

RICE CRUMB #1

- ... or “road to risotto”
- What is the asymptotic behaviour of the expected value of the output of MLE?

Let's assume $\underline{\theta}_0$ is the true value
Let N be the number of data points
Then

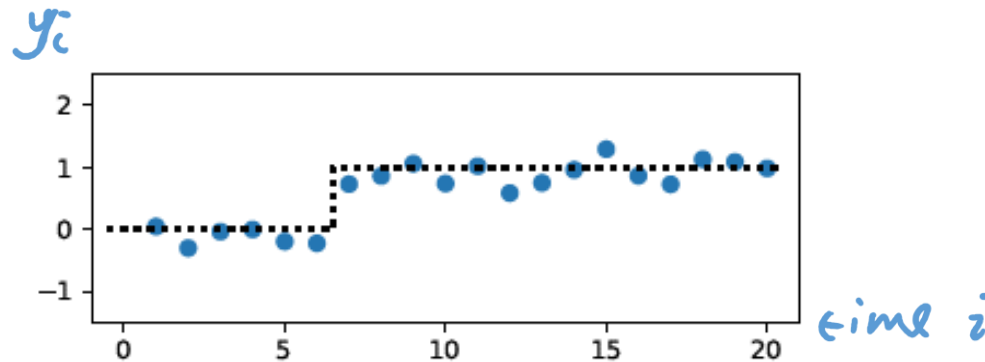
$$\lim_{N \rightarrow \infty} E[\hat{\underline{\theta}}_{ML}] = \underline{\theta}_0$$

(ASYMPTOTICALLY UNBIASED)

- (a) A 0/1 signal is being transmitted. The transmitted signal at timeslot $i \in \{1, \dots, n\}$ is $x_i \in \{0, 1\}$, and we have been told that this signal starts at 0 and then flips to 1, i.e. there is a parameter $\theta \in \{1, \dots, n-1\}$ such that $x_i = 1_{i>\theta}$. The value of this parameter is unknown. The channel is noisy, and the received signal in timeslot i is

$$Y_i \sim x_i + \text{Normal}(0, \varepsilon^2)$$

where ε is known.



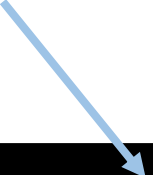
- (i) Given received signals (y_1, \dots, y_n) , find an expression for the log likelihood, $\log \Pr(y_1, \dots, y_n; \theta)$. Explain your working. [5 marks]
- (ii) Give pseudocode for finding the maximum likelihood estimator $\hat{\theta}$. [5 marks]

1. Skim read for keywords. What's the topic?
2. Look for question words. What is it asking you to do?
3. Think through the course. What sections are relevant?

(b) I have been monitoring average annual river levels for many years, and I have collected a dataset (z_0, \dots, z_n) where z_i is the level in year i since I started monitoring. I believe that for the first few years the level each year was roughly what it was the previous year, plus or minus some random variation; but that some year a drought started, and since then the level has decreased on average each year. I would like to estimate when the drought started. I do not know the other parameters.

(i) Propose a probability model for my dataset. [5 marks]

(ii) Explain how to fit your model. [5 marks]



This “propose a probability model” is open-ended and scary. How should we even begin to think about it?

1. Skim read for keywords. What’s the topic?
 2. Look for question words. What is it asking you to do?
 3. Think through the course. What sections are relevant?
 4. Read the whole question. What’s the link?
- Part (a) gave us a hammer. Can we see part (b) as a nail?

Summary so far

We have seen

- How probabilistic modelling plays a role in defining the relationships between ML and data science
- How to define a probabilistic model and fit the model from the data by MLE
- How a proper definition of likelihoods can help in navigating through discrete and continuous RVs
- How learning tasks can be categorised in supervised learning and generative AI
- How probabilistic ML is used to have a common framework to address supervised learning and generative AI (whilst algorithmic ML might struggle in some cases to define tasks, goals and procedures)

Deep learning with PyTorch (and a look back on neural networks)*

* non-examinable

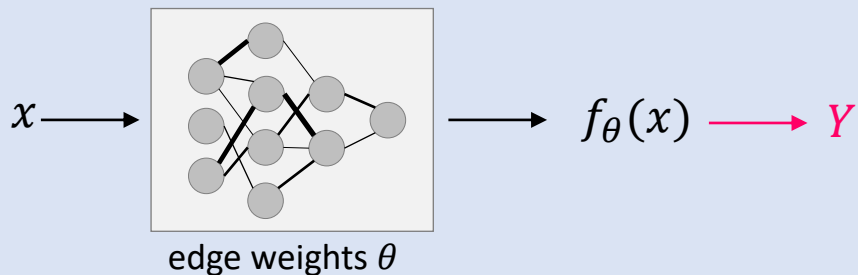
Supervised Learning

Data: $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$

Labels: y_1, y_2, \dots, y_n

Task: fit the probability model
 $\Pr_Y(y; f_\theta(x))$

Training goal: MLE



Generative Modelling

Data: $(x_1, y_1), \dots, (x_n, y_n)$

Labels: y_1, y_2, \dots, y_n

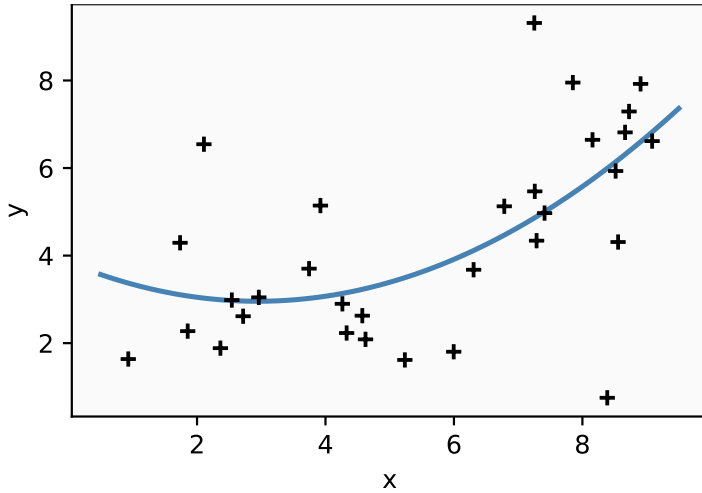
Task: fit the probability model
 $\Pr_{X,Y}(x, y)$

Training goal: MLE



Example (regression)

Given a labelled dataset consisting of pairs (x_i, y_i) of real numbers, fit the model $Y_i \sim \alpha + \beta x_i + \gamma x_i^2 + N(0, \sigma^2)$



Model for a single observation:

$$Y \sim \alpha + \beta x + \gamma x^2 + N(0, \sigma^2) \\ \sim N(\alpha + \beta x + \gamma x^2, \sigma^2)$$

Likelihood of a single observation:

$$\Pr_Y(y ; x, \alpha, \beta, \gamma, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y - (\alpha + \beta x + \gamma x^2))/2\sigma^2}$$

Log likelihood of the dataset:

$$\log \Pr(y_1, \dots, y_n; \alpha, \beta, \gamma, \sigma) = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

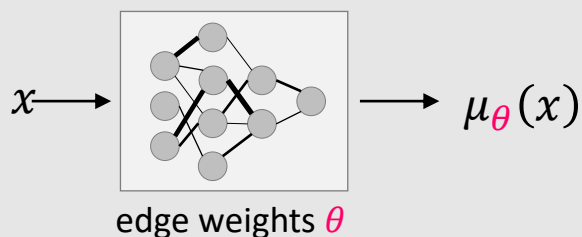
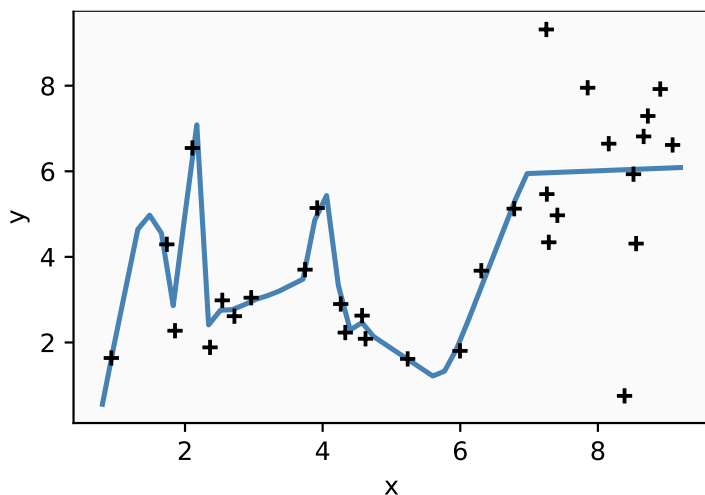
$$\text{where } \hat{y}_i = \alpha + \beta x_i + \gamma x_i^2$$

Optimize over the unknown parameters:

Example (regression)

Given a labelled dataset consisting of pairs (x_i, y_i) of real numbers, fit the model $Y_i \sim \mu_{\theta}(x_i) + N(0, \sigma^2)$.

(Here $\mu_{\theta}(\cdot)$ is some specified function with unknown parameters θ .)



Log likelihood of the dataset:

$$\log \Pr(y_1, \dots, y_n; \theta, \sigma) = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - \mu_{\theta}(x_i))^2$$

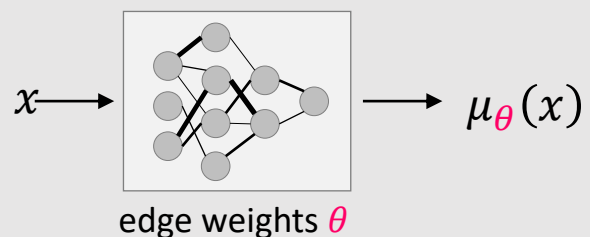
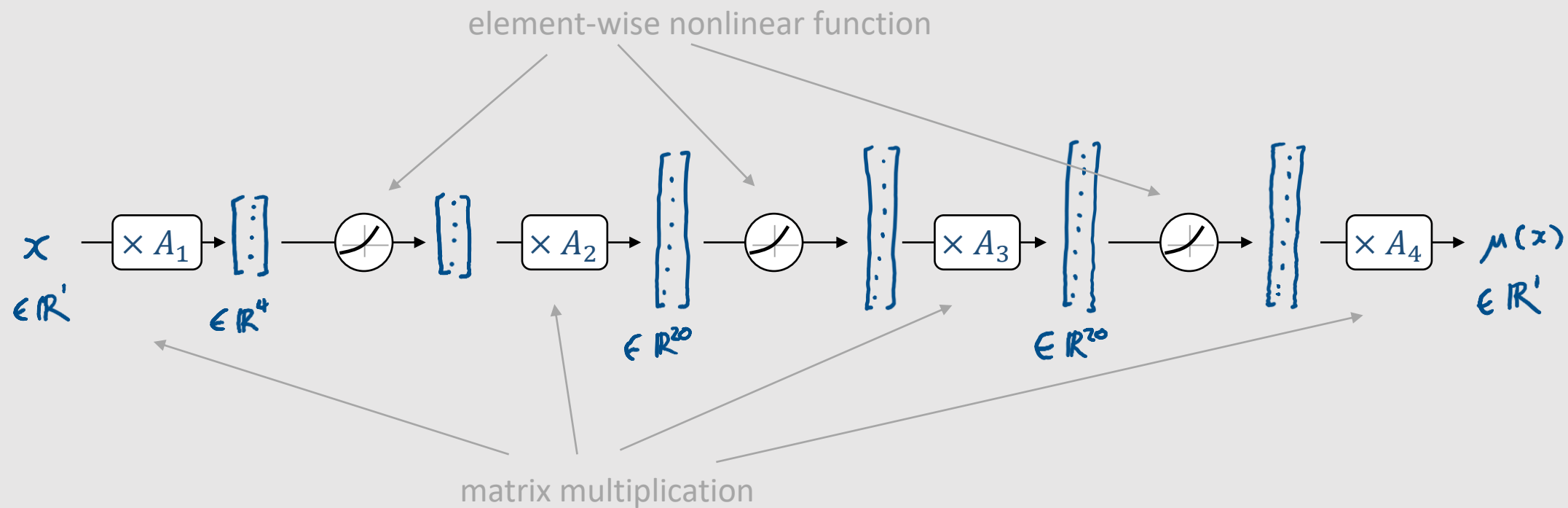
Optimize over the unknown parameters θ and σ :

```

1 class RWiggle(nn.Module):
2     def __init__(self):
3         super().__init__()
4         self.μ = ... # has parameters θ
5         self.σ = nn.Parameter(torch.tensor(1.0))
6
7         # compute log Pr(y;x)
8         def forward(self, y, x):
9             σ2 = self.σ ** 2
10            return - 0.5*torch.log(2*π*σ2) - ((y - self.μ(x)) ** 2) / (2*σ2)
11
12 x,y = ...
13 mymodel = RWiggle()
14
15 optimizer = optim.Adam(mymodel.parameters())
16 for epoch in range(10000):
17     optimizer.zero_grad()
18     loglik = torch.sum(mymodel(y, x))
19     (-loglik).backward()
20     optimizer.step()

```

See section 3.3 of printed notes.

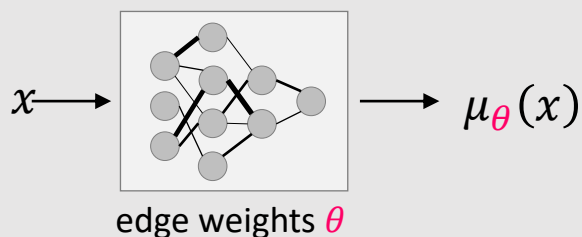
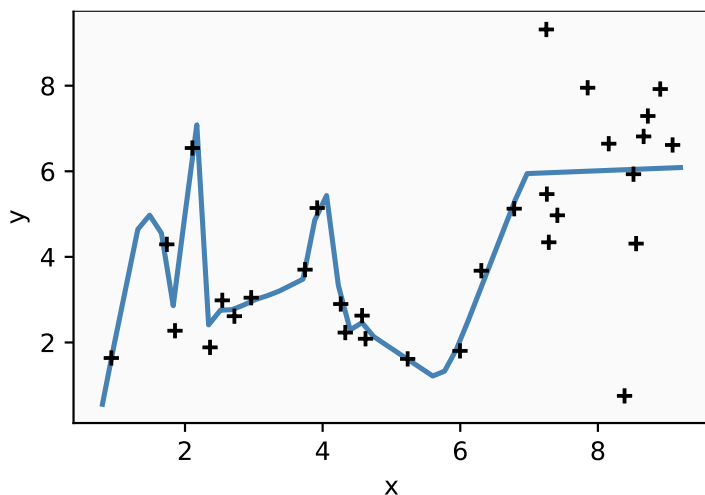


```
self.μ = nn.Sequential(
    nn.Linear(1,4), nn.LeakyReLU(),
    nn.Linear(4,20), nn.LeakyReLU(),
    nn.Linear(20,20), nn.LeakyReLU(),
    nn.Linear(20,1) )
```

Example (regression)

Given a labelled dataset consisting of pairs (x_i, y_i) of real numbers, fit the model $Y_i \sim \mu_{\theta}(x_i) + N(0, \sigma^2)$.

(Here $\mu_{\theta}(\cdot)$ is some specified function with unknown parameters θ .)



Log likelihood of the dataset:

$$\log \Pr(y_1, \dots, y_n; \theta, \sigma) = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - \mu_{\theta}(x_i))^2$$

Optimize over the unknown parameters θ and σ :

```

1 class RWiggle(nn.Module):
2     def __init__(self):
3         super().__init__()
4         self.μ = ... # has parameters θ
5         self.σ = nn.Parameter(torch.tensor(1.0))
6
7         # compute log Pr(y;x)
8         def forward(self, y, x):
9             σ2 = self.σ ** 2
10            return - 0.5*torch.log(2*π*σ2) - ((y - self.μ(x)) ** 2) / (2*σ2)
11
12 x,y = ...
13 mymodel = RWiggle()
14
15 optimizer = optim.Adam(mymodel.parameters())
16 for epoch in range(10000):
17     optimizer.zero_grad()
18     loglik = torch.sum(mymodel(y, x))
19     (-loglik).backward()
20     optimizer.step()

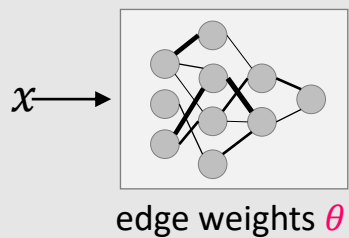
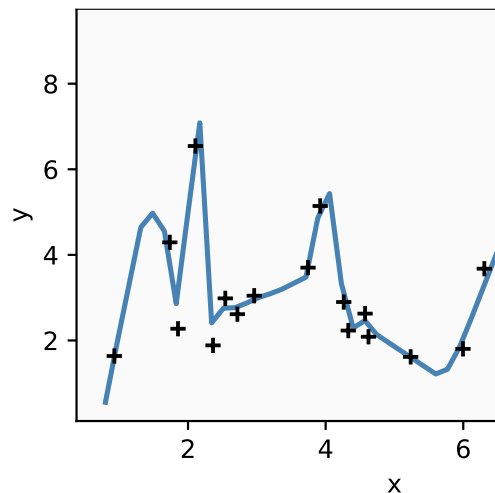
```

See section 3.3 of printed notes.

Example (regression)

Given a labelled dataset consisting of pairs (x_i, y_i) of real numbers, we model $Y_i \sim \mu_{\theta}(x_i) + N(0, \sigma^2)$

(Here $\mu_{\theta}(\cdot)$ is some specified function with unknown parameters θ)



$$- \mu_{\theta}(x_i))^2$$

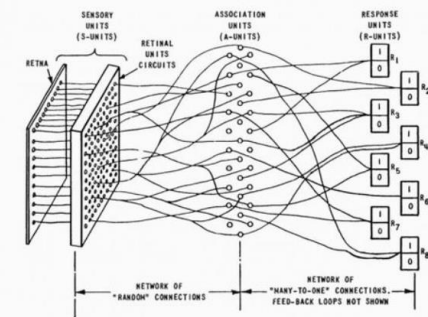
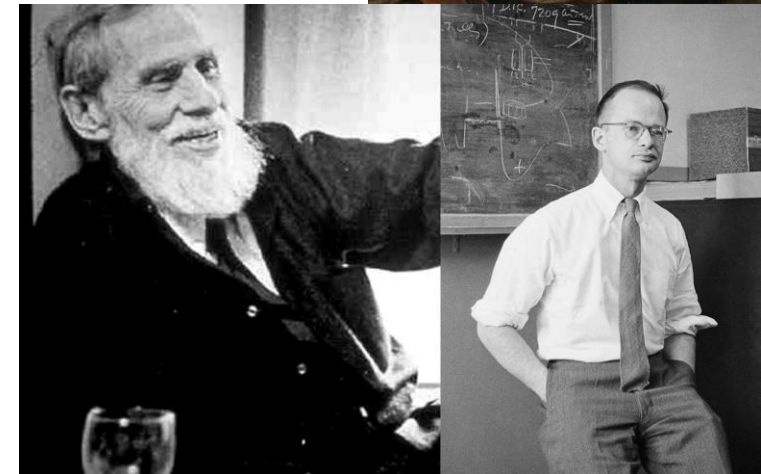
$$(\mu_{\theta}(x) - y_i)^2 / (2\sigma^2)$$

on 3.3 of printed notes.

A look back on NNs

Neural networks have been:

- Theorised out of the thoughts of Leibniz
- Built up on the first neuron by McCulloch & Pitts
- Developed from the perceptron by Rosenblatt
- *Started to address the problem of nonlinear classifiers*

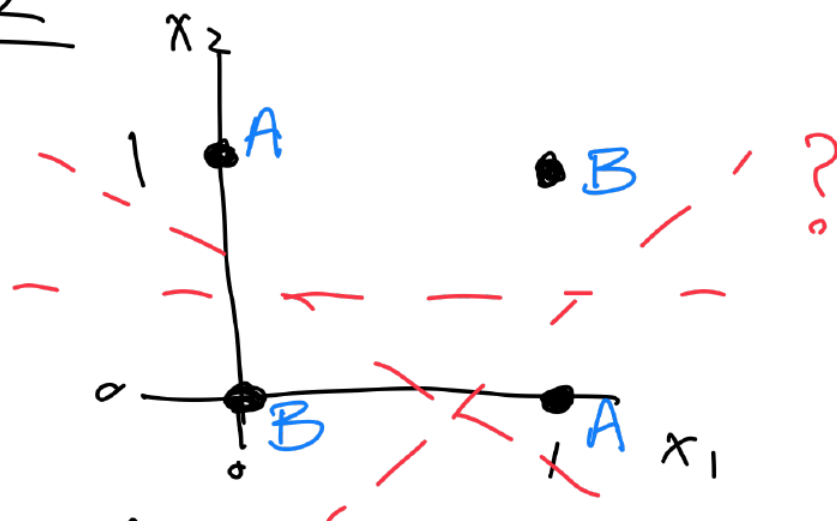


XOR

"exclusive or"

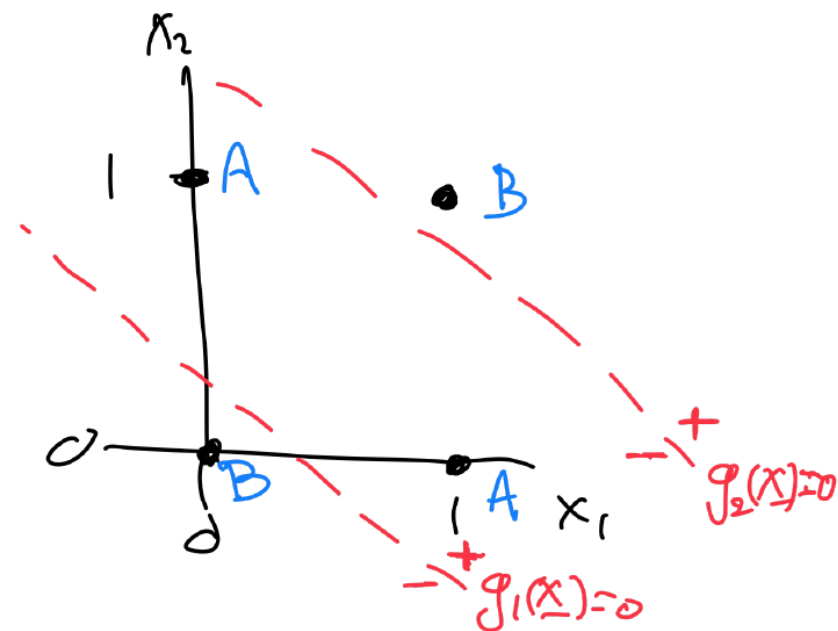
x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

XOR

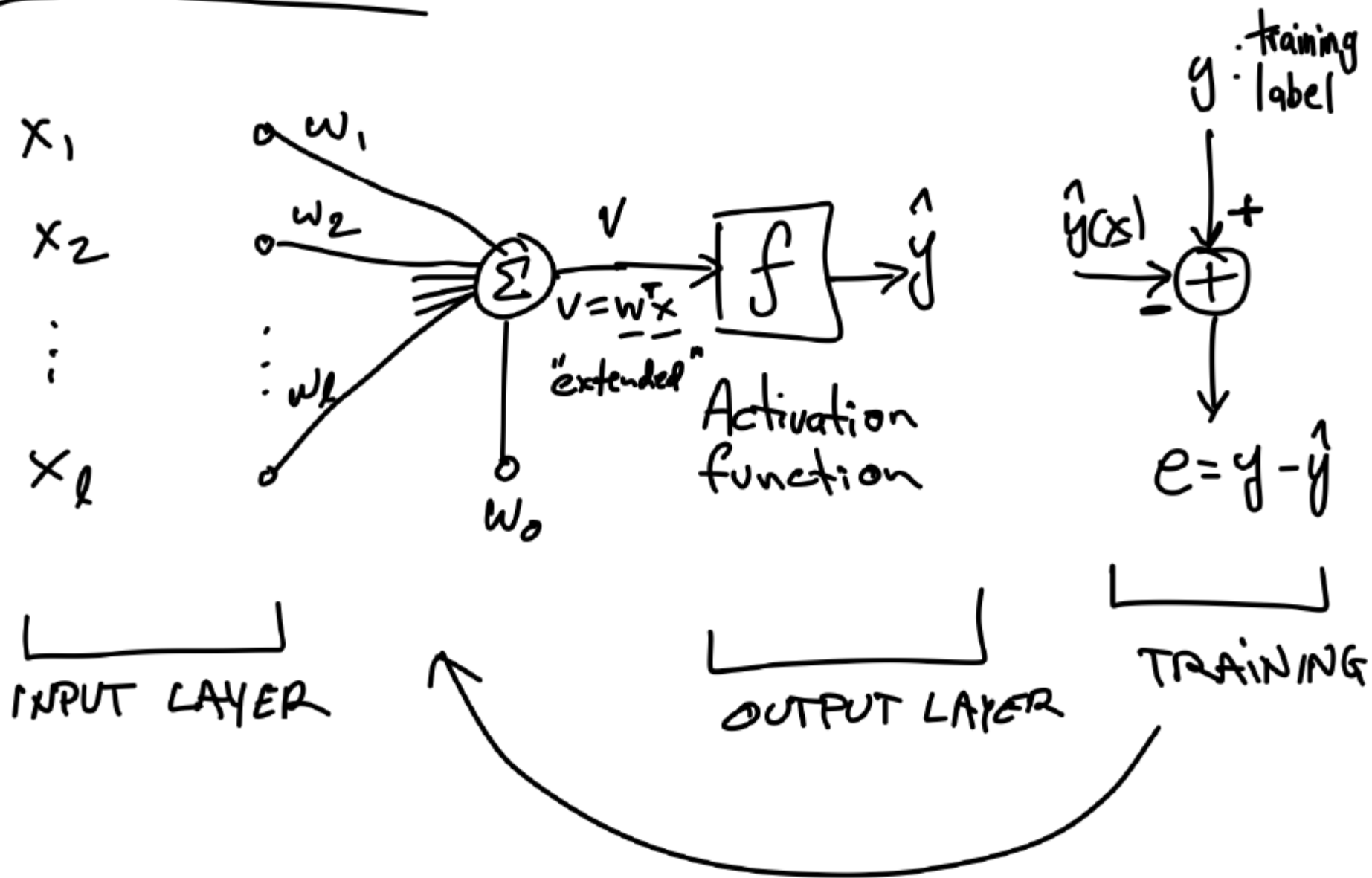


⇒ CAN'T DO LINEARLY!
· Always has some errors.

But: Can do with 2 lines.



Perceptron : 2 class



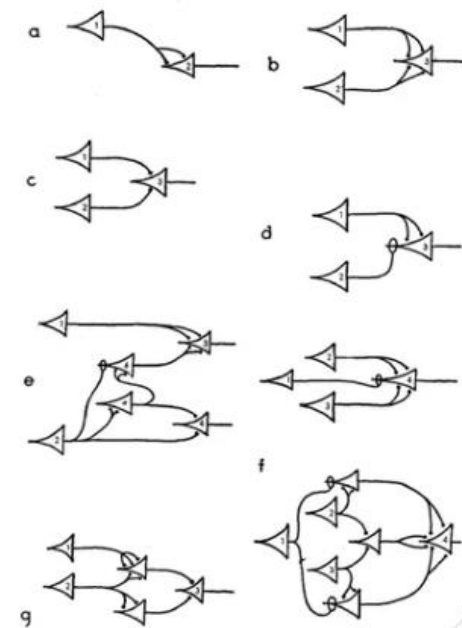
1943

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH AND WALTER PITTS

FROM THE UNIVERSITY OF ILLINOIS, COLLEGE OF MEDICINE,
DEPARTMENT OF PSYCHIATRY AT THE ILLINOIS NEUROPSYCHIATRIC INSTITUTE,
AND THE UNIVERSITY OF CHICAGO

Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.



- Attempted to demonstrate that
- the neuron was the base logic unit of the brain
 - a Turing machine program could be implemented in a finite network of formal neurons

1940

PROCEEDINGS OF THE IRE

November

What the Frog's Eye Tells the Frog's Brain*

J. Y. LETTVIN†, H. R. MATURANA‡, W. S. McCULLOCH||, SENIOR MEMBER, IRE,
AND W. H. PITTS||

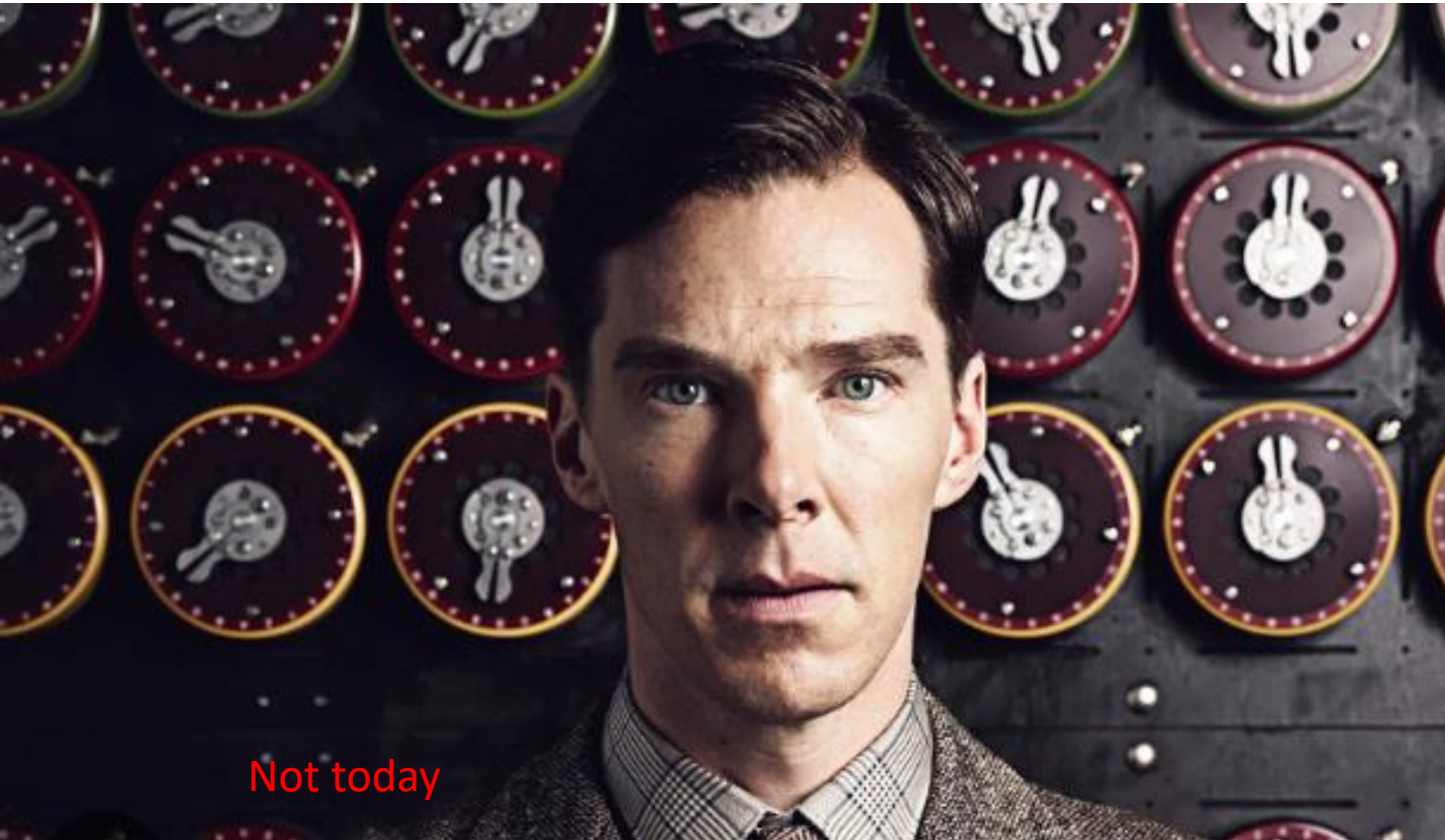
Summary—In this paper, we analyze the activity of single fibers in the optic nerve of a frog. Our method is to find what sort of stimulus causes the largest activity in one nerve fiber and then what is the exciting aspect of that stimulus such that variations in everything else cause little change in the response. It has been known for the past 20 years that each fiber is connected not to a few rods and cones in the retina but to very many over a fair area. Our results show that for the most part within that area, it is not the light intensity itself but rather the pattern of local variation of intensity that is the exciting factor. There are four types of fibers, each type concerned with a different sort of pattern. Each type is uniformly distributed over the whole retina of the frog. Thus, there are four distinct parallel distributed channels whereby the frog's eye informs his brain about the visual image in terms of local pattern independent of average illumination. We describe the patterns and show the functional and anatomical separation of the channels. This work has been done by the frog, and our interpretation applies only to the frog.

it moves like one. He can be fooled easily not only by a bit of dangled meat but by any moving small object. His sex life is conducted by sound and touch. His choice of paths in escaping enemies does not seem to be governed by anything more devious than leaping to where it is darker. Since he is equally at home in water and on land, why should it matter where he lights after jumping or what particular direction he takes? He does remember a moving thing providing it stays within his field of vision and he is not distracted.

Anatomy of Frog Visual Apparatus

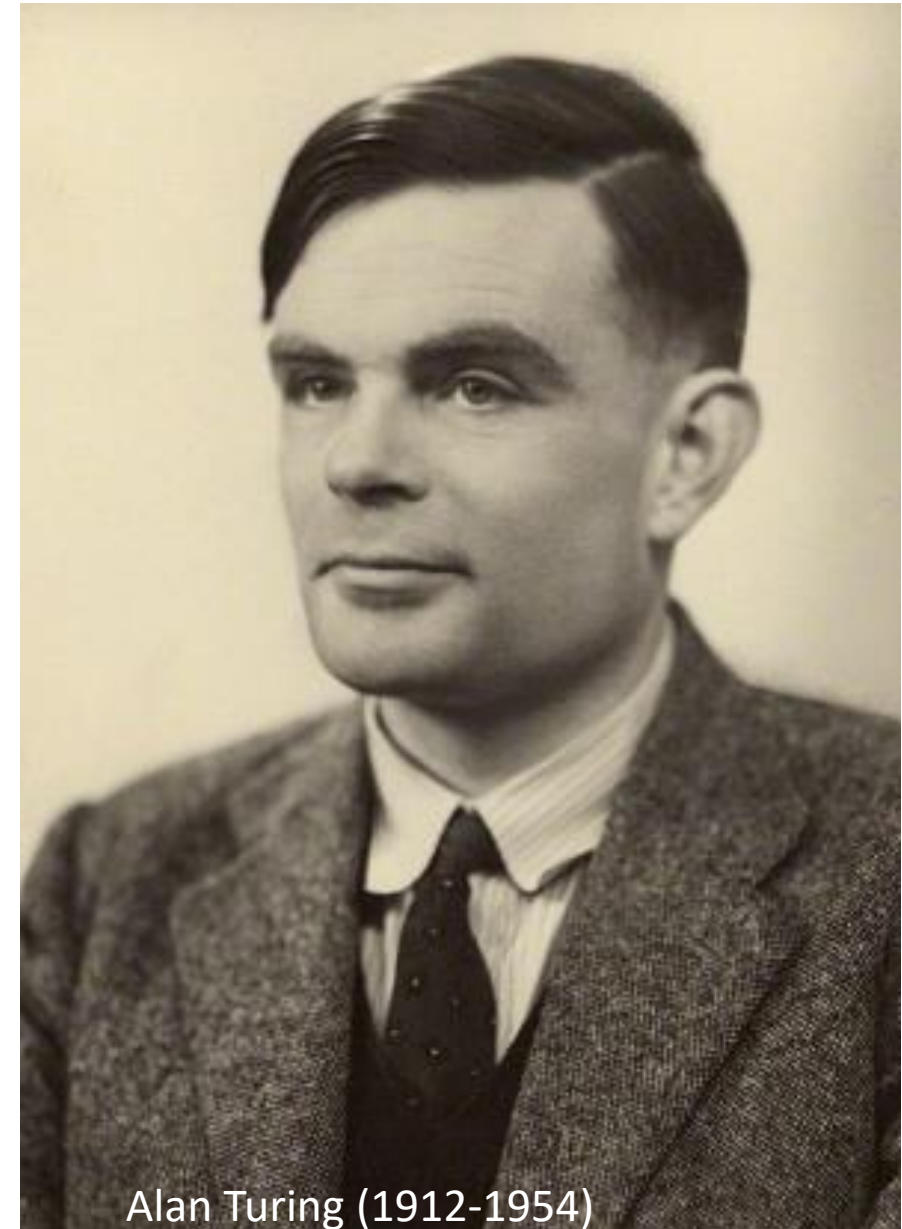
The retina of a frog is shown in Fig. 1(a). Between the rods and cones of the retina and the ganglion cells, whose axons form the optic nerve, lies a layer of con-

1959

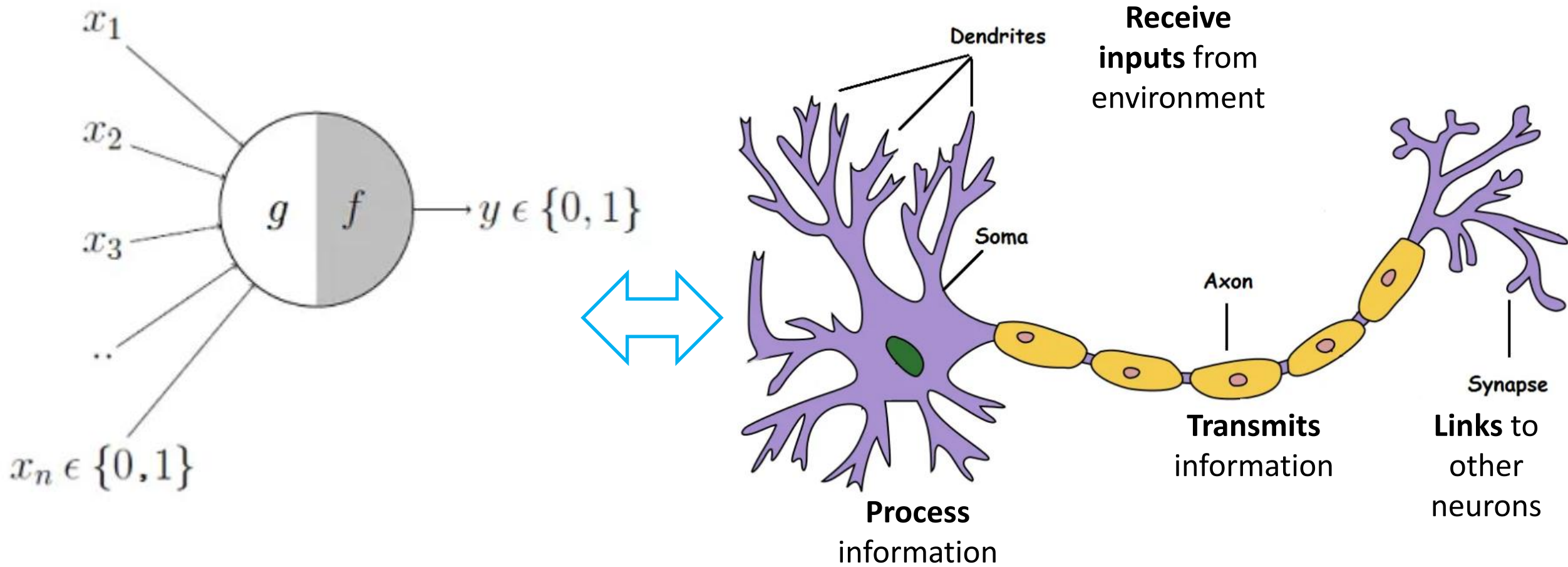


Not today

A Turing machine is a mathematical model of computation describing an abstract machine that manipulates symbols on a strip of tape according to a table of rules. Despite the model's simplicity, it is capable of implementing any computer algorithm.



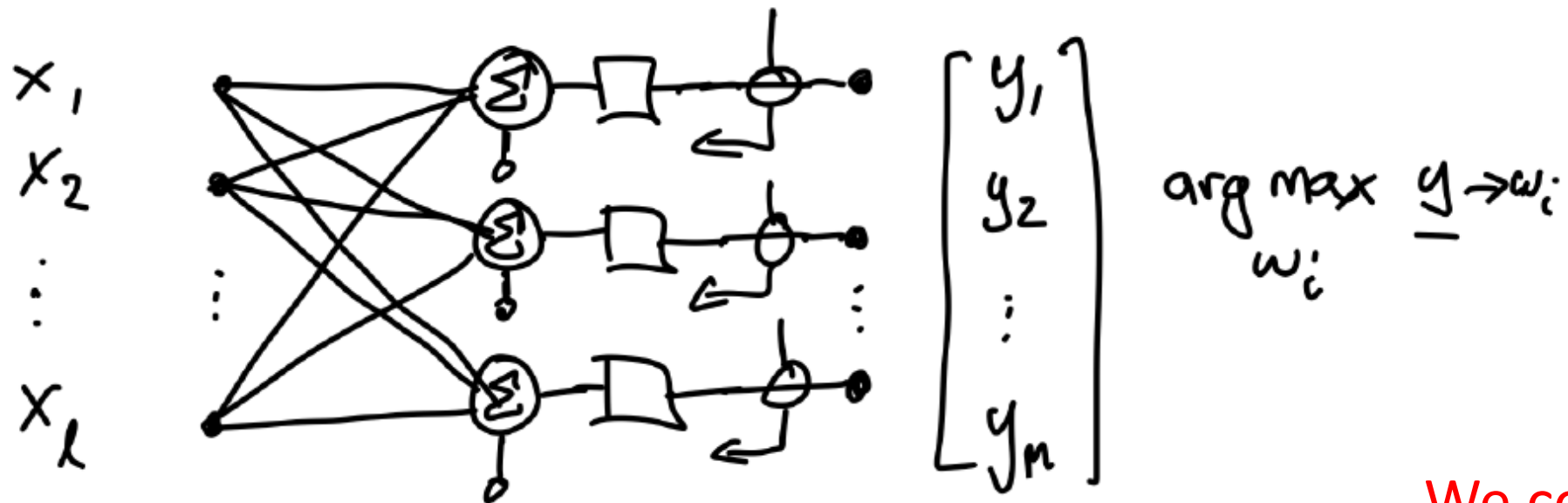
Alan Turing (1912-1954)



In particular:

- g models the ability to weigh the inputs \rightarrow typically weighted sum
- f models the ability to choose \rightarrow typically step (indicator) function

Multi-class case.



Training vectors

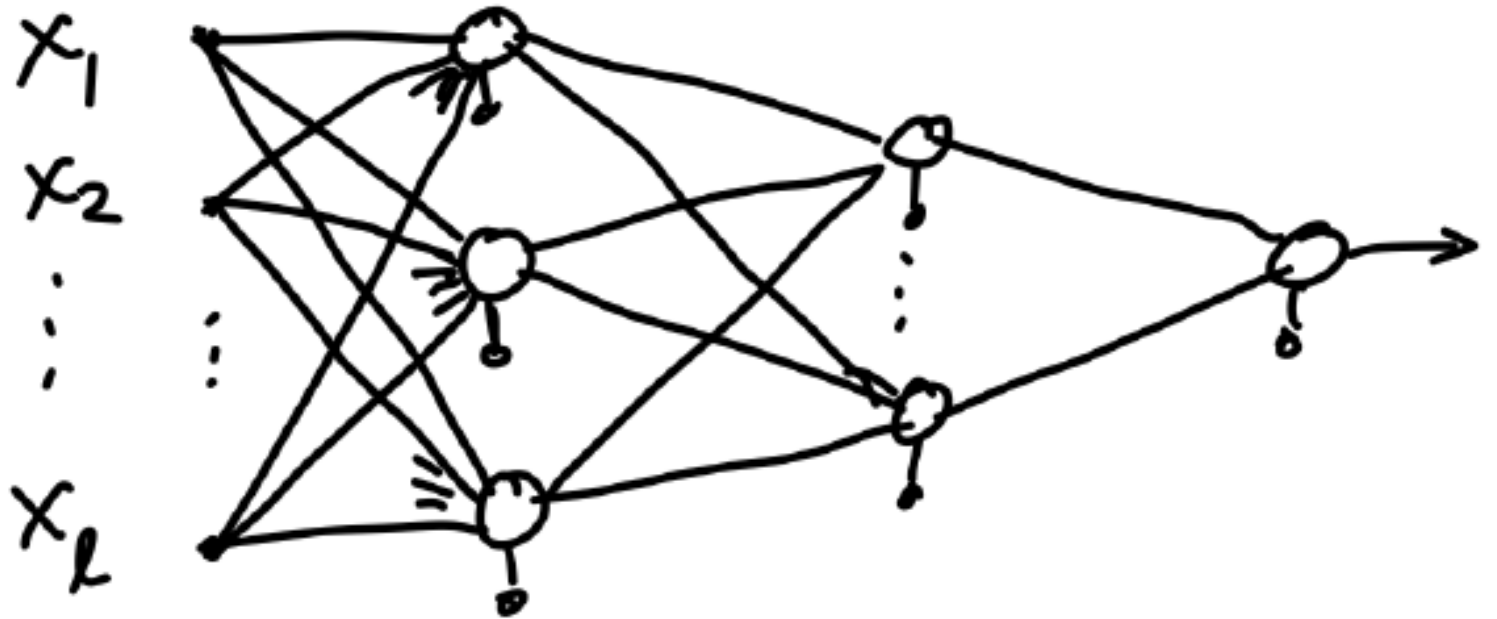
index $w_k = 1$
else 0

$$\begin{bmatrix} 1 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

$\underline{y}(x_i)$

We could add more neurons “vertically”, but also “horizontally”

Multi-Layer Perceptron



The number of neurons in the second hidden layer can be reduced by exploiting the geometry that results from each specific problem

In general:

- The number of neurons in the hidden layers help us to better define the hyperplanes to take decisions (*classifications*)
- The number of neurons in the output layer define the number of classes we can recognise

Ok... but what about the *limits*?!

Main problem: we do not have all this info typically

- How the classes are composed, where they are located in the data space, the analytical expressions of the hyperplanes...

We have to learn all this from the training data that we have (if/when possible)

What are our degrees of freedom?

Assume : L layers of neurons.

$K_0 = l$ nodes in input layer.

$K_r ; r=1, \dots, L$ nodes in layer r .

Same activation function in all neurons.

N training pairs $(y(i), \underline{x}(i))$
 $i = 1 \dots N$

K_L output neurons

\Rightarrow a vector \underline{K} containing number of nodes in layers

Feature vectors $\underline{x}(i) = \begin{bmatrix} x_1(i) \\ x_2(i) \\ \vdots \\ x_{K_L}(i) \end{bmatrix}$

$i = 1, \dots, N$

Desired output

$$i=1, \dots, N$$

$$\underline{y}(i) = \begin{bmatrix} y_1(i) \\ y_2(i) \\ \vdots \\ y_{K_L}(i) \end{bmatrix}$$

System output

$$\underline{\hat{y}}(i) = \begin{bmatrix} \hat{y}_1(i) \\ \hat{y}_2(i) \\ \vdots \\ \hat{y}_{K_L}(i) \end{bmatrix}$$

Hence, the j^{th} neuron in the r^{th} layer will have weights

$$\underline{W}_j^r = \begin{bmatrix} W_{j0}^r \\ W_{j1}^r \\ \vdots \\ W_{j K_{r-1}}^r \end{bmatrix}$$

NB: j out of K_r nodes in layer r
but length of \underline{W}_j^r depends on
 $K_{(r-1)}$ in previous layer.

In summary:

- There are several degrees of freedom (parameters) that we should estimate
- Exact methods for estimation (e.g., using analytical expressions) might be very cumbersome
 - Lots of potential local minima that the optimisation strategy might fall in
- This could have been a major problem for the development (and success) of NNs
- A solution for this issue came from...

Backpropagation

Training mode

Adjust the weights in each neuron
such that a cost function $J(y(i), \hat{y}(i))$
minimises for $i=1, \dots, N$.

Skip to slide 39 during class

Adjust the weights iteratively
until the cost function minimises

$$\underline{w}_j^r(\text{new}) = \underline{w}_j^r(\text{old}) + \Delta \underline{w}_j^r$$

$$\Delta \underline{w}_j^r = -\mu \frac{\partial J}{\partial \underline{w}_j^r}$$

↑
rate parameter

Standard Cost function

$$J = \sum_{i=1}^N \mathcal{E}(i)$$

$$\therefore \mathcal{E}(i) = \frac{1}{2} \sum_{m=1}^{K_L} e_m^2$$

$$= \frac{1}{2} \sum_{m=1}^{K_L} [y_m(i) - \hat{y}_m(i)]^2$$

The sum of squared errors at output layer.

Computation of gradients $\frac{\partial J}{\partial \underline{w}_j^r}$

Let $\underline{y}_k^{r-1}(i)$ be the output of the k^{th} neuron in layer $(r-1)$ for training pair i , and w_{jk}^r an estimate of weight j in the r^{th} layer, $k = 1, 2, \dots, K_{r-1}$.

The argument of the activation function $f(\cdot)$

is

$$V_j^r(i) = \sum_{k=1}^{K_{r-1}} W_{jk}^r y_k^{r-1} + W_{j0}^r$$



$$= \underline{W}^r{}^T \underline{y}^{r-1}(i)$$

NB: $y_0^r(i) = 1$
"extended"
 $\underline{y}' = \begin{bmatrix} 1 \\ \underline{y} \end{bmatrix}$ $\underline{w}' = \begin{bmatrix} w_0 \\ \underline{w} \end{bmatrix}$

observe that $\mathcal{E}(i)$ depends on \underline{w}_j^r
through $V_j^r(i)$. Hence,

$$\frac{\partial \mathcal{E}(i)}{\partial \underline{w}_j^r} = \frac{\partial \mathcal{E}(i)}{\partial V_j^r(i)} \cdot \frac{\partial V_j^r(i)}{\partial \underline{w}_j^r}$$

"chain rule"

Furthermore,

$$\frac{\partial V_j^r(i)}{\partial \underline{w}_j^r} = \frac{\partial}{\partial \underline{w}_j^r} \left\{ (\underline{w}_j^r)^T \underline{y}^{r-1}(i) \right\} = y^{r-1}(i)$$



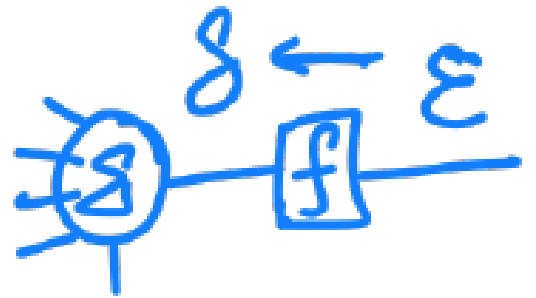
Define $\delta_j^r(i) \triangleq \frac{\partial \mathcal{E}(i)}{\partial V_j^r(i)}$

Then

$$\Delta \underline{w}_j^r = -\mu \sum_{i=1}^N \delta_j^r(i) y^{r-1}(i)$$

Delta learning rule.

The form of $\delta_j^r(i)$ depends on the cost function we choose.



Apply the total squared error cost function and note that we must start at the output layer where this error can be calculated, and work backwards through the network.

→ the Backpropagation algorithm.

i) $r = L$, $\delta_j^L(i) = \frac{\partial \mathcal{E}(i)}{\partial V_j^L(i)}$,

(output layer)

$$\mathcal{E}(i) = \frac{1}{2} \sum_{m=1}^{K_L} \underbrace{\left(f(V_m^L(i)) - y_m(i) \right)^2}_{e_m^2}$$

$$\Rightarrow \delta_j^L(i) = e_j(i) f'(V_j^L(i))$$

function derivative

* see below

ii) $r < L$ (hidden layers)



value of $V_j^{r-1}(i)$ influences all $V_j^r(i)$,
 $j = 1, \dots, K_r$, of the next layer.

$$\frac{\partial \mathcal{E}(i)}{\partial V_j^{r-1}(i)} = \sum_{k=1}^{K_r} \frac{\partial \mathcal{E}(i)}{\partial V_k^r(i)} \cdot \frac{\partial V_k^r(i)}{\partial V_j^{r-1}(i)} \quad \text{chain rule}$$

$$\Rightarrow \delta_j^{r-1}(i) = \sum_{k=1}^{K_r} \delta_k^r(i) \frac{\partial V_k^r(i)}{\partial V_j^{r-1}(i)}$$

but,
$$\frac{\partial V_k^r(i)}{\partial V_j^{r-1}(i)} = \frac{\partial}{\partial V_j^{r-1}(i)} \left\{ \sum_{m=0}^{K_{r-1}} w_{km}^r y_m^{r-1}(i) \right\}$$

where
$$y_m^{r-1}(i) = f(V_m^{r-1}(i))$$

If we choose f = indicator function,
then $f' \rightarrow +\infty$

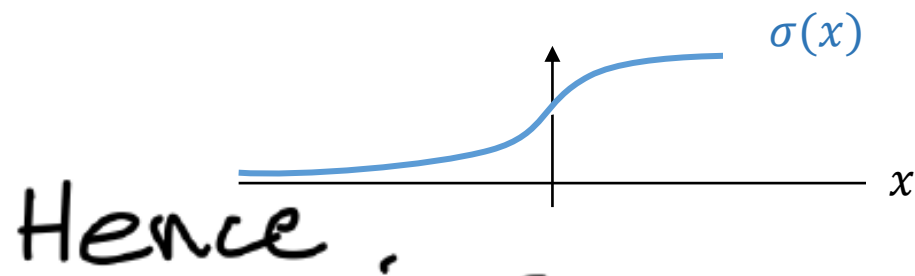
Hence,

$$\frac{\partial V_k^r(i)}{\partial V_j^{r-1}(i)} = w_{km}^r f'(V_j^{r-1}(i))$$

*

$$\Rightarrow \delta_j^{r-1}(i) = \left[\sum_{k=1}^{K_r} \delta_k^r(i) w_{kj}^r \right] f'(V_j^{r-1}(i))$$

we have all these terms, hence
iteration can be done for all layers.



A function like sigmoid can unlock the potential of this approach

$$\frac{\partial V_k^r(i)}{\partial V_j^{r-1}(i)} = w_{km}^r f'(V_j^{r-1}(i))$$



$$\Rightarrow \delta_j^{r-1}(i) = \left[\sum_{k=1}^{K_r} \delta_k^r(i) w_{kj}^r \right] f'(V_j^{r-1}(i))$$

we have all these terms, hence
iteration can be done for all layers.

A lifetime NNs

- Research on NNs started from the 60s, and flourished until the 80s (-ish)
- At one point, NNs were found unfeasible
 - The technology to support the computational power required for complex systems was not there yet
- Research and development of NNs restarted in early 2000 (-ish), thanks to...



Supervised Learning

Data: $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$

Labels: y_1, y_2, \dots, y_n

Task: fit the probability model
 $\Pr_Y(y : f_\theta(x))$

Training goal: MLE



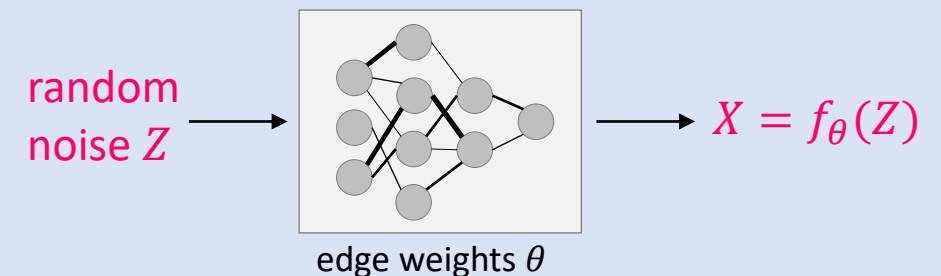
Generative Modelling

Data: $\{x_1, x_2, \dots, x_n\}$

Labels: n/a

Task: fit the probability model
 $\Pr_X(x ; \theta)$

Training goal: MLE



RICE CRUMB #2

$$\lim_{N \rightarrow \infty} P(\|\hat{\underline{\theta}}_{ML} - \underline{\theta}_0\| < \varepsilon) =$$

$$\lim_{N \rightarrow \infty} E[\|\hat{\underline{\theta}}_{ML} - \underline{\theta}_0\|^2] =$$