M9. (Artificial) Neural Networks (ANNs)

Balaraman Ravindran

PRML Aug-Dec 2021 (BR section)

Acknowledgment of Sources

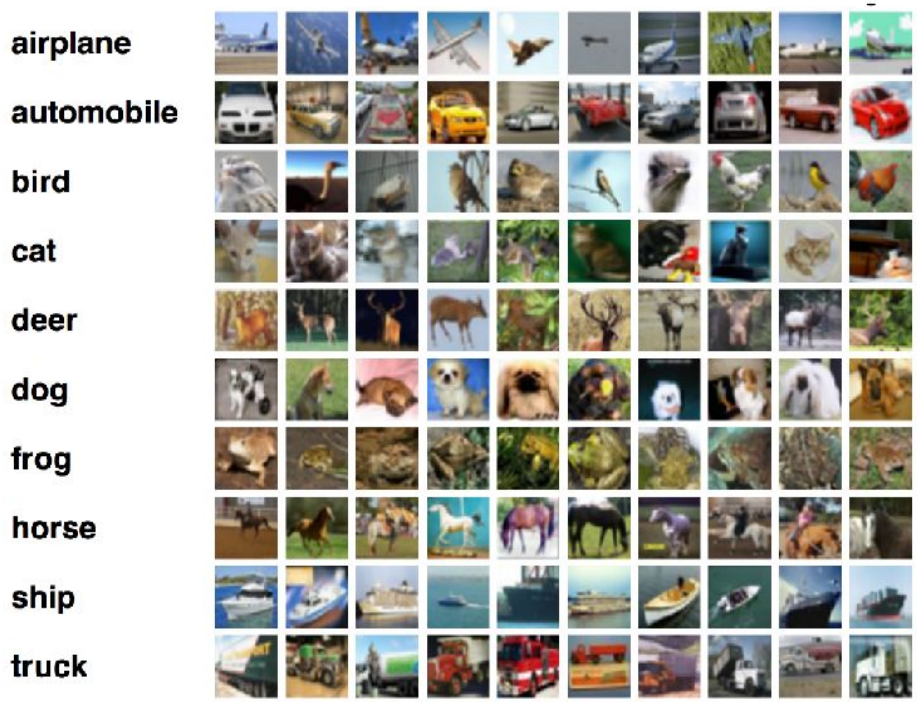
- Slides based on content from related
  - Courses:
    - IITM – Profs. Arun/Harish/Chandra's PRML offerings (slides, quizzes, notes, etc.), Prof. Ravi's "Intro to ML" slides – cited respectively as [AR], [HR], [CC], [BR] in the bottom right of a slide.
    - India – NPTEL PR course by IISc Prof. PS. 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 M9

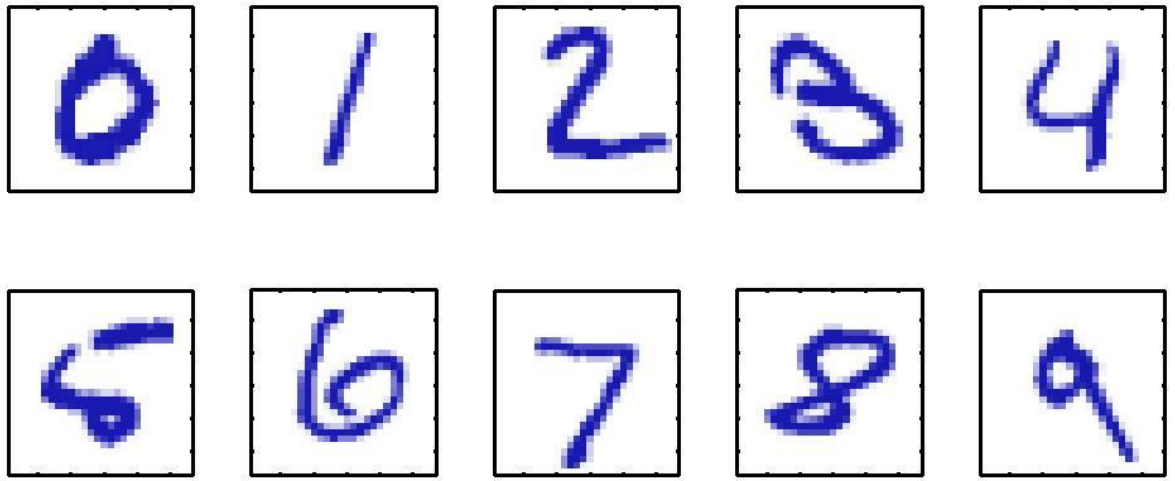
- M9. Neural Networks
  - **M9.0 Introduction/Motivation**
  - M9.1 Feed-forward neural networks
    - (Key Idea: Adaptive Basis functions and its Generalizations)
    - (Network architecture, Network training & Backpropagation algo. sketch)
  - M9.2 Mixture Density Networks
  - M9.3 Concluding thoughts

Recall: popular examples

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

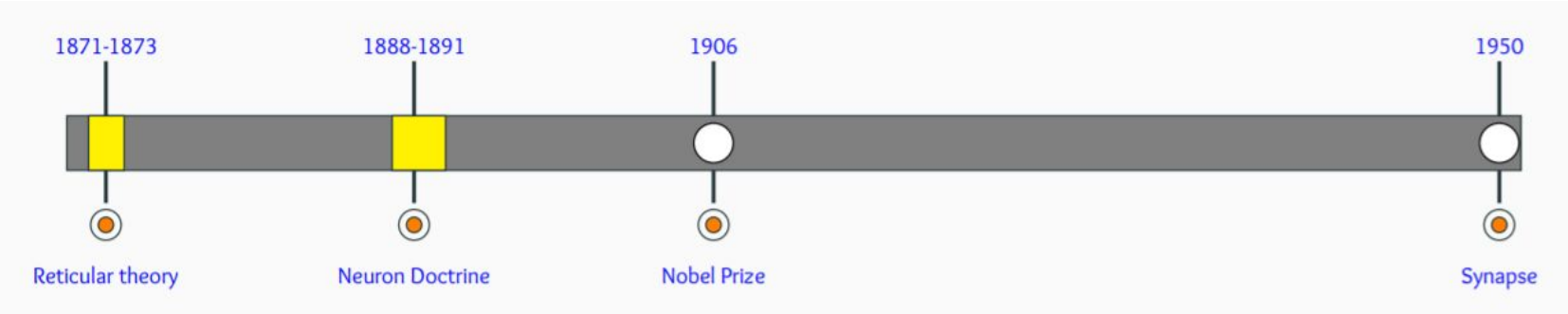


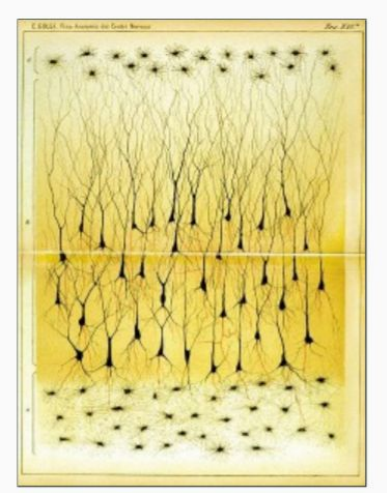
Handwritten Digit Recognition

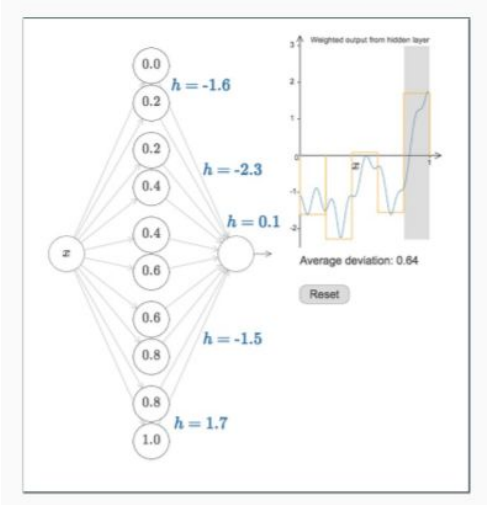


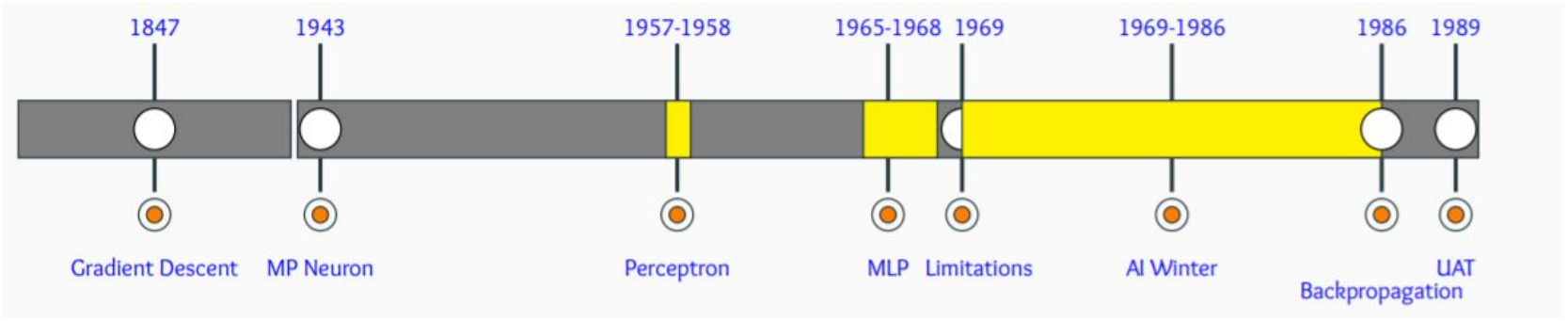
[HR]

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







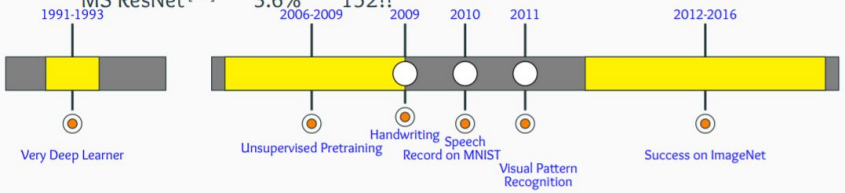


[From MK: https://www.cse.iitm.ac.in/~miteshk/CS6910/Slides/Lecture1.pdf]

(Recent) Success stories of ANNs/DL

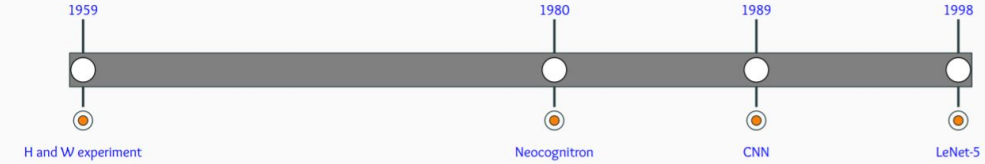
Winning more visual recognition challenges

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!!

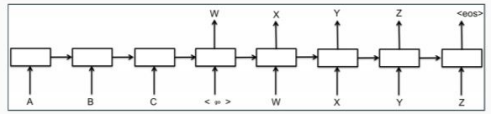


LeNet-5

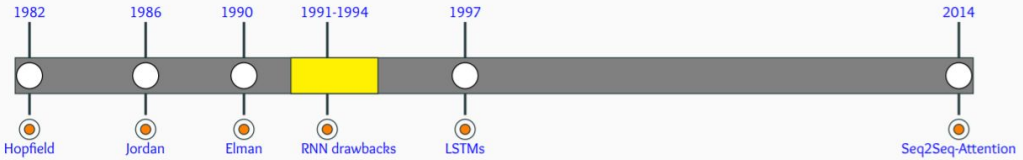
Introduced the (now famous) MNIST dataset (LeCun et. al.) [21]



Sequence To Sequence Models



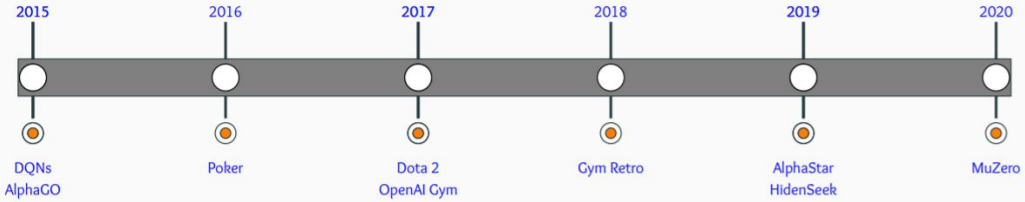
- Initial success in using RNNs/LSTMs for large scale Sequence To Sequence Learning Problems
- Introduction of Attention which is perhaps the idea of the decade!



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

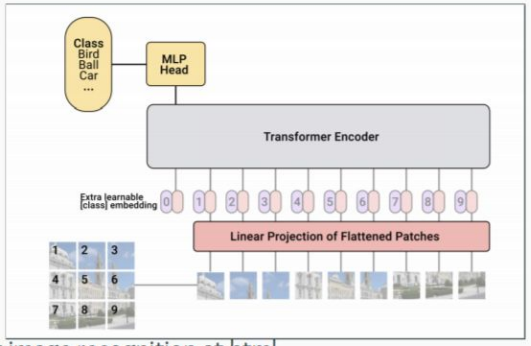


[MK]

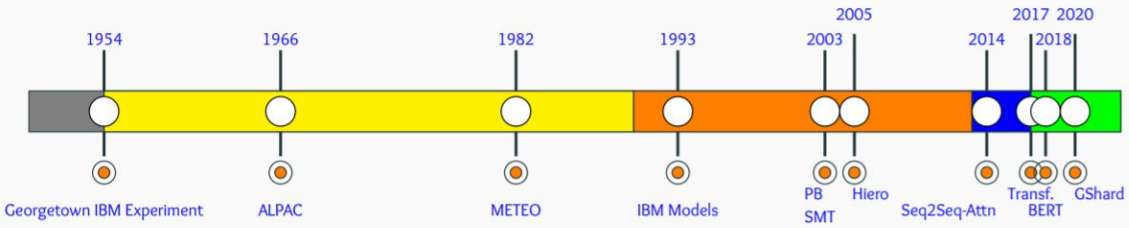
More (recent) success stories of ANNs/DL

From Language To Vision

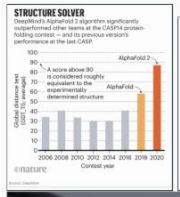
A vision model[a] based as closely as possible on the Transformer architecture originally designed for text-based tasks (another paradigm shift from CNNs which have been around since 1980s!)

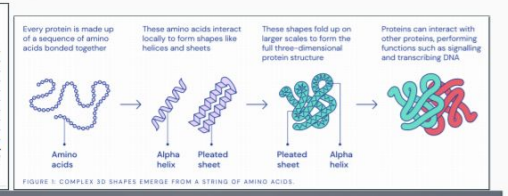


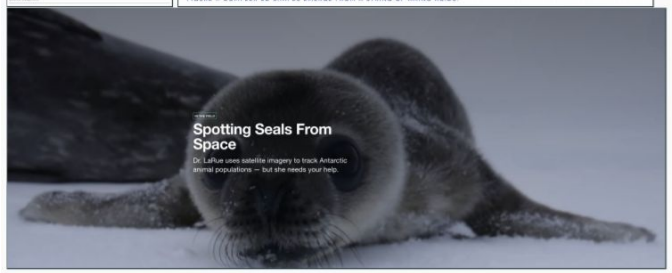
[a]Source:https://ai.googleblog.com/2020/12/transformers-for-image-recognition-at.html

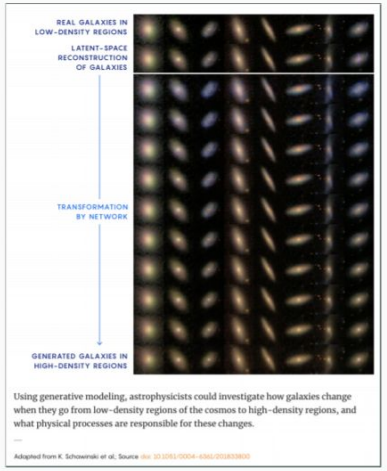


Accelerating Scientific Discovery[a]









[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/

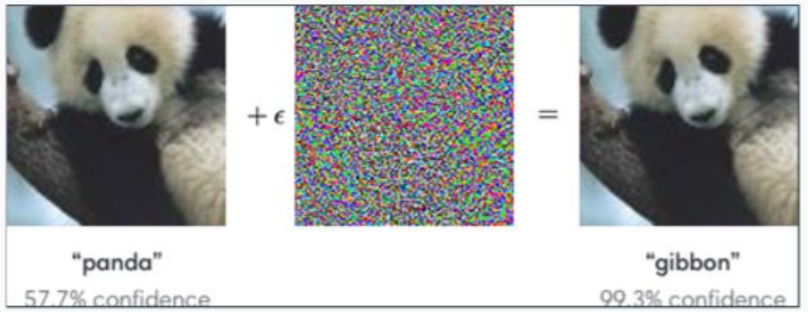
[MK]

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 M9

- M9. Neural Networks
  - M9.0 Introduction/Motivation
  - **M9.1 Feed-forward neural networks**
    - **(Key Idea: Adaptive Basis functions and its Generalizations)**
    - (Network architecture, Network training & Backpropagation algo. sketch)
  - M9.2 Mixture Density Networks
  - M9.3 Concluding thoughts

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

Sec 2 Beyond Linear models:

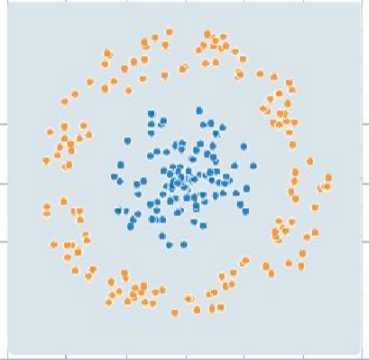
Linear Parameterisation : $f(x) = w^T \phi(x)$ for some fixed $\phi : \mathbb{R}^d \to \mathbb{R}^{d'}$

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

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

This may not always be the case.

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:

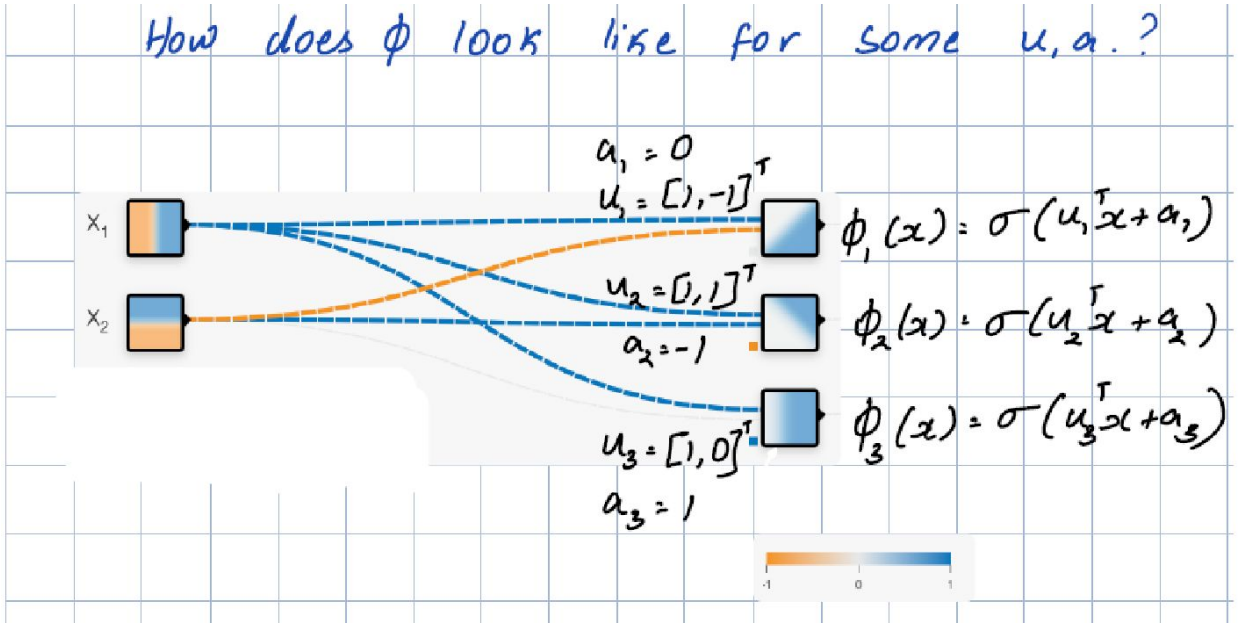
$\sigma(t) = \frac{1}{1+e^{-t}}$

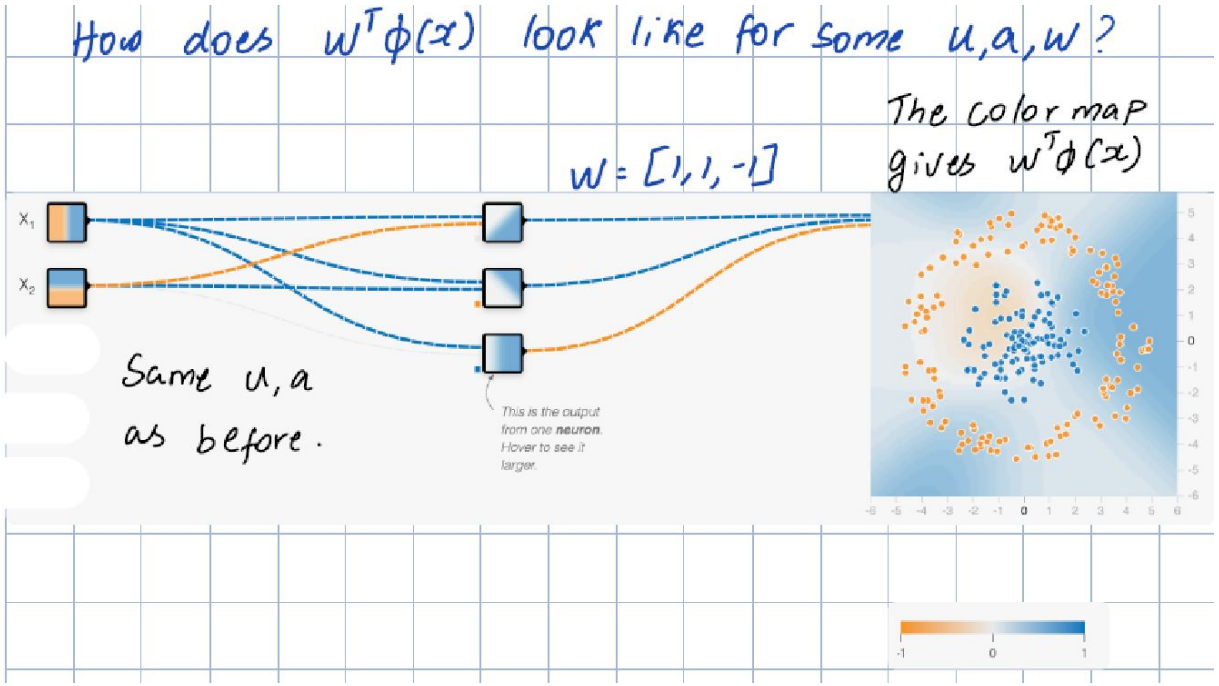
$\sigma(t) = \tanh(t)$

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

[HR]

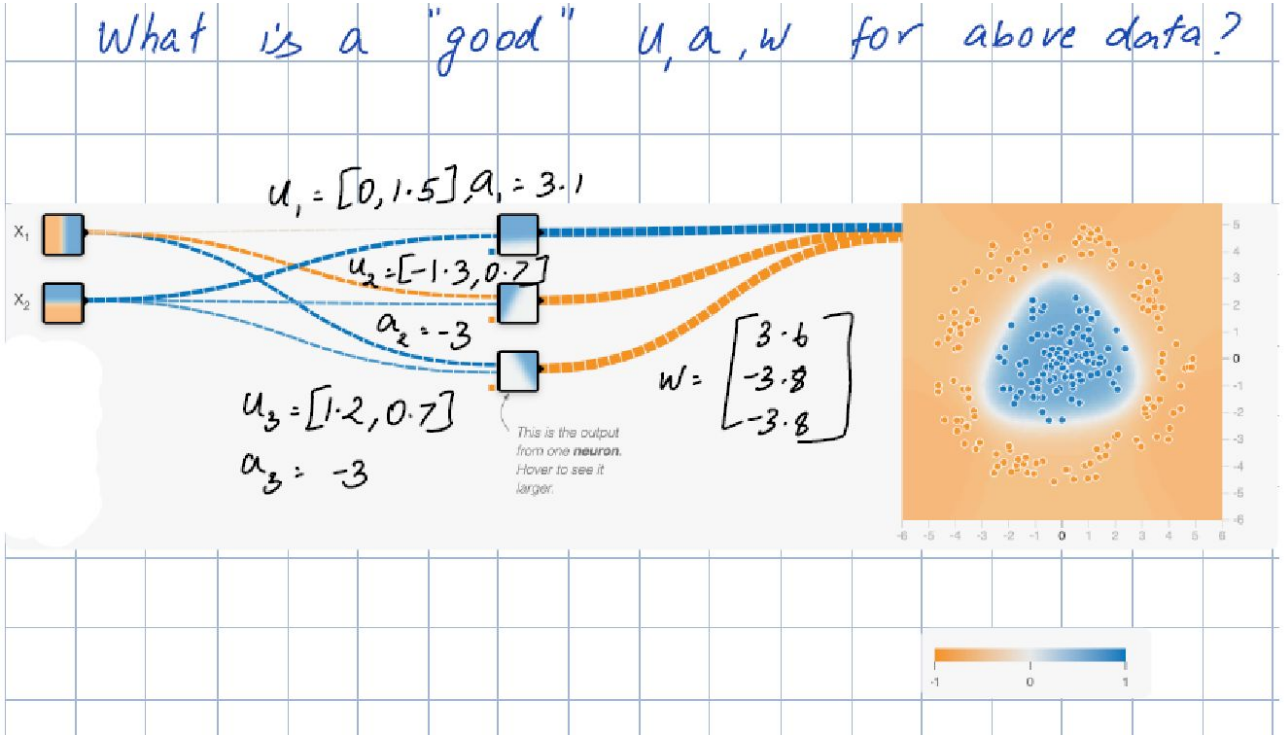
ANNs: Key Idea (example – specific u,a,w)





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

ANNs: Key Idea (example – good u,a,w)



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.

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

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

[HR]

Generalizations of the key idea

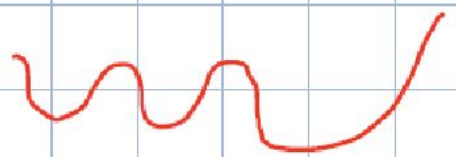
Generalisations:

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

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

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

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



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

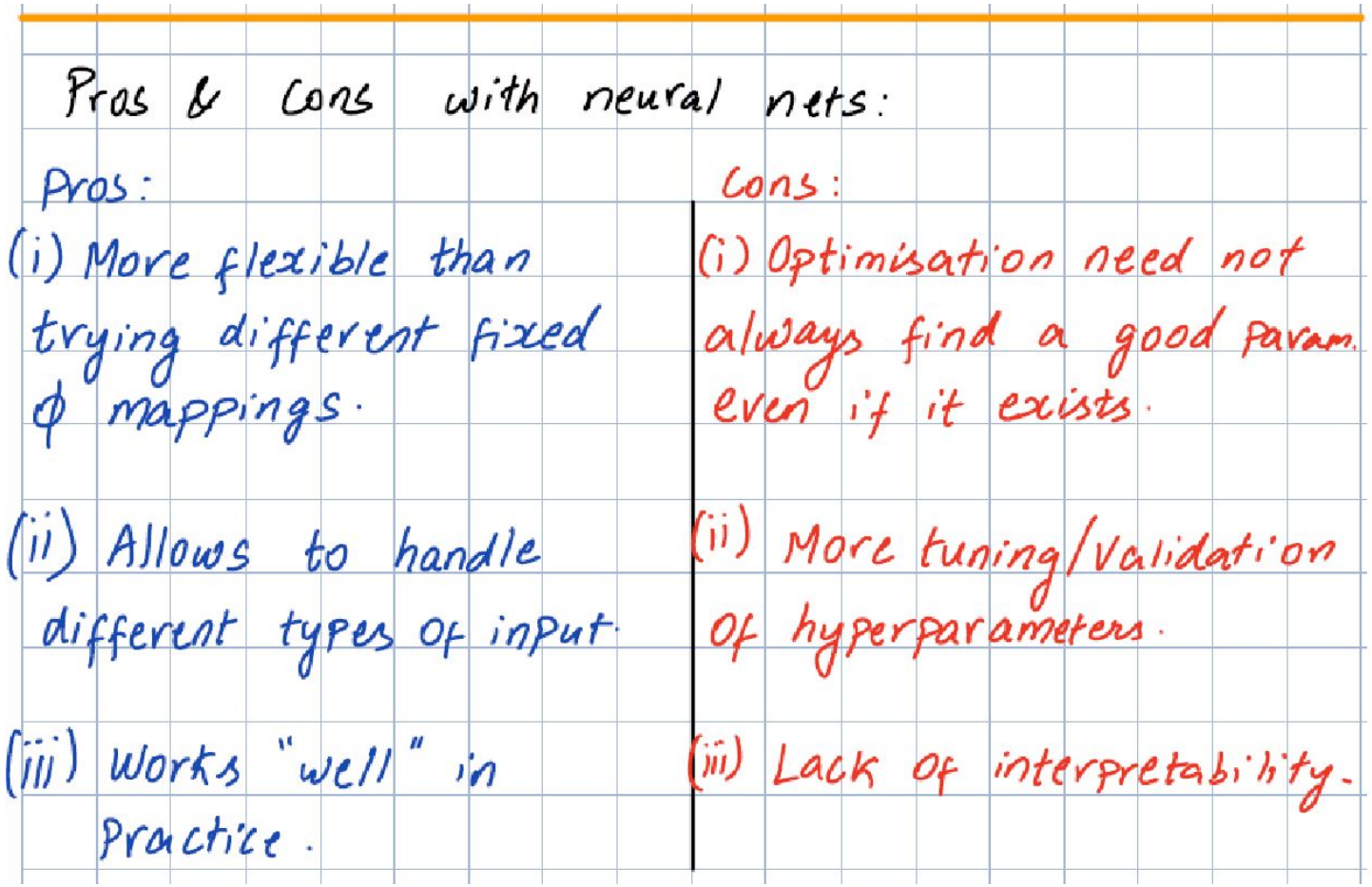
(ii) Can use other non-linearities, e.g tanh, ReLU

(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)

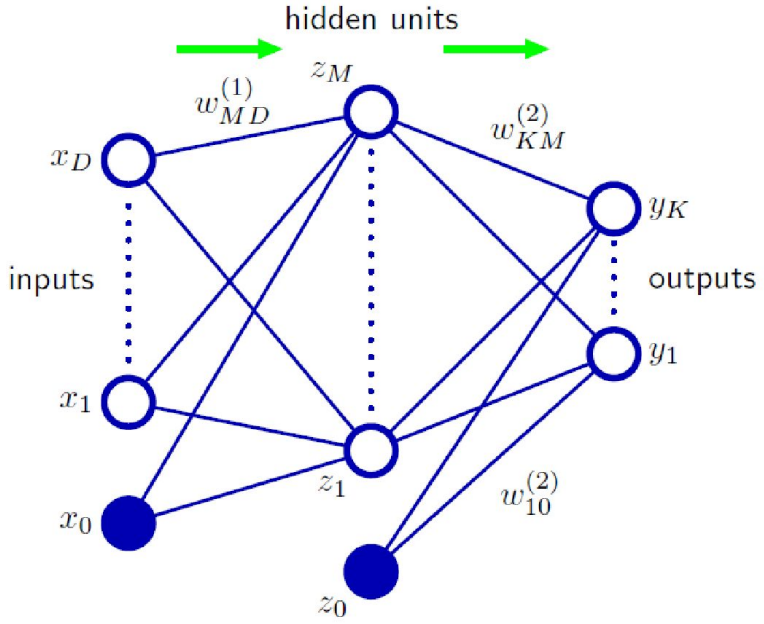


[HR]

Outline for Module M9

- M9. Neural Networks
  - M9.0 Introduction/Motivation
  - **M9.1 Feed-forward neural networks**
    - (Key Idea: Adaptive Basis functions and its Generalizations)
    - **(Network architecture, Network training & Backpropagation algo. sketch)**
  - M9.2 Mixture Density Networks
  - M9.3 Concluding thoughts

Feed-forward Neural Network (FFN)



Note change in notations:

Training data: $\{(x^{(n)}, t^{(n)})\}_{n=1\ to\ N'}$ with $x_n \in R^d$;

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

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

Learning K models together permits sharing of weights of earlier layers across all K outputs!

Note FFNs aka Multilayer Perceptron -- 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! [CMB]

Output layer - popular choices

Problem	**Activation fn. for output**	
(1) Regression (K regression problems)		
(2) Classification (K two-class problems)		
(3) Classification (one K-class problem; K > 2)		

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

[CMB]

Exercise: Derivative for output layer

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

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

- Why are we interested in above?

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

[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

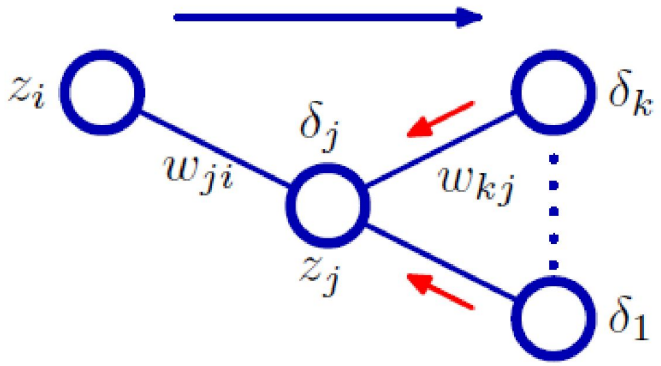
1. Apply an input vector $\mathbf{x}_n$ to the network and forward propagate through the network using (5.48) and (5.49) to find the activations of all the hidden and output units.
2. Evaluate the δ_k for all the output units using (5.54).
3. Backpropagate the δ's using (5.56) to obtain δ_j for each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

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

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

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

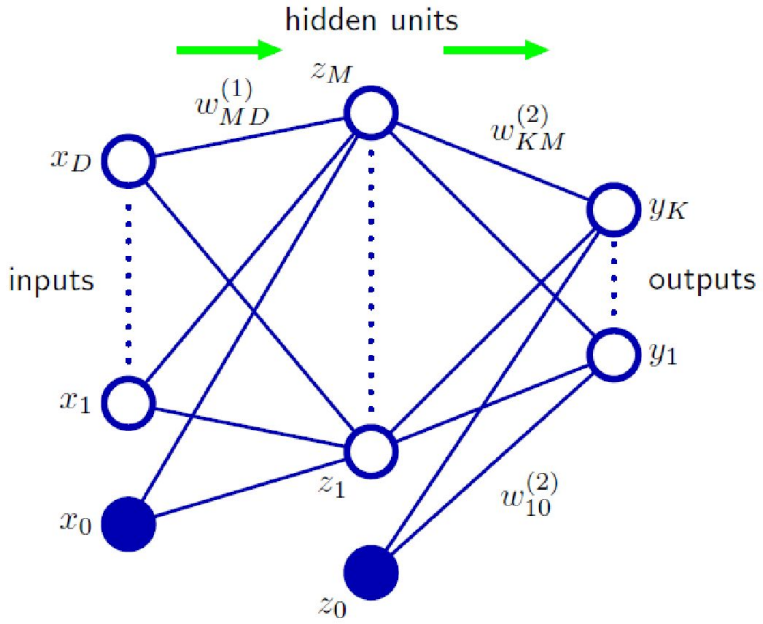
$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i. \tag{5.53}$$



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

[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

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

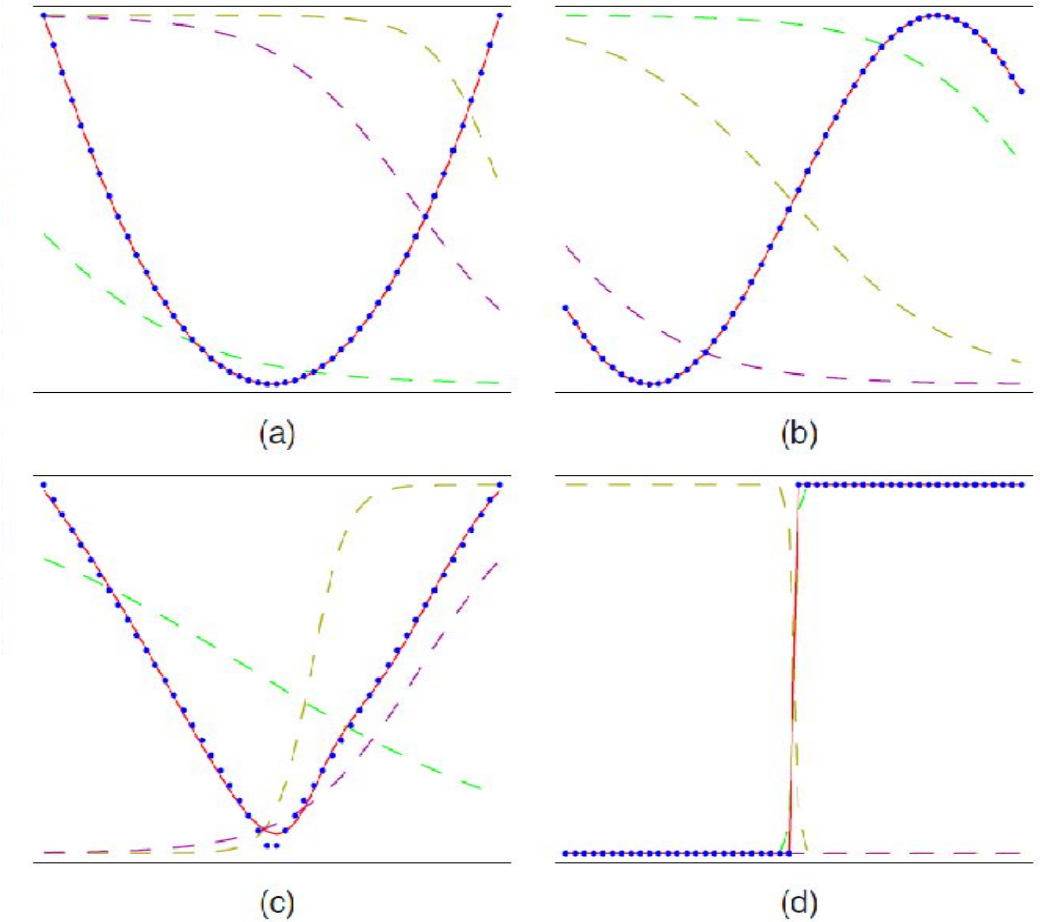
Finally, the derivatives with respect to the first-layer and second-layer weights are given by

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

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

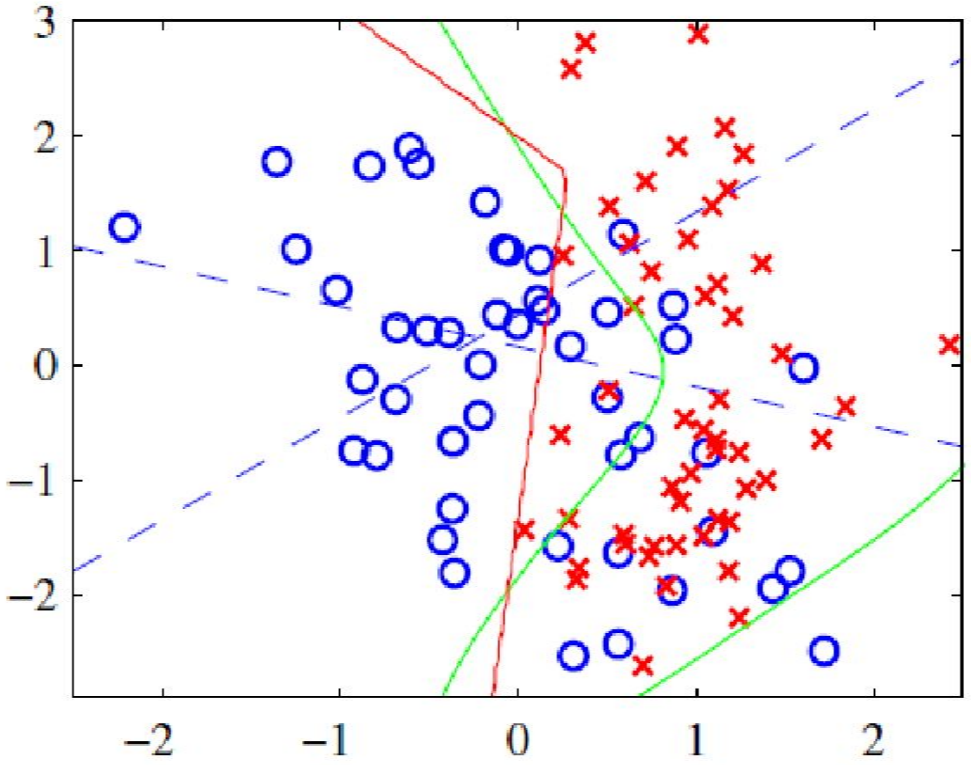
[CMB]

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



[CMB]

Example ANN for Classification



[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.

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

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

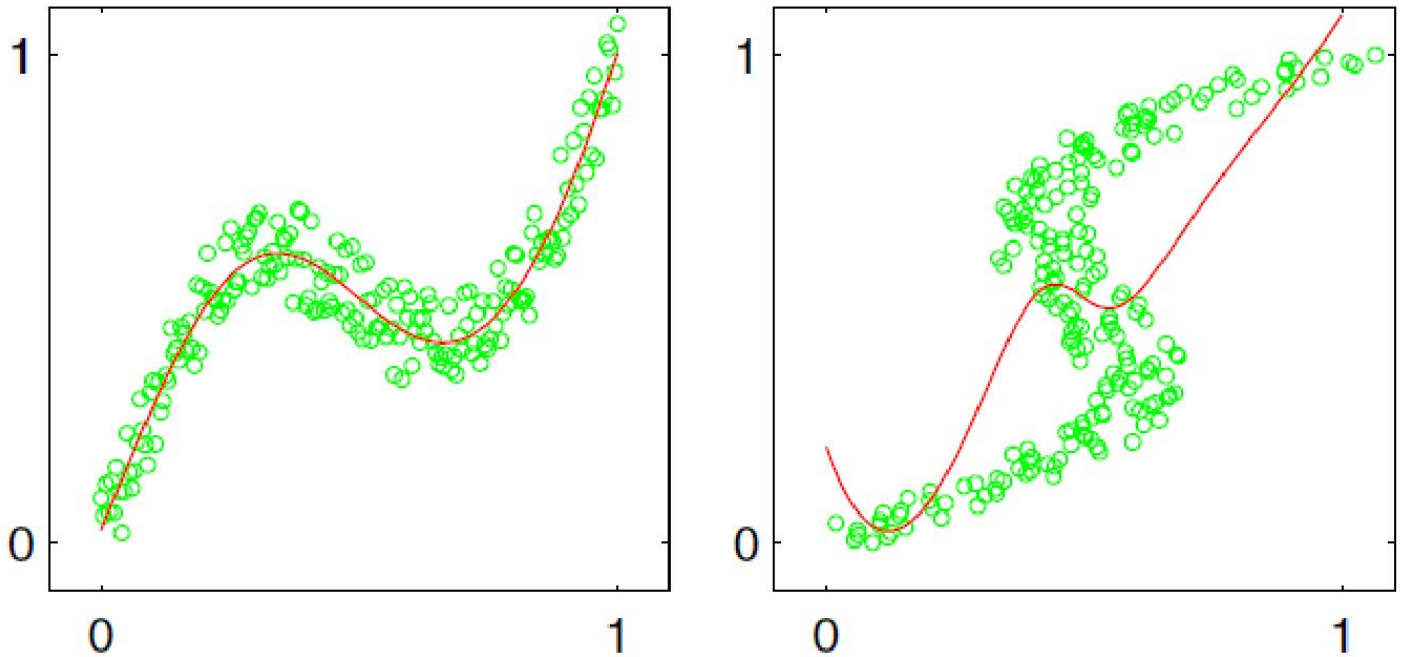
- Backprop. is simply chain rule realized via an $O(W)$ Dynamic Programming (DP) algo.
  - essentially stores already computed derivatives wrt weights in later layers of the network to avoid redundant computations when computing derivatives wrt weights in earlier layers

[CMB]

Outline for Module M9

- M9. Neural Networks
  - M9.0 Introduction/Motivation
  - M9.1 Feed-forward neural networks
    - (Key Idea: Adaptive Basis functions and its Generalizations)
    - (Network architecture, Network training & Backpropagation algo. sketch)
  - **M9.2 Mixture Density Network**
  - M9.3 Concluding thoughts

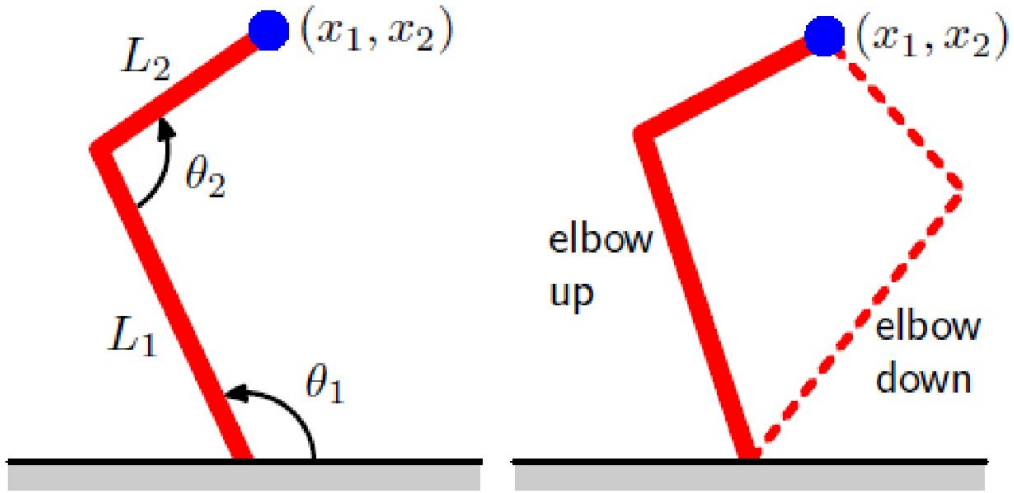
Can ANN find a curve approximating $t|x$ in both these datasets?



[CMB]

Applications: robotics and disease prediction from symptoms

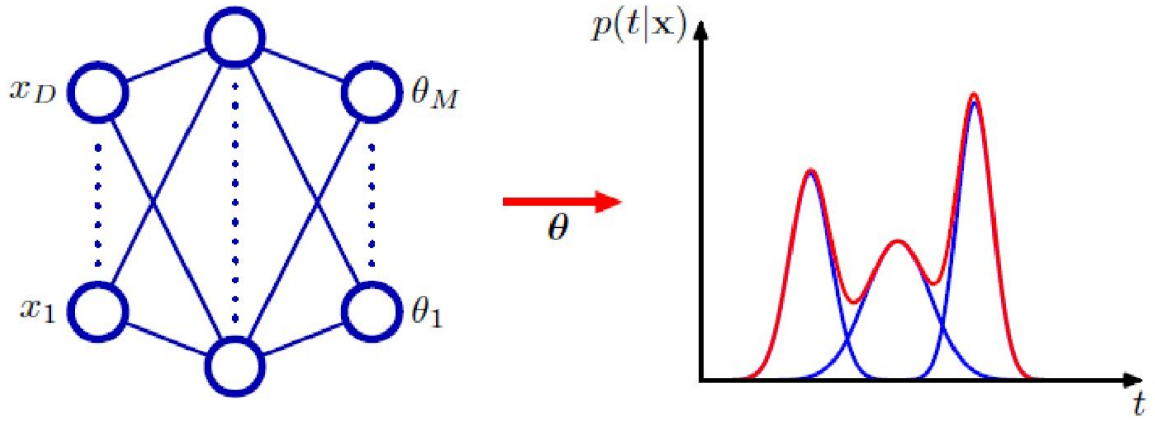
- Forward problem usually has a causal interpretation and a unique solution, whereas the inverse problem has multiple solutions/modes.



[CMB]

Mixture density network (MDN): probabilistic model, network architecture & output unit choices

$$p(\mathbf{t}|\mathbf{x}) = \sum_{k=1}^{K} \pi_k(\mathbf{x}) \mathcal{N}\left(\mathbf{t}|\boldsymbol{\mu}_k(\mathbf{x}), \sigma_k^2(\mathbf{x})\right).$$



$$\pi_k(\mathbf{x}) = \frac{\exp(a_k^\pi)}{\sum_{l=1}^{K} \exp(a_l^\pi)}.$$

$$\mu_k(x) = a_k^\mu \qquad \sigma_k(\mathbf{x}) = \exp(a_k^\sigma).$$

[CMB]

MDN Backprop algo. steps

Step 1) $$E(\mathbf{w}) = -\sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{k} \pi_k(\mathbf{x}_n, \mathbf{w}) \mathcal{N}\left(\mathbf{t}_n | \boldsymbol{\mu}_k(\mathbf{x}_n, \mathbf{w}), \sigma_k^2(\mathbf{x}_n, \mathbf{w})\right) \right\}$$

$$\gamma_k(\mathbf{t}|\mathbf{x}) = \frac{\pi_k \mathcal{N}_{nk}}{\sum_{l=1}^{K} \pi_l \mathcal{N}_{nl}}$$

Step 2) Partial derivatives related to output layer:

$$\frac{\partial E_n}{\partial a_k^{\pi}} = \pi_k - \gamma_k.$$

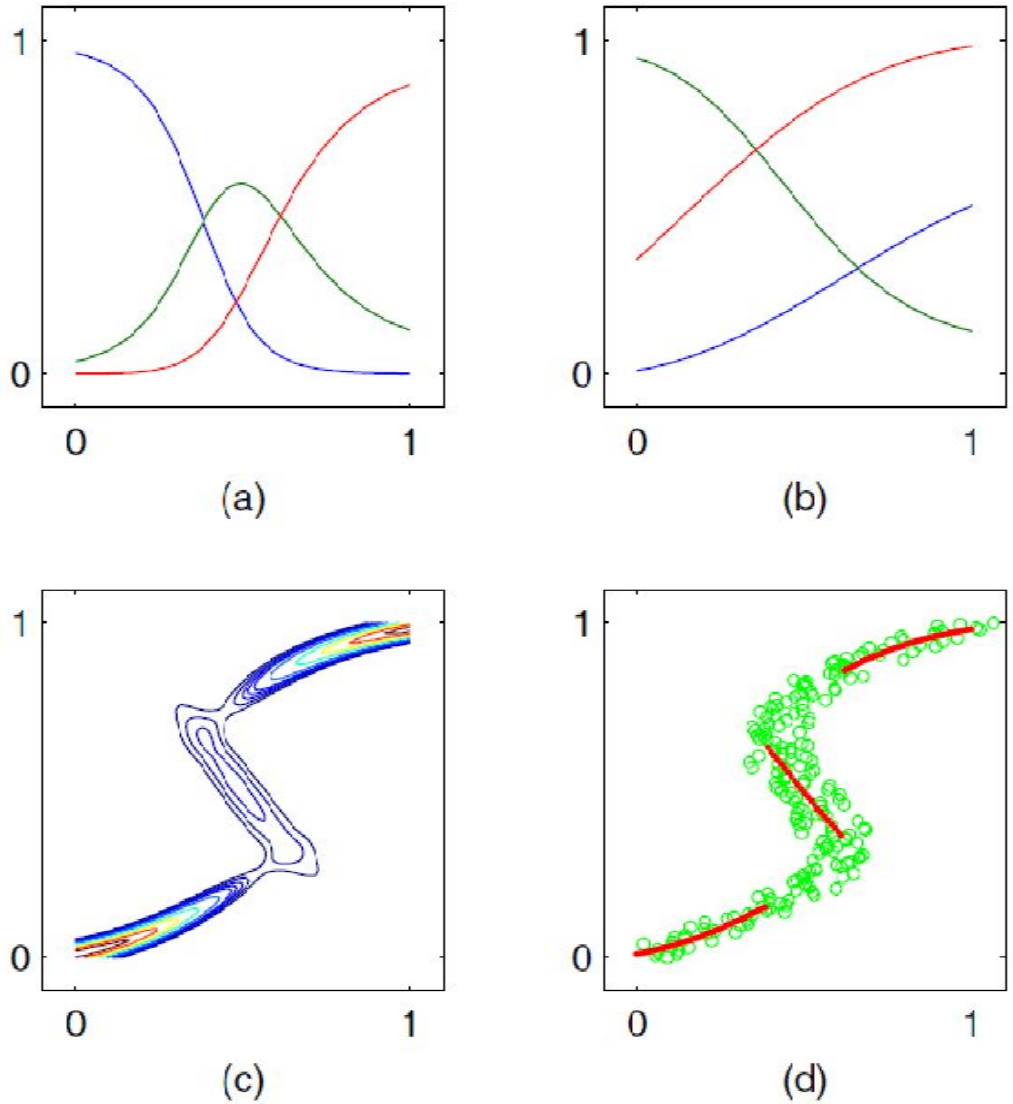
$$\frac{\partial E_n}{\partial a_{kl}^{\mu}} = \gamma_k \left\{ \frac{\mu_{kl} - t_l}{\sigma_k^2} \right\}.$$

$$\frac{\partial E_n}{\partial a_k^{\sigma}} = -\gamma_k \left\{ \frac{\|\mathbf{t} - \boldsymbol{\mu}_k\|^2}{\sigma_k^3} - \frac{1}{\sigma_k} \right\}.$$

Steps 3,4) Propagate above derivatives backwards to compute gradient wrt all weights.

[CMB]

Visualizing the MDN model



[CMB]

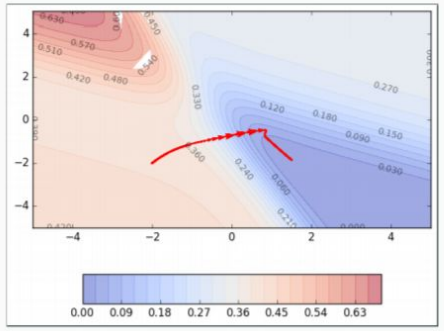
Outline for Module M9

- M9. Neural Networks
  - M9.0 Introduction/Motivation
  - M9.1 Feed-forward neural networks
    - (Key Idea: Adaptive Basis functions and its Generalizations)
    - (Network architecture, Network training & Backpropagation algo. sketch)
  - M9.2 Mixture Density Network
  - **M9.3 Concluding thoughts**

If you are interested to learn more…

Better Optimization Methods

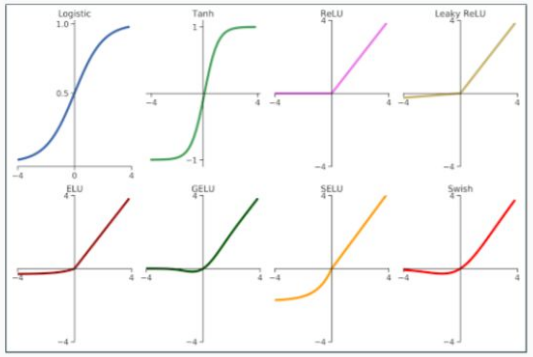
Faster convergence, better accuracies



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

Non-linear method	**(Non-linear) Basis functions**	**Objective function / OP**
Vanilla extn. of linear models		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

Next steps: MDN is one way to probabilistically (soft-ly) combine different regression models (aka conditional mixtures), but there are also other complementary combined models / ensemble methods!

[CMB]

Thank you!