Machine Learning for Robotics: **Classification**

Prof. Navid Dadkhah Tehrani

WORCESTER POLYTECHNIC INSTITUTE
1865
WPI

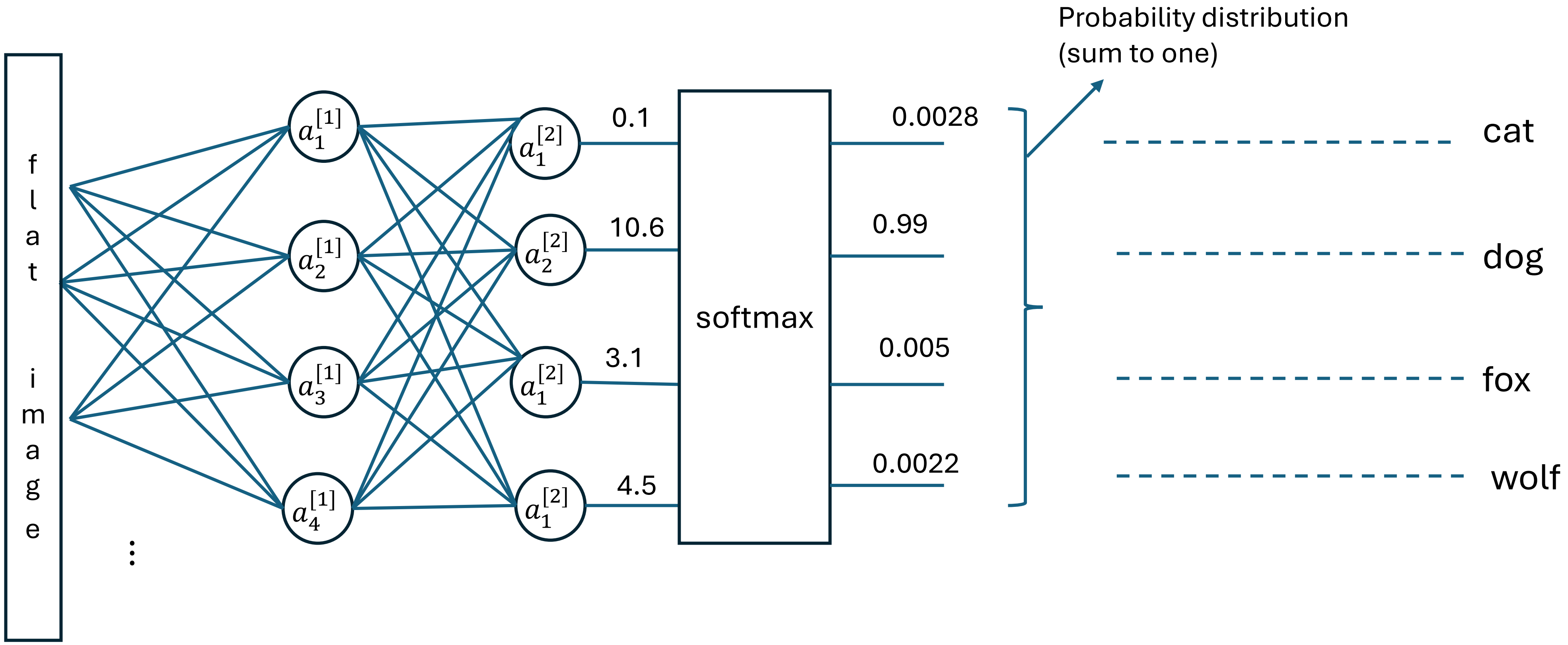
In regression, we're predicting a float value as output.

In classification, we're predicting a class type that's an integer number. For example, what object is in the image out of 100 possible objects ($i \in [1,2, \dots, 100]$)

We have two problems:

- The output of the neural network is a float number (tensor) not an integer.
- How do we define the continuous and differentiable loss function for classification?

assume that we want to classify an image in 4 categories (cat, dog, fox, wolf)
Say the image is 32-by-32 gray scale. We first flatten the image into a 1024 vector.
Then we need is a neural network that output 4 values.
We then pass the 4 output to a softmax function, to convert the output to a probability distribution.



<u>Definition of softmax</u>

$$\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{o}) \quad \text{where} \quad \hat{y}_i = \frac{\exp(o_i)}{\sum_j \exp(o_j)}.$$

Example: $\mathrm{o} = \begin{bmatrix} 1 \\ 5 \\ 10 \\ 0.1 \end{bmatrix} \rightarrow \hat{y} = \begin{bmatrix} 0.001 \\ 0.0067 \\ 0.9931 \\ 0.000 \end{bmatrix}$

Softmax turn a vector into a probability distribution.
Softmax is also a soft max! in other words, it’s continuous and differentiable way of taking maximum of bunch of numbers.

One-hot encoding of class labels

before we define the loss function, we must define one-hot encoding of class labels.

	cat class	Dog class	Fox class	Wolf class
Training example #1	1	0	0	0
Training example #2	0	0	1	0
Training example #3	1	0	0	0
Training example #4	0	0	1	0
…	0	1	0	0
…	1		0	0
…	0	0	1	0
…	0	0	0	1
Training example #n	0	0	0	1

Integer class label
1
3
1
3
2
1
3
4
4

Cross-entropy Loss function

$$L = \sum_{i=1}^{n}\sum_{j=1}^{h} -y_j^{[i]}\log(\hat{y}_j^{[i]}) \qquad or \qquad L = \sum_{i=1}^{n} -y_j \,.\, \log(\hat{y}_j) \qquad ('.' \; is \; dot \; product)$$

n: $\quad number \; of \; training \; examples$

h: $\quad number \; of \; classes$

$\hat{y}_j^{[i]}$: $\quad predicted \; output \; of \; the \; NN$

$y_j^{[i]}$: $\quad$ onehot encoding of the label

To understand this, let assume $n = 6$ (batch size) .

$y_1 = [1, 0, 0, 0], y_2 = [0,1,0,0], y_3 = [0,0,1,0], y_4 = [0,0,0,1] \;, y_5 = [1,0,0,0], y_6 = [1,0,0,0]$

And the output of NN is:

$\hat{y}$ =

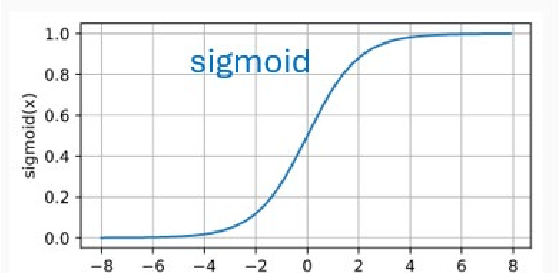
0.8	0.1	0.1	0
0	0.5	0.2	0.3
0	0	0.9	0.1
0.9	0	0	0.1
0.8	0	0.1	0.1
0.7	0.1	0.1	0.1

As it can be seen, the cross-entropy loss is minimized only if the NN predicts all the classes correctly.

Cross-entropy loss is a maximum likelihood estimator as opposed to MSE loss that we saw before.

To simplify the derivation, assume binary classification. For example, is this an image of a cat or not.
This is also called **logistic regression**.
The NN has now only one output. And the output goes through a sigmoid function:

$$\hat{y} = \sigma(w^T x + b)$$



$$\mathrm{sigmoid}(x) = \frac{1}{1+\exp(-x)}.$$

$$P(y|x) = \begin{cases} \sigma(w^T x + b) & if\ y = 1 \\ 1 - \sigma(w^T x + b) & if\ y = 0 \end{cases}$$

Can be written in compact form:

$$p(y|x) = \left(\sigma(w^T x + b)\right)^y \left(1 - \sigma(w^T x + b)\right)^{1-y}$$

We want to maximize the probability of predicting the correct class (y), given all n training inputs (x).
In other words, a classifier is a good classifier if it predicts **all the training examples** with high accuracy.

$$
\begin{aligned}
L(\mathbf{w}) &= P(\mathbf{y} \mid \mathbf{x}; \mathbf{w}) \\
&= \prod_{i=1}^{n} P\left(y^{(i)} \mid x^{(i)}; \mathbf{w}\right) \\
&= \prod_{i=1}^{n} \left(\sigma\left(z^{(i)}\right)\right)^{y^{(i)}} \left(1 - \sigma\left(z^{(i)}\right)\right)^{1-y^{(i)}}
\end{aligned}
$$

$$
\begin{aligned}
l(\mathbf{w}) &= \log L(\mathbf{w}) \\
&= \sum_{i=1}^{n} \left[y^{(i)} \log\left(\sigma\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - \sigma\left(z^{(i)}\right)\right)\right]
\end{aligned}
$$

Some final notes:

Pytorch has the cross-entropy loss and it automatically does the one-hot encoding of the class labels.

Logits:
refer to the raw, unnormalized scores output by a model before applying softmax.

We can show analytically that when taking derivative of the cross-entropy loss w.r.t the weights of the NN we get a clean expression for gradient:

$\nabla_w L = -X^T(Y - A)^T, \qquad W \in R^{k\times m}, X \in R^{n\times m}, A \in R^{n\times h}, Y \in R^{n\times h}$

One-hidden layer.
n: *batch size*
h: *number of nerons in the hidden unit hidden*
Y: *onehot encoded matrix*
m: *input size*

But if we had used MSE loss, the gradient would have been very flat → difficult training.