# ULTIWAY CLASSIFICATION

① Recall the architecture for a 3-layer feedforward neural network:

$$\begin{tikzcd}
 & \dot{W} \arrow[d] & & \ddot{W} \arrow[d] & & \dddot{W} \arrow[d] & \\
x \arrow[r] & \dot{\pi} \arrow[r] & \dot{x} \arrow[r] & \ddot{\pi} \arrow[r] & \ddot{x} \arrow[r] & \dddot{\pi} \arrow[r] & o
\end{tikzcd}$$

② So far we've seen a couple different instances of this final "output layer":

$$\begin{tikzcd}
 & & \dddot{W} \arrow[d] & \\
{} \arrow[r] & \ddot{x} \arrow[r] & \dddot{\pi} \arrow[r] & o
\end{tikzcd}$$

$\begin{pmatrix} \text{linear} \\ \text{regression} \\ o = \dddot{\pi}_1 \end{pmatrix}$

$$\begin{tikzcd}
 & \begin{bmatrix} \dddot{w}_{11} \\ \vdots \\ \dddot{w}_{H1} \end{bmatrix} \arrow[d] & \\
\begin{bmatrix} \ddot{x}_1 & \cdots & \ddot{x}_H \end{bmatrix} \arrow[r] & \begin{bmatrix} \dddot{\pi}_1 \end{bmatrix} \arrow[r] & \dddot{\pi}_1
\end{tikzcd}$$

$\begin{pmatrix} \text{logistic} \\ \text{regression} \\ o = \sigma(\dddot{\pi}_1) \end{pmatrix}$

logistic sigmoid function

$$\begin{tikzcd}
 & \begin{bmatrix} \dddot{w}_{11} \\ \vdots \\ \dddot{w}_{H1} \end{bmatrix} \arrow[d] & \\
\begin{bmatrix} \ddot{x}_1 & \cdots & \ddot{x}_H \end{bmatrix} \arrow[r] & \begin{bmatrix} \dddot{\pi}_1 \end{bmatrix} \arrow[r] & \sigma(\dddot{\pi}_1)
\end{tikzcd}$$

③ These output layers address two distinct tasks:

- regression: the output variable is an unbounded real number (e.g. a child's height)
- classification: the output variable is a probability (e.g. of whether someone comes down with a particular disease)

# MULTIWAY CLASSIFICATION

④ But sometimes you want to classify something into one of several discrete categories. For instance, given an image of an animal, we might want to automatically identify whether it is a horse, a zebra, or a panda.

- multiway classification: the output variable is a member of a finite, unordered set (e.g. {horse, zebra, panda}).

⑤ How do we represent the response variable $y^{(n)}$ for a multiway classification task? Strings aren't particularl convenient:

| X (evidence vector) | y (response) |
|---|---|
| $x^{(1)}$ | "horse" |
| $x^{(2)}$ | "zebra" |
| $x^{(3)}$ | "zebra" |
| $x^{(4)}$ | "panda" |
| $x^{(5)}$ | "horse" |

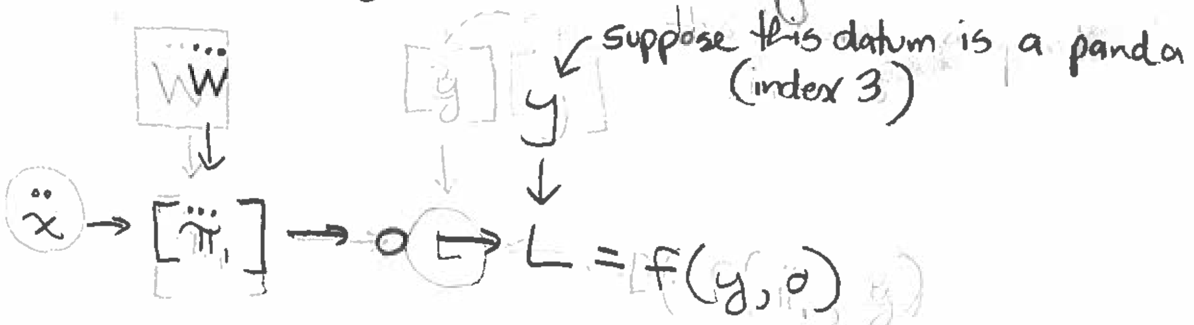
because how do we compare a string with our output vector $\dddot{\pi}$?

# MULTIWAY CLASSIFICATION

⑥ We could represent each response as its index in an ordered list of the possible categories, e.g. for $\langle \overset{1}{\text{horse}}, \overset{2}{\text{zebra}}, \overset{3}{\text{panda}} \rangle$:

| X (evidence vector) | y (response) |
|---|---|
| $x^{(1)}$ | 1 |
| $x^{(2)}$ | 2 |
| $x^{(3)}$ | 2 |
| $x^{(4)}$ | 3 |
| $x^{(5)}$ | 1 |

⑦ If we do this, then we end up comparing numbers to numbers, but in a way that's kind of weird.



The loss needs to be some differentiable function of response and output $o$. If this subject is a panda, then we want to reward "panda predictions" $o$. Let's say we just use the simple loss:

$$L = (y - o)^2$$

# MULTIWAY CLASSIFICATION

⑧ That means we want $\hat{o}$ to be close to 3 for panda images. That's ok, but it doesn't penalize horses and zebras equally. For a zebra, the loss is:

$$L = (3-2)^2 = 1$$

while for a horse, the loss is:

$$L = (3-1)^2 = 4$$

---

⑨ Essentially, if we use the index of an arbitarily ordered list to represent our response, we are implicitly saying that neighbors in the list (e.g. panda, ze are "closer" than elements that are further apart (e.g. panda and horse).

It's hard to imagine a loss function L that doesn't impose this bias, if we need L to be differentiable.

# MULTIWAY CLASSIFICATION

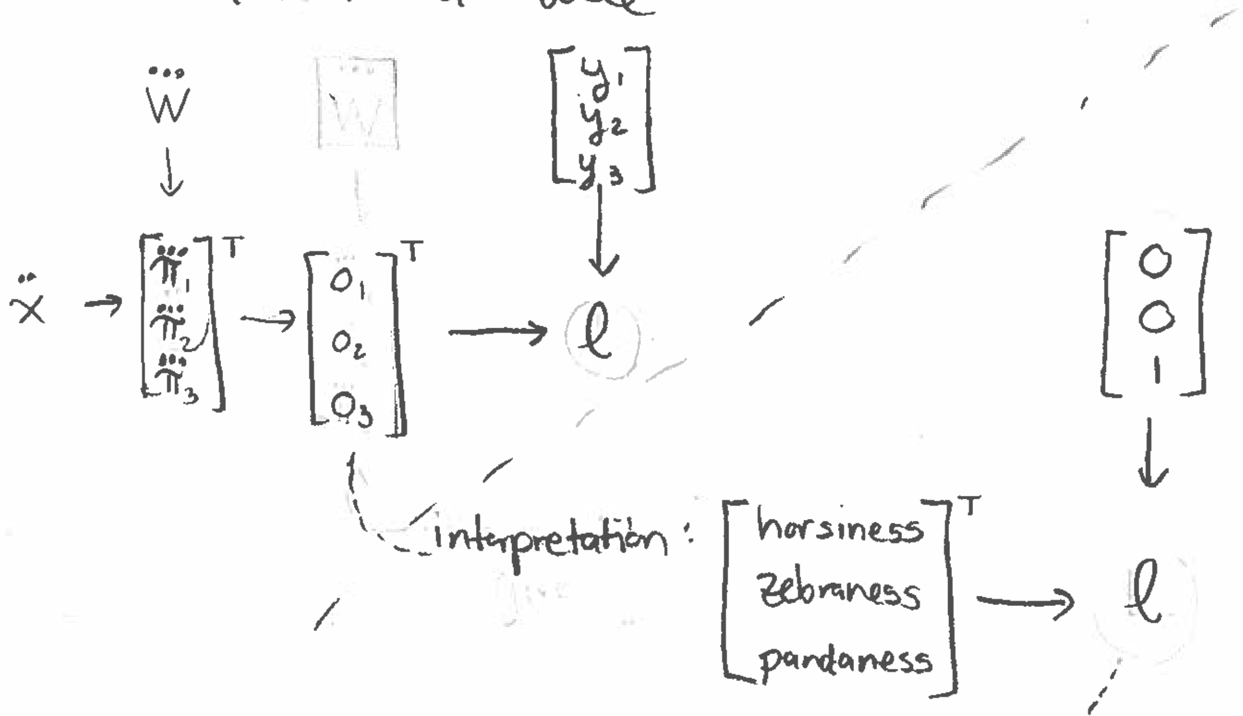
⑩ Back to the drawing board. How else calld we represent the response? What if, again we provide an ordered list of the categories (e.g. < horse, zebra, panda >), but now we represent the response as a vector whose kth element is equal to 1 if the response is the kth element of the list?

| X<br>(evidence vector) | y<br>(response) | |
|---|---|---|
| $x^{(1)}$ | $\begin{bmatrix}1\\0\\0\end{bmatrix}$ | ← "horse" vector |
| $x^{(2)}$ | $\begin{bmatrix}0\\1\\0\end{bmatrix}$ | ← "zebra" vector |
| $x^{(3)}$ | $\begin{bmatrix}0\\1\\0\end{bmatrix}$ | ← "zebra" vector |
| $x^{(4)}$ | $\begin{bmatrix}0\\0\\1\end{bmatrix}$ | ← "panda" vector |
| $x^{(5)}$ | $\begin{bmatrix}1\\0\\0\end{bmatrix}$ | ← "horse" vector |

These are referred to as "one-hot" vectors.

# MULTIWAY CLASSIFICATION

⑪ If we go this route, then we'd want our output $\dddot{\pi}$ to be a vector as well:



loss L should penalize outputs with a high horsiness or zebraness, and reward outputs with high pandaness

---

⑫ One straightforward implementation of this intuition is to take the output vector, replace its max value with 1, and replace the other values w. 0:

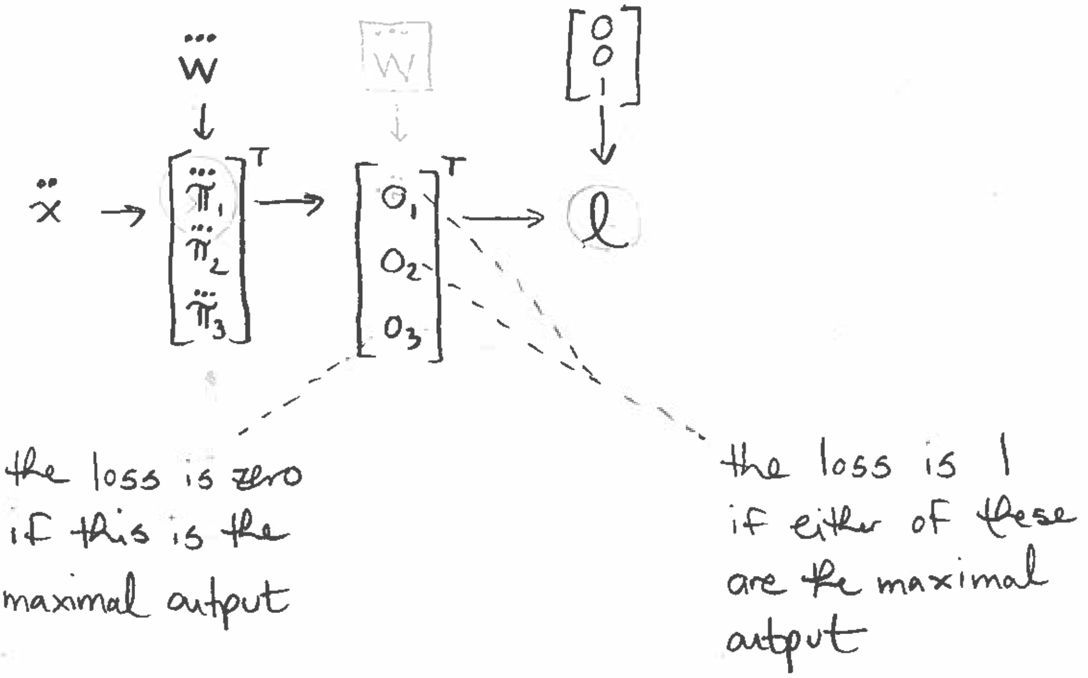
e.g. $\begin{bmatrix} 2.4 \\ 4.2 \\ 1.0 \end{bmatrix}^T \rightarrow \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}^T$ $\quad$ $\begin{bmatrix} 1.2 \\ -2.5 \\ 1.5 \end{bmatrix}^T \rightarrow \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}^T$

Then, take the dot product of the resulting vector with the response:

e.g. $\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}^T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0$ $\quad$ $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}^T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 1$

# MULTIWAY CLASSIFICATION

⑬ This gives us a reward of 1 if the maximal value that is output by the neural network coincides with the "true" category (and a reward of zero otherwise). To convert this into a loss function, we can just take 1 minus the reward.



---

⑭ We can formalize this by defining onehot(k, d) to be the d-dimensional one-hot vector whose $k^{th}$ element is 1, e.g. $\text{onehot}(2,3) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$, $\text{onehot}(1,3) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$.

Then we define the loss as:

$$L = -\log\left(y^T \cdot \text{onehot}\left(\underset{i}{\text{argmax}}\ \dddot{\pi}_i,\ C\right)\right)$$

where C is the number of categories (e.g. C=3 in our running example).

⑮ There's only one problem. This argmax function isn't differentiable. We can't compute $\frac{\partial o}{\partial \dddot{\pi}}$, so we can't compute $\frac{\partial \ell}{\partial \theta}$, and thus we can't use gradient descent to optimize the weights $\theta$.

---

⑯ But can we find an alternative loss function that is similar in spirit, but which <u>is</u> differentiable?

First, observe that our "hard max" function is essentially mapping the vector $\dddot{\pi}$ to a probability distribution:

$$\begin{bmatrix} 1.2 \\ -2.5 \\ 3.1 \end{bmatrix} \longrightarrow \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

only thing is — all the probability mass is concentrated on one value.

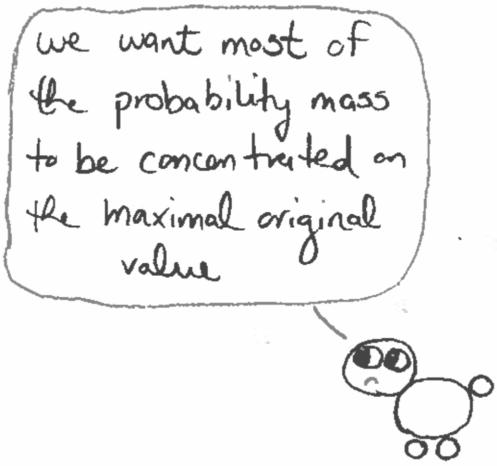
So one idea would be to map $\dddot{\pi}$ to a probability distribution for which <u>most</u> of the probability mass is concentrated on one value, e.g.

$$\begin{bmatrix} 1.2 \\ -2.5 \\ 3.1 \end{bmatrix} \longrightarrow \begin{bmatrix} .130 \\ .003 \\ .867 \end{bmatrix}$$

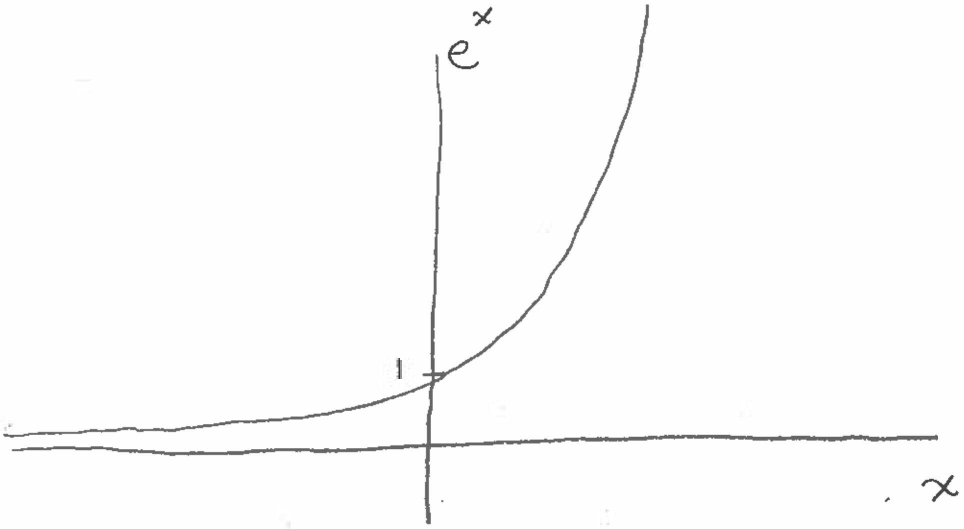
# MULTIWAY CLASSIFICATION

⑰ How do we map a vector of reals into a probability distribution such that most of the probability mass is concentrated on the highest original value?

$$\begin{bmatrix} 1.2 \\ -2.5 \\ 3.1 \end{bmatrix}$$



⑱ One cool trick is to notice that the exponential function maps the real numbers to strictly positive numbers:



It also does so in a way that magnifies the differences between the original numbers.

# MULTIWAY CLASSIFICATION

⑲ For instance, applying the exponential function, we get

$$\begin{bmatrix} 1.2 \\ -2.5 \\ 3.1 \end{bmatrix} \xrightarrow{\text{exponentiate}} \begin{bmatrix} 3.32 \\ 0.08 \\ 22.2 \end{bmatrix}$$

Now that we have a vector of positive numbers, we can simply normalize to get a probability distribution

$$\begin{bmatrix} 1.2 \\ -2.5 \\ 3.1 \end{bmatrix} \xrightarrow{\text{exponentiate}} \begin{bmatrix} 3.32 \\ 0.08 \\ 22.2 \end{bmatrix} \xrightarrow{\text{normalize}} \begin{bmatrix} .130 \\ .003 \\ .867 \end{bmatrix}$$

⑳ This function is referred to as <u>softmax</u>:

$$\text{softmax}\left(\begin{bmatrix} z_1 \\ \vdots \\ z_N \end{bmatrix}\right) = \begin{bmatrix} \frac{e^{z_1}}{\sum_n e^{z_n}} \\ \vdots \\ \frac{e^{z_N}}{\sum_n e^{z_n}} \end{bmatrix}$$

# MULTIWAY CLASSIFICATION

㉑ Retrofitting our loss function from ⑮ to use <u>softmax</u> instead of <u>hardmax</u>, we get:

$$\ell = +\log\left(y^T \cdot \text{softmax}(\dddot{\pi})\right)$$

---

(22a) Examples:

→ if output $\dddot{\pi} = \begin{bmatrix} 1.2 \\ -2.5 \\ 3.1 \end{bmatrix}$ and response $y = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

then:

$$\ell = -\log \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} .130 \\ .003 \\ .867 \end{bmatrix} = -\log .867 \approx 0.062$$

→ if output $\dddot{\pi} = \begin{bmatrix} 1.2 \\ -2.5 \\ 3.1 \end{bmatrix}$ and response $y = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

then:

$$\ell = -\log \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} .130 \\ .003 \\ .867 \end{bmatrix} = -\log .003 \approx 2.52$$

In other words, the loss is the $\overset{\text{negative log of the}}{\vee}$ total probability mass accorded to correct response.