M9. (Artificial) Neural Networks (ANNs)

Manikandan Narayanan

Week 15 (Nov 3- 2025)

PRML Jul-Nov 2025 (Grads Section)

Acknowledgment of Sources

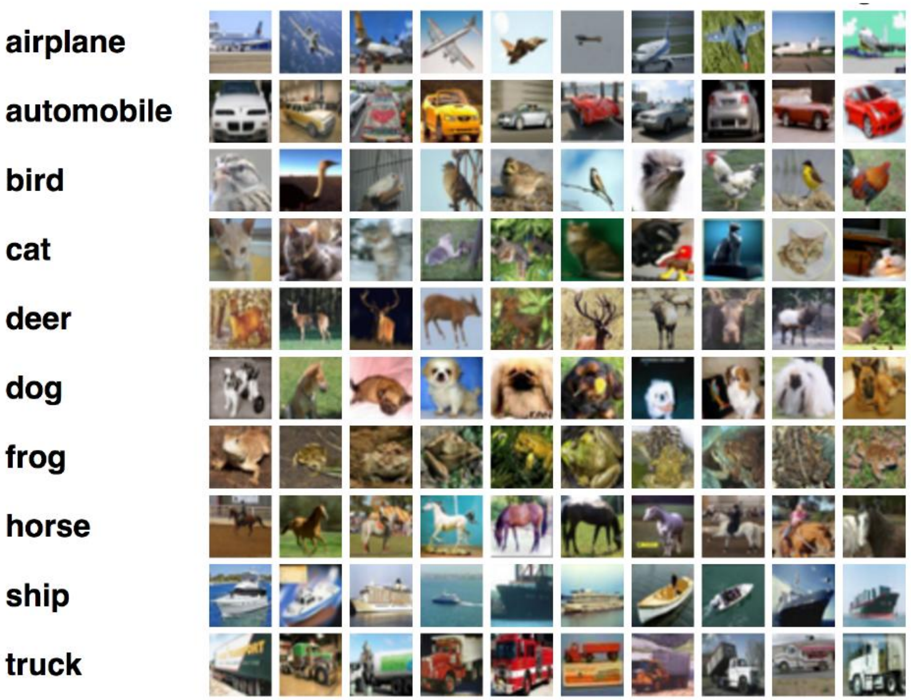
- Slides based on content from related
  - Courses:
    - IITM – Profs. Arun/Harish/Chandra’s PRML offerings (slides, quizzes, notes, etc.), Prof. Ravi’s “Intro to ML” slides – cited respectively as [AR], **[HR]**, [CC], [BR] in the bottom right of a slide.
    - India – NPTEL PR course by IISc Prof. PS. Sastry (slides, etc.) – cited as [PSS] in the bottom right of a slide.
  - Books:
    - PRML by Bishop. (content, figures, slides, etc.) – cited as **[CMB]**
    - Pattern Classification by Duda, Hart and Stork. (content, figures, etc.) – [DHS]
    - Mathematics for ML by Deisenroth, Faisal and Ong. (content, figures, etc.) – [DFO]
    - Information Theory, Inference and Learning Algorithms by David JC MacKay – [DJM]

Outline for Module M7

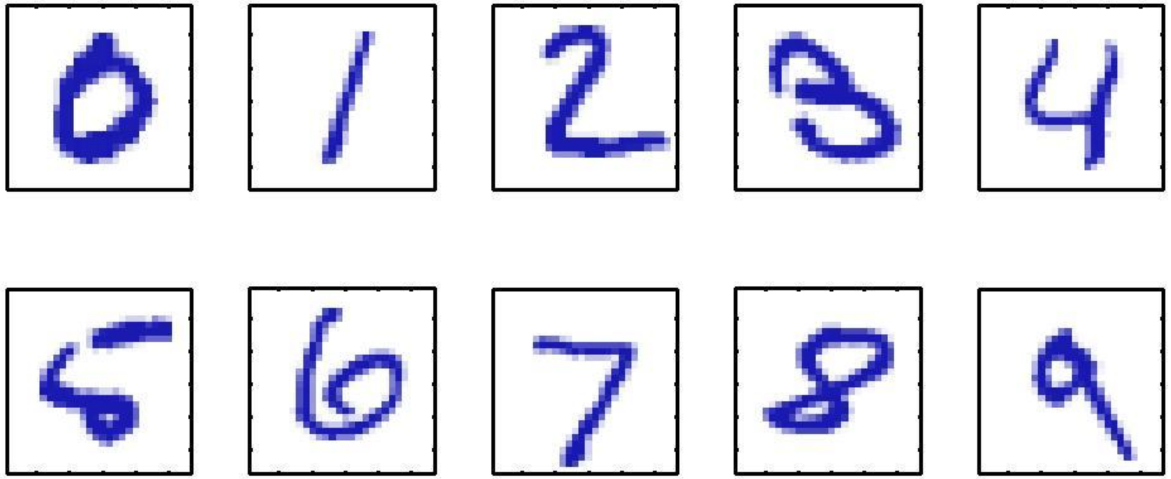
- M7. Neural Networks
  - **M7.0 Introduction/Motivation**
  - M7.1 Feed-forward neural networks
    - (Key Idea: Adaptive Basis functions and its Generalizations)
    - (Network architecture, Network training & Backpropagation algo. sketch)
  - M7.2 Concluding thoughts

Recall: popular examples

Classify an image into one of 10 classes, given a “training set” of images with classes.

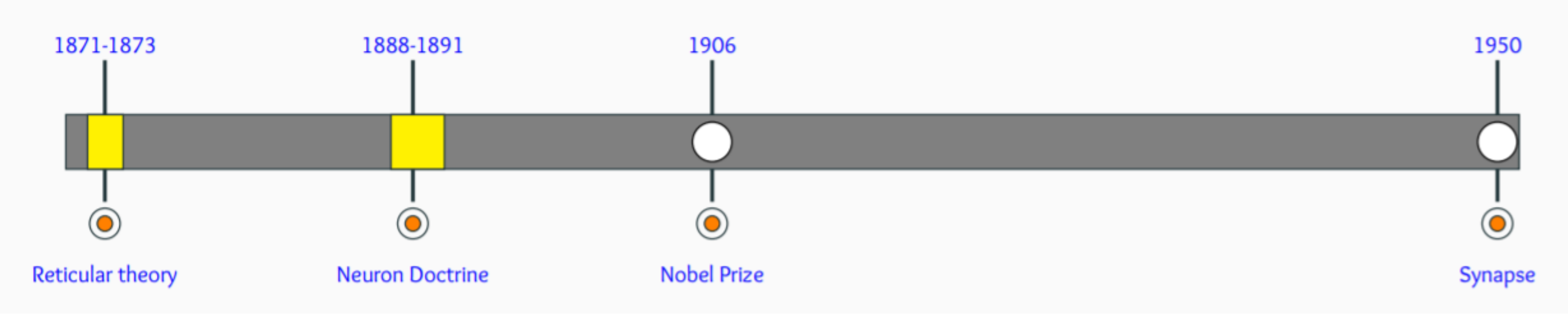


Handwritten Digit Recognition



[HR]

Very brief history of early research on Biological & Artificial Neural Networks

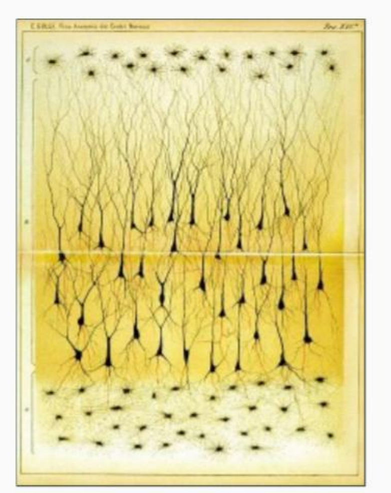


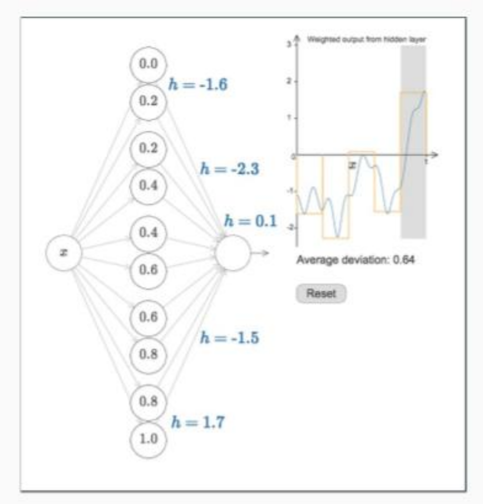
1871-1873 Reticular theory

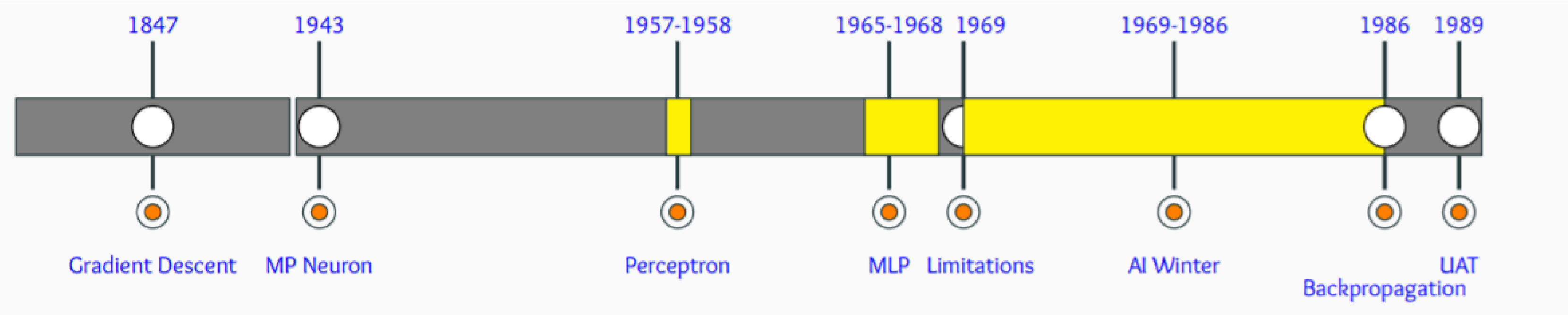
1888-1891 Neuron Doctrine

1906 Nobel Prize

1950 Synapse







1847 Gradient Descent

1943 MP Neuron

1957-1958 Perceptron

1965-1968 MLP

1969 Limitations

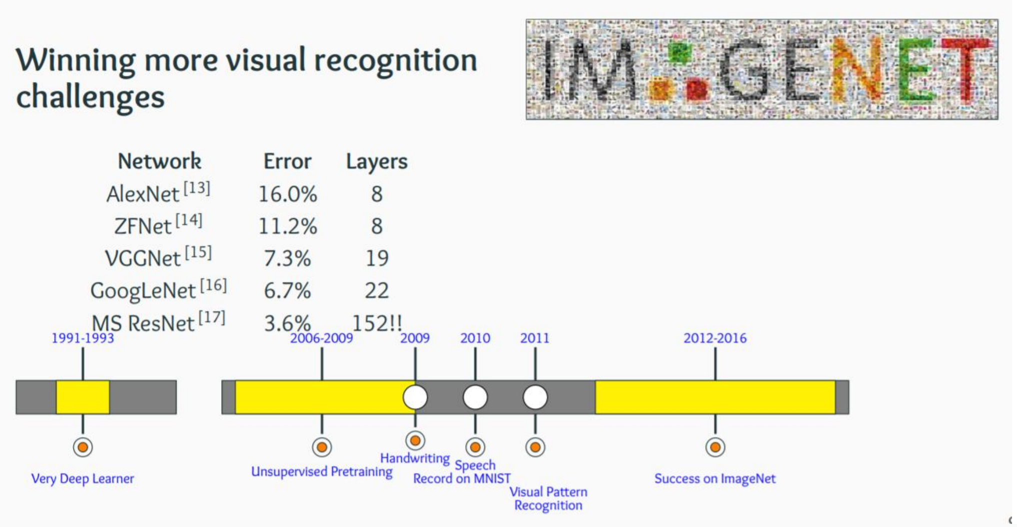
1969-1986 AI Winter

1986 Backpropagation

1989 UAT

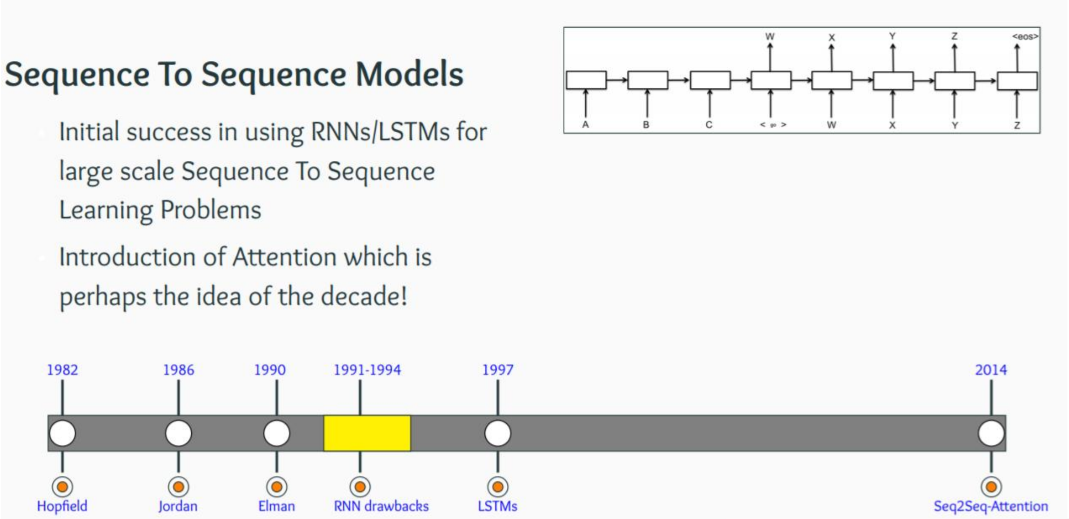
[From MK: https://www.cse.iitm.ac.in/~miteshk/CS6910/Slides/Lecture1.pdf]

(Recent) Success stories of ANNs/DL



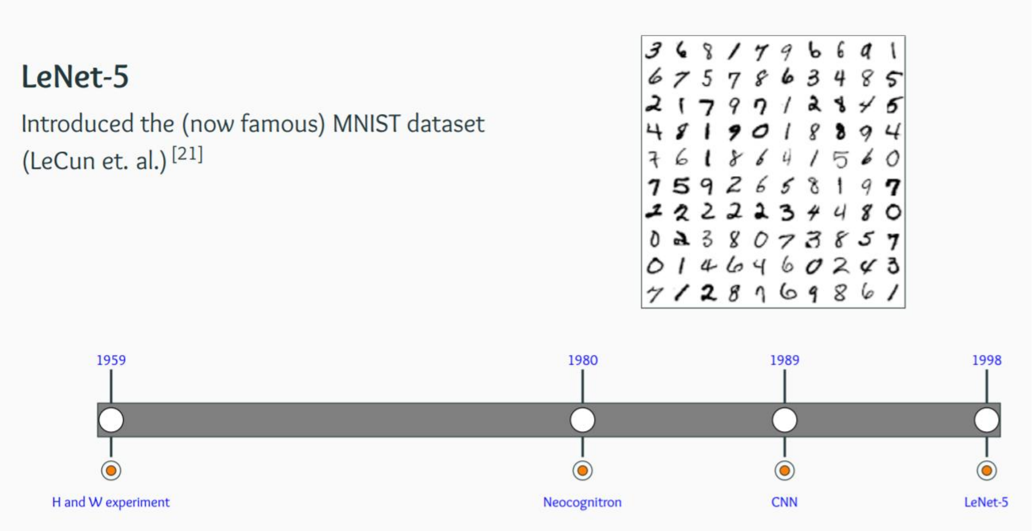
Winning more visual recognition challenges

Network	Error	Layers
AlexNet [13]	16.0%	8
ZFNet [14]	11.2%	8
VGGNet [15]	7.3%	19
GoogLeNet [16]	6.7%	22
MS ResNet [17]	3.6%	152!!



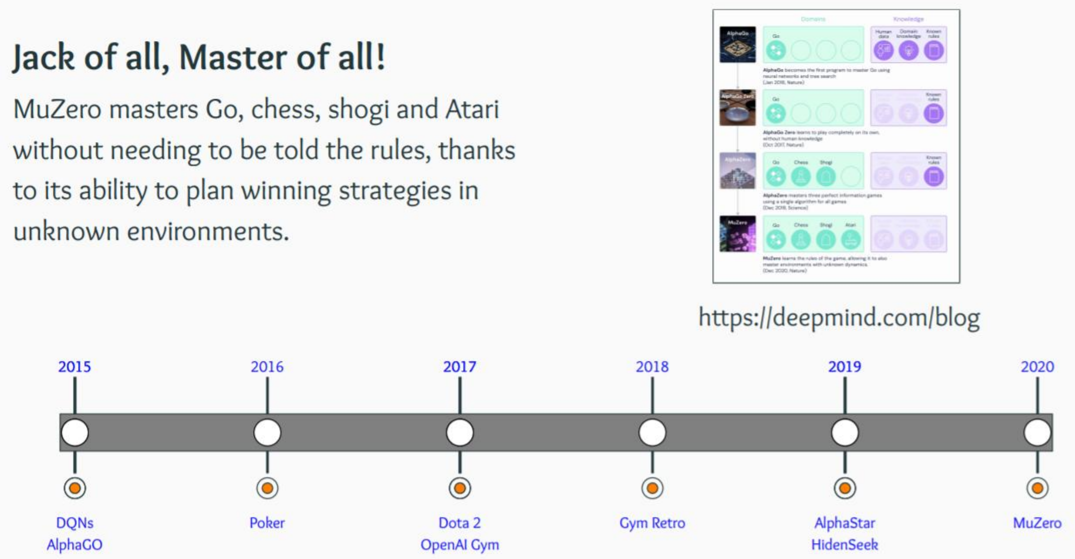
Sequence To Sequence Models

- Initial success in using RNNs/LSTMs for large scale Sequence To Sequence Learning Problems
- Introduction of Attention which is perhaps the idea of the decade!



LeNet-5

Introduced the (now famous) MNIST dataset (LeCun et. al.) [21]



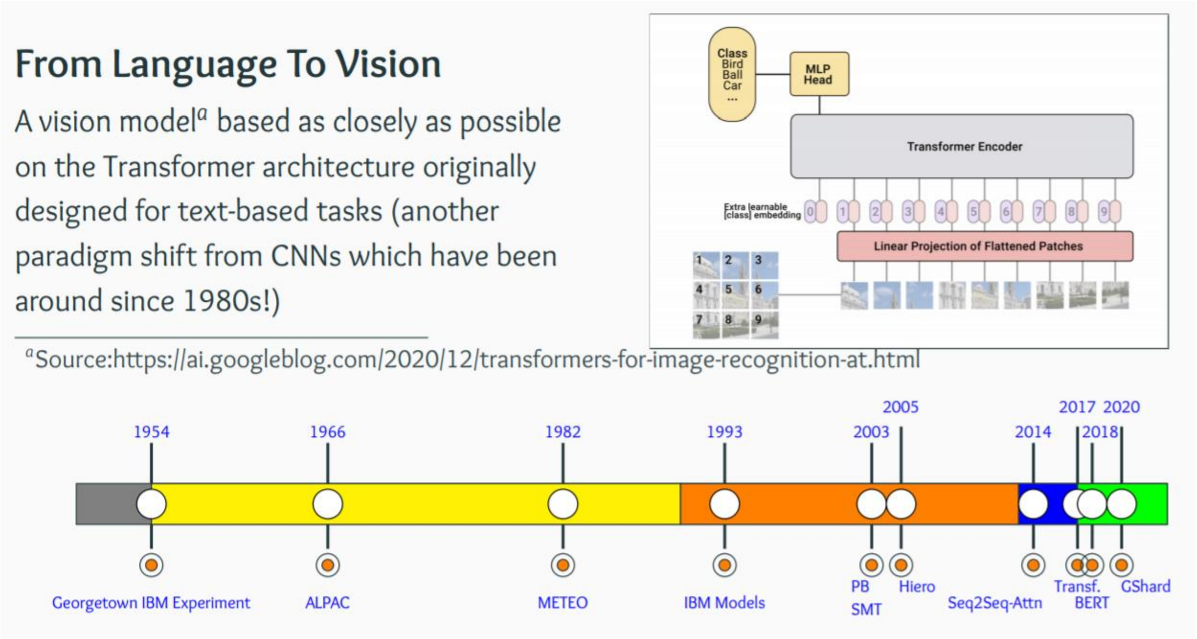
Jack of all, Master of all!

MuZero masters Go, chess, shogi and Atari without needing to be told the rules, thanks to its ability to plan winning strategies in unknown environments.

https://deepmind.com/blog

[MK]

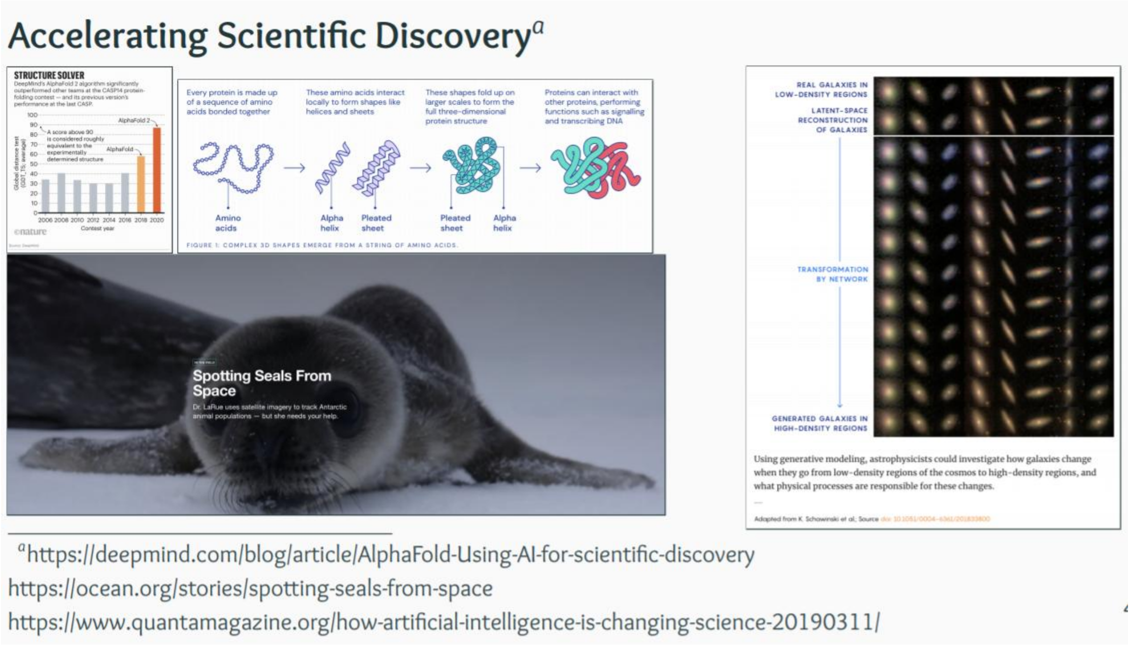
More (recent) success stories of ANNs/DL



From Language To Vision

A vision model[a] based as closely as possible on the Transformer architecture originally designed for text-based tasks (another paradigm shift from CNNs which have been around since 1980s!)

[a] Source:https://ai.googleblog.com/2020/12/transformers-for-image-recognition-at.html



Accelerating Scientific Discovery[a]

STRUCTURE SOLVER

DeepMind's AlphaFold 2 algorithm significantly outperformed other teams at the CASP14 protein-folding contest — and its previous version's performance at the last CASP.

Global distance test (GDT_TS; average)
100
90
80
70
60
50
40
30
20
10
0
2006
2008
2010
2012
2014
2016
2018
2020
Contest year
A score above 90 is considered roughly equivalent to the experimentally determined structure
AlphaFold
AlphaFold 2
©nature

Every protein is made up of a sequence of amino acids bonded together

These amino acids interact locally to form shapes like helices and sheets

These shapes fold up on larger scales to form the full three-dimensional protein structure

Proteins can interact with other proteins, performing functions such as signalling and transcribing DNA

FIGURE 1: COMPLEX 3D SHAPES EMERGE FROM A STRING OF AMINO ACIDS.

Spotting Seals From Space

Dr. LaRue uses satellite imagery to track Antarctic animal populations — but she needs your help.

REAL GALAXIES IN LOW-DENSITY REGIONS
LATENT-SPACE RECONSTRUCTION OF GALAXIES
TRANSFORMATION BY NETWORK
GENERATED GALAXIES IN HIGH-DENSITY REGIONS

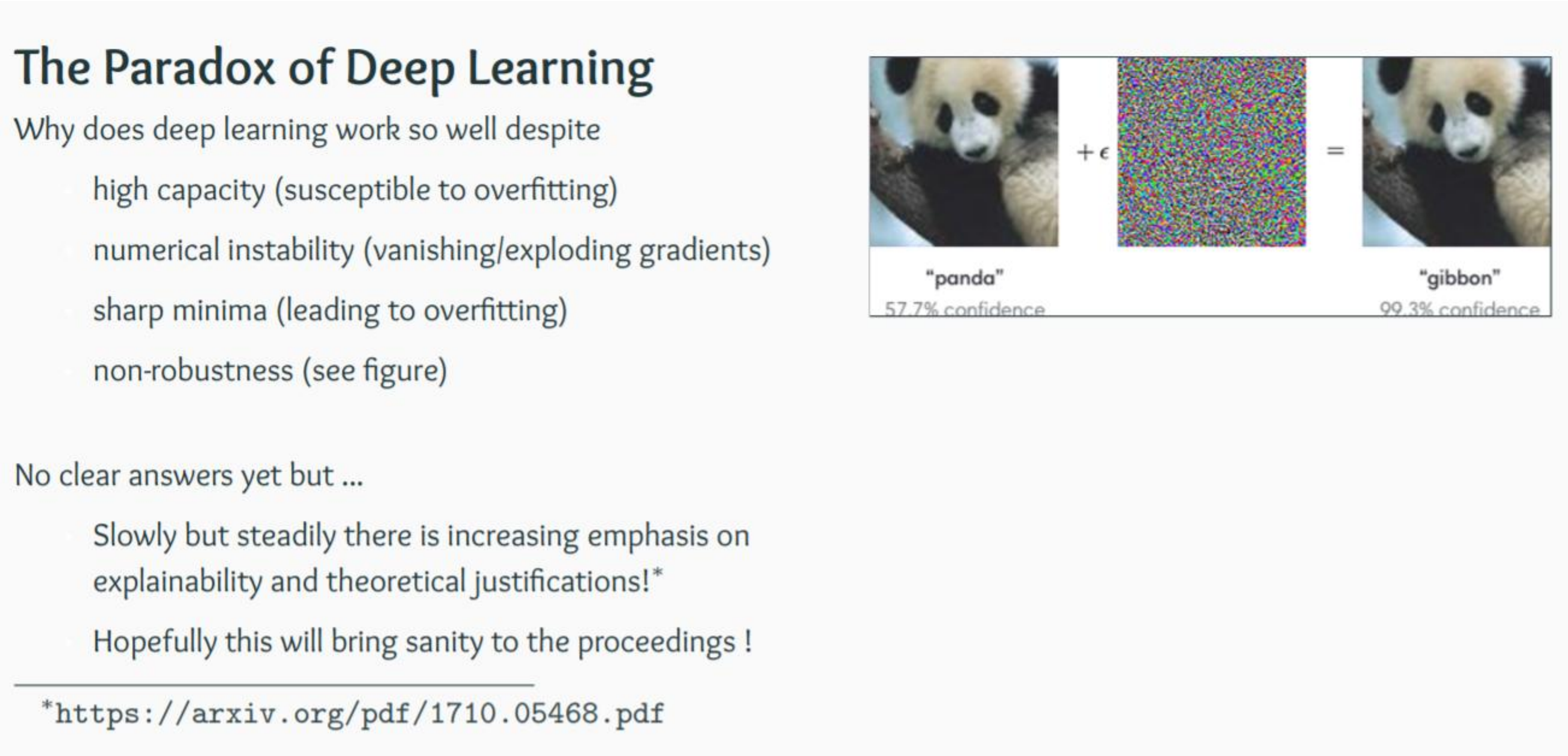
Using generative modeling, astrophysicists could investigate how galaxies change when they go from low-density regions of the cosmos to high-density regions, and what physical processes are responsible for these changes.

Adapted from K. Schawinski et al.; Source [illegible]

[a] https://deepmind.com/blog/article/AlphaFold-Using-AI-for-scientific-discovery
https://ocean.org/stories/spotting-seals-from-space
https://www.quantamagazine.org/how-artificial-intelligence-is-changing-science-20190311/

[MK]

Why get to the basics of ANNs?



The Paradox of Deep Learning

Why does deep learning work so well despite

- high capacity (susceptible to overfitting)
- numerical instability (vanishing/exploding gradients)
- sharp minima (leading to overfitting)
- non-robustness (see figure)

No clear answers yet but ...

- Slowly but steadily there is increasing emphasis on explainability and theoretical justifications!*
- Hopefully this will bring sanity to the proceedings !

*https://arxiv.org/pdf/1710.05468.pdf

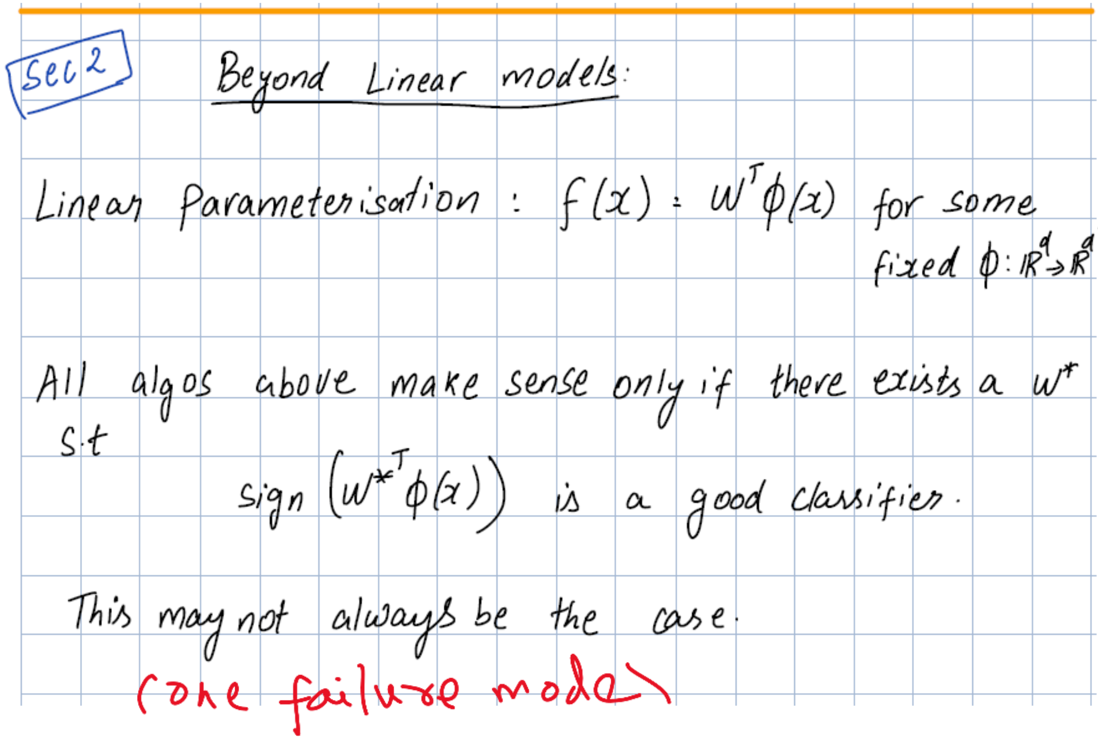
We would like to understand the key idea and algorithm that underlies almost all ANN/DL models.

[MK]

Outline for Module M7

- M7. Neural Networks
  - M7.0 Introduction/Motivation
  - **M7.1 Feed-forward neural networks**
    - **(Key Idea: Adaptive Basis functions and its Generalizations)**
    - (Network architecture, Network training & Backpropagation algo. mention (very brief))
  - M7.2 Concluding thoughts

Key Idea: Adaptive Basis fns (aka Feature/Representation Learning)



$\boxed{\text{Sec 2}}$ Beyond Linear models:

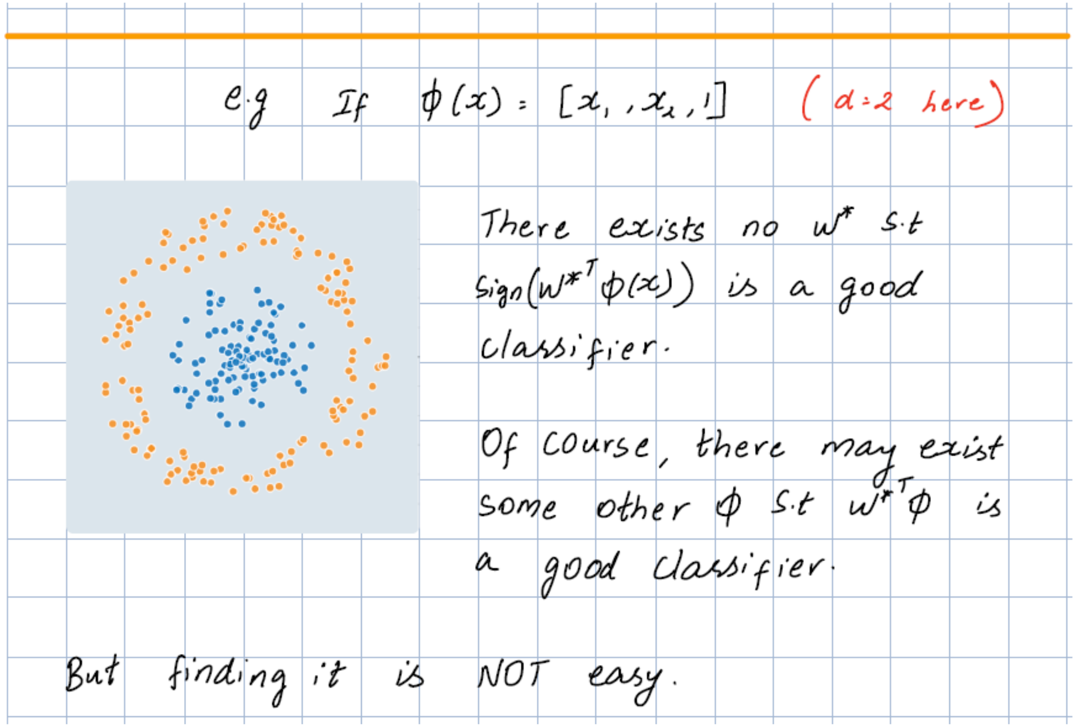
Linear Parameterisation : $f(x) = W^T\phi(x)$ for some fixed $\phi : \mathbb{R}^d \to \mathbb{R}^{d'}$

All algos above make sense only if there exists a w^* s.t

$$\text{sign}\left(w^{*T}\phi(x)\right) \text{ is a good classifier.}$$

This may not always be the case.

(one failure mode)



e.g If $\phi(x) = [x_1, x_2, 1]$ ($d=2$ here)

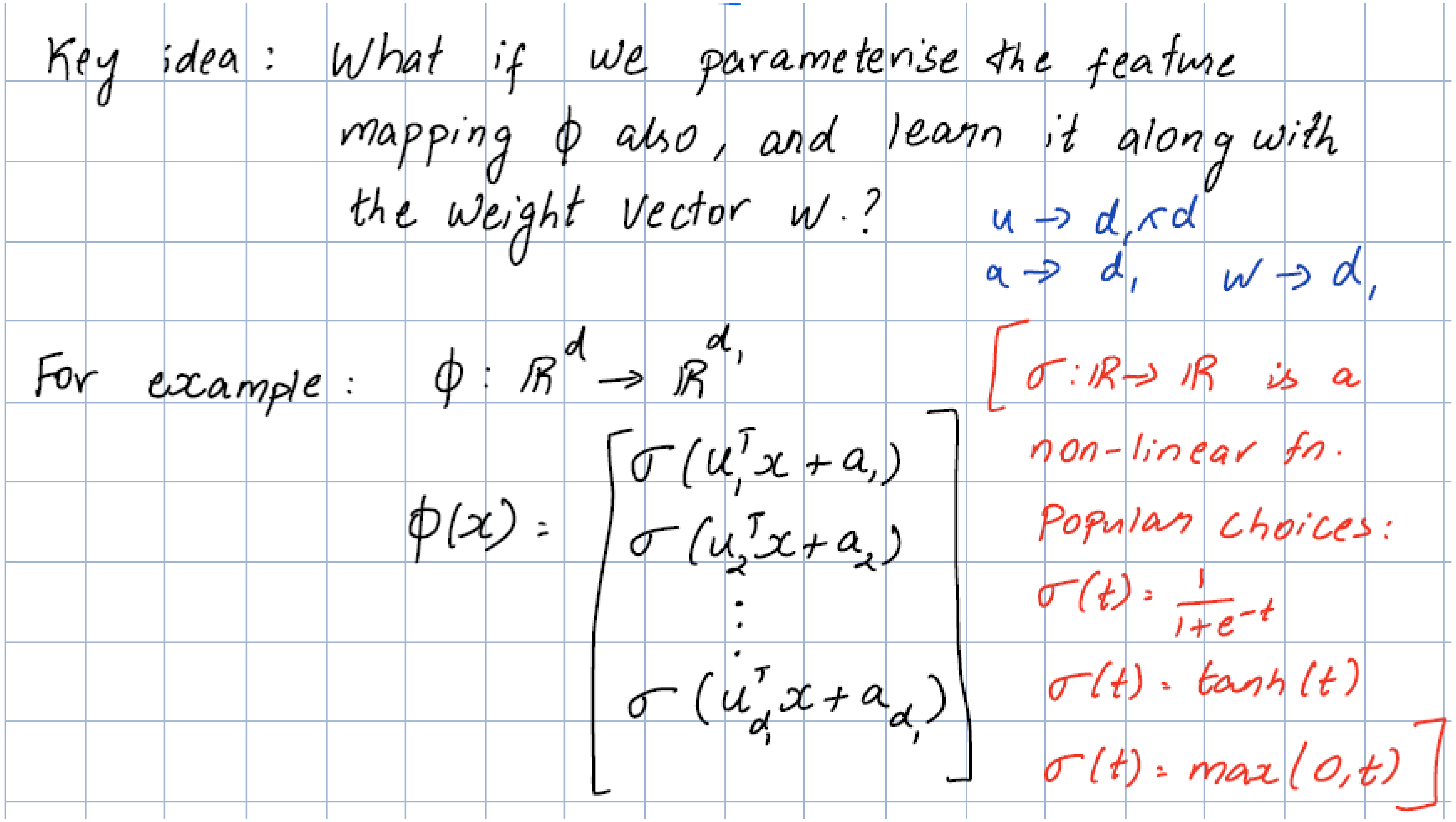
There exists no w^* s.t $\text{sign}(w^{*T}\phi(x))$ is a good classifier.

Of course, there may exist some other ϕ s.t $w^{*T}\phi$ is a good classifier.

But finding it is NOT easy.

[HR]

ANNs: Key Idea



Key idea: What if we parameterise the feature mapping ϕ also, and learn it along with the weight vector w.?

$u \to d_1 \times d$

$a \to d_1$ $\quad w \to d_1$

For example: $\phi : \mathbb{R}^d \to \mathbb{R}^{d_1}$

$$\phi(x) = \begin{bmatrix} \sigma(u_1^T x + a_1) \\ \sigma(u_2^T x + a_2) \\ \vdots \\ \sigma(u_{d_1}^T x + a_{d_1}) \end{bmatrix}$$

[$\sigma : \mathbb{R} \to \mathbb{R}$ is a non-linear fn.

Popular choices:

$\sigma(t) = \frac{1}{1+e^{-t}}$

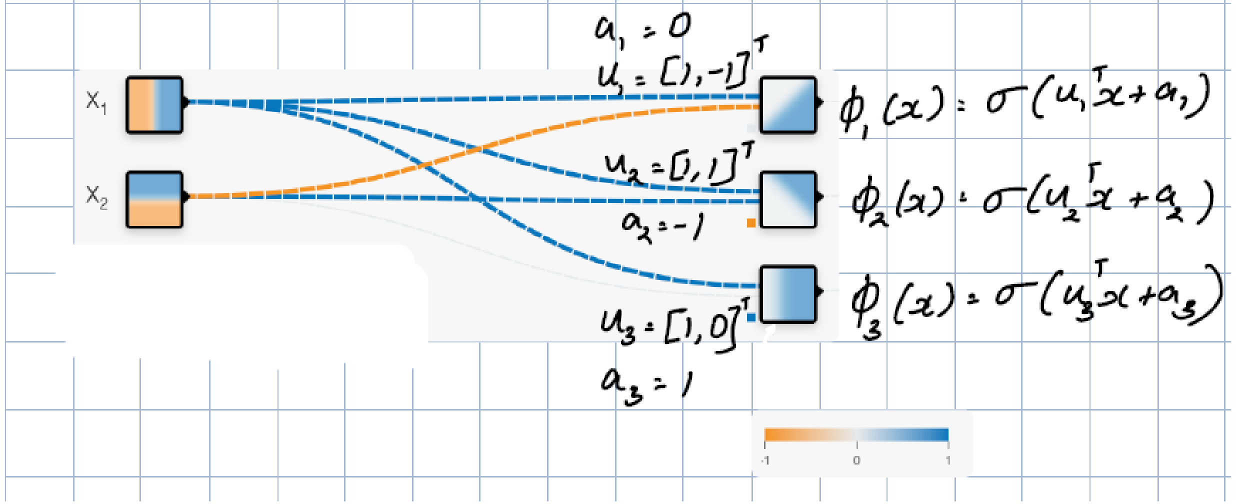
$\sigma(t) = \tanh(t)$

$\sigma(t) = \max(0,t)$]

[HR]

ANNs: Key Idea (example – specific u,a,w)

How does ϕ look like for some $u, a.?$



$a_1 = 0$

$u_1 = [1,-1]^T$

$\phi_1(x) = \sigma(u_1^T x + a_1)$

$u_2 = [1,1]^T$

$\phi_2(x) = \sigma(u_2^T x + a_2)$

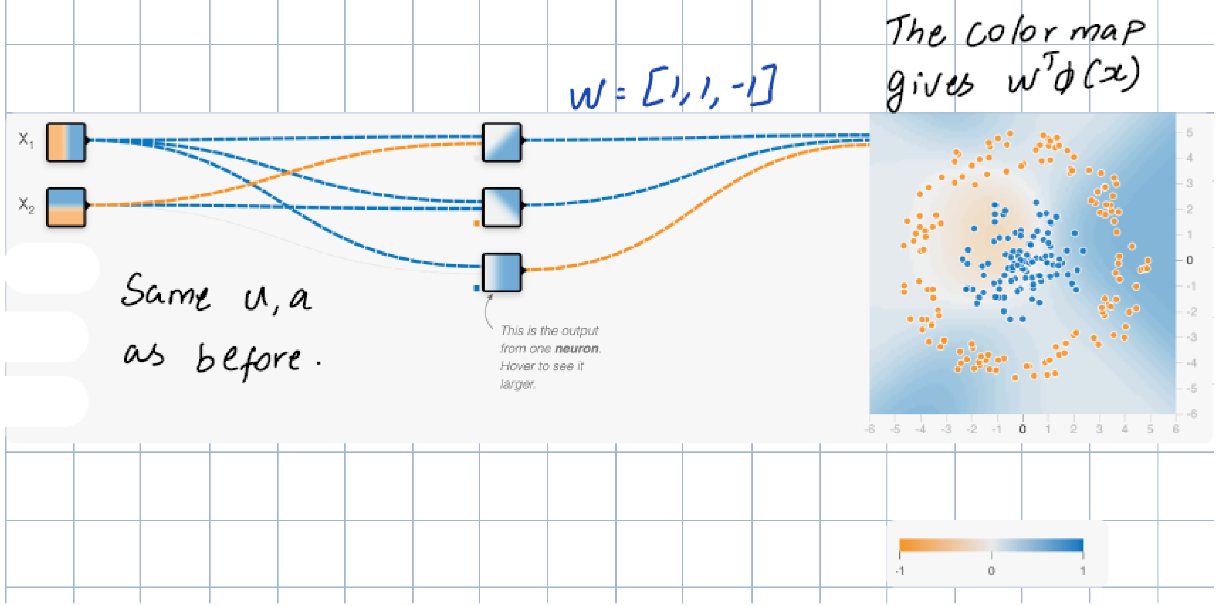
$a_2 = -1$

$\phi_3(x) = \sigma(u_3^T x + a_3)$

$u_3 = [1,0]^T$

$a_3 = 1$

How does $w^T \phi(x)$ look like for some u,a,w?



$w = [1,1,-1]$

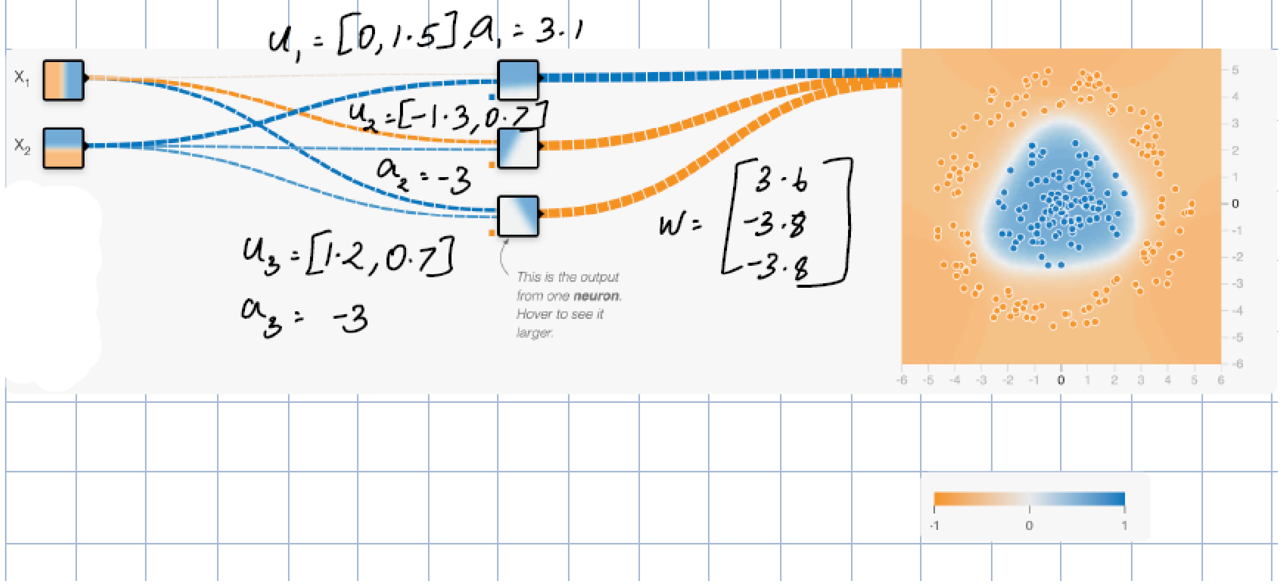
The color map gives $w^T \phi(x)$

Same u, a as before.

https://playground.tensorflow.org/ [HR]

ANNs: Key Idea (example – good u,a,w)

What is a "good" u, a, w for above data?



$u_1 = [0, 1.5], a_1 = 3.1$

$u_2 = [-1.3, 0.7]$

$a_2 = -3$

$u_3 = [1.2, 0.7]$

$a_3 = -3$

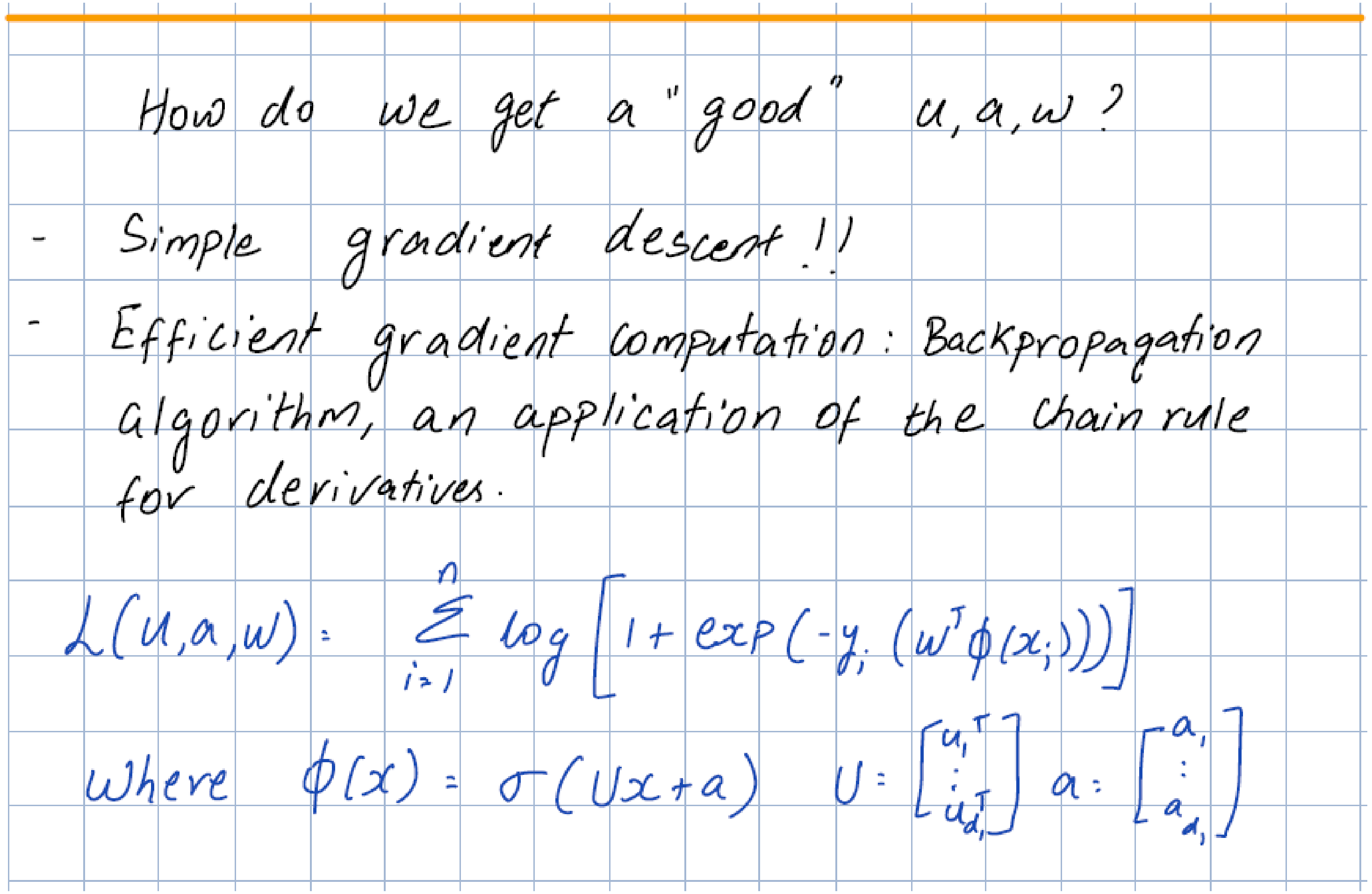
$$W = \begin{bmatrix} 3.6 \\ -3.8 \\ -3.8 \end{bmatrix}$$

This is the output from one **neuron**. Hover to see it larger.

Exercise: Construct the simplest ANN classifer you can to compute the XOR function.

https://playground.tensorflow.org/ [HR]

Network training (getting a good u,a,w) helps us realize the key idea of feature/repn. learning!



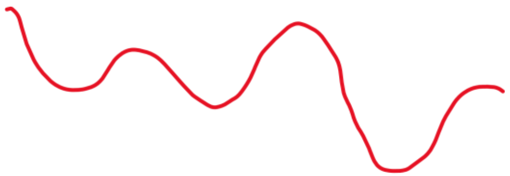
How do we get a "good" u, a, w ?

- Simple gradient descent !!
- Efficient gradient computation: Backpropagation algorithm, an application of the chain rule for derivatives.

$$\mathcal{L}(u,a,w) = \sum_{i=1}^{n} \log\left[1+\exp(-y_i\,(w^T\phi(x_i)))\right]$$

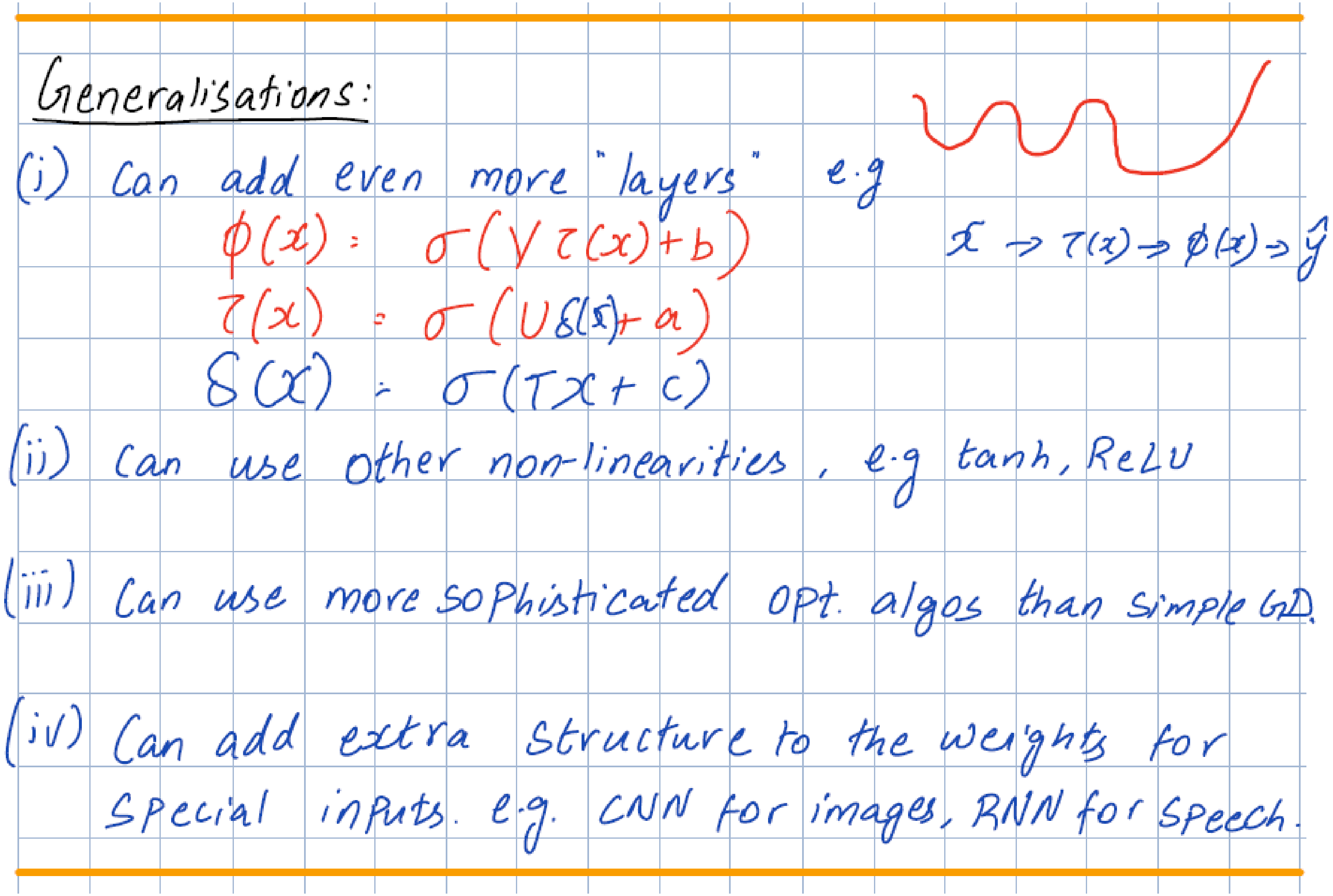
where $\phi(x) = \sigma(Ux+a)$ $U = \begin{bmatrix} u_1^T \\ \vdots \\ u_{d_1}^T \end{bmatrix}$ $a = \begin{bmatrix} a_1 \\ \vdots \\ a_{d_1} \end{bmatrix}$

(two failure modes)



[HR]

Generalizations of the key idea



Generalisations:

(i) Can add even more "layers" e.g

$$\phi(x) = \sigma(V\tau(x)+b)$$

$$\tau(x) = \sigma(U\delta(x)+a)$$

$$\delta(x) = \sigma(Tx+c)$$

$$x \to \tau(x) \to \phi(x) \to \hat{y}$$

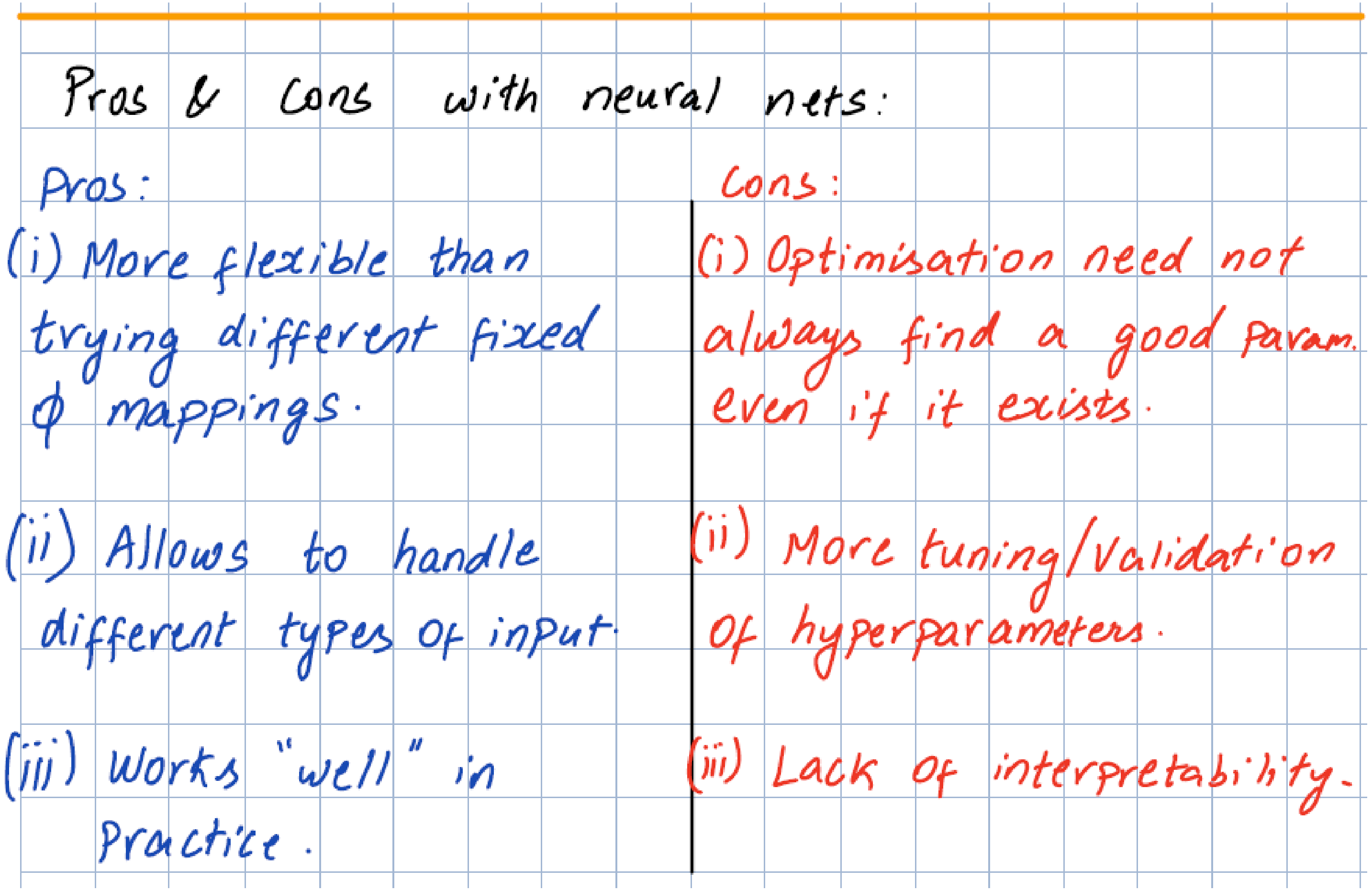
(ii) Can use other non-linearities, e.g tanh, ReLU

(iii) Can use more sophisticated opt. algos than simple GD.

(iv) Can add extra structure to the weights for special inputs. e.g. CNN for images, RNN for speech.

[HR]

Pros/cons of these ANN (generalizations)



Pros & Cons with neural nets:

Pros:	Cons:
(i) More flexible than trying different fixed ϕ mappings.	(i) Optimisation need not always find a good param. even if it exists.
(ii) Allows to handle different types of input.	(ii) More tuning/Validation of hyperparameters.
(iii) Works "well" in Practice.	(iii) Lack of interpretability.

[HR]

Outline for Module M7

- M7. Neural Networks
  - M7.0 Introduction/Motivation
  - **M7.1 Feed-forward neural networks**
    - (Key Idea: Adaptive Basis functions and its Generalizations)
    - **(Network architecture, Network training & Backpropagation algo. sketch)**
  - M7.3 Concluding thoughts

FFNs for regression and classification

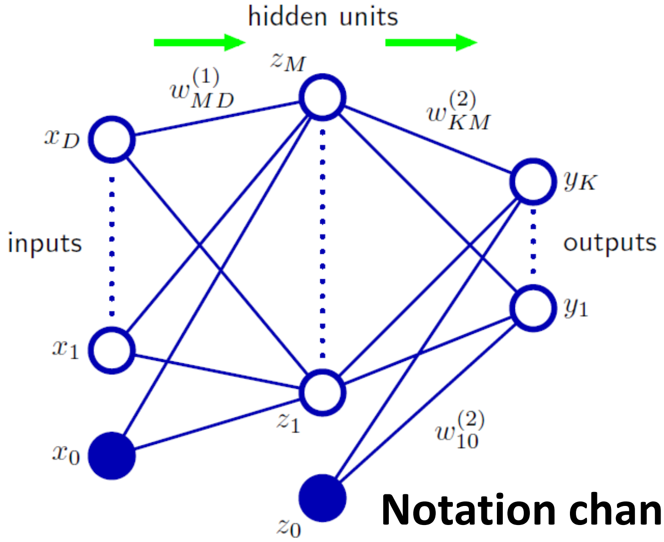
- FFN typically taught as a single-output model, but K-outputs model is a straight-forward extension.
    - Learning K models together permits sharing of weights of earlier layers of a neural network across all K outputs!

- Some notations:

  Predictions/outputs are: $\{y_k\}_{k=1 \; to \; K}$

  Training data: $\left\{\left(x^{(n)}, t^{(n)}\right)\right\}_{n=1 \; to \; N}$, with $x_n \in R^D$;

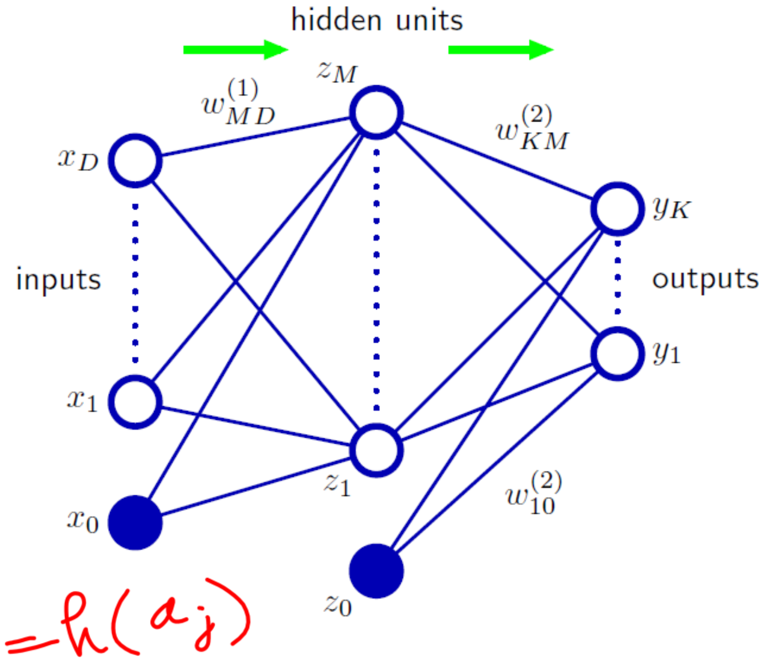
    - and $t_n \in R^K$ for **K regression** problems,
    - or $\mathrm{t_n} \in \{-1, +1\}^K$ for **K two-class** problems,
    - or $\mathrm{t_n} \in \{one.hot.encodings\}$ for **one multi(K)-class** problem.



Notation change from [HR]→[CMB]:
$\boldsymbol{\phi}$ **to** $\boldsymbol{z}$
1 to K outputs

- Note: FFN aka Multilayer Perceptron (MLP) -- but a misnomer as it's not multiple layers of **perceptrons** (which are non-continuous, **non-differentiable step functions**), but multiple layers of **continuous non-linear functions like logistic functions**!

Feed-forward Neural Network (FFN) Architecture



$z_j = h(a_j)$

$a_j = \sum_{i=0}^{D} x_i w_{ji}$

eg. $h(t) = \frac{1}{1+e^{-t}}$

$a_k = \sum_{i=0}^{M} w_{ki} z_i$

$y_k = \underline{\underline{f_k}}(a_k)$

??

[CMB]

Output layer - popular choices

Problem	**Activation fn. for output**	**Loss fn.** $\boldsymbol{L = \sum_n L_n}$ (aka error E $= \sum_n \; E_n$ in ANN lingo)
(1) Regression (K regression problems)	$y_k = a_k$	(squared loss): $L_n = \sum_k \frac{1}{2}\left(y_k^{(n)} - t_k^{(n)}\right)^2$
(2) Classification (K two-class problems)	$y_k = \sigma(a_k) = \dfrac{1}{1+\exp(-a_k)}$	(logistic loss): $L_n = \sum_k \log(1+\exp\left(-t_k^{(n)} a_k^{(n)}\right))$ $(\text{==cross entropy loss}): L_n = -\sum_k \;\; (t_k^{(n)} \log y_k^{(n)} + \left(1-t_k^{(n)}\right)\log\left(1-y_k^{(n)}\right))$
(3) Classification (one K-class problem; K > 2)	$y_k = \text{softmax}(a_1, \ldots, a_K)$ $= \dfrac{\exp(a_k)}{\sum_{k'} \exp(a_{k'})}$	(cross entropy loss): $L_n = -\sum_k t_k^{(n)} \log y_k^{(n)}$

$$a_K = \sum_{i=0}^{M} w_{Ki} z_i$$

MLE of a discriminative model is the same as minimizing the error/loss fn. above!

Lingo/notation change: NLL -> Loss -> Error

[CMB]

Backprop begin (optional)

Exercise: Derivative for output layer

- Compute derivative $\delta_k \coloneqq \frac{\partial L_n}{\partial a_k} \coloneqq \frac{\partial E_n}{\partial a_k}$ using chain rule for problems in previous page.

- Chain rule for:
  - Problems (1),(2): $\frac{\partial E_n}{\partial a_k} = \frac{\partial E_n}{\partial y_k^{(n)}} \frac{\partial y_k^{(n)}}{\partial a_k}$
  - Problem (3): $\frac{\partial E_n}{\partial a_k} = \sum_{k\prime} \frac{\partial E_n}{\partial y_{k\prime}^{(n)}} \frac{\partial y_{k\prime}^{(n)}}{\partial a_k}$

Eg: for Problem (1)

$$\delta_k = y_k - t_k$$

- Why are we interested in above?

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i.$$

$a_j = \sum_i w_{ji} z_i$

[CMB]

Numerical differentiation wrt all weights

- If W is the total # of params., then below takes $O(W)$ time for each w_{ji}, and hence $O(W^2)$ for all params.

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n(w_{ji} + \epsilon) - E_n(w_{ji})}{\epsilon} + O(\epsilon)$$

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n(w_{ji} + \epsilon) - E_n(w_{ji} - \epsilon)}{2\epsilon} + O(\epsilon^2).$$

- Backprop. is simply chain rule realized via an $O(W)$ Dynamic Programming (DP) algo.
    - essentially stores already computed derivatives wrt weights in later layers of the network to avoid redundant computations when computing derivatives wrt weights in earlier layers

[CMB]

Backpropagation algorithm

Error Backpropagation

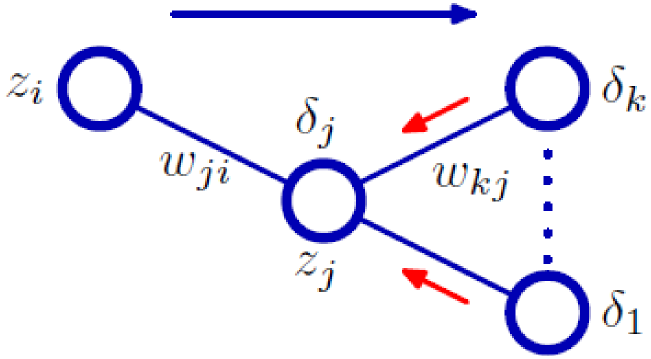
1. Apply an input vector $\mathbf{x}_n$ to the network and forward propagate through the network using (5.48) and (5.49) to find the activations of all the hidden and output units.
2. Evaluate the δ_k for all the output units using (5.54).
3. Backpropagate the δ's using (5.56) to obtain δ_j for each hidden unit in the network.
4. Use (5.53) to evaluate the required derivatives.

$$a_j = \sum_i w_{ji} z_i \tag{5.48}$$

$$z_j = h(a_j). \tag{5.49}$$

$$\delta_k = y_k - t_k \tag{5.54}$$

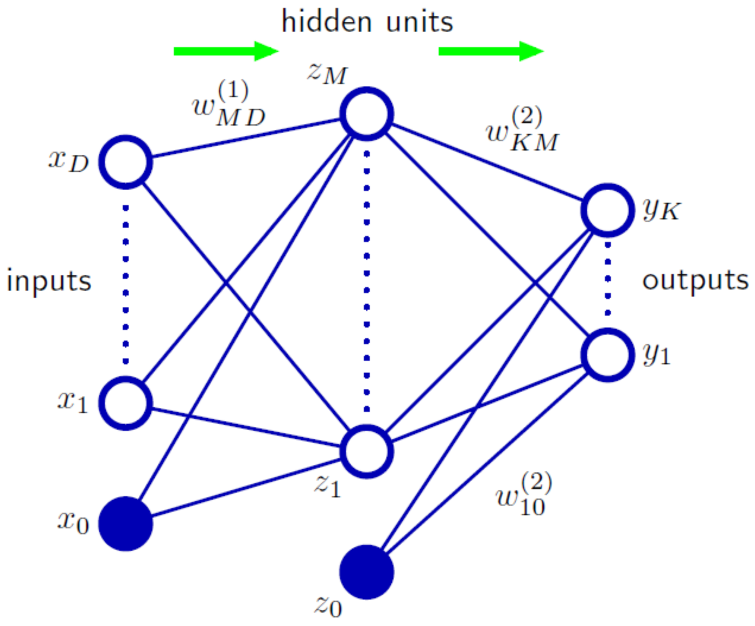
$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i. \tag{5.53}$$



$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} = h'(a_j) \sum_k w_{kj} \delta_k \tag{5.56}$$

[CMB]

Exercise: When tanh(a) is the hidden unit activation fn, show that backprop. reduces to:



Next we compute the δ's for each output unit using

$$\delta_k = y_k - t_k. \tag{5.65}$$

Then we backpropagate these to obtain δs for the hidden units using

$$\delta_j = (1 - z_j^2) \sum_{k=1}^{K} w_{kj}\delta_k. \tag{5.66}$$

Finally, the derivatives with respect to the first-layer and second-layer weights are given by

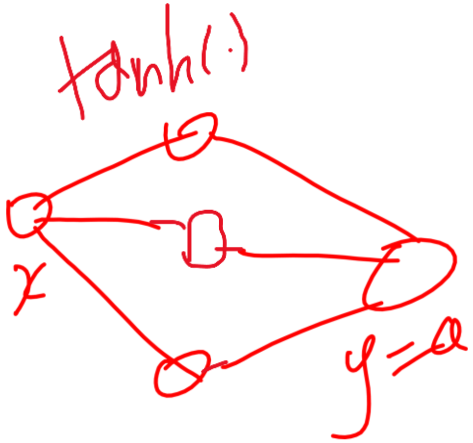
$$\frac{\partial E_n}{\partial w^{(1)}_{ji}} = \delta_j x_i, \qquad \frac{\partial E_n}{\partial w^{(2)}_{kj}} = \delta_k z_j. \tag{5.67}$$

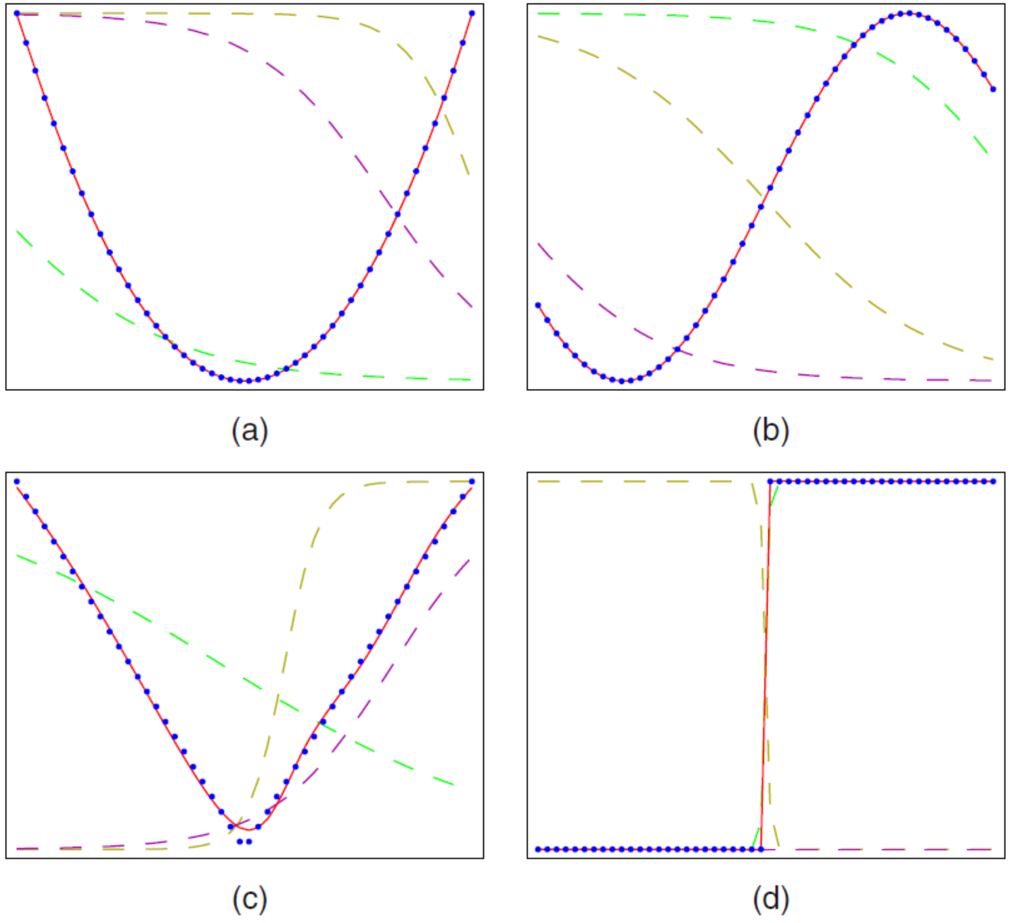
$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}.$$

[CMB]

Backprop end (optional)

What can you achieve with backprop + grad. descent? Example ANN for Regression



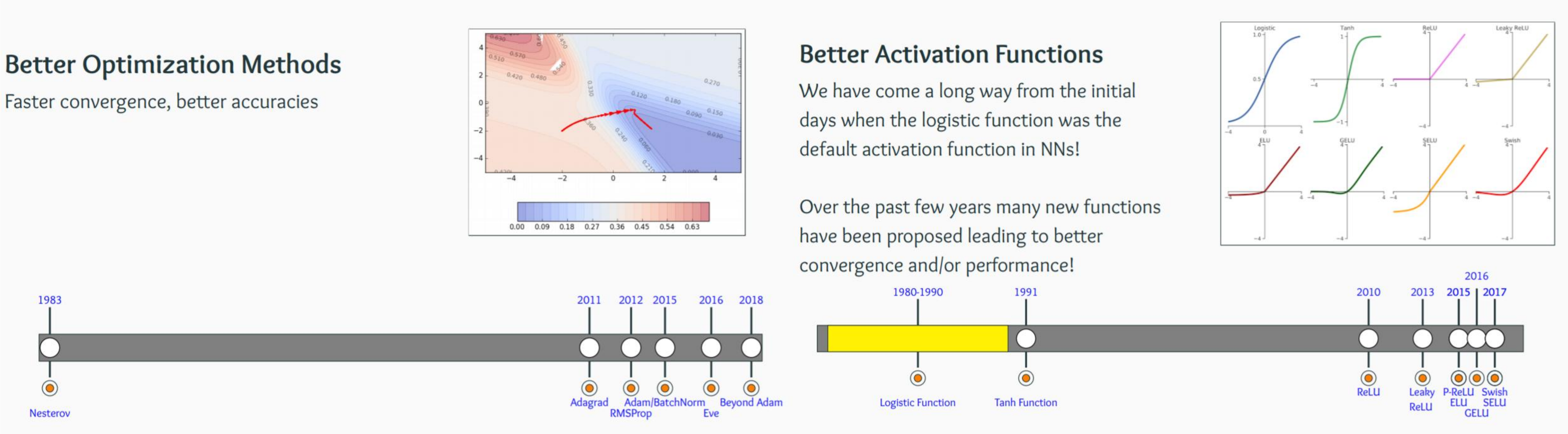

(a) (b)
(c) (d)

[CMB]

Outline for Module M7

- M7. Neural Networks
  - M7.0 Introduction/Motivation
  - M7.1 Feed-forward neural networks
    - (Key Idea: Adaptive Basis functions and its Generalizations)
    - (Network architecture, Network training & Backpropagation algo. sketch)
  - **M7.2 Concluding thoughts**

If you are interested to learn more…



Better Optimization Methods

Faster convergence, better accuracies

Better Activation Functions

We have come a long way from the initial days when the logistic function was the default activation function in NNs!

Over the past few years many new functions have been proposed leading to better convergence and/or performance!

Regularization of network (weights tying/sharing, etc.)

Calls for Sanity (Interpretable, Fair, Responsible, Green AI)

[MK]

Concluding thoughts & next steps

Recall: From linear to non-linear regression/classifn. models:

Non-linear method	**(Non-linear) Basis functions**	**Objective function / OP**
Vanilla extn. of linear models	**Fixed** – non-linear basis fns (feature map, $\phi: \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$) fixed before seeing training data (manually via feature engineering); only weights of these basis fns learnt using training data.	Convex (unconstrained opt.)
SVM	**Selective** – center basis fns on training data points (dual/kernel view) and use training data to learn their weights and select a subset of them (non-zero weight support vectors) for eventual predictions.	Convex (constrained opt.)
Neural networks	**Adaptive** – Fix # of basis fns in advance, but allow them to be adaptive; parameterize basis fns and learn these parameters using training data.	Non-convex

<u>Next steps:</u> (Conditional) Mixture Models (including Mixture Density Network MDN) is one framework to probabilistically (soft-ly) combine different regression models (aka conditional mixtures), but there are also other complementary combined models / ensemble methods!

[CMB]

Thank you!