Backpropagation

- A technique used to compute gradients efficiently using chain rule of partial derivatives
- Calculated by moving backwards in the network.

Gradient Descent "Apply the changes to descent"

- Updates weights & bias using gradients computed during backpropagation

Learning?

- Finding the optimal value for weights & biases such that cost is minimum.
- Finds the derivative of cost fn (gradient) & move in that dirn.

weights & bias → [cost fn] → cost : Since input to cost fn is weights & bias,

Gradient descent → performed for each weights & bias per node, then → ∇C will also have size of weights & bias.

$$\nabla C = \begin{bmatrix} 0.1 \\ 3.2 \\ 4.5 \\ \vdots \end{bmatrix}$$

: magnitude of gradient indicates which weight is more important in network

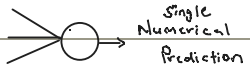
3.2 is 32 times more than 0.1 ↑ Effect more on cost fn.

Cost Function

- Average loss for all samples
- Gradient descent performed on weights & biases of each node

Understanding learning intuitively.

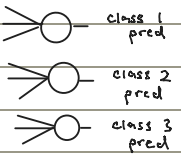
Numerical prediction



Suppose predict = 12000 ← Activation from last layer.
Actual = 10 000

Goal = -2000

Categorical prediction



Activation = $\sigma(W \cdot A + B)$

$-2000 = \sigma(w_1 a_1 + \dots w_n a_n + B)$

Things Adjustable:

1) Activation fn (Not very effective)
2) Alter weights$_i$ (Where gradient magnitudes are higher) ← most impt update
3) Adjust a_i — +ve values reduces, -ve values increases ↑ (No direct influence, dependent on previous layer weights & bias)
4) decrease B_i

Gradient vector

* Repeat same steps for all nodes in previous layer till input layer.

Repeat same step for all examples.

Take the average change of all examples.

	Sample 1	Sample 2	Sample 3 ...
W_0	-0.8	+0.2	-0.4
W_1	+0.7	+0.3	+0.1
W_2	-0.1	-0.4	+0.5
⋮			

$$-\nabla C = \begin{bmatrix} \frac{-0.1}{3} \\ \frac{0.11}{3} \\ 0 \\ \vdots \end{bmatrix}$$

↙ changes required for all weight & bias of each node for current iteration

Stochastic Gradient Descent

1) Randomly shuffle training data
2) Split data into mini batches
3) compute gradient descent for each batch
   ↳ Not the Actual gradient, thus not the most efficient but a good approximation
   ↳ faster computation