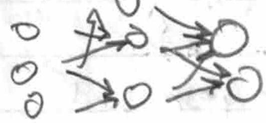
ADVANCED ALGORITHMS

← Neural networks
← Decision trees
← Practical advice

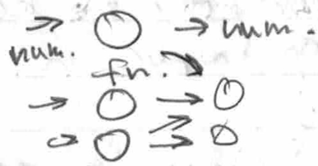
NEURAL NETWORKS

Original idea: mimic how the brain works. Still used as a metaphor even if the implementation has evolved. Now we call it Deep Learning but it's just NN. Applications include image + speech recognition, NLP, etc.

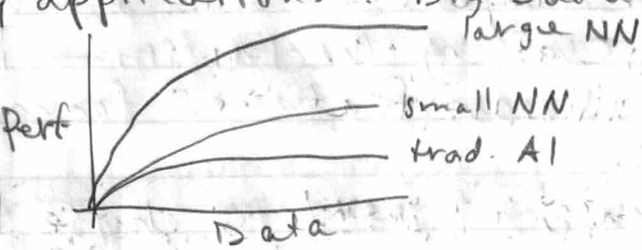
Brain: neurons in a network



Artificial NN: simplified math model of neuron

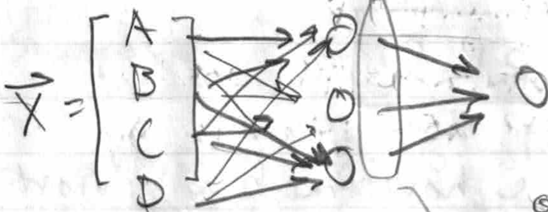


Why has this taken off only recently for many applications? Big Data.



Example: Demand Prediction (product sales)
- sell t-shirts
- for each: top seller? yes/no
- logit now used as activation fn. for one "neuron"/node that computes probability. need to wire them together.
- We have several features: price, material, marketing, ship cost. Use neurons to combine them, then combine those outputs

features layer 1 layer 2



= prob. of class 1 = $\vec{a}$

output layer

input layer | hidden layer (can have any #)

$\vec{y}$ = vector of layer outputs = "activations"
each = activation value

each node in layer n-1 connected to each in layer n

Each hidden layer is doing a boring fn fit (e.g. logit) but it's then fed into another layer. This is a kind of auto-feature crossing mechanism — it learns new nonlinear features from the original features.
→ You don't need to choose the features!

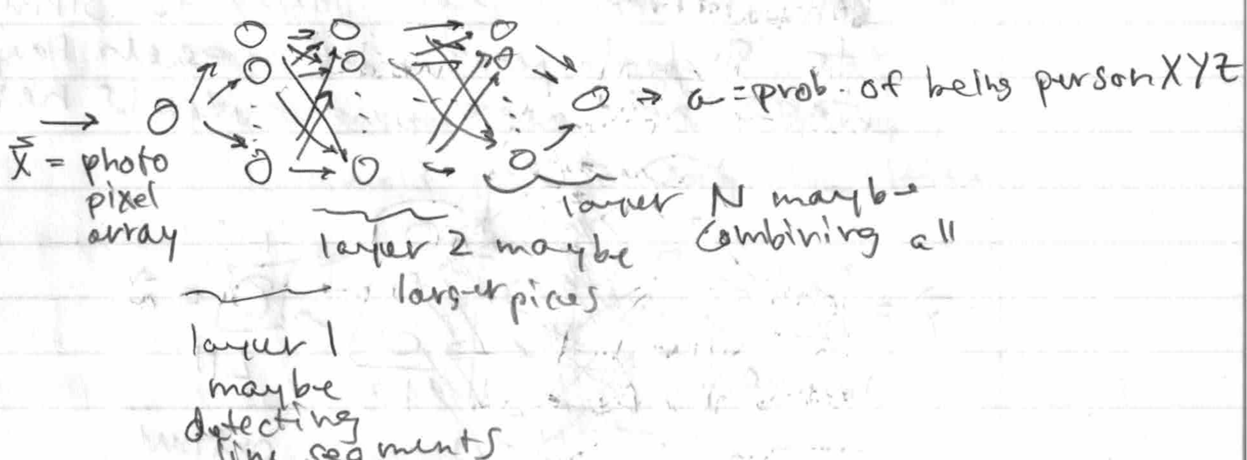
Parameters: (determines architecture)
— # of hidden layers
— hidden layer size (# nodes/neurons)

NN aka "multi-layer perceptron"

(Computer Vision)

Example: Image Recognition

- Input: pixel array as ft. vec.
- Output: identify of person in picture



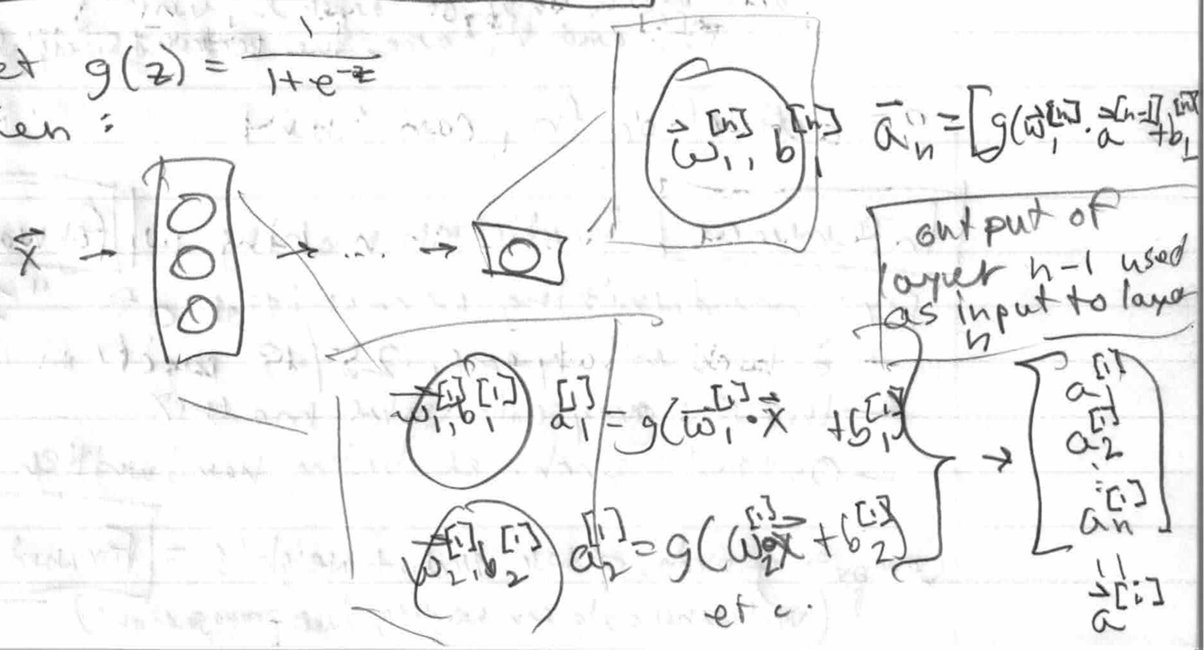
Key Intuition:

★ The hidden layers "learn"/"recognize" different levels of pattern and the layer size allows depth of recognition at that level.

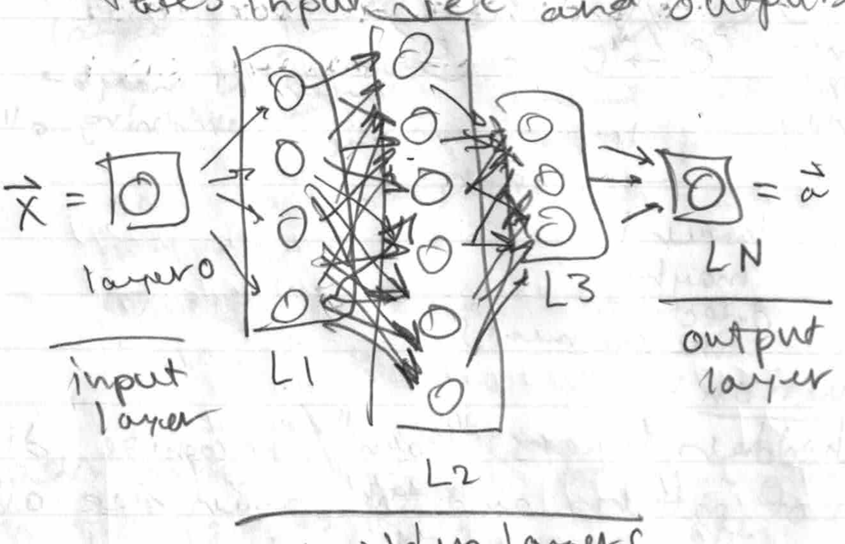
$$\boxed{\text{NEURAL NETWORK MODEL}}$$

Let $g(z) = \frac{1}{1+e^{-z}}$

Then:



More complex networks
- Can have any # of layers w/ any # of neurons
- Computation at each layer is similar to single-layer model: each layer takes input vec and outputs next vec



$$\boxed{a_j^{[l]} = g(\vec{w}_j^{[l]} \cdot \vec{a}^{[l-1]} + b_j^{[l]})}$$

activation value of layer l, unit j

$\vec{w}_j^{[l]}$ and $b_j^{[l]}$ are the learned weights

g = activation fn, can vary

Inference: Prediction making w/ forward prop.

Ex: handwriting recognition of "1"
- 2 hidden layers, 25/15 units
- Input: grayscale value matrix
- Output: prob. of being handwritten 1

Computing prob. given arch. + weights = Forward Propagation
(vs. learning alg for weights, "back propagation")

* You can reframe parametric models (e.g. lin. reg., log. reg.) in neural network architecture:
  - single layer
  - single unit
  - no activation fn. (or "linear") for lin. reg., sigmoid for log. reg. classifier

TENSORFLOW w/ keras

- Create "Dense" layers that take # of units, activation fn, and can take set_weights
- You can just call each layer w/ prev. activation vector to get output (do inference)
- "Sequential" can group the layers for you
- Uses NumPy for representation, but used by TF slightly differently / inconsistently
  - ↳ TF needs matrices (mxn) for vectors instead of 1d vec.
    i.e. np.array([[1,2,3]]) instead of np.array([1,2,3])
  - ↳ Can get an np.array matrix from tf.Tensor w/ tensor.numpy()
  - ↳ Can pass np.array to TF but it will convert internally — but converting output is manual
- So
  - prep input
  - build Dense layers, pass to Sequential
    ↳ "model" variable
  - model.compile + model.fit (more later)
  - model.predict(input)

AGI

- Compare ANI (narrow) that solves one kind of problem/domain
  - → Massive progress
- w/ AGI (general), similar to humans
  - → Unclear if <u>any</u> progress

Challenges
- Activation functions are so simple compared w/ bio neurons
- Also very insufficient understanding of brain!

"One Learning Algorithm" hypothesis
- Auditory cortex learns to see when it gets visual data instead of aural
- Same w/ somatosensory cortex — learns to see
- Can learn to "see" w/ tongue for blind
- Sonar (clicks) can "do sight"
- etc.

→ Can we replicate this algorithm ??

VECTORIZATION

- Lots of matrix multiplications in NNs
- GPUs and some CPUs can do this very efficiently w/ vectorization / parallelism
- np.matmul instead of per-neuron computation in loop → vectorized!

$A^T = 1 \times n$ feature vec
$W = n \times j$ weights matrix for j units
$B = 1 \times j$ bias vector for j units

↳ $Z = A_i^T W + B$

↳ $A_{i+1} = g(Z)$ activation for layer i