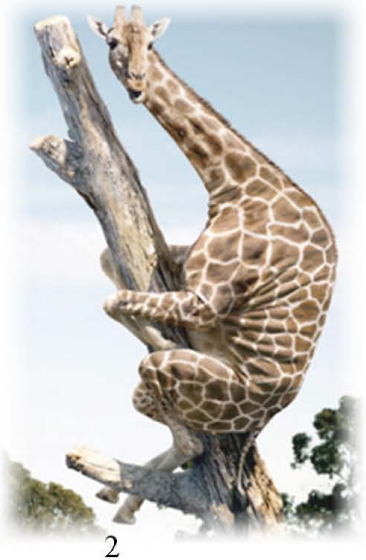
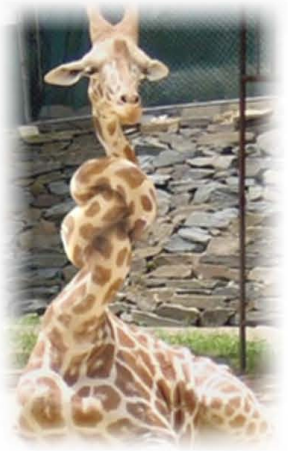
CS 437 / CS 5317
Deep Learning

Murtaza Taj
murtaza.taj@lums.edu.pk

Lecture 17: Deep Learning
Linear Regression, Neuron, Gradient Descent
Mon 25th Mar 2019

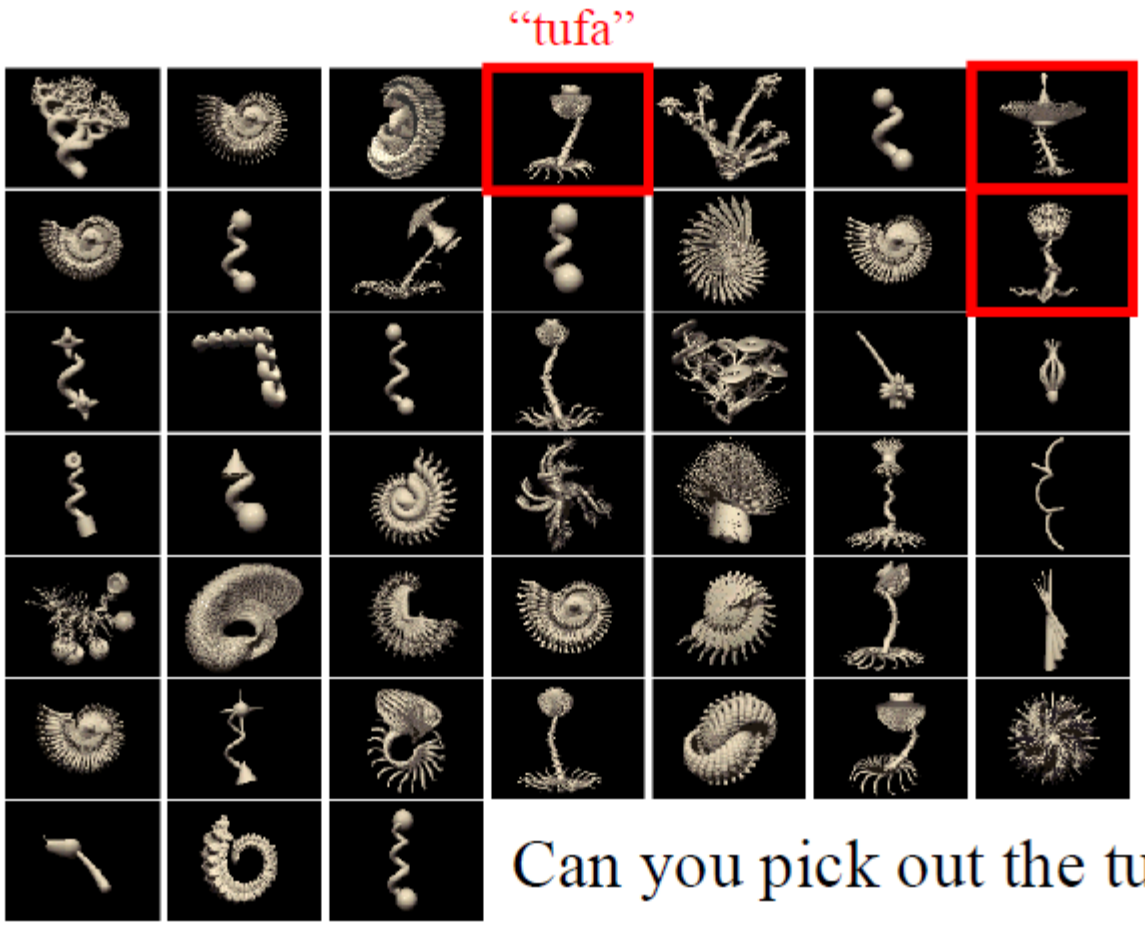
Human Perception





[Thomas Serre 2012]

Challenge: One shot learning



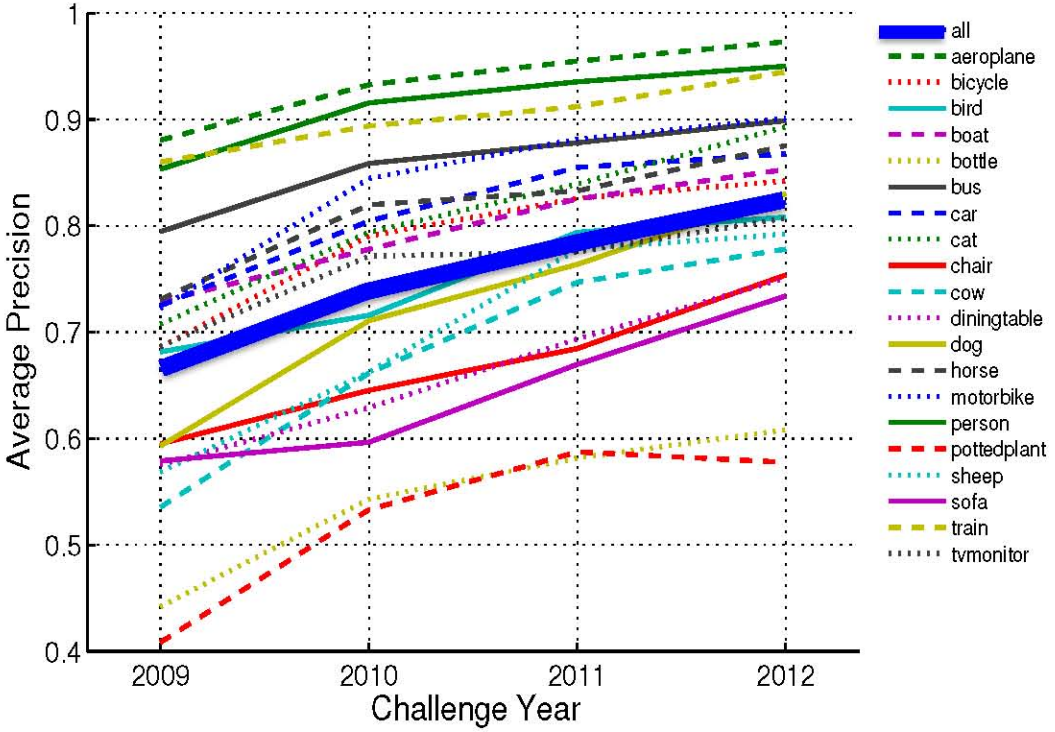
Josh Tenenbaum

Our goal is to develop example based or data driven learning capabilities for computers

Deep Learning is recent break through in achieving this goal

Pascal Visual Object Challenge

- 20 Object Categories



IMAGENET

www.image-net.org

22K categories and **14M** images

- Animals
  - Bird
  - Fish
  - Mammal
  - Invertebrate
- Plants
  - Tree
  - Flower
- Food
- Materials
- Structures
- Artifact
  - Tools
  - Appliances
  - Structures
- Person
- Scenes
  - Indoor
  - Geological Formations
- Sport Activities

Deng, Dong, Socher, Li, Li, & Fei-Fei, 2009

IMAGENET Large Scale Visual Recognition Challenge

Steel drum

The Image Classification Challenge:

1,000 object classes

1,431,167 images

Output:
Scale
T-shirt
Steel drum
Drumstick
Mud turtle

✔

Output:
Scale
T-shirt
Giant panda
Drumstick
Mud turtle

✗

Russakovsky et al. arXiv, 2014

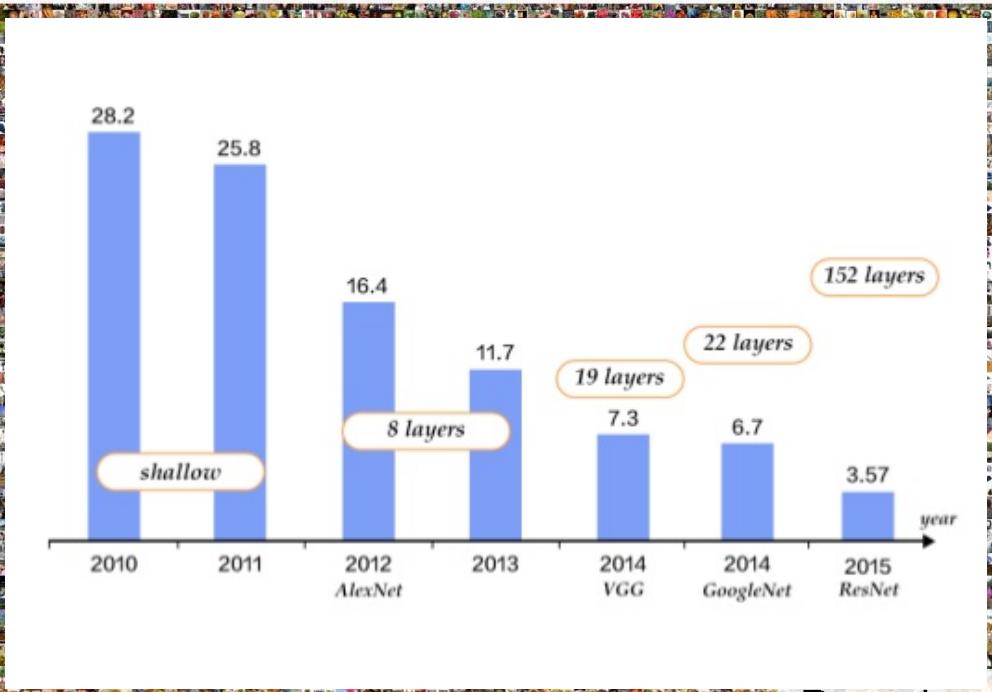
IMAGENET Large Scale Visual Recognition Challenge

Steel drum

The Image Classification Challenge:

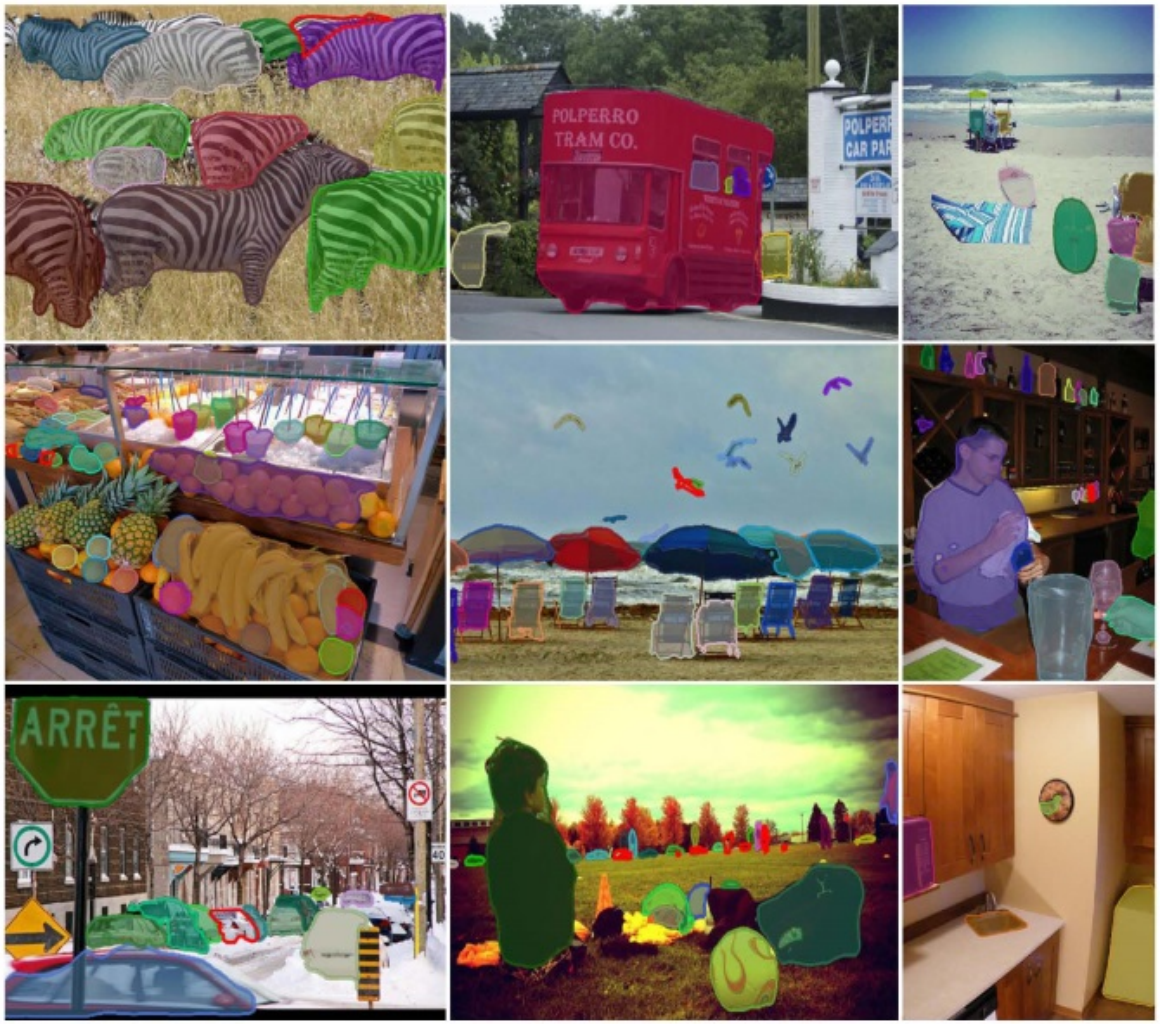
1,000 object classes

1,431,167 images



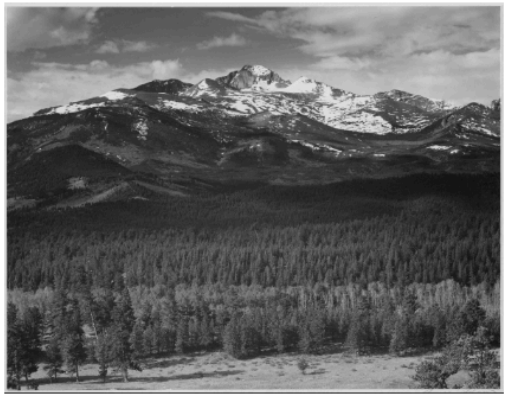
Russakovsky et al. arXiv, 2014

Object Detection and Segmentation



(Pinheiro et al., 2016)

Restore colors in B&W photos and videos

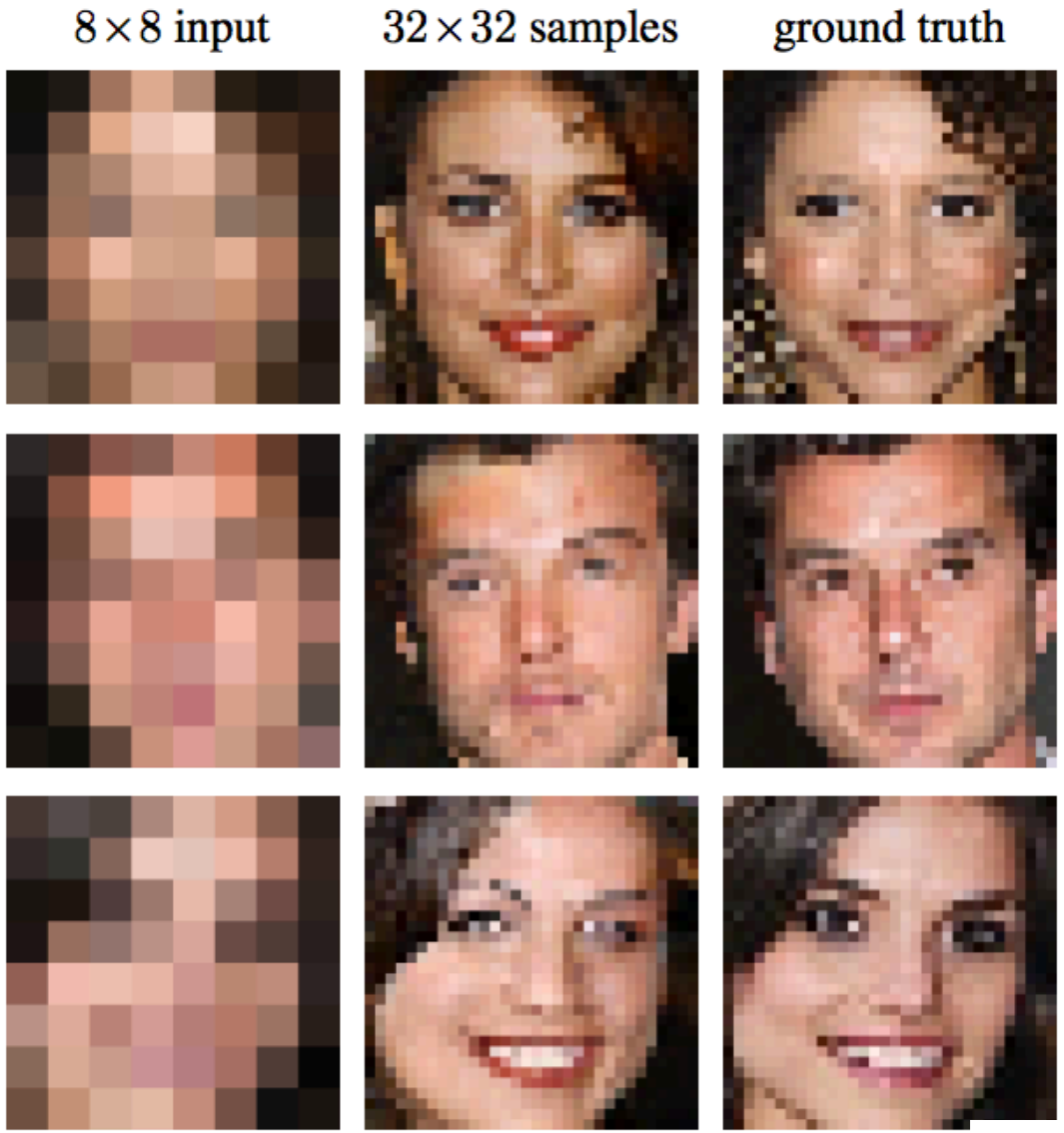







Colorado National Park, 1941 | Textile Mill, June 1937 | Berry Field, June 1909 | Hamilton, 1936

Pixel Restoration

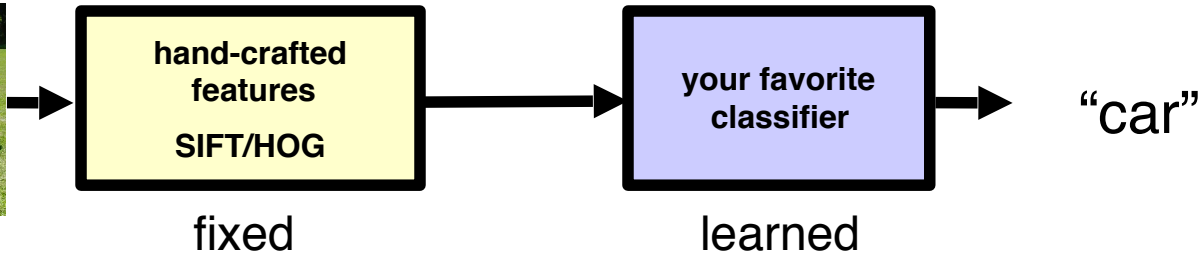


(Google Brain 2017)

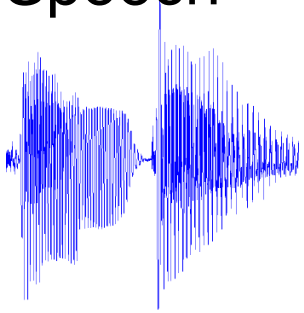
Shallow vs. Deep Learning

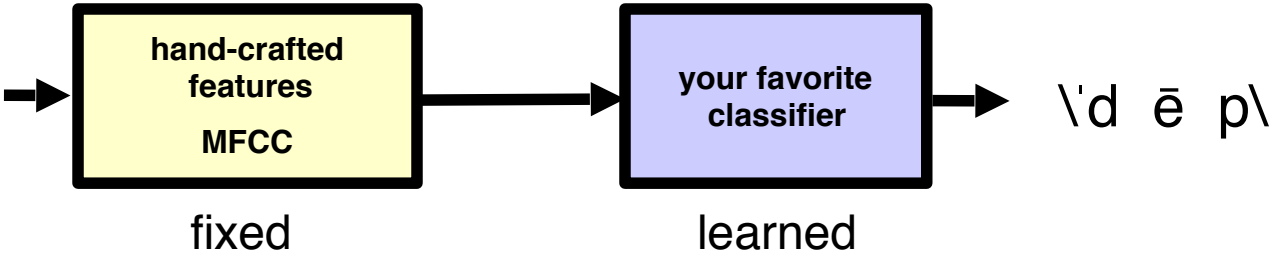
Traditional Machine Learning

- Vision



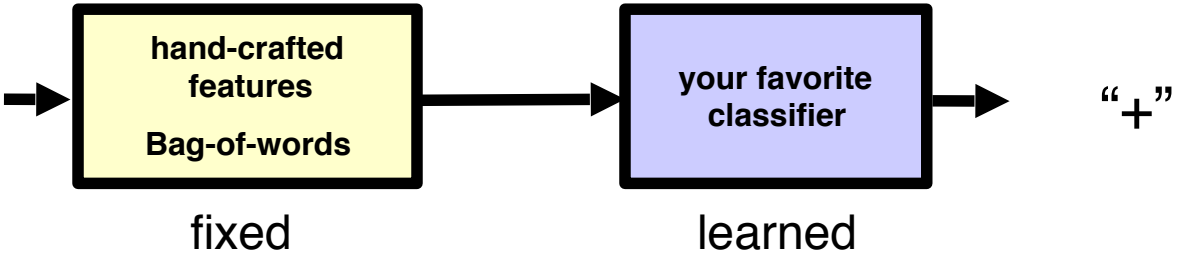
- Speech





- NLP

This burrito place is yummy and fun!



Hierarchical Compositionality

- Vision

pixels → edge → texton → motif → part → object

- Speech

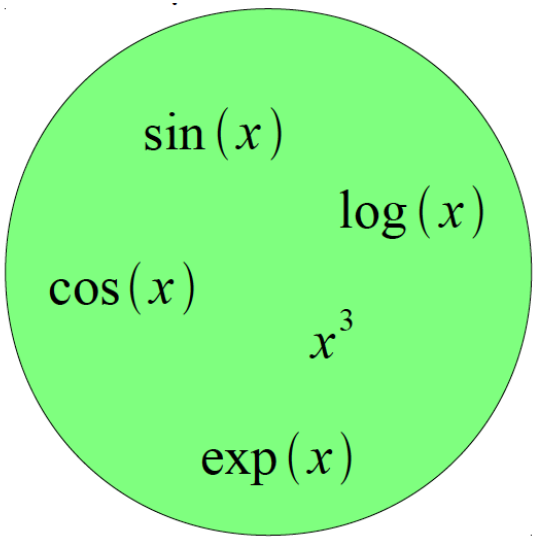
sample → spectral band → formant → motif → phone → word

- NLP

character → word → NP/VP/.. → clause → sentence → story

Building A Complicated Function

Given a library of simple functions



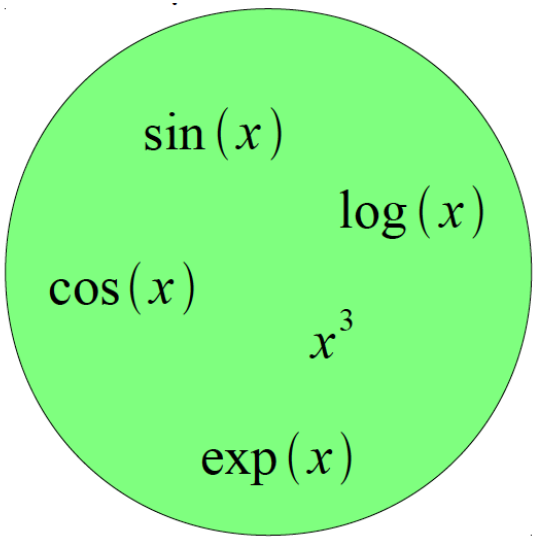
Compose into a

complicate function

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Building A Complicated Function

Given a library of simple functions



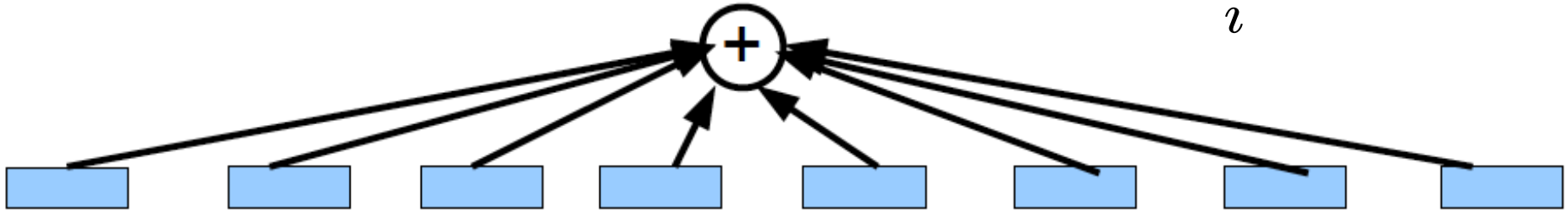
Compose into a

complicate function

Idea 1: Linear Combinations

- Boosting
- Kernels
- …

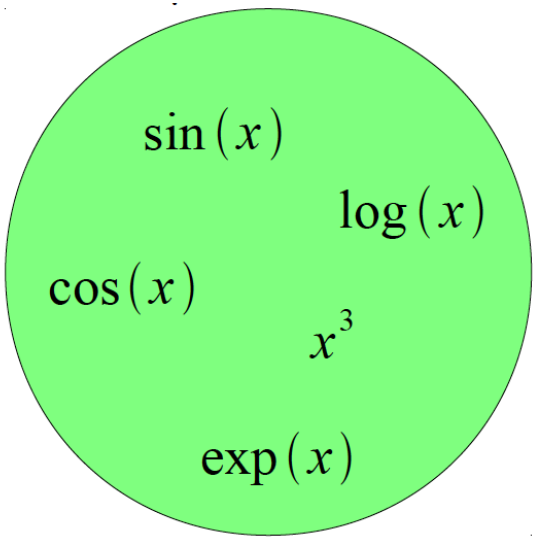
$$f(x) = \sum_i \alpha_i g_i(x)$$



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Building A Complicated Function

Given a library of simple functions



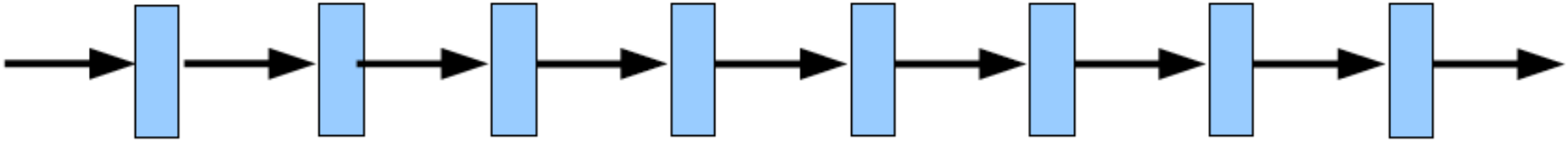
Compose into a

complicate function

Idea 2: Compositions

- Deep Learning
- Grammar models
- Scattering transforms…

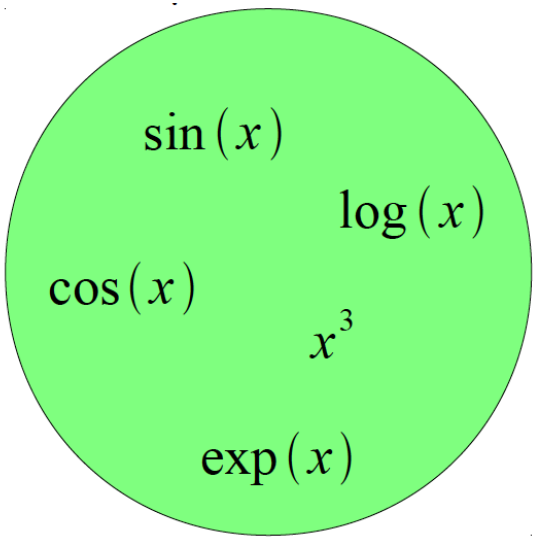
$$f(x) = g_1(g_2(\ldots(g_n(x)\ldots))$$



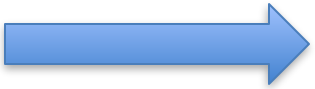
Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Building A Complicated Function

Given a library of simple functions



Compose into a

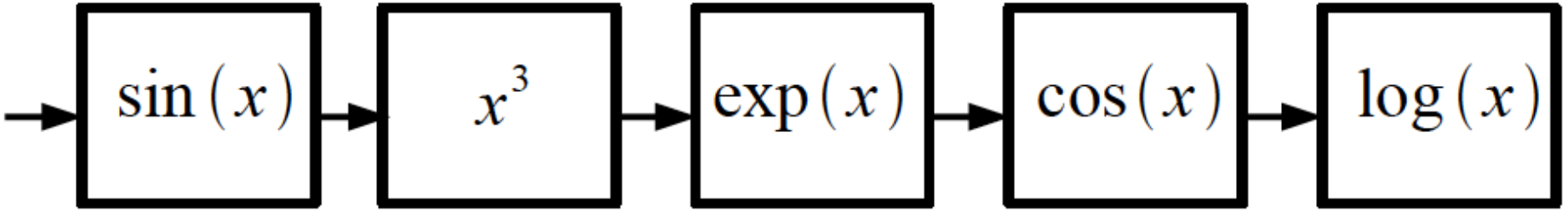


complicate function

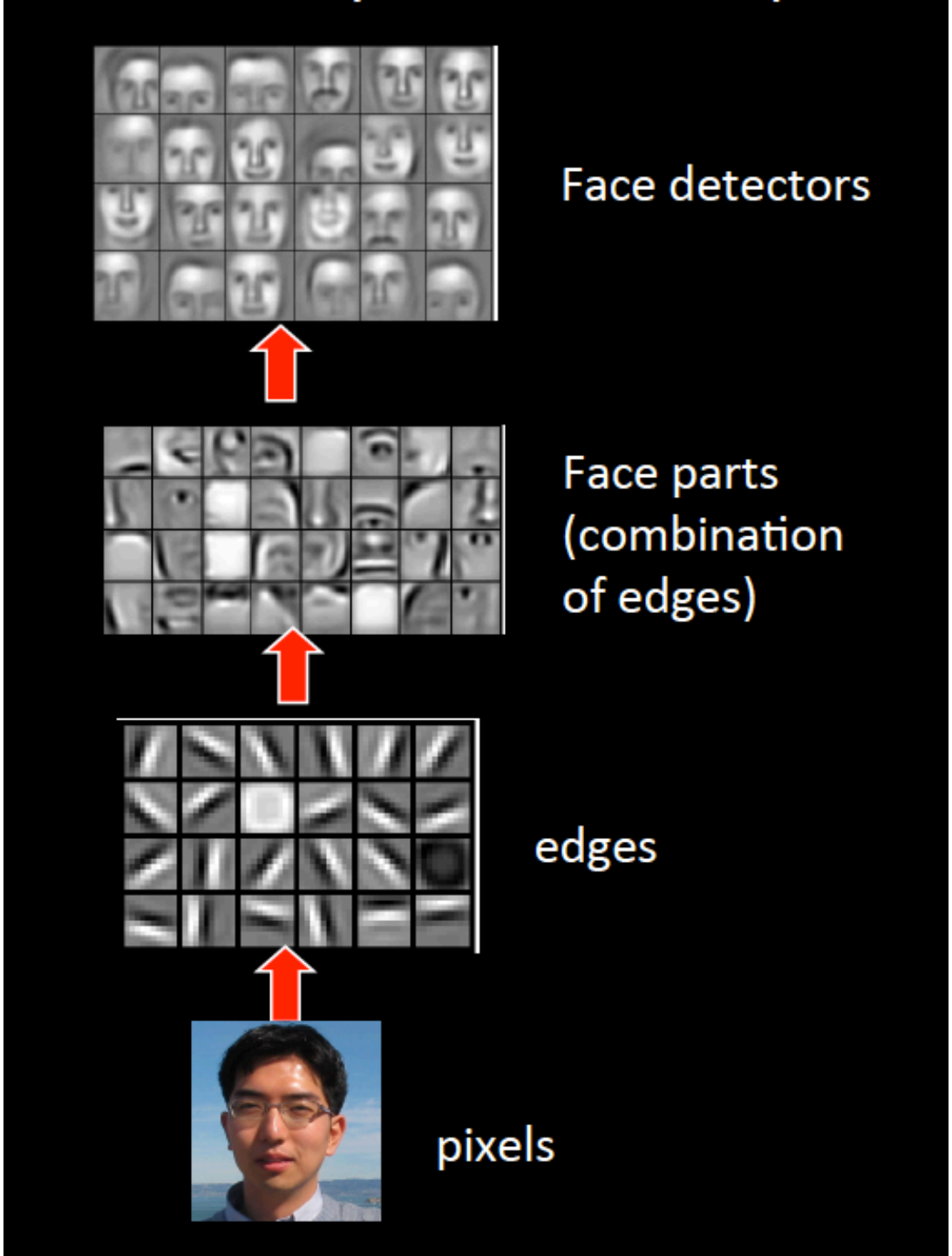
Idea 2: Compositions

- Deep Learning
- Grammar models
- Scattering transforms…

$$f(x) = \log(\cos(\exp(\sin^3(x))))$$



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun



Sparse DBNs

[Lee et al. ICML '09]

Figure courtesy: Quoc Le

Deep Learning = Hierarchical Compositionality

Low-Level Feature → Mid-Level Feature → High-Level Feature → Trainable Classifier → "car"

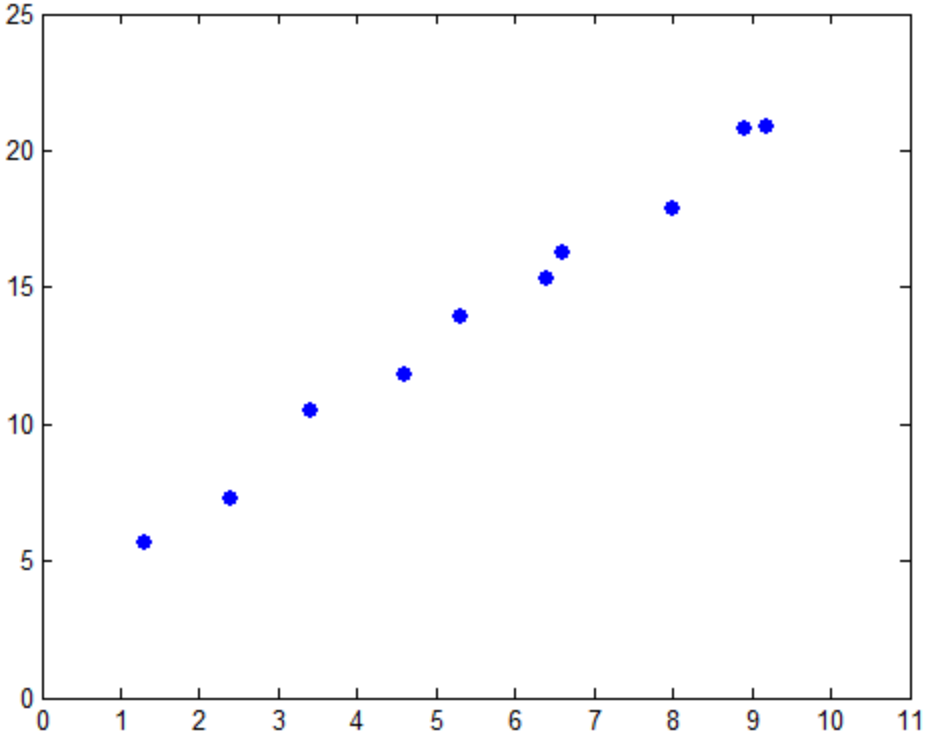
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Linear Regression

Parameter Optimization: Least Squared Error Solutions

- Let us first consider the ‘simpler’ problem of fitting a line to a set of data points…

x	y
1.3	5.7
2.4	7.3
3.4	10.5
4.6	11.8
5.3	13.9
6.6	16.3
6.4	15.3
8.0	17.9
8.9	20.8
9.2	20.9



- Equation of best fit line ?

Line Fitting: Least Squared Error Solution

- Step 1: Identify the model
  - Equation of line: $y = mx + c$
- Step 2: Set up an error term which will give the goodness of every point with respect to the (unknown) model
  - Error induced by i^{th} point: $e^{(i)} = (t^{(i)} - mx^{(i)} - c)^2$
  - Error for whole data: $E = \sum_i (t^{(i)} - y^{(i)})^2$

    $E = \sum_i (t^{(i)} - mx^{(i)} - c)^2$
- Step 3: Differentiate Error w.r.t. parameters, put equal to zero and solve for minimum point
- In other words, find the set of parameters m and c for which the error term E is minimized

Line Fitting: Least Squared Error Solution

$$E = \sum_i (t^{(i)} - mx^{(i)} - c)^2$$

$$\frac{\partial E}{\partial m} = -\sum_i (t^{(i)} - mx^{(i)} - c)x^{(i)}$$

$$\frac{\partial E}{\partial c} = -\sum_i (t^{(i)} - mx^{(i)} - c)$$

$$\begin{bmatrix} \sum_i (x^{(i)})^2 & \sum_i x^{(i)} \\ \sum_i x^{(i)} & \sum_i 1 \end{bmatrix} \begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} \sum_i x^{(i)} t^{(i)} \\ \sum_i t^{(i)} \end{bmatrix}$$

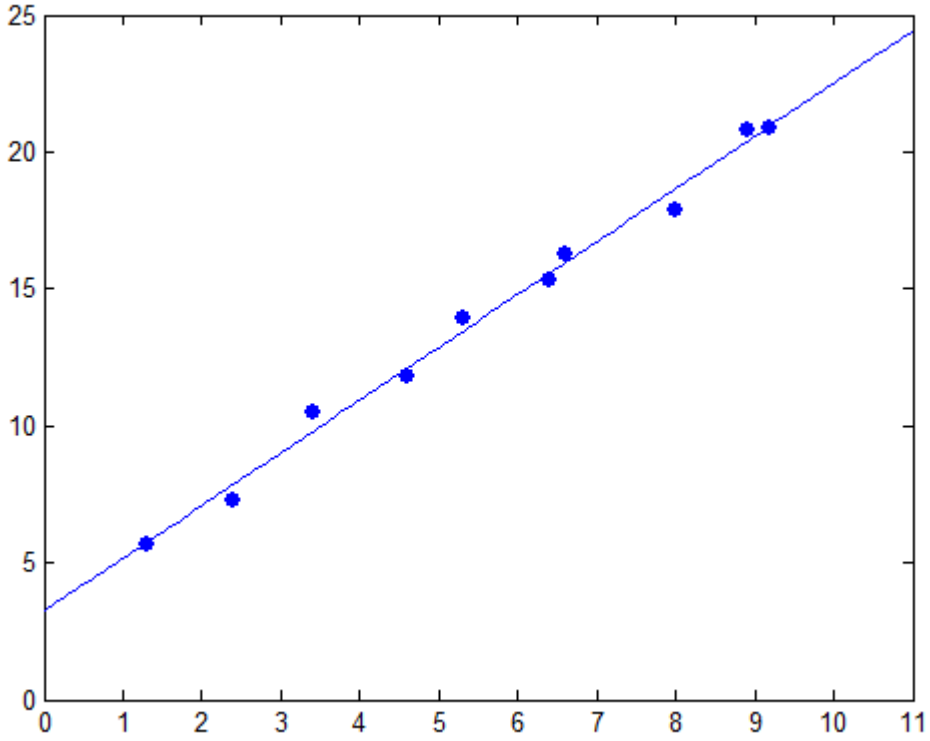
Ax = b

x	t
1.3	5.7
2.4	7.3
3.4	10.5
4.6	11.8
5.3	13.9
6.6	16.3
6.4	15.3
8.0	17.9
8.9	20.8
9.2	20.9

$$\begin{pmatrix} 380.63 & 56.1 \\ 56.1 & 10 \end{pmatrix} \begin{pmatrix} m \\ c \end{pmatrix} = \begin{pmatrix} 914.68 \\ 140.4 \end{pmatrix}$$

Solution: $m = 1.9274$ $c = 3.227$

Line Fitting: Least Squared Error Solution



Line Fitting: Least Squared Error Solution

$$y = mx + c$$

$$\mathbf{w} = \{w_1, w_2, \cdots\}^T$$
$$\mathbf{x} = \{x_1, x_2, \cdots\}^T$$
$$y = \mathbf{w}^T\mathbf{x} + w_0 = \sum_j w_j x_j + w_0$$

$$E = \sum_i (t^{(i)} - mx^{(i)} - c)^2$$

$$E = \sum_i (t^{(i)} - \mathbf{w}^T\mathbf{x} - w_0)^2$$

$$\frac{\partial E}{\partial m} = -\sum_i (t^{(i)} - mx^{(i)} - c)x^{(i)}$$

$$\frac{\partial E}{\partial c} = -\sum_i (t^{(i)} - mx^{(i)} - c)$$

$$\frac{\partial E}{\partial w_j} = -\sum_i (t^{(i)} - y^{(i)})x_j^{(i)}$$

$$\begin{bmatrix} \sum_i (x^{(i)})^2 & \sum_i x^{(i)} \\ \sum_i x^{(i)} & \sum_i 1 \end{bmatrix} \begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} \sum_i x^{(i)} t^{(i)} \\ \sum_i t^{(i)} \end{bmatrix}$$

Linear Classifier

Linear Regression for Classification

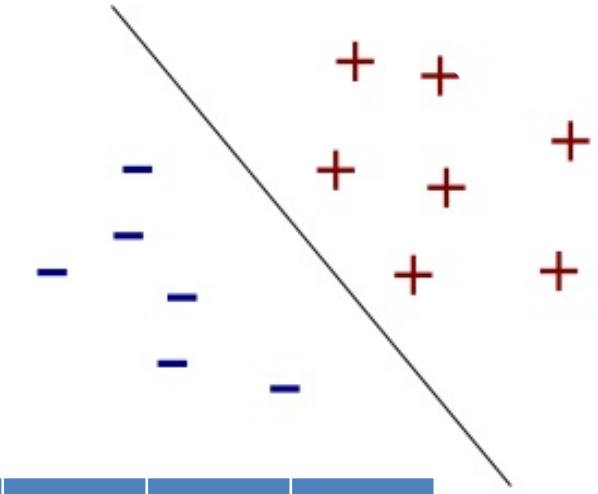
$$\mathbf{w} = \{w_0, w_1, w_2, \cdots\}^T$$
$$\mathbf{x} = \{1, x_1, x_2, \cdots\}^T$$
$$y = \mathbf{w}^T\mathbf{x} = \sum_j w_j x_j$$

$$t^{(i)} \in \{-1, +1\}$$

$$E = \sum_i (t^{(i)} - \mathbf{w}^T\mathbf{x})^2$$

$$\frac{\partial E}{\partial w_j} = -\sum_i (t^{(i)} - y^{(i)}) x_j^{(i)}$$

$$\begin{cases} +1 & y > 0 \\ -1 & y \le 0 \end{cases}$$



x1	x2	x3	y
1	3	2	+ve
1	4	3	+ve
1	4	8	-ve
1	8	9	-ve

- $\mathbf{w} = \{10, -1, -1\}$
- $1x10 + (-1)x3 + (-1)x2 > 0$
- $1x10 + (-1)x4 + (-1)x8 < 0$

Linear Regression for Classification

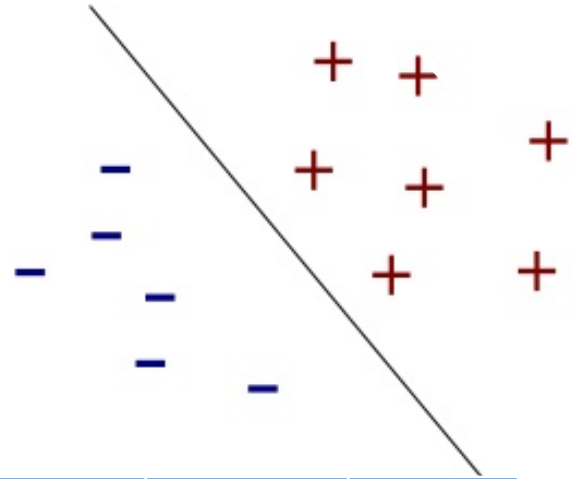
$$\mathbf{w} = \{w_0, w_1, w_2, \cdots\}^T$$
$$\mathbf{x} = \{1, x_1, x_2, \cdots\}^T$$
$$y = \mathbf{w}^T\mathbf{x} = \sum_j w_j x_j$$

$$t^{(i)} \in \{-1, +1\}$$

$$E = \sum_i (t^{(i)} - \mathbf{w}^T\mathbf{x})^2$$

$$\frac{\delta E}{\delta w_j} = -\sum_i (t^{(i)} - y^{(i)}) x_j^{(i)}$$

$$\begin{cases} +1 & y > 0 \\ -1 & y \le 0 \end{cases}$$



Lbs	Width	Heigth	y
1	3	2	Insect
1	4	3	Insect
2	11	12	Bird
2	14	9	Bird

- $\mathbf{w} = \{10, -1, -1\}$
- $1x10 + (-1)x3 + (-1)x2 > 0$
- $1x10 + (-1)x4 + (-1)x8 < 0$

Neuron

Neuron

$$\mathbf{x} = \{1, x_1, x_2, \cdots\}^T$$

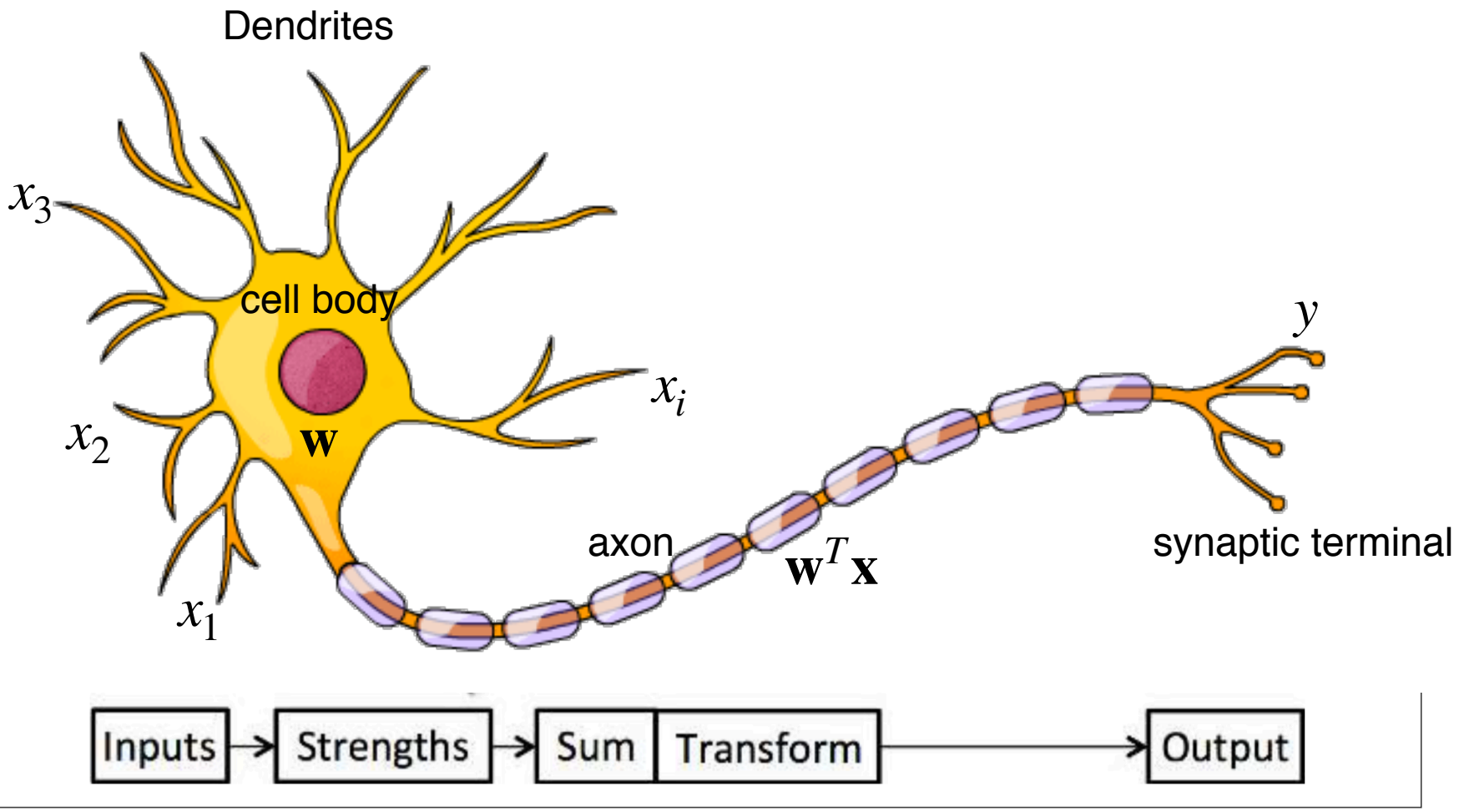


Figure 1-6. A functional description of a biological neuron's structure

Linear Perceptron as Linear Neuron

Linear Neuron (also called Linear Filter)

- The neural has a real-valued output which is a weighted sum of its input

$$y = \mathbf{w}^T \mathbf{x} = \sum_j w_j x_j$$

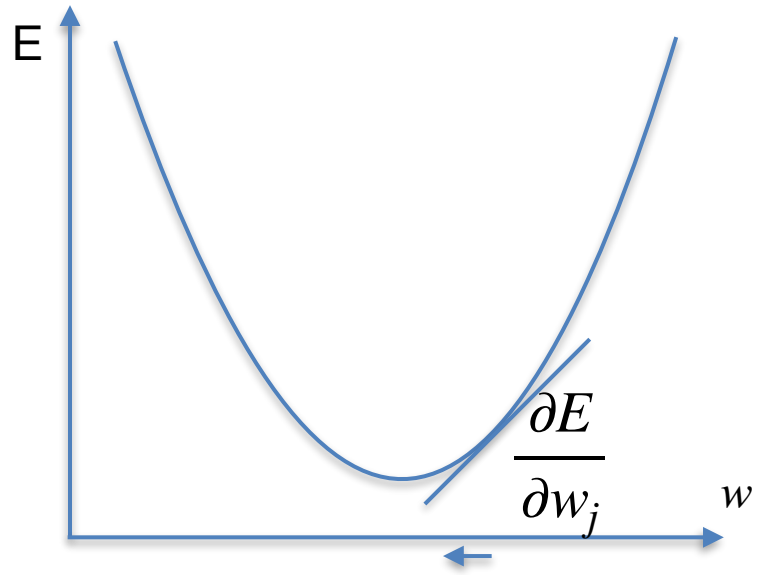
- The aim of learning is to minimise the sum over all training cases
  - The error is squared difference between the desired output and actual output

$$E = \sum_i (t^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)})^2$$

Error Function

Error function

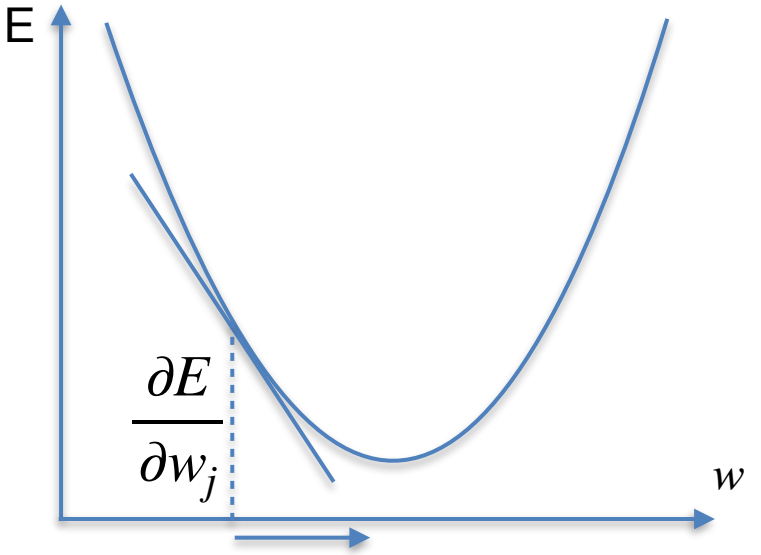
$$E = \sum_i (t^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)})^2$$



$$w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}$$

$$w_j \leftarrow w_j + \Delta w_j$$

$$\Delta w_j = \eta (t - y) x_j$$

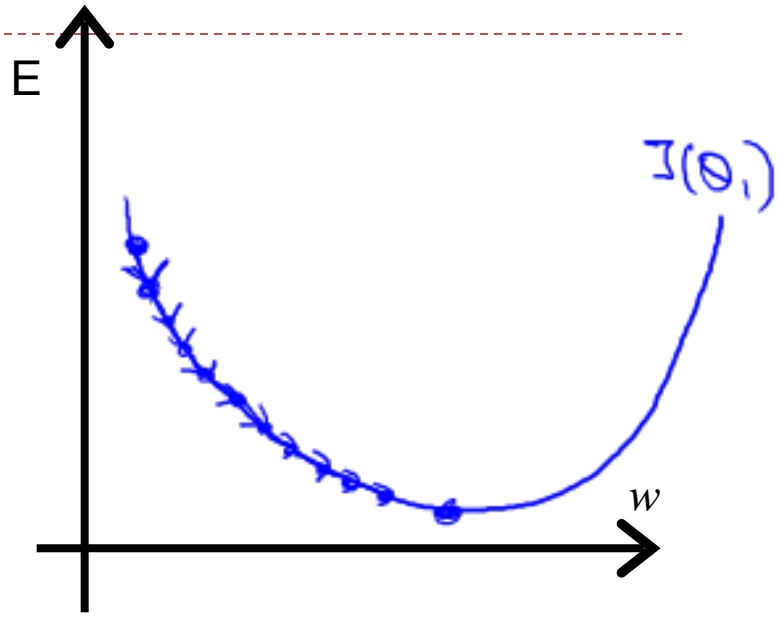


where η is the learning rate

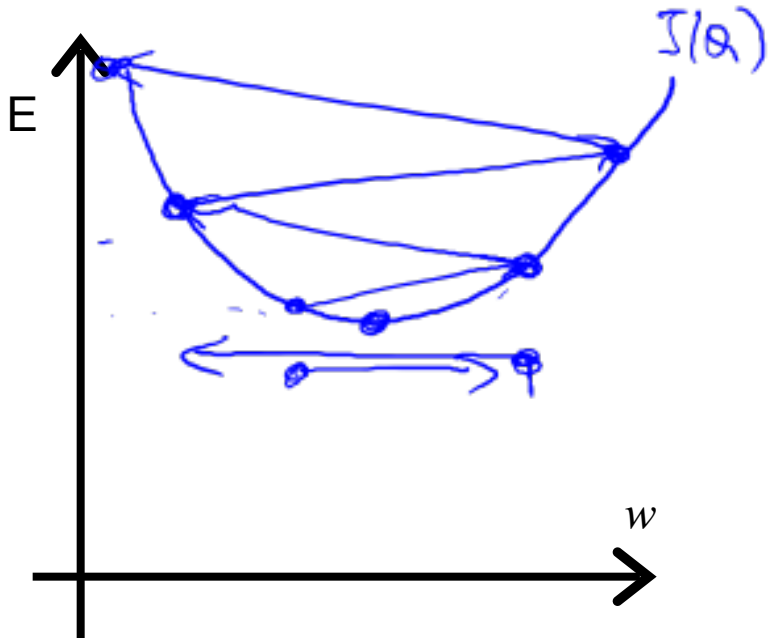
Error function

$$w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}$$

If η is too small, gradient descent can be slow.



If η is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



Analytical vs. Iterative Solution

$$\mathbf{w} = \{w_0, w_1, w_2, \cdots\}^T$$
$$\mathbf{x} = \{1, x_1, x_2, \cdots\}^T$$
$$y = \mathbf{w}^T\mathbf{x} = \sum_j w_j x_j$$
$$E = \sum_i (t^{(i)} - \mathbf{w}^T\mathbf{x})^2$$

- Analytical Solution

$$\frac{\partial E}{\partial w_j} = -\sum_i (t^{(i)} - y^{(i)}) x_j^{(i)}$$

Aw = b

- Iterative Solution

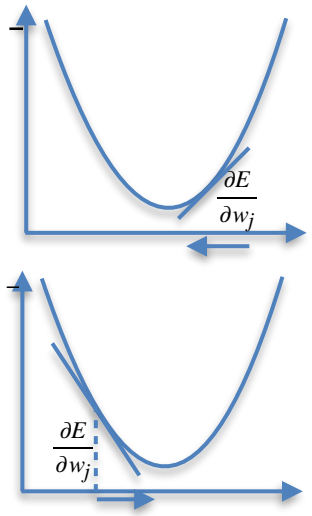
$$w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}$$

$$w_j \leftarrow w_j + \Delta w_j$$

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j}$$

$$\Delta w_j = \eta (t - y) x_j$$

where η is the learning rate



Why don't we solve it analytically?

- It is straight forward to write down a set of equations, one per training case and to solve for best set of weights
  - This is standard engineering approach so why don't we use it?

- Scientific Answer: We want a method that real neurons could use

- Engineering Answer: We want a method that generalised to multi-layer, non-linear neural network
  - The analytics solution relies on it being linear and having a squared error measure
  - Iterative methods are usually less efficient but are much easier to generalise

https://www.coursera.org/learn/neural-networks/lecture/bD3OB/learning-the-weights-of-a-linear-neuron-12-min

Iterative Method

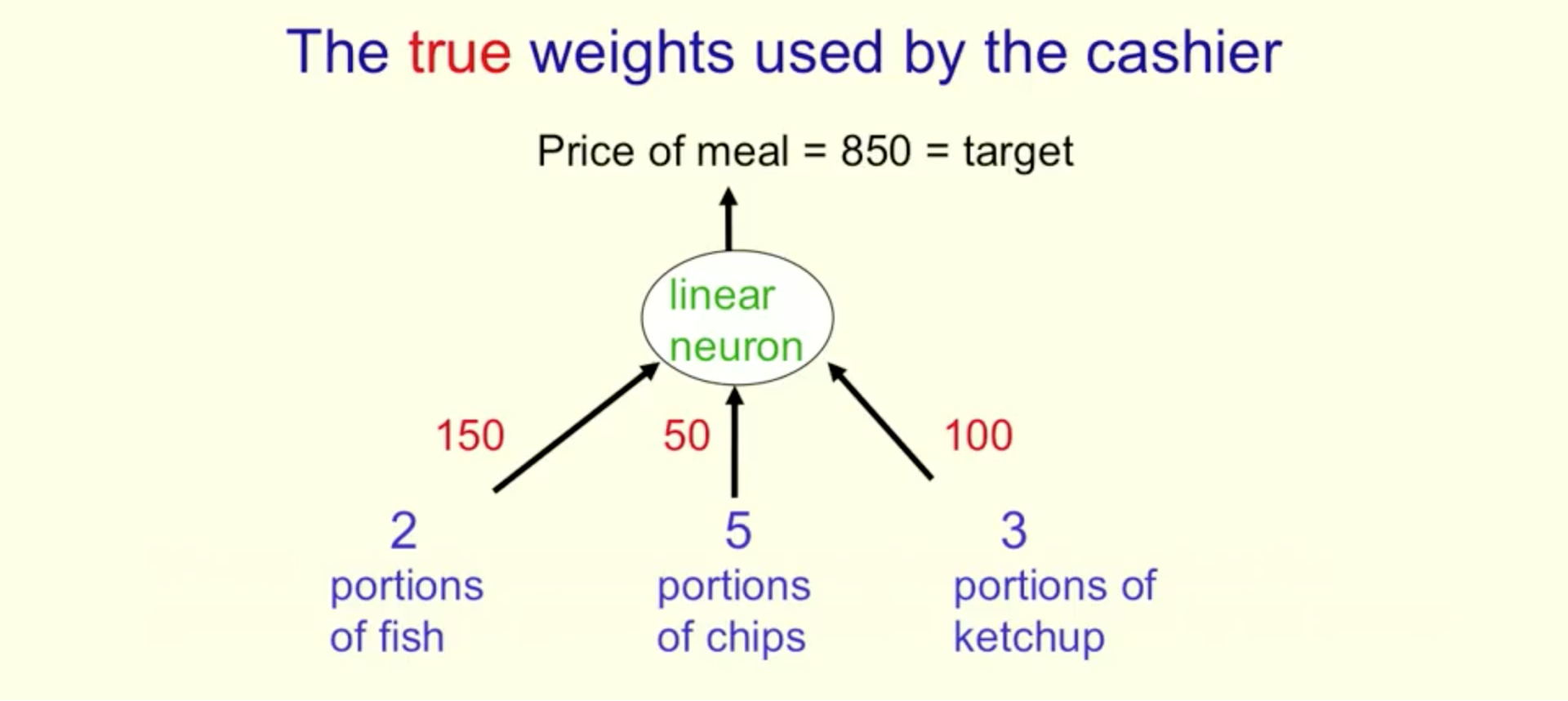
A Toy Example to Illustrate the Iterative Method

- Each day you get the lunch at the cafeteria
  - Your diet consists of fish, chips and ketchup
  - You get serval portions of each
- The cashier only tells you the total price of the meal
  - After several days, you should be able to figure out the price of each portions
- The iterative approach: Start with random guesses for the price and then adjust them to get a better fit to the observed prices of whole meals

Solving the equation iteratively

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

Solving the equation iteratively



Solving the equation iteratively

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

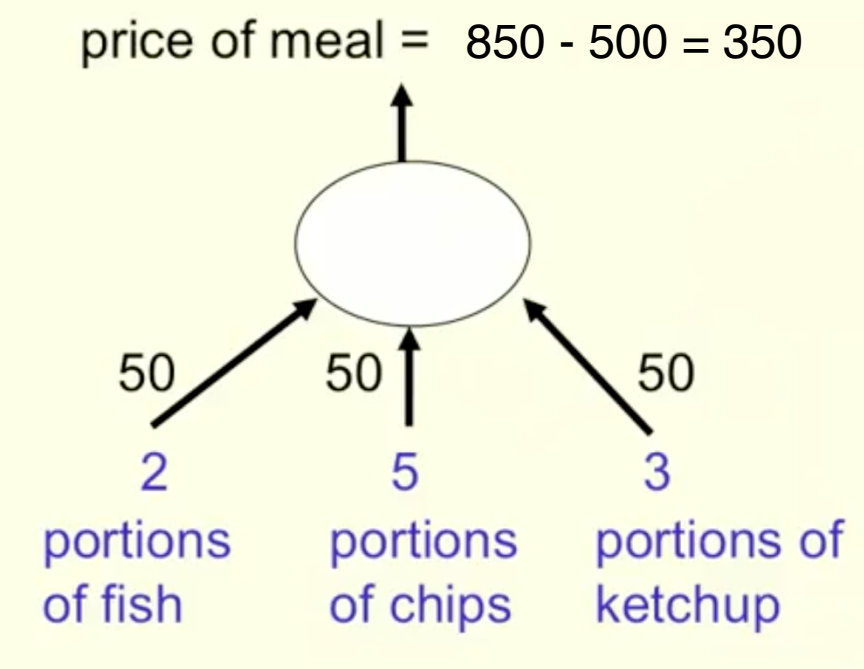
$$price = x_{fish}w_{fish} + x_{chips}w_{chips} + x_{ketchup}w_{ketchup}$$

- The prices of the portions are like the weights of a linear neuron

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

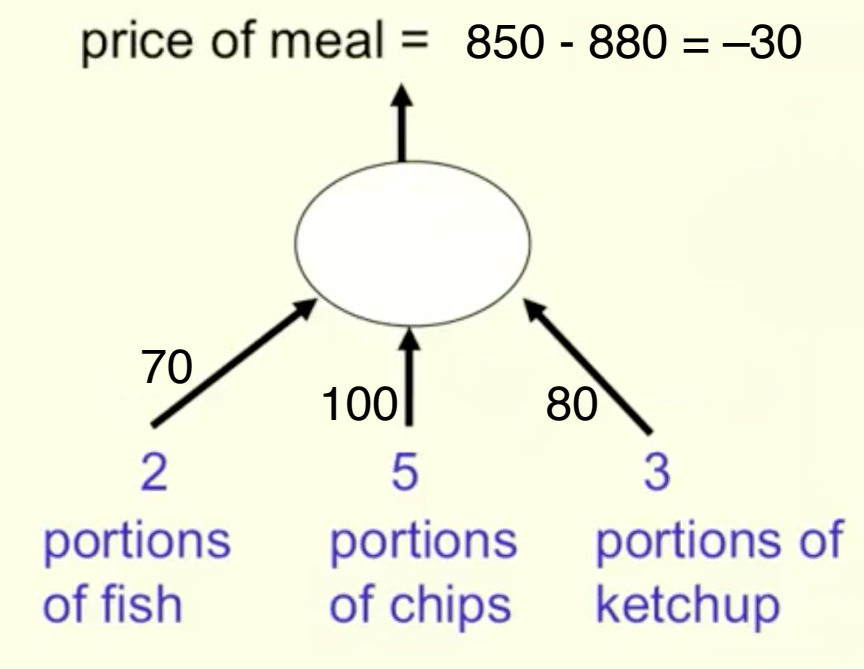
- We will start with guesses for the weights and then adjust the guesses slightly to give a better fit to the prices given by the cashier

A model of the cashier with arbitrary initial weights



- Residual error = 350
- The "delta-rule" for learning is $\Delta w_i = \eta(t - y)x_i$
- With a learning rate η of 1/35, the weight changes are +20, +50, +30
- This gives the new weights of 70, 100, 80
  - Notice that the weights for chips got worse!

A model of the cashier with arbitrary initial weights



- Residual error = -30
- The "delta-rule" for learning is $\Delta w_i = \eta(t - y)x_i$
- With a learning rate η of 1/35, the weight changes are -1.71, -4.29, -2.57
- This gives the new weights of 68.29, 95.71, 77.43

Deriving the delta rule

- Define the error as the squared residuals summed over at training cases:

$$E = \frac{1}{2} \sum_{i \in train} (t^{(i)} - y^{(i)})^2$$

- Now differentiate to get error derivates for weights

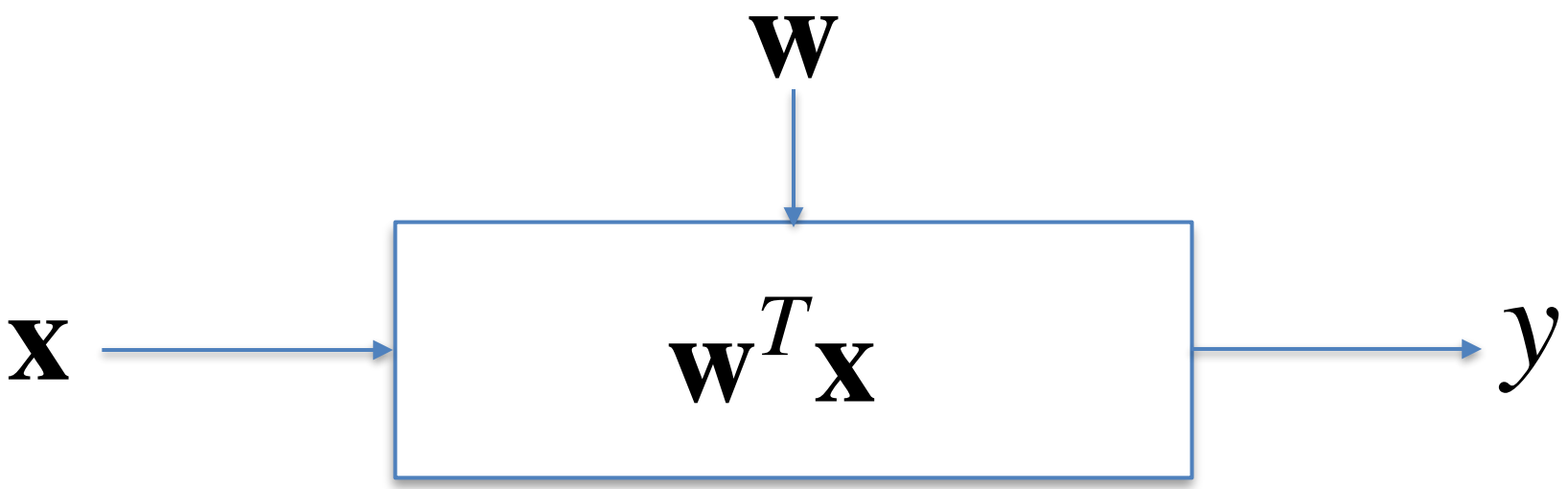
$$\frac{\partial E}{\partial w_j} = \frac{1}{2} \sum_i \frac{\partial y^{(i)}}{\partial w_j} \frac{\partial E^{(i)}}{\partial y^{(i)}}$$

$$= - \sum_i x_j^{(i)} (t^{(i)} - y^{(i)})$$

- The batch delta rule changes the weights in proportion to their error derivates summed over all training cases

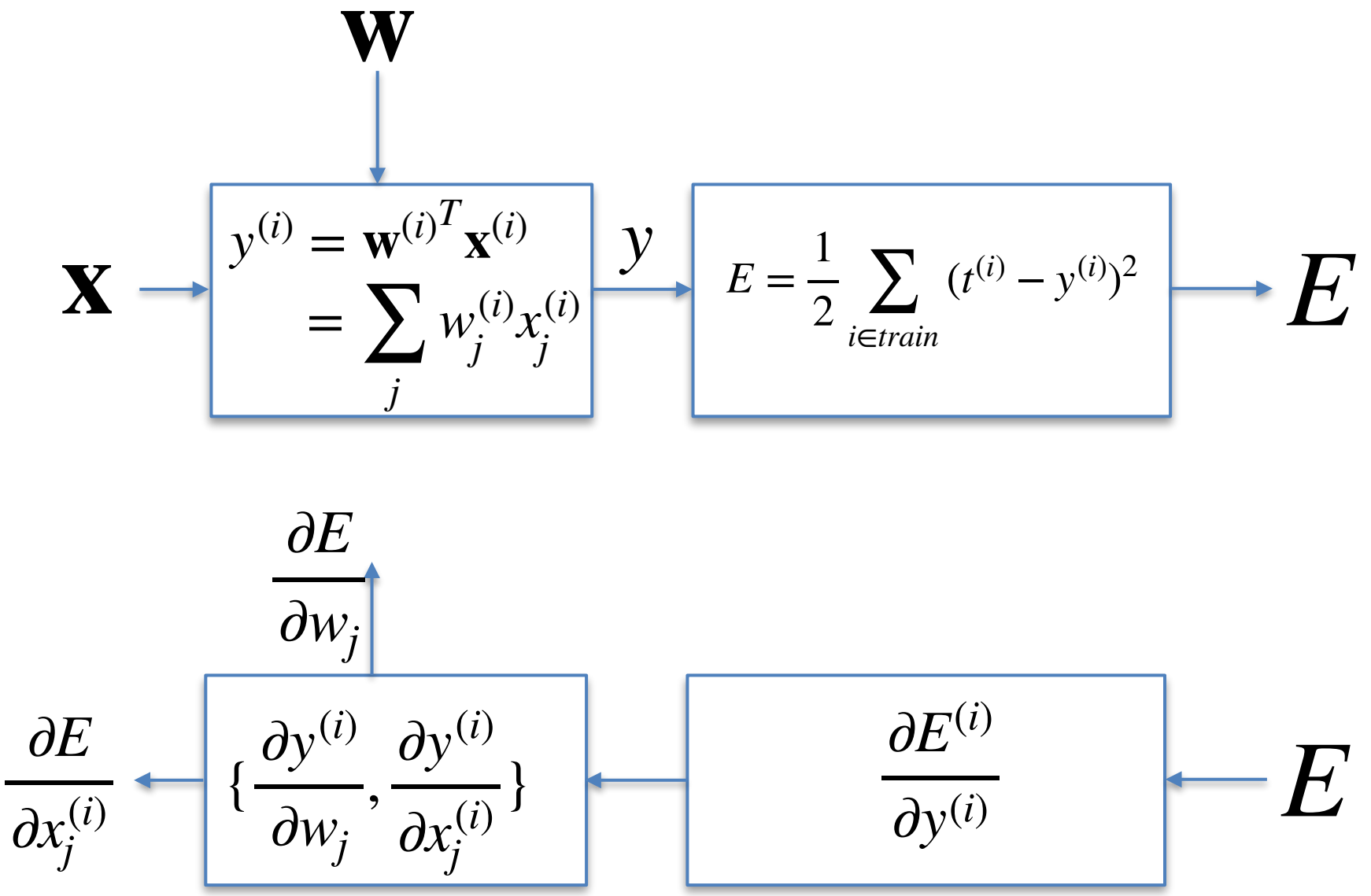
$$\Delta w_j = - \eta \frac{\partial E}{\partial w_j} = \eta \sum_i x_j^{(i)} (t^{(i)} - y^{(i)})$$

Key Computation: Forward-Prop



Key Computation: Back-Prop

$$\frac{\partial E}{\partial w_j} = \frac{1}{2}\sum_i \frac{\partial y^{(i)}}{\partial w_j}\frac{\partial E^{(i)}}{\partial y^{(i)}}$$



Gradient Descent

- For each iteration

  $\Delta w_j \leftarrow Initialize\ to\ ZEROS$

  - For each training example $(\mathbf{x}^{(i)}, t^{(i)})$
    - For each weight w_j

      $$\Delta w_j \leftarrow \Delta w_j + \eta x_j^{(i)}(t^{(i)} - y^{(i)})$$

  - For each weight w_j

    $$w_j \leftarrow w_j + \Delta w_j$$

$$\Delta w_j = \eta \sum_i x_j^{(i)}(t^{(i)} - y^{(i)})$$

Next…

- Multi-layer Neural Network