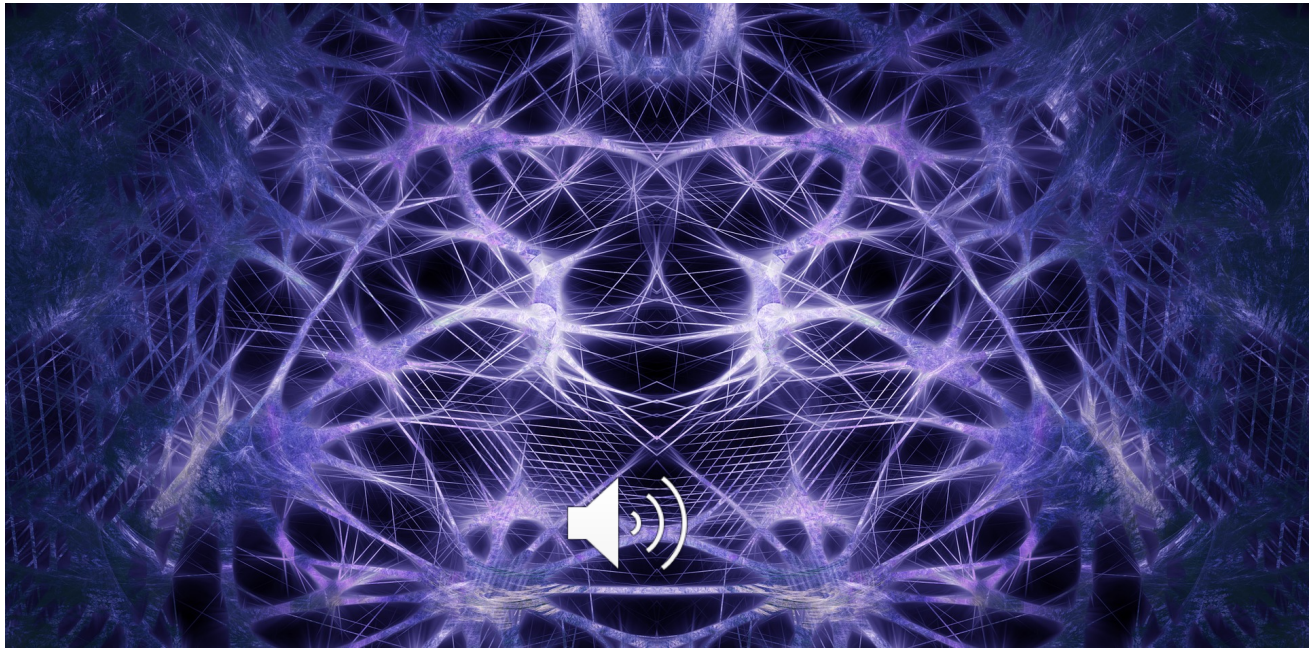
6.034 Artificial Intelligence: Lecture 14
Neural networks I

Robert C. Berwick

October 7, 2020



AI Methods

- **Problem solving**
  - **G+T, *search, optimal search, games, constraint satisfaction***
- **Inference**
  - ***rule-based systems, Bayesian inference***
- **Machine learning**
  - ***k-nearest neighbors, id trees, neural nets*, deep neural nets, *support vector machines*, genetic algorithms, near miss/one-shot**
- **Communication, perception, action**
  - **natural language processing, vision, robotics**

Model the brain as a neural net: an old idea!

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The Bulletin of Mathematical Biophysics, 5(4), 115–133. http://doi.org/10.1007/BF02478259

Warren McCullough

Walter Pitts

BULLETIN OF
ATHEMATICAL BIOPHYSICS
VOLUME 5, 1943

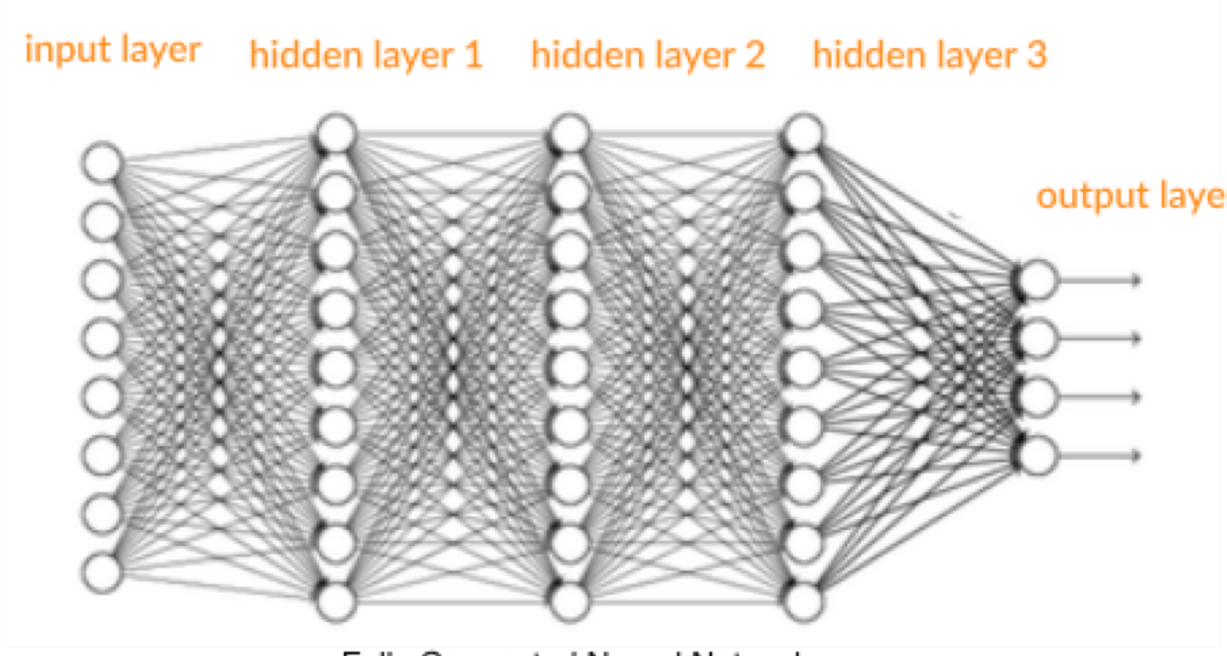
A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

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.

A multi-layer, fully connected neural net



Fully-Connected Neural Network

Why did it take so long to get these working?

Menu for today

- ❑ Models of problem solving
- ❑ Models of learning → Biologically inspired
- ❑ Neural Nets

  Naive neurobiology

  Hill climbing

  Threshold trick

  Sigmoid trick

  World's simplest neural net

  The world's next most simplest net – is it practical?

Marvin Minsky

Publication No.: 9438

MINSKY, Marvin Lee. THEORY OF NEURAL-ANALOG REINFORCEMENT SYSTEMS AND ITS APPLICATION TO THE BRAIN-MODEL PROBLEM.

Princeton University, Ph.D., 1954
Mathematics

Please Note: Find page numbered as 1–8 at the end of film copy.

University Microfilms, Inc., Ann Arbor, Michigan

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

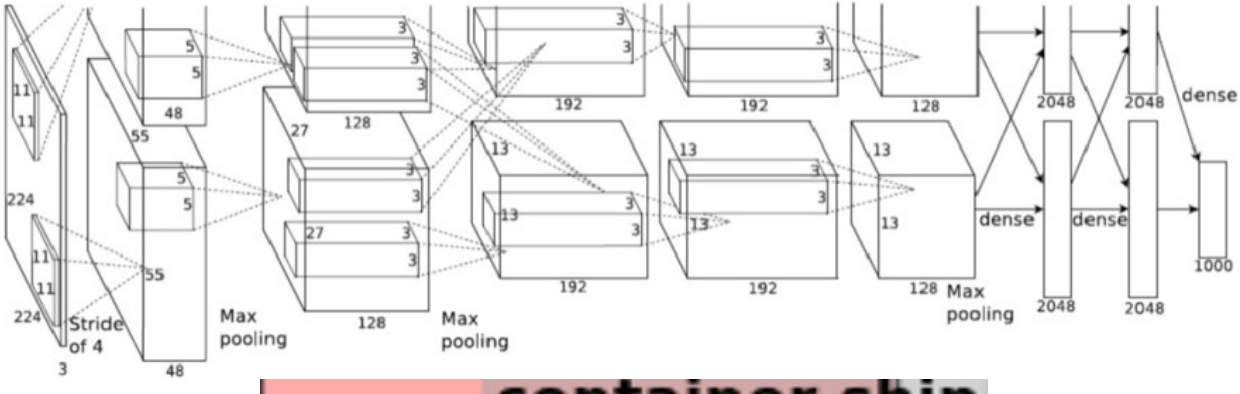
1 Introduction

Current approaches to object recognition make essential use of machine learning methods. To improve their performance, we can collect larger datasets, learn more powerful models, and use better techniques for preventing overfitting. Until recently, datasets of labeled images were relatively small — on the order of tens of thousands of images (e.g., NORB [16], Caltech-101/256 [8, 9], and CIFAR-10/100 [12]). Simple recognition tasks can be solved quite well with datasets of this size, especially if they are augmented with label-preserving transformations. For example, the current-best error rate on the MNIST digit-recognition task (<0.3%) approaches human performance [4]. But objects in realistic settings exhibit considerable variability, so to learn to recognize them it is necessary to use much larger training sets. And indeed, the shortcomings of small image datasets have been widely recognized (e.g., Pinto et al. [21]), but it has only recently become possible to collect labeled datasets with millions of images. The new larger datasets include LabelMe [23], which consists of hundreds of thousands of fully-segmented images, and ImageNet [6], which consists of over 15 million labeled high-resolution images in over 22,000 categories.

To learn about thousands of objects from millions of images, we need a model with a large learning capacity. However, the immense complexity of the object recognition task means that this problem cannot be specified even by a dataset as large as ImageNet, so our model should also have lots of prior knowledge to compensate for all the data we don't have. Convolutional neural networks (CNNs) constitute one such class of models [16, 11, 13, 18, 15, 22, 26]. Their capacity can be controlled by varying their depth and breadth, and they also make strong and mostly correct assumptions about the nature of images (namely, stationarity of statistics and locality of pixel dependencies). Thus, compared to standard feedforward neural networks with similarly-sized layers, CNNs have much fewer connections and parameters and so they are easier to train, while their theoretically-best performance is likely to be only slightly worse.

Geoffrey Hinton, The persistent

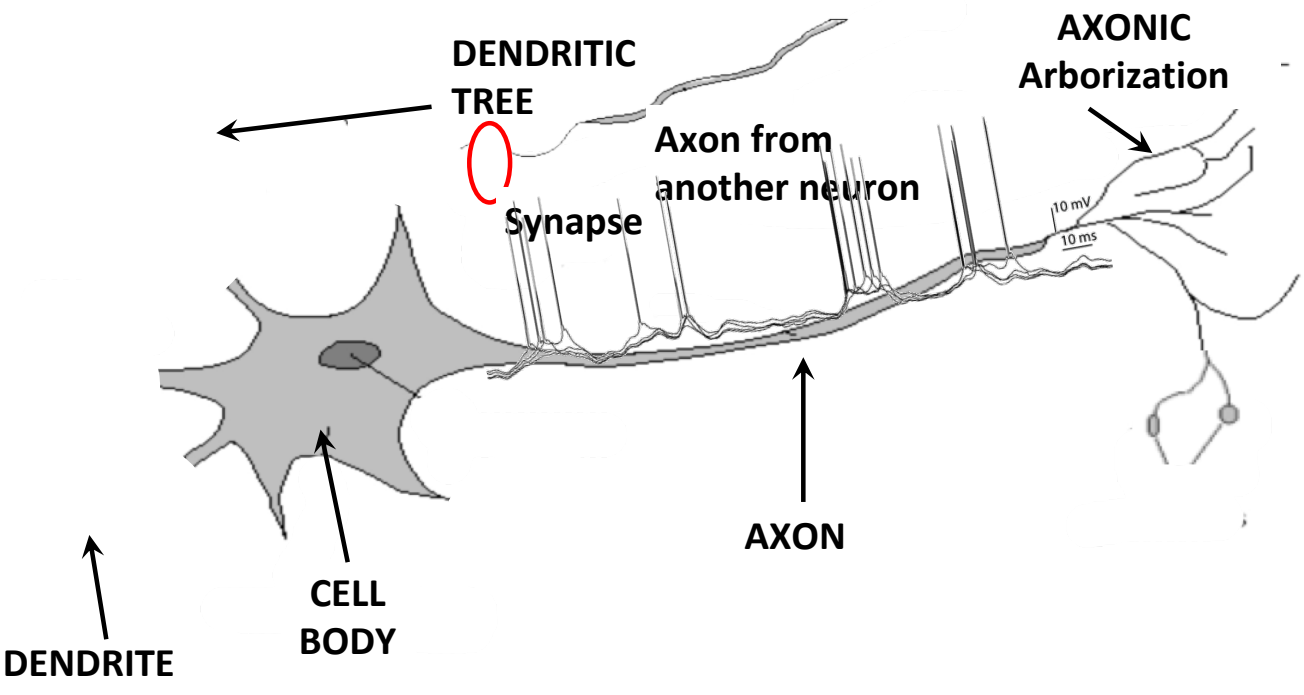
INPUT



OUTPUT

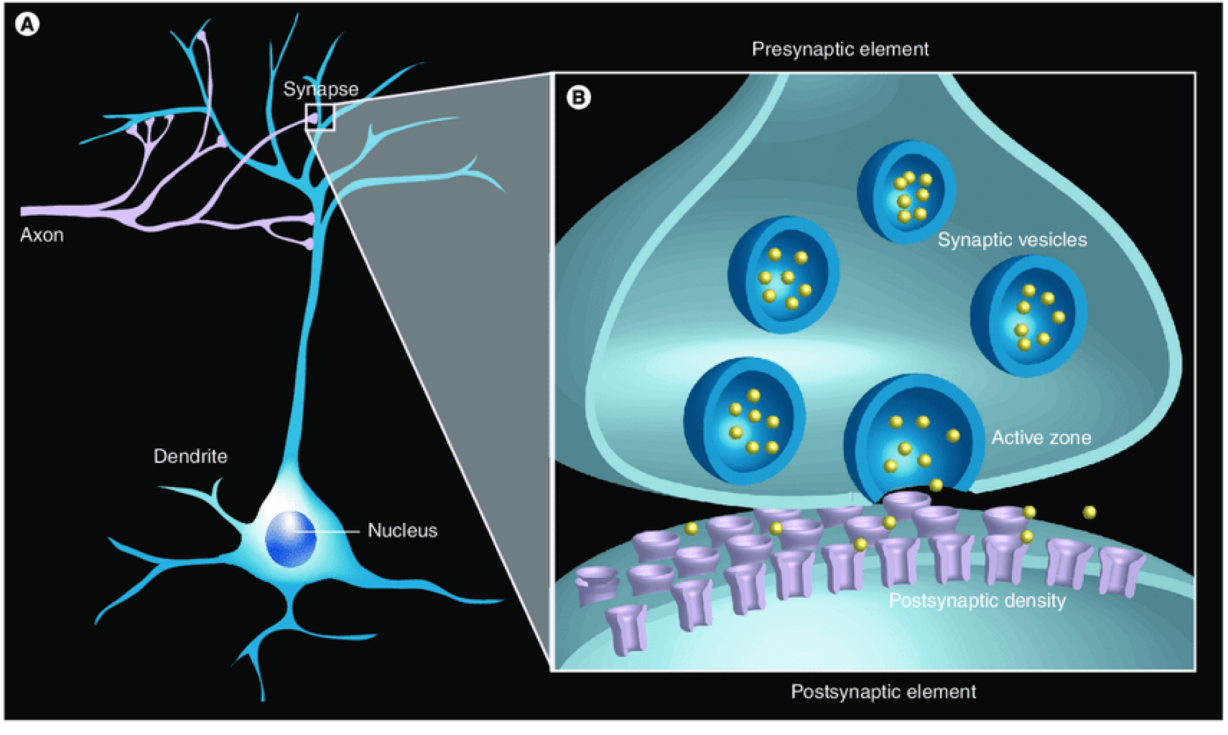

mite
container ship
motor scooter
leopard
mite
black widow
cockroach
tick
starfish
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
go-kart
moped
bumper car
golfcart
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
mushroom
cherry
Madagascar cat
convertible
grille
pickup
beach wagon
fire engine
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
dalmatian
grape
elderberry
ffordshire bullterrier
currant
squirrel monkey
spider monkey
titi
indri
howler monkey

A neuron


DENDRITIC
TREE
Synapse
Axon from
another neuron
AXONIC
Arborization
10 mV
10 ms
AXON
CELL
BODY
DENDRITE

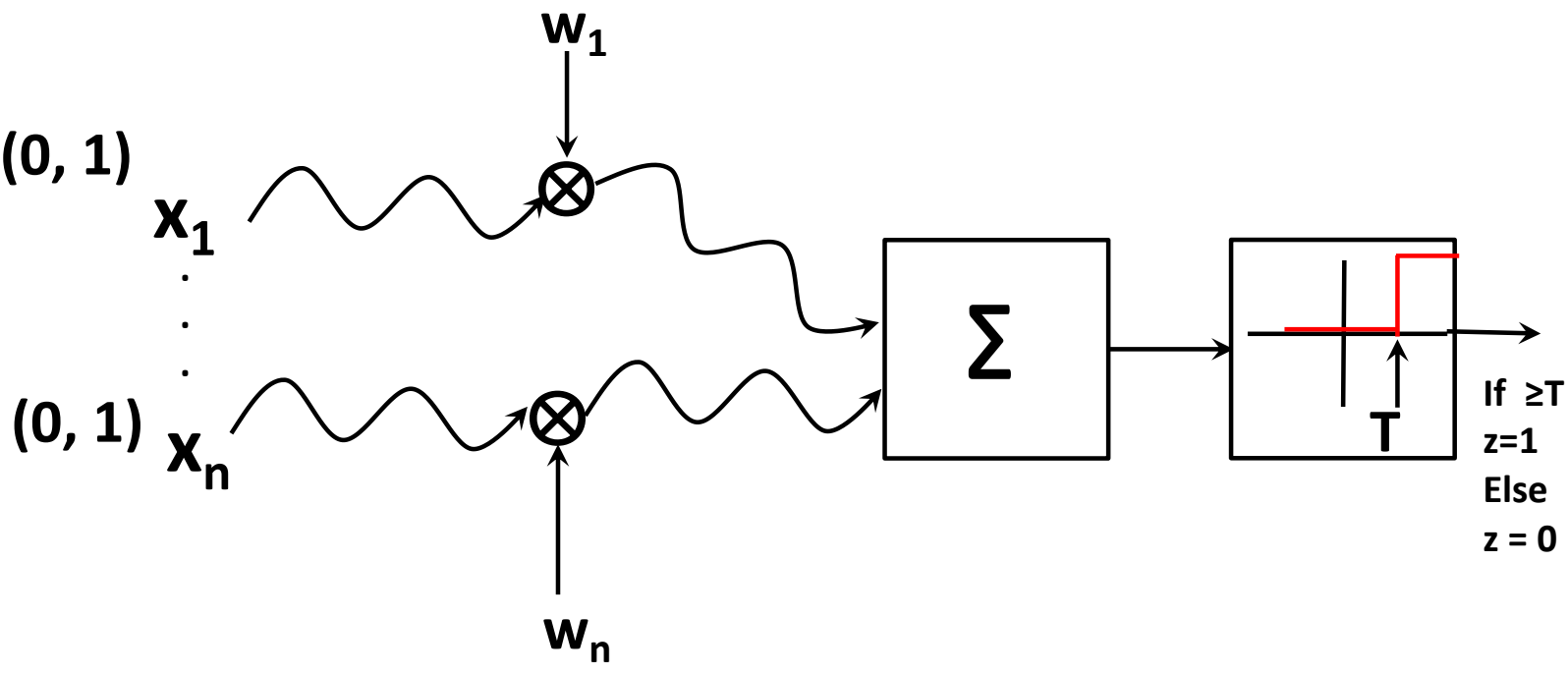
Dendrite-axon synapse connection - blowup

Axon



Dendrite

How to model?



What are we modeling?

What are we modeling?

1. **All or none**
2. **Cumulative influence**
3. **Synaptic weights**

What are we not modeling?

1. **Refractory period**
2. **Axonal bifurcation**
3. **Time patterns**

The simplest neural model
McCulloch-Pitts, 1943

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The Bulletin of Mathematical Biophysics, 5(4), 115–133. http://doi.org/10.1007/BF02478259

1. The activity of the neuron is an "all-or-none" process
2. A certain fixed number of synapses must be excited within the period of latent addition in order to excite a neuron at any time, and this number is independent of previous activity and position on the neuron
3. The only significant delay within the nervous system is synaptic delay
4. The activity of any inhibitory synapse absolutely prevents excitation of the neuron at that time
5. The structure of the net does not change with time

BULLETIN OF
MATHEMATICAL BIOPHYSICS
VOLUME 5, 1943

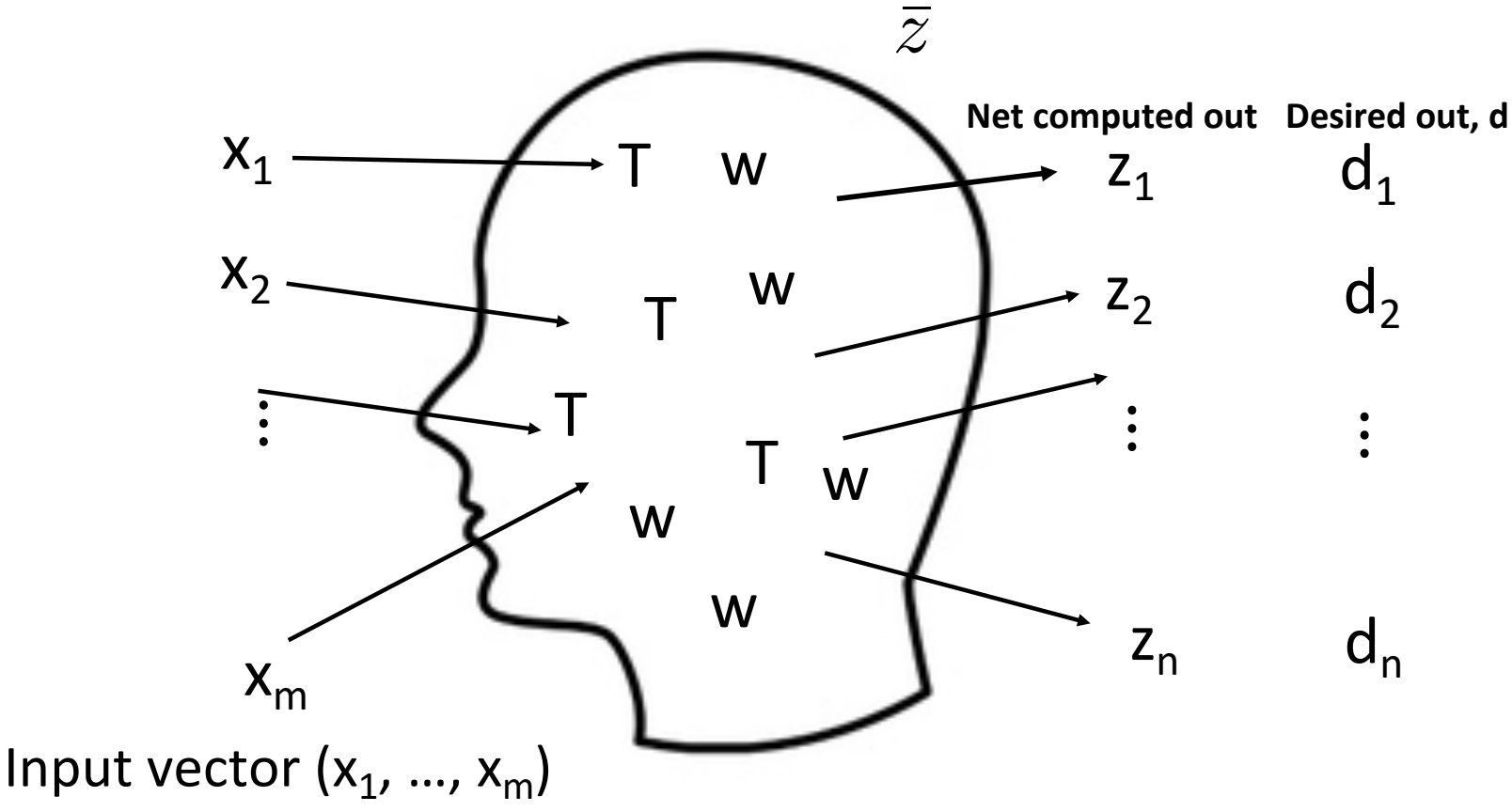
A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

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

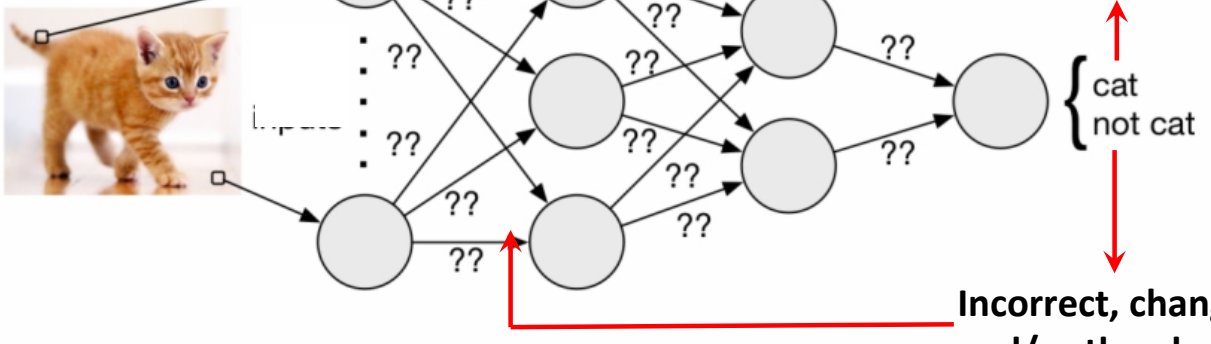
Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; 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.

Brain as transducer – or a function approximator?



How to wire up “neurons,” weights & thresholds?

Simplest idea: error-driven reinforcement model



Desired output OK, do nothing;
(or, perhaps, reinforce existing weights)
Get next input

Incorrect, change weight(s) & and/or thresholds/topology
Error-driven learning
Get net input

Problem: what weights to change if performance not OK?
Who gets credit/blame?

Publication No.: 9438

MINSKY, Marvin Lee. THEORY OF NEURAL-ANALOG REINFORCEMENT SYSTEMS AND ITS APPLICATION TO THE BRAIN-MODEL PROBLEM.

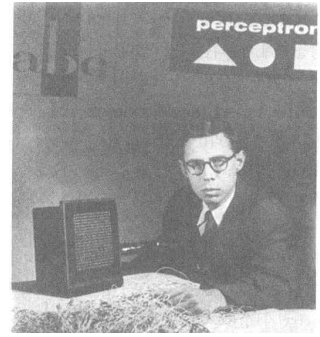
Princeton University, Ph.D., 1954
Mathematics

Please Note: Find page numbered as 1–8 at the end of film copy.

University Microfilms, Inc., Ann Arbor, Michigan

Simplest implementation of McCullough-Pitts: the Perceptron

(output is linear combination of weights)



Frank Rosenblatt, 1958

$x_1 \longrightarrow w_1x_1$

$x_2 \longrightarrow w_2x_2$

$x_3 \longrightarrow w_3x_3$

$\vdots$

$x_n \longrightarrow w_nx_n$

Summer Σ

Threshold T

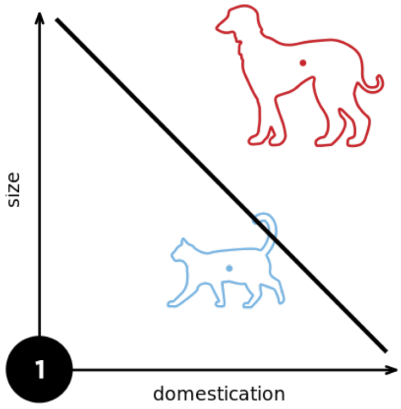
Output z

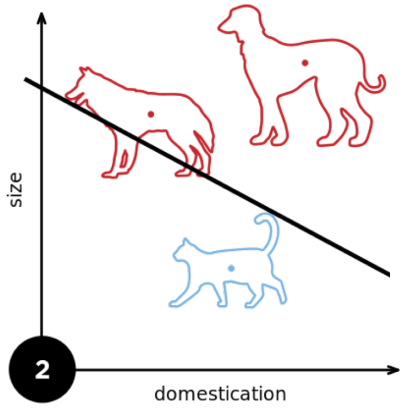
$$z = f(\boldsymbol{x} \cdot \boldsymbol{w}) \geq T = 1 \text{ a cat}$$

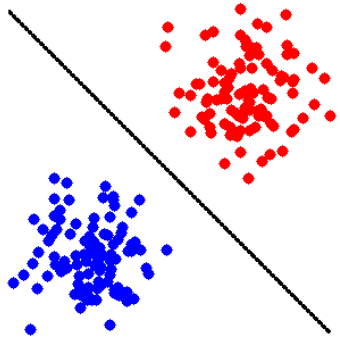
$$< T = 0 \text{ not a cat}$$

<u>Single</u> layer of weights – output is a linear function of weights (line/plane/hyperplane)
Makes it easy to figure out which weights to change in response to difference between desired output, d, and net computed output, z

Train by providing s examples (x_t, d_t) for t=1, …, s, where each x is an *n*-dimensional vector; & then altering initial random weights w_i

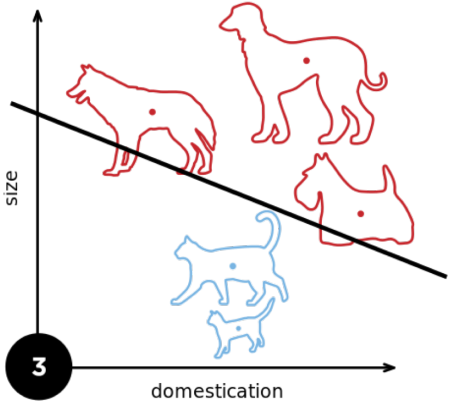


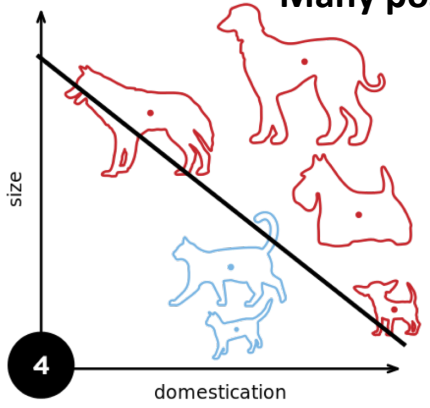


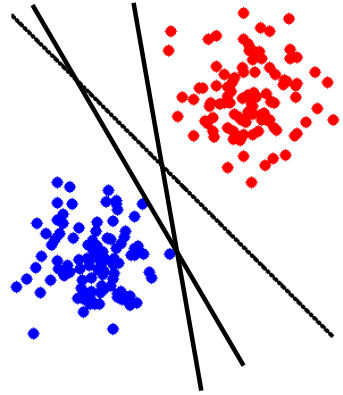


Linear separation of groups

Many possible (∞) linear separation of groups







From Wikipedia

Perceptron learning rule is simple: "credit" or "blame" is linked directly to output by changing weights proportional to error

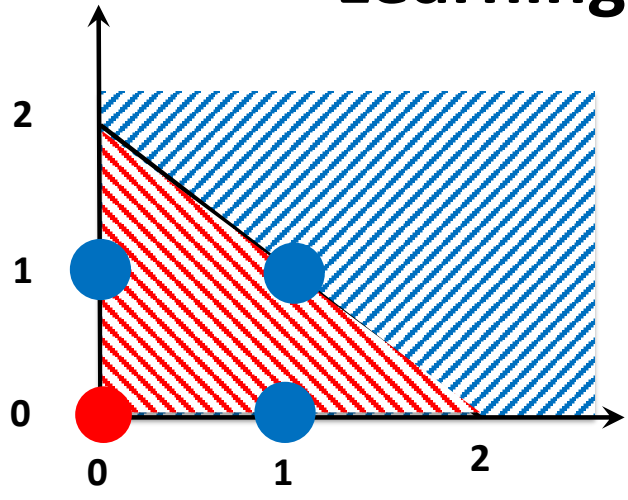
$$\{(\boldsymbol{x}_1, d_1), \ldots, (\boldsymbol{x_s}, d_s)\}\,; \text{training set}$$

Error: (desired–output z from net)
"backpropagate" error to correct weights

$$w_i(t+1) = w(t) + r \cdot (d - z(t))x_i, \forall \text{ features } i$$

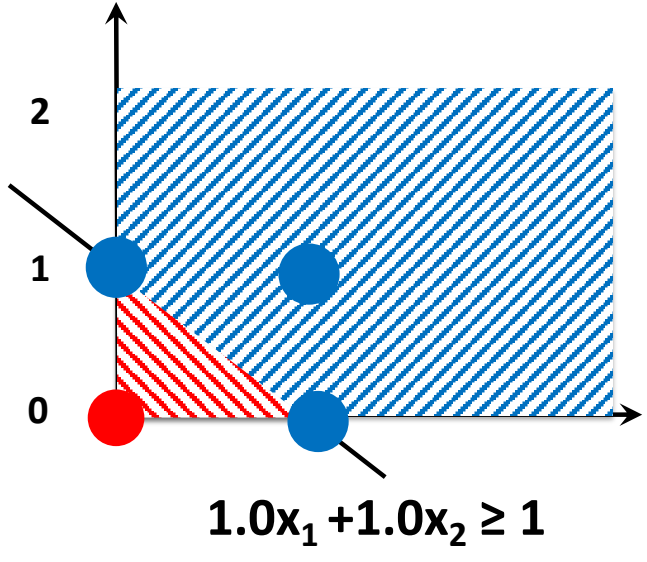
This is error-driven hill-climbing in space of weights
Here *r* is some "learning rate" (alters stepsize)

Learning logical OR



Start: ½ x_1 + ½ x_2 ≥ 1

Threshold= 1; r= ½



1.0x_1 +1.0x_2 ≥ 1

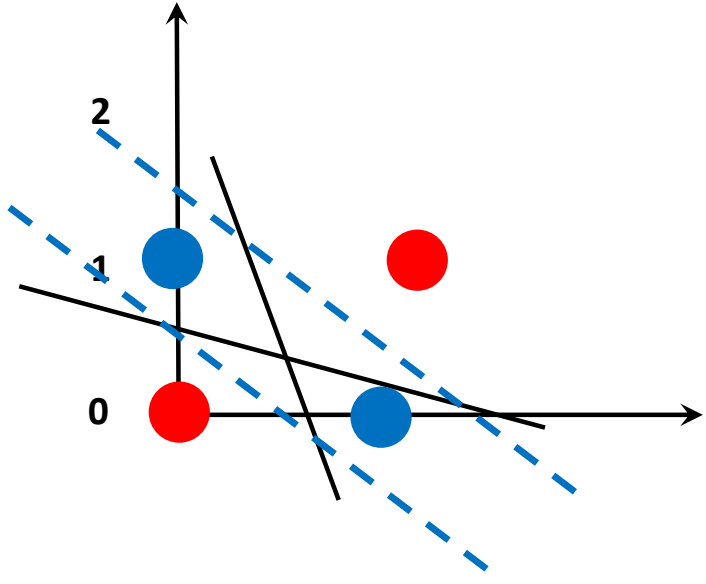
x_1	x_2	output	z	desired	d-z	r(d–z)	Δw		New w_i	
0	0	0	0	0	0	0	0	0	½	½
0	1	½	0	1	1	½	0	½	½	1
1	0	½	0	1	1	½	½	0	1	1
1	1	1	1	1	0	0	0	0	1	1

= output 1

= output 0

$$w_i(t+1) = w(t) + r \cdot (d - z(t))x_i, \forall \text{ features } i$$

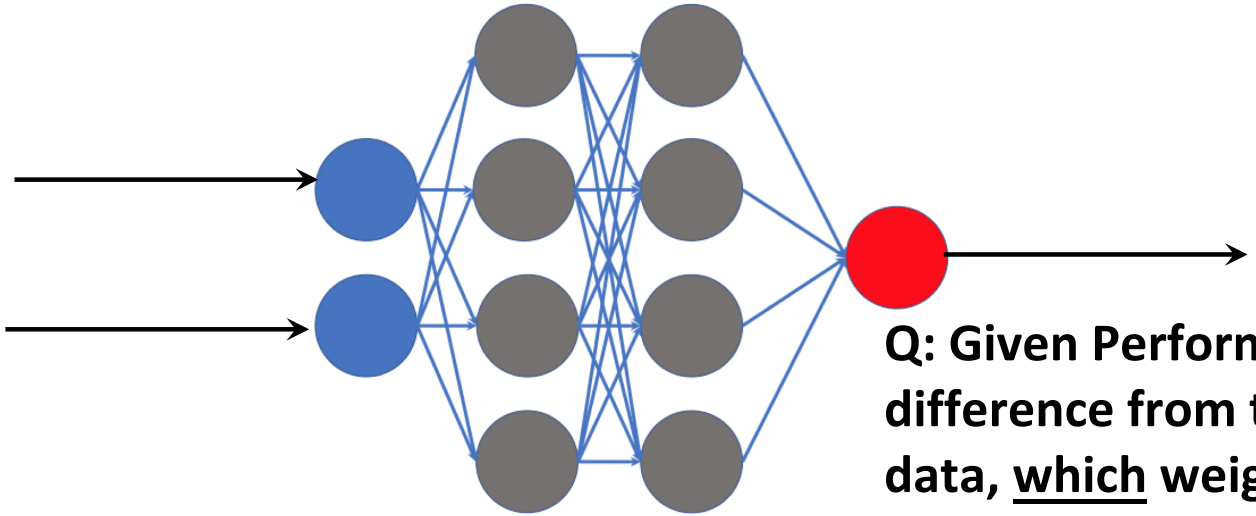
Single-layer net can only do linear separation
No single linear cut to separate boundaries for logical "exclusive or" (Xor)



Note that if we had two separate cuts (say by using two layers), we could do it…

We need a way to model nonlinear patterns and curves….

And we seem to have all the components for this sort of neural net…except…

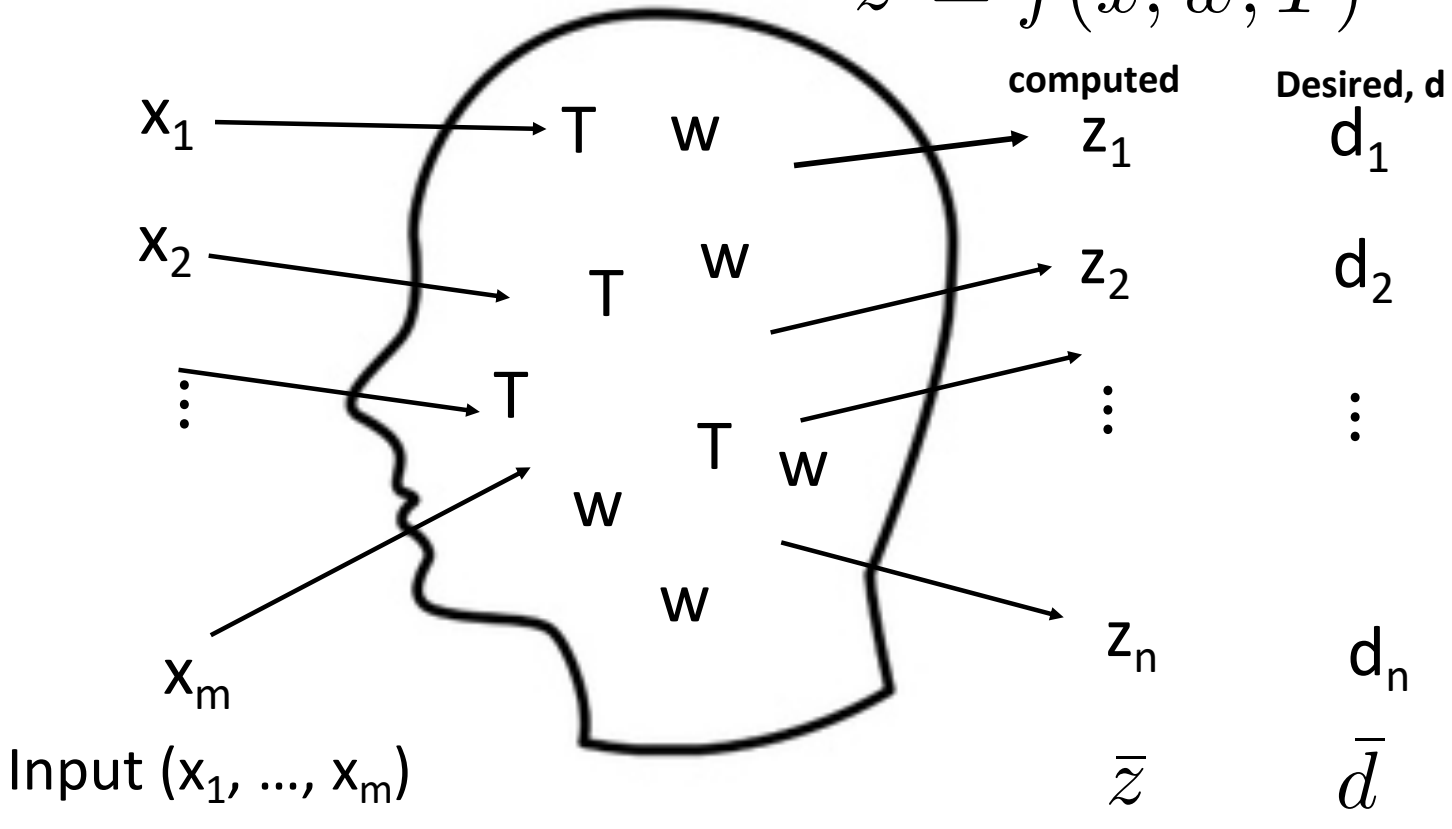


Q: Given Performance difference from training data, which weight internally do we tweak and by how much?

Answer: backpropagation of error to change weights, but a more sophisticated kind than in perceptron

Brain as transducer – or a function approximator?

$$\bar{z} = f(\bar{x}, \bar{w}, \bar{T})$$

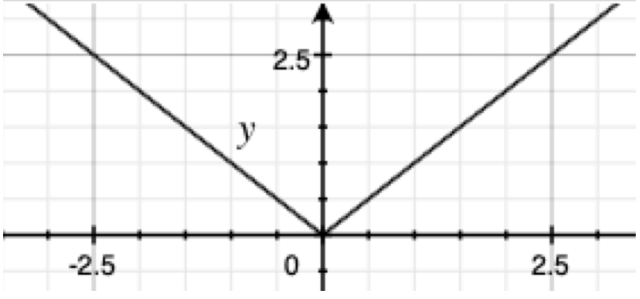


How to measure Performance?

Measuring Performance–for hill climbing search in general neural networks

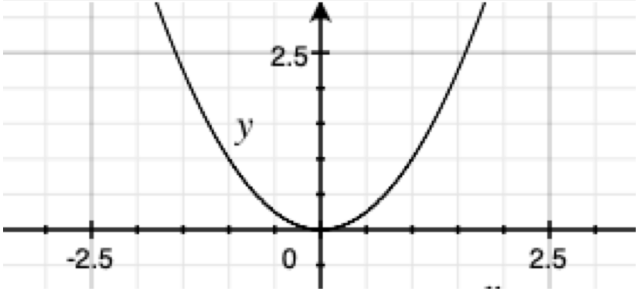
$$P = \left\| \bar{d} - \bar{z} \right\|$$

Problems?



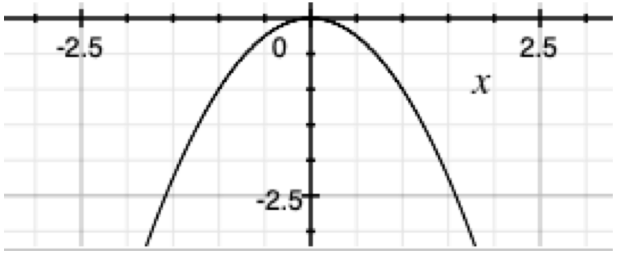
$$P = \left\| \bar{d} - \bar{z} \right\|^2$$

Smoother!, descent to minimum @ 0



$$P = -\left\| \bar{d} - \bar{z} \right\|^2$$

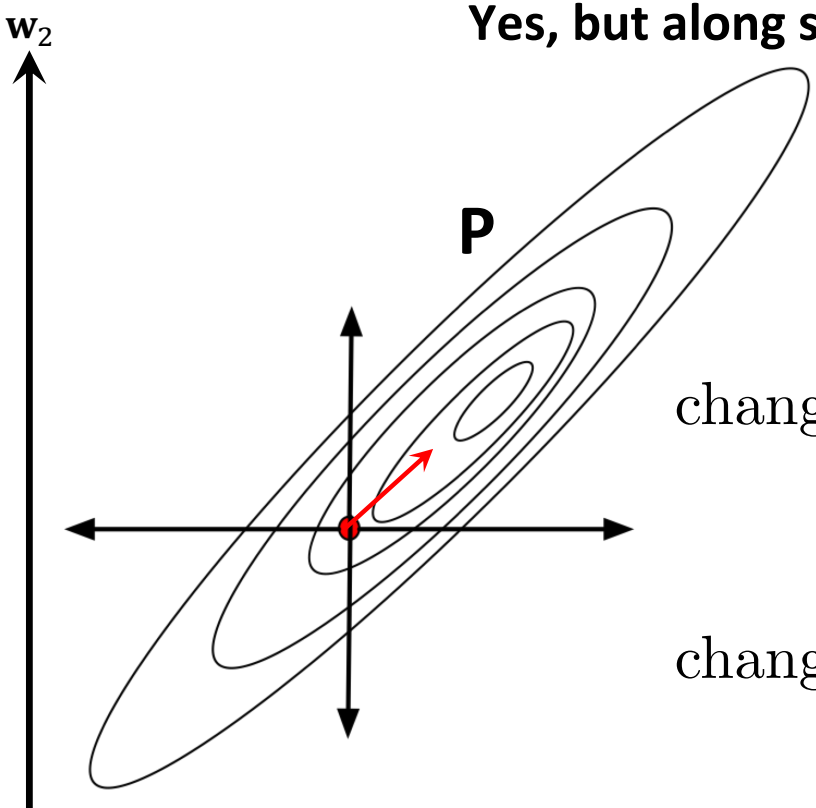
Smoother!, ascent to maximum @ 0



Improving performance by hill climbing

Use Hill-Climbing?

Yes, but along slope, i.e., gradient!



$$\frac{\partial P}{\partial w_1} \qquad \frac{\partial P}{\partial w_2}$$

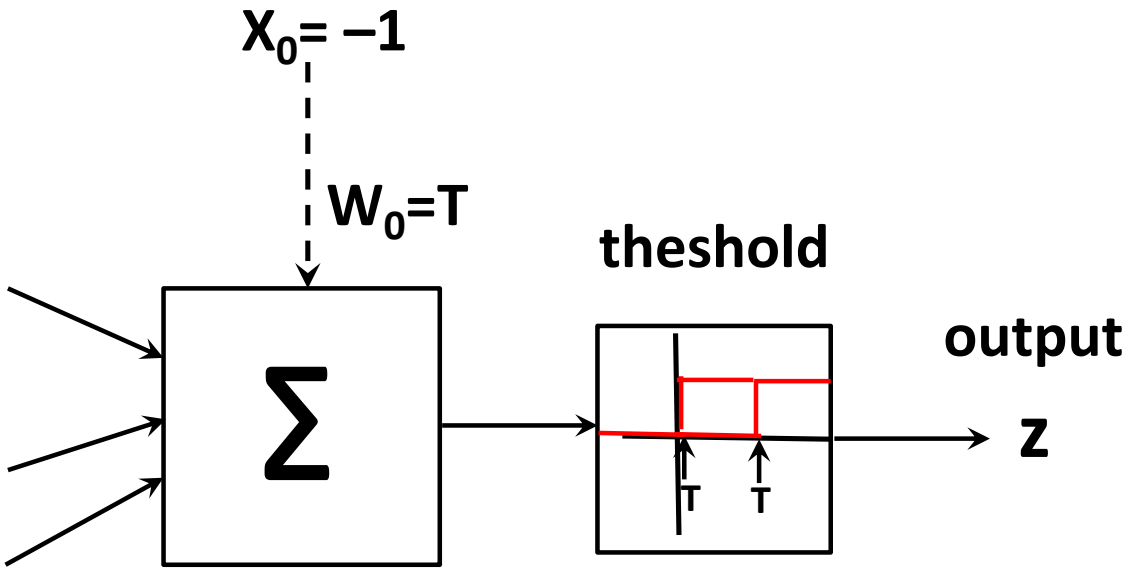
$$\text{change } \triangle \boldsymbol{w} = \left(\frac{\partial P}{\partial w_1} i + \frac{\partial P}{\partial w_2} j \right)$$

$$\text{change } \triangle \boldsymbol{w} = r \left(\frac{\partial P}{\partial w_1} i + \frac{\partial P}{\partial w_2} j \right)$$

$$\text{change } \triangle \boldsymbol{w} = r \nabla \boldsymbol{w}$$

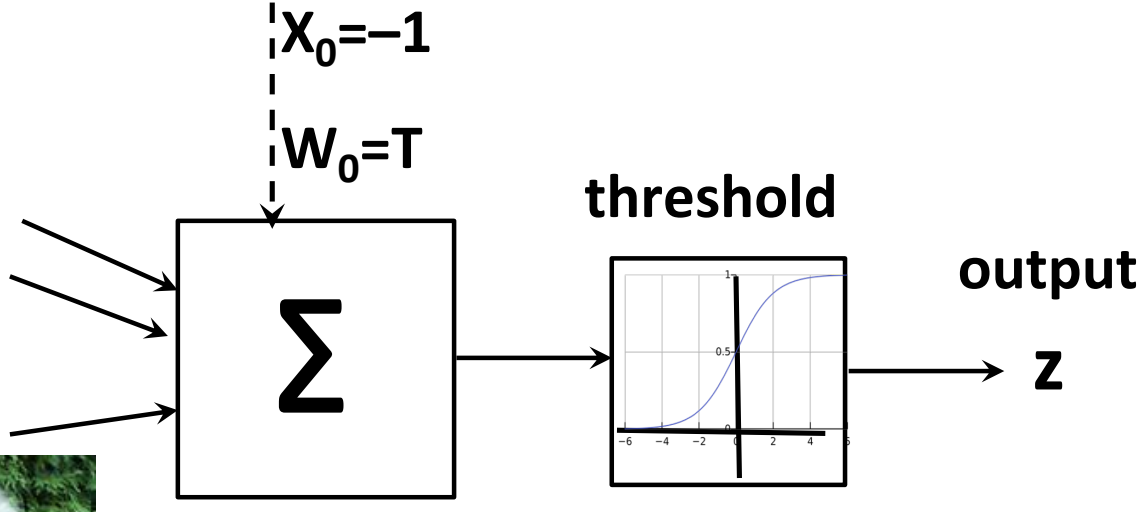
We will need 2 more hacks + genius insight…

Hack #1: Push threshold T left to 0 by adding "bias" input & weight



(same hack used in Perceptron with Threshold)

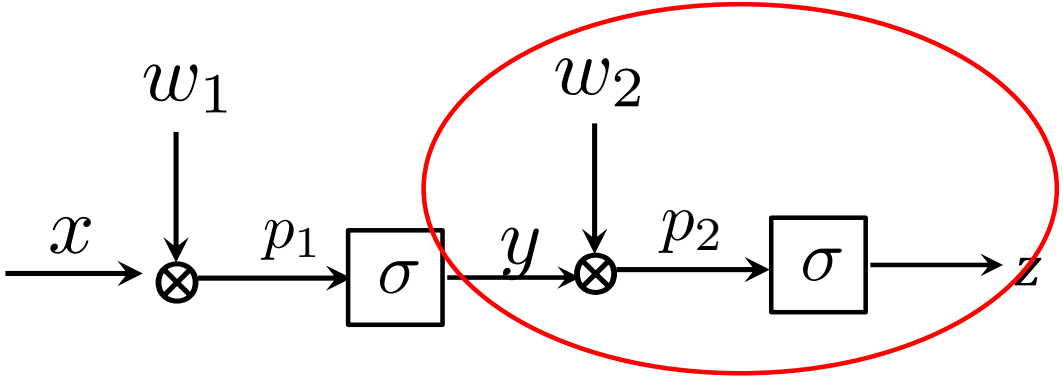
Hack #2: Change threshold function to sigmoid



Paul Werbos, The insightful genius

Solves a problem, but this also introduces a new problem…let's see how this works in net, how to adjust weights

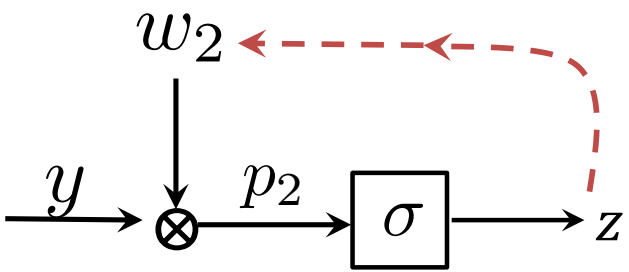
The simplest neural net – how do we adjust weights given current performance?



$$P = -\frac{1}{2}(d - z)^2$$

Consider just <u>right</u>-hand side of the net first, with weight w_2

The simplest possible neural network: how to tweak the weights, given performance ("backpropagation")



$$P = -\frac{1}{2}(d - z)^2$$

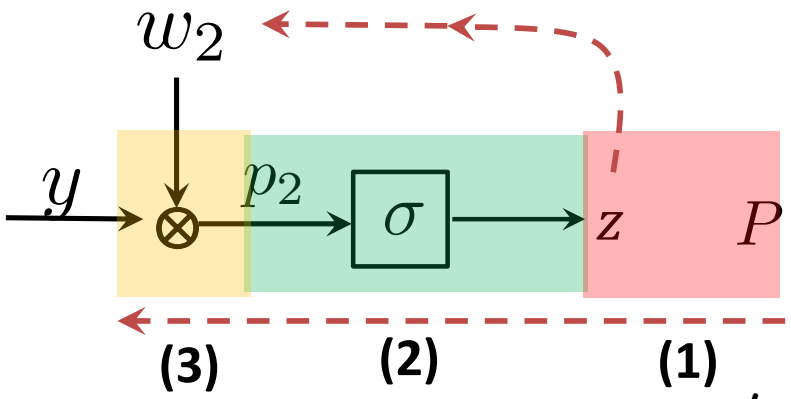
$$\frac{\partial P}{\partial w_2}$$

Q: Given desired value d, and network computed value z, how to change w_2 to get P closer to 0? (i.e., perfect performance)

Q: How much does Performance P change given a small change in output z?

A: We need to "backpropagate" any error to the weight using partial derivatives – the partial of P with respect to w_2 tells us how much P changes if we change w_2 by a little bit (improving it, we hope)

Updating weights in the simplest net by backpropagation of error



$$\frac{\partial P}{\partial w_2}$$

$$P = -\frac{1}{2}(d - z)^2$$

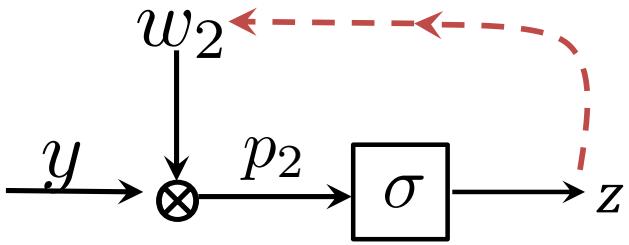
$$w_2' = w_2 + r \cdot \frac{\partial P}{\partial w_2}$$

By chain rule,

$$\frac{\partial P}{\partial w_2} = \frac{\partial P}{\partial z}\frac{\partial z}{\partial w_2}$$

$$\frac{\partial P}{\partial w_2} = \underset{(1)}{\frac{\partial P}{\partial z}}\underset{(2)}{\frac{\partial z}{\partial p_2}}\underset{(3)}{\frac{\partial p_2}{\partial w_2}}$$

How to compute these partial derivatives?



$$P = -\frac{1}{2}(d - z)^2$$

$$\frac{\partial P}{\partial w_2} = \frac{\partial P}{\partial z}\frac{\partial z}{\partial p_2}\frac{\partial p_2}{\partial w_2}$$

$$P = -\frac{1}{2}(d - z)^2 \qquad z = \sigma(p_2) \qquad p_2 = w_2 \cdot y$$

$$\frac{\partial P}{\partial z} = (d - z) \qquad ??? \qquad y$$

Need to compute derivative of Sigmoid function to find partial derivative!

Derivative of sigmoid function

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

$$d\frac{\sigma(x)}{x} = (1-\sigma(x))(\sigma(x))$$

$$d\frac{\sigma(x)}{x} = \frac{d}{dx}(1+e^{-x})^{-1} = -(1+e^{-x})^{-2}(-e^{-x})$$

$$= \frac{1}{(1+e^{-x})} \cdot \frac{e^{-x}}{(1+e^{-x})}$$

$$= \frac{e^{-x}+1-1}{(1+e^{-x})} \cdot \frac{1}{(1+e^{-x})}$$

$$= \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right)$$

1 σ σ

$$z = \sigma(p_2)$$

$$\frac{\partial z}{\partial p_2} = \frac{\partial}{\partial p_2}\sigma(p_2) = \frac{\partial}{\partial p_2}\frac{1}{1+e^{-p_2}}$$

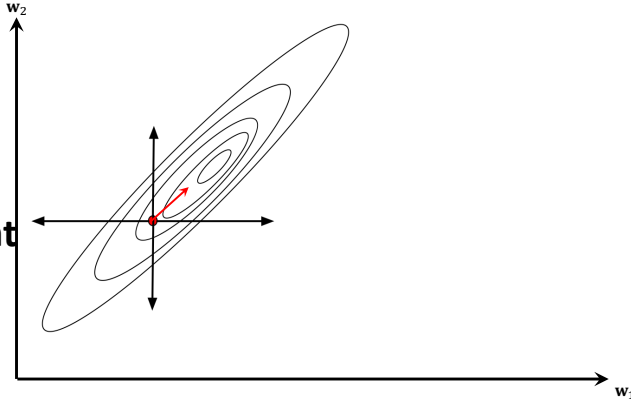
$$= \sigma(p_2) \cdot (1 - \sigma(p_2))$$

$$= z(1-z)$$

$$\frac{\partial P}{\partial w_2} = \frac{\partial P}{\partial z}\frac{\partial z}{\partial p_2}\frac{\partial p_2}{\partial w_2}$$

$$(d - z)\ (z)(1-z)\quad y$$

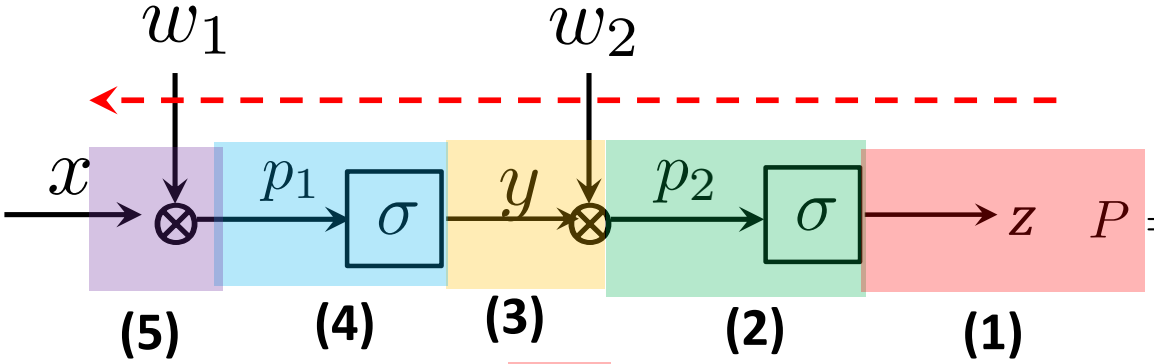
New weight = learning rate x original weight times gradient



$$w_2' = w_2 + r \cdot \frac{\partial P}{\partial w_2}$$

$$= w_2 + r \cdot (d - z)(z)(1-z)(y)$$ **Yay!**

Full Two-layer neural net: must tweak next weight back, w_1, by <u>one</u> more step of backpropagation using chain rule



$$\frac{\partial P}{\partial w_1} = \frac{\partial P}{\partial z}\frac{\partial z}{\partial w_1}$$

$$= \frac{\partial P}{\partial z}\frac{\partial z}{\partial p_2}\frac{\partial p_2}{\partial w_1}$$

$$= \frac{\partial P}{\partial z}\frac{\partial z}{\partial p_2}\frac{\partial p_2}{\partial y}\frac{\partial y}{\partial w_1}$$

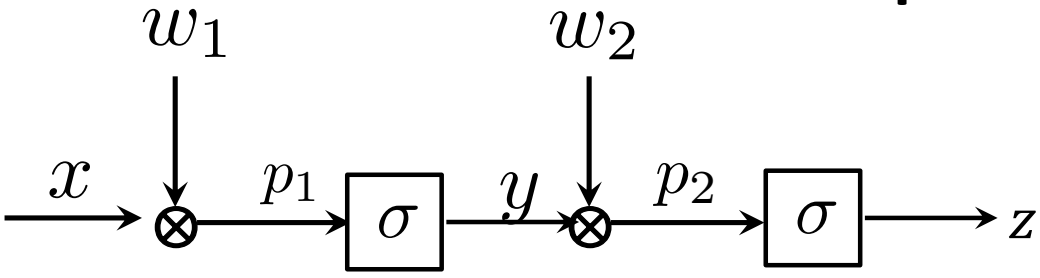
$$= \frac{\partial P}{\partial z}\frac{\partial z}{\partial p_2}\frac{\partial p_2}{\partial y}\frac{\partial y}{\partial p_1}\frac{\partial p_1}{\partial w_1}$$

Follow the path Luke!
Note it's just the chain thru the net all the way back to the w_i – (not including w_2)

Expand once more via chain rule to get all the way back to w_1

(1) (2) (3) (4) (5)

How to remember these chain rule patterns



- Start with partial derivative wrt weight you want to find, e.g., $\frac{\partial P}{\partial w_1}$
- Write down P & then starting from output z, write down the sequence of non-weights leading back to that weight:

  P z p_2 y p_1
- Shift left by 1 & write underneath, ending w/ desired weight – the partial of the first wrt variable below forms the desired chain rule sequence

  z p_2 y p_1 $w_1 \Rightarrow$ $\frac{\partial P}{\partial w_1} = \frac{\partial P}{\partial z}\frac{\partial z}{\partial p_2}\frac{\partial p_2}{\partial y}\frac{\partial y}{\partial p_1}\frac{\partial p_1}{\partial w_1}$

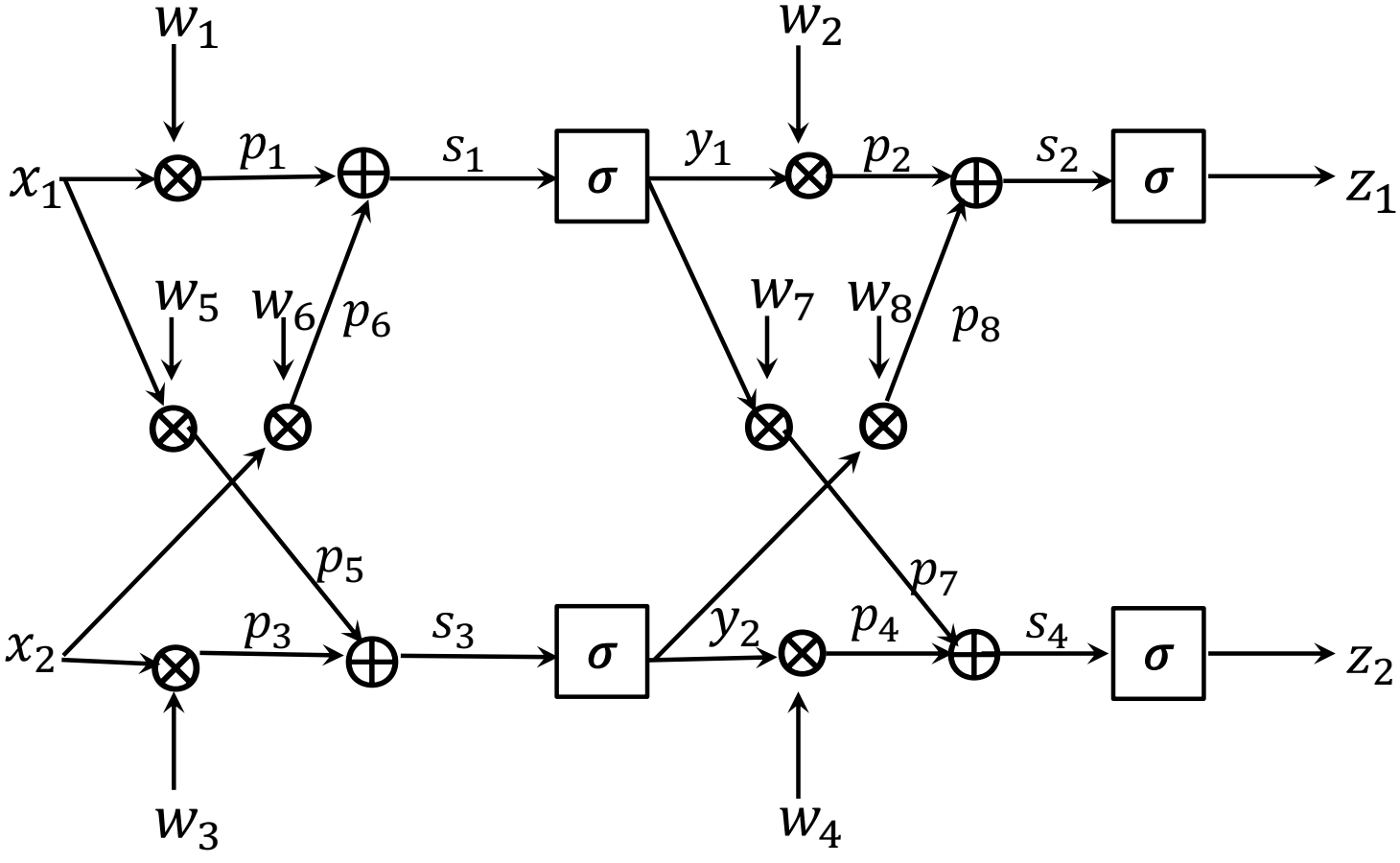
Is this practical?
Note repeated computation cascade as we work backwards from output

$$\frac{\partial P}{\partial w_2} = \frac{\partial P}{\partial z}\frac{\partial z}{\partial p_2}\frac{\partial p_2}{\partial w_2}$$

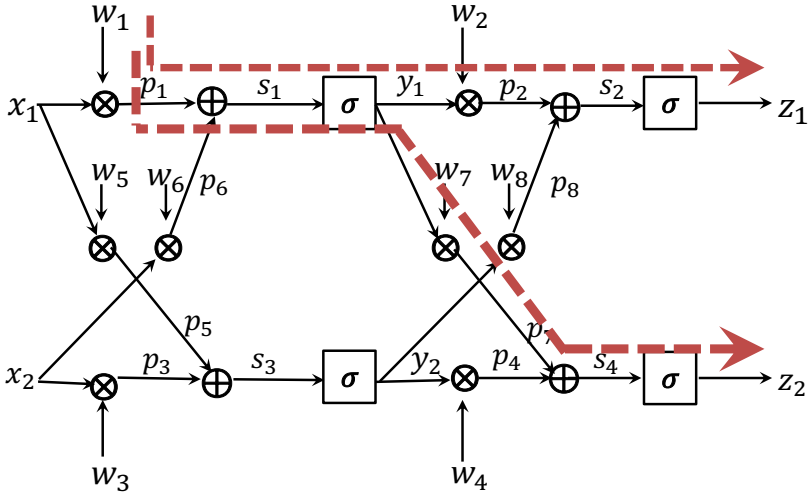
$$\frac{\partial P}{\partial w_1} = \frac{\partial P}{\partial z}\frac{\partial z}{\partial p_2}\frac{\partial p_2}{\partial y}\frac{\partial y}{\partial p_1}\frac{\partial p_1}{\partial w_1}$$

Further, in more complex networks, <u>many</u> of these paths get repeated such that the computations are <u>linear</u> in the depth of the net, and <u>quadratic</u> in its width, rather than exponential

A 2-layer, 2-input network
Is backpropagation practical? Does it blow-up exponentially?

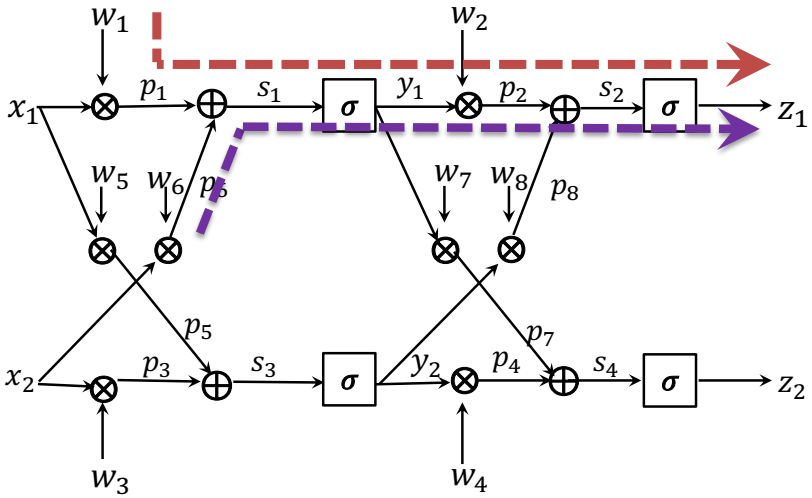


Lots of computational re-use – shared paths thru net = shared computation



$$\frac{\partial P}{\partial w_1} = \frac{\partial P}{\partial z_1}\frac{\partial z_1}{\partial s_2}\frac{\partial s_2}{\partial p_2}\frac{\partial p_2}{\partial y_1}\frac{\partial y_1}{\partial s_1}\frac{\partial s_1}{\partial p_1}\frac{\partial p_1}{\partial w_1} + \frac{\partial P}{\partial z_2}\frac{\partial z_2}{\partial s_4}\frac{\partial s_4}{\partial p_7}\frac{\partial p_7}{\partial y_1}\frac{\partial y_1}{\partial s_1}\frac{\partial s_1}{\partial p_1}\frac{\partial p_1}{\partial w_1}$$

Lots of computational re-use



$$\frac{\partial P}{\partial w_1} = \frac{\partial P}{\partial z_1}\frac{\partial z_1}{\partial s_2}\frac{\partial s_2}{\partial p_2}\frac{\partial p_2}{\partial y_1}\frac{\partial y_1}{\partial s_1}\frac{\partial s_1}{\partial p_1}\frac{\partial p_1}{\partial w_1} + \frac{\partial P}{\partial z_2}\frac{\partial z_2}{\partial s_4}\frac{\partial s_4}{\partial p_7}\frac{\partial p_7}{\partial y_1}\frac{\partial y_1}{\partial s_1}\frac{\partial s_1}{\partial p_1}\frac{\partial p_1}{\partial w_1}$$

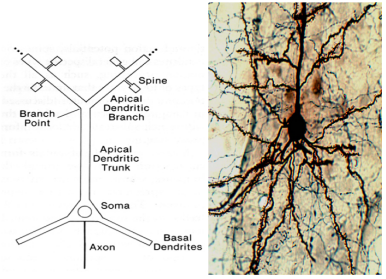
$$\frac{\partial P}{\partial w_6} = \frac{\partial P}{\partial z_1}\frac{\partial z_1}{\partial s_2}\frac{\partial s_2}{\partial p_2}\frac{\partial p_2}{\partial y_1}\frac{\partial y_1}{\partial s_1}\frac{\partial s_1}{\partial p_6}\frac{\partial p_6}{\partial w_6} + \frac{\partial P}{\partial z_2}\frac{\partial z_2}{\partial s_4}\frac{\partial s_4}{\partial p_7}\frac{\partial p_7}{\partial y_1}\frac{\partial y_1}{\partial s_1}\frac{\partial s_1}{\partial p_6}\frac{\partial p_6}{\partial w_6}$$

Gold star ideas

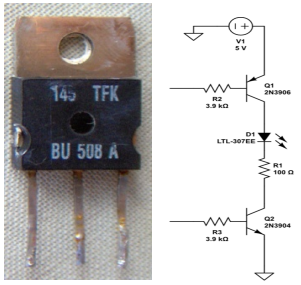
- INSIGHT + 2 TRICKS = GENIUS
- REUSE PRINCIPLE
- ALL GREAT IDEAS ARE SIMPLE

Differences between brains and computers – is there backprop in the brain?

Neuron/Synapses **Transistor/Gates**



VS.



OPERATING/COMPUTING CHARACTERISTICS

~10^3 Hz	Clock Rate	10^9 Hz
~10^2 m/s	Signal Velocity	10^8 m/s
~1	Signal-to-Noise	10^6
~10^4	Parallel Connections	~1

MEMORY STORAGE/RETRIEVAL

Distributed Circuits	Medium	Address Registers
Content Addressable	Mechanism	Instruction Addressable

Slide courtesy of Robert Ajemian