

# Computational Intelligence

Samaneh Hosseini

Isfahan University of Technology

# Outline

- Multi-class classification
  - Softmax Regression
  - Training a Softmax Classifier
  - Loss Function
  - Gradient Descent with Softmax

# Multi-class classification:

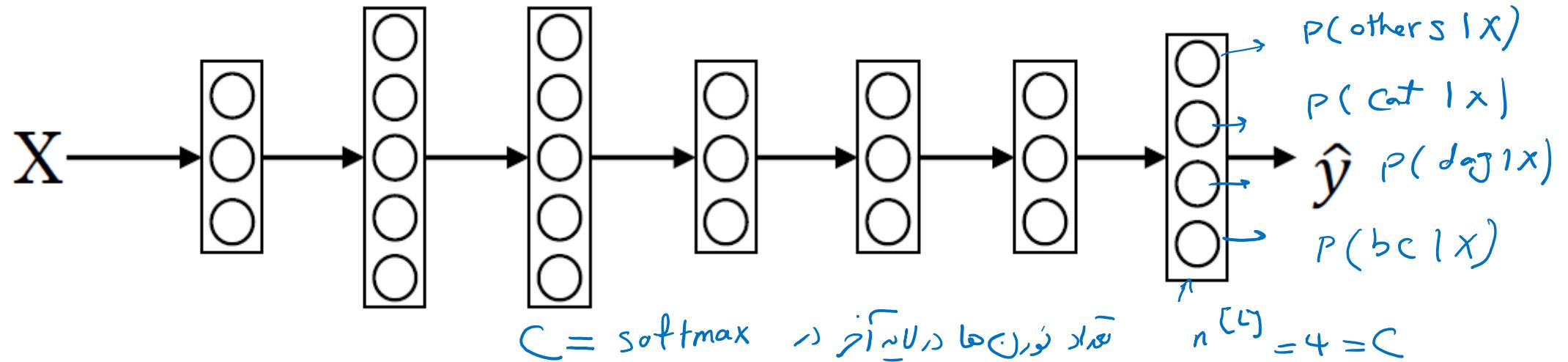
## Softmax Regression

# Recognizing cats, dogs, and baby chicks

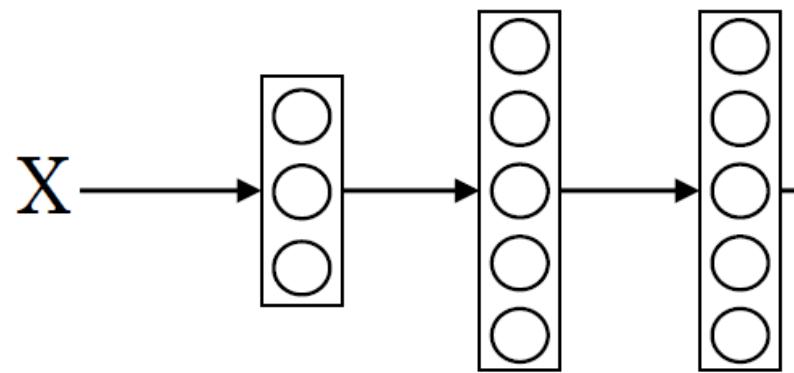


3            1            2            0            3            2            0            1

# classes = 4 = C



# Softmax layer



$$z^{(l)} = \omega^{(l)} a^{(l-1)} + b^{(l)} \quad (4 \times 1)$$

softmax activation function:

$$t = e^{(z^{(l)})} \rightarrow (4 \times 1)$$

activation function

$$a^{(l)} = g^{(l)}(z^{(l)}) \quad , \quad a_i^{(l)} = \frac{t_i}{\sum_{j=1}^4 t_j}$$

softmax layer

$$\begin{aligned} z^{(l)} &= \begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \end{bmatrix} \\ t &= \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^{-3} \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 0.1 \end{bmatrix} \\ a^{(l)} &= \frac{t}{176.3} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix} \end{aligned}$$

# Can You Use Argmax Activations Instead?

- The argmax function returns the index of the maximum value in the input array.
- Example:  $z = [5, 2, -1, 3]$  → image is predicted to be as others →  $a = \underline{[1, 0, 0, 0]}$
- Limitation:
  - gradients with respect to the raw outputs of the neural networks are always zero.
  - No learning during backpropagation as gradients are zero.

# Can You Use Argmax Activations Instead?

- From a probabilistic viewpoint:
- notice how the argmax function puts all the mass on index 1:
  - the predicted class and 0 elsewhere.
  - it's straightforward to infer the predicted class label from the argmax output.
  - However, we would like to know how likely the image is to be that of a cat, a dog, or a baby chicks,
  - The softmax scores help us with just that!

# Tips on Softmax

- Applying softmax preserves the relative ordering of scores.
- All entries in the softmax output vector are between 0 and 1.
- the classes are mutually exclusive
  - One input can not belong to two class
  - entries of the softmax output sum up to 1

# Softmax Implementation

```
1 import numpy as np
2
3 def softmax(z):
4     '''Return the softmax output of a vector.'''
5     exp_z = np.exp(z)
6     sum = exp_z.sum()
7     softmax_z = np.round(exp_z/sum, 3)
8     return softmax_z
```

# Why Won't Normalization by the Sum Suffice

- Why use something math-heavy as the softmax activation? Can we not just divide each of the output values by the sum of all outputs?

```
1 | def div_by_sum(z):  
2 |     sum_z = np.sum(z)  
3 |     out_z = np.round(z/sum_z,3)  
4 |     return out_z
```

# Why Won't Normalization by the Sum Suffice

1 Consider  $\mathbf{z1} = [0.25, 1.23, -0.8]$

```
1 | z1 = [0.25, 1.23, -0.8]
2 | div_by_sum(z1)
3 |
4 | # Output
5 | array([ 0.368, 1.809, -1.176])
```



We still aren't able to interpret the entries as probability scores.

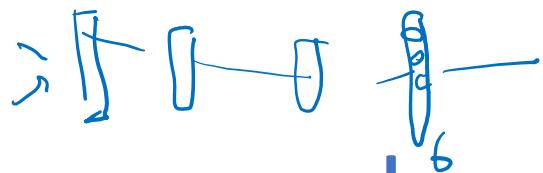
# Why Won't Normalization by the Sum Suffice

2 Let ~~z2~~ = [-0.25, 1, -0.75].

```
1 z2 = [-0.25, 1, -0.75]
2 div_by_sum(z2)
3
4 # Output
5 RuntimeWarning: divide by zero encountered in true_divide
6 array([-inf, inf, -inf])
```

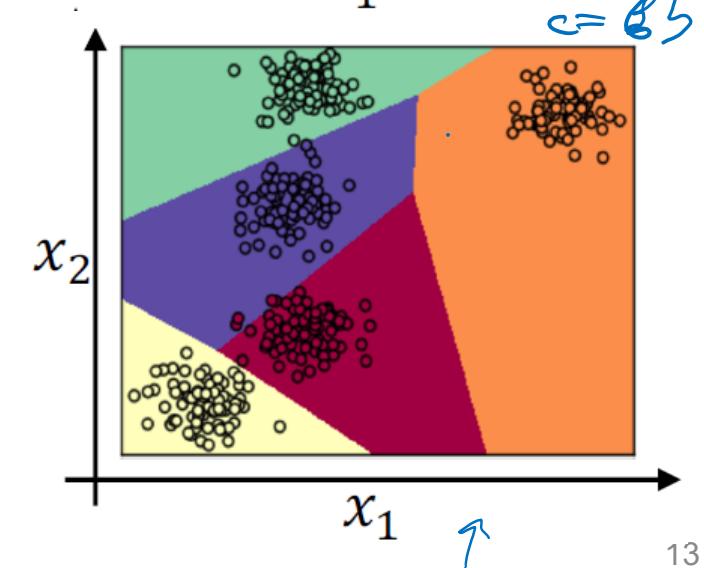
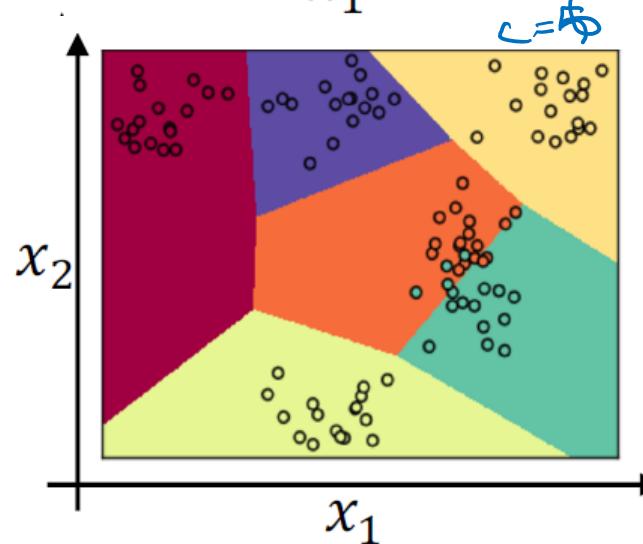
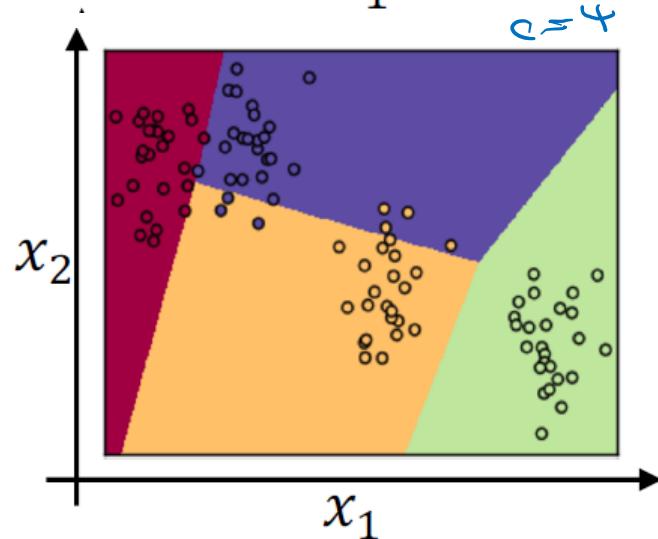
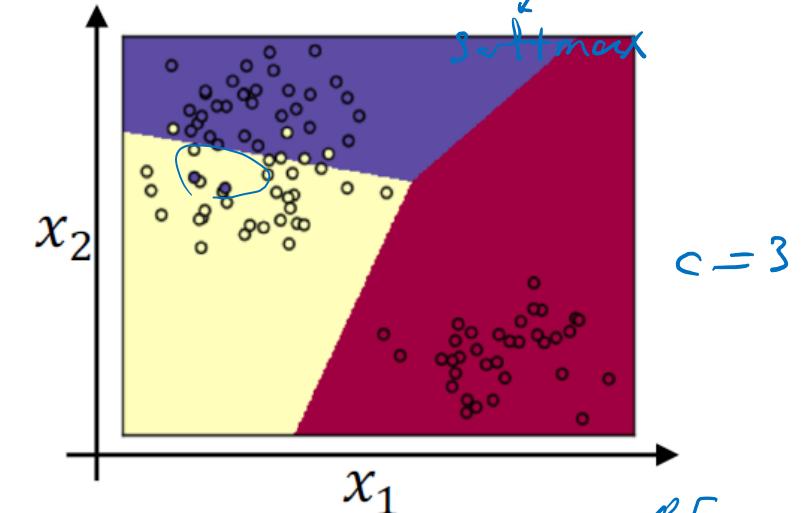
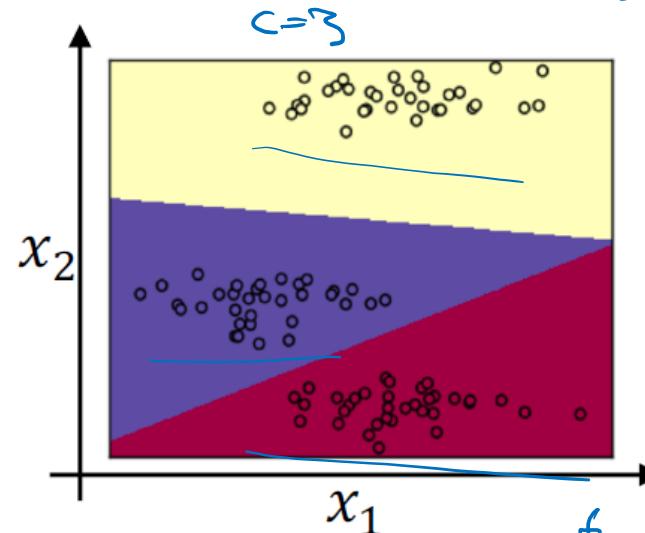
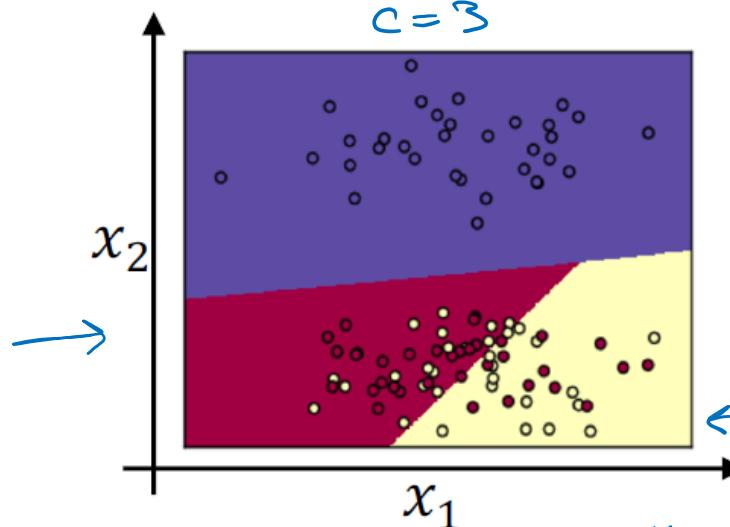
When you divide by the sum to normalize, you'll face runtime warnings, as division by zero is not defined.

# Softmax examples



$$z^{[1]} = \omega^{[1]} x + b^{[1]}$$

$$\alpha^{[1]} = \hat{y} = g(z^{[1]})$$



# Multi-class classification: Training a Softmax Classifier

# Understanding Softmax

$$z^{(c)} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$$

$$t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} \quad c=4$$

$$a^{(c)} = g^{(c)}(z^{(c)}) = \begin{bmatrix} e^5 / (e^5 + e^2 + e^{-1} + e^3) \\ e^2 / (e^5 + e^2 + e^{-1} + e^3) \\ e^{-1} / (e^5 + e^2 + e^{-1} + e^3) \\ e^3 / (e^5 + e^2 + e^{-1} + e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

$$g^{(c)}(\cdot)$$

hard max

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Softmax regression generalized form of logistic regression

If  $c=2$

$$\begin{bmatrix} e^1 / (e^1 + e^2) \\ e^2 / (e^1 + e^2) \end{bmatrix} \times \begin{bmatrix} 1 \\ 1-e^1 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

# Understanding Softmax

There's only one node as the other is not given any weight at all. The raw output vector now becomes  $z = [z, 0]$ .

$$\text{softmax}(\mathbf{z})_0 = \frac{e^z}{e^0 + e^z} = \frac{e^z}{1 + e^z} = \sigma(z)$$
$$\text{softmax}(\mathbf{z})_1 = \frac{e^0}{e^0 + e^z} = \frac{1}{1 + e^z} = 1 - \sigma(z)$$

# Loss Function

$$J^{(1)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \rightarrow \text{Cat}$$

$$\alpha^{[l]} J^{(1)} = \hat{y}^{(1)} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}^{(4 \times 1)}$$

$$J(\omega^{[1]}, b^{[1]}, \dots) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)})$$

$$\underbrace{L(\hat{y}, y)}_{\text{Small}} = - \sum_{j=1}^4 y_j \log \hat{y}_j$$

$\hat{y}_2 \log \hat{y}_2 = -\log \hat{y}_2$

small

$\Rightarrow \hat{y}_2 \text{ big}$

$$Y = [J^{(1)} \ J^{(2)} \ \dots \ J^{(n)}]_{4 \times m}$$

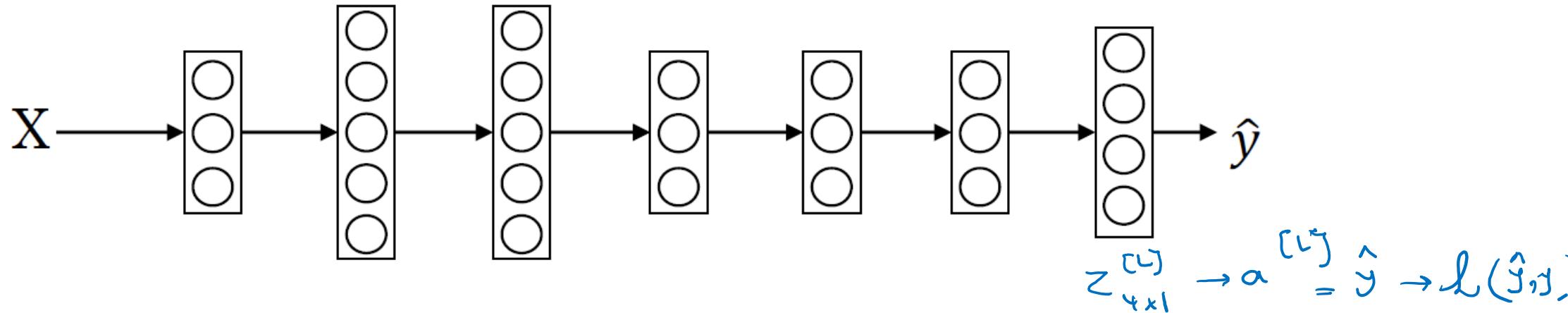
$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \dots$$

$$\hat{Y} = [\hat{J}^{(1)} \ \dots \ \hat{J}^{(n)}]$$

$$= \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \dots$$

4  $\sqrt{m}$  17

# Gradient Descent with Softmax



$$\text{Backprob : } \frac{\partial \hat{y}}{\partial z^{[l]}} = \hat{y} - y$$

$$\frac{\partial \hat{y}}{\partial z^{[l]}}$$

# Core foundation Review

- Multi-class classification
  - Softmax Regression
  - Training a Softmax Classifier

# Core Foundation Review

- Multi-class classification
  - Softmax Regression
  - Training a Softmax Classifier
  - Loss Function
  - Gradient Descent with Softmax