Why is the gradient the direction of steepest ascent?

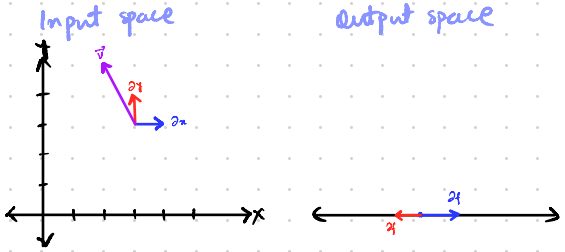
$f(x, y) = x^2 y$

Primer on partial derivatives

The question that directional derivative asks is:

what does this nudge in the direction of the vector do to the function itself?

$$\vec{v} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$



Partial derivative: How does change in x or y only) changes the output?

Formally, it gives the rate of change of function as you move in the direction of that vector

So what we are asking is, given an arbitrary vector $\vec{a} = [a_1 \; a_2 \; a_3]^T$

we want to find what is

$$\lim_{h \to 0} \frac{f(\vec{a} + h\vec{v}) - f(\vec{a})}{h|\vec{v}|} := \nabla_{\vec{v}} f(\vec{a}) \quad \text{(Definition)}$$

The definition of $\partial f / \partial x$ is

$$\lim_{h \to 0} \frac{f(\vec{a} + h\hat{i}) - f(\vec{a})}{h} := \frac{\partial f(\vec{a})}{\partial x}$$

So if we have a vector $\vec{v} = [p \; q]^T$, $\nabla_{\vec{v}} f(\vec{a})$ would be equal to (rate of change in x direction) $\times p$ + (rate of change in y dir.) $\times q$.

↳ This can be intuitively understood by looking at the graph

Therefore, $\nabla_{\vec{v}} f(\vec{a})$

$$= p \cdot \frac{\partial f(\vec{a})}{\partial x} + q \frac{\partial f(\vec{a})}{\partial y} = \vec{v} \cdot \vec{\nabla} f(\vec{a})$$

we define $\vec{\nabla} = [\partial/\partial x \;\; \partial/\partial y]^T$ = Gradient operator

We want to answer the following question: In which direction does the function f increase the fastest?

In short, we want to maximize $\nabla_{\vec{v}} f(\vec{a}) = \vec{v} \cdot \vec{\nabla} f(\vec{a})$ for some $\vec{v}$.

Clearly, $\vec{v} \cdot \vec{\nabla} f(\vec{a})$ is maximum when $\vec{v}$ is in the same direction as $\vec{\nabla} f(\vec{a})$

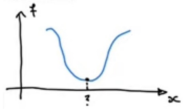
$\therefore \vec{\nabla} f(\vec{a})$ is the direction of steepest ascent.

Vector and Matrix derivatives

Instead of tediously doing partial derivative w.r.t. each variable, why not group all the variable into a vector and define a vector derivative, that when acted upon gives $\vec{0}$ (=vector) when it is either at a local maxima or minima?

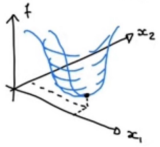
Main idea

How do we find minimum of a scalar function?



Set $\frac{df}{dx} = 0$.

And for a function of two variables?



Set $\frac{\partial f}{\partial x_1} = 0$

Set $\frac{\partial f}{\partial x_2} = 0$

$\frac{\partial f}{\partial \vec{x}} = ?$

- What if we have a function with N variables?
- Functions with intermediate variables?
- Functions producing a vector as output instead of a scalar?

Main idea:
Define vector and matrix derivatives to allow us to differentiate directly in vector/matrix form.

Definitions

- Derivative of a scalar function $f : \mathbb{R}^N \to \mathbb{R}$ with respect to vector $\mathbf{x} \in \mathbb{R}^N$:

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \triangleq \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_N} \end{bmatrix}$$

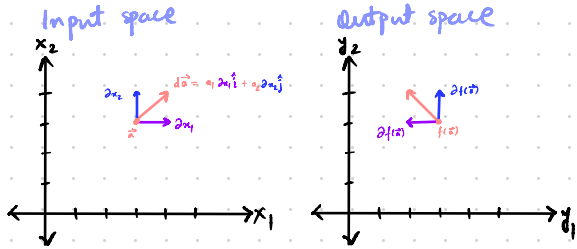
- Derivative of a vector function $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^M$ with respect to vector $\mathbf{x} \in \mathbb{R}^N$:

$$\frac{\partial \boldsymbol{f}(\mathbf{x})}{\partial \mathbf{x}} \triangleq \begin{bmatrix} \frac{\partial \boldsymbol{f}(\mathbf{x})}{\partial x_1} \\ \frac{\partial \boldsymbol{f}(\mathbf{x})}{\partial x_2} \\ \vdots \\ \frac{\partial \boldsymbol{f}(\mathbf{x})}{\partial x_N} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_2(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_M(\mathbf{x})}{\partial x_1} \\ \frac{\partial f_1(\mathbf{x})}{\partial x_2} & \frac{\partial f_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_M(\mathbf{x})}{\partial x_2} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_1(\mathbf{x})}{\partial x_N} & \frac{\partial f_2(\mathbf{x})}{\partial x_N} & \cdots & \frac{\partial f_M(\mathbf{x})}{\partial x_N} \end{bmatrix}$$

$f: \mathbb{R}^2 \to \mathbb{R}^2$

$f(x_1, x_2) = (y_1, y_2)$

$\frac{\partial f(x_1, x_2)}{\partial x_1} =$



Jensen's Inequality

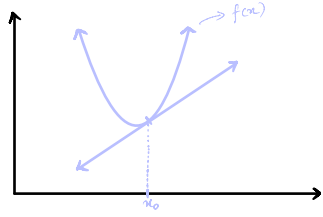
Definition: A random variable is called integrable if expected value exists and is well-defined. i.e.

$$E(X) < \infty$$

$$\Rightarrow \int_X x f_X(x)\,dx < \infty \quad \text{if } X \text{ is continuous, } f_X(x) = \text{pdf}$$

$$\Rightarrow \sum_{x \in X} x\, p_X(x) < \infty \quad \text{for discrete RV, } p_X(x) = \text{pmf.}$$

Definition: A function f is convex <u>iff</u> all the points lie above the tangent drawn at arbitrary point x_0.



Jensen's Inequality: For a convex function $f: \mathbb{R} \to \mathbb{R}$, and an integrable random variable X,

$$E(f(X)) \geq f(E(X))$$

Proof: Let the slope of the tangent at any arbitrary point of the function x_0, be m.

The equation for the line will be: $y = m(x - x_0) + f(x_0)$

clearly, $f(x) \geq m(x - x_0) + f(x_0) \quad \forall\ x_0, x \in \mathbb{R}$

We replace $x \to X$ & $x_0 \to E(X)$ and get

$$f(X) \geq m(X - E(X)) + f(E(X))$$

Taking Expectation on both side and using linearity of Expectation, we get

$$E(f(X)) \geq m(E(X) - E(X)) + E(f(E(X)))$$
$$= 0 + f(E(X))$$

$\therefore$ we have,

$$E(f(X)) \geq f(E(X))$$

Q.E.D.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} = U$$

$$U^{-1} = U^{\dagger}$$

$$U^{-1} = U^{\dagger} = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$$

$$\frac{\pi}{4} + \cos\left(\frac{\pi}{4}\right) + \cos\left(\frac{\pi}{4} + \cos\left(\frac{\pi}{4}\right)\right) + \cos\left(\frac{\pi}{4} + \cos\left(\frac{\pi}{4}\right) + \cos\left(\frac{\pi}{4} + \cos\left(\frac{\pi}{4}\right)\right)\right)$$

$$= \frac{\pi}{2}$$

$$\begin{pmatrix}1\\0\end{pmatrix}(0\ 1) = \begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix}$$

$$\begin{pmatrix}0\\1\end{pmatrix}(1\ 0) = \begin{pmatrix}0 & 0\\ 1 & 0\end{pmatrix}$$

$$= \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$$

$$\begin{pmatrix}x_1\\ x_2\end{pmatrix}, \begin{pmatrix}y_1\\ y_2\end{pmatrix} \rightarrow x_1 y_1 - (x_1 y_2 - x_2 y_1) + 2 x_2 y_2$$

$$\begin{pmatrix}y_1\\ y_2\end{pmatrix}, \begin{pmatrix}x_1\\ x_2\end{pmatrix} \rightarrow y_1 x_1 - (y_1 x_2 - y_2 x_1) + 2 y_2 x_2$$

$$A = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$$

let $f(x) = \sin(x)$

$$x + f'(x) + f'(x + f'(x)) + f'(x + f'(x) + f'(x + f'(x)))$$

$$\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix} = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$$

$$A^2 = 1$$

$$f\left(\frac{w_1 x_1 + w_2 x_2}{w_1 + w_2}\right) \le \frac{w_1 f(x_1) + w_2 f(x_2)}{w_1 + w_2}$$

$$\langle (\psi_1 \vec{b}_1 + \psi_2 \vec{b}_2 + \cdots \psi_n \vec{b}_n), (\lambda_1 \vec{b}_1 + \lambda_2 \vec{b}_2 + \cdots + \lambda_n \vec{b}_n) \rangle$$

$$b_{ij} = \langle b_i, b_j \rangle$$

$$\sum_{i=1}^{n} \psi_i \langle \vec{b}_i, \lambda_1 \vec{b}_1 + \cdots + \lambda_n \vec{b}_n \rangle$$

$$\sum_{i=1}^{n} \psi_i \langle \vec{b}_i, \sum_{j=1}^{n} \lambda_j \vec{b}_j \rangle$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \psi_i \lambda_j \langle \vec{b}_i, \vec{b}_j \rangle$$

$$[\psi_1\ \psi_2 \cdots \psi_n] \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ \vdots & & & \\ b_{n1} & \cdots & & b_{nn} \end{bmatrix}_{n \times n} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{bmatrix}_{n \times 1}$$

$$[\psi_1 \cdots \psi_n] \begin{bmatrix} \sum_{j=1}^{n} b_{1j} \lambda_j \\ \sum_{j=1}^{n} b_{nj} \lambda_j \end{bmatrix} = \psi_1 \cdot \sum b_{1j} \lambda_j + \cdots \psi_n \sum b_{nj} \lambda_j = \sum_{i=1}^{n} \sum_{j=1}^{n} \overbrace{\psi_i b_{ij} \lambda_j}^{\psi_i \langle b_i, b_j \rangle \lambda_j}$$

$A^2 = I$

$A^4 = I$

$e^{tA} = I + At + \frac{A^2t^2}{2!} + \frac{A^3t^3}{3!} + \dots$

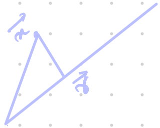
$= I + At + \frac{t^2}{2!}I + \frac{At^3}{3} + \dots$

$= I(1 + \frac{t^2}{2!} + \dots)t + At(t + \frac{t^3}{3} + \dots)$

$1 - \frac{t^2}{2!} + \frac{t^4}{4!} - \frac{t^6}{6!}$

$\cos(t)$ $\quad I\cos(it) + At$

$I t \cos(it) + A t \sin(it)$



$|\vec{x}||\vec{r}|\cos\theta$

$|\vec{x}|\sin\theta$

$\vec{x}\cdot\vec{r} = xr\cos\theta$

$\sin\theta = \sqrt{1 - \left(\frac{\vec{x}\cdot\vec{r}}{xr}\right)^2}$

$\sqrt{x^2 - \left(\frac{\vec{x}\cdot\vec{r}}{r}\right)}$

$\begin{bmatrix}0&1\\0&0\end{bmatrix}\begin{bmatrix}0&1\\0&0\end{bmatrix}$ $\quad A_1 \begin{bmatrix}0&0\\1&0\end{bmatrix}\begin{pmatrix}x\\y\end{pmatrix} \to \begin{pmatrix}0\\x\end{pmatrix}$

$\begin{bmatrix}0&0\\0&0\end{bmatrix} = A_1^2 \quad A_1^n = 0 \quad n \geq 2$

A_1

$\begin{bmatrix}0&1\\0&0\end{bmatrix}\begin{pmatrix}x\\y\end{pmatrix} \to \begin{pmatrix}y\\0\end{pmatrix} \to 0$

$e^{A_1t} = I + A_1t + 0$

$= |I + A_1t|$

$e^{A_1t}e^{A_2t} = (I + A_1t)(I + A_2t)$

$= I + A_2t + A_1t + At^2$

$= I + It + At^2$

$\nabla \to$ operator

$\nabla : \mathbb{R} \to \mathbb{R}^d$

$f(x,y) = \overset{\mathbb{R}}{(x^2+y^2)}$

$\nabla := \begin{pmatrix}\frac{\partial}{\partial x_1}\\ \vdots \\ \frac{\partial}{\partial x_n}\end{pmatrix}$

$f(x,y) = 2x^2 + 3y^3$

$\nabla f = \begin{pmatrix}\frac{\partial f}{\partial x}\\ \frac{\partial f}{\partial y}\end{pmatrix}$ → column → Vector ↳ Row vector

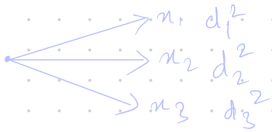
$\nabla \to$

$\frac{\partial \vec{f}(\vec{x})}{\partial \vec{x}}$ = Jacobian

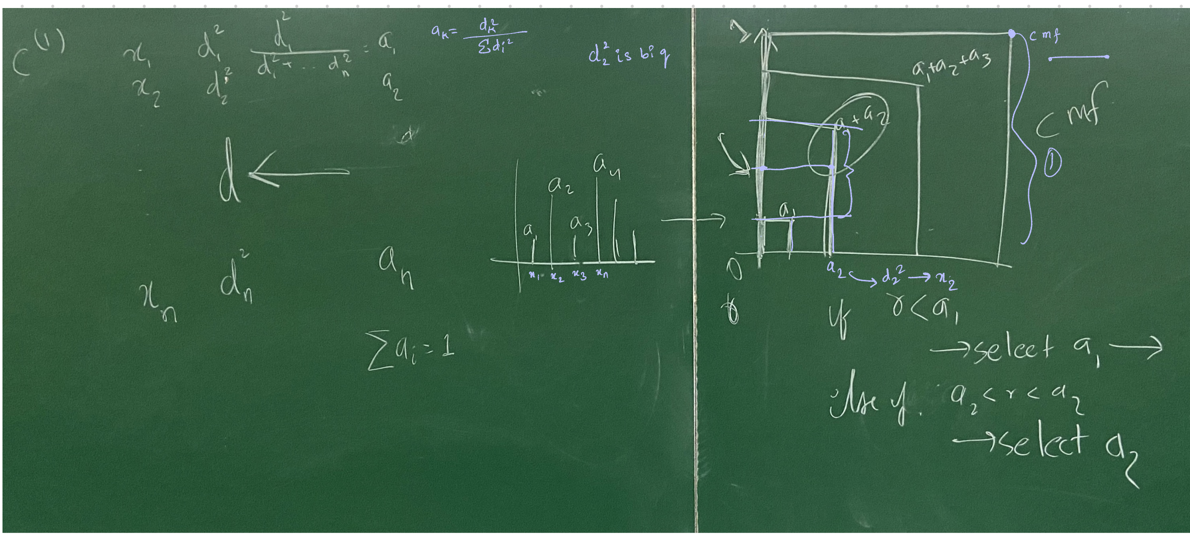
$\nabla f(x,y) \to$ Vector → Gradient

∇ direction of steepest

F	S	Sun	Monday	Tuesday	Wednesday	Thursday	Friday
15th	16th	17th	18th	19th	20th	21st	22nd
K	N	N	N	N	S	S	N
Y	S	N	N	N	S	S	N
S	N	N	S	S	N	N	–

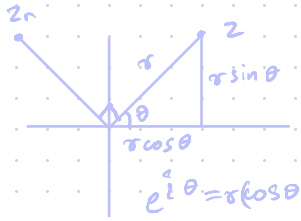


$\{d_n^2\}$

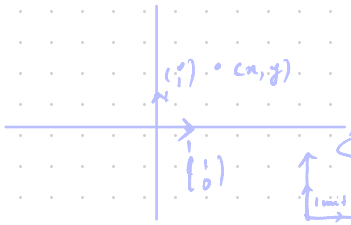


$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

matrices as transformation/function



$e^{i\theta} = r(\cos\theta + i\sin\theta)$



$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$

270°

$i = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$ $z \cdot i \, i$

$j = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$e^{\pi i/2} = \cos(\pi/2) + i\sin(\pi/2)$

transformation $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

rotation $\begin{bmatrix} \quad & \quad \\ \quad & \quad \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} = 2 + 2i$

$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$

$z_n = z\, e^{\pi i/2}$

$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$

$z = r e^{i\alpha}$

$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$z_n = r e^{i\alpha} \times e^{\pi i/2}$

$= r e^{i(\alpha + \pi/2)}$

$\frac{1}{2}(|0\rangle + |1\rangle)(\langle 0| + \langle 1|)$

$\frac{1}{2}(|0\rangle\langle 0| + |0\rangle\langle 1|$

$\frac{1}{2}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $\begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix}$ $\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 4 \\ 3 \end{bmatrix} = \begin{bmatrix} -3 \\ 4 \end{bmatrix}$

$4 + 3i$

$-3 + 4i$

$|\psi\rangle = \underbrace{\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle}_{\text{superpos}}$ $+ |1\rangle\langle 0|$ $+ |1\rangle\langle 1|)$

$\rho = \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1|$

$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$

$\rho = \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1|$

Born

$I + |0\rangle\langle 1|$

$+ |1\rangle\langle 0|$

$\frac{1}{2} + \frac{1}{2}|0\rangle\langle 1|$ Bm

$\frac{1}{2}|1\rangle\langle 0|$

(det)

what is a jacobian

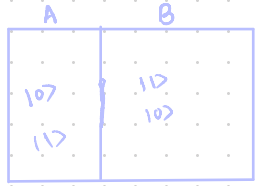
$f(x,y) = (x^2, y^2)$

$\rho =$ $\det(M)$ = how much does the matrix squish the space

in super classical / Noise / ensemble / static

$\frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1|$

$|\psi_{AB}\rangle = \frac{1}{2}|0\rangle + \frac{1}{2}|1\rangle$



det | Jacobian

$\frac{\partial f}{\partial}$ $dy \Big| dx$

$|\psi_A\rangle = |0\rangle$

$|\psi_B\rangle = |1\rangle$

$\int\int 1$ (Andy)

$|0\rangle\langle 0| = \rho \rightarrow$ pure state

$U = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

n →

$\begin{bmatrix} 1 \\ 0 \end{bmatrix}\begin{bmatrix} 1 & 0 \end{bmatrix}$

$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} \quad \end{bmatrix}$

$U^{-1} = U^{\dagger} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

$[\,] \rightarrow 1$

$|00\rangle \rightarrow (|++\rangle + |--\rangle/\sqrt{2})$

$|++\rangle \rightarrow$

$\sum_{i=1}^{n}\sum_{j=1}^{n}$

$\begin{vmatrix} 1-\lambda & 0 \\ 0 & -\lambda \end{vmatrix} = 0$

$\rho = \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1|$

$|0\rangle$ ↕ $|1\rangle$

ρ_0 ρ_1

1, 0 → degenerate

$\rho = |0\rangle\langle 0|$

A = spectral decomp

0 $\rho = 1|0\rangle\langle 0| + 0\times|1\rangle\langle 0|$

λ_1 λ_2

≥ 2

Born

$$U_{AB} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & -1 \end{bmatrix}$$

$$K_{ij} = \sqrt{\lambda_i}\, \mathbb{I}_S \otimes \langle j| U_{SE} |\mathbb{I}_S \otimes |i\rangle$$

$$K_{ij} = \sqrt{\lambda_i}$$

Back propogation

we have a cost function $C(X)$, where X = training data and X is $n \times d$ dimensional matrix containing n datapoints of dimension d.

In case where the dataset is MNIST, $d = 28 \times 28 = 784$ and $n = 60,000$

We define x^i as the i^{th} datapoint in the dataset. $\therefore X = [x^1\ x^2\ x^3 \dots x^n]^T$

and define $C(x^i)$ the cost per example/datapoint

which is $C(x^i) = \sum_{k=1}^{m} (y_k^i - \hat{y}_k^i)^2$, here $\hat{y}^i$ is the prediction (a vector of dimension m) of the neural network of the data point x^i and y^i is the actual label (also a vector of dimension m) that of the datapoint x^i. (meaning $\hat{y}^i = f(x^i)$)

In case of MNIST, $m = 10$ since there are 10 digits we want to predict.

Now, we define $C(X) = \frac{1}{n} \sum_{i=1}^{n} C(x^i)$ = average cost over all training dataset = Total cost of the network.

$\therefore C(X)$ = Total cost of the network $= \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{m} (y_k^i - \hat{y}_k^i)^2$

Here $\hat{y}^i = f_\theta(x^i)$, meaning f_θ is our neural network parameterized by θ that takes in input x^i gives $\hat{y}^i$ as output.

Here θ = weights and biases.

In case of 3B1B MNIST example, we have

i) 784 neurons as an input layer

ii) 2 hidden layers each consisting of 16 neurons

iii) Output layer consisting of 10 neuron

$\therefore$ Total number of weights = $\underbrace{16 \times 784}_{\text{(going from inputs to 1st hidden layer)}} + \underbrace{16 \times 16}_{\text{(1st to 2nd hidden layer)}} + \underbrace{16 \times 10}_{\text{(2nd to output)}} = 12,960$

and total bias terms = $\underbrace{16 + 16}_{\text{for hidden layer neurons}} + \underbrace{10}_{\text{for output layer neurons}} = 42$

$\therefore$ In total, there are $12,960 + 42 = 13,002$ weights and biases

θ is a collection of all these weights and biases

So we can write $\hat{y}^i$ as $\hat{y}^i = f(x^i; w_0, w_1, \ldots, w_{13,001})$

and the cost of the network as $C(X; w_0, w_1, w_2, \ldots w_{13,001})$

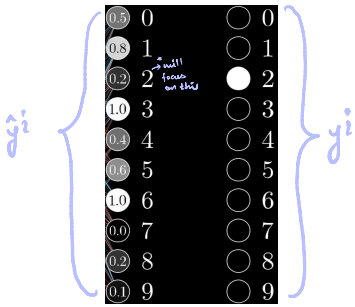
we want $-\nabla_w C(X; w_0, w_1, \ldots, w_{13,001})$ so that we can (iteratively) update the weights and biases and decrease the cost of the network.

Backpropagation is the algorithm for computing this crazy complicated gradient efficiently.

We will now be focusing our attention how the cost of the network is affected by a single arbitrary datapoint, say x^i $\longrightarrow$ an image of 2 from MNSIT

$\therefore$ $x^i \longrightarrow$  ,

For an untrained network,

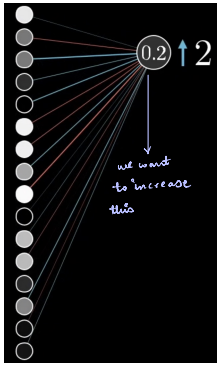


we wish to change the activations of the output layer such that it correctly classifies x^i as a 2.

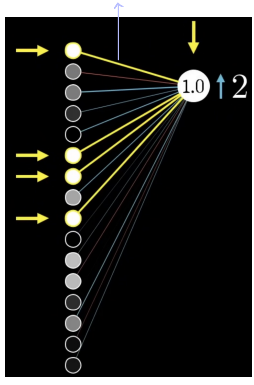
But we cannot directly change the activations, since we only have control over weights & biases but we will see it will be helpful to keep track of what changes we need to make

focusing on the neuron $\hat{y}_3^i$ as shown the the figure above* and below, we see

$$0.2 = \sigma(w_0 a_0 + w_1 a_1 + \cdots + w_{n-1} a_{n-1} + b)$$



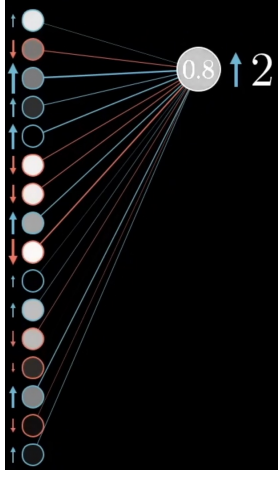
we can do that by increasing the weights in proportion to the activations of the previous layer i.e. $\Delta w_i \propto a_i$



or by changing the activations of the neurons connected to it in proportion to the weights i.e. neuron with high positive weights should be increased more while those with negative weights should be decreased. i.e. $\Delta a_i \propto w_i$

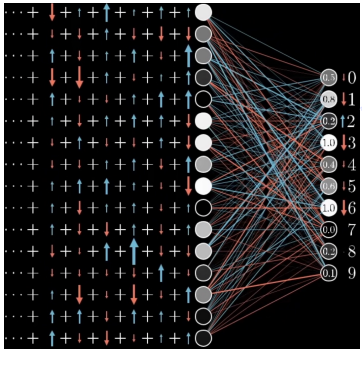
We obviously cannot change the activations directly, but (we will see) it helps keep track of this

Clearly, we can also change the biases to get the desired value of the output neurons

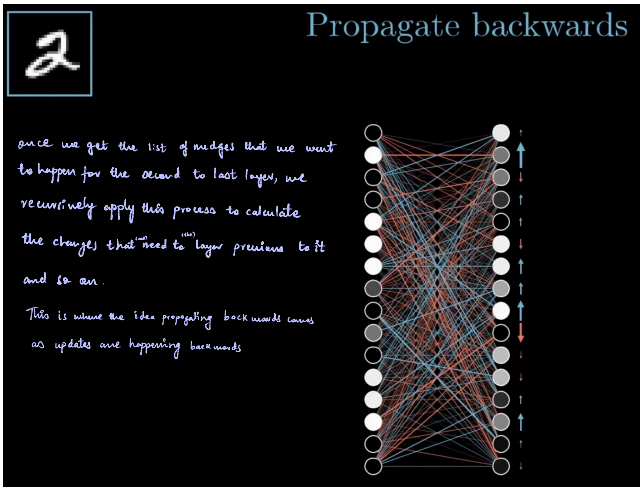


---

we remember that the changes that we wanted to happen to the W+B and activations as described above is only based on y_3^i, the 3rd neuron of our output layer, and therefore we add the relative changes based on other output neurons as well i.e. y_K^i, $1 \le K \le 10$, $K \ne 3$, in proportion to the corresponding weights and also in proportion to how much each of these neurons need to change.



After adding nudges from all the output neurons, the final (list of) nudges that we get for the second to last layer, represents the desire of y^i, and how the network should change so that it minimizes the cost function.



We remind ourselves that all of this is how a single training example y^i wishes nudge all the weights and biases. We need to consider how other training wish of nudging all the weights and biases.

	2	5	0	4	1	9	...	Average over all training data
w_0	−0.08	+0.02	−0.02	+0.11	−0.05	−0.14	...	→ −0.08
w_1	−0.11	+0.11	+0.07	+0.02	+0.09	+0.05	...	→ +0.12
w_2	−0.07	−0.04	−0.01	+0.02	+0.13	−0.15	...	→ −0.06
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮
$w_{13,001}$	+0.13	+0.08	−0.06	−0.09	−0.02	+0.04	...	→ +0.04

Averaging all the training example's nudges, we get a final list of averaged nudges over all training examples/dataset (end) is (proportional to) -ve gradient of the cost function $C(X;\theta)$

$$-\eta \nabla C(w_1, w_2, \ldots w_{13,001}) = \begin{bmatrix} -0.08 \\ +0.12 \\ -0.06 \\ \vdots \\ +0.04 \end{bmatrix}$$

we use this gradient descent step in our gradient descent algorithm.

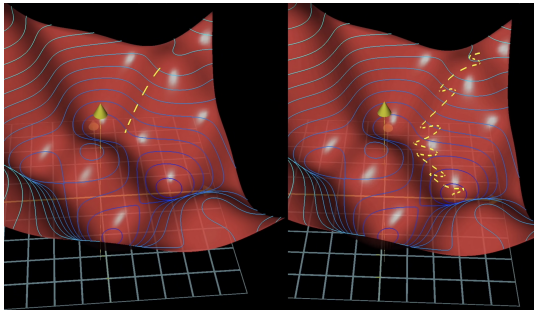
In practice, it takes computers an extremely long time to add up the influence of every single training example, every single gradient step. Therefore we will use an approximation called **stochastic gradient descent** which would speed up the process of our gradient descent algorithm.

(SDG) (This is mini-batch actually)

Stochastic Gradient Descent: We randomly shuffle our training data and divide it into "Mini batches", for example, each one having 100 training example, then we compute the gradient descent step according to the mini-batches using backpropogation. This would not be the true gradient descent step i.e. this wouldn't be the most efficient step downhill, but it each of the mini-batch does give a pretty good approximation and more importantly, it will gives us a significant computational speed-up

If we are to plot the trajectories of our network under the relevant cost surface

Rather than a carefully calculating man who takes slow, deliberate steps downhill



SDG would be like a drunk man stumbling aimlessly downhill but taking quick steps

---

Summary

Backpropogation is the algorithm for determining how **a single training example** would like to nudge the weights and biases.

A true gradient descent step would involve doing this for all tens of thousands of training examples and averaging the desired changes that you get. But since doing this computationally slow, we do **SDG** to speed up the process and converge to a **local minimum**

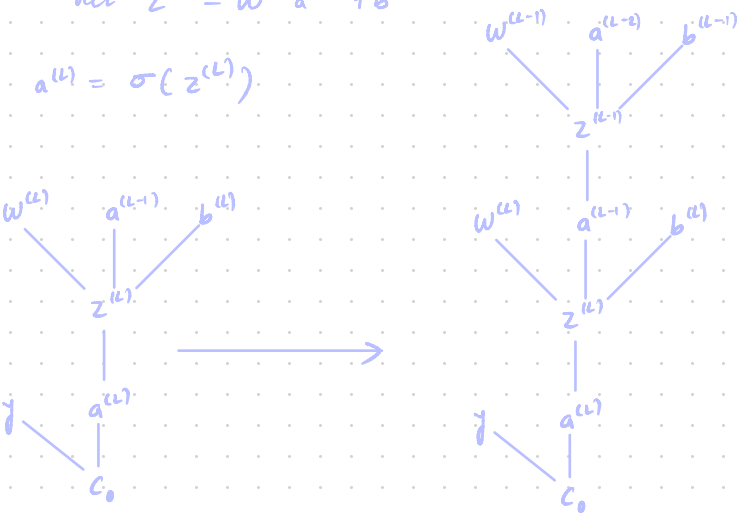
Backpropogation Calculus

$$C_0(\dots) = (a^{(L)} - y)^2$$

$$a^{(L)} = \sigma(w^{(L)} a^{(L-1)} + b^{(L)})$$

$$\text{let } z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$



$$A_{n \times n} = \sum_{i=1}^{n} \lambda_i |i\rangle\langle i|$$

$$f(A) = \sum_{i=1}^{n} f(\lambda_i) |i\rangle\langle i|$$

$Y = B + X * K$; Want $\frac{\partial E}{\partial K}$, Given $\frac{\partial E}{\partial Y}$

$$\frac{\partial E}{\partial K} = \frac{\partial Y}{\partial K}\frac{\partial E}{\partial Y} = \frac{\partial}{\partial K}(B + X*K)\frac{\partial E}{\partial Y}$$

$$= (0 + X*1)\frac{\partial E}{\partial Y} = X * \frac{\partial E}{\partial Y}$$

$$\frac{\partial E}{\partial K_{12}} = \frac{\partial Y_1}{\partial K_{12}}\cdot\frac{\partial E}{\partial Y_1} + \frac{\partial Y_2}{\partial K_{12}}\cdot\frac{\partial E}{\partial Y_2} + \cdots + \frac{\partial Y_d}{\partial K_{12}}\cdot\frac{\partial E}{\partial Y_d}$$

$$= X_2 * \frac{\partial E}{\partial Y_1} + 0$$

$$\therefore \frac{\partial E}{\partial K_{12}} = X_2 * \frac{\partial E}{\partial Y_1}$$

$$\therefore \boxed{\frac{\partial E}{\partial K_{ij}} = X_j * \frac{\partial E}{\partial Y_i}}$$

we have, $Y_i = B_i + \sum_{j=1}^{n} X_j * K_{ij}$

Given $\frac{\partial E}{\partial Y}$, what is $\frac{\partial E}{\partial B}$?

$$\frac{\partial E}{\partial B_2} = \frac{\partial Y_1}{\partial B_2}\frac{\partial E}{\partial Y_1} + \frac{\partial Y_2}{\partial B_2}\frac{\partial E}{\partial Y_2} + \cdots$$

$$= \frac{\partial E}{\partial Y_2}$$

$$\therefore \frac{\partial E}{\partial B_2} = \frac{\partial E}{\partial Y_2} \Rightarrow \frac{\partial E}{\partial B_i} = \frac{\partial E}{\partial Y_i}$$

$$\Rightarrow \boxed{\frac{\partial E}{\partial B} = \frac{\partial E}{\partial Y}}$$

wrong way of solving the question

$$\frac{\partial E}{\partial x_3} = \frac{\partial Y_1}{\partial x_3}\frac{\partial E}{\partial Y_1} + \frac{\partial Y_2}{\partial x_3}\frac{\partial E}{\partial Y_2} + \frac{\partial Y_3}{\partial x_3}\frac{\partial E}{\partial Y_3} + \cdots \frac{\partial Y_d}{\partial x_3}\frac{\partial E}{\partial Y_d}$$

$$= K_{13} * \frac{\partial E}{\partial Y_1} + K_{23} * \frac{\partial E}{\partial Y_2} + \cdots + K_{d3} * \frac{\partial E}{\partial Y_d}$$

$$\frac{\partial E}{\partial x_1} = K_{11} * \frac{\partial E}{\partial Y_1} + K_{21} * \frac{\partial E}{\partial Y_2} + \cdots + K_{d1}\frac{\partial E}{\partial Y_d}$$

Now, Given $\frac{\partial E}{\partial Y}$, calculate $\frac{\partial E}{\partial X}$

Given $Y = B + X * K$, $\frac{\partial E}{\partial x} = \frac{\partial E}{\partial Y} *_{full} K$

$\therefore$ Given $Y_i = B_i + \sum_{j=1}^{n} x_j * K_{ij}$, $\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial Y_1} *_{full} K_{1j} + \frac{\partial E}{\partial Y_2} *_{full} K_{2j} + \cdots + \frac{\partial E}{\partial Y_d} *_{full} K_{dj}$

$$\frac{\partial E}{\partial x_j} = \sum_{i=1}^{d} \frac{\partial E}{\partial Y_i} *_{full} K_{ij}$$

$$E = -\frac{1}{n}\sum_{i=1}^{n} \log\left(\hat{y}_i^{y_i^*} \cdot (1-\hat{y}_i)^{1-y_i^*}\right)$$

$$= \log\left(\prod_{i=1}^{n} \hat{y}_i^{y_i^*} \cdot (1-\hat{y}_i)^{1-y_i^*}\right)$$

$y_i^* = 1 \quad \hat{y}_i = 0.9$

$0.9 : 1$