superdatascience.com/deep-learning

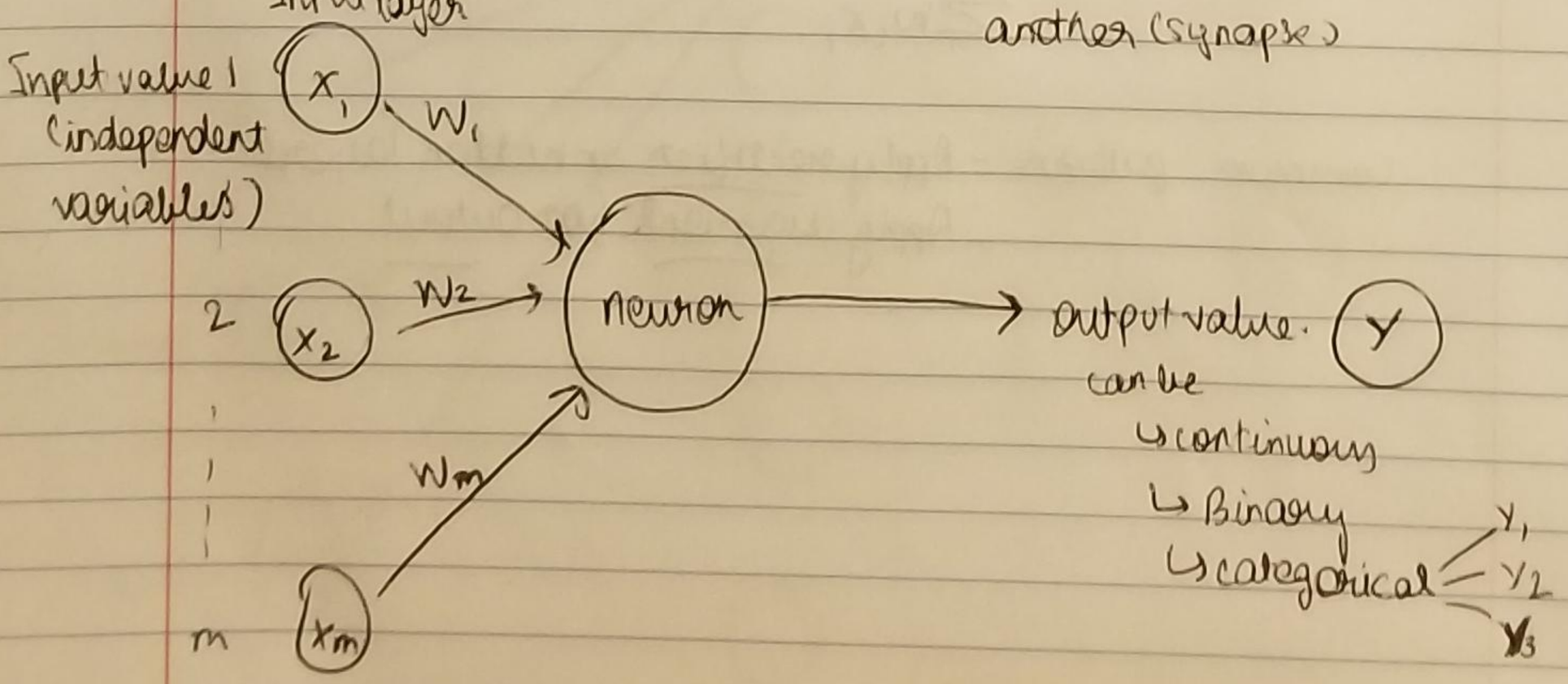
Deep learning A-Z - Hands-on - ANN

Section: ANN intuition

① Plan of attack.
- ↳ Neuron - replicate
- ↳ Activation function
- ↳ How do neural networks work? (example)
- ↳ How do neural networks learn.
- ↳ Gradient descent.
- ↳ Stochastic Gradient Descent
- ↳ Backpropagation.

② The neuron.

↳ Basic building block of neural network.
↳ mimic human learning mechanism.
↳ Basic structure of neuron - axon, dendrites, ~~then~~ connected to one another (synapse)



- Need to standardize input normalize them.

- Each input is given a weight for every case (observation)
- weights are adjusted in the process of learning.

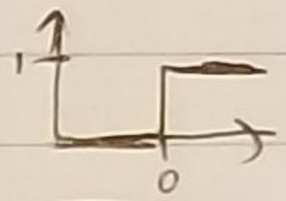
- What happens in the neuron?

1st step $\sum_{i}^{m} w_i x_i$

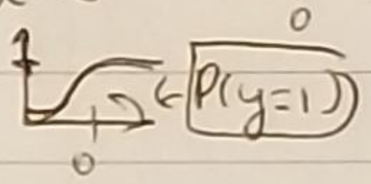
2nd Apply activation function $\phi(\sum_{i}^{m} w_i x_i)$

3rd Pass on signal in next neuron.

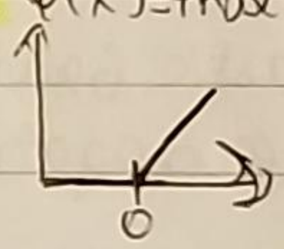
The activation function
① Threshold function $\phi(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$



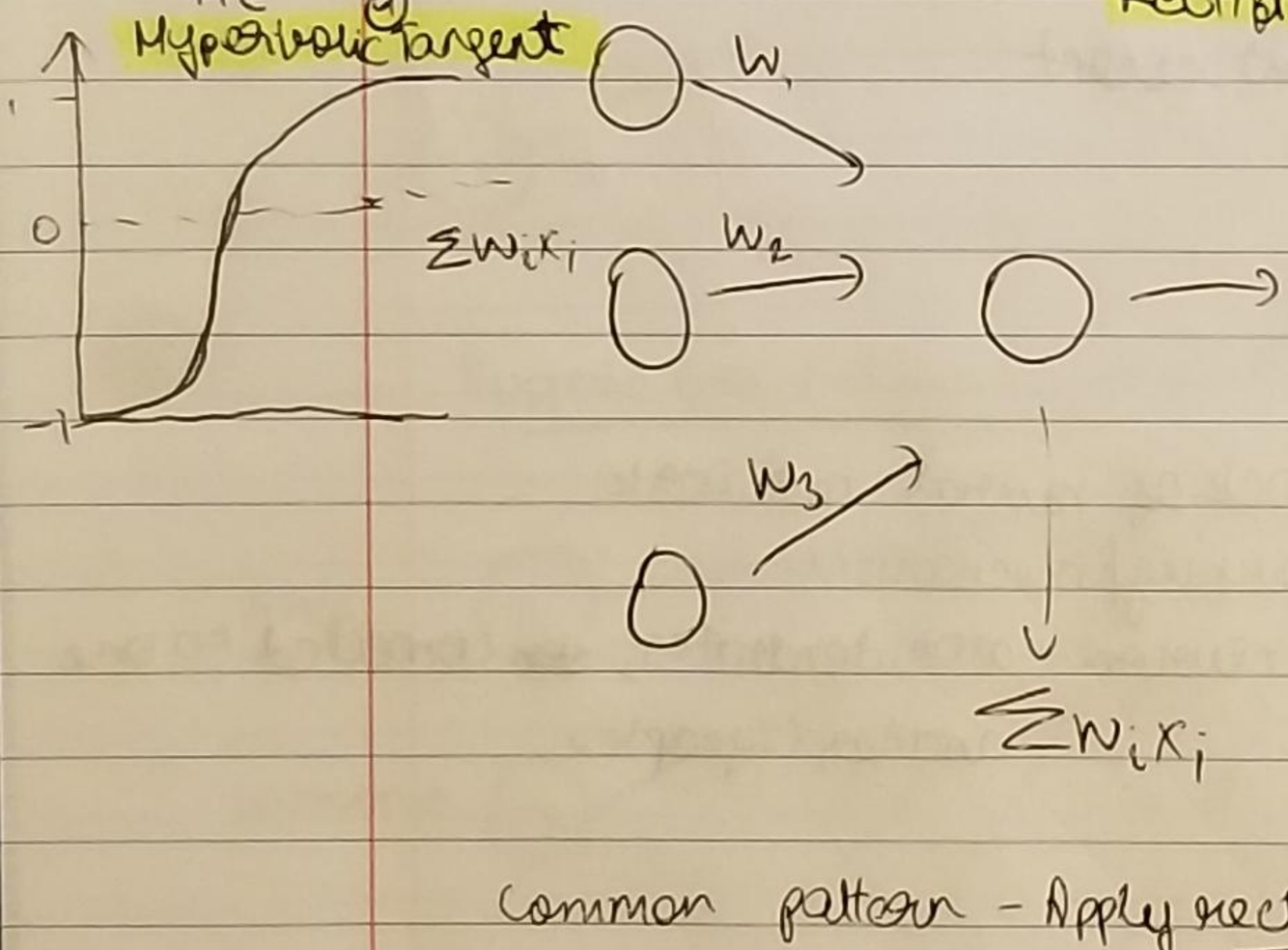
② Sigmoid function $\phi(x) = \frac{1}{1+e^{-x}}$



③ Rectifier $\phi(x) = \max(x, 0)$



④ Hyperbolic tangent $\phi(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$

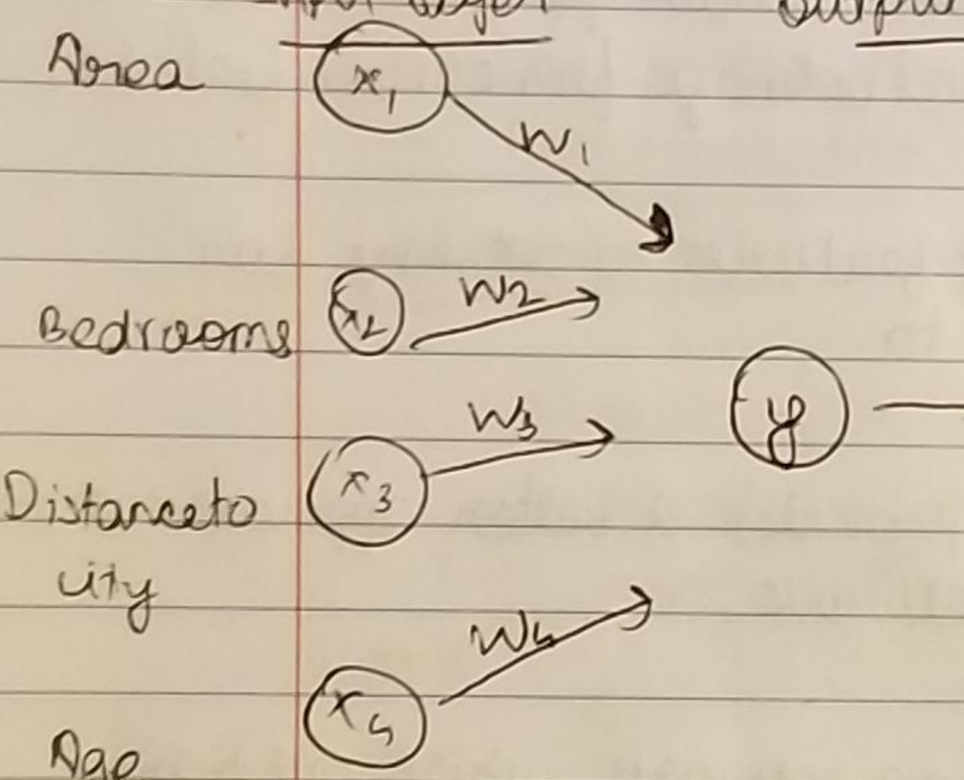


Common pattern - Apply rectifier in hidden layers
Apply sigmoid for output

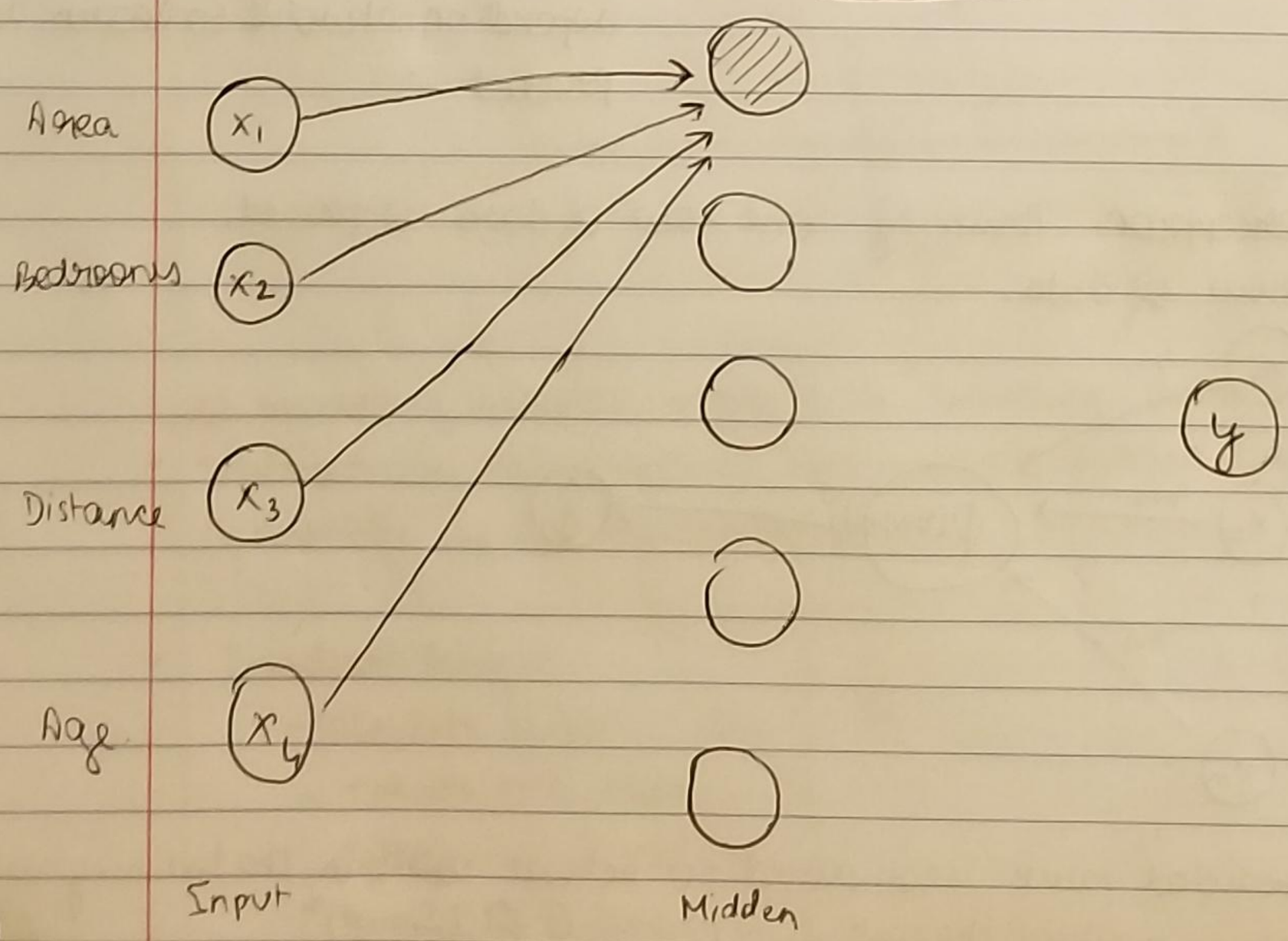
(3) How do neural networks work? Not discuss the training of network. For this lecture the network is trained.

example

→ evaluate property prices.



Suppose use hidden layers - where the real impact lies.



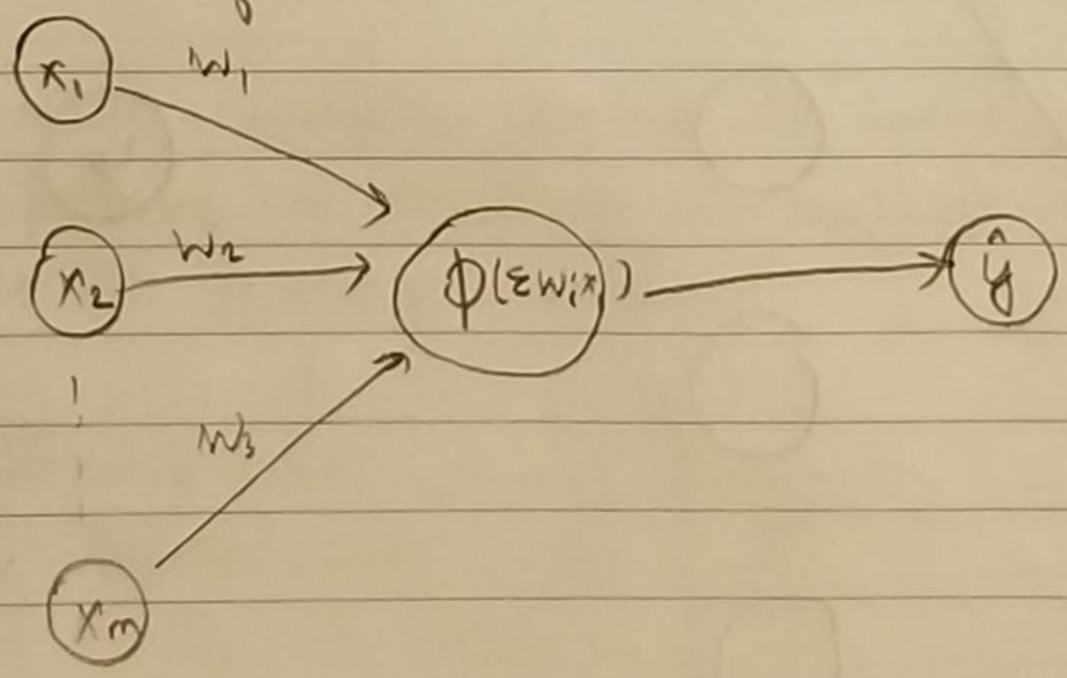
- Suppose the model is already trained.
- Suppose the neuron highlighted only focusses on certain features like - Area and Distance in this case.
  ↳ Neuron is focussing on ~~prope~~ certain properties and activate for certain ~~pro~~ features. and optimize for certain features.

each
Neuron takes in combinations of features to optimize for and apply its activation ~~to~~ function to

→ considering different features provides a better approximation.
→ increase flexibility of neural network

(4) How do Neural Networks learn? → a network which self learns rather than hardcoding
→ code the architecture but depend on network to learn and predict.

eg perceptron. Assuming one row of data is passed. for one row of data.



→ predicted value is compared to actual value in the training value by computing the cost function $C = \frac{1}{2}(\hat{y} - y)^2$

→ The goal is to min C. Feedback is provided to network and weights are adjusted to lower cost function and wrong

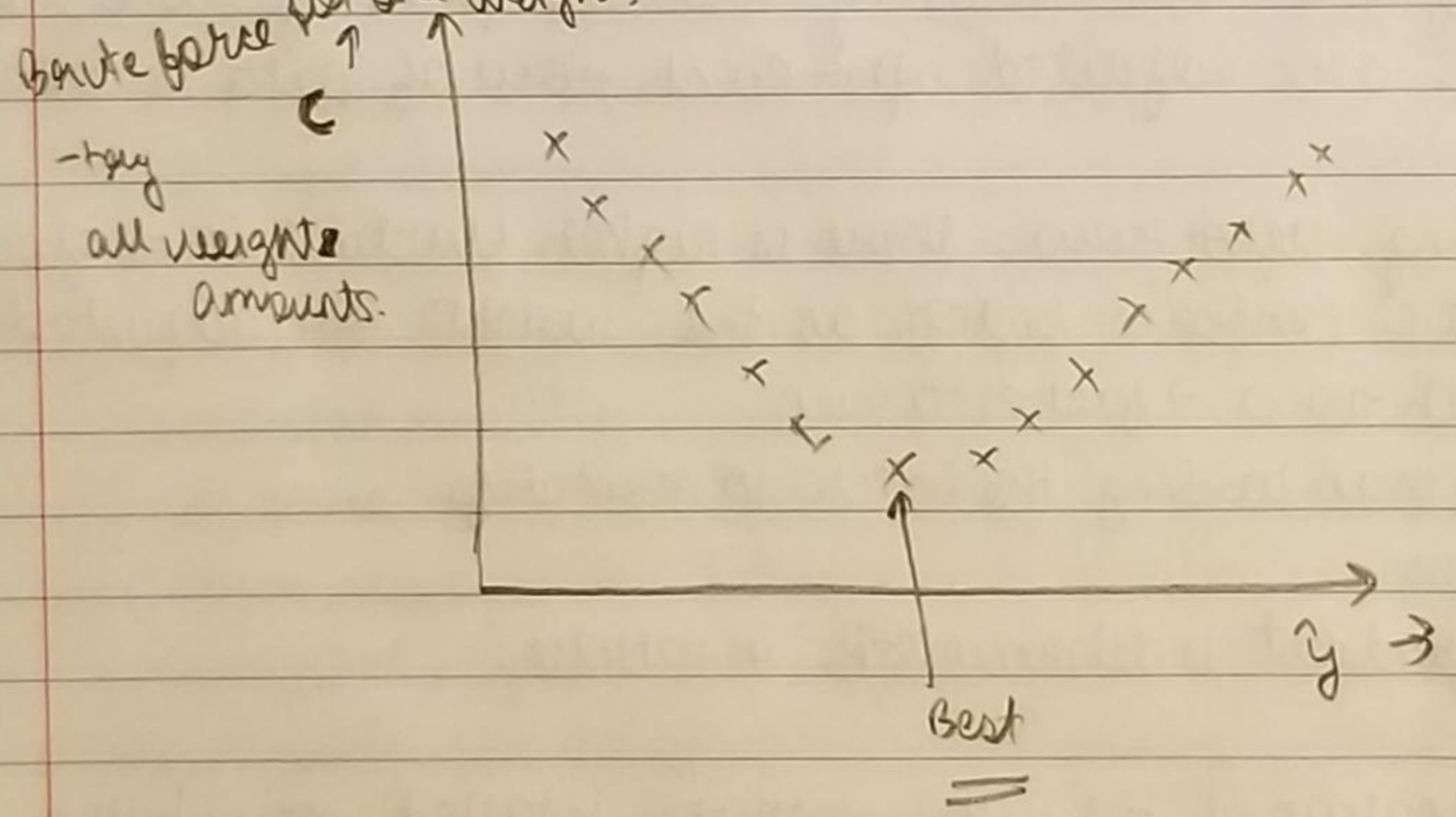
$\hat{y}$ as close to y. Process is repeated multiple times.

For ~~one~~ ~~exp~~ epoch = trained on all rows once.
For multiple rows, same process, weights are readjusted and rows share the same weights. until $\min \sum \frac{1}{2}(\hat{y} - y)^2$
↳ "Backpropagation" - process of constant feedback and readjusting weights to minimize cost function

⑤ **Gradient Descent**

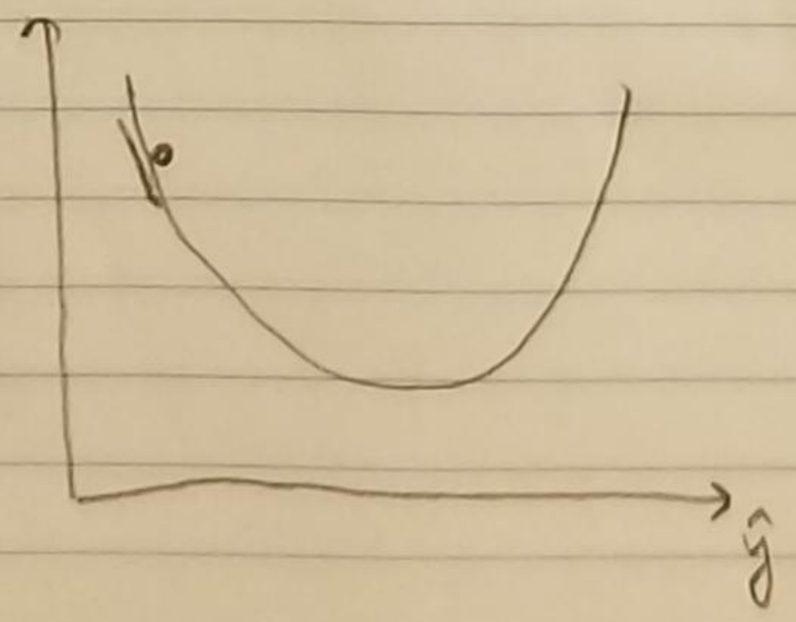
↳ explain how weights are readjusted. And in turn how the cost function is reduced.

Brute force per one weight.



- As number of weights increase ⇒ "curse of dimensionality"
- Brute forcing not practical for multiple hidden layers as combination of number of weights increase.

- Gradient Descent.
  - Calculate Slope
  - -ve or +ve Slope determines direction to move towards. at each step.



(6) Stochastic Gradient descent.

⇒ Gradient descent required the cost function to be convex. (one local minimum)

⇒ The risk of applying gradient descent on a non convex is we might ~~Stochastic Gra~~ find a local min and not a global one.

⇒ This is where stochastic gradient descent comes in.

- Unlike gradient descent or "Batch" gradient descent where weights are adjusted after training with all rows of data, in stochastic weights are adjusted after each row of data is tested.

- By testing after row, there is higher fluctuation ⇒ global min
- Stochastic perform faster as the weights are adjusted for each row → faster to run. rather than loading the batch in memory.

- Batch gradient is deterministic in nature.

- A combination of the two methods is called mini batch gradient descent. where instead of a single row or the batch, groups of rows are considered.

⑦ Backpropagation.

↳ allows us to adjust weights at the same time.

Input layer → Hidden layers → output (forward propagation)
← ← (backpropagation)

↳ which weights contribute to the errors.

Summarizing the steps: (with stochastic)

① Randomly initialise the weights to small numbers close to 0 (but not 0)
② Input the first observation of your dataset in the input layer, each feature in one input node.
③ Forward propagation: from left to right the neurons are activated in a way that the impact of each neuron's activation is limited by the weights. Propagate activations until getting the predicted result y.
④ Compare the predicted to actual. Measure generated error.
⑤ Back propagation from right to left, the error is back propagated. Update the weights according to how much they are responsible for the error. The learning rate decides by how much the weights are updated
⑥ Repeat ① to ⑤ for each observation (REINFORCEMENT LEARNING) or for each batch (BATCH LEARNING)
⑦ When the whole set passes through ANN => one epoch
⑧ Redo more epochs.