SHALLOW NEURAL NETWORKS

decision boundaries often hyperplanes

Matrix Multiplication = distortion of space

Jupyter demo

$C = AB$ → Multiply, every entry takes this form

$$C_{ij} = \sum_k a_{i,k} b_{k,j}$$ Have 4 indexes

contract inner indexes and keep outer ones during matrix multiplication

$$y_{il} = \sum_{jk} x_{ijkl} a_{jk}$$

collapse jk — multiply by jk entry for each A

can multiply any dimensionality of tensors as long as we specify what we want to collapse to

Simplified Notation

$y_{il} = x_{ijk} a_{jk}$ } Drop summation symbol

`np.einsum("ij, j → i", A, v)`

↕ Returns same answer

A: matrix, v: vector

`A.dot(v)`

$C_{kl} = \sum_{ij} b_{ijk} a_{il} v_j$ ↕ Code equivalent

`np.einsum("ijk, il, j → kl", B, A, v)`

More general than matrix multiplication

`np.einsum(B, [0, 1, 2], A, [0, 3], V, [1], [2, 3])`

(i, j, k; same as other i; new index; j ↕)

Output indexes to keep

can write notation in numbers get the same values

4th power - 4 tensors multiplied together

np.linalg.matrix_power(A, 4)

```
np.einsum(A,[0,1],A[1,2],
          A,[2,3],A[3,4],
          [0,4]
          )
```

$= \begin{bmatrix} 199 & 290 \\ 435 & 634 \end{bmatrix}$

each item is A since we're raising it to the 4th power

- Collapse middle, only keep first + last index

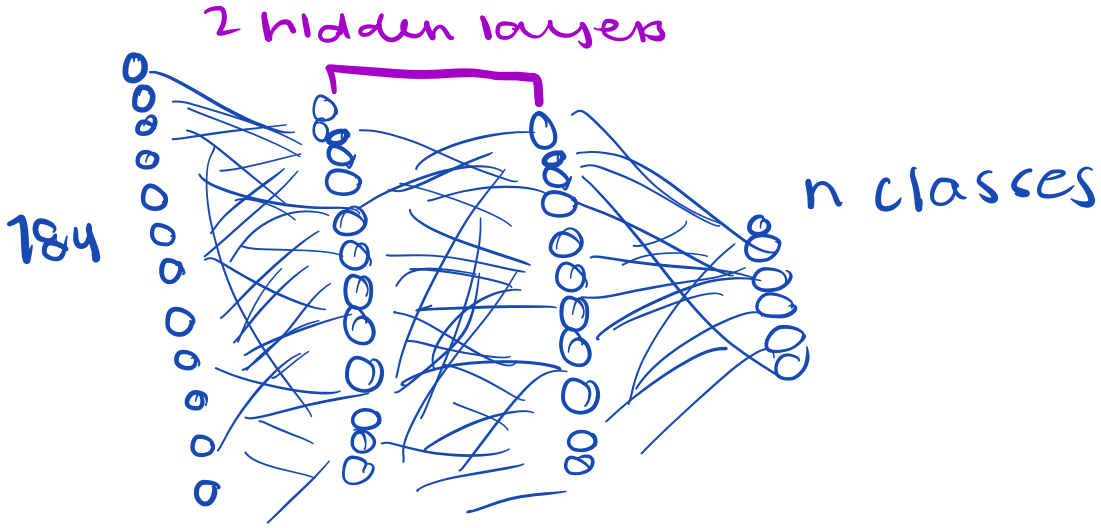
einstein summation very commonly used in deep learning

Dot product is a special case of the einstein summation

SHALLOW NEURAL NETWORKS

Optimize cost function, which adds up loss for every example
- optimize w/ gradient descent

shallow = only 1 hidden layer



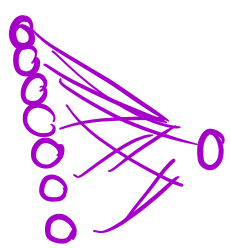
Parse image to vector of #s
feed numbers in to neurons

logistic:

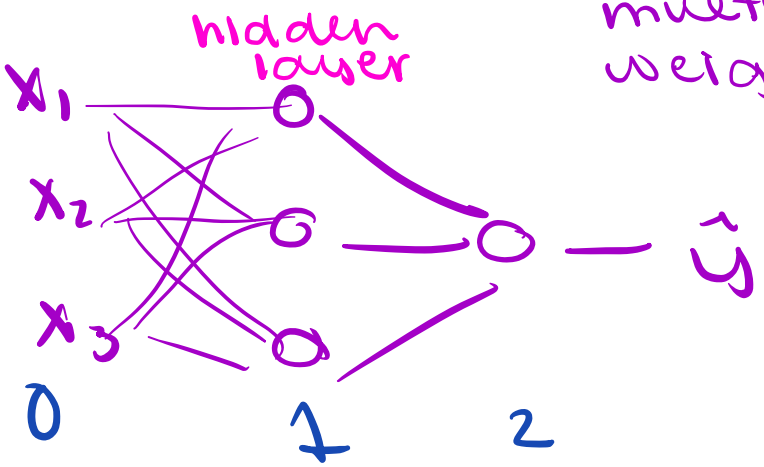
$$\hat{y} = \sigma(w^T x + b)$$

$$= \sigma\left(\sum_{j=1}^{n_x} w_j x_j + b\right)$$

inputs all feed to 1 output neuron



Neural Net



now have multiple sets of weights + biases

$w^{[1]}$ $w^{[2]}$

$b^{[1]}$ $b^{[2]}$

$z^1 = w^1 x + b^1$

↳ $a^1 = \sigma(z^1)$

$z^2 = w^2 x + b^2$

$a^2 = \sigma(z^2)$

↓

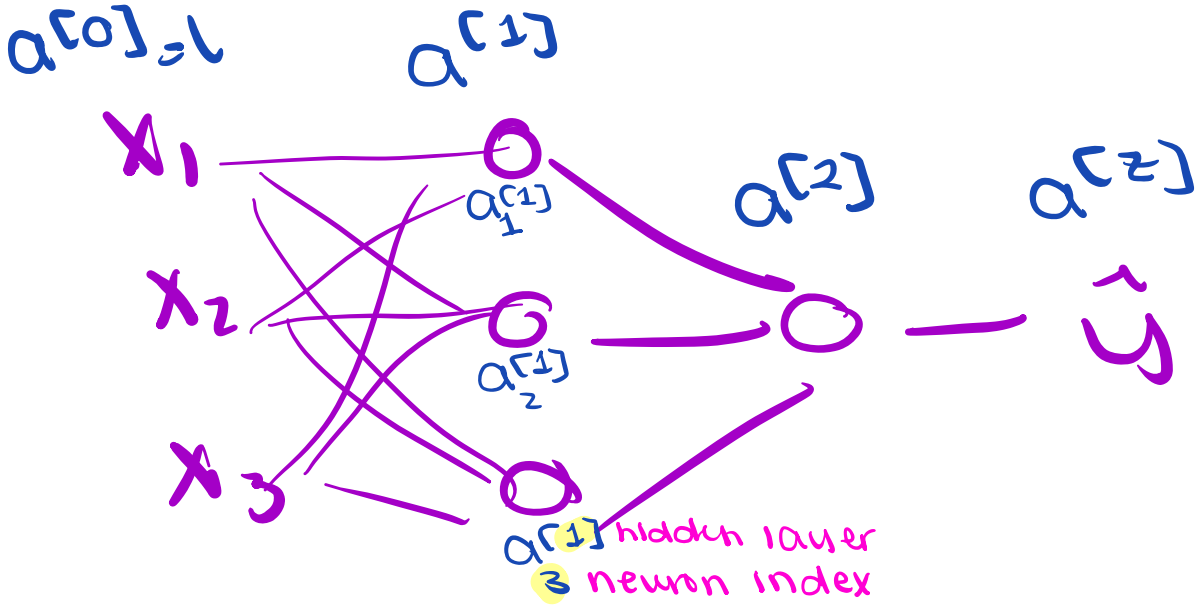
$L(a^2, y)$

Calculate loss function

we don't count inputs in the # of layers

→ 2 layers, hidden + output

activation - how activated is it? 0 or 1



Logistic Regression: $\hat{y} = a$

Shallow network: $\hat{y} = a^{[2]}$

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{bmatrix}$$

Organize them as a column vector

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$$

$$a_1^{[1]} = \sigma(z_1^{[1]})$$

$a_i^{[\ell]}$ → layer

i → neuron in layer

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}$$

$$a_2^{[1]} = \sigma\left(z_2^{[1]}\right)$$

Same operation, just different weights + biases

- we do this computation for every neuron
- vectorize operation when we do it in code

$$z^{[1]} = \begin{bmatrix} - & w_1^{[1]T} & - \\ - & w_2^{[1]T} & - \\ - & w_3^{[1]T} & - \\ & \vdots & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ \vdots \end{bmatrix}$$

Organize weights by row

$$= \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \end{bmatrix}$$

$$= \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{bmatrix}$$

Just organizing computation in matrix form

Apply calculations to every layer

dim weight matrix

(#outputs, #inputs)

for each layer