Lecture 18 EE 559 Mar 27, 2024

Machine Learning I: Supervised Methods

B. Keith Jenkins

Announcements

- Slido event code: 1583423
- Graded midterms returned at end of lecture today
- Homework 6 is due Friday
- Project assignment is coming

Reading

- Bishop 5.0-5.3
  - Feedforward ANN

Today's lecture

- Artificial neural networks (part 1)
  - Single neuron unit
  - Activation functions
  - Learning algorithms
  - Single layer of neuron units
- Multilayer ANNs (feedforward)
  - Capabilities
  - Interpretations of multiple layers for pattern classification
  - Learning algorithms

- Annotations to figures on pp. 11-12 revised for clarity post lecture.

Artificial Neural Networks (ANN)

Ref.'s: 1. Bishop, Ch.5
2. I. Goodfellow, et al.
3. C.M. Bishop, H. Bishop, Deep Learning: Foundations and Concepts (Springer, 2024).

What is A.N.N.?

- Sequence of linear combinations of inputs, followed by thresholding
- Network of neuron units and interconnections

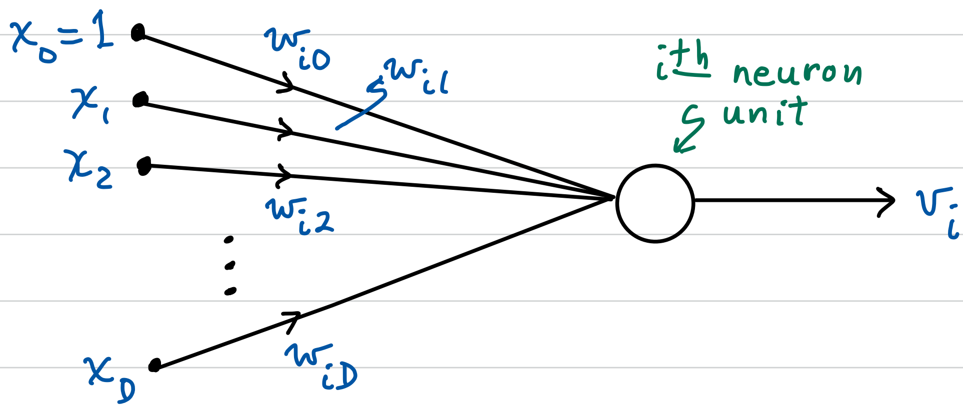
Why A.N.N.?

- Parallel, distributed representation of algorithms
- Mammalian brains are very good at pattern recognition
- Recent successes in applying A.N.N. to problems
- Can be proven to be universal function approximators (with a few caveats)

- Annotations to figures on pp. 11-12 revised for clarity post lecture.

Consider one neuron unit and its input connections

→ augmented notation



↑ $\underline{x}$ ↑ $\underline{w}_i$

Conventional inner-product neuron unit:

$$(1) \qquad v_i = h(\underline{w}_i^T \underline{x}) = h(a_i)$$

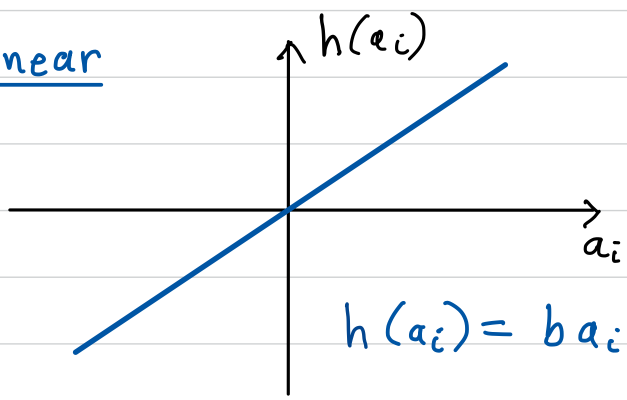
↑ activation function ↑ (membrane) potential or activation

Common Activation Functions

Note: $v_i = h(\underline{w}_i^{(0)T} \underline{x}^{(0)} - \theta_i)$

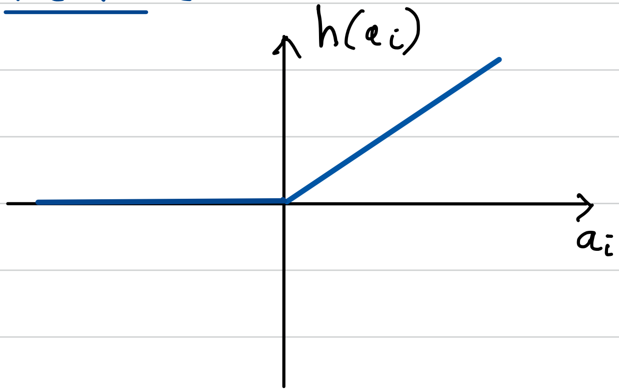
$\theta_i \triangleq$ threshold $= -w_{i0}$.

Linear



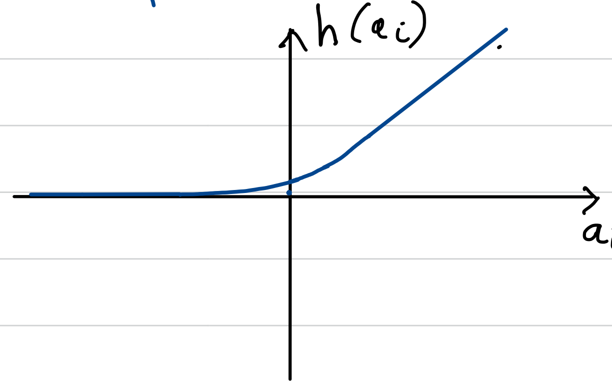
$h(a_i) = b a_i, \quad b > 0.$

ReLU (Rectified linear unit)



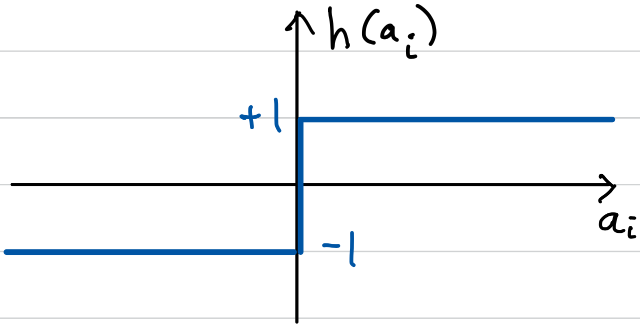
$h(a_i) = \max\{0, b a_i\}, \quad b > 0.$

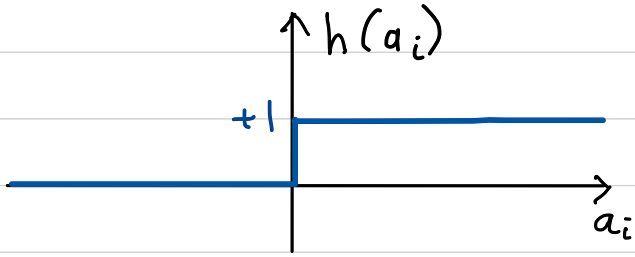
Softplus



$h(a_i) = \ln(1 + e^{a_i})$

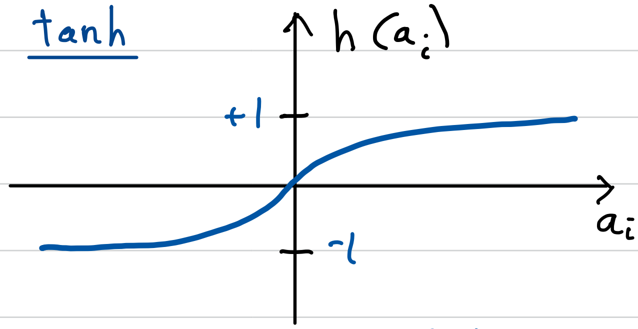
Hard threshold





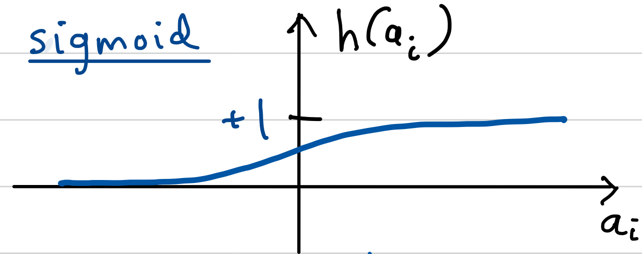
Soft threshold

tanh



$$h(a_i) = \tanh(a_i)$$

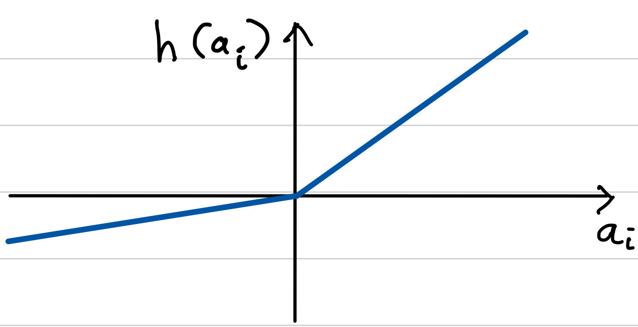
sigmoid



$$h(a_i) = \frac{1}{1+e^{-a_i}}$$

Leaky ReLU

(Leaky rectified linear unit)



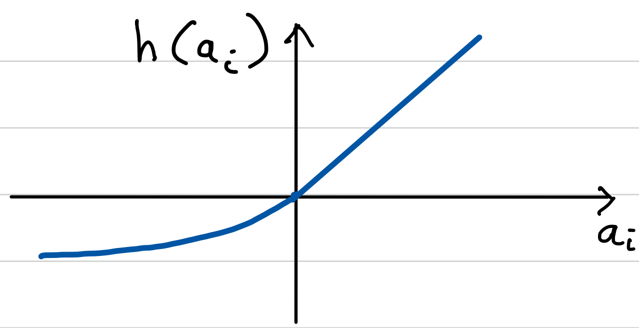
S

$$h(a_i) = \begin{cases} a_i, & \text{if } a_i > 0 \\ b a_i, & \text{otherwise} \end{cases}$$

typically $0 < b < 1$.

ELU

(Exponential LU)



$$h(a_i) = \begin{cases} a_i, & \text{if } a_i \geq 0 \\ b(e^{a_i} - 1), & \text{otherwise} \end{cases}$$

$b > 0$.

Consider 1 neuron unit, h = hard threshold

$$v = h(\underline{w}^T \underline{x}) = \begin{cases} 1, & \underline{w}^T \underline{x} > 0 \\ 0, & \underline{w}^T \underline{x} < 0 \end{cases}$$

What can this neuron unit implement in pattern classification?

→ a 2-class linear classifier: $g(\underline{x}) = \underline{w}^T \underline{x}$

output representation:

$$v = 1 \Rightarrow \underline{x} \in S_1$$
$$v = 0 \Rightarrow \underline{x} \in S_2.$$

ADALINE: adaptive linear element

w's are the adapting mechanism, and provide storage for "long-term memory".

LEARNING ALGORITHM EXAMPLES (classification)

1 neuron unit

Let:

$\hat{v}$ = actual output (classifier prediction) $\in \{0, 1\}$

v_t = target output (true class label) $\in \{0, 1\}$

Perceptron Learning

Original perceptron:

$n = (i \bmod N) + 1$

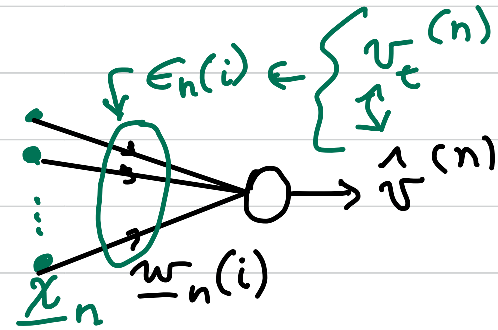
If $\underline{x}_n$ is misclassified:

$$\underline{w}(i+1) = \underline{w}(i) + \eta(i)\, z_n \underline{x}_n$$

Otherwise

$$\underline{w}(i+1) = \underline{w}(i)$$

Next i



Now:

(2) $$\underline{w}(i+1) = \underline{w}(i) + \eta(i) \underbrace{\left[v_t^{(n)} - \hat{v}^{(n)} \right]}_{\epsilon_n(i)} \underline{x}_n$$

$\Delta \underline{w} = \underline{w}(i+1) - \underline{w}(i) = \eta \cdots$

	Target v_t	NN o/p $\hat{v}$	z_n	$v_t^{(n)} - \hat{v}^{(n)}$
miscl.	0 (S_2)	1 (S_1)	-1	-1
	1 (S_1)	0 (S_2)	$+1$	$+1$
correct (o.w.)	$v_t = \hat{v}$			0

Many other linear-classifier learning algorithms can also be posed in A.N.N. framework:

e.g., <u>LMS</u> for regression:

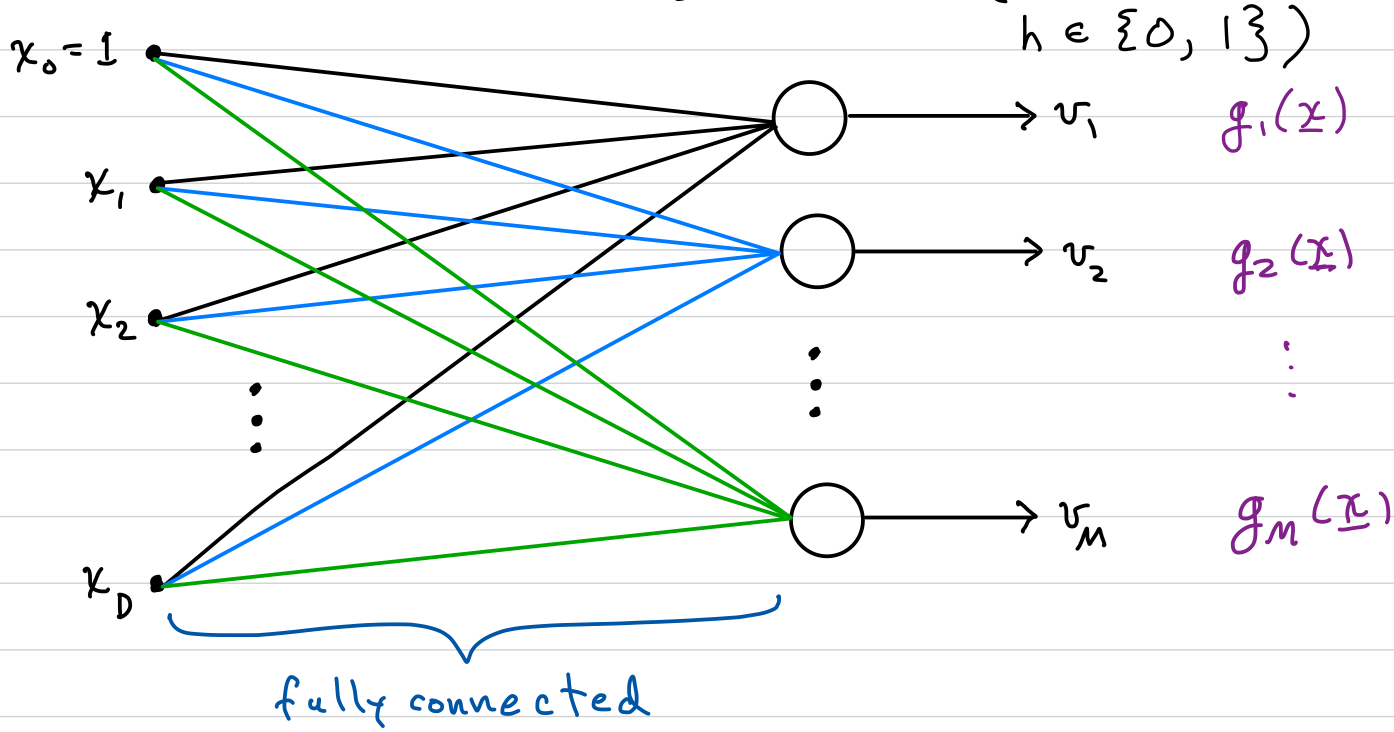
$$\underline{w}(i+1) = \underline{w}(i) - \eta(i) \big(\underbrace{y_n}_{v_t^{(n)}} - \underbrace{\underline{w}^T(i)\, \underline{x}_n}_{\hat{v}^{(n)}(i)} \big) \underline{x}_n, \qquad n = (i \bmod N) + 1$$

Use $h(\cdot) =$ linear $\longrightarrow$

$$\underline{w}(i+1) = \underline{w}(i) - \eta(i) \underbrace{\left(v_t^{(n)} - \hat{v}^{(n)}(i) \right)}_{\epsilon_n(i)} \underline{x}_n$$

$M > 1$ neuron units in a single layer

(h = hard threshold, $h \in \{0, 1\}$)



What can this implement in pattern classification?

$$v_1 = h(a_1) = h(\underline{w}_1^T \underline{x})$$
$$v_2 = h(a_2) = h(\underline{w}_2^T \underline{x})$$
$$\vdots$$
$$v_M = h(a_M) = h(\underline{w}_M^T \underline{x})$$

S → M 2-class classifiers

or → Single OvR M-class classifier, using "one-hot" representation:

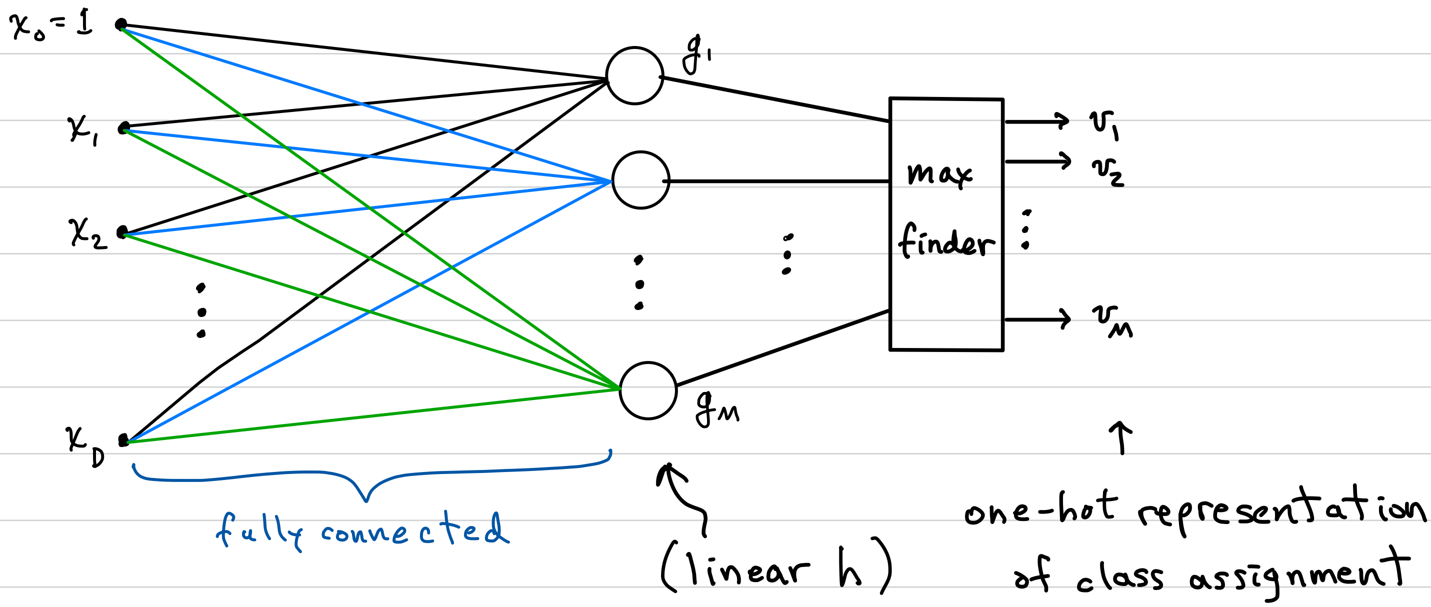
→ Use hard thresholding $h \in \{0, 1\}$.

→ $v_i = 1, \; v_j = 0 \;\; \forall j \neq i \;\Rightarrow\; \underline{x} \in S_i$.

Decision rule

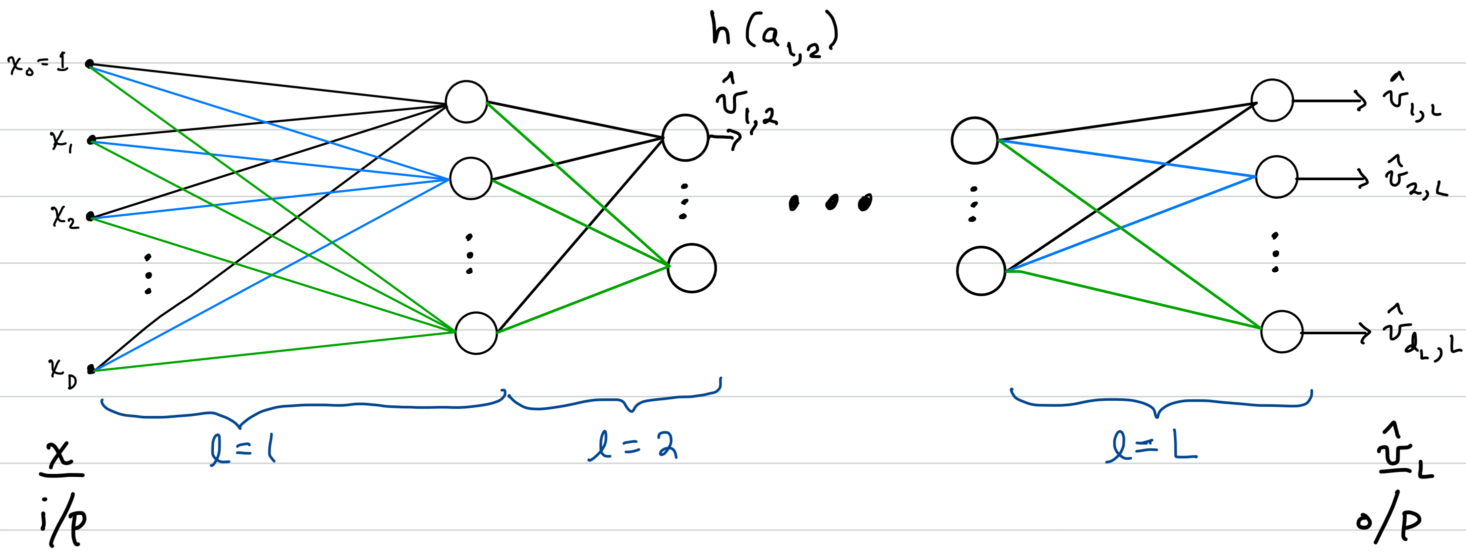
MVM?

→ Can do it with additional layer(s):



MULTILAYER A.N.N.'s [Bishop Ch. 5]

Consider feedforward networks



$h(a) = h(a_{i,l})$ can be a nonlinear fcn.

What can a multilayer feedforward A.N.N. compute in pattern classification? (for nonlinear h)

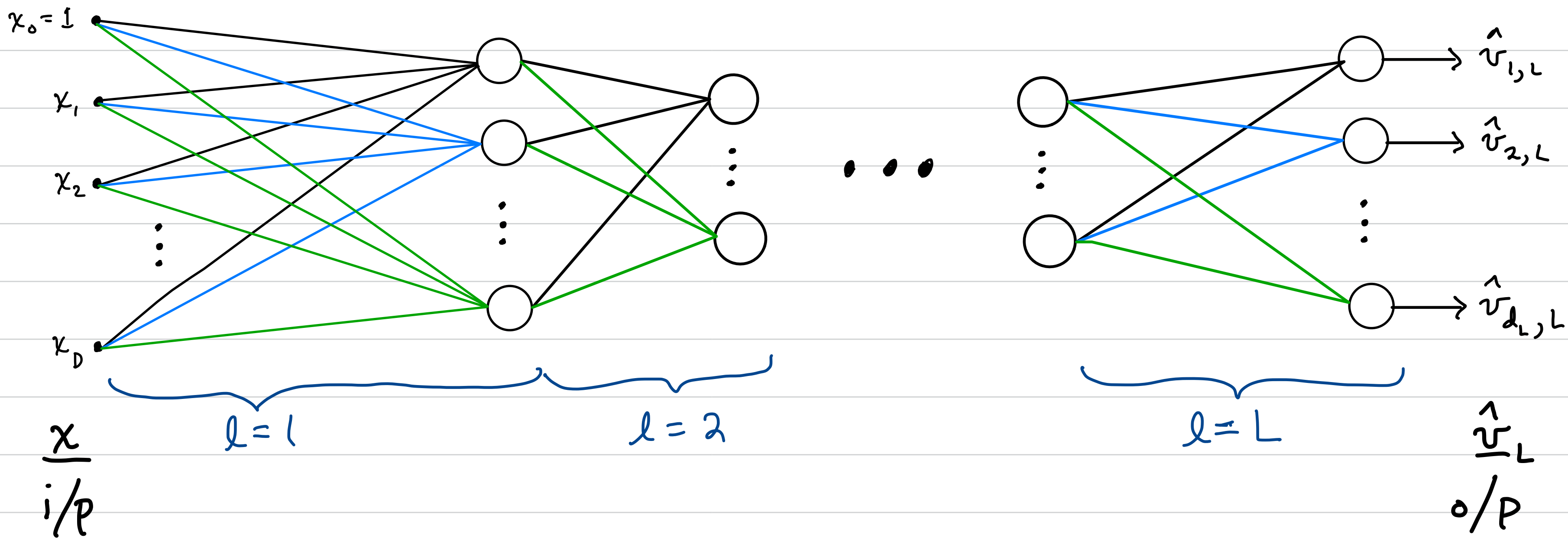
1. i/p is feature space; A.N.N. is a nonlinear classifier.

2. i/p is original feature space; layers $l=1$ to $l=L-1$ perform nonlinear mapping to new feature space. Layer L is a linear classifier.

3. Combination of 1 & 2. (n.l. mapping → n.l. classifier).

4. Inputs are pattern space (inputs before feature extraction). First F layers perform feature extraction. Subsequent $(L-F)$ layers perform classification.

Multilayer Feedforward ANN's – Learning algorithms



The number of units in layer l is d_l.

Criterion function : MSE

$$J = \sum_{n=1}^{N} \mathcal{E}^{(n)} = \sum_{n=1}^{N} \frac{1}{2} \sum_{i=1}^{d_L} \left[v_{i,L}^{(n)} - \hat{v}_{i,L}^{(n)} \right]^2$$

For classification problems, typ. $d_L = C = \#$ classes.

For regression " , $d_L = \#$ output values to predict for each input $\underline{x}$

(often $d_L = 1$ for regression)

Minimize J by gradient descent.

Choose activation functions $(h(a))$.

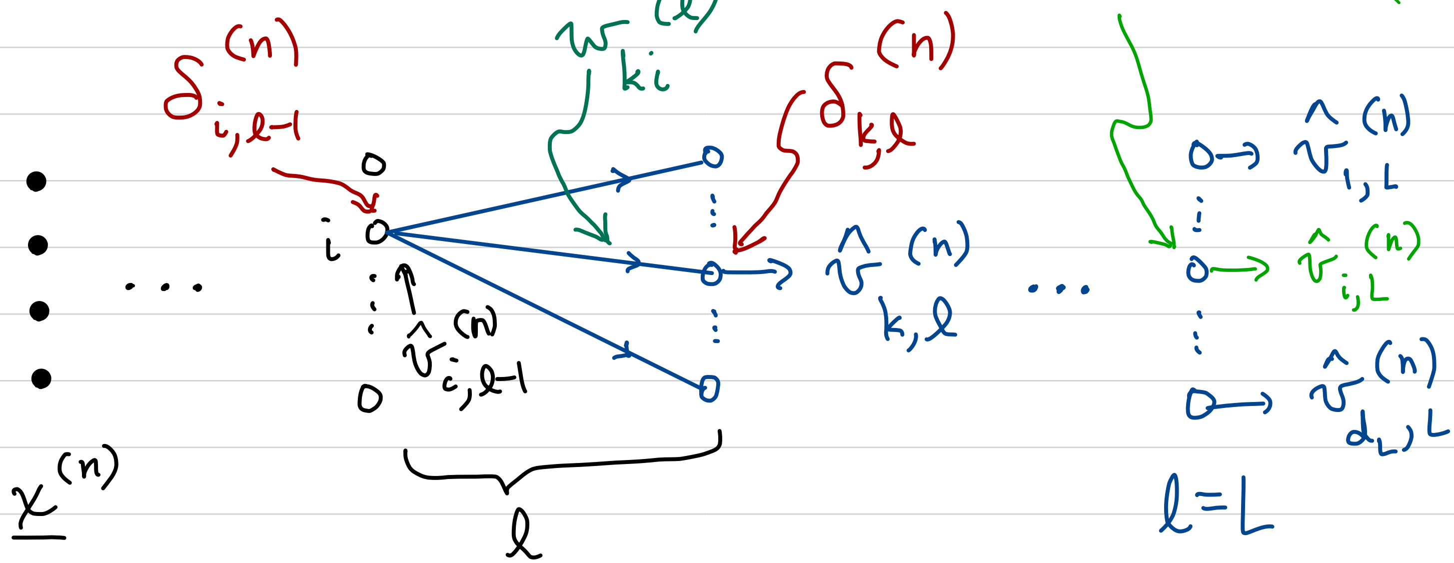
Want differentiable $h(a)$ (almost everywhere, at least). ← Assume we have this

Take $\nabla_{\underline{w}} J(\underline{w})$ [Done in Bishop]

↑ all weights in the ANN.

Results:

$$\delta_{i,L}^{(n)} = h'\left(a_{i,L}^{(n)}\right)\left(v_{i,L}^{(n)} - \hat{v}_{i,L}^{(n)}\right)$$



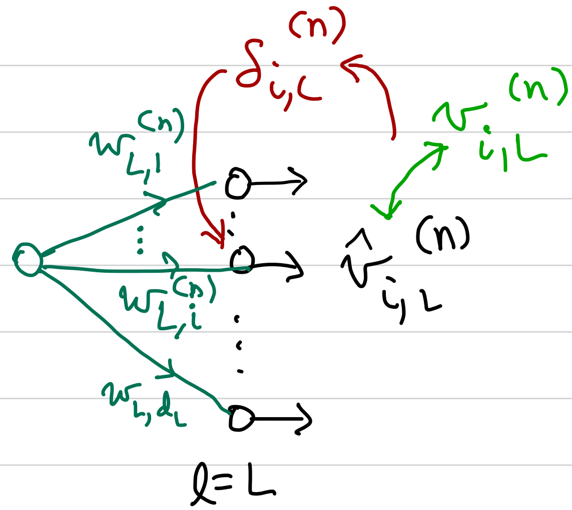
→ Backward error propagation

(1) $$\boxed{\Delta w_{ki}^{(\ell)} = \eta\, \delta_{k,\ell}^{(n)}\, \hat{v}_{i,\ell-1}^{(n)}}$$

$$\begin{cases} \Delta w_{ki}^{(\ell)} = w_{ki}^{(\ell)}(j+1) - w_{ki}^{(\ell)}(j) \\ j = \text{iteration index}\,; \quad \eta = \eta(j) \end{cases}$$

(2) $$\boxed{\delta_{i,L}^{(n)} = \left[v_{i,L}^{(n)} - \hat{v}_{i,L}^{(n)} \right] h'\!\left(a_{i,L}^{(n)}\right)}$$

(3) $$\boxed{\delta_{i,\ell-1}^{(n)} = h'\!\left(a_{i,\ell-1}^{(n)}\right) \sum_k \delta_{k,\ell}^{(n)}\, w_{ki}^{(\ell)}}$$



(i) Forward pass: calculate $\hat{v}_L^{(n)}$ for input $\underline{x}_n$.

(ii) Backward pass: calculate error terms $\delta_{k,\ell}^{(n)}$ and weight updates $\Delta w_{ki}^{(\ell)}$