

Lecture 15: Neural Networks

COMP90049

Introduction to Machine Learning

Semester 1, 2024

Ting Dang, CIS

Copyright @ University of Melbourne 2024. All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm or any other means without written permission from the author.

Acknowledgement: Lea Frermann



So far ... Classification and Evaluation

- KNN, Naive Bayes, Logistic Regression, Perceptron
- Probabilistic models
- Loss functions, and estimation
- Evaluation

Today... Neural Networks

- Multilayer Perceptron
- Motivation and architecture
- Linear vs. non-linear classifiers



Perceptron

$$\hat{y} = f(\theta \cdot x) = \begin{cases} 1 & \text{if } \theta \cdot x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

- Single processing 'unit'
- Inspired by neurons in the brain
- Objective: minimize errors
- Activation: step-function (discrete, non-differentiable)

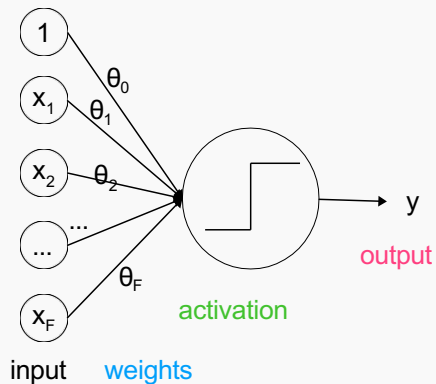
Logistic Regression

$$P(y = 1|x; \theta) = \frac{1}{1 + \exp(-(\sum_{f=0}^F \theta_f x_f))}$$

- View 1: Model of $P(y = 1|x)$
- View 2: Single processing 'unit'
- Objective: minimize the negative data log likelihood
- Activation: sigmoid (continuous, differentiable)



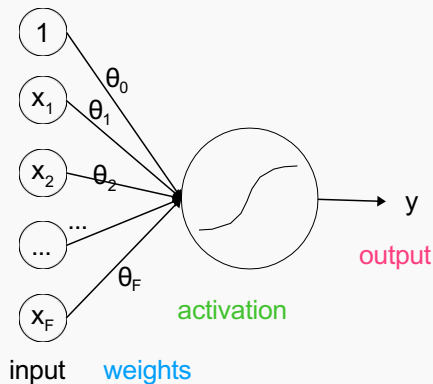
Perceptron and Logistic Regression



$$y = f(\theta^T x)$$

$$f: \begin{array}{ll} y = 1 & \text{if } f(\theta^T x) \geq 0 \\ y = -1 & \text{if } f(\theta^T x) < 0 \end{array}$$

Perceptron and Logistic Regression

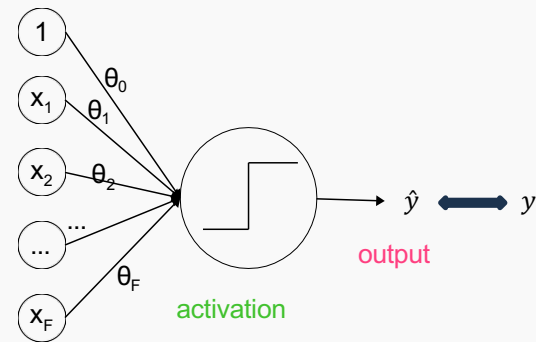


$$y = f(\theta^T x)$$

$$f: y = \frac{1}{(1 + e^{-\theta^T x})}$$

$$\begin{aligned} f: y &> 0.5 && \text{if } \theta^T x > 0 \\ y &< 0.5 && \text{if } \theta^T x < 0 \\ y &= 0.5 ? && \text{if } \theta^T x = 0 \end{aligned}$$

Perceptron and Logistic Regression



input weights

if $y = 1$ and $\hat{y} = 1$ then **do nothing**

if $y = -1$ and $\hat{y} = -1$ then **do nothing**

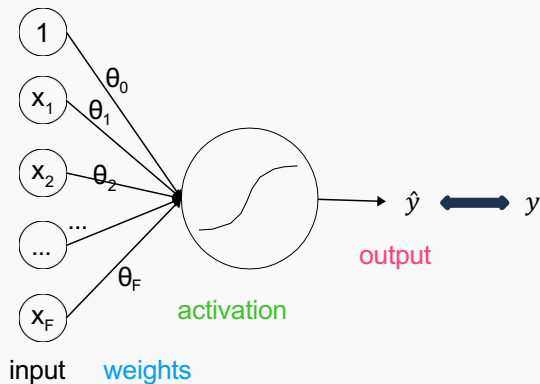
if $y = 1$ but $\hat{y} = -1$ then **increase** weights

if $y = -1$ but $\hat{y} = 1$ then **decrease** weights

Perceptron Rule

$$\theta_j^t \leftarrow \theta_j^{t-1} + \eta(y^i - \hat{y}^i)x_j^i$$

Perceptron and Logistic Regression



$$L(\theta) = -P(y|x; \theta)$$

Gradient descent

$$\theta^{t+1} \leftarrow \theta^t - \eta \frac{\partial L(\theta)}{\partial \theta^t}$$

Perceptron

$$\hat{y} = f(\theta \cdot x) = \begin{cases} 1 & \text{if } \theta \cdot x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

- Single processing 'unit'
- Inspired by neurons in the brain
- Objective: minimize errors
- Activation: step-function (discrete, non-differentiable)

Logistic Regression

$$P(y = 1|x; \theta) = \frac{1}{1 + \exp(-(\sum_{f=0}^F \theta_f x_f))}$$

- View 1: Model of $P(y = 1|x)$
- View 2: Single processing 'unit'
- Objective: minimize the negative data log likelihood
- Activation: sigmoid (continuous, differentiable)



Neural Networks: Motivation and Intuition

(Non-linear) classification with the Multilayer Perceptron

Feature learning with neural networks

Designing Neural Networks

Reflections

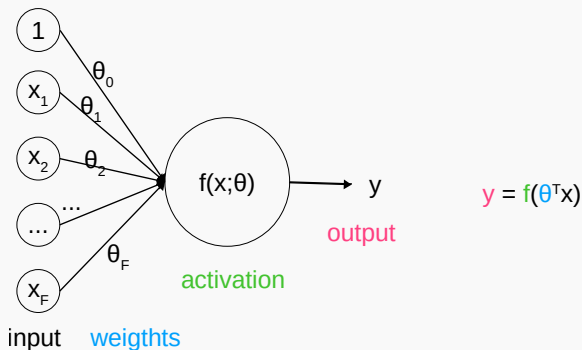
Activation Functions and refresher on derivatives



Neural Networks: Motivation and Intuition

Perceptron Unit (recap)

A single processing unit



A **neural network** is a combination of lots of these units.

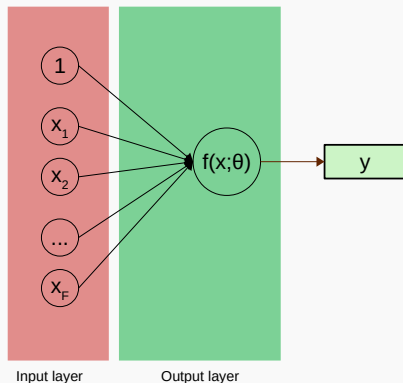
- Units must have continuous activation functions
- Connected into many layers → **Deep** Learning



Multi-layer Perceptron (schematic)

Three Types of layers

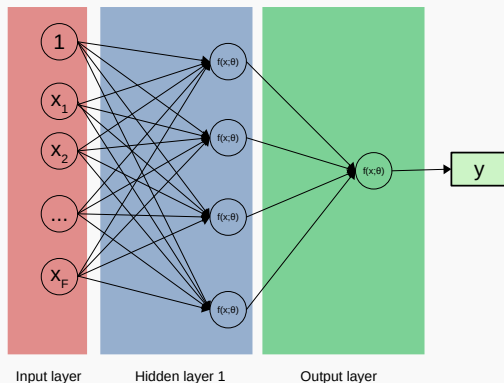
- **Input layer** with input units x : the first layer, takes features x as inputs
- **Output layer** with output units y : the last layer, has one unit per possible output (e.g., 1 unit for binary classification)
- **Hidden layers** with hidden units h : all layers in between.



Multi-layer Perceptron (schematic)

Three Types of layers

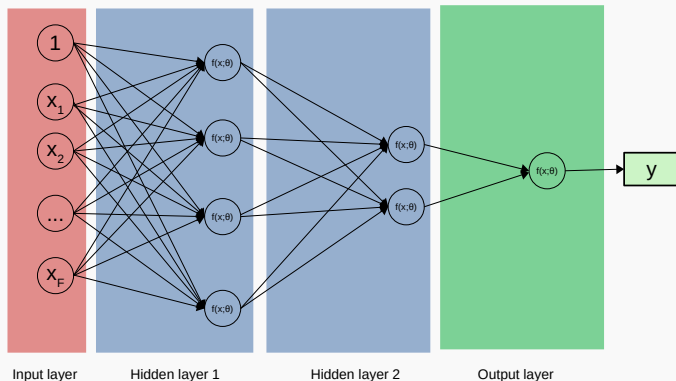
- **Input layer** with input units x : the first layer, takes features x as inputs
- **Output layer** with output units y : the last layer, has one unit per possible output (e.g., 1 unit for binary classification)
- **Hidden layers** with hidden units h : all layers in between.



Multi-layer Perceptron (schematic)

Three Types of layers

- **Input layer** with input units x : the first layer, takes features x as inputs
- **Output layer** with output units y : the last layer, has one unit per possible output (e.g., 1 unit for binary classification)
- **Hidden layers** with hidden units h : all layers in between.



Multi-layer Perceptron

- One specific type of neural network
- Feed-forward
- Fully connected
- Supervised learner

Other types of neural networks

- Convolutional neural networks
- Recurrent neural networks
- Autoencoder (unsupervised)

Linear classification

- The perceptron, naive bayes, logistic regression are linear classifiers
- Decision boundary is a linear combination of features $\sum_i \theta_i x_i$
- Cannot learn 'feature interactions' naturally

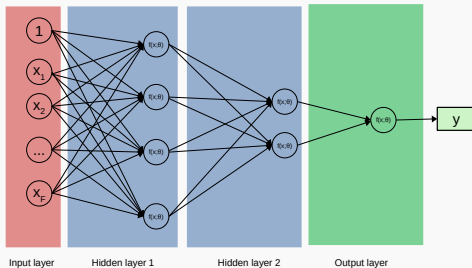
Non-linear classification

- Neural networks with at least 1 hidden layer and non-linear activations
- Decision boundary is a non-linear function of the inputs

(Non-linear) classification with the Multilayer Perceptron

Multilayer Perceptron: Terminology

- input units x_j , one per feature j
- Multiple **layers** $l = 1 \dots L$ of nodes. L is the **depth** of the network.
- Each layer l has a number of units K_l . K_l is the **width** of layer l .
- The width can vary from layer to layer
- output unit y
- Each layer l is **fully connected** to its neighboring layers $l - 1$ and $l + 1$
- one weight $\theta_{ij}^{(l)}$ for each connection ij (including 'bias' θ_0)
- non-linear activation function for layer l as $\phi^{(l)}$



Passing an input through a neural network with 2 hidden layers

$$h_i^{(1)} = \phi^{(1)} \left(\sum_j \theta_{ij}^{(1)} x_j \right)$$

$$h_i^{(2)} = \phi^{(2)} \left(\sum_j \theta_{ij}^{(2)} h_j^{(1)} \right)$$

$$y_i = \phi^{(3)} \left(\sum_j \theta_{ij}^{(3)} h_j^{(2)} \right)$$

Passing an input through a neural network with 2 hidden layers

$$h_i^{(1)} = \phi^{(1)}\left(\sum_j \theta_{ij}^{(1)} x_j\right)$$

$$h_i^{(2)} = \phi^{(2)}\left(\sum_j \theta_{ij}^{(2)} h_j^{(1)}\right)$$

$$y_i = \phi^{(3)}\left(\sum_j \theta_{ij}^{(3)} h_j^{(2)}\right)$$

Or in vectorized form

$$h^{(1)} = \phi^{(1)}\left(\theta^{(1)T} x\right)$$

$$h^{(2)} = \phi^{(2)}\left(\theta^{(2)T} h^{(1)}\right)$$

$$y = \phi^{(3)}\left(\theta^{(3)T} h^{(2)}\right)$$

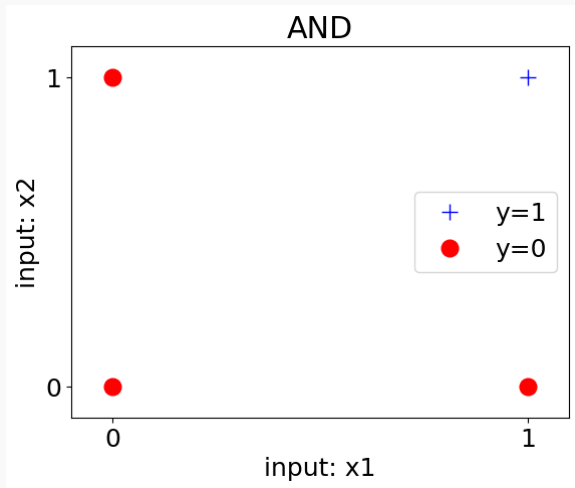
where the activation functions $\phi^{(l)}$ are applied **element-wise** to all entries



Toy example: Boolean Functions

1. Can the **perceptron** learn this function? Why (not)?
2. Can a **multilayer perceptron** learn this function? Why (not)?

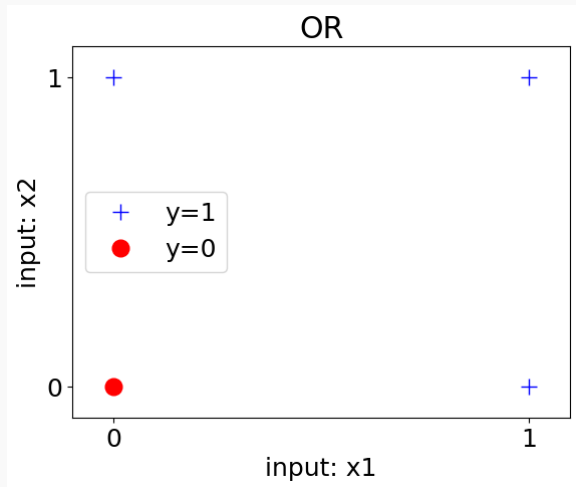
x_1	x_2	y
1	1	1
1	0	0
0	1	0
0	0	0



Toy example: Boolean Functions

1. Can the **perceptron** learn this function? Why (not)?
2. Can a **multilayer perceptron** learn this function? Why (not)?

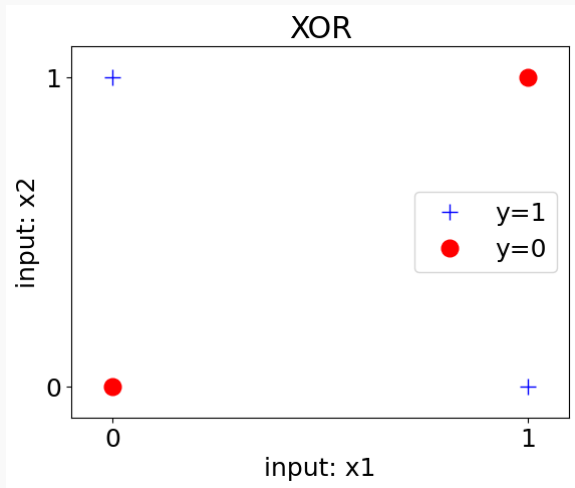
x_1	x_2	y
1	1	1
1	0	1
0	1	1
0	0	0



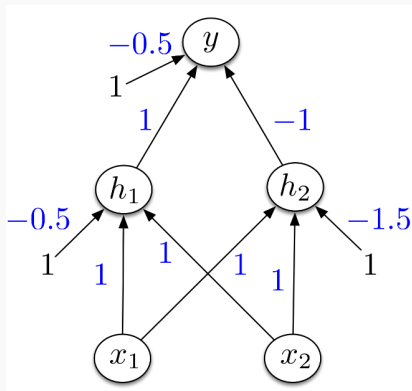
Toy example: Boolean Functions

1. Can the **perceptron** learn this function? Why (not)?
2. Can a **multilayer perceptron** learn this function? Why (not)?

x_1	x_2	y
1	1	0
1	0	1
0	1	1
0	0	0



A Multilayer Perceptron for XOR



$$\phi(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \quad \text{and recall: } h_i^{(l)} = \phi^{(l)} \left(\sum_j \theta_{ij}^{(l)} h_j^{(l-1)} + b_i^{(l)} \right)$$

Source: https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/readings/L05%20Multilayer%20Perceptrons.pdf

[//www.cs.toronto.edu/~rgrosse/courses/csc321_2018/readings/L05%20Multilayer%20Perceptrons.pdf](https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/readings/L05%20Multilayer%20Perceptrons.pdf)



Feature learning with neural networks

Feature Engineering

- The perceptron, naive Bayes and logistic regression require a fixed set of informative features
- e.g., `outlook` $\in \{overcast, sunny, rainy\}$, `wind` $\in \{high, low\}$, ...
- Requiring **domain knowledge**

Feature learning

- Neural networks take as input 'raw' numeric data
- They learn features as intermediate representations as part of their target task (e.g., classification)
- **Representation learning**: learning abstract features that are informative for the target task
- Note: feature engineering is often replaced at the cost of additional parameter tuning (layers, activations, learning rates, ...)



Example Classification dataset

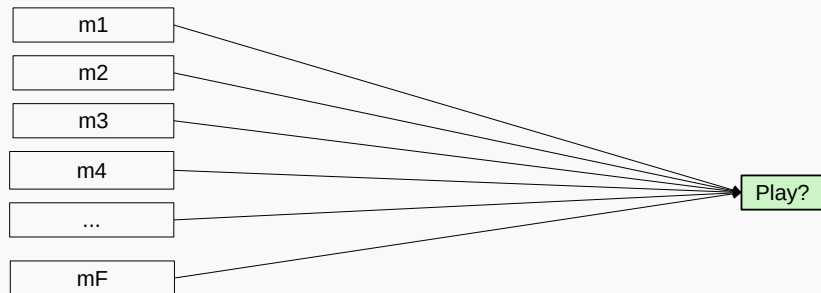
Outlook	Temperature	Humidity	Windy	True Label
sunny	hot	high	FALSE	no
sunny	hot	high	TRUE	no
overcast	hot	high	FALSE	yes
rainy	mild	high	FALSE	yes
...				

We really observe raw data

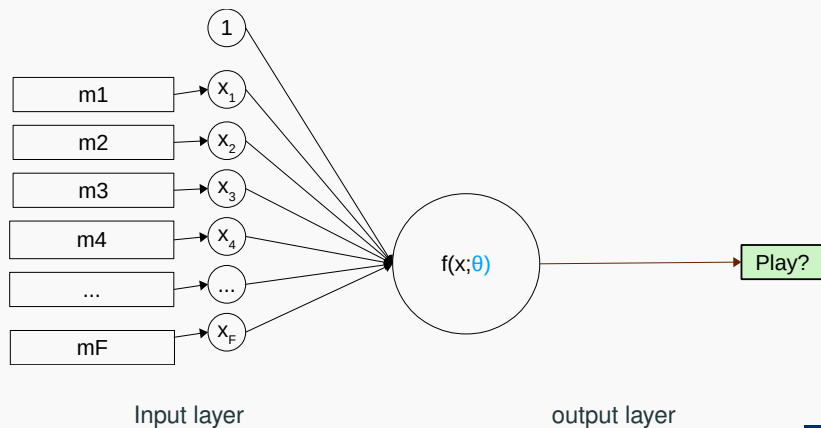
measurements					True Label
m1	m2	m3	m4	...	
0.4	4.7	1.5	12.7	...	no
3.4	-0.7	3.8	18.7	...	no
0.3	8.7	136.9	17	...	yes
5.5	5.7	65.5	2.7	...	yes



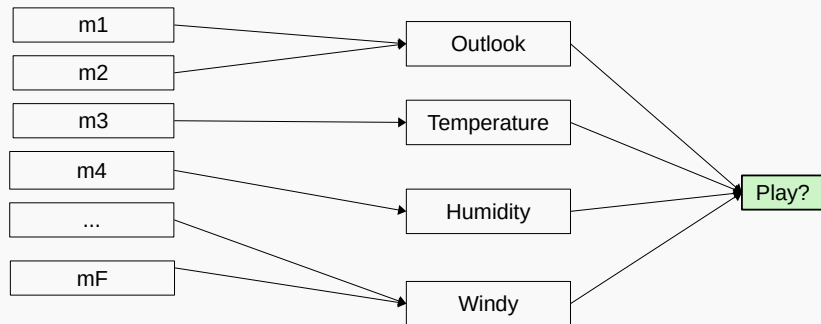
Example Problem: Weather Dataset



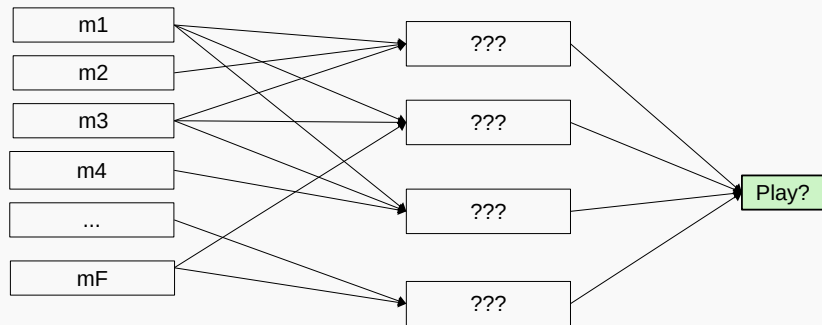
Example Problem: Weather Dataset



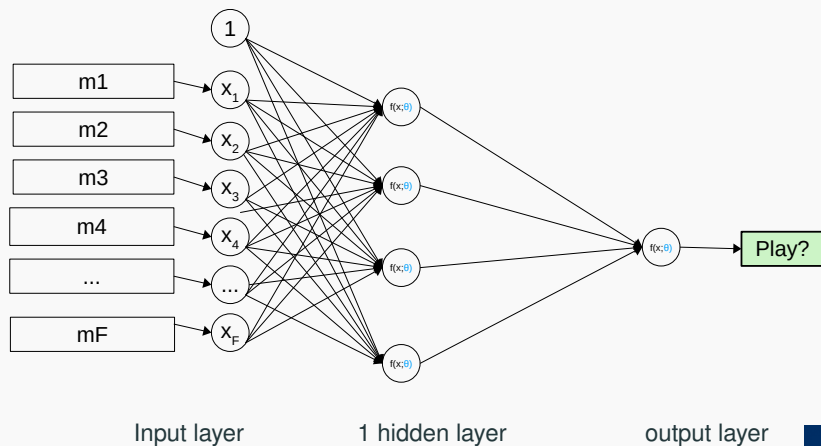
Example Problem: Weather Dataset



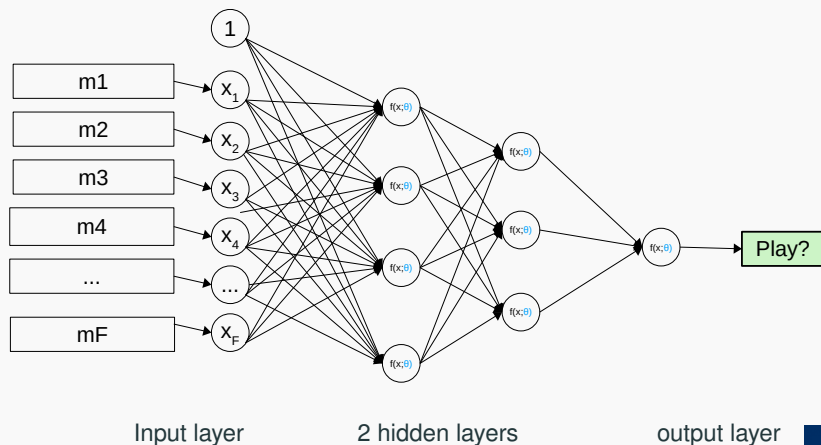
Example Problem: Weather Dataset



Example Problem: Weather Dataset

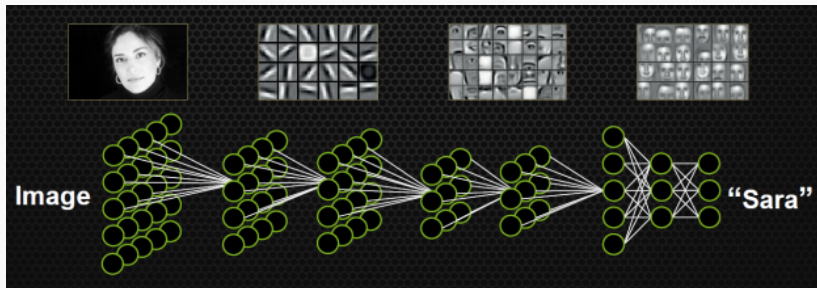


Example Problem: Weather Dataset



Feature learning III: Face Recognition example

- the **hidden layers** learn increasingly high-level feature representations
- e.g., given an image, predict the person:



Source: <https://devblogs.nvidia.com/accelerate-machine-learning-cudnn-deep-neural-network-library/>

Designing Neural Networks

Inputs and feature functions

- x could be a patient with features {blood pressure, height, age, weight, ...}
- x could be a texts, i.e., a sequence of words
- x could be an image, i.e., a matrix of pixels

Non-numerical features need to be mapped to numerical

- For language, typical to map words to **pre-trained embedding vectors**
 - for 1-hot: $\dim(x) = V$ (words in the vocabulary)
 - for embedding: $\dim(x) = k$, dimensionality of embedding vectors
- Alternative: **1-hot encoding**
- For pixels, map to RGB, or other visual features



Network Structure

- Sequence of hidden layers l_1, \dots, l_L for a network of depth L
- Each layer l has K_l parallel neurons (breadth)
- Many layers (depth) vs. many neurons per layer (breadth)? Empirical question, theoretically poorly understood.

Advanced tricks include allowing for exploiting data structure

- convolutions (convolutional neural networks; CNN), Computer Vision
- recurrences (recurrent neural networks; RNN), Natural Language Processing
- attention (efficient alternative to recurrences)
- ...

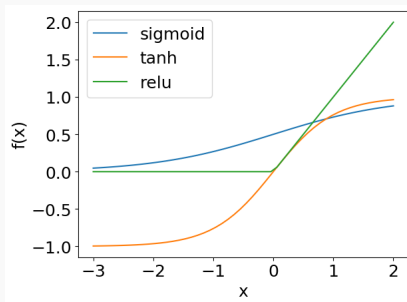
Beyond the scope of this class.



Activation Functions

Each layer has an associated activation function which performs a **non-linear transformation** of the input. Popular choices include:

- Logistic or Sigmoid (“ σ ”): $f(x) = \frac{1}{1+e^{-x}}$
- Hyperbolic tan (“tanh”): $f(x) = \frac{e^{2x}-1}{e^{2x}+1}$
- Rectified linear unit (“ReLU”): $f(x) = \max(0, x)$



Neural networks can learn different concepts: **classification**, **regression**, ...
The **output function** depends on the concept of interest.

- **Binary classification:**

- one neuron, with step function (as in the perceptron)

- **Multiclass classification:**

- typically **softmax** to normalize K outputs from the pre-final layer into a probability distribution over classes

$$p(y_i = j | x_i; \theta) = \frac{\exp(z_j)}{\sum_{k=1}^K \exp(z_k)}$$

- **Regression:**

- identity function
- possibly other continuous functions such as sigmoid or tanh



Classification Loss: negative conditional log-likelihood (cross-entropy)

$$\mathcal{L}^i = -\log p(y^{(i)}|x^{(i)}; \theta) \quad \text{for a single instance } i$$

$$\mathcal{L} = -\sum_i \log p(y^{(i)}|x^{(i)}; \theta) \quad \text{for all instances}$$

- Binary classification loss

$$\hat{y}_1^{(i)} = p(y^{(i)} = 1|x^{(i)}; \theta)$$

$$\mathcal{L} = \sum_i -[y^{(i)} \log(\hat{y}_1^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}_1^{(i)})]$$

- Multiclass classification

$$\hat{y}_j^{(i)} = p(y^{(i)} = j|x^{(i)}; \theta)$$

$$\mathcal{L} = -\sum_i \sum_j y_j^{(i)} \log(\hat{y}_j^{(i)})$$

for j possible labels; $y_j^{(i)} = 1$ if j is the true label for instance i , else 0.



Regression Loss: typically mean-squared error (MSE)

- Here, the output, as well as the target are real-valued numbers

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N (y^{(i)} - \hat{y}^{(i)})^2$$

How to Train a NN with Hidden Layers

Unfortunately, the perceptron algorithm can't be used to train neural nets with hidden layers, as we can't directly observe the labels

Instead, train neural nets with **back propagation**. Intuitively:

- compute errors at the output layer wrt each weight using partial differentiation
- propagate those errors back to each of the input layers

Essentially just gradient descent, but using the chain rule to make the calculations more efficient

Next lecture: Backpropagation for training neural networks



Reflections

- The **universal approximation theorem** states that a feed-forward neural network with a single hidden layer (and finite neurons) is able to approximate any continuous function on \mathbb{R}^n
- Note that **the activation functions must be non-linear**, as without this, the model is simply a (complex) linear model

When is Linear Classification Enough?

- If we know our classes are linearly (approximately) separable
- If the feature space is (very) high-dimensional
...i.e., the number of features exceeds the number of training instances
- If the training set is small
- If *interpretability* is important, i.e., understanding how (combinations of) features explain different predictions

...but with increasing availability of data, and more powerful computers, **non-linear models** are gaining popularity, with currently **Neural Networks** being their most popular variant.



Why the success of Deep Learning?



Pros and Cons of Neural Networks

Pros

- Powerful tool!
- Neural networks with at least 1 hidden layer can approximate any (continuous) function. They are **universal approximators**
- Automatic feature learning
- Empirically, very good performance for many diverse tasks

Cons

- Powerful model increases the danger of 'overfitting'
- Requires large training data sets
- Often requires powerful compute resources (GPUs)
- Lack of interpretability

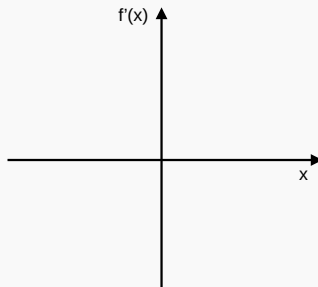
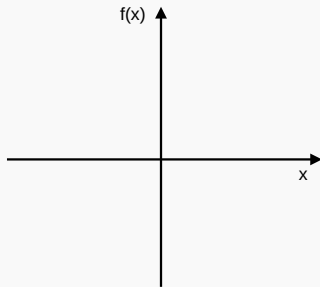


Activation Functions and refresher on derivatives

Activation Functions

Each layer has an associated activation function which performs a **non-linear transformation** of the input. Popular choices include:

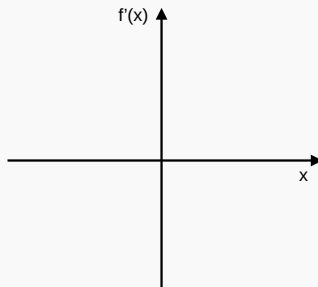
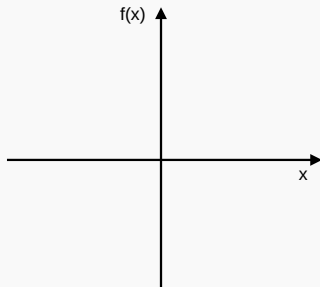
Logistic or Sigmoid (“ σ ”): $f(x) = \frac{1}{1+e^{-x}}$



Activation Functions

Each layer has an associated activation function which performs a **non-linear transformation** of the input. Popular choices include:

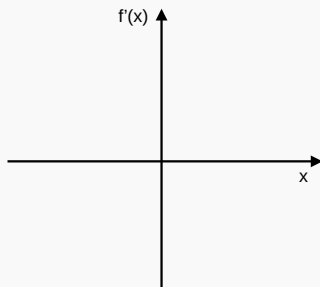
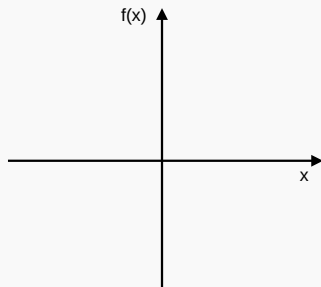
Hyperbolic tan ("tanh"): $f(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$



Activation Functions

Each layer has an associated activation function which performs a **non-linear transformation** of the input. Popular choices include:

Rectified linear unit (“ReLU”): $f(x) = \max(0, x)$



Today

- From perceptrons to neural networks
- multilayer perceptron
- some examples
- features and limitations

Next Lecture

- Learning parameters of neural networks
- The Backpropagation algorithm

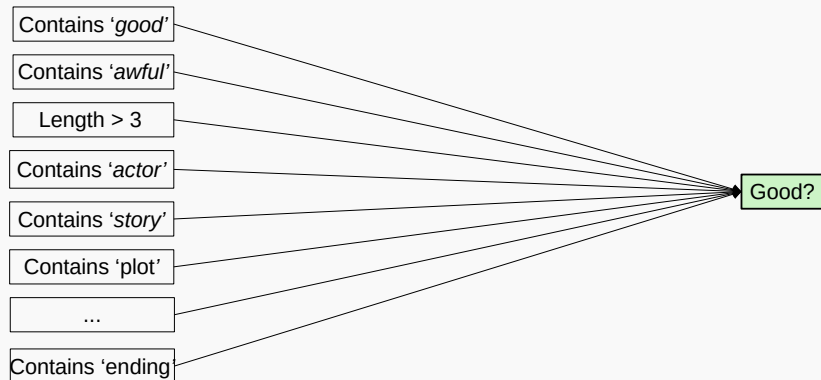
References

Jacob Eisenstein (2019). *Natural Language Processing*. MIT Press.
Chapters 3 (intro), 3.1, 3.2. <https://github.com/jacobeisenstein/gt-nlp-class/blob/master/notes/eisenstein-nlp-notes.pdf>

Dan Jurafsky and James H. Martin. *Speech and Language Processing*.
Chapter 7.2, 7.3. Online Draft V3.0.
<https://web.stanford.edu/~jurafsky/slp3/>

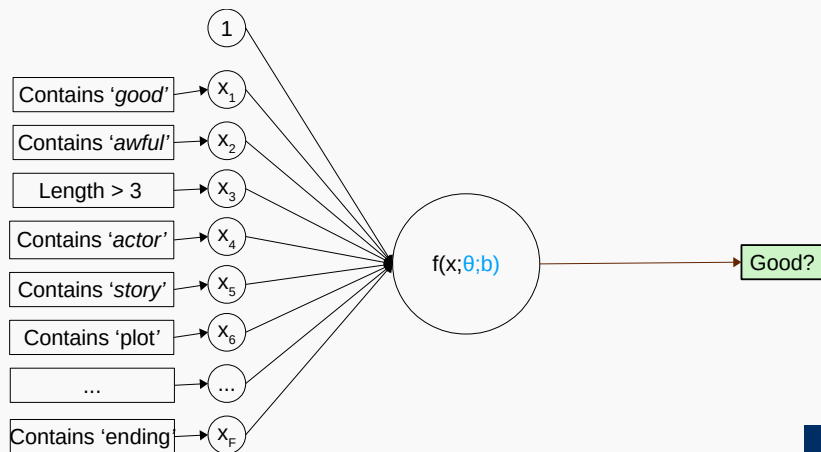


Another Example Problem: Sentiment analysis of movie reviews



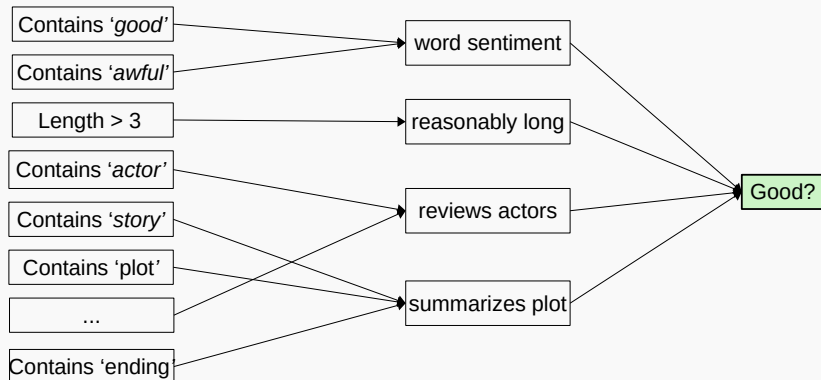
Multilayer Perceptron: Motivation II

Another Example Problem: Sentiment analysis of movie reviews

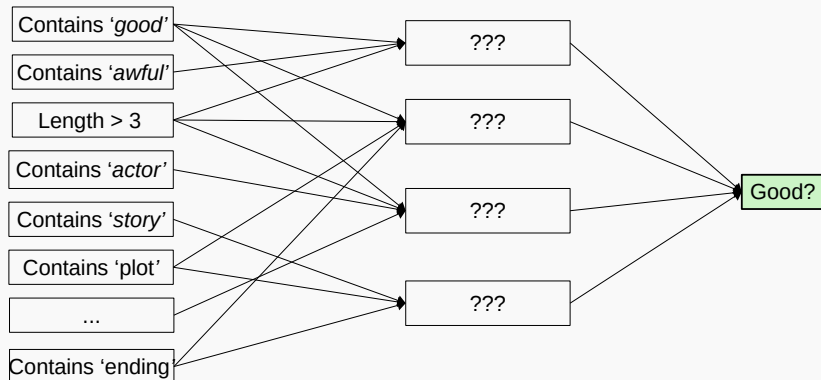


Input layer, 1 unit, output layer

Another Example Problem: Sentiment analysis of movie reviews

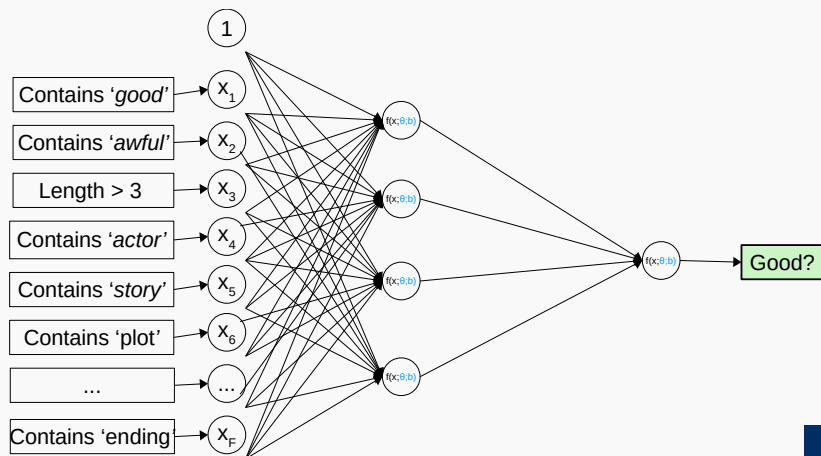


Another Example Problem: Sentiment analysis of movie reviews



Multilayer Perceptron: Motivation II

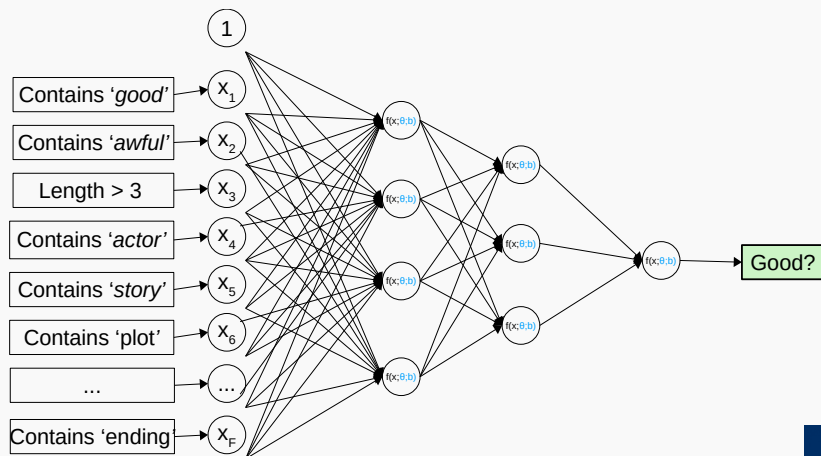
Another Example Problem: Sentiment analysis of movie reviews



Input layer, 1 hidden layer, output layer

Multilayer Perceptron: Motivation II

Another Example Problem: Sentiment analysis of movie reviews



Input layer, 2 hidden layer, output layer