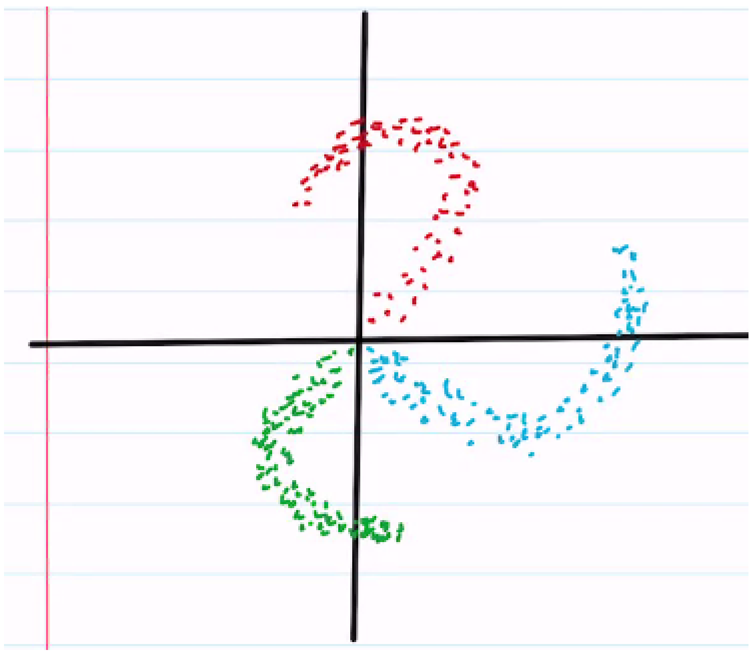
Day 8: Image Classification

Classification and the Tendril Problem

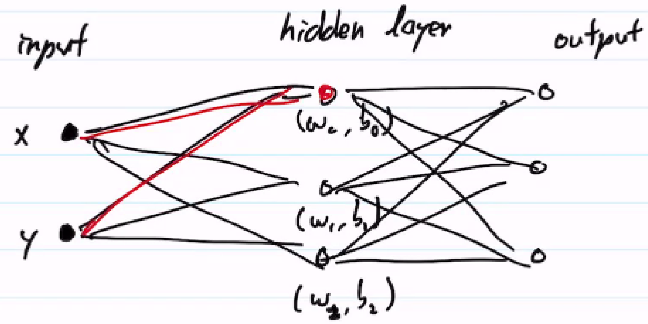


Example of Multidimensional Input/Output

- Example
    - input D_{in} : $\vec{x}$: $[x, y]$
    - output D_{out} : $\vec{y}$:
        - 1 value- what color is is of~ `datatype`
        - 3 values for RBG ~ `uint8`
        - 3 values for the probabilities ~ `P(R), P(G), P(B)`
        - $\vec{y} : [s_r, s_b, s_g]$

$$\vec{y} = \sum_{i=0}^{N-1} \vec{v}_i \phi(\vec{x}_i \cdot \vec{w}_i + b_i)$$

example: say $N = 3$ $\quad N$ = # of neurons in our 1-layer NN



$w_i = (2,)$

$\uparrow$

inputs into each neuron

b_i = scalar

equal to # of neurons

$w_i = (2,)$

$\uparrow$

inputs into each neuron

b_i = scalar

. # equal to # of neurons

. no relation to I/O size

$v_i = (3,)$

$\uparrow$

match output

$\underbrace{[s_r, s_b, s_g]}$

- How do we train in batches?

$$\vec{x} -> (\vec{x})^M$$
$$(2,) -> (M, 2)$$
$$\vec{y} -> (\vec{y})^M$$
$$(3,) -> (3, M)$$

$$(M, 2) \cdot (2, N) = (M, N)$$

- **Loss function**

$$Loss_{cross-entropy} = -\sum_{i=0}^{c-1} p_i^{(true)} log(p_i^{(pred)})$$

- Compares 2 probability distributions (true vs predicted)
- Minimized predictions when $\vec{p}^{(true)} = \vec{p}^{(pred)}$
  - Example:
    - $\vec{p}^{(true)} = [1, 0, 0]; p^{(pred)} = [0.75, 0.2, 0.05]$
    - In the above example, if the loss function encounters such values, it will have the effect of pushing the correct prediction up and the incorrect predictions down to minimize the truth/prediction differences.
- Softmax Cross-Entropy
  - $\vec{s}$ → softmax → $\vec{p}$ (scores are given to softmax which are converted into probability-like values)

$$softmax(\vec{s}) = [\frac{e^{s_0}}{\sum e^{s_j}}, \frac{e^{s_1}}{\sum e^{s_j}}, \frac{e^{s_2}}{\sum e^{s_j}} ...]$$

where

$$0 \leq \frac{e^{s_0}}{\sum e^{s_j}} \leq 1$$

$$\sum_k (\frac{e^k}{\sum e^{s_j}}) = 1$$

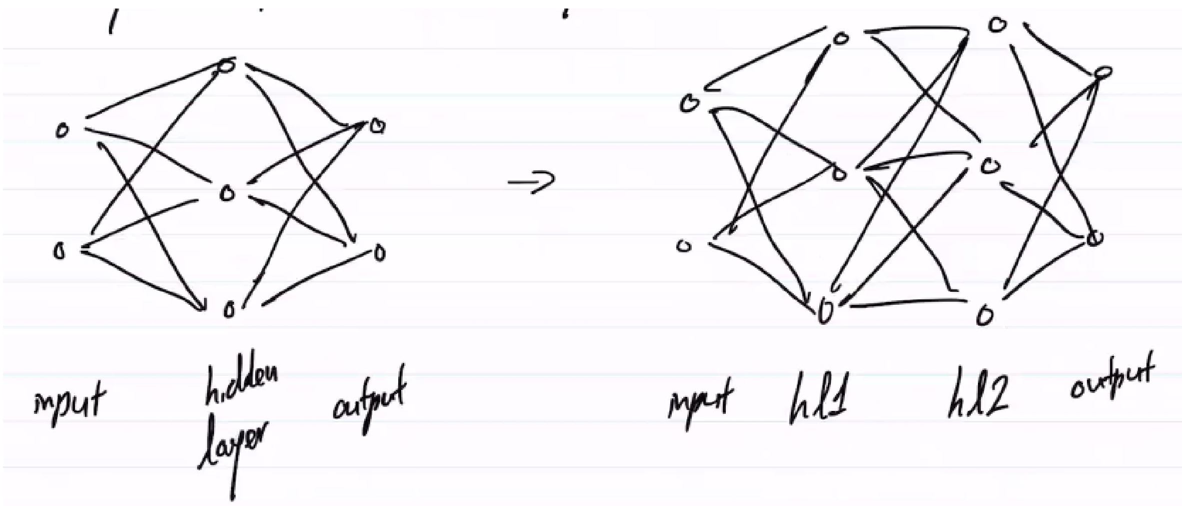
```
import mygrad as mg

# Pseudocode of mg.softmax_crossentropy()
def softmax_crossentropy(scores, labels):
  # scores: (M, D_out)
  # labels: (M)

  # run scores through mg.softmax() to get an array of the same shape
  # transform labels to one-hot encoding (p_true)

  # computes cross-entropy loss between truths, predictions
```

Multilayer Perceptrons / Multilayer NN



- Connections to the same node from different nodes is equal to the inputting the summation of the results from the different (prior) nodes
    - One neuron ~ $\phi(x_i \cdot w_i + b_i)$

  $L_0 = [\phi(x_0 w_0 + b_0), \phi(x_1 w_1 + b1)...\phi(x_{N-1} w_{N-1} + b_{N-1})] = \phi[XW_0 + b_0]$

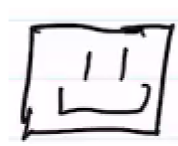
    - Previous layer's output are now used as input

$$L_1 = \phi[L_0 W_1 + b_1]$$

$$y = \sum_{i=0}^{N-1} v_i \phi(l_{0,i} w_i + b$$

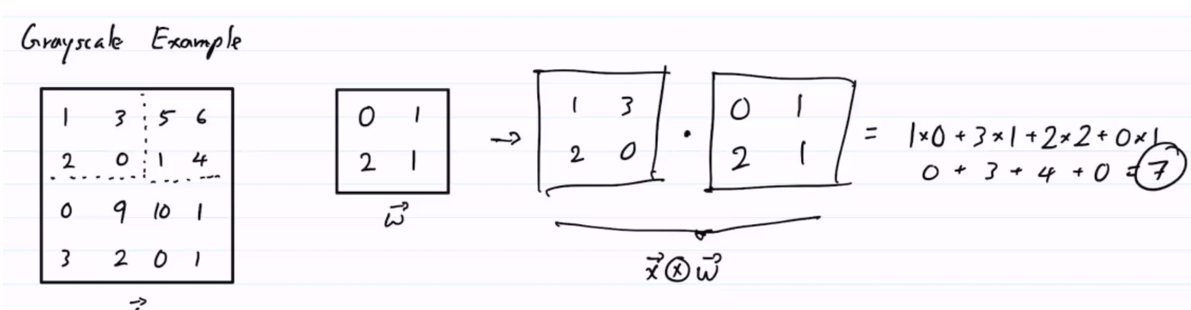
- Where L refers to a whole layer, whereas l refers to the individual outputs of a previous layer's nodes

Convolutional Layers and Images

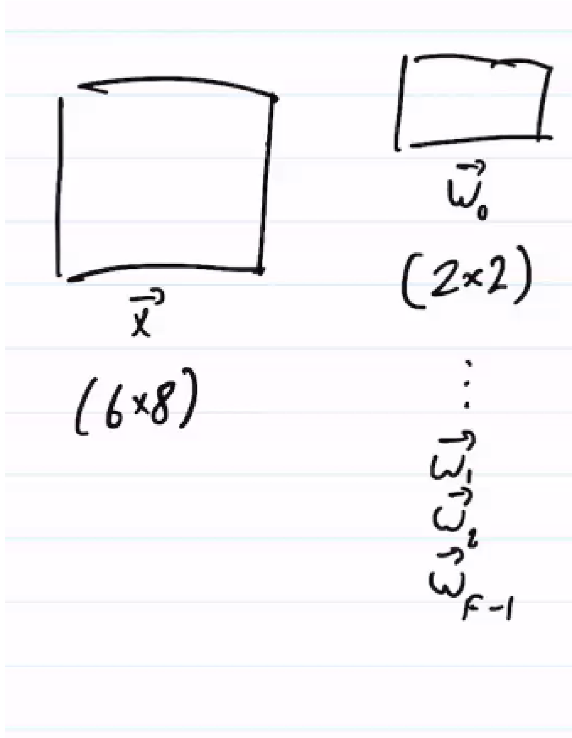


How can we handle images?

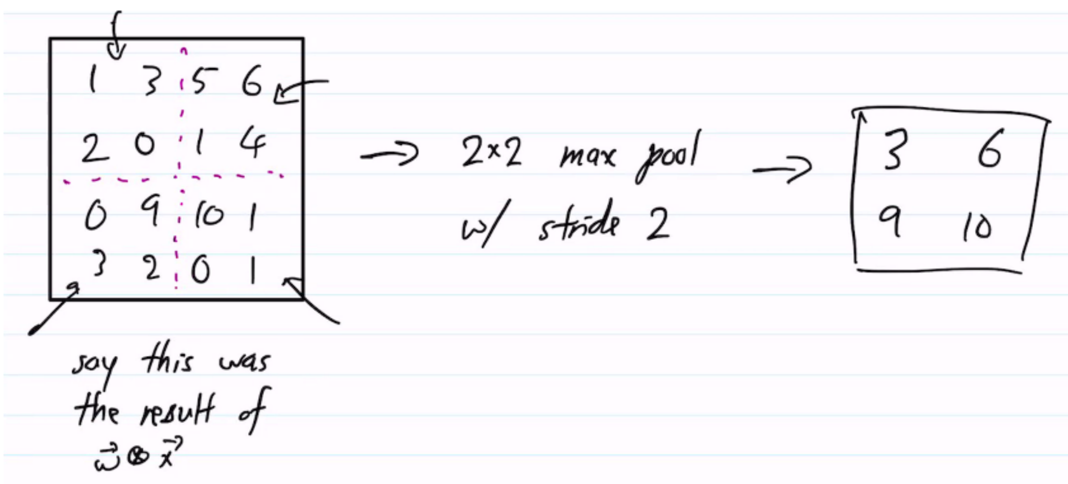
- It is possible to transform an array (representing a matrix of pixels) into a single vector by flattening it and then feeding it to a regular multi-layer neural network (with an output of 10 nodes is using softmax).
  - What are the problems with using images as vectors?
    - The image loses positional information because the main features of the original image are now spread out and disconnected
    - Different neurons have to deal with the same features, depending on where they are in the image
      - Redundant effort
- **Convolution**
  - Filters (sub-matrices with their own weights and biases) “slide” along the image, “looking” at it. *Recall that the dot product of two matrices is equivalent to their similarity. A filter’s output from a portion of the image is equivalent to how much the filter “recognizes” that sub-section*.
  - Q: Would a filter activate for a rotation of what it is designed to detect?
    - No, it would not activate optimally. translational invariance ≠ rotational invariance



  - Having an image with an image with size (6, 8) and filter size (2, 2) along with stride-1, would yield a new array of size `(F, 5, 7)` where is `F` is the number of filters



- **Max Pool**



- Operation that often follows one or more convolutional layers
  - Purely a discretization/downsampling process—no learnable parameters
  - $f(\vec{x}) = max(\vec{x})$
- Why would this be useful?

    - high values come from high similarity, therefore max pool helps later parts of the network “focus on” high-similarity detections and reject low-similarity and redundant detections