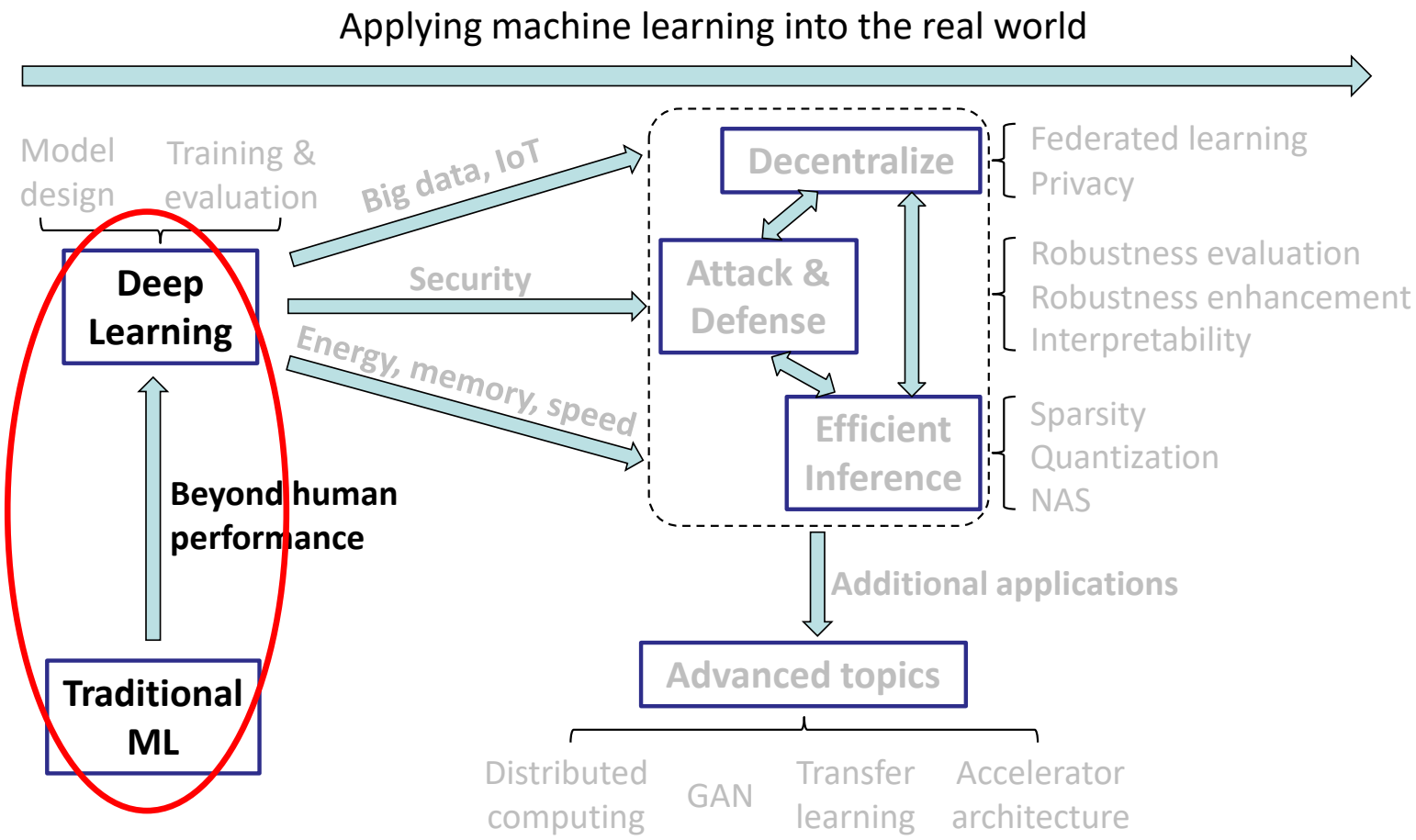
ECE 661 COMP ENG ML & DEEP NEURAL NETS

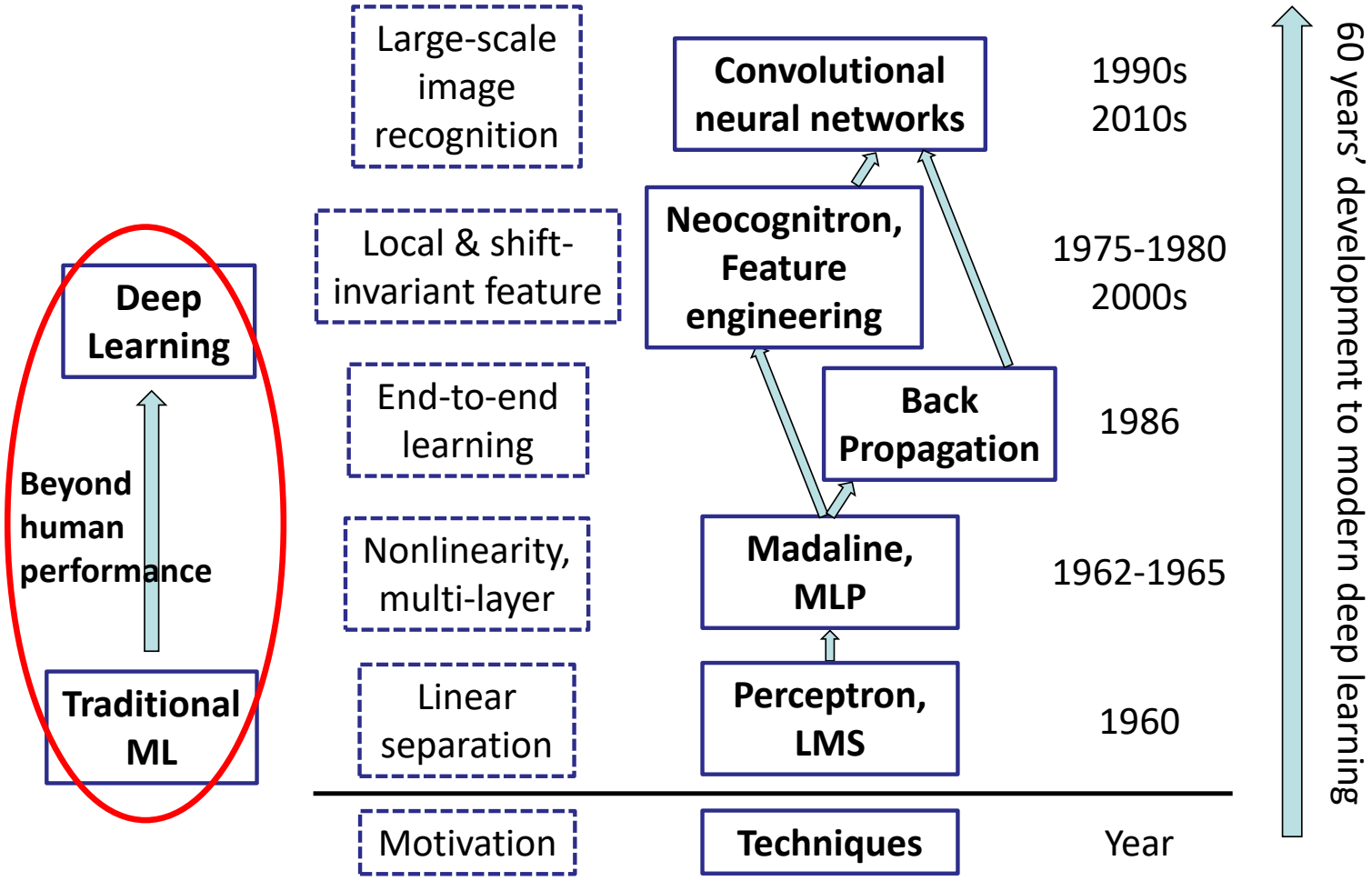
2. 60 YEARS OF NEURAL NETWORK (1/3): PERCEPTRON TO BACK PROPAGATION

YIRAN CHEN, SPRING 2024

Roadmap of the course



Next 3 lectures



Outline

- Lecture 2: NN and Back Propagation
  - Linear separation and Least Mean Square (LMS) learning rule
  - Multi-layer perceptron (MLP)
  - Nonlinear activation functions
  - Back propagation
  - Training issues

- Lecture 3: Emergence of convolution
- Lecture 4: Building a CNN model

Outline

- Lecture 2: NN and Back Propagation
  - Linear separation and Least Mean Square (LMS) learning rule
  - Multi-layer perceptron (MLP)
  - Nonlinear activation functions
  - Back propagation
  - Training issues

- Lecture 3: Emergence of convolution
- Lecture 4: Building a CNN model

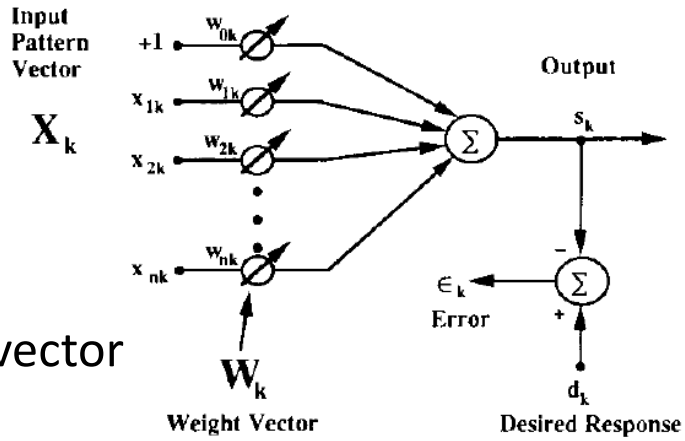
Objective of machine learning

- Machine learning is a process of finding a function, a.k.a. **Model**, to process input **data** to fulfill a learning task
  - Supervised learning: F*(data) → **label**
- **Loss function**: Define learning task
  - Difference between the current model $\mathbf{F_W}$ we have and the golden objective function **F*:** $L\left(F_W(x), F^*(x)\right)$
- The process of minimizing the loss is called **Optimization**
  - a.k.a. **Training**

Starting point: Linear model (Adaline)

- Simplest (yet useful) function class for ML model

Data: N feature-vector
+ a constant



Weight: N+1 element vector

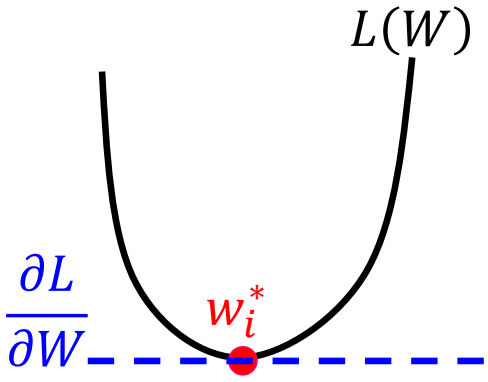
Label

- Model
  - Regression $s_k = W^T X_k = w_0 + \sum_{i=1}^{N} w_i x_{ik}$ w_0 is called Bias
  - Classification $s_k = \text{sgn}(W^T X_k)$ sgn() is an Activation Function
- Loss: Mean Squared Error
  - Error term $\epsilon_k = \frac{1}{2}(d_k - s_k)^2$, loss $L(W) = \frac{1}{K}\sum_{k=1}^{K} \epsilon_k$

Widrow, Bernard, and Michael A. Lehr. "30 years of adaptive neural networks: perceptron, madaline, and backpropagation." *Proceedings of the IEEE* 78.9 (1990): 1415-1442.

Minimizing the loss

- Loss function: $L(W) = \frac{1}{2K}\sum_{k=1}^{K}(d_k - W^T X_k)^2$
- Training: given d_k and X_k, tuning W to minimize $L(W)$
- How? Find minima of loss function $L(W)$
  - **Minima**: the points with zero gradients



- We can find the W^* directly

by solving the gradient formula:

$$\frac{\partial L}{\partial W}(W^*) = 0$$

  - Least square (Wiener) solution for linear model

$$D = \begin{bmatrix} d_1 \\ \vdots \\ d_K \end{bmatrix}, X = \begin{bmatrix} X_1^T \\ \vdots \\ X_K^T \end{bmatrix} \qquad W^* = (X^T X)^{-1} X^T D$$

Prove it yourself. Hint:

$L(W) = (D - XW)^T (D - XW)/2K$

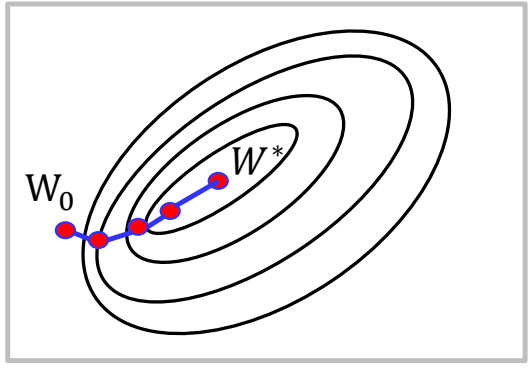
What if gradient formula can't be solved?

- L is complex and there's no closed-form solution to the gradient formula? (True for deep learning)
- Don't have all the data (True for most cases)

- We can find the W^* in an iterative way

$$W^{k+1} = W^k - r\frac{\partial L}{\partial W}(W^k)$$

r: learning rate, defining the step size



- This process is called **Gradient Descent**

Gradient descent for linear model: LMS

- Single loss: $L_k = \frac{1}{2}(d_k - s_k)^2 = \frac{1}{2}\left(d_k - \sum_{i=0}^{N} w_i x_{ik}\right)^2$
- Gradient computation
  - Scalar form: $\frac{\partial L_k}{\partial w_i} = -x_{ik}(d_k - s_k)$
  - Vector form: $\frac{\partial L_k}{\partial W} = -X_k(d_k - s_k)$
- **Least Mean Square (LMS)** algorithm

$$W^{k+1} = W^k + rX_k(d_k - s_k)$$

- For classification, $d_k = \pm 1$
  - $(d_k - s_k) = 0$ when correct, $2\,\mathrm{sgn}(d_k)\,X_k$ when wrong

$$W^{k+1} = \begin{cases} W^k, & d_k = s_k \\ W^k + r\,\mathrm{sgn}(d_k)\,X_k, & d_k \neq s_k \end{cases}$$

Perceptron

A toy example

A toy example to illustrate LMS.

- Each day you get lunch at the cafeteria.
  - Your diet consists of **fish**, **chips**, and **ketchup**.
  - You get several portions of each.

- The cashier only tells you the total price of the meal.
  - Start with random guesses for the prices
  - After several days, you should be able to figure out the price of each portion with LMS

Solving the equations iteratively

- Each meal price gives a linear constraint on the prices of the portions:

$$price = x_{fish}w_{fish} + x_{chips}w_{chips} + x_{ketchup}w_{ketchup}$$

- The prices of the portions are like the weights of a linear model:

$$\mathbf{w} = (w_{fish}, w_{chips}, w_{ketchup})$$

- We will start with guesses for the weights and then adjust the guesses slightly to give a better fit to the prices given by the cashier.

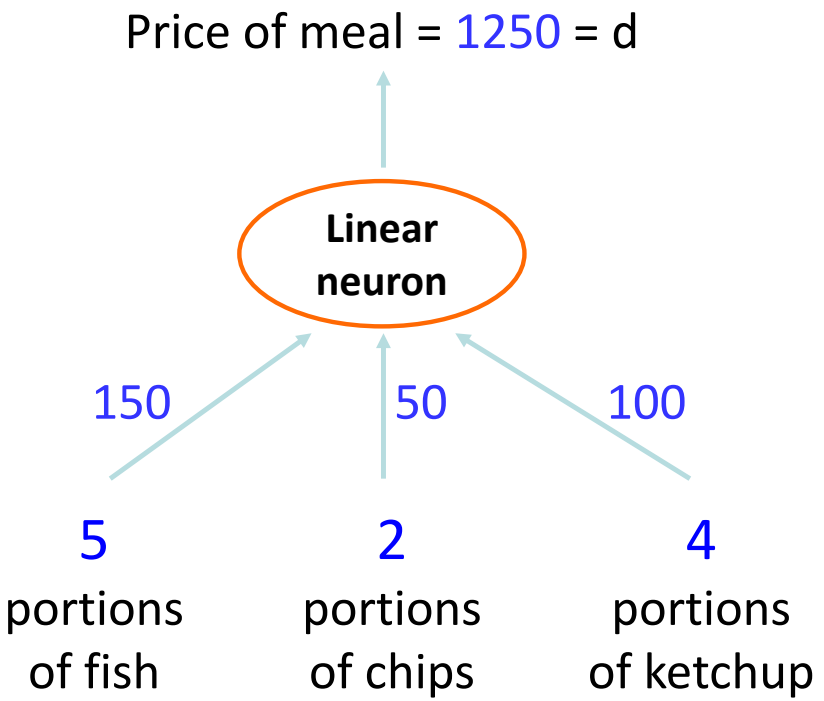
The true weights used by the cashier

- We have a dataset:

- Data:
  (# fish, # chips, # ketchup)
- Target (desired output):
  Total money

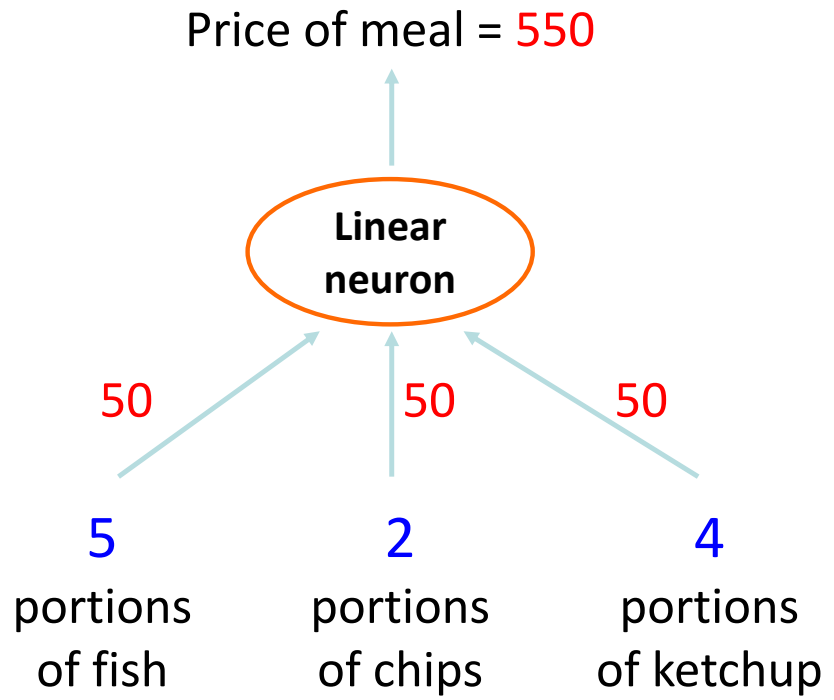
Examples:

- X: (5, 2, 4), d: 1250
- X: (3, 3, 3), d: 900
- X: (0, 5, 1), d: 350
- …



Training step 0 with arbitrary initial weights

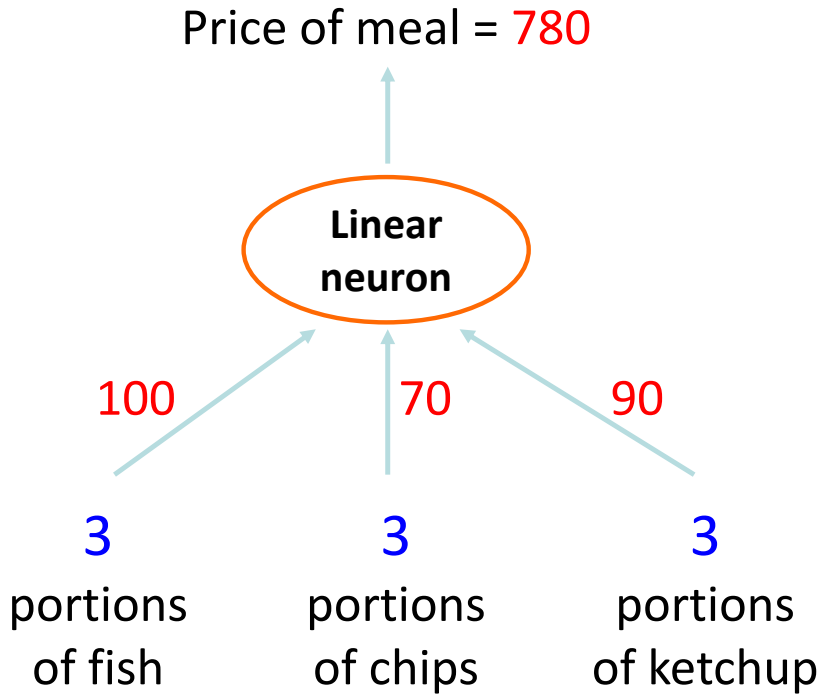
- Error
  = Target – Prediction
  = 1250 – 550 = 700
- The LMS rule:
  $\Delta w_i = \varepsilon\, x_i\, (d - y)$
  - (d-y) = 700
  - x_f for fish = 5
  - x_c for chips = 2
  - x_k for ketchup = 4
- With a learning rate ε of 1/70, the weight changes are :
  +50, +20, +40.



X: (5, 2, 4), d: 1250
X: (3, 3, 3), d: 900
X: (0, 5, 1), d: 350

Training step 1

- Error
  = Target – Prediction
  = 900 – 780 = 120
- The LMS rule:
  $\Delta w_i = \varepsilon\, x_i\, (d - y)$
  - (d-y) = 120
  - x_f for fish = 3
  - x_c for chips = 3
  - x_k for ketchup = 3
- With a learning rate ε of 1/12, the weight changes are :
  +30, +30, +30.



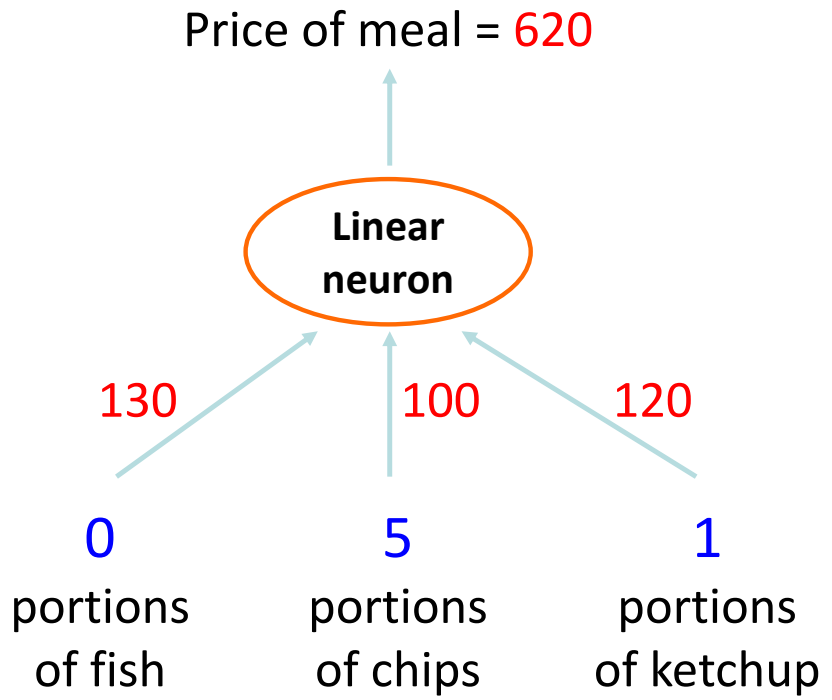
X: (5, 2, 4), d: 1250

X: (3, 3, 3), d: 900

X: (0, 5, 1), d: 350

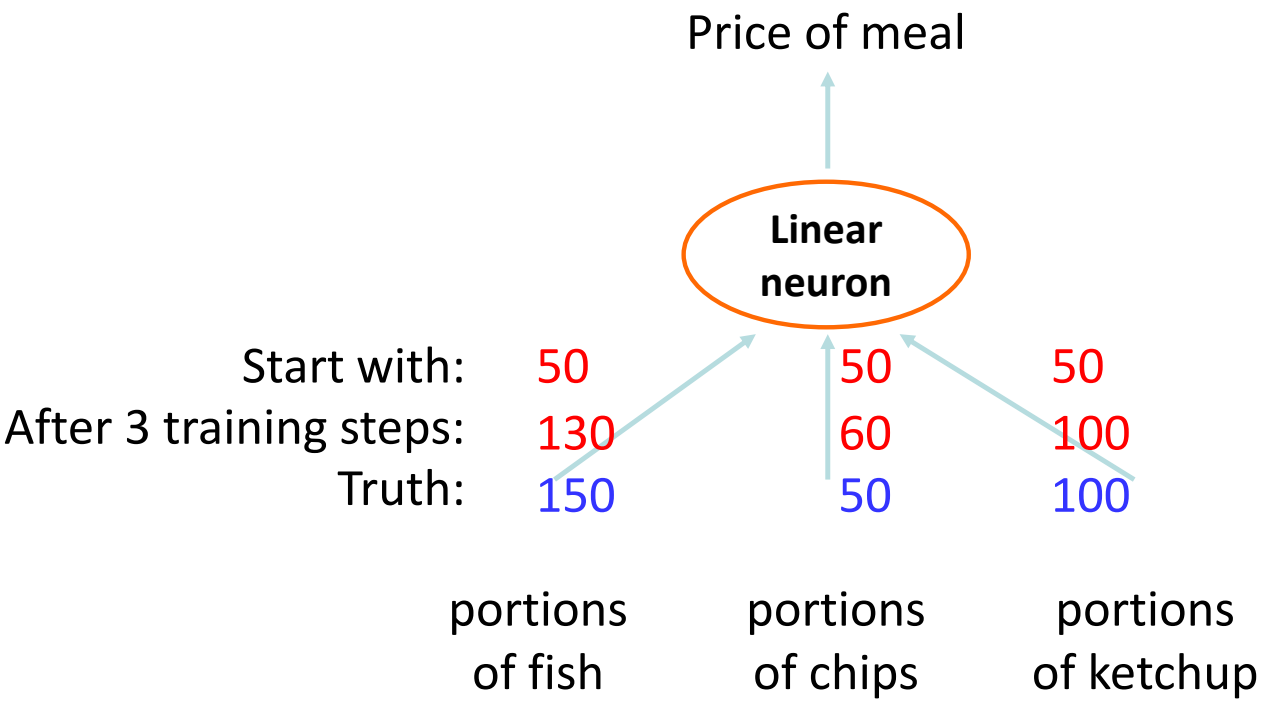
Training step 2

- Error
  = Target – Prediction
  = 350 – 620 = -270
- The LMS rule:
  $\Delta w_i = \varepsilon\, x_i\, (d - y)$
  - (d-y) = -270
  - x_f for fish = 0
  - x_c for chips = 2
  - x_k for ketchup = 1
- With a learning rate ε of 2/27, the weight changes are :
  0, -40, -20



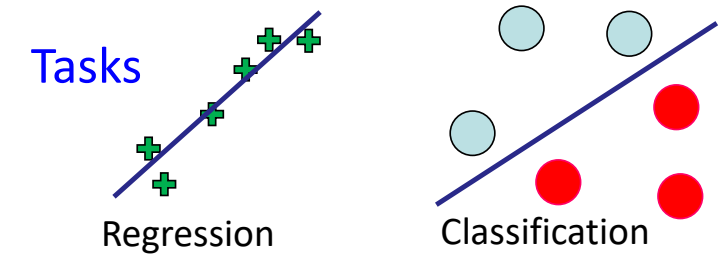
X: (5, 2, 4), D: 1250
X: (3, 3, 3), D: 900
X: (0, 5, 1), D: 350

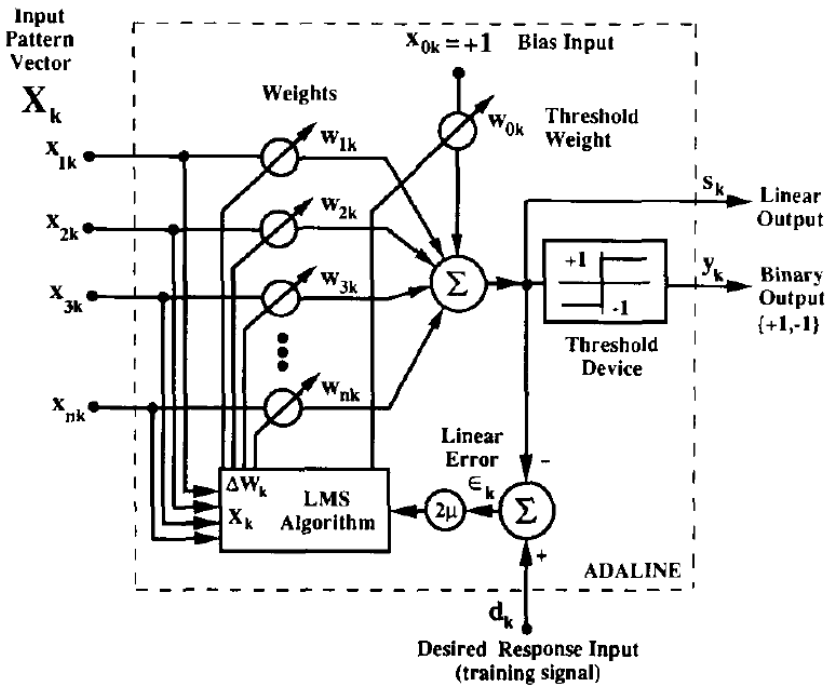
Result of training



gradient descent 的过程. 也就是这样了

Linear model summary





Inference:

Regression $s_k = W^T X_k$

Classification $s_k = \text{sgn}(W^T X_k)$

Loss:

MSE $L_k = \frac{1}{2}(d_k - s_k)^2$

Training:

LMS

$W^{k+1} = W^k + rX_k(d_k - s_k)$

Perceptron

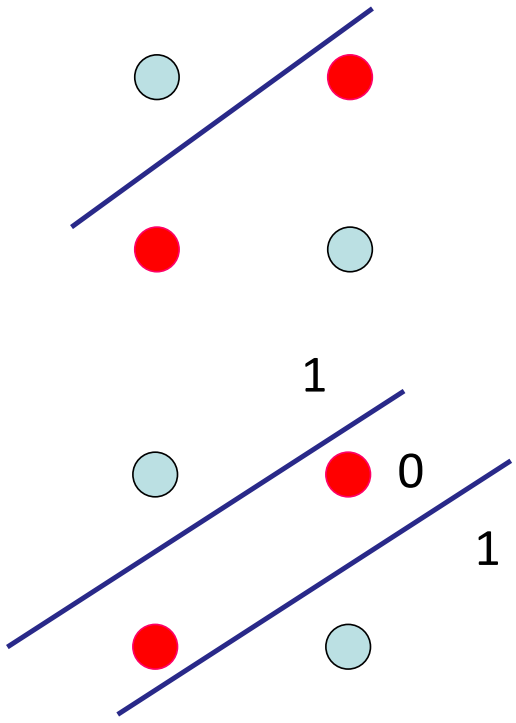
$W^{k+1} = W^k + r\,\text{sgn}(d_k)\,X_k, d_k \neq s_k$

Widrow, Bernard, and Michael A. Lehr. "30 years of adaptive neural networks: perceptron, madaline, and backpropagation." *Proceedings of the IEEE* 78.9 (1990): 1415-1442.

Representing XOR?

- Linear model is useful, but the capacity is limited
- Consider XOR function

X_1	X_2	$d = X_1 \oplus X_2$
0	0	0
0	1	1
1	0	1
1	1	0



- Not linearly separable!
  - But can be separated with two lines
  - Multi-layer perceptron

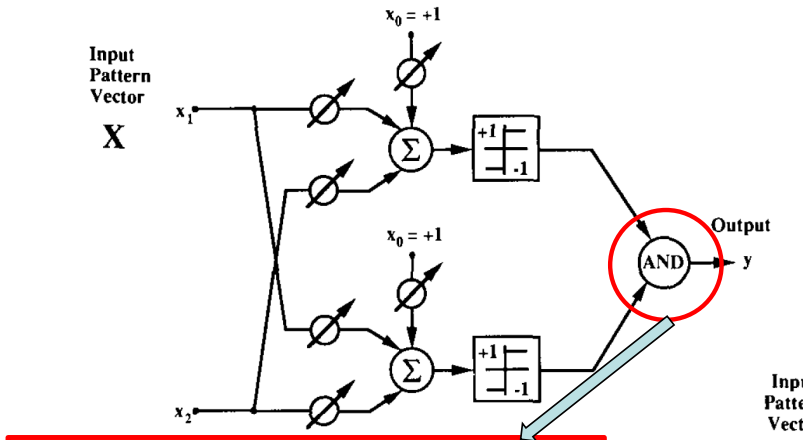
Outline

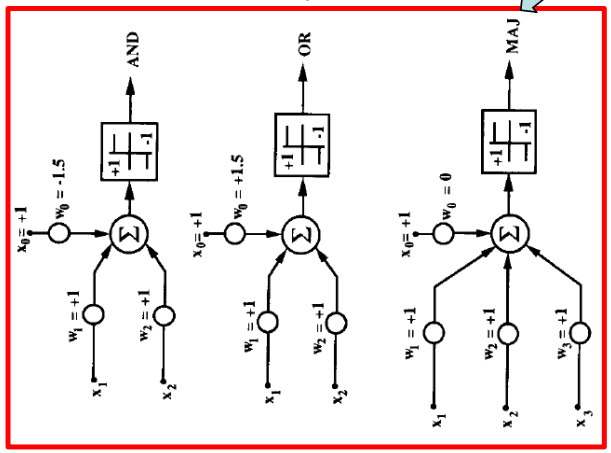
- Lecture 2: NN and Back Propagation
  - Linear separation and Least Mean Square (LMS) learning rule
  - Multi-layer perceptron (MLP)
  - Nonlinear activation functions
  - Back propagation
  - Training issues

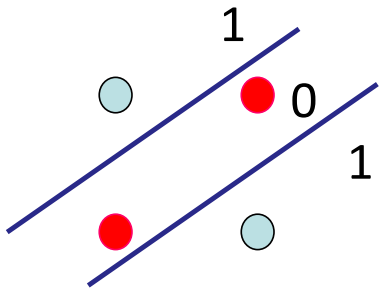
- Lecture 3: Emergence of convolution
- Lecture 4: Building a CNN model

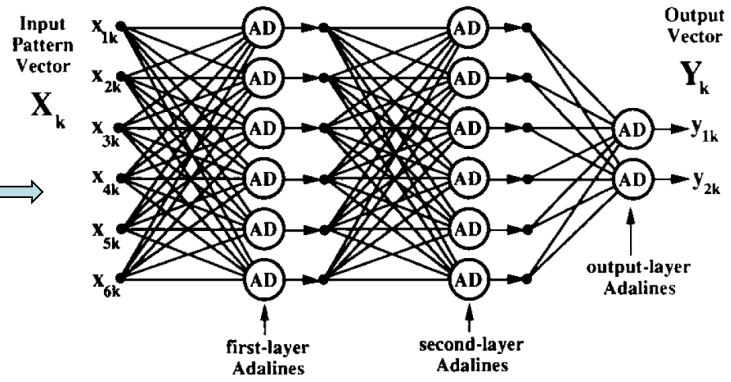
Early MLP: Madaline

- Combining multiple perceptron with logic functions





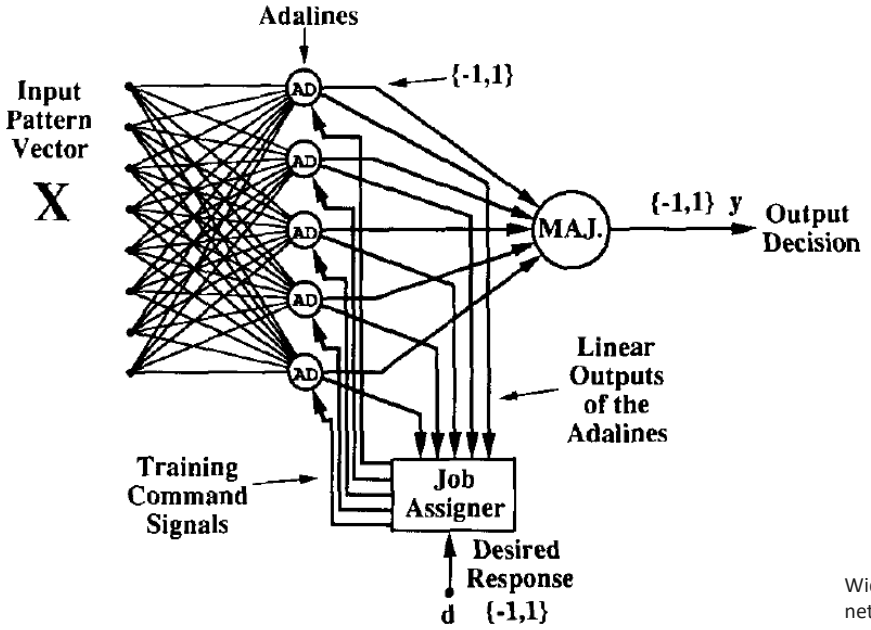




Widrow, Bernard, and Michael A. Lehr. "30 years of adaptive neural networks: perceptron, madaline, and backpropagation." *Proceedings of the IEEE* 78.9 (1990): 1415-1442.

Training Madaline: Error-Correction Rule

- MDI: Madaline with fixed second layer
  - Responsibility assignment with **minimal disturbance**: Assign responsibility to the Adaline that can most easily assume it to correct output error

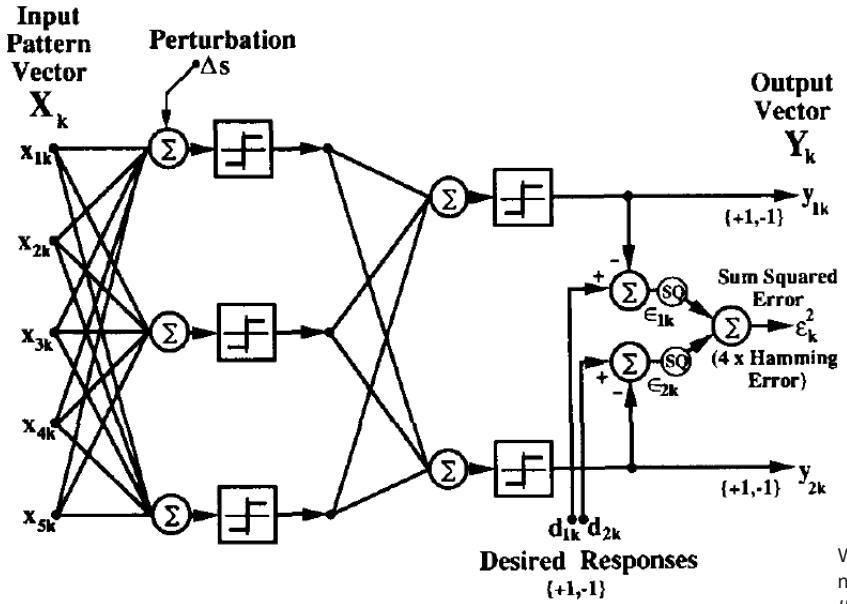


If majority vote wrong 2-3, ask the linear model with **closest-to-0** wrong output to adapt

Widrow, Bernard, and Michael A. Lehr. "30 years of adaptive neural networks: perceptron, madaline, and backpropagation." *Proceedings of the IEEE* 78.9 (1990): 1415-1442.

Training Madaline: Error-Correction Rule

- MDII: Arbitrary multi-layer Madaline
  - Trial adaption: start from layer 1, randomly select 1 Adaline and flip result (perturb), adapt the selected Adaline if output error reduced after flipping, otherwise keep same



Randomly select 2 after each single Adaline is tried, then 3 after all pairs are tried, then 4, 5 etc.

Go to layer 2 after exhausting possibilities with the layer 1, then layer 3, 4, etc. Finally go back to layer 1 after final layer with new random selection pattern

Widrow, Bernard, and Michael A. Lehr. "30 years of adaptive neural networks: perceptron, madaline, and backpropagation." *Proceedings of the IEEE* 78.9 (1990): 1415-1442.

Drawback of Error-Correction Rule

- Complexity
  - N elements in 1 layer requires $2^N - 1$ trials to exhaust possibilities, hard to scale to large models

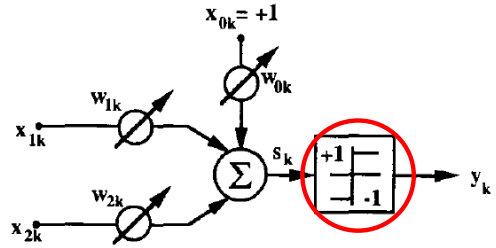
- No theoretical guarantee on convergence
  - Result sensitive to random selection pattern and initialization value, don't know when to stop

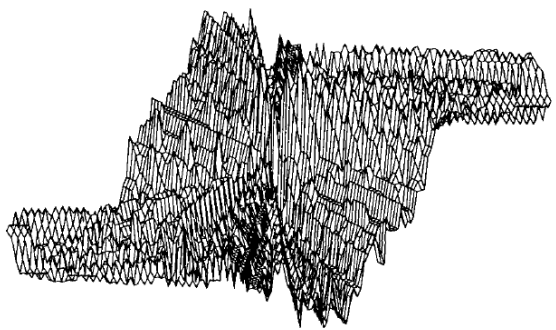
- This leads to the "abandonment of connectionism" in 1969, and partially leads to the first "AI winter" in 1970s'

Widrow, Bernard, and Michael A. Lehr. "30 years of adaptive neural networks: perceptron, madaline, and backpropagation." *Proceedings of the IEEE* 78.9 (1990): 1415-1442.

Why not gradient descent?

- Madaline use step function (signum) as activation, not smooth
- 1960s' computing architecture do not support the computation of smooth activation
- It takes additional 15-20 years to see researcher replacing signum with smooth sigmoid function, paving the path for end-to-end gradient-based training





Example loss surface of model with signum activation

Widrow, Bernard, and Michael A. Lehr. "30 years of adaptive neural networks: perceptron, madaline, and backpropagation." *Proceedings of the IEEE* 78.9 (1990): 1415-1442.

Outline

- Lecture 2: NN and Back Propagation
  - Linear separation and Least Mean Square (LMS) learning rule
  - Multi-layer perceptron (MLP)
  - Nonlinear activation functions
  - Back propagation
  - Training issues

step (non-smooth) func, no gradient
limited
sigmoid func (smooth one)

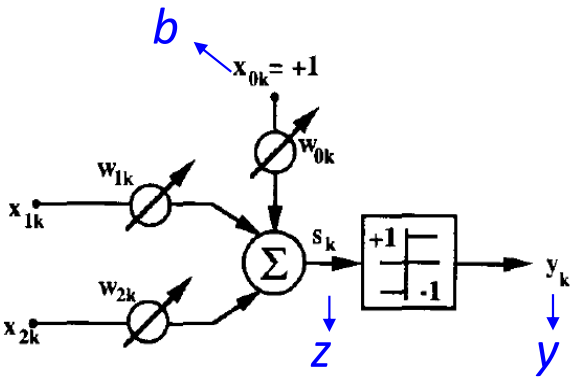
- Lecture 3: Emergence of convolution
- Lecture 4: Building a CNN model

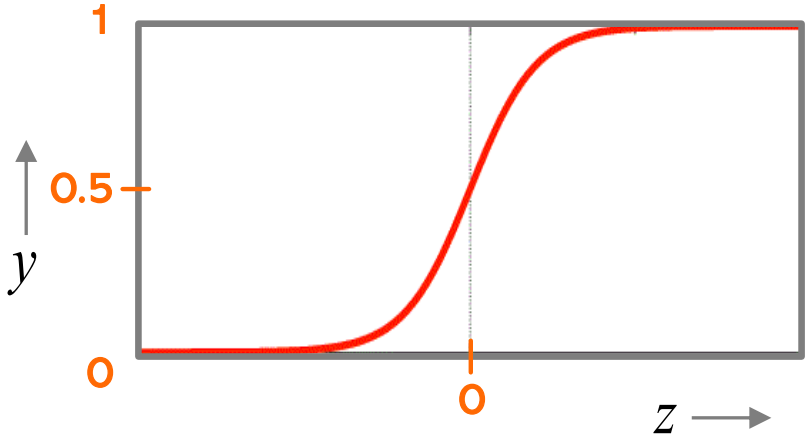
Logistic neuron

- Linear model with **Sigmoid activation**
- A smooth and bounded function
  - Have nice derivatives which make learning easy.

$$z = b + \sum_i x_i w_i$$

$$y = \frac{1}{1 + e^{-z}}$$





The derivatives of a logistic neuron

- The derivatives of *z*, with respect to the inputs and the weights are very simple:

$$z = b + \sum_i x_i w_i$$

$$\frac{\partial z}{\partial w_i} = x_i \qquad \frac{\partial z}{\partial x_i} = w_i$$

- The derivative of the output with respect to the logit is simple if you express it in terms of the output:

$$y = \frac{1}{1 + e^{-z}}$$

★ $$\frac{dy}{dz} = y(1 - y)$$

Prove it yourself.

Putting logistic neurons into neural network

- A **neural network** follows a layer-wise fashion
- Each layer: a function $f_i(W_i,\cdot)$ with weight W_i
  - Logistic neuron: $f_i(W_i, x_i) = sigmoid\left(W_i^T x_i\right)$
  - Whole network: concatenating all layers

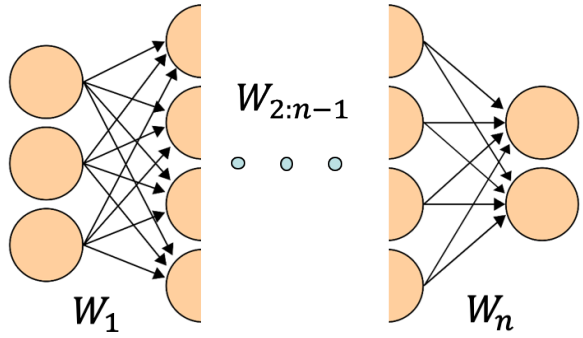
- Evaluate the output given an input
  - **Forward propagation**

$$y = F(x) \coloneqq f_n(W_n, f_{n-1}(W_{n-1}, f_{\dots} f_1(W_1, x)))$$

- Compute gradient for training
  - **Backward propagation**

$$\partial L\left(F(x), F^*(x)\right)/\partial W_i$$

  - Details coming up



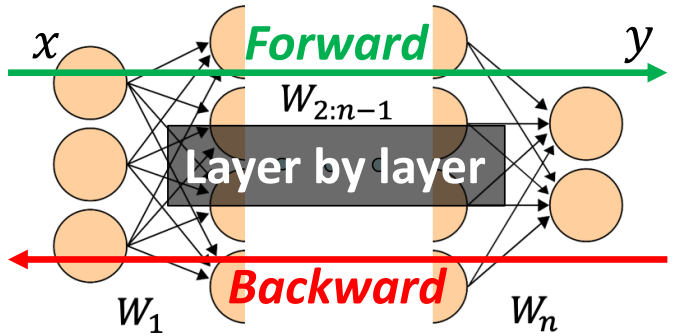
Outline

- Lecture 2: NN and Back Propagation
  - Linear separation and Least Mean Square (LMS) learning rule
  - Multi-layer perceptron (MLP)
  - Nonlinear activation functions
  - Back propagation
  - Training issues

- Lecture 3: Emergence of convolution
- Lecture 4: Building a CNN model

Training loop of neural network

$$y = F(x) \coloneqq f_n(W_n, f_{n-1}(W_{n-1}, f_{\dots} f_1(W_1, x)))$$



Another batch of data comes in

Refresh parameters

Calculate Prediction y

Compare with target d

$$W^{m+1} = W^m - r\frac{\partial Loss}{\partial W}(W^m)$$

W^{m+1}: new weight

W^m: current weight

$$Loss = L(y, d)$$

For i_{th} layer's parameters

$$\frac{\partial Loss}{\partial W_i} = \frac{\partial L}{\partial f_n} \cdot \frac{\partial f_n}{\partial f_{n-1}} \cdot \frac{\partial f_{n-1}}{\partial f_{n-2}} \cdots \frac{\partial f_i}{\partial W_i}$$

The idea behind backpropagation

- Forward propagation

W_1 : Layer 1's weights

$$y = F(x) := f_n(W_n, f_{n-1}(W_{n-1}, f_{\dots} f_1(W_1, x)))$$

f_1 : Layer 1's output vector　　x : input vector

- Backward propagation
  - Essentially: Chain Rule

Gradient w.r.t. input feature

要要根据

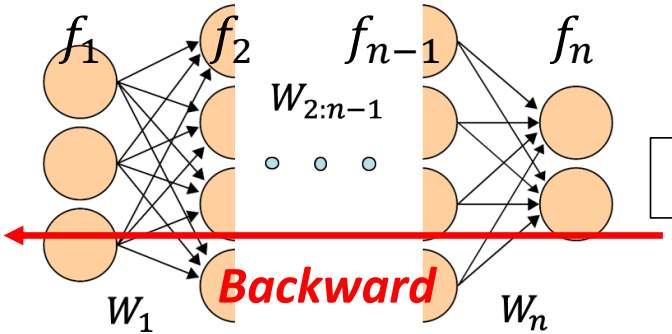
$$\frac{\partial Loss}{\partial W_i} = \frac{\partial L}{\partial f_n} \cdot \frac{\partial f_n}{\partial f_{n-1}} \cdot \frac{\partial f_{n-1}}{\partial f_{n-2}} \cdots \frac{\partial f_i}{\partial W_i}$$

Gradient w.r.t. weight

Loss 调整 W_i

MSE gradient: $d - f_n(\dots)$

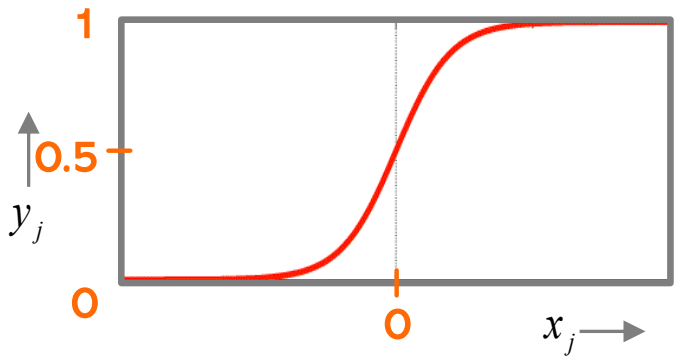
Namely the difference between targets and outputs



$$Loss = L(f_n, d)$$

Gradient within an intermediate layer

- Logistic neuron at layer k



Note: only on this page
Superscript represents layer
Subscript represents element

$$z_j^k = \sum_i f_i^{k-1} w_{ij}^k$$

$$f_j^k = \frac{1}{1 + e^{-z_j^k}}$$

Gradient w.r.t. weight element

$$\frac{\partial f_j^k}{\partial w_{ij}^k} = \frac{\partial f_j^k}{\partial z_j^k} \frac{\partial z_j^k}{\partial w_{ij}^k} = f_j^k \left(1 - f_j^k\right) f_i^{k-1}$$

Gradient w.r.t. input feature

$$\frac{\partial f_j^k}{\partial f_i^{k-1}} = \frac{\partial f_j^k}{\partial z_j^k} \frac{\partial z_j^k}{\partial f_i^{k-1}} = f_j^k \left(1 - f_j^k\right) w_{ij}^k, \qquad \frac{\partial f^k}{\partial f_i^{k-1}} = \sum_j \frac{\partial f_j^k}{\partial f_i^{k-1}}$$

Converting backpropagation into a learning procedure

- The backpropagation algorithm is an efficient way of computing the error derivative dL/dw for every weight on a single training case.

- To get a fully specified learning procedure, we still need to make a lot of other decisions about how to use these error derivatives:

  - **Optimization issues:** How do we use the error derivatives on individual cases to discover a good set of weights?

  - **Generalization issues:** How do we ensure that the learned weights work well for cases we did not see during training?

Outline

- Lecture 2: NN and Back Propagation
  - Linear separation and Least Mean Square (LMS) learning rule
  - Multi-layer perceptron (MLP)
  - Nonlinear activation functions
  - Back propagation
  - Training issues

- Lecture 3: Emergence of convolution
- Lecture 4: Building a CNN model

Overfitting

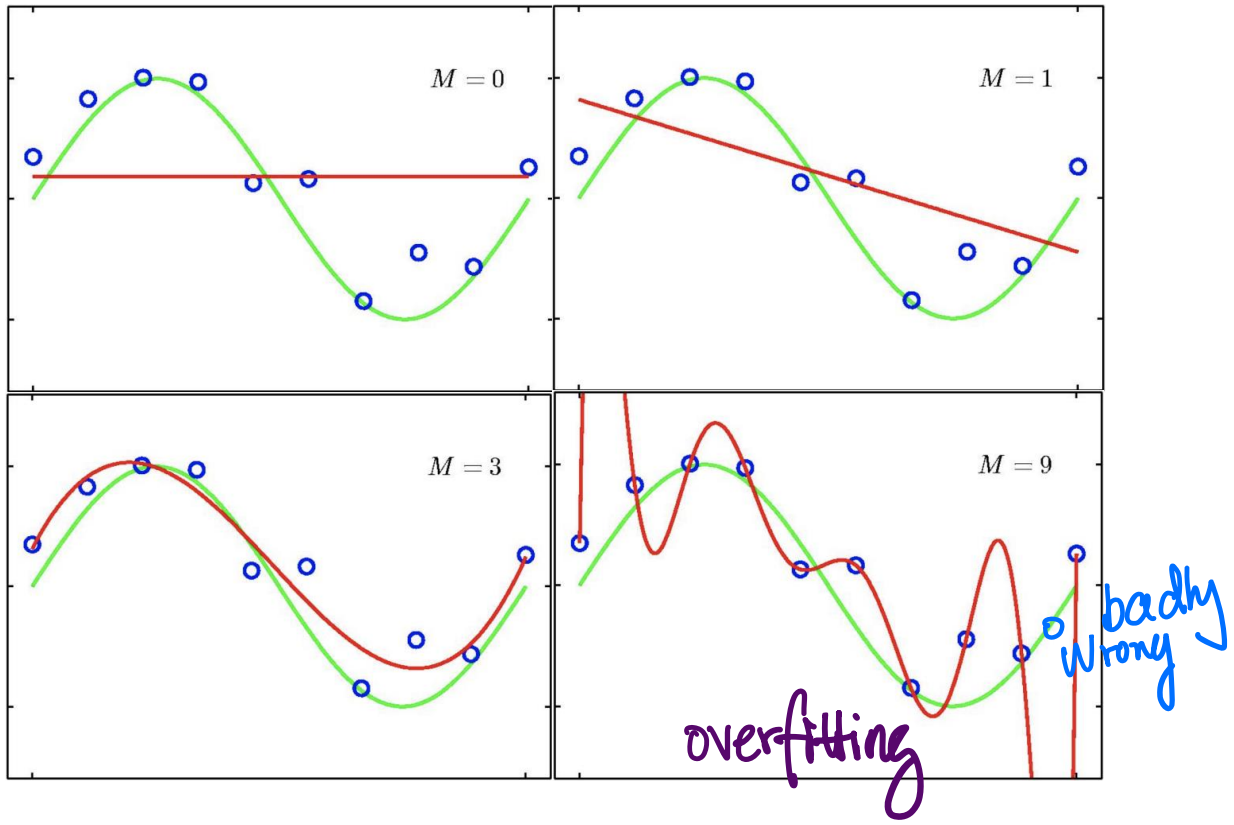
The downside of using powerful models

- The training data contains information about the regularities in the mapping from input to output. But it also contains two types of noises.
  - The target values may be unreliable (usually only a minor worry).
  - There is sampling error. There will be accidental regularities just because of the particular training cases that were chosen.
- When we fit the model, it cannot tell which regularities are real and which are caused by sampling error.
  - So it fits both kinds of regularity.
  - If the model is very flexible it can model the sampling error really well. This is a disaster.

会把training data里有的但future data里不会有的pattern也train到去

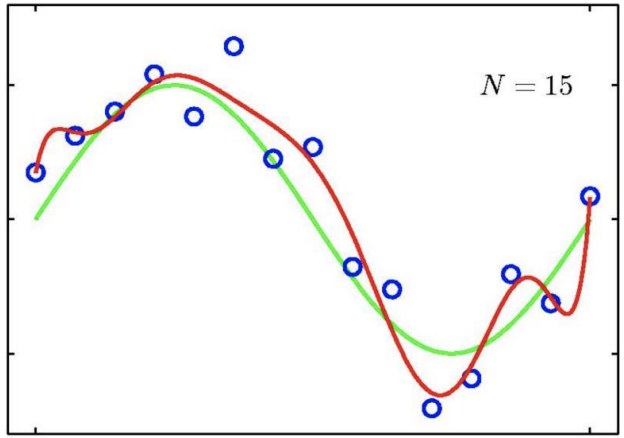
Overfitting Example: Which is the best fit?

- The green curve is the true function.
- The data points are uniform in X but have noise in Y



Ways to reduce overfitting

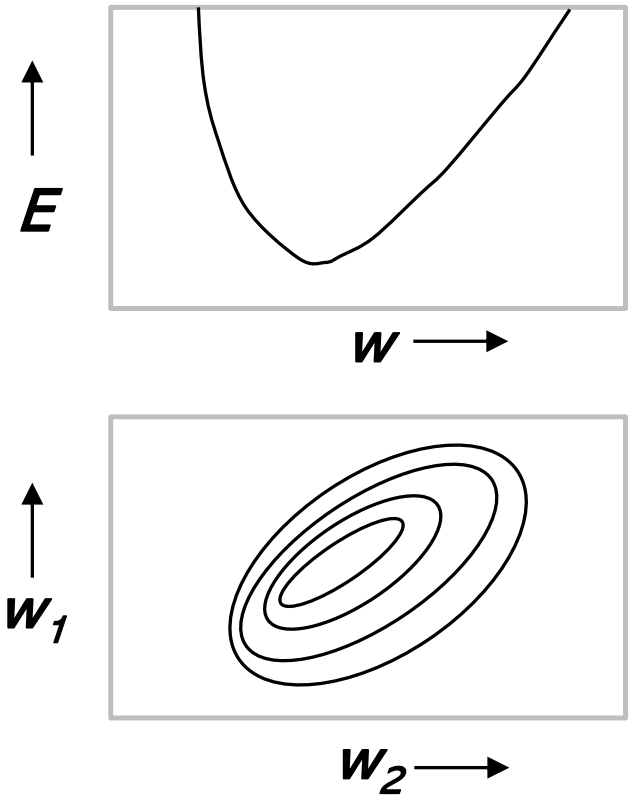
- Many different methods have been developed.
  - L-norm regularization
  - Weight-decay
  - Using a validation set
    - Early stopping
  - Model averaging
  - Dropout
  - Generative pre-training



- Many of these methods will be introduced in later lectures.

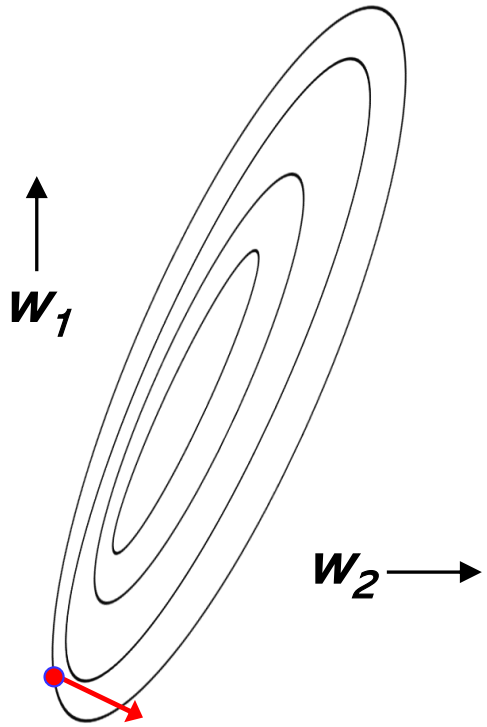
The error surface in extended weight space

- The error surface lies in a space with a horizontal axis for each weight and one vertical axis for the error.
  - For a linear neuron with a squared error, it is a quadratic bowl.
  - Vertical cross-sections are parabolas.
  - Horizontal cross-sections are ellipses (contour lines).
- For multi-layer, non-linear nets the error surface is much more complicated.



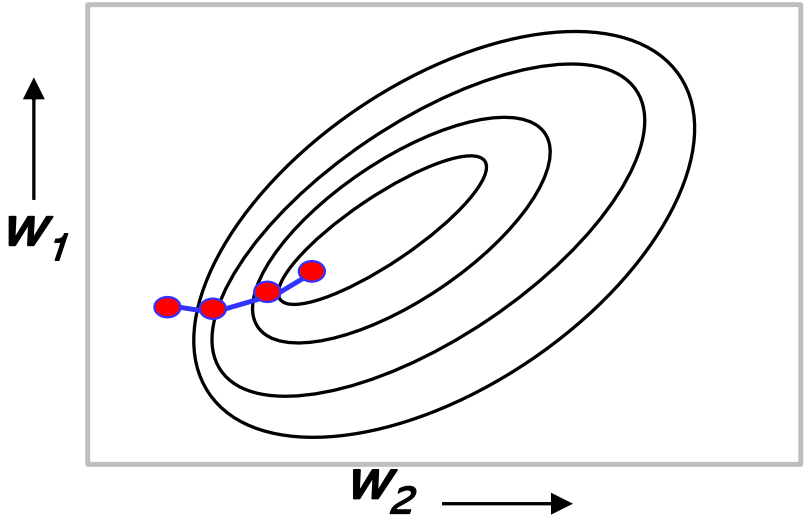
Optimization issues in using weight derivatives

- How often to update the weights
  - Online: after each training case. too costly
  - Full batch: after a full sweep through the training data.
  - Mini-batch: after each of a small sample set of training cases.
- How much to update
  - Use a fixed learning rate?
  - Adapt the global learning rate?
  - Adapt the learning rate on each connection separately?
- Don't use the steepest descent?

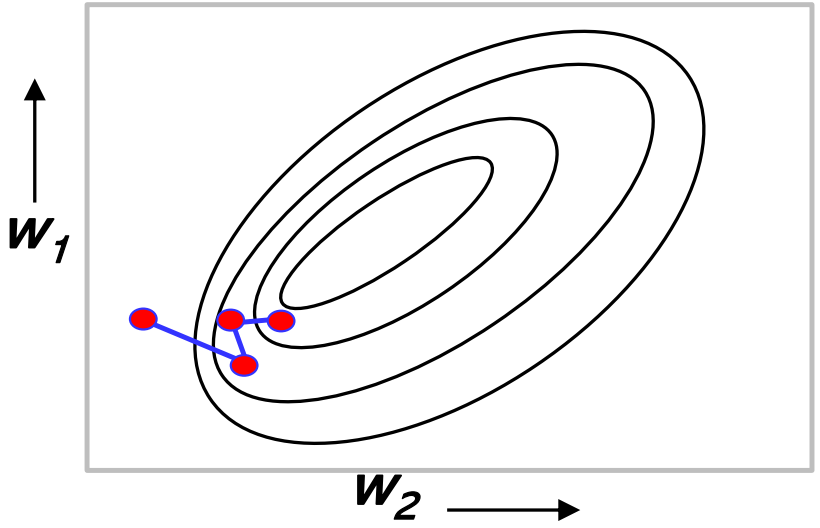


Full batch vs. mini-batch or online learning

- Full batch follows the steepest descent on the error surface.
  - This travels perpendicular to the contour lines.

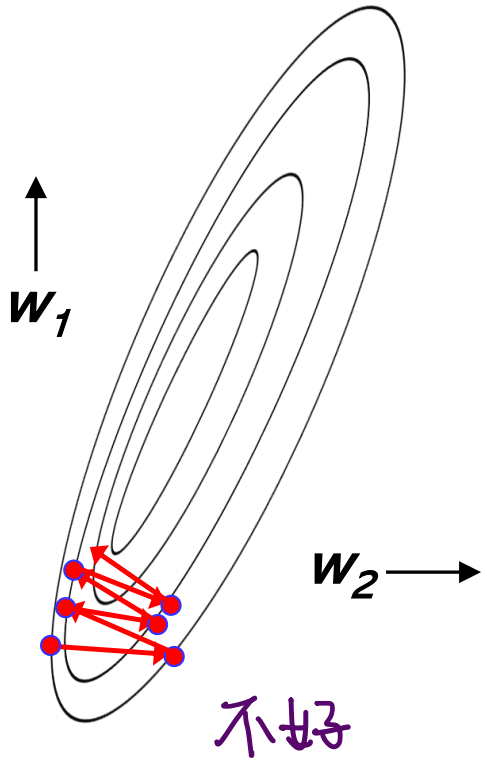


- Mini-batch or online learning zig-zags around the direction of steepest descent.



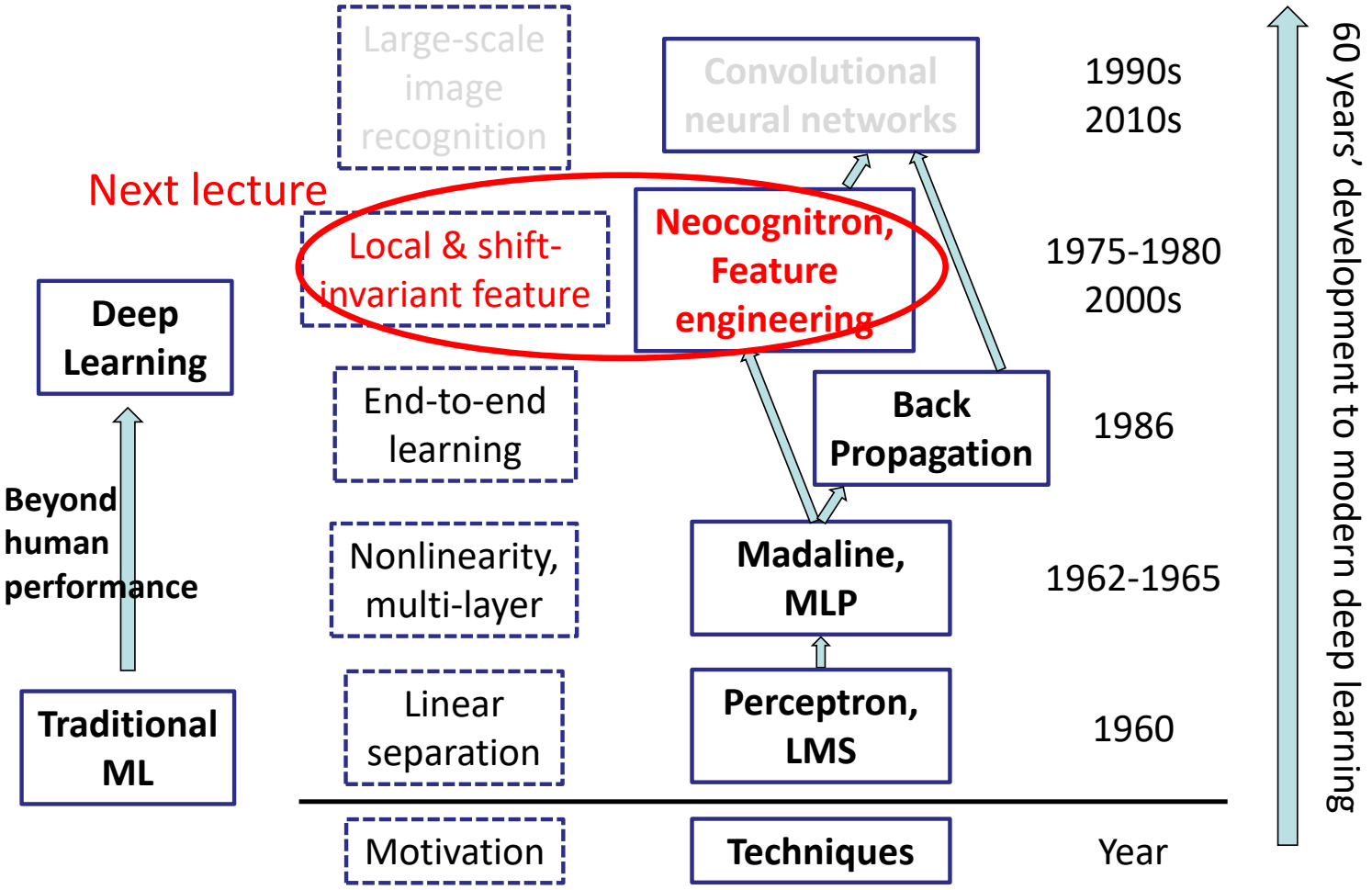
Why learning can be slow

- If the ellipse is very elongated, the direction of the steepest descent can deviate from the direction towards the minimum!
  - The red gradient vector has a large component along the short axis of the ellipse and a small component along the long axis of the ellipse.
  - This is not what we want.



Optimization techniques mitigating these issues will be introduced later

In this lecture, we learned:



Reading Material

- Widrow, Bernard, and Michael A. Lehr. "30 years of adaptive neural networks: perceptron, madaline, and backpropagation." *Proceedings of the IEEE* 78.9 (1990): 1415-1442.

30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation

BERNARD WIDROW, FELLOW, IEEE, AND MICHAEL A. LEHR

Fundamental developments in feedforward artificial neural networks from the past thirty years are reviewed. The central theme of this paper is a description of the history, origination, operating characteristics, and basic theory of several supervised neural network training algorithms including the Perceptron rule, the LMS algorithm, three Madaline rules, and the backpropagation technique. These methods were developed independently, but with the perspective of history they can all be related to each other. The concept underlying these algorithms is the "minimal disturbance principle," which suggests that during training it is advisable to inject new information into a network in a manner that disturbs stored information to the smallest extent possible.

I. Introduction

This year marks the 30th anniversary of the Perceptron rule and the LMS algorithm, two early rules for training adaptive elements. Both algorithms were first published in 1960. In the years following these discoveries, many new techniques have been developed in the field of neural networks, and the discipline is growing rapidly. One early development was Steinbuch's Learning Matrix [1], a pattern recognition machine based on linear discriminant functions. At the same time, Widrow and his students devised Madaline Rule I (MRI), the earliest popular learning rule for neural networks with multiple adaptive elements [2]. Other early work included the "mode-seeking" technique of Stark, Okajima, and Whipple [3]. This was probably the first example of competitive learning in the literature, though it could be argued that earlier work by Rosenblatt on "spontaneous learning" [4], [5] deserves this distinction. Further pioneering work on competitive learning and self-organization was performed in the 1970s by von der Malsburg [6] and Grossberg [7]. Fukushima explored related ideas with his biologically inspired Cognitron and Neocognitron models [8], [9].

Manuscript received September 12, 1989; revised April 13, 1990. This work was sponsored by SDIO Innovative Science and Technology office and managed by ONR under contract no. N00014-86-K-0718, by the Dept. of the Army Belvoir RD&E Center under contracts no. DAAK 70-87-P-3134 and no. DAAK-70-89-K-0001, by a grant from the Lockheed Missiles and Space Co., by NASA under contract no. NCA2-389, and by Rome Air Development Center under contract no. F30602-88-D-0025, subcontract no. E-21-T22-S1.

The authors are with the Information Systems Laboratory, Department of Electrical Engineering, Stanford University, Stanford, CA 94305-4055, USA.

IEEE Log Number 9038824.

Widrow devised a reinforcement learning algorithm called "punish/reward" or "bootstrapping" [10], [11] in the mid-1960s. This can be used to solve problems when uncertainty about the error signal causes supervised training methods to be impractical. A related reinforcement learning approach was later explored in a classic paper by Barto, Sutton, and Anderson on the "credit assignment" problem [12]. Barto *et al.*'s technique is also somewhat reminiscent of Albus's adaptive CMAC, a distributed table-look-up system based on models of human memory [13], [14].

In the 1970s Grossberg developed his Adaptive Resonance Theory (ART), a number of novel hypotheses about the underlying principles governing biological neural systems [15]. These ideas served as the basis for later work by Carpenter and Grossberg involving three classes of ART architectures: ART 1 [16], ART 2 [17], and ART 3 [18]. These are self-organizing neural implementations of pattern clustering algorithms. Other important theory on self-organizing systems was pioneered by Kohonen with his work on feature maps [19], [20].

In the early 1980s, Hopfield and others introduced outer product rules as well as equivalent approaches based on the early work of Hebb [21] for training a class of recurrent (signal feedback) networks now called Hopfield models [22], [23]. More recently, Kosko extended some of the ideas of Hopfield and Grossberg to develop his adaptive Bidirectional Associative Memory (BAM) [24], a network model employing differential as well as Hebbian and competitive learning laws. Other significant models from the past decade include probabilistic ones such as Hinton, Sejnowski, and Ackley's Boltzmann Machine [25], [26] which, to oversimplify, is a Hopfield model that settles into solutions by a simulated annealing process governed by Boltzmann statistics. The Boltzmann Machine is trained by a clever two-phase Hebbian-based technique.

While these developments were taking place, adaptive systems research at Stanford traveled an independent path. After devising their Madaline I rule, Widrow and his students developed uses for the Adaline and Madaline. Early applications included, among others, speech and pattern recognition [27], weather forecasting [28], and adaptive controls [29]. Work then switched to adaptive filtering and adaptive signal processing [30] after attempts to develop learning rules for networks with multiple adaptive layers were unsuccessful. Adaptive signal processing proved to

0018-9219/90/0900-1415$01.00 © 1990 IEEE