CS60010: Deep Learning

Sudeshna Sarkar

Spring 2018

15 Jan 2018

ML BASICS

Maximum Likelihood Estimation

- principle from which we can derive specific functions that are good estimators for different models.
- $X = \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ drawn independently from $p_{data}(x)$
- Let $p_{model}(x; \theta)$ be a parametric family of probability distributions over the same space indexed by θ - maps any configuration x to a real number estimating the true probability $p_{data}(x)$.
- MLE of x

$$\theta_{ML} = \underset{\theta}{argmax}\, p_{model}(X; \theta)$$

$$= \underset{\theta}{argmax} \prod_{i=1}^{m} p_{model(x^{(i)};\theta)}$$

$$\theta_{ML} = \underset{\theta}{argmax} \sum_{i=1}^{m} \log p_{model(x^{(i)};\theta)}$$

$$\theta_{ML} = \underset{\theta}{argmax}\, \mathbb{E}_{x \sim \hat{p}_{data}} \log p_{model}(x; \theta)$$

MLE

- One way to interpret maximum likelihood estimation is to view it as minimizing the dissimilarity between the empirical distribution $\hat{p}_{data}$ defined by the training set and the model distribution.
- Measured by KL-divergence

$$D_{KL}(\hat{p}_{data}||p_{model}) = \mathbb{E}_{x\sim\hat{p}_{data}}[\log \hat{p}_{data}(x) - \log p_{model}(x)]$$

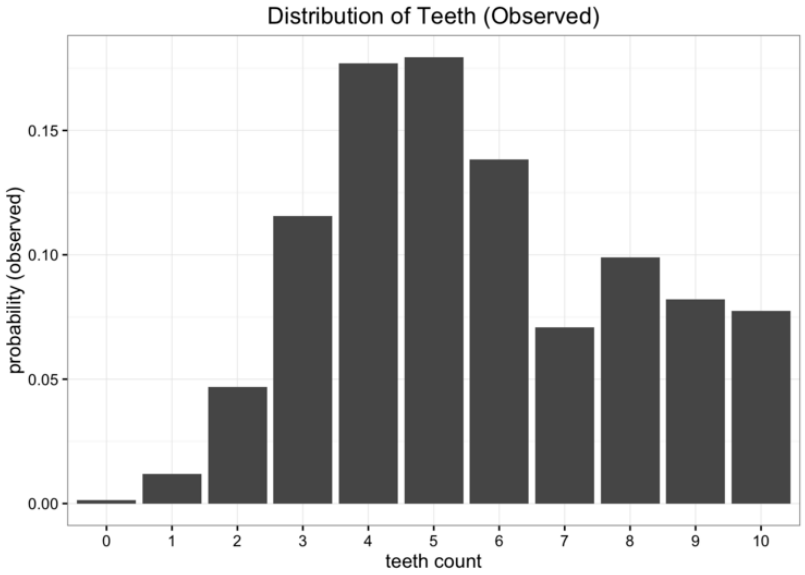
- The term on the left is a function only of the data-generating process, not the model. So we only need to minimize

$$-\mathbb{E}_{x\sim\hat{p}_{data}}[\log p_{model}(x)]$$

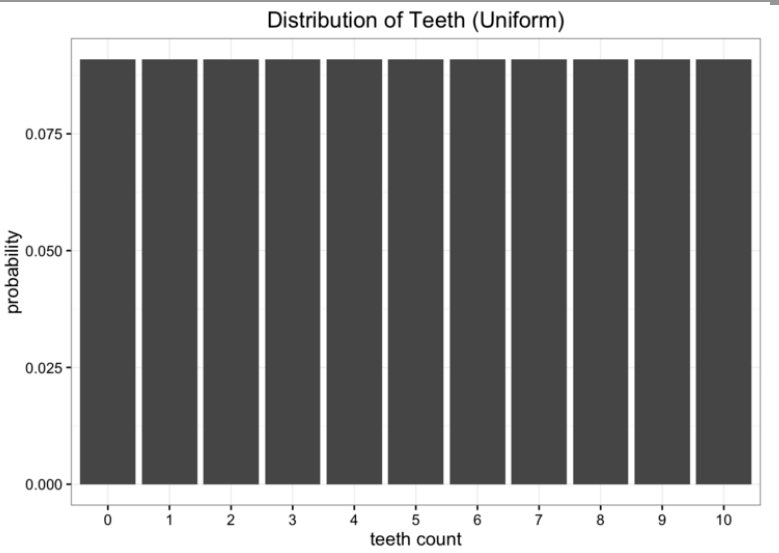
- Minimizing this KL divergence corresponds exactly to minimizing the cross-entropy between the distributions.

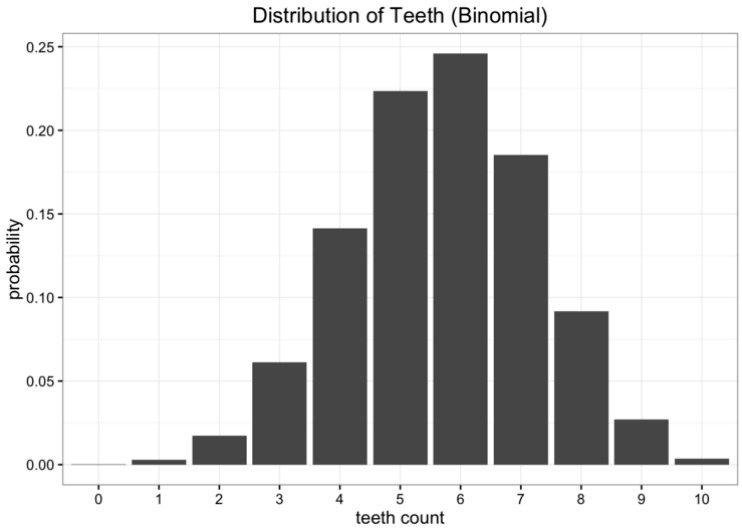
Kullback-Leibler Divergence Explained

KL Divergence measures how much information we lose when we choose an approximation.



The empirical probability distribution of the data collected

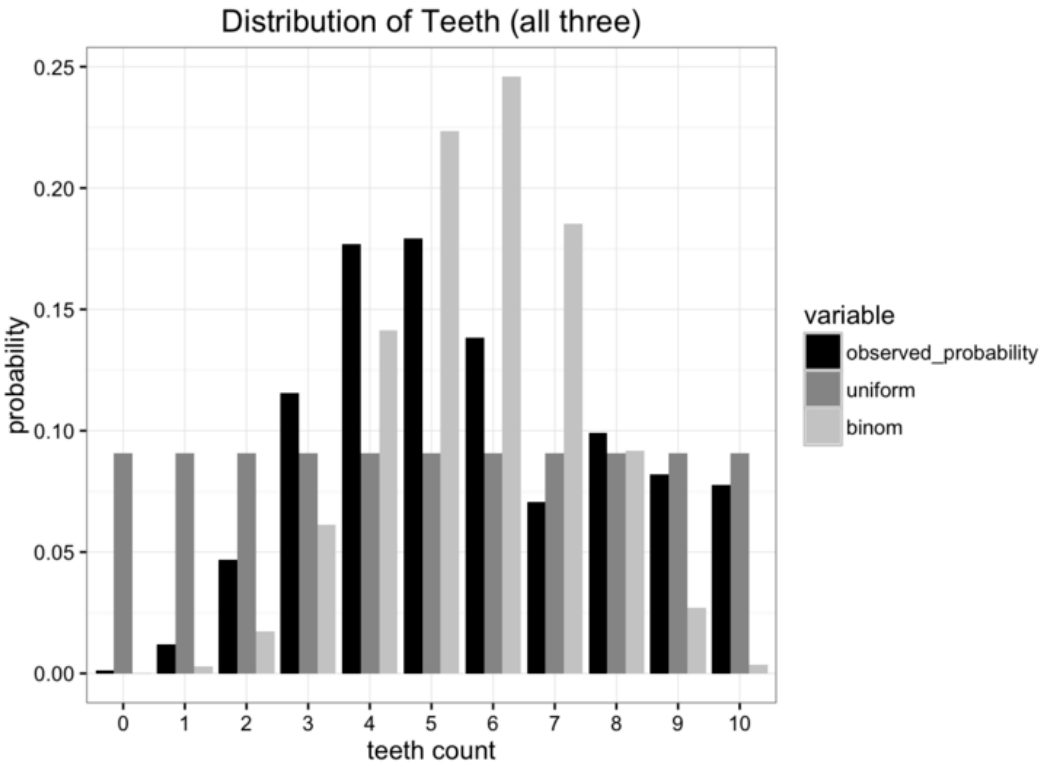




Binomial distr: best estimate of p is 0.57

https://www.countbayesie.com/blog/2017/5/9/kullback-leibler-divergence-explained

Comparing with the observed data, which model is better?



Which distribution preserves the most information from our original data source?

Information theory quantifies how much information is in data.

The most important metric in information theory is Entropy.

$$H = -\sum_{i=1}^{n} p(x_i) \log p(x_i)$$

Measuring information lost using KL Divergence between probability distribution p we and approximating distribution q.

$$D_{KL}(p||q) = p \sum_{i=1}^{n} p(x_i) \left(\log p(x_i) \ - \log q(x_i)\right)$$

the expectation of the log difference between the probability of data in the original distribution with the approximating distribution.

KL Divergence

$$D_{KL}(p||q) = p\sum_{i=1}^{n} p(x_i)\,(\log p(x_i)\ - \log q(x_i))$$

the expectation of the log difference between the probability of data in the original distribution with the approximating distribution. May interpret this as "how many bits of information we expect to lose".

$$D_{KL}(p||q) = \mathrm{E}[\log p(x) - \log q(x)]$$

Can be also written as

$$D_{KL}(p||q) = p\sum_{i=1}^{n} p(x_i).\log\frac{p(x_i)}{q(x_i)}$$

- KL Divergence is not symmetric. Thus it is not a distance metric.

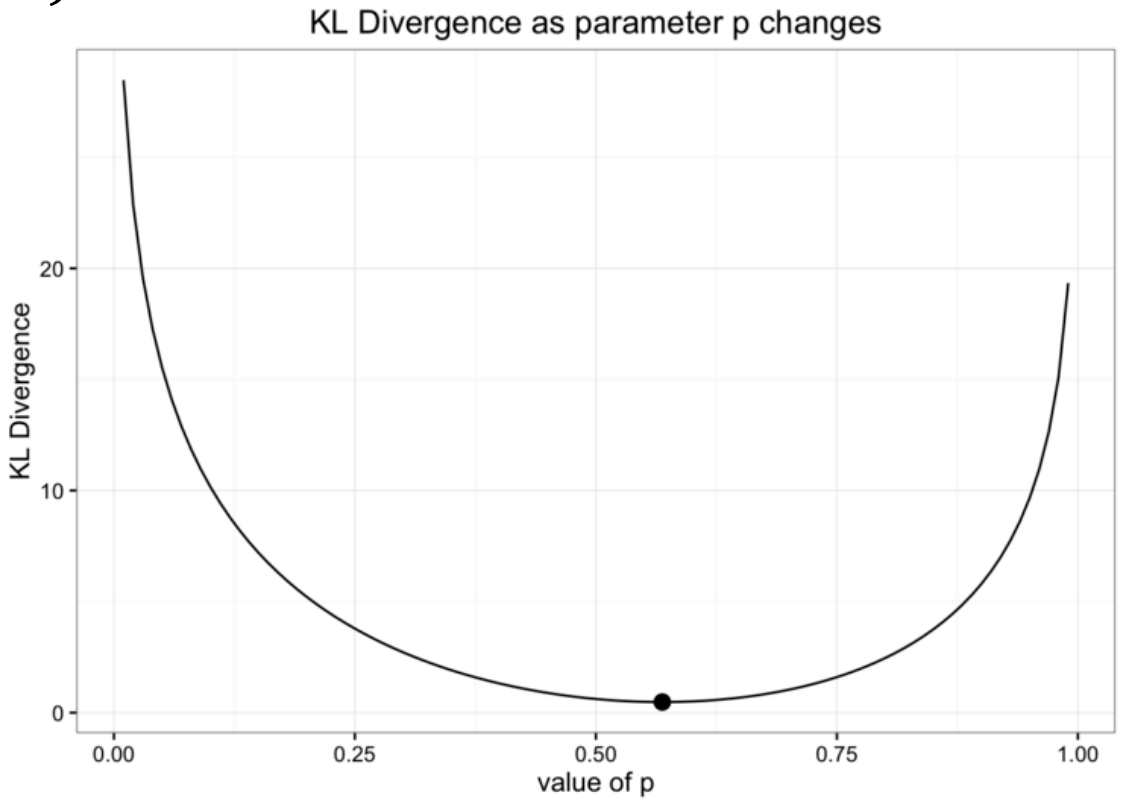
Comparing our approximating distributions

- For the uniform distribution:

$$D_{KL}(\text{Observed}||\text{Uniform}) = 0.338$$

- For our Binomial approximation:

$$D_{KL}(\text{Observed}||\text{Binomial}) = 0.477$$



Conditional Log-Likelihood

- ML estimator can be generalized to estimate a conditional probability $P(y|x;\theta)$

$$\theta_{ML} = \underset{\theta}{argmax}\, P(Y|X;\theta)$$

For i.i.d. examples,

$$\theta_{ML} = \underset{\theta}{argmax} \sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)};\theta)$$

Stochastic Gradient Descent

- Nearly all of deep learning is powered by one very important algorithm: stochastic gradient descent or SGD
- large training sets computationally expensive but necessary for generalization
- The cost function used by a machine learning algorithm often decomposes as a sum over training examples of some per-example loss function.
- E,g, the negative conditional log-likelihood of the training data can be written as

$$J(\theta) = \mathbb{E}_{x \sim \hat{p}_{data}} L(x, y, \theta) = \frac{1}{m} \sum_{i=1}^{m} L(x^{(i)}, y^{(i)}, \theta)$$

where L is the per-example loss $L(x, y, \theta) = -\log p(y|x; \theta)$

SGD

- gradient descent requires computing

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L(x^{(i)}, y^{(i)}, \theta)$$

- The insight of stochastic gradient descent is that the gradient is an expectation. The expectation may be approximately estimated using a small set of samples.
- We can sample a minibatch of examples of size m'
- The estimate of the gradient is formed as

$$g = \frac{1}{m'}\sum_{i=1}^{m'} \nabla_\theta L(x^{(i)}, y^{(i)}, \theta)$$

FEEDFORWARD NETWORKS

Background: Supervised ML

- Train Set $= \{(x^{(i)}, y^{(i)})\}$
  - $x^{(i)} \in X$, $y^{(i)} \in Y$
- Classification: Finite Y
- Regression: Continuous Y

Building a ML Algorithm

- Nearly all learning algorithms can be described by a specification of
  - A dataset
  - A cost function
  - An optimization procedure
  - A model

- Choose a model class of functions
- Design a criteria to guide the selection of one function from the selected class

Loss Functions

- L maps decisions to costs.
- $L(\hat{y}, y)$ is the penalty for predicting $\hat{y}$ when the correct answer is y.
- Examples of loss function
  - Classification: 0/1 loss
  - Regression: $L(\hat{y}, y) = (\hat{y} - y)^2$

- Empirical Loss of a function $y = f(x; \theta)$ on set X

$$L(\theta . X, y) = \frac{1}{m} \sum_{i=1}^{m} L(f(x_i; \theta), y_i)$$

Parametric approach: **Linear classifier**

$$f(x, W) = Wx$$

10x1 10x3072 3072x1

(+b) 10x1

[32x32x3]
array of numbers 0...1

10 numbers, indicating class scores

parameters, or “weights”

Going forward: Loss functions/optimization



airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

1. The **loss function** quantifies our unhappiness with the scores across the training data.

2. Come up with a way of efficiently finding the parameters that minimize the loss function. **(optimization)**

Loss functions

Suppose: 3 training examples, 3 classes.
For some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Multiclass Support Vector Machine loss

- SVM “wants” the correct class for each input to a have a score higher than the incorrect classes by some fixed margin Delta.
- Given (x_i, y_i), $f(x_i, W)$ computes the class scores.
- The score of the jth class : $s_j = f(x_i, W)_j$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Multiclass SVM loss:

Suppose: 3 training examples, 3 classes.
For some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,
and using the shorthand for the scores vector:

$$s = f(x_i, W)$$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Multiclass SVM loss:

Suppose: 3 training examples, 3 classes.
For some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9		

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,
and using the shorthand for the scores vector:

$$s = f(x_i, W)$$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

= max(0, 5.1 - 3.2 + 1)
+max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

Multiclass SVM loss:

Suppose: 3 training examples, 3 classes.
For some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,
and using the shorthand for the scores vector:

$$s = f(x_i, W)$$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

= max(0, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

Slides based on cs231n by Fei-Fei Li & Andrej Karpathy & Justin Johnson

Multiclass SVM loss:

Suppose: 3 training examples, 3 classes.
For some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	10.9

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,
and using the shorthand for the scores vector:

$$s = f(x_i, W)$$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

= max(0, 2.2 - (-3.1) + 1)
+max(0, 2.5 - (-3.1) + 1)
= max(0, 5.3) + max(0, 5.6)
= 5.3 + 5.6
= 10.9

Multiclass SVM loss:

Suppose: 3 training examples, 3 classes.
For some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	10.9

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,
and using the shorthand for the scores vector:

$$s = f(x_i, W)$$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

and the full training loss is the mean over all the examples:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

L = (2.9 + 0 + 10.9)/3 = **4.6**

Weight Regularization

λ= regularization strength (hyperparameter)

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max\left(0, f(x_i, W)_j - f(x_i, W)_{y_i} + 1\right) + \lambda R(W)$$

In common use:

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization $R(W) = \sum_k \sum_l \left|W_{k,l}\right|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + \left|W_{k,l}\right|$

Dropout (will see later)

Max norm regularization (might see later)

Softmax Classifier (Multinomial Logistic Regression)

Scores = unnormalized log prob. of the classes

$$P(Y = k|X = x_i) = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \quad \text{where} \quad s = f(x, W)$$

Softmax function

cat	**3.2**
car	5.1
frog	-1.7

Slides based on cs231n by Fei-Fei Li & Andrej Karpathy & Justin Johnson

Softmax Classifier (Multinomial Logistic Regression)

cat **3.2**

car 5.1

frog -1.7

Scores = unnormalized log prob. of the classes

$$P(Y = k|X = x_i) = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \quad \text{where} \quad s = f(x, W)$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$L_i = -\log P(Y = y_i|X = x_i)$$

Softmax Classifier (Multinomial Logistic Regression)

cat **3.2**

car 5.1

frog -1.7

Scores = unnormalized log prob. of the classes

$$P(Y = k|X = x_i) = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \quad \text{where} \quad s = f(x, W)$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$L_i = -\log P(Y = y_i|X = x_i)$$

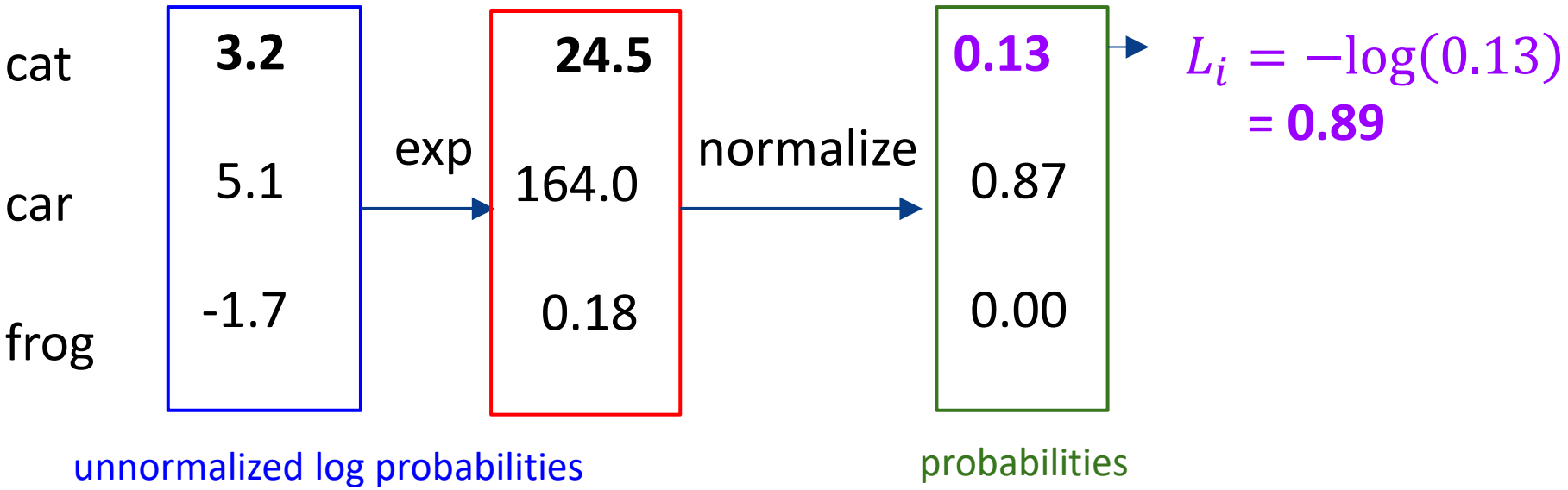
In summary:

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

Softmax Classifier (Multinomial Logistic Regression)

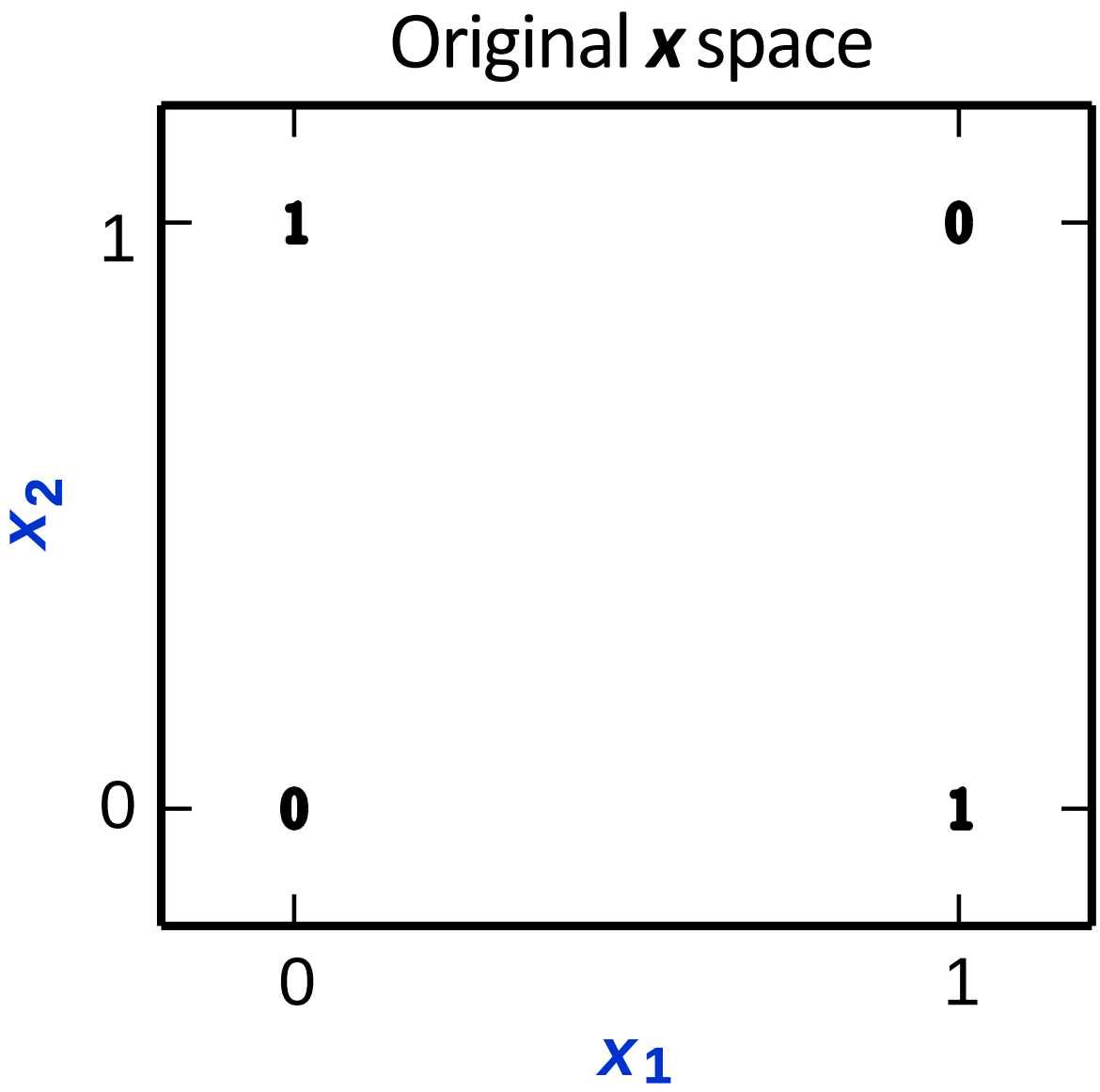
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities



Slides based on cs231n by Fei-Fei Li & Andrej Karpathy & Justin Johnson

XOR is not linearly separable



Hidden Input

- Hidden input with an activation function.
- Several features of the input
- Each feature defined using an activation function
- Linear function on the data – followed by a nonlinear activation function

Multilayer Networks

- Deep feedforward networks
- Feedforward neural networks
- Multilayer perceptrons (MLPs)

- Defines a mapping $y = f(x; \theta)$
  - Learns θ that result in the best function approximation
  - FFNs are typically represented by composing together many different functions.
  - The model is associated with a directed acyclic graph describing how the functions are composed together.

- For example, we might have three functions $f^{(1)}$, $f^{(2)}$ and $f^{(3)}$ connected in a chain, to form

$$f(x) = f^{(1)}(f^{(2)}(f^{(3)}(x)))$$

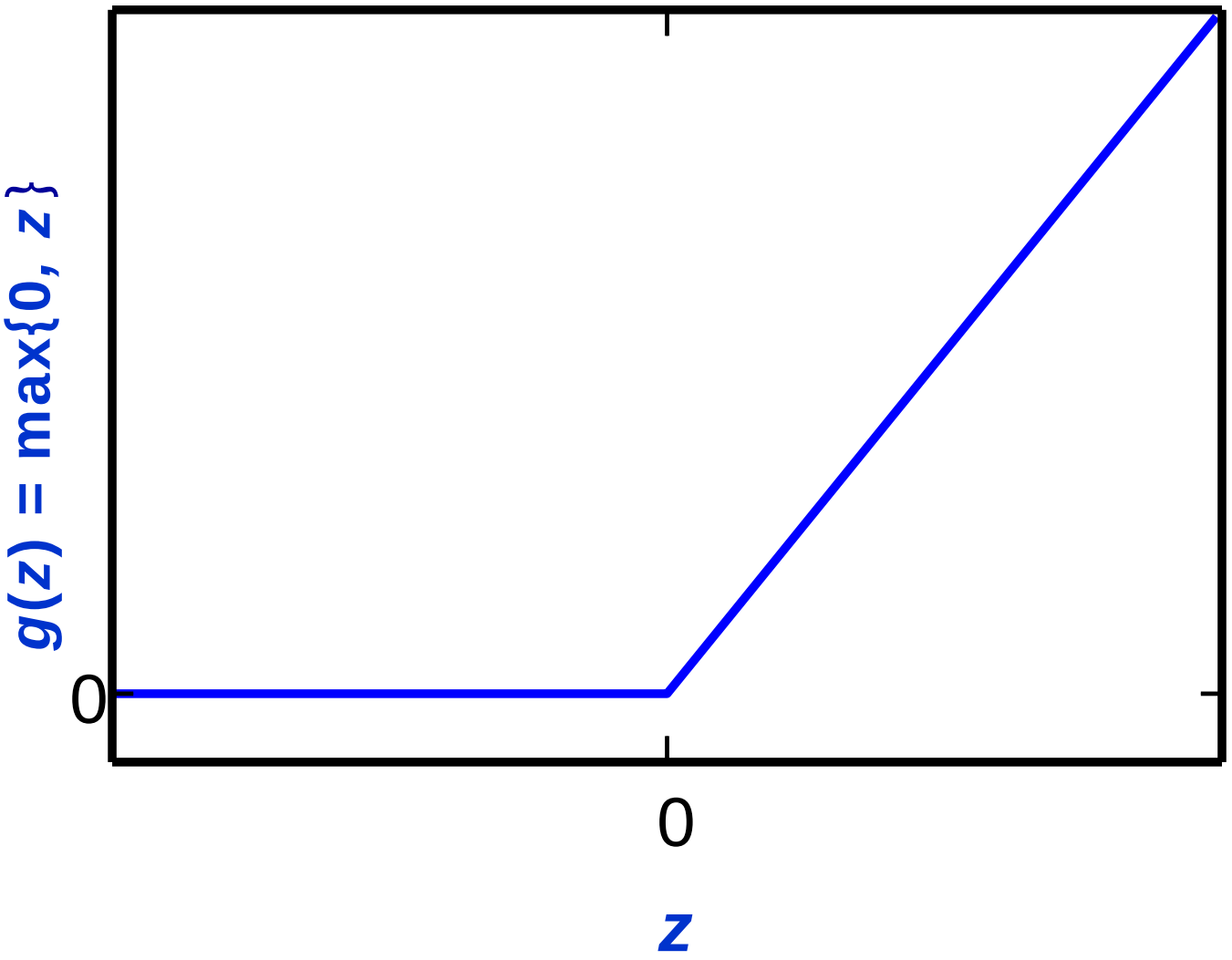
$f^{(1)}$: first layer

$f^{(2)}$: second layer

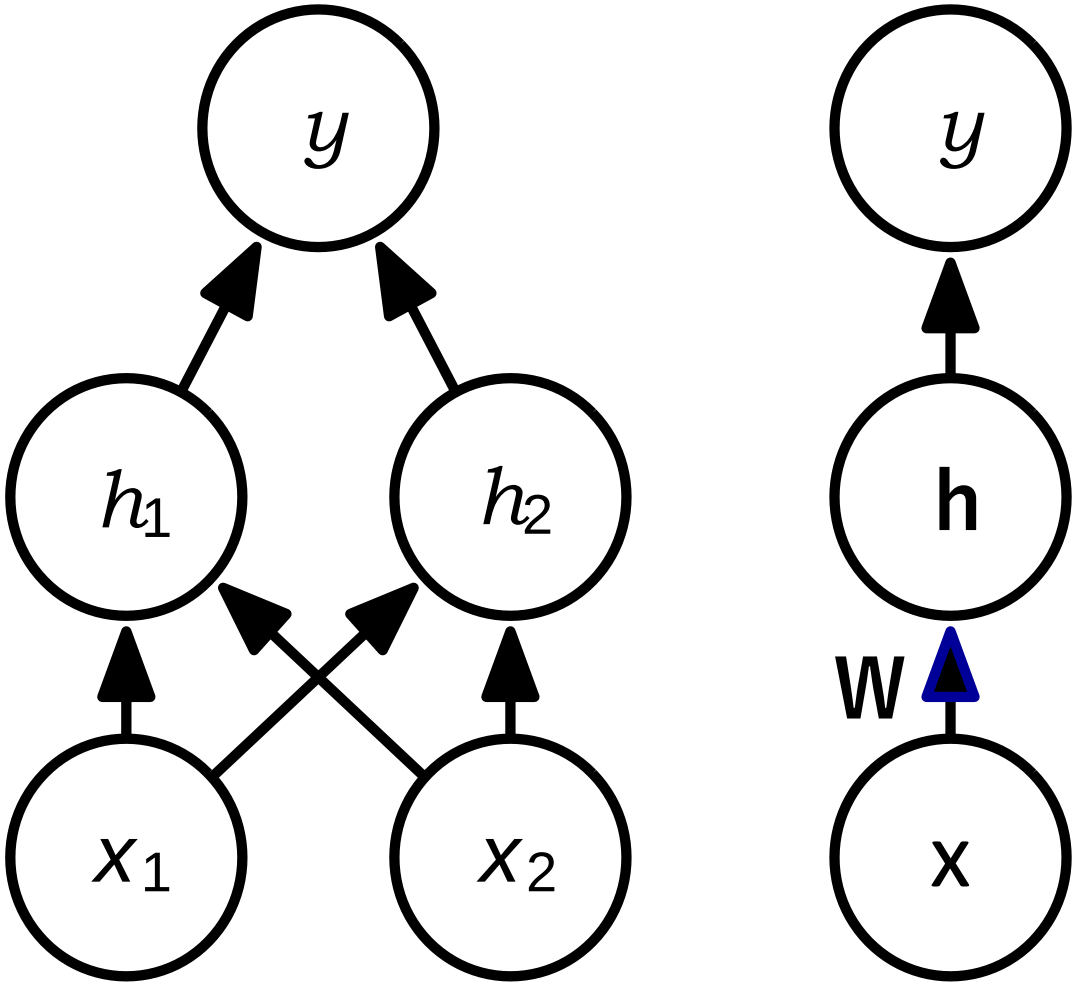
$f^{(3)}$: third layer

- Depth = length of the chain
- During neural network training, we drive f(x) to match f*(x).
- The training data provides us with noisy, approximate examples of f*(x) evaluated at different training points
- Because
- the training data does not show the desired output for each of these layers, these layers are called hidden layers

Rectified Linear Activation

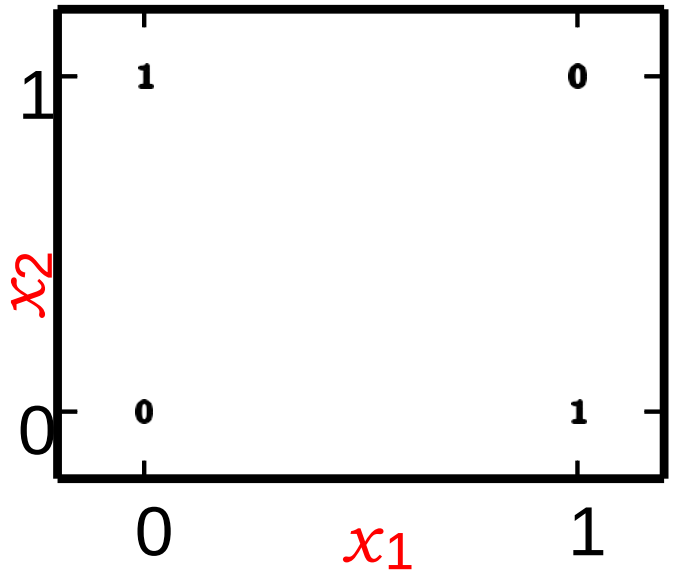


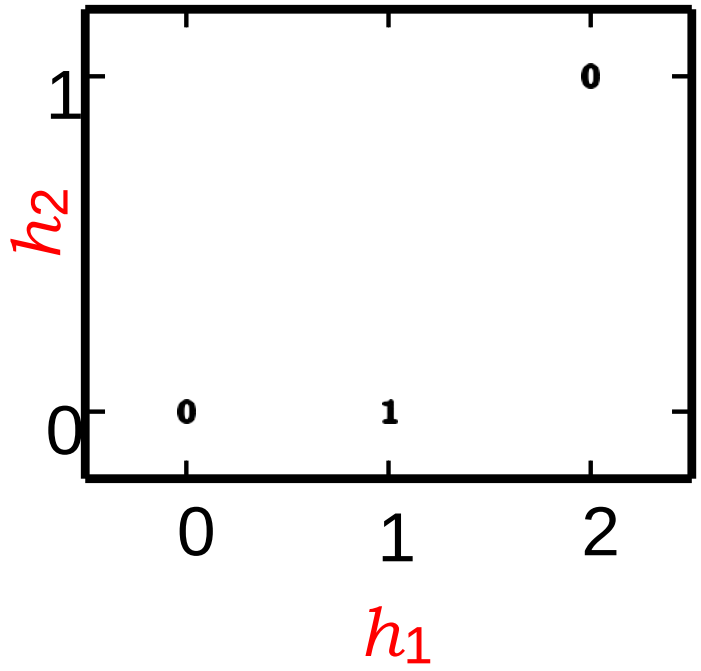
Network Diagrams



Solving X-OR

- $f(x; W, c, w, b) = w^T . max\{0, W^T x + c\} + b$
- $W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$
- $c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$
- $w = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$
- $b = 0$





Gradient-Based Learning

- Specify
  - Model
  - Cost (smooth)
  - Minimize cost using gradient descent or related techniques

- The nonlinearity of a neural network causes most interesting loss functions to become nonconvex.
- Stochastic gradient descent applied to nonconvex loss functions has no convergence guarantee and is sensitive to the values of the initial parameters.
- Initialize all weights to small random values. The biases may be initialized to zero.

Gradient

- In 1-d, the derivative of a function:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- In multiple dimensions, the gradient is the vector of (partial derivatives).

Evaluating the gradient numerically

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

```python
def eval_numerical_gradient(f, x):
  """
  a naive implementation of numerical gradient of f at x
  - f should be a function that takes a single argument
  - x is the point (numpy array) to evaluate the gradient at
  """

  fx = f(x) # evaluate function value at original point
  grad = np.zeros(x.shape)
  h = 0.00001

  # iterate over all indexes in x
  it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
  while not it.finished:

    # evaluate function at x+h
    ix = it.multi_index
    old_value = x[ix]
    x[ix] = old_value + h # increment by h
    fxh = f(x) # evalute f(x + h)
    x[ix] = old_value # restore to previous value (very important!)

    # compute the partial derivative
    grad[ix] = (fxh - fx) / h # the slope
    it.iternext() # step to next dimension

  return grad
```

Slides based on cs231n by Fei-Fei Li & Andrej Karpathy & Justin Johnson

The loss is just a function of W:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

$$\nabla_W L = \ldots$$