Lecture 11: Neural Networks

COMP90049
Introduction to Machine Learning

Semester 2, 2020

Hadi Khorshidi, CIS

The presentation adapted from the slides prepared by Lea Frermann, CIS

Roadmap

So far ... Classification and Evaluation

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

Roadmap

So far ... Classification and Evaluation

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

Today... Neural Networks

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

Introduction

Classifier Recap

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
- Activation: step-function (discrete, non-differentiable)

Classifier Recap

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
- Activation: step-function (discrete, non-differentiable)

Logistic Regression

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

- View 1: Model of $P(y = 1|x)$, maximizing the data log likelihood
- View 2: Single processing ‘unit’
- Activation: sigmoid (continuous, differentiable)

Neural Networks and Deep Learning

Neural Networks

- Connected sets of many such units
- Units must have continuous activation functions
- Connected into many layers $\rightarrow$ **Deep** Learning

Multi-layer Perceptron

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

Neural Networks and Deep Learning

Neural Networks

- Connected sets of many such units
- Units must have continuous activation functions
- Connected into many layers $\rightarrow$ **Deep** Learning

Multi-layer Perceptron

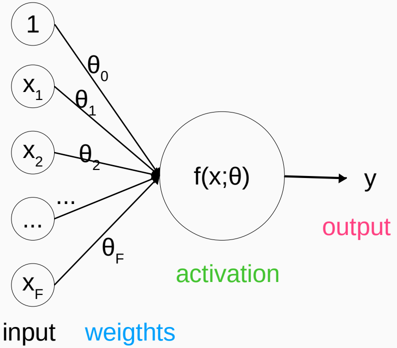
- One specific type of neural network
- Feed-forward
- Fully connected
- Supervised learner

Other types of neural networks

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

Perceptron Unit (recap)

A single processing unit



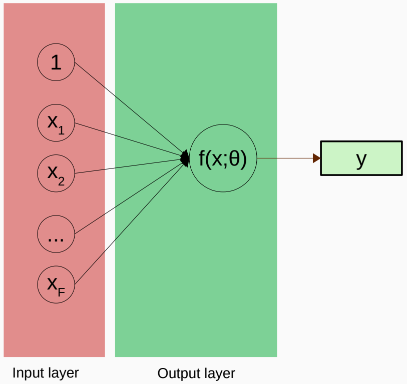
$y = f(\theta^T x)$

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

Multi-layer Perceptron (schematic)

Three Types of layers

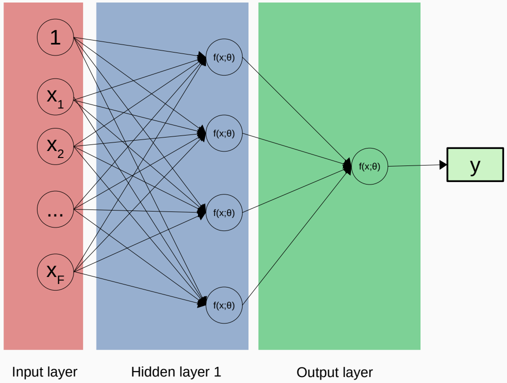
- **Input layer** with input units x: the first layer, takes features x as inputs
- **Output layer** with output units y: the last layer, has one unit per possible output (e.g., 1 unit for binary classification)
- **Hidden layers** with hidden units h: all layers in between.



Multi-layer Perceptron (schematic)

Three Types of layers

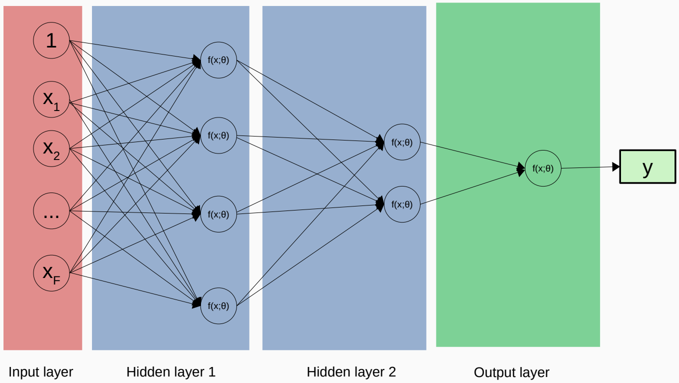
- **Input layer** with input units x: the first layer, takes features x as inputs
- **Output layer** with output units y: the last layer, has one unit per possible output (e.g., 1 unit for binary classification)
- **Hidden layers** with hidden units h: all layers in between.



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.



Why the Hype? I

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
- Perceptron can solve only linearly separable problems

Non-linear classification

- Neural networks with at least 1 hidden layer and non-linear activations are non-linear classifiers
- Decision boundary is a non-linear function of the inputs
- Capture ‘feature interactions’

Why the Hype? II

Feature Engineering

- (more next week!)
- The perceptron, naive Bayes and logistic regression require a fixed set of informative **features**
- e.g., $\texttt{outlook} \in \{\textit{overcast}, \textit{sunny}, \textit{rainy}\}$, $\texttt{wind} \in \{\textit{high}, \textit{low}\}$ etc
- Requiring **domain knowledge**

Feature learning

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

Multilayer Perceptron: Motivation I

Example Classification dataset

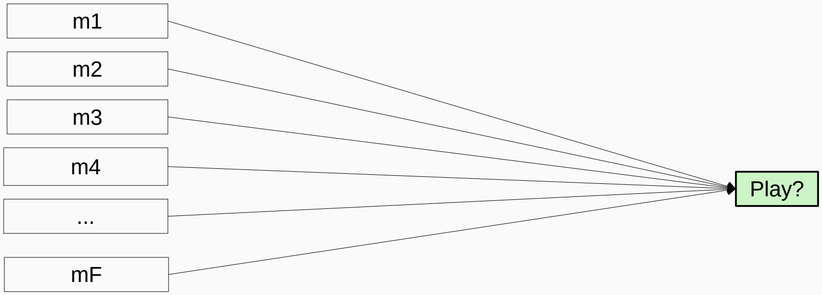
Outlook	Temperature	Humidity	Windy	True Label
sunny	hot	high	FALSE	no
sunny	hot	high	TRUE	no
overcast	hot	high	FALSE	yes
rainy	mild	high	FALSE	yes
...				

We really observe raw data

Date	measurements					True Label
01/03/1966	0.4	4.7	1.5	12.7	...	no
01/04/1966	3.4	-0.7	3.8	18.7	...	no
01/05/1966	0.3	8.7	136.9	17	...	yes
01/06/1966	5.5	5.7	65.5	2.7	...	yes

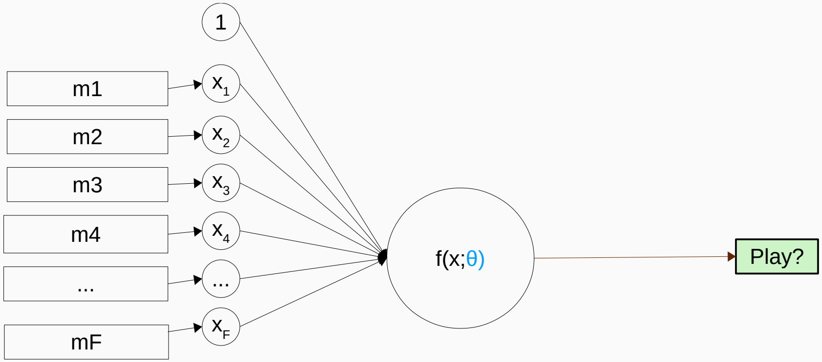
Multilayer Perceptron: Motivation II

Example Problem: Weather Dataset



Multilayer Perceptron: Motivation II

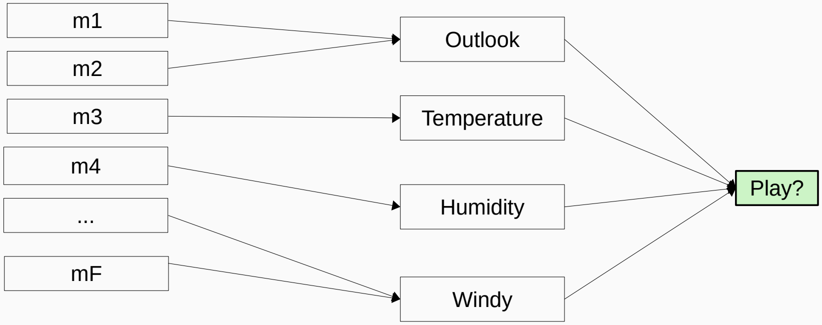
Example Problem: Weather Dataset



Input layer, 1 unit, output layer

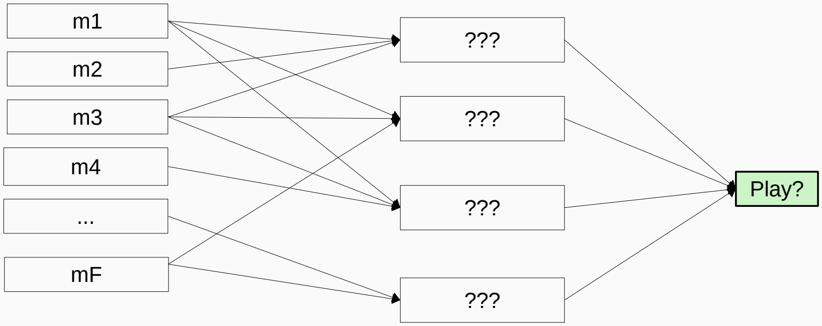
Multilayer Perceptron: Motivation II

Example Problem: Weather Dataset



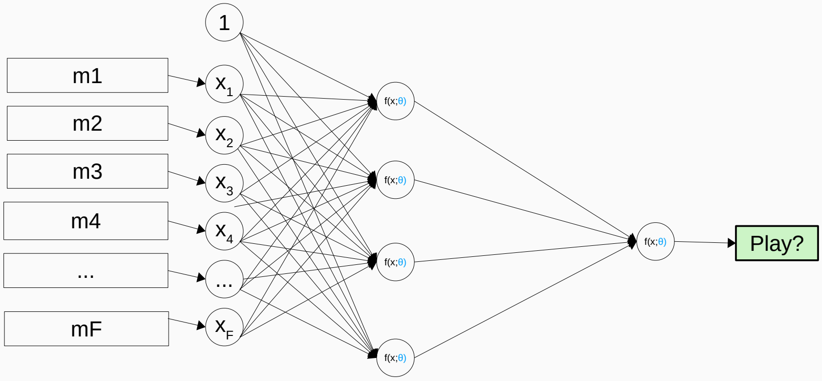
Multilayer Perceptron: Motivation II

Example Problem: Weather Dataset



Multilayer Perceptron: Motivation II

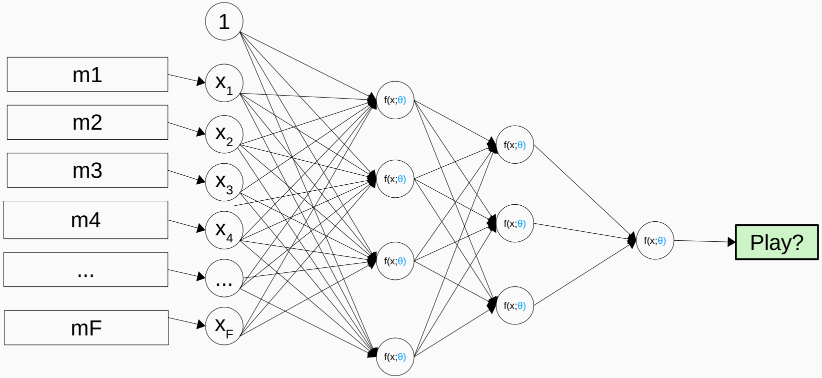
Example Problem: Weather Dataset



Input layer, 1 hidden layer, output layer

Multilayer Perceptron: Motivation II

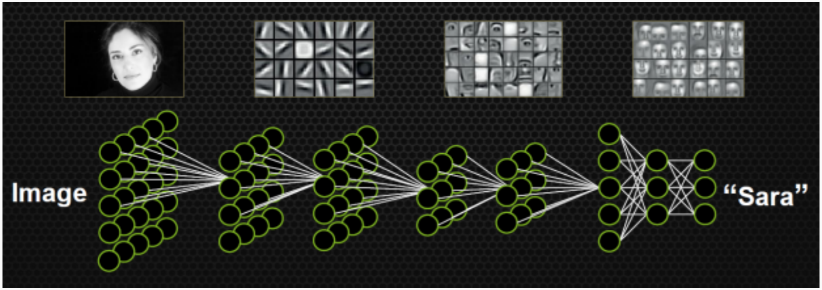
Example Problem: Weather Dataset



Input layer, 2 hidden layer, output layer

Another Example: Face Recognition

- 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/

Multilayer Perceptron

Terminology

- input units x_j, one per feature j
- Multiple **layers** $l = 1...L$ 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 $\phi^{(l)}$

Prediction with a feedforward Network

Passing an input through a neural network with 2 hidden layers

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

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

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

Prediction with a feedforward Network

Passing an input through a neural network with 2 hidden layers

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

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

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

Or in vectorized form

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

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

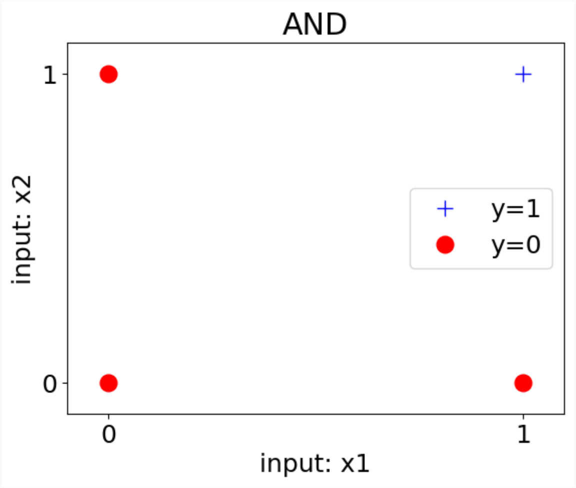
$$y = \phi^{(3)}\Big(\theta^{(3)T} h^{(2)}\Big)$$

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

Boolean Functions

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

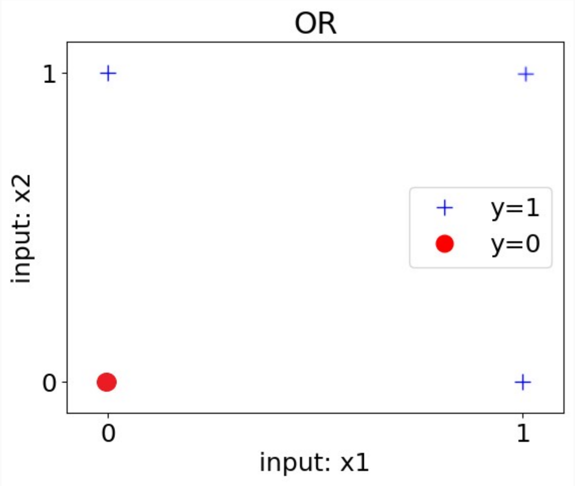
x_1	x_2	y
1	1	1
1	0	0
0	1	0
0	0	0



Boolean Functions

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

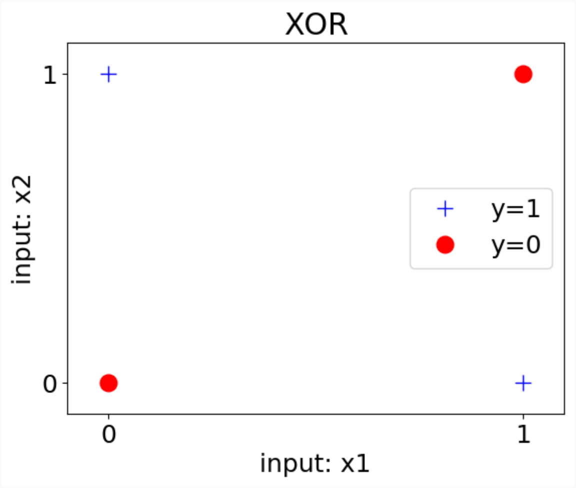
x_1	x_2	y
1	1	1
1	0	1
0	1	1
0	0	0



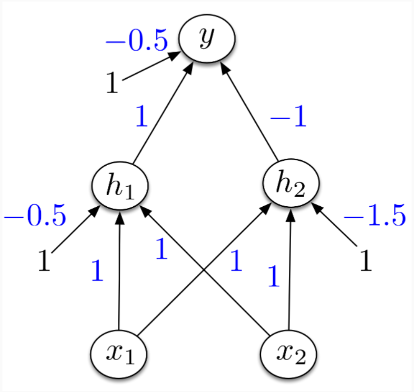
Boolean Functions

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



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

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

Designing Neural Networks I: Inputs

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

Designing Neural Networks II: Activation Functions

- Each layer has an associated activation function (e.g., sigmode, RelU, ...)
- Represents the extent to which a neuron is ‘activated’ given an input
- Each hidden layer performs a **non-linear transformation** of the input
- Popular choices include

Designing Neural Networks II: Activation Functions

- Each layer has an associated activation function (e.g., sigmode, RelU, ...)
- Represents the extent to which a neuron is ‘activated’ given an input
- Each hidden layer performs a **non-linear transformation** of the input
- Popular choices include

1. logistic (aka sigmoid) (“σ”):

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

2. hyperbolic tan (“tanh”):

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

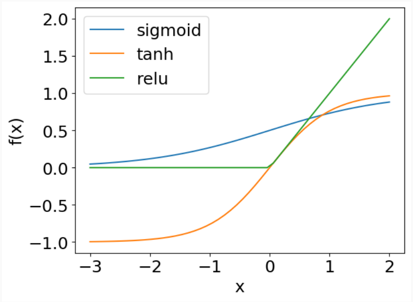
3. rectified linear unit (“ReLU”):

$$f(x) = \max(0, x)$$

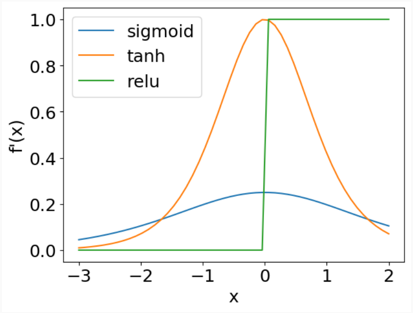
note not differentiable at $x = 0$

Designing Neural Networks II: Activation Functions

function values:



derivatives:



Designing Neural Networks III: Structure

Network Structure

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

Advanced tricks include allowing for exploiting data structure

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

Beyond the scope of this class.

Designing Neural Networks IV: Output Function

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

- 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

Designing Neural Networks V: Loss Functions

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

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

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

- 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.

Designing Neural Networks V: Loss Functions

Regression Loss: typically mean-squared error (MSE)

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

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

Representational Power of Neural Nets

- 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

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

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 traning set is small
- If *interpretability* is important, i.e., understanding how (combinations of) features explain different predictions

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

Summary

Today

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

Next Lecture

- Learning parameters of neural networks
- The Backpropagation algorithm

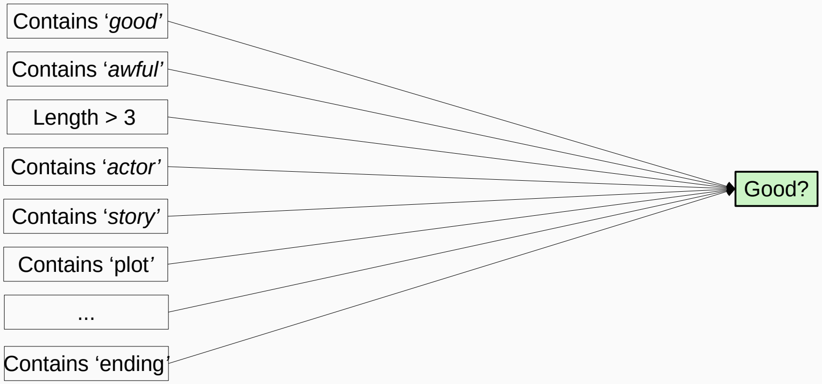
References

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/

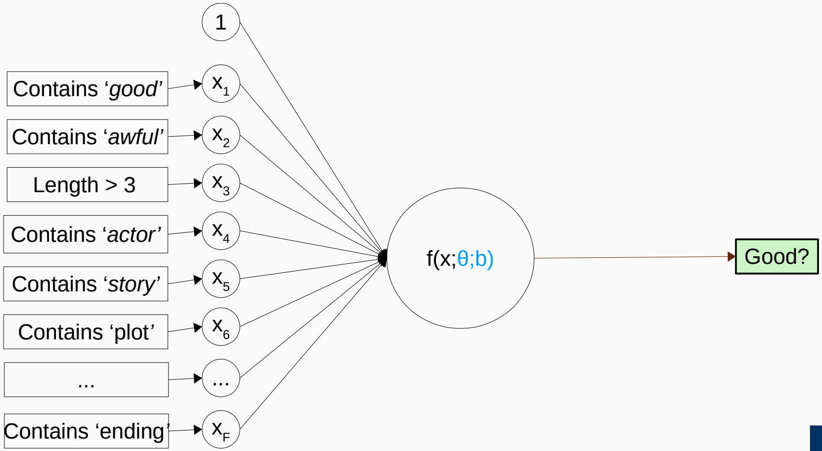
Multilayer Perceptron: Motivation II

Another Example Problem: Sentiment analysis of movie reviews



Multilayer Perceptron: Motivation II

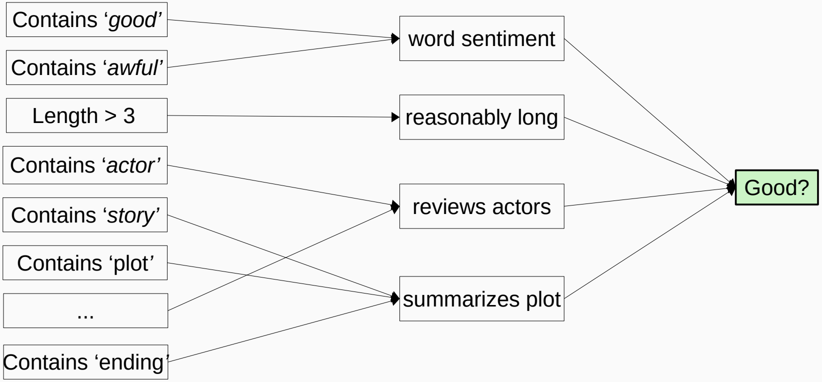
Another Example Problem: Sentiment analysis of movie reviews



Input layer, 1 unit, output layer

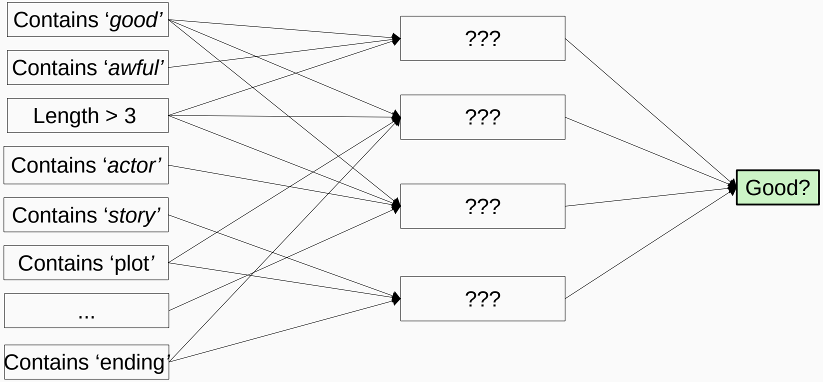
Multilayer Perceptron: Motivation II

Another Example Problem: Sentiment analysis of movie reviews



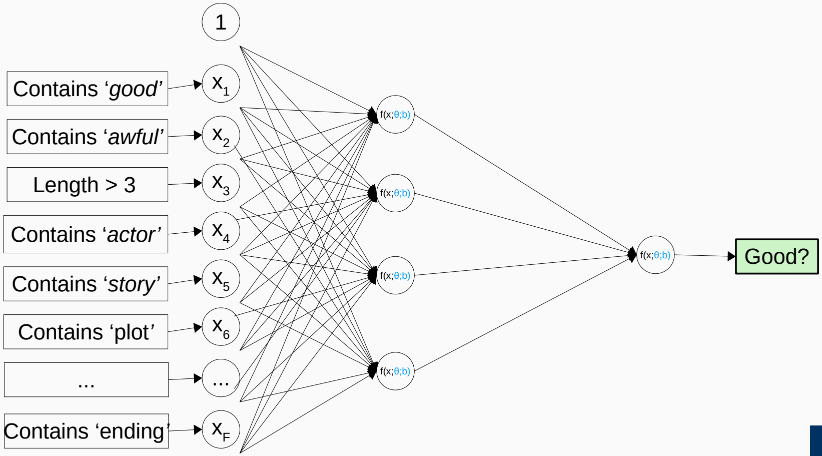
Multilayer Perceptron: Motivation II

Another Example Problem: Sentiment analysis of movie reviews



Multilayer Perceptron: Motivation II

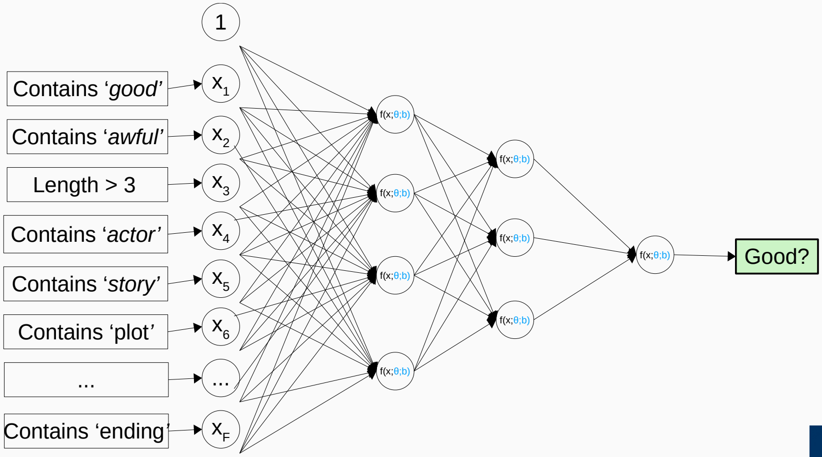
Another Example Problem: Sentiment analysis of movie reviews



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