CS-466/566: Math for AI

Module 07: Deep Learning Fundamentals-3

Dr. Mahmoud Mahmoud
The University of Alabama

2026-03-26

TABLE OF CONTENTS

Neural Networks Definition (a.k.a. Deep Learning)

A deep learning model is a **computational graph** that try to map inputs, each drawn from some dataset with common characteristics to outputs drawn from a related distribution.

It is a graph that has **many** layers (Simply Nested Functions).

That is why gradient descent is a bit tricky here.

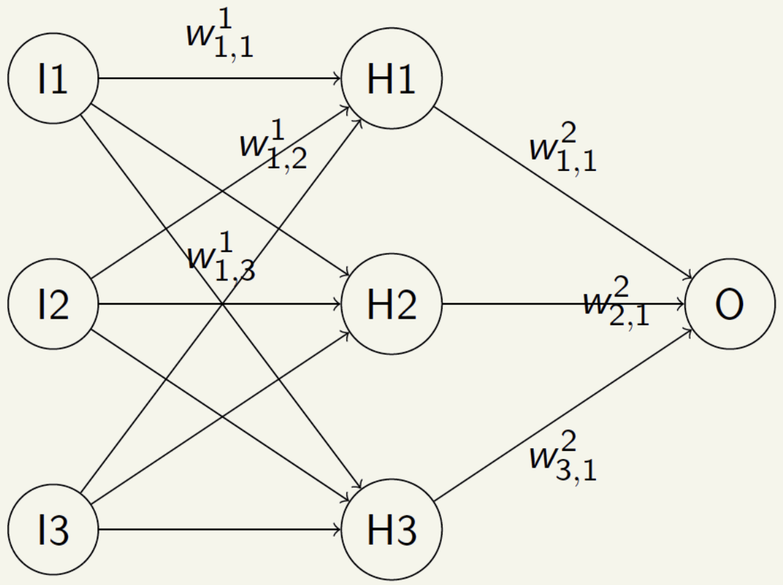
Regression vs Classification Neural Networks

CLASSIFICATION

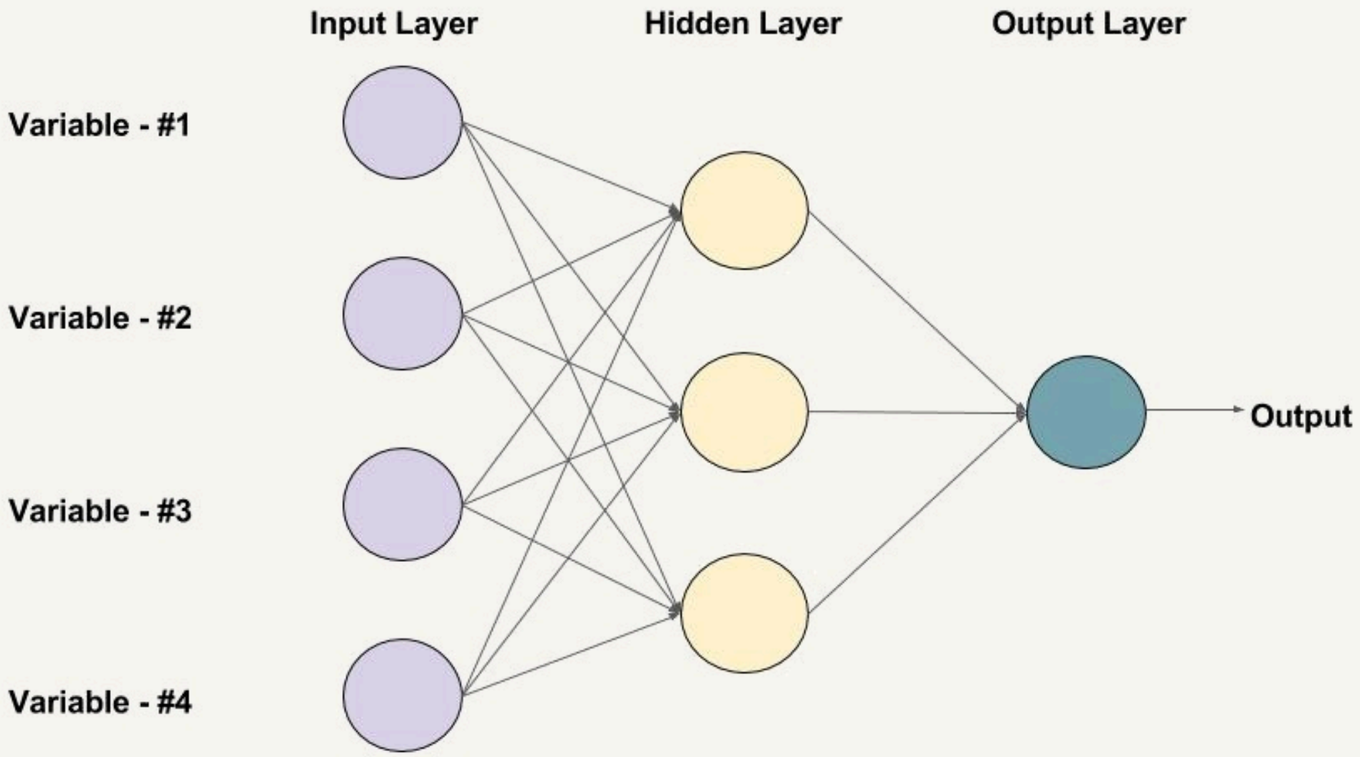
Use a **sigmoid** (or another squashing nonlinearity) on a **single** output so the readout behaves like a probability or score — that setup is **classification**.

REGRESSION

Use a **linear** readout from the hidden layer, e.g. $\sum_i w^2_{i,1} H_i$, to predict a **real number** — that setup is **regression**.



Binary Classification

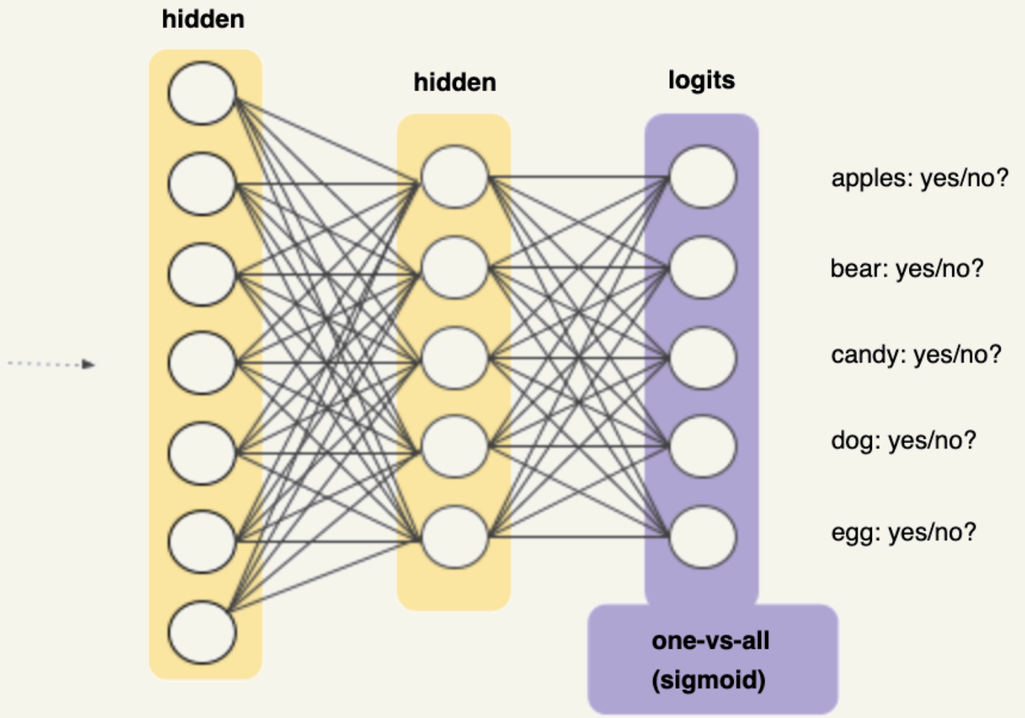


An example of a Feed-forward Neural Network with one hidden layer (with 3 neurons)

Binary Classification Neural Network

(One vs All) Multi-class Classification

Given a classification problem with N possible solutions, a one-vs.-all solution consists of N separate **binary classifiers**



Note. The output vector has to be encoded using **one hot encoding**

Training a Neural Network

- L is the loss function.
- Y is the target value.
- $L(Y, O)$ is the loss between the target value and the output.
- We need to find each $\frac{\partial L}{\partial w_{ij}^{k}}$
- Calculus chain rule is needed to find the derivatives.

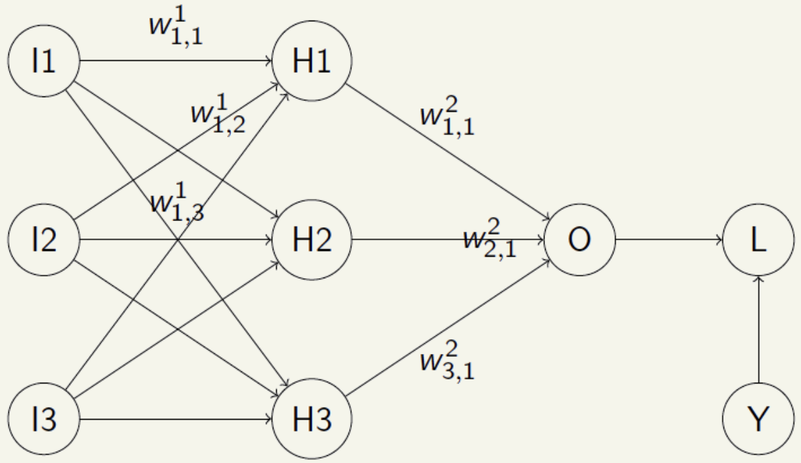


TABLE OF CONTENTS

More Accurate Multi-class Classification

Previously, we used sigmoid with mean squared error (MSE) as loss function

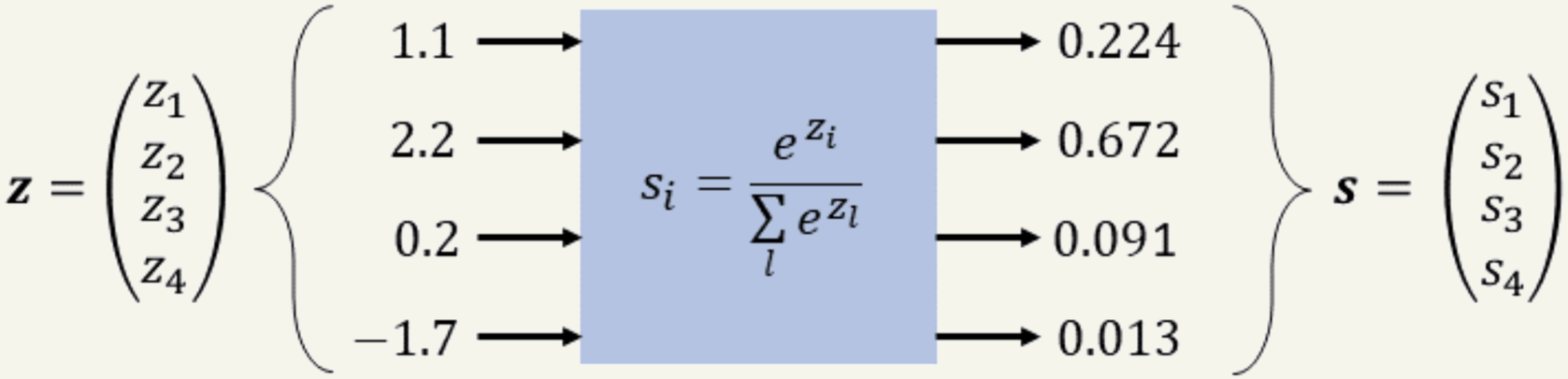
- **Pros**
  - Convex (i.e., steeper gradient as you are away from the correct output)
  - Acceptable performance
- **Cons**
  - Outputs can not be interpreted as probability.
  - Does not work with all problem types

In practice, we use Softmax + cross Entropy loss

Softmax Function

- Softmax extends binary logistic regression idea (probability adds up to 1) into a multi-class world.
- It helps training converge more quickly than it otherwise would.

Let Z denote the logits (the raw, unnormalized scores output by the network before applying softmax).



Softmax Function

How it works?

Suppose we have an input vector from the previous layer

$$[5, 3, 2]$$

One way to transform these values into a vector of probabilities

$$\text{Normalize}\begin{pmatrix}\begin{bmatrix}x_1\end{bmatrix}\\ x_2\\ \begin{bmatrix}x_3\end{bmatrix}\end{pmatrix} = \begin{bmatrix}\frac{x_1}{x_1+x_2+x_3}\end{bmatrix}\\ \frac{x_2}{x_1+x_2+x_3}\\ \begin{bmatrix}\frac{x_3}{x_1+x_2+x_3}\end{bmatrix}$$

A better way

$$\text{Softmax}\begin{pmatrix}\begin{bmatrix}x_1\end{bmatrix}\\ x_2\\ \begin{bmatrix}x_3\end{bmatrix}\end{pmatrix} = \begin{bmatrix}\frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}}\end{bmatrix}\\ \frac{e^{x_2}}{e^{x_1}+e^{x_2}+e^{x_3}}\\ \begin{bmatrix}\frac{e^{x_3}}{e^{x_1}+e^{x_2}+e^{x_3}}\end{bmatrix}$$

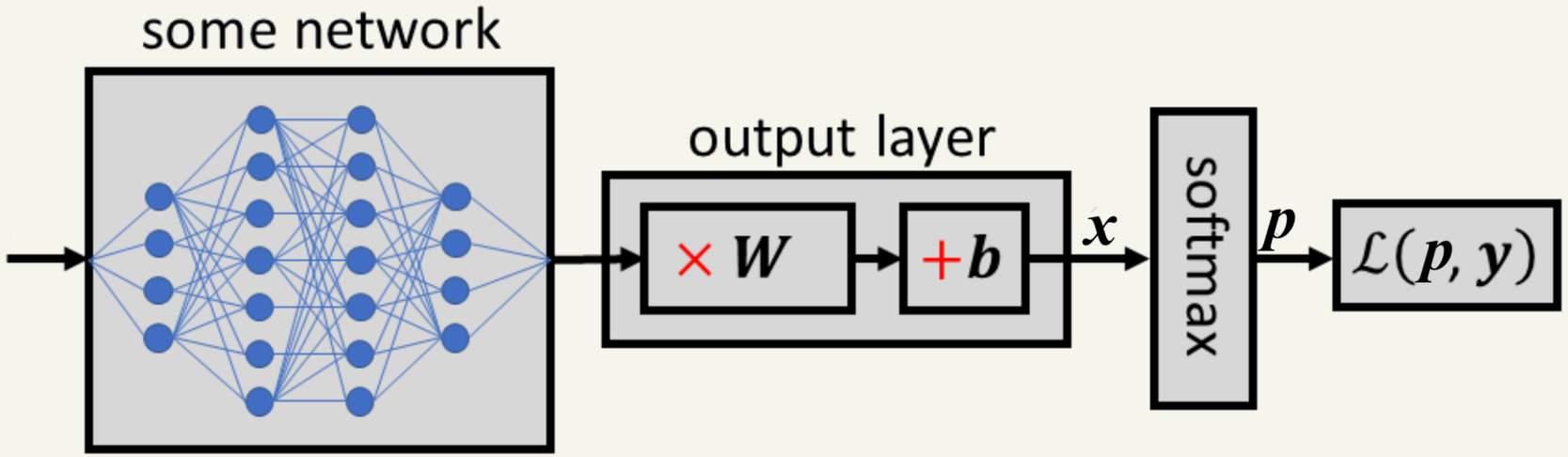
Intuition

```
normalize(np.array([5,3,2]))

array([0.5, 0.3, 0.2])

softmax(np.array([5,3,2]))

array([0.84, 0.11, 0.04])
```

Softmax Intuition

- Softmax is more strongly amplifies the maximum value relative to the other values.
- Softmax is partway between normalizing the values and actually applying the **max** function!

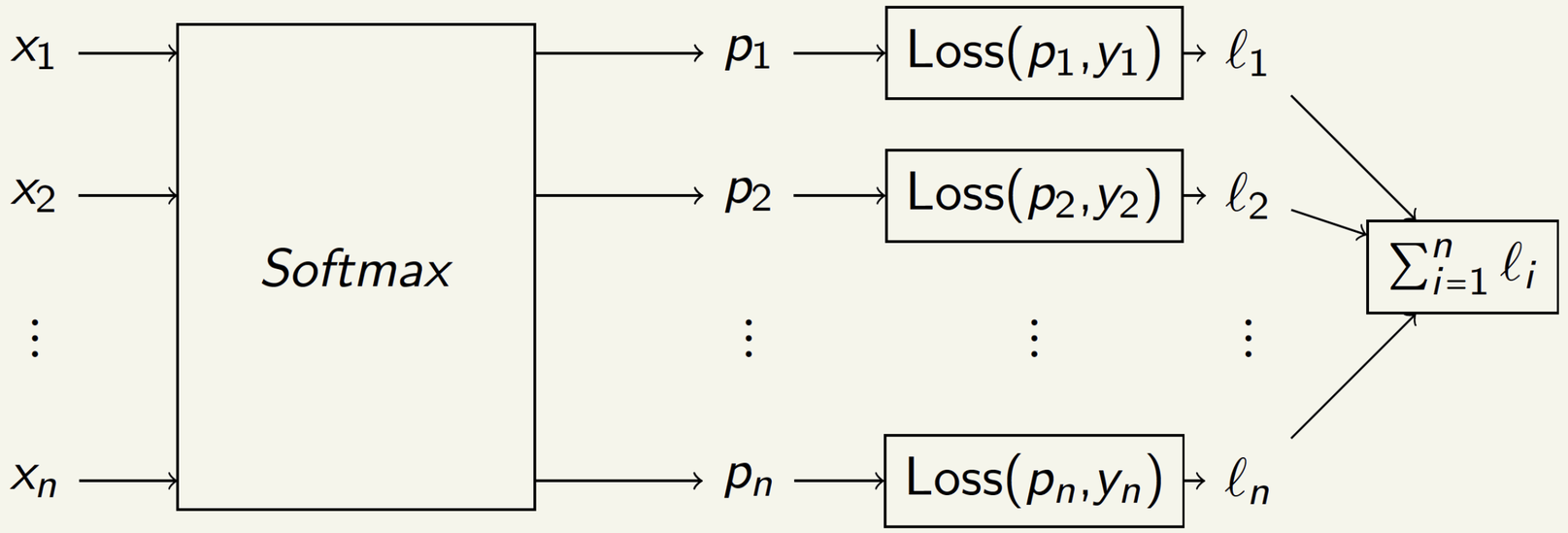
Our Neural Network



Neural Network Architecture

Loss Function Diagram View

- The incoming vector $\mathbf{x}$ is transformed by the **softmax** function to produce the vector of probabilities $\mathbf{p}$.
- The vector of probabilities $\mathbf{p}$ is then compared to the vector of actual values $\mathbf{y}$ using the **cross entropy loss** function resulting in a scalar loss value.



Cross Entropy

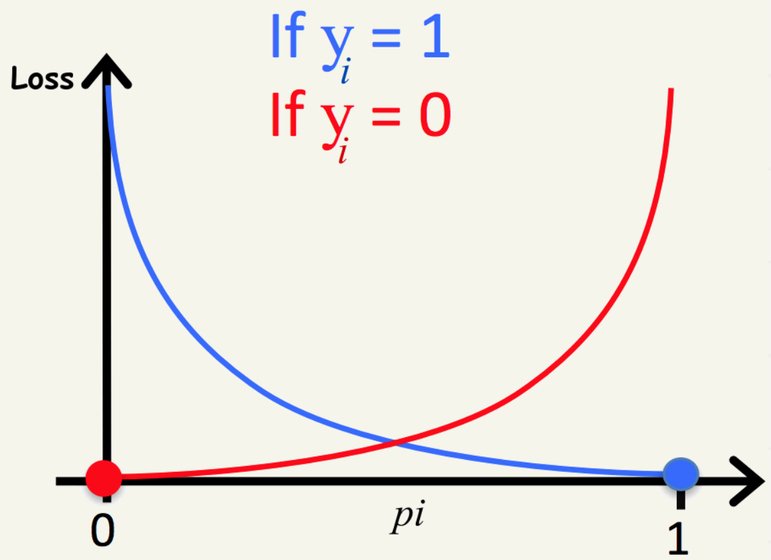
Recall that any loss function will take in a vector of probabilities $\begin{bmatrix} p_1 \\ \vdots \\ p_n \end{bmatrix}$ and a vector of actual values $\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$

The cross entropy loss function, for each index i in these vectors, is:

$$\ell_i = -y_i log(p_i) - (1 - y_i) log(1 - p_i)$$

Intuition

- Since p_i is a probability between 0 and 1.
- Our loss can be defined as follows.
  - if $y_i = 1$: Loss = $-log(p_i)$
  - if $y_i = 0$: Loss = $-log(1 - p_i)$



Gradient Computation

The real magic happens when we combine this loss with the softmax function

$$-\ell_1 = y_1 \log(p_1) + (1 - y_1) \log(1 - p_1)$$

$$\Downarrow$$

$$-\ell_1 = y_1 \log\left(\frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}}\right) + (1 - y_1) \log\left(1 - \frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}}\right)$$

Based on this expression, the gradient would seem to be a bit trickier for this loss. (**Proof Is Required**)

$$\frac{\partial L_i}{\partial x_i} = p_i - y_i$$

Proof: $\frac{\partial L}{\partial x} = p - y$

$$-\ell_1 = y_1 \log\left(\frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}}\right) + (1-y_1)\log\left(1 - \frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}}\right)$$

Proof.

1. **Split logarithm of fraction for first term**

$$-\ell_1 = y_1 \left[\log(e^{x_1}) - \log(e^{x_1} + e^{x_2} + e^{x_3})\right] + (1-y_1)\log\left[1 - \frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}}\right]$$

2. **Simplify** $\log(e^{x_1}) = x_1$ **and rewrite complement fraction**

$$-\ell_1 = y_1 x_1 - y_1 \log(e^{x_1} + e^{x_2} + e^{x_3}) + (1-y_1)\log\left[\frac{e^{x_2}+e^{x_3}}{e^{x_1}+e^{x_2}+e^{x_3}}\right]$$

3. **Split logarithm of fraction in last term**

$$-\ell_1 = y_1 x_1 - y_1 \log(e^{x_1} + e^{x_2} + e^{x_3}) + (1-y_1)\log(e^{x_2} + e^{x_3}) - (1-y_1)\log(e^{x_1} + e^{x_2} + e^{x_3})$$

4. **Combine like terms with** $\log(e^{x_1} + e^{x_2} + e^{x_3})$

$$-\ell_1 = y_1 x_1 + (1-y_1)\log(e^{x_2} + e^{x_3}) - \log(e^{x_1} + e^{x_2} + e^{x_3})$$

5. **Take derivative with respect to** x_1

$$-\frac{\partial \ell_1}{\partial x_1} = y_1 - \frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}}$$

6. **Recognize softmax probability definition** $p_1 = \frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}}$

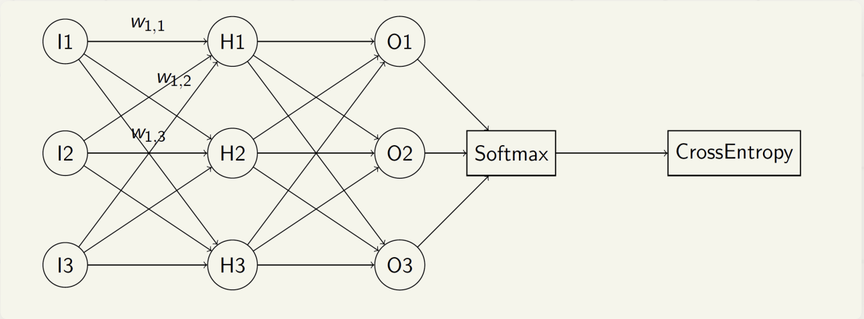
$$\frac{\partial \ell_1}{\partial x_1} = p_1 - y_1 \quad \text{(derivative of the log-likelihood } \ell_1\text{)}$$

TABLE OF CONTENTS

Neural Network As Computational Graph

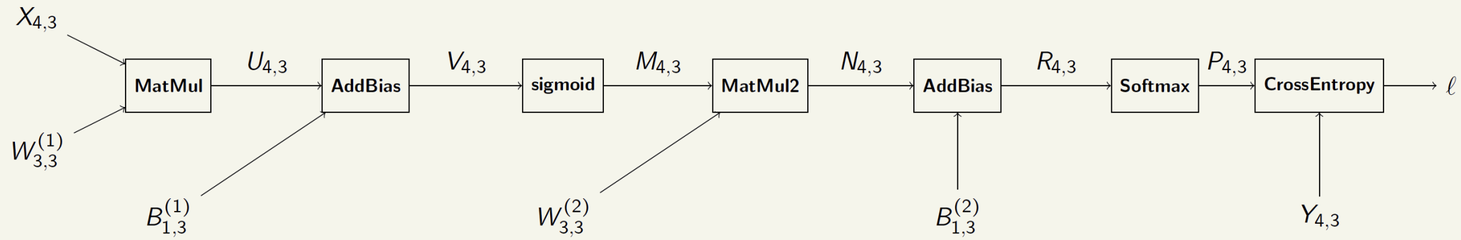
Two-layer network for multi-class classification (softmax output).

- **Architecture:** one **hidden** layer (here with **sigmoid** activations) and one **output** layer whose activations are **logits** N; applying **softmax** yields class probabilities P (e.g. dog / cat / horse, or 3-way softmax in the diagram).
- **Task:** predict a **probability vector** over classes — this is **multi-class** classification, not binary.



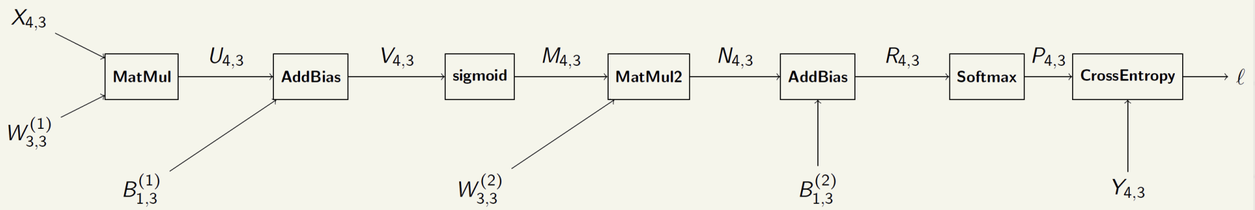
Neural Network As Computational Graph

Neural Network Computational Graph



- $\mathbf{X}$ is a 4x3 matrix (4 samples, 3 features)
- $\mathbf{W}^{(1)}$ is a 3x3 matrix
- $\mathbf{B}^{(1)}$ is a 1x3 vector
- $\mathbf{W}^{(2)}$ is a 3x3 matrix
- $\mathbf{B}^{(2)}$ is a 1x3 vector

Computational Graph: $\partial L/\partial W^{(2)}_{3\times 3}$



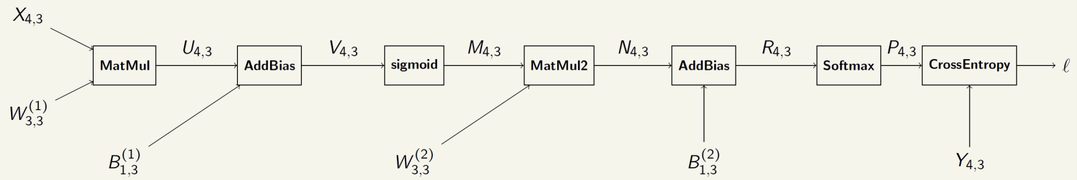
Compute the gradient of L (cross-entropy loss) with respect to $W^{(2)}_{3,3}$ (backward path: $L \leftarrow N \leftarrow W^{(2)}$; N are logits before softmax).

CHAIN RULE

$$\frac{\partial L}{\partial W^{(2)}_{3,3}} = \frac{\partial L}{\partial R_{4,3}} \odot \frac{\partial R_{4,3}}{\partial N_{4,3}} \odot \frac{\partial N_{4,3}}{\partial W^{(2)}_{3,3}}$$ (softmax + CE upstream gradient $\partial L/\partial R = P - Y$)

1. $\frac{\partial L}{\partial R_{4,3}} = P_{4,3} - Y_{4,3}$
2. $\frac{\partial R_{4,3}}{\partial N_{4,3}} = \mathbf{1}_{4,3}$
3. $\frac{\partial N_{4,3}}{\partial W^{(2)}_{3,3}} = \text{transpose}(M)_{3,4} \cdot \mathbf{1}_{4,3}$

Computational Graph: $\partial L / \partial W^{(1)}_{3\times 3}$



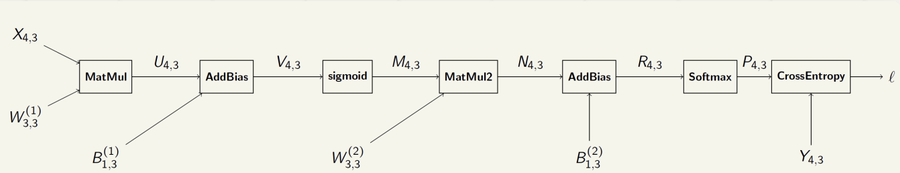
Compute the gradient of L with respect to $W^{(1)}_{3,3}$ (full backward path through hidden sigmoid: $L \leftarrow N \leftarrow M \leftarrow V \leftarrow U \leftarrow W^{(1)}$).

CHAIN RULE

$$\frac{\partial L}{\partial W^{(1)}_{3,3}} = \frac{\partial L}{\partial R_{4,3}} \odot \frac{\partial R_{4,3}}{\partial N_{4,3}} \odot \frac{\partial N_{4,3}}{\partial M_{4,3}} \odot \frac{\partial M_{4,3}}{\partial V_{4,3}} \odot \frac{\partial V_{4,3}}{\partial U_{4,3}} \odot \frac{\partial U_{4,3}}{\partial W^{(1)}_{3,3}}$$

1. $\frac{\partial L}{\partial R_{4,3}} = P_{4,3} - Y_{4,3}$
2. $\frac{\partial R_{4,3}}{\partial N_{4,3}} = \mathbf{1}_{4,3}$
3. $\frac{\partial N_{4,3}}{\partial M_{4,3}} = \mathbf{1}_{4,3} \cdot (W^{(2)}_{3,3})^{\top}$
4. $\frac{\partial M_{4,3}}{\partial V_{4,3}} = sigmoid(V_{4,3}) \odot (\mathbf{1}_{4,3} - sigmoid(V_{4,3}))$
5. $\frac{\partial V_{4,3}}{\partial U_{4,3}} = \mathbf{1}_{4,3}$
6. $\frac{\partial U_{4,3}}{\partial W^{(1)}_{3,3}} = \text{transpose}(X)_{3,4} \cdot \mathbf{1}_{4,3}$

Computational Graph: $\partial L/\partial B^{(2)}_{1\times 3}$



Compute the gradient of L with respect to $B^{(2)}_{1,3}$ (bias added into logits R; broadcast 4×3).

CHAIN RULE

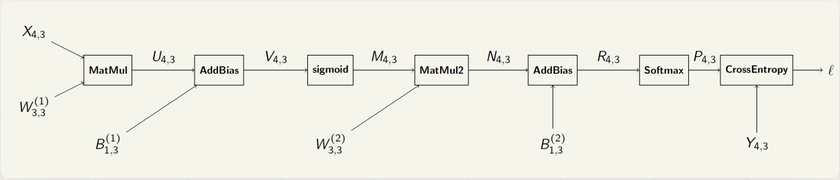
$$\frac{\partial L}{\partial B^{(2)}_{1,3}} = \frac{\partial L}{\partial R_{4,3}} \odot \frac{\partial R_{4,3}}{\partial B^{(2)}_{1,3}}$$

1 $\dfrac{\partial L}{\partial R_{4,3}} = P_{4,3} - Y_{4,3}$

2 $\dfrac{\partial R_{4,3}}{\partial B^{(2)}_{1,3}} = \mathbf{1}_{4,3}$

Note. $\odot$ gives 4×3 (one row per sample). $B^{(2)}_{1,3}$ is shared across rows — **sum across rows** (column-wise over the batch) for $\partial L/\partial B^{(2)}_{1,3}$.

Computational Graph: $\partial L/\partial B^{(1)}_{1\times 3}$



Compute the gradient of L with respect to $B^{(1)}_{1,3}$ (bias in V; broadcast 4×3).

CHAIN RULE

$$\frac{\partial L}{\partial B^{(1)}_{1,3}} = \frac{\partial L}{\partial R_{4,3}} \odot \frac{\partial R_{4,3}}{\partial N_{4,3}} \odot \frac{\partial N_{4,3}}{\partial M_{4,3}} \odot \frac{\partial M_{4,3}}{\partial V_{4,3}} \odot \frac{\partial V_{4,3}}{\partial B^{(1)}_{1,3}}$$

1. $\frac{\partial L}{\partial R_{4,3}} = P_{4,3} - Y_{4,3}$
2. $\frac{\partial R_{4,3}}{\partial N_{4,3}} = \mathbf{1}_{4,3}$
3. $\frac{\partial N_{4,3}}{\partial M_{4,3}} = \mathbf{1}_{4,3} \cdot (W^{(2)}_{3,3})^\top$
4. $\frac{\partial M_{4,3}}{\partial V_{4,3}} = sigmoid(V_{4,3}) \odot (\mathbf{1}_{4,3} - sigmoid(V_{4,3}))$
5. $\frac{\partial V_{4,3}}{\partial B^{(1)}_{1,3}} = \mathbf{1}_{4,3}$

Note. $\odot$ gives 4×3 (one row per sample). $B^{(1)}_{1,3}$ is shared across rows — **sum across rows** (column-wise over the batch) for $\partial L/\partial B^{(1)}_{1,3}$.

Thank You!