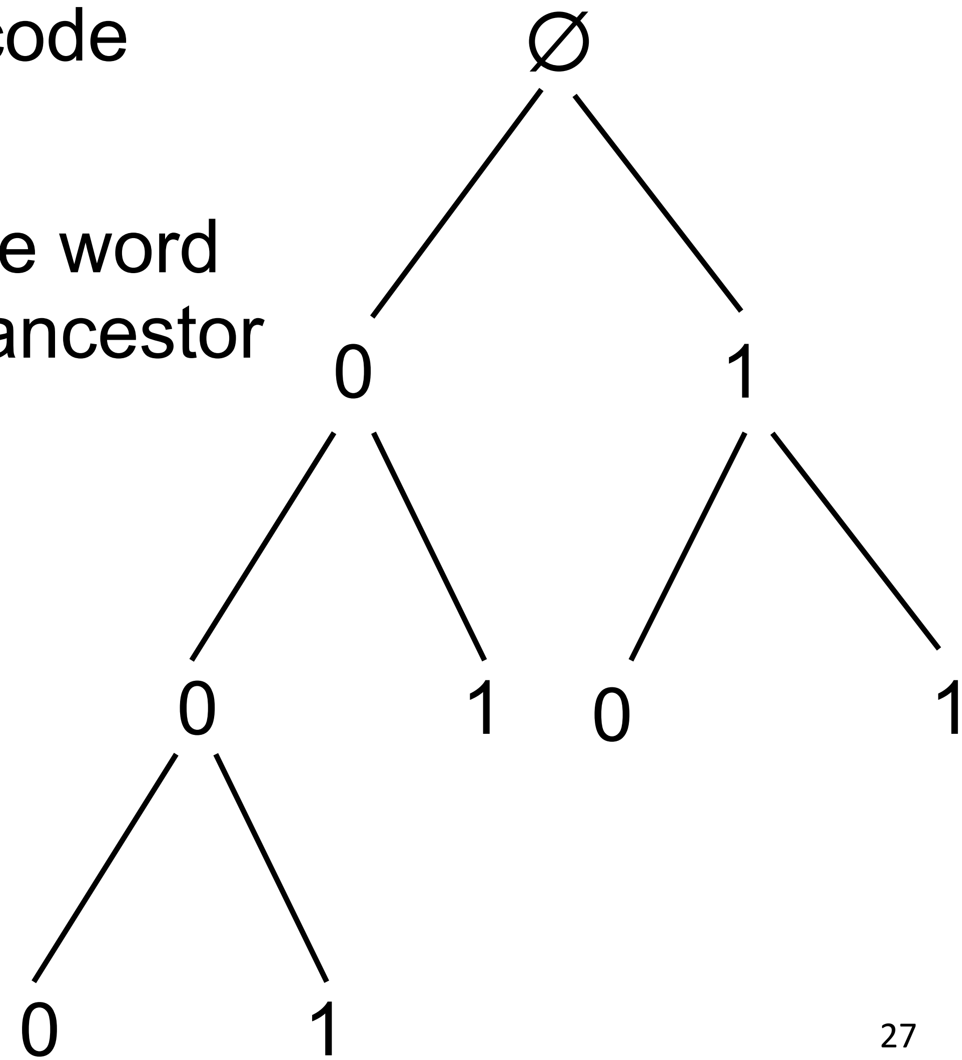
$$\sum_{i=1}^{m} 2^{-l_i} \le 1$$

Prefix Codes on Binary Trees

path from root represent code

Prefix code:

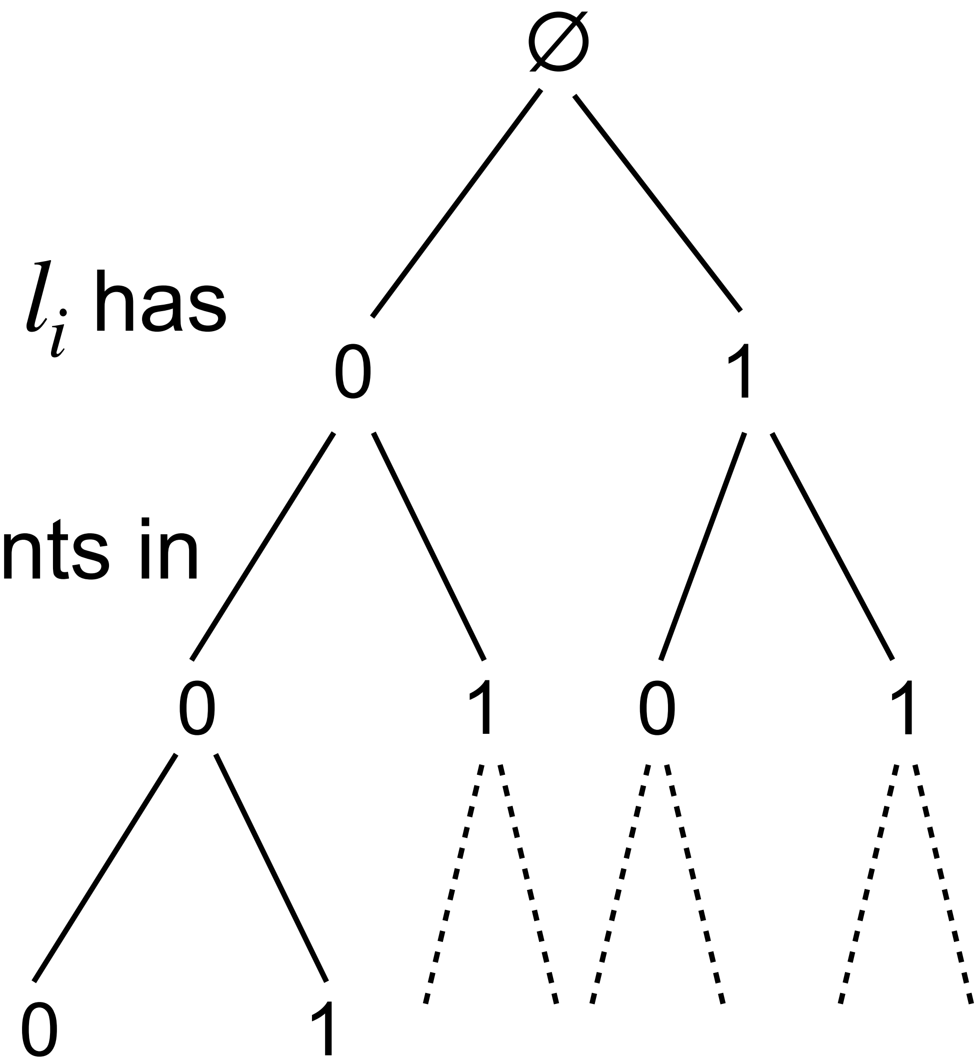
only leaf node can be code word
(code word can not be ancestor of another code node)



Proof of Kraft Inequality

- l_{max} be the length of longest codeword
- A codeword at length l_i has $2^{l_{max}-l_i}$ descendants
- How many descendants in total?

$$\sum_i 2^{l_{max}-l_i} \leq 2^{l_{max}}$$



Proof Kraft Inequality

- Conversely, if $l_1, l_2, \ldots, l_m$ satisfy Kraft inequality, then we can
- explicitly construct prefix code recursively
  - Pick set of {x} with smallest l(x) and generate code
  - Use leftovers and break them up into sets of weight $2^{-l(x)}$
  - Give each of them prefix and rescale by $2^{l(x)}$

Kraft Inequality

- Forward part

$$\begin{cases} a \to 0 & \frac{1}{2} \\ b \to 10 & \frac{1}{4} \\ c \to 110 & \frac{1}{8} \\ d \to 11110 & \frac{1}{32} \end{cases}$$

110001010

- Backwards part
  lengths (1, 2, 3, 5)
- Pick 1
  - Use code ‘0’ for it
  - Use prefix ‘1’ for the rest
  - Remaining set is (1, 2, 4)
  - Pick 1
    - Use code ‘0’ for it (thus ’10’)
    - Use prefix ‘1’ for the rest
    - Remaining set is (1,3)

Optimal expected code length

- Entropy is lower bound on expected number of bits

$$E[\#bits] = \sum_j p_j l_j \geq H_2[p] = -\sum_j p_j \log_2 p_j$$

- Proof:

$$E[\#bits] - H(p) = \sum_i p_i l_i - \sum p_i \log 1/p_i$$

$$= KL(p\,||\,q) + \log \frac{1}{\sum_j 2^{-l_j}} \geq 0$$

$$q_i = \frac{2^{-l_i}}{\sum_j 2^{-l_j}}$$

Optimal expected code length

- Entropy is lower bound on expected number of bits

$$E[\#bits] = \sum_j p_j l_j \geq H_2[p] = -\sum_j p_j \log_2 p_j$$

- Generate prefix code with length $l(x) = \lceil \log_2 p(x) \rceil$
  This is within 1 bit of optimal code
  Kraft inequality shows that such a thing exists.

$$\sum_x 2^{-\lceil -\log_2 p(x) \rceil} \leq \sum_x 2^{\log_2 p(x)} = \sum_x p(x) = 1$$

- Combine data in k-tuples to encode (within 1/k bit of optimal)

Kullback-Leibler Divergence

- Distance between distributions (e.g. truth & estimate)

Number of extra bits when using the wrong code

$$D[p\|q] = \int dp(x) \log \frac{p(x)}{q(x)} = \int dp(x) \underbrace{[-\log q(x)]}_{\text{Inefficient bits}} - \underbrace{[-\log p(x)]}_{\text{Optimal bits}}$$

- Nonnegativity of KL Divergence

Jensen Inequality
log is concave

$$D[p\|p] = \int dp(x) \log \frac{p(x)}{p(x)} = 0$$

$$D[p\|q] = -\int dp(x) \log \frac{q(x)}{p(x)} \geq -\log \int dp(x) \frac{q(x)}{p(x)} = 0$$

Minimizing Cross-Entropy is equivalent to Minimizing the KL divergence!

- Cross entropy loss

$$\ell(y, x) = H(y, f(x)) = -\log f(x)_y$$

- Cross entropy loss for softmax

$$\ell(y, h(x)) = \log \sum_i \exp(h(x)_i) - y^\top h(x)$$

- Kullback Leiber divergence

$$\begin{aligned} D(q \| p(\hat{y} \mid x)) = D(q \| \mathrm{softmax}(h(x))) \\ = \sum_i q_i \log q_i - q_i \log \mathrm{softmax}(h(x))_i \\ = -H[q] + \log \sum_i \exp(h(x)_i) - \sum_i q_i h(x)_i \end{aligned}$$

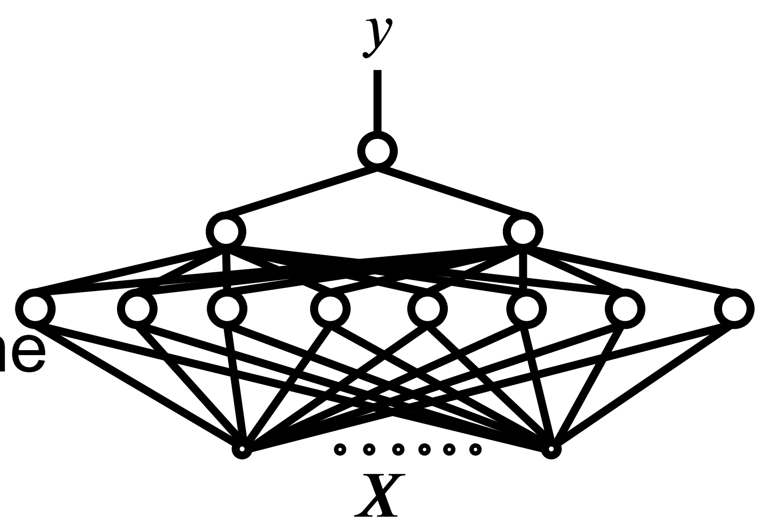
Independent of h()x

Recap

- The smallest number of bits to encode message is lower-bounded by entropy
- Minimizing cross entropy is equivalent to minimizing Kullback-Leibler Divergence

The Learning Problem

- Given a training set of input-output pairs $D = \{(x_n, y_n)\}_{n=1}^{N}$
  - x_n and y_n may both be vectors
- To find the model parameters such that the model produces the most accurate output for each training input
  - Or a close approximation of it
- Learning the parameter of a neural network is an instance!
  - The network architecture is given



The *Empirical* risk

- The expected risk is the average risk (loss) over the entire (x, y) data space

$$R(\theta) = E_{\langle x,y \rangle \in P}\left[\ell(y, f(x;\theta))\right] = \int \ell(y, f(x;\theta))dP(x,y)$$

- The empirical risk: average loss over the samples (using empirical data distribution)

$$R_{emp}(\theta) = \frac{1}{N}\sum_{n} \ell(y_n, f(x_n;\theta))$$

The general learning framework: Empirical Risk Minimization (ERM)

- Ideally, we want to minimize the expected risk
  - but, unknown data distribution …
- Instead, given a training set of empirical data $D = \{(x_n, y_n)\}_{n=1}^{N}$
- Minimize the empirical risk over training data

$$\hat{\theta} \leftarrow \arg\min_{\theta} L(\theta) = \frac{1}{N} \sum_{n} \ell(y_n, f(x_n; \theta))$$

The general learning framework: Empirical Risk Minimization (ERM)

- Ideally, we want to minimize the expected

Note : Its really a measure of error, but using standard terminology, we will call it a "Loss"

Note 2: The empirical risk $L(\theta)$ is only an empirical approximation to the true risk $R(\theta) = E_{\langle x,y \rangle \in P}\left[\ell(y, f(x;\theta))\right]$, which is our ultimat optimization objective

Note 3: For a given training set the loss is only a function of θ

$$\theta^* \leftarrow \arg\min_{\theta} L(\theta) = \frac{1}{N}\sum_{n} \ell(y_n, f(x_n, \theta))$$

The Training Problem

- Finding the parameter θ to minimize the empirical risk over training data
  $D = \{(x_n, y_n)\}_{n=1}^{N}$
  $$\hat{\theta} \leftarrow \arg\min_{\theta} L(\theta) = \frac{1}{N}\sum_{n} \ell(y_n, f(x_n; \theta))$$
- This is an instance of function optimization problem
  - Many algorithms exist (following lectures)

Defining the Training Objective

- The empirical risk (loss) is determined by the loss function
- Cross entropy loss is one common loss for classification

$$\min \mathscr{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} H(y_n, f(x_n)) = \frac{1}{N}\sum_{n=1}^{N} -\log f(x_n)_{y_n}$$

Other Loss for Classification

- Hinge loss

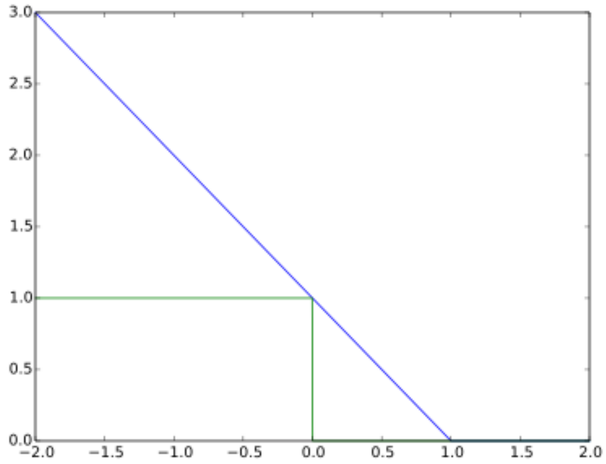
  - Binary classification:

$$\ell(y, \hat{y}) = \max(0, 1 - y\hat{y})$$

When ground-truth y is +1, prediction $\hat{y}$<0 lead to larger penalty

  - Multi-class

$$\ell(y, \hat{y}) = \sum_{k \neq y} \max(0, 1 - \hat{y}_y + \hat{y}_k)$$



Recap

- General framework to formulate a learning task is through empirical risk minimization (ERM)
- Minimizing cross-entropy is a realization of ERM

Next Up

- Multilayer Perceptron / Feedforward Network
- More on neural networks as universal approximators
  - And the issue of depth in networks
- How to train neural network from data