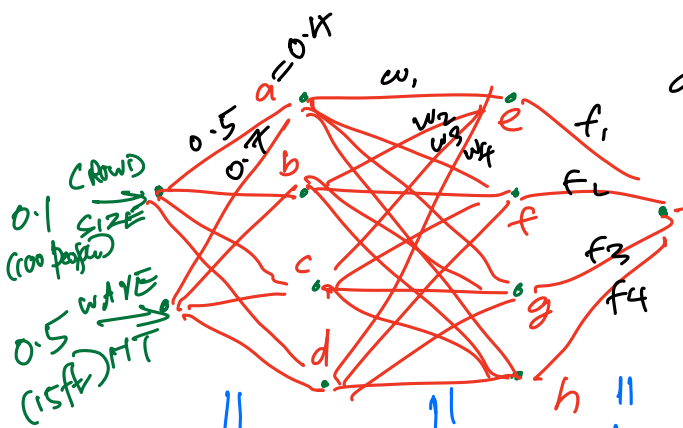
INTRO TO NNs

decide whether to go to a Beach

inputs: crowd size (10-1000) SCALE (0-1)
e.g. 500 people 0.5
wave size (1-30ft) e.g. 10ft 0.3333



$a = (0.1 \times 0.5) + (0.5 \times 0.7)$
$= 0.05 + 0.35 = 0.4$

go-or-no-go
(>0.5 = go
≤0.5 = no-go)

$e = (a \times w_1) + (b \times w_2) + (c \times w_3) + (d \times w_4)$

$\text{output} = (e \times f_1) + (f \times f_2) + (g \times f_3) + (h \times f_4)$

√ random initial values

weights typically 2 or 4 byte floats

-1 to +1
-2 to +2
-3 to +3

e.g. 0.239872346
-2.49817893251 67

# of connections = 28 = # of weights

these 28 weights are called the "parameters" of this NN

So, a NN is comprised of:

- its parameters (in a text file)
- the runnable executable to do a prediction flow using the parameters

Building a NN

- decide on architecture (# layers, # nodes)
- start with random parameters (weights) W_1
- TRAIN the model with training data set — THE MODEL "LEARNS" ITS WEIGHTS — to modify the weights (W_{FINAL}) — VERY COMPUTE HEAVY
- TEST model for reliability, quality
- deploy !!

Use the model (NN)

- give it input (unseen data) — also called getting a "prediction"
- let NN compute the results using the deployed W_{FINAL} and generate output — COMPUTE HEAVY
- continuously monitor model's quality of results with user feedback

OPEN AI (Large Lang Models)

- GPT 3.5 – 175B params
- GPT 4 – 1.7T params

Some Chinese LLMs – 4 or 5 T params
Google's Gemini Pro 1.5 – ~ 2T params

- NN architecture (layers, nodes, conns)
- weights being "learned" via training data
- deploy & predict

175 B weights → GPT 3.5 (OpenAI)

175B × 2 bytes = 350 B bytes
= 350 GB RAM

∴ the weight file ≈ 350 GB bytes
may not be possible in laptop or ordinary servers

LLama 2 ⇒ 70B params
⇒ 31B
⇒ 7B → 140 GB RAM on cloud

7B × 2 bytes = 14 GB RAM

Gemini Pro 1.5 = 2T Params

4000 GB RAM

MISTRAL 8×7B