Numerical Methods and Machine Learning for Image Processing

Week 6, Class 1: Basic Neural Nets, Part 1a

November 1, 2021

Damon M. Chandler and Yi Zhang

Last time: Classification and Regression, Part 2b

1. Dimensionality reduction and SVC image classification
   - PCA—Principal Components Analysis
   - SVC on MINST database
2. Homework 2
   - Classification into three levels of importance
   - Use features that you measure via code

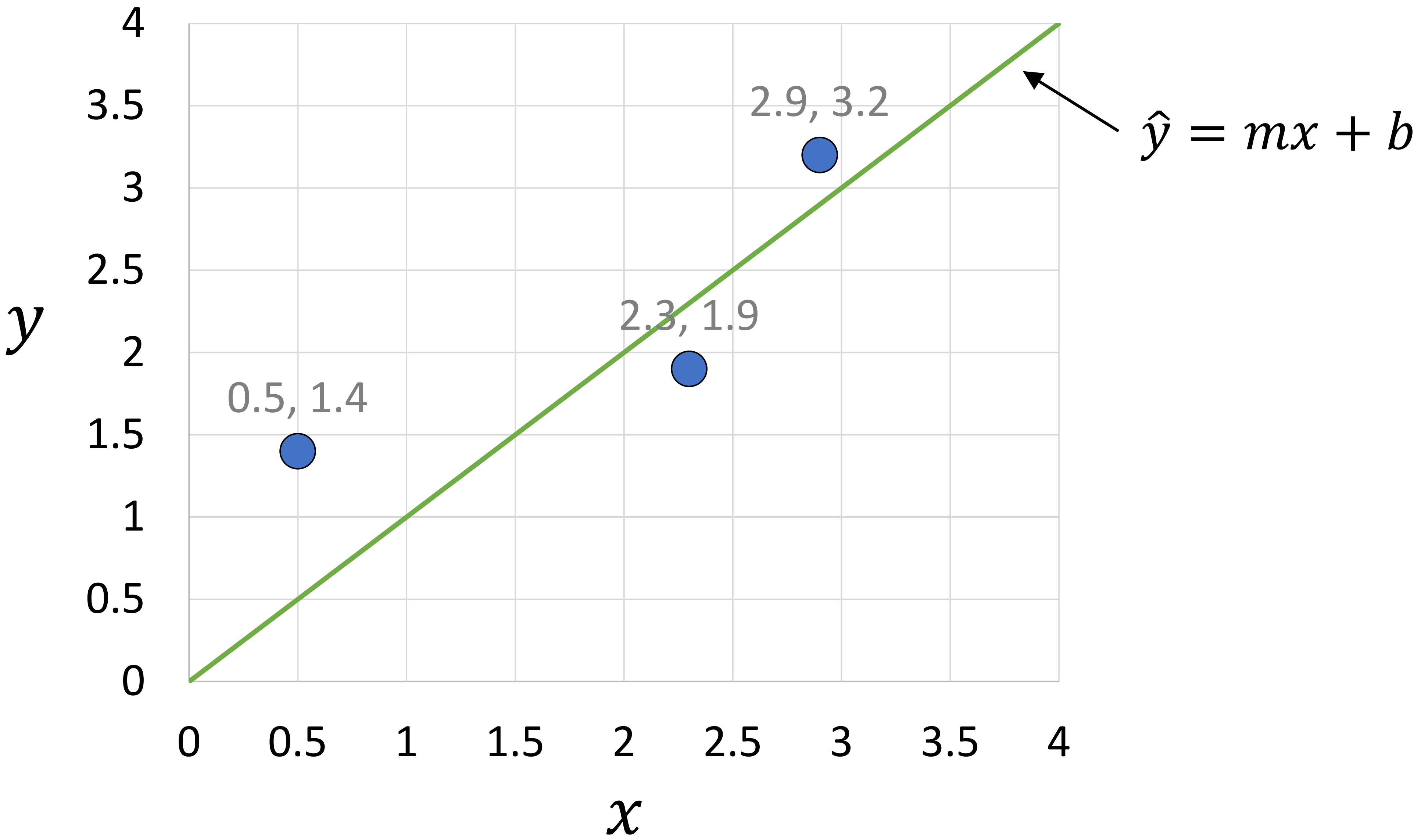
Today: Basic Neural Nets, Part 1a

- **Gradient descent**
- **Perceptron model**
  1. Background and history of neural networks
  2. Perceptron model math

Today: Basic Neural Nets, Part 1a

- **Gradient descent**
- **Perceptron model**
  1. Background and history of neural networks
  2. Perceptron model math

Gradient descent: *Fitting a linear function to data points*



Gradient descent: *Computing the gradients for a linear function*

$$E = y - \hat{y} = y - (mx + b) \qquad E^2 = (y - \hat{y})^2 = \big(y - (mx + b)\big)^2$$

$$SSE = \sum_{i=1}^{3} E_i^2 = \sum_{i=1}^{3} (y_i - \widehat{y_i})^2 = \sum_{i=1}^{3} \big(y_i - (mx_i + b)\big)^2$$

$$= \big(y_1 - (mx_1 + b)\big)^2 + \big(y_2 - (mx_2 + b)\big)^2 + \big(y_3 - (mx_3 + b)\big)^2$$

$$\frac{\partial}{\partial b}\{SSE\} = \frac{\partial}{\partial b}\left\{\sum_{i=1}^{3} E_i^2\right\} = \sum_{i=1}^{3} \frac{\partial}{\partial b}\{E_i^2\} = \sum_{i=1}^{3} \frac{\partial}{\partial b}\{(y_i - \widehat{y_i})^2\} = \sum_{i=1}^{3} \frac{\partial}{\partial b}\left\{\big(y_i - (mx_i + b)\big)^2\right\}$$

$$= \frac{\partial}{\partial b}\left\{\big(y_1 - (mx_1 + b)\big)^2\right\} + \frac{\partial}{\partial b}\left\{\big(y_2 - (mx_2 + b)\big)^2\right\} + \frac{\partial}{\partial b}\left\{\big(y_3 - (mx_3 + b)\big)^2\right\}$$

$$= 2\big(y_1 - (mx_1 + b)\big)(-1) + 2\big(y_2 - (mx_2 + b)\big)(-1) + 2\big(y_3 - (mx_3 + b)\big)(-1)$$

$$= -2\big(y_1 - (mx_1 + b)\big) - 2\big(y_2 - (mx_2 + b)\big) - 2\big(y_3 - (mx_3 + b)\big)$$

$$= -2\sum_{i=1}^{3} \big(y_i - (mx_i + b)\big)$$

Gradient descent: *Computing the gradients for a linear function*

$$E = y - \hat{y} = y - (mx + b) \qquad E^2 = (y - \hat{y})^2 = \big(y - (mx + b)\big)^2$$

$$SSE = \sum_{i=1}^{3} E_i^2 = \sum_{i=1}^{3} (y_i - \widehat{y_i})^2 = \sum_{i=1}^{3} \big(y_i - (mx_i + b)\big)^2$$

$$= \big(y_1 - (mx_1 + b)\big)^2 + \big(y_2 - (mx_2 + b)\big)^2 + \big(y_3 - (mx_3 + b)\big)^2$$

$$\frac{\partial}{\partial m}\{SSE\} = \frac{\partial}{\partial m}\left\{\sum_{i=1}^{3} E_i^2\right\} = \sum_{i=1}^{3} \frac{\partial}{\partial m}\{E_i^2\} = \sum_{i=1}^{3} \frac{\partial}{\partial m}\{(y_i - \widehat{y_i})^2\} = \sum_{i=1}^{3} \frac{\partial}{\partial m}\left\{\big(y_i - (mx_i + b)\big)^2\right\}$$

$$= \frac{\partial}{\partial m}\left\{\big(y_1 - (mx_1 + b)\big)^2\right\} + \frac{\partial}{\partial m}\left\{\big(y_2 - (mx_2 + b)\big)^2\right\} + \frac{\partial}{\partial m}\left\{\big(y_3 - (mx_3 + b)\big)^2\right\}$$

$$= 2\big(y_1 - (mx_1 + b)\big)(-x_1) + 2\big(y_2 - (mx_2 + b)\big)(-x_2) + 2\big(y_3 - (mx_3 + b)\big)(-x_3)$$

$$= -2\big(y_1 - (mx_1 + b)\big)x_1 - 2\big(y_2 - (mx_2 + b)\big)x_2 - 2\big(y_3 - (mx_3 + b)\big)x_3$$

$$= -2\sum_{i=1}^{3} \big(y_i - (mx_i + b)\big)x_i$$

Demo of gradient descent done in Excel.

See video recording of lecture and the accompanying Excel file.

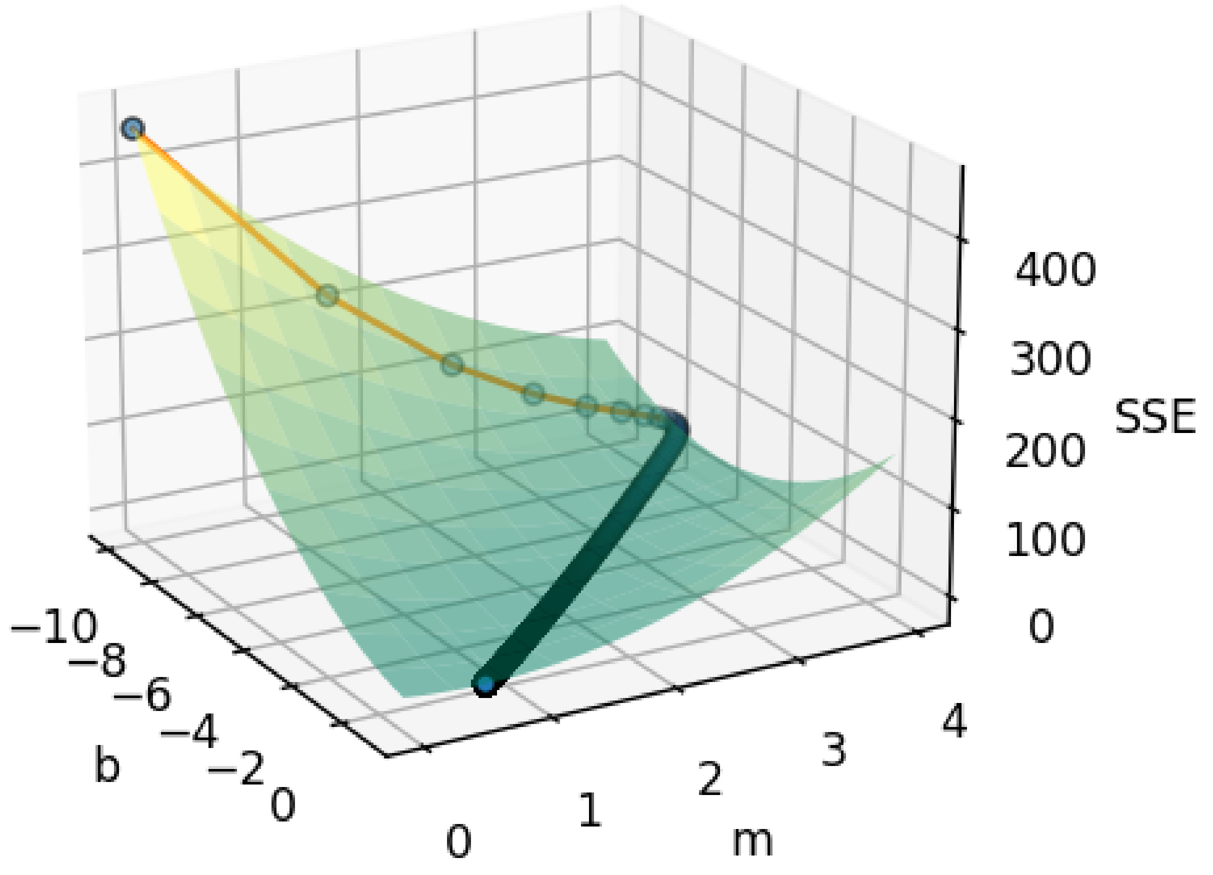
Gradient descent for linear fitting in Python

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


#%% DO THE GRADIENT DESCENT FITTING
x = np.array([0.5, 2.3, 2.9])
y = np.array([1.4, 1.9, 3.2])

df = pd.DataFrame(
  columns=["b", "m", "SSE", "grad_b", "next_b", "grad_m", "next_m"])

b = -10
m = 0
learning_rate = 0.01

for iter in range(1000):
  print("iter:", iter)

  y_hat = m*x + b
  SSE = ((y - y_hat)**2).sum()

  grad_b = -2*(y - y_hat).sum()
  grad_m = -2*((y - y_hat)*x).sum()
  next_b = b - learning_rate*grad_b
  next_m = m - learning_rate*grad_m

  df.loc[iter] = [b, m, SSE, grad_b, next_b, grad_m, next_m]

  if (np.abs(grad_b) < 0.001 and np.abs(grad_m) < 0.001):
    break

  b = next_b
  m = next_m

print("\nFinal optimized parameters:")
print("b =", "{:0.4f}".format(b))
print("m =", "{:0.4f}".format(m))

df.to_excel("linear_gradient_data.xlsx")
```

```
#%% PLOT SSE VS. B AND M
ax = plt.subplot(111, projection="3d")

B = np.linspace(df.iloc[:,0].min(), df.iloc[:,0].max(), 100)
M = np.linspace(df.iloc[:,1].min(), df.iloc[:,1].max(), 100)
B, M = np.meshgrid(B, M)

x = np.array([0.5, 2.3, 2.9])
y = np.array([1.4, 1.9, 3.2])

SSE = np.zeros((100, 100))
for r in range(100):
  for c in range(100):
    b = B[r, c]
    m = M[r, c]
    y_hat = m*x + b
    SSE[r, c] = ((y - y_hat)**2).sum()

ax.plot_surface(B, M, SSE, cmap="summer", rstride=10, cstride=10,
  alpha=0.5)

bs_tested = df.iloc[:, 0]
ms_tested = df.iloc[:, 1]
SSEs_tested = df.iloc[:, 2]

ax.scatter(bs_tested, ms_tested, SSEs_tested, edgecolors="k")
ax.plot(bs_tested, ms_tested, SSEs_tested)

ax.view_init(20, -30)

ax.set_xlabel("b")
ax.set_ylabel("m")
ax.set_zlabel("SSE")
```

Gradient descent for linear fitting in Python

```
iter: 0
iter: 1
iter: 2
iter: 3
iter: 4
iter: 5
iter: 6
iter: 7
iter: 8
iter: 9
iter: 10
iter: 11
iter: 12
iter: 13
iter: 14
iter: 15
iter: 16
iter: 17
iter: 18
iter: 19
iter: 20
iter: 21
iter: 22
iter: 23
iter: 24
show more (open the raw output data in a text editor) ...

iter: 865

Final optimized parameters:
b = 0.9483
m = 0.6412
```

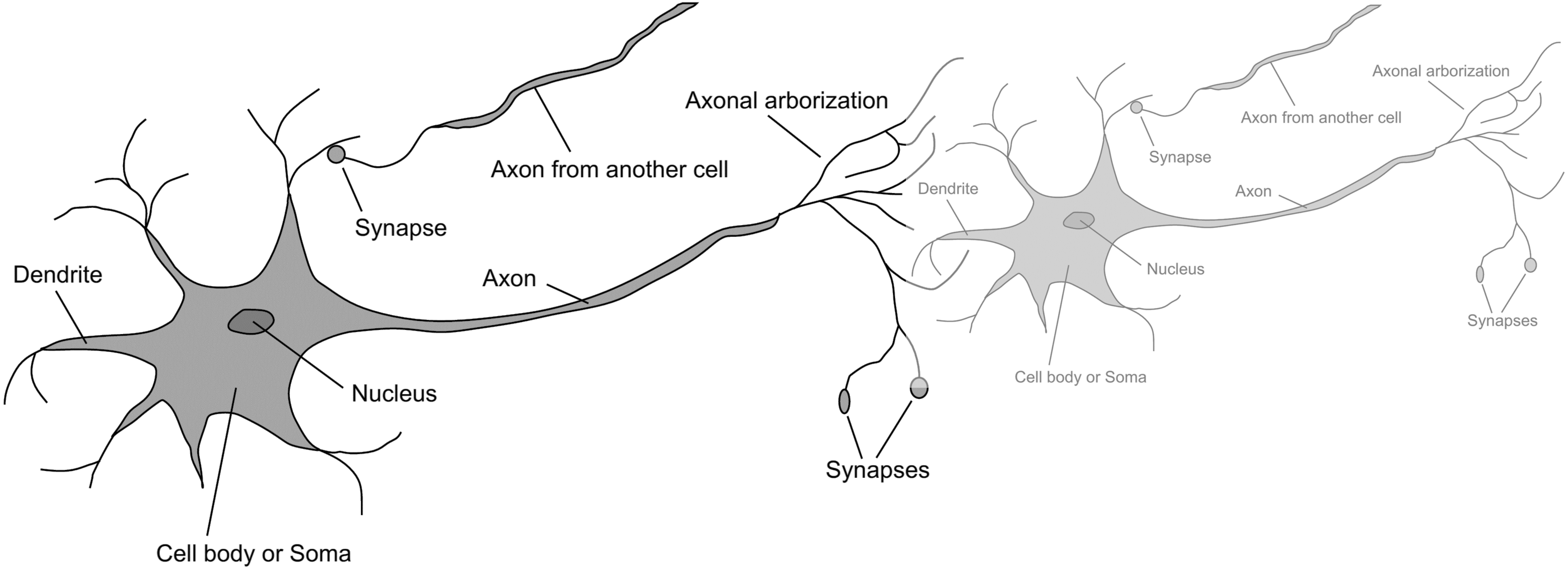


Today: Basic Neural Nets, Part 1a

- **Gradient descent**
- **Perceptron model**
  1. Background and history of neural networks
  2. Perceptron model math

What is a neural network?

- NN = Neural Network (or ANN = Artificial Neural Network)
- Collection of interconnected artificial neurons (neural cells)
- Uses simplified mathematical models of neurons



1940s – 1950s: The birth of AI

- 1943: McCulloch and Pitts
  - Binary model of a neuron
  - Any computable function could be computed by some network of connected neurons
  - 1949: Donald Hebb demonstrated an updating rule for changing the connection strengths (learning)
- 1950: Alan Turing
  - British mathematician
  - Famous paper: “Computing Machinery and Intelligence”
  - Don’t ask ‘Can machines think?’; instead, ask ‘Can machines pass a behavior test for intelligence?’
- 1950: Claude Shannon
  - Bell Labs in USA; founder of information theory
  - A typical chess game involved about 10^{120} possible moves
  - Proved that heuristics were needed for chess (searching all possible moves would take forever)
- 1951: Marvin Minsky and Dean Edmonds
  - Mathematics grad students; built the first neural network
- 1956: Dartmouth Workshop sponsored by IBM (organized by John McCarthy)
  - Two-month summer workshop that brought together machine intelligence researchers
  - Agreement to adopt new name for the field: *artificial intelligence*

Late 1950s – 1960s: The rise and hope of AI

- 1958: John McCarthy
  - Invented LISP programming language
  - Wrote paper: “Programs with Common Sense”
  - Wrote program to generate driving route planning using simple axioms
- 1958-1962: Frank Rosenblatt
  - Improved learning methods in NNs
  - 1958: Invented the *perceptron* algorithm
  - 1962: Proved perceptron convergence theorem: A learning algorithm can adjust the connection strengths of a perceptron
- 1959: Herbert Gelernter
  - Geometry Theorem Prover (used axioms to prove theorems)
- 1961: Allen Newell and Herbert Simon
  - General Problem Solver program
  - Designed to simulate human problem-solving skills
- In summary, researchers tried to simulate the complex thinking by inventing general methods
  - Used general-purpose search mechanism to find a solution to the problem
  - These approaches are now called “weak methods” (used weak information about problem domain → weak performance)

Late 1960s – 1970s: Unfulfilled promises

AI methods in the 1960s suffered from **three main problems**:

1. Contained little or no knowledge
   - Basically used simple rules and/or search strategies
   - US-govt.-funded AI failed miserably at language translation (Russian → English)
2. Could not deal with larger problems
   - Could not scale to harder problems (“combinatorial explosion”)
   - 1971, 1972: Theory of NP-completeness
     - Hard/intractable problems require times that are exponential functions of the problem size
3. Basic structures were too simple
   - Single-layer perceptron could not solve XOR problem (tell when its two inputs were different)

- 1966: US government cancelled funding of all machine translation research
- 1971: British government suspends funding of AI research
  - No major results from AI research → no need for a separate science
- 1970s: Research funding for NNs also dried up

Late 1970s – 1980s: Using *knowledge*

- 1970s: DENDRAL (“Dendritic Algorithm”)
  - First knowledge-based system
  - Developed at Stanford University to analyze chemicals
  - Rather than searching all possible molecular structures, use “knowledge”
  - “Knowledge” = “heuristics” and “rules of thumb” used by human experts
  - Paradigm shift in AI: From general-purpose, knowledge-sparse, weak methods to domain-specific, knowledge-intensive methods
- Many other successful rule-based expert systems
  - MYCIN: an expert system for the diagnosis of infectious blood diseases
    - About 450 rules. One example:

```
IF the site of the culture is blood AND
  the gram of the organism is neg AND
  the morphology of the organism is rod AND
  the burn of the patient is serious
THEN the identity of the organism might be pseudomonas (0.4; weakly suggestive evidence)
```

  - PROSPECTOR: an expert system for mineral exploration
  - Many others used widely in industry
  - Several limitations:
    - Restricted to a very narrow domain of expertise
    - Have little or no ability to learn from their experience
    - Have limited explanation capabilities

Mid 1980s – Early 2000s: Rebirth of NNs

- 1980: Grossberg
  - Established principle of self-organization (adaptive resonance theory)
- 1982: Hopfield
  - NNs with feedback (Hopfield networks)
- 1982: Kohonen
  - Self-organizing maps
- 1983: Barto, Sutton, and Anderson
  - Reinforcement learning
- 1986: Rumelhart and McClelland
  - "Parallel Distributed Processing: Explorations in the Microstructures of Cognition"
  - Back-propagation learning
  - Has become popular technique for training multilayer perceptrons
- Some limitations
  - Lacking mathematical rigor
  - Requires time-consuming training on lots of data (computers were still slow)

Mid 1980s – Early 2000s: Rebirth of NNs

- 1990: IEEE Neural Networks Council was created
- 2001: Transformed into IEEE Neural Networks Society
- 2005: Renamed to IEEE Computational Intelligence Society
- "**Computational intelligence** (CI) is the theory, design, application and development of biologically and linguistically motivated computational paradigms."

  https://cis.ieee.org/about/what-is-ci
- Based on three main pillars:
  1. Neural Networks
  2. Fuzzy Systems
  3. Evolutionary Computation

1990s-Present: Mathematics/statistics

- Machine learning
  - Train the computer to make decisions
  - Can be considered a subset of AI
  - Can be considered a superset of NNs
  - Classification and prediction

Some examples include:

- Decision trees
- Support-vector machines (SVMs)
- Logistic regression
- Genetic algorithms
- Hidden Markov models (HMMs)
  - Used for speech recognition
  - Widely used in bioinformatics
- Bayesian networks
  - Invented to deal with uncertain knowledge
  - Combines expert systems with NNs
  - Allows for learning from experience

2000s-Present: Big data and data mining

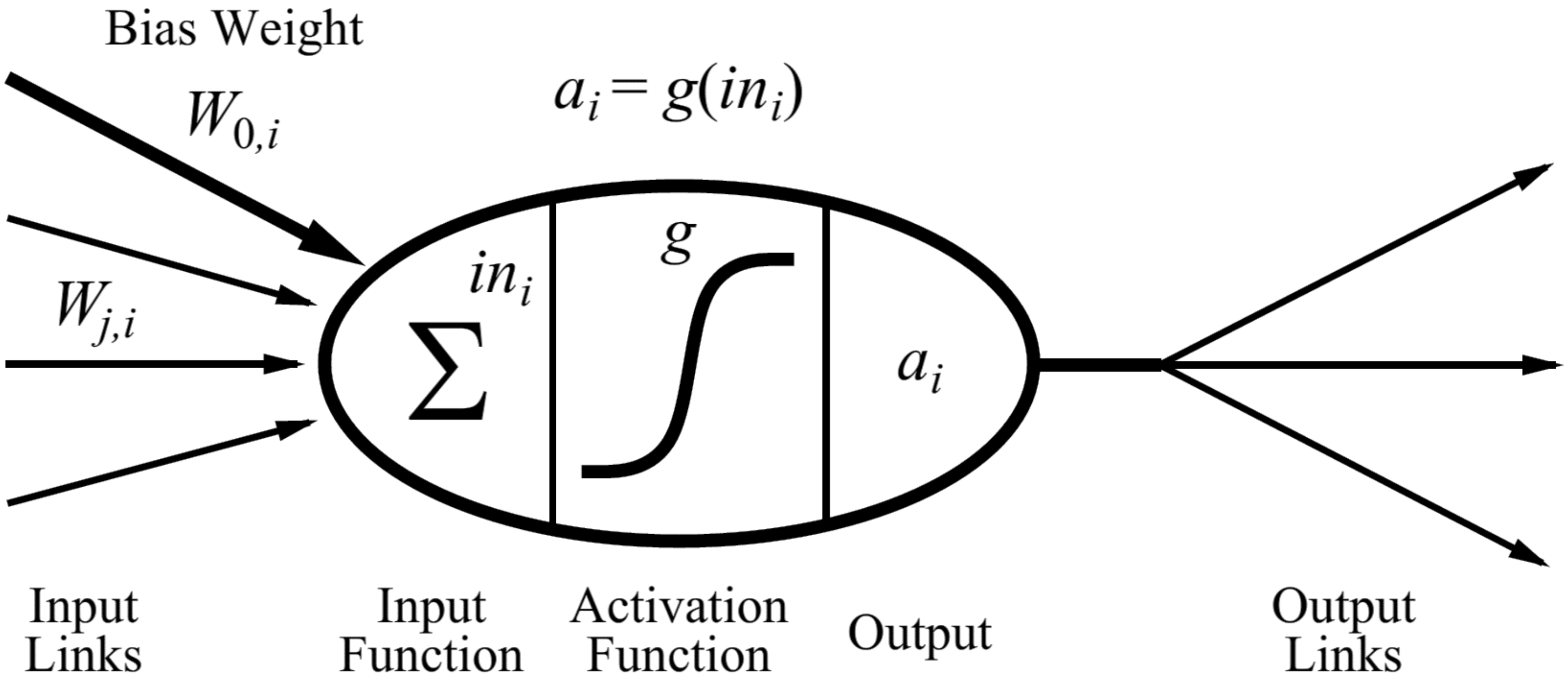
- Cheap sensors + internet → lots of readily available data
  - Trillions of words
  - Billions of images, videos
- Faster computers with many cores
- The "knowledge bottleneck" in AI (the problem of how to express all the knowledge that a system needs) may be solved by learning methods rather than hand-coded knowledge engineering
- Deep learning
  - First example: multilayer perceptrons
  - "Deep" = NNs with many layers
  - Main advantage: Eliminated the need for features as input (learns features from raw data)
  - Convolutional neural networks (CNNs)
  - Very popular in speech recognition, computer vision, image processing

Today: Basic Neural Nets, Part 1a

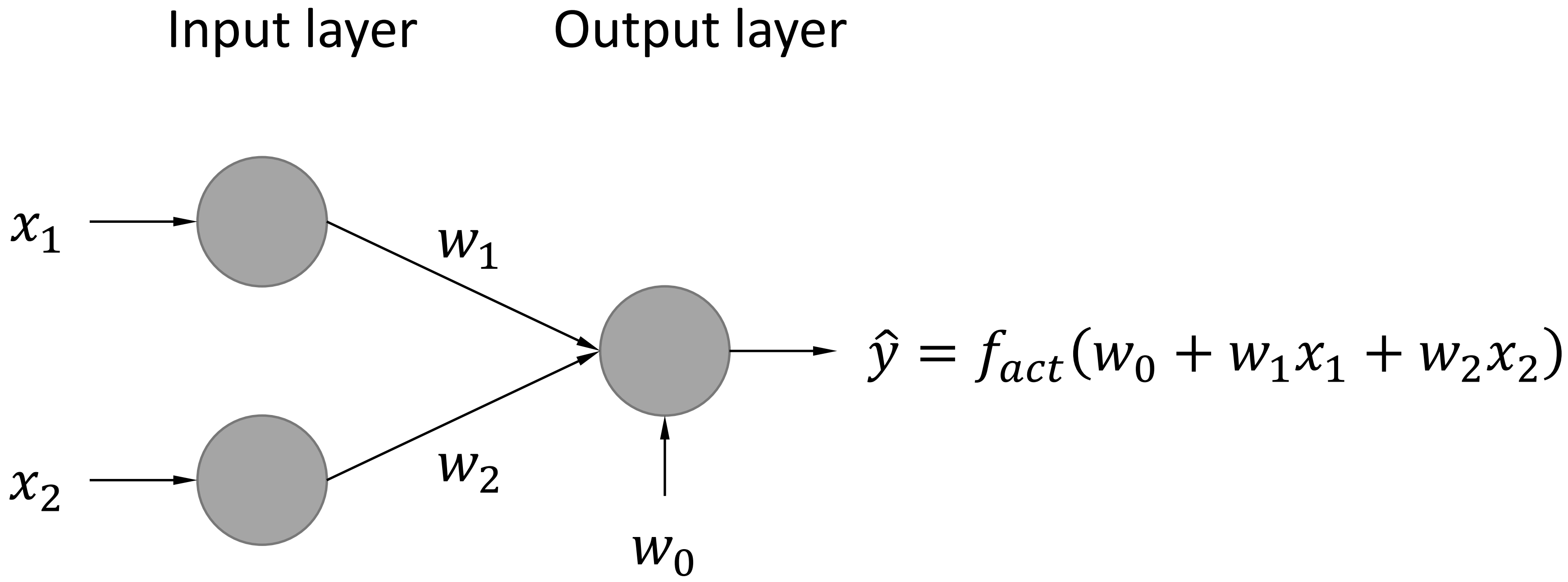
- **Gradient descent**
- **Perceptron model**

1. Background and history of neural networks
2. Perceptron model math

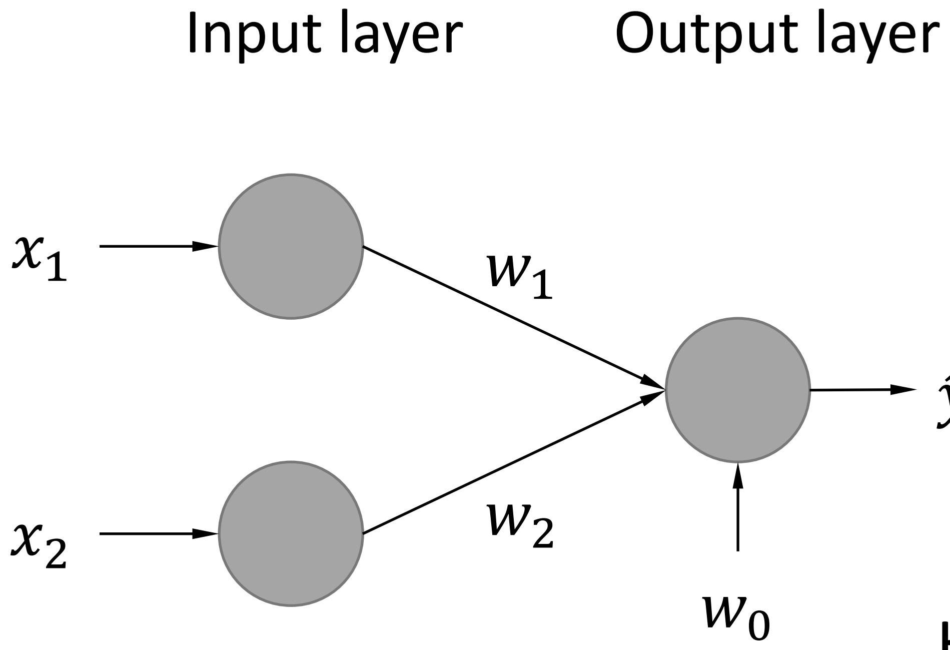
McCulloch–Pitts “unit” (1943)



Perceptron (1958)



Perceptron (1958)



$$\hat{y} = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

How to optimize the weights?

1. Compute error: $E = \frac{1}{2}(y - \hat{y})^2$

2. Adjust w_0, w_1, and w_2 based on the error gradient.

$$E = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}\left(y - \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}\right)^2$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_i} = -(y - \hat{y}) \frac{\partial \hat{y}}{\partial w_i} = -(y - \hat{y}) x_i \hat{y}(1 - \hat{y}) \qquad (x_0 = 1)$$

$$\frac{\partial E}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}} \left\{ \frac{1}{2}(y - \hat{y})^2 \right\} = 2\frac{1}{2}(y - \hat{y}) \cdot (-1) = -(y - \hat{y})$$

$$\frac{\partial \hat{y}}{\partial w_0} = \frac{1 e^{-(w_0 + w_1 x_1 + w_2 x_2)}}{(1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)})^2} = \hat{y}(1 - \hat{y})$$

$$\frac{\partial \hat{y}}{\partial w_1} = \frac{x_1 e^{-(w_0 + w_1 x_1 + w_2 x_2)}}{(1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)})^2} = x_1 \hat{y}(1 - \hat{y})$$

$$\frac{\partial \hat{y}}{\partial w_2} = \frac{x_2 e^{-(w_0 + w_1 x_1 + w_2 x_2)}}{(1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)})^2} = x_2 \hat{y}(1 - \hat{y})$$

$$E = \frac{1}{2}(y-\hat{y})^2 = \frac{1}{2}\left(y - \frac{1}{1+e^{-(w_0+w_1x_1+w_2x_2)}}\right)^2$$

This slide shows the details of how to compute $\frac{\partial \hat{y}}{\partial w_i}$.

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_i} = -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_i}$$

$$\frac{\partial \hat{y}}{\partial w_i} = \frac{\partial}{\partial w_i}\left\{\frac{1}{1+e^{-(w_0+w_1x_1+w_2x_2)}}\right\}$$

Note: $\frac{d}{dw}\left(\frac{1}{1+\beta e^{-\alpha w}}\right) = \frac{\alpha\beta e^{-\alpha w}}{(1+\beta e^{-\alpha w})^2}$

$$\frac{\partial \hat{y}}{\partial w_0} = \frac{1e^{-(w_0+w_1x_1+w_2x_2)}}{(1+e^{-(w_0+w_1x_1+w_2x_2)})^2}$$

Note: $\frac{a}{(1+a)^2} = \frac{1}{1+a}\left(1-\frac{1}{1+a}\right)$

$$= \frac{1}{1+e^{-(w_0+w_1x_1+w_2x_2)}}\left(1-\frac{1}{1+e^{-(w_0+w_1x_1+w_2x_2)}}\right) = \hat{y}(1-\hat{y})$$

$$\frac{\partial \hat{y}}{\partial w_1} = \frac{x_1e^{-(w_0+w_1x_1+w_2x_2)}}{(1+e^{-(w_0+w_1x_1+w_2x_2)})^2} = x_1\hat{y}(1-\hat{y})$$

$$\frac{\partial \hat{y}}{\partial w_2} = \frac{x_2e^{-(w_0+w_1x_1+w_2x_2)}}{(1+e^{-(w_0+w_1x_1+w_2x_2)})^2} = x_2\hat{y}(1-\hat{y})$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_i} = -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_i} = -(y-\hat{y})x_i\hat{y}(1-\hat{y}) \qquad (x_0 = 1)$$

How to change the weights (*update rule*):

$$w_i = w_i - \eta\frac{\partial E}{\partial w_i} = w_i + \eta(y-\hat{y})x_i\hat{y}(1-\hat{y})$$

```
for X_row, y_row in zip(X, y):
   hat_y_row = neural_response(X_row, w)

   err = y_row - hat_y_row
   sse += err**2

   delta = err*hat_y_row*(1.0 - hat_y_row)

   w[0] = w[0] + lrate*delta
   for i in range(len(X_row)):
     w[i+1] = w[i+1] + lrate*delta*X_row[i]
```

$$\overbrace{\texttt{X[:,0]} \quad \texttt{X[:,1]} \quad \texttt{X[:,2]} \quad \texttt{X[:,3]}}^{\mathbf{X}} \quad \texttt{y}$$

wav_var	wav_skw	wav_krt	pix_ent	class
0.964	5.616	2.214	-0.125	0
0.259	5.010	-5.039	-6.386	1
0.331	4.573	2.057	-0.190	0
-0.531	-0.097	-0.218	1.043	1
-3.137	0.422	2.623	-0.064	1
-7.042	9.200	0.259	-4.683	1
3.184	7.232	-1.071	-2.591	0
-1.119	3.336	-1.346	-1.957	1
-0.234	3.241	-3.067	-2.778	1
-1.279	-2.409	4.574	0.476	1
-2.410	3.743	-0.402	-1.295	1
-0.394	-0.021	-0.066	-0.447	1
-2.380	-1.440	1.127	0.161	1
3.776	7.178	-1.520	0.401	0

We will look at the code next time.