Lecture 1: Introduction

André Martins, Francisco Melo, Mário Figueiredo

Deep Learning Course, Winter 2022-2023

Deep Learning Course

- A new MSc-level course
- Offered jointly by DEEC and DEI
- MSc programs: MEEC, MECD, MEIC-A, MEIC-T
- 350 students enrolled this year! (192 DEEC, 158 DEI).

# Course Website(s)

There we can find:

- Syllabus

- Lecture slides

- Literature pointers

- Practical and homework assignments

- Announcements

- ...

Instructors

- **Main instructors:** André Martins (DEI Alameda), Francisco Melo (DEI Tagus), Mário Figueiredo (DEEC)
- **Practical classes:** Andreas Wichert, Ben Peters, Chrysoula Zerva, Gonçalo Correia, João Fonseca, João Santinha, José Moreira, Margarida Campos, Tomás Costa
- **Location & schedule:** see course webpage in Fenix (previous slide)
- **Office hours:** see information in Fenix
- **Communication:**
  piazza.com/tecnico.ulisboa.pt/fall2022/c88

Please register in Piazza!!

What is "'Deep Learning"?
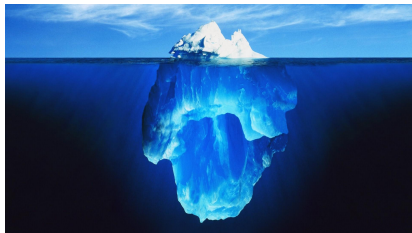
- Neural networks?
- Neural networks with many hidden layers?
- Anything beyond shallow (linear) models for machine learning?
- Anything that learns representations?
- A form of learning that is really intense and profound?

Why Did Deep Learning Become Mainstream?

Lots of recent breakthroughs:

- Object recognition
- Speech and language processing (Transformers, BERT, GPT-3)
- Machine translation
- Chatbots and dialog systems
- Self-driving cars
- Solving games (Atari, Go, StartCraft II)
- Protein design (AlphaFold)

No signs of slowing down...

Microsoft's Deep Learning Project Outperforms Humans In Image Recognition

Michael Thomsen, CONTRIBUTOR

I write about tech, video games, science and culture. FULL BIO ∨

Opinions expressed by Forbes Contributors are their own.

Big Data on AWS

Microsoft's new breakthrough: AI that's as good as humans at listening... on the phone

Microsoft's new speech-recognition record means professional transcribers could be among the first to lose their jobs to artificial intelligence.

By Liam Tung | October 19, 2016 -- 10:10 GMT (11:10 BST) | Topic: Innovation

Who is wearing glasses?
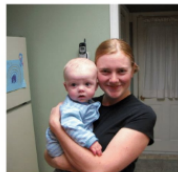man   woman
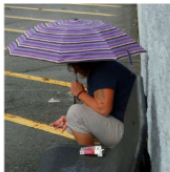
Where is the child sitting?
fridge   arms

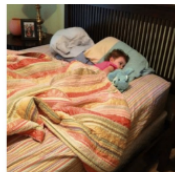Is the umbrella upside down?
yes   no

How many children are in the bed?
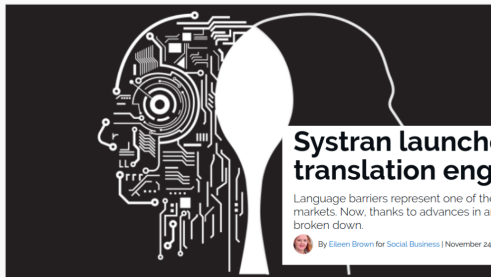2   1

The Great A.I. Awakening

How Google used artificial intelligence to transform Google Translate, one of its more popular services — and how machine learning is poised to reinvent computing itself.

BY GIDEON LEWIS-KRAUS   DEC. 14, 2016

Google unleashes deep learning tech on language with Neural Machine Translation

Posted Sep 27, 2016 by Devin Coldewey, Contributor

Systran launches neural machine translation engine in 30 languages

Language barriers represent one of the biggest challenges to develop business strategies among global markets. Now, thanks to advances in artificial intelligence and machine translation, these barriers are being broken down.

By Eileen Brown for Social Business | November 24, 2016 -- 13:49 GMT (13:49 GMT) | Topic: Artificial Intelligence

AlphaGo Beats Go Human Champ: Godfather Of Deep Learning Tells Us Do Not Be Afraid Of AI

21 March 2016, 10:16 am EDT **By Aaron Mamiit Tech Times**

Last week, Google's artificial intelligence program

Last week, Google's artificial intelligence program AlphaGo dominated its match with South Korean world Go champion Lee Sedol, winning with a 4-1 score.

The achievement stunned artificial intelligence experts, who previously thought that Google's computer program would need at least 10 more years before developing enough to be able to beat a human world champion.

A robot wrote this entire article. Are you scared yet, human?

We asked GPT-3, OpenAI's powerful new language generator, to write an essay for us from scratch. The assignment? To convince us robots come in peace
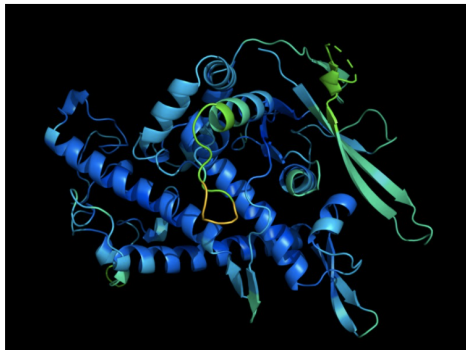
- For more about GPT-3 and how this essay was written and edited, please read our editor's note below

'It will change everything': DeepMind's AI makes gigantic leap in solving protein structures

Google's deep-learning program for determining the 3D shapes of proteins stands to transform biology, say scientists.
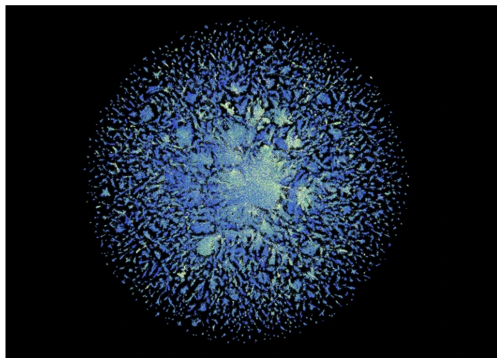
Ewen Callaway

A protein's function is determined by its 3D shape. Credit: DeepMind

AlphaFold's new rival? Meta AI predicts shape of 600 million proteins

Microbial molecules from soil, seawater and human bodies are among the planet's least understood.

Ewen Callaway

The ESM Metagenomic Atlas contains structural predictions for 617 million proteins. Credit: ESM Metagenomic Atlas (CC BY 4.0)

How to use DALL•E 2 to turn your wildest imaginations into tangible art

AI art platform, DALL•E 2, creates images from text descriptions in seconds. In this article, we show you how to get the results you desire.

Written by **Christina Darby**, Associate Editor on Oct. 31, 2022

Why Now?

Why does deep learning work now, but not 30 years ago?

Many of the core ideas were there, after all.

But now we have:

- more data
- more computing power
- (much) better software engineering (e.g. auto-diff)
- some algorithmic innovations: many layers, ReLUs, better learning algorithms, dropout, CNNs, LSTMs, transformers, etc.
  All of these will be covered in this course.

"But It's Non-Convex"

For many years (1990–2010), NNs were not popular in machine learning, because they were hard to learn.

Why does gradient-based optimization work at all in NNs despite the non-convexity?

One possible, partial answer (this is an open research topic)

- there are generally many hidden units
- there are many ways a neural net can approximate the desired input-output relationship
- we only need to find one

One turning point: AlexNet

- Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton; 2012

- ImageNet: Large Scale Visual Recognition Challenge (14 million images, 20000 categories)
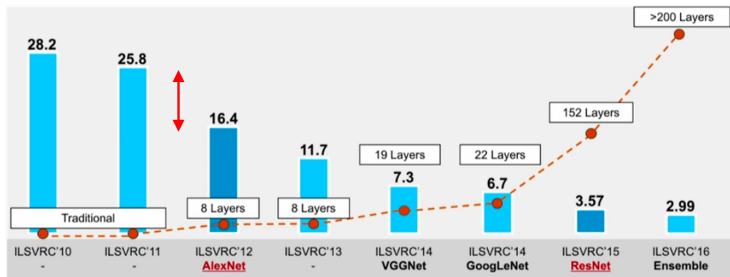
- Large CNN, much deeper than anything else at the time

- Used parallel processing (one of the first uses of GPUs in NNs)

- Convinced many people that deep learning would change the field.

Recommended Books

Main book:

- **Deep Learning.** Ian Goodfellow, Yoshua Bengio, and Aaron Courville. MIT Press, 2016. Chapters available at `http://deeplearningbook.org`
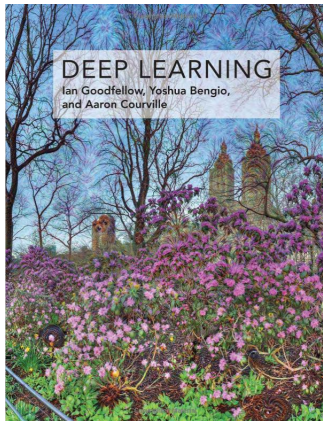
Recommended Books

Secondary books:

- **Artificial Intelligence Engines: A Tutorial Introduction to the Mathematics of Deep Learning.** James Stone. Sebtel Press, 2019.
- **Dive into Deep Learning.** Aston Zhang, Zach Lipton, Mu Li, Alex Smola (https://d2l.ai/)
- **Deep Learning with Python.** François Chollet. Manning Publications, 2017.
- **Machine Learning – A Journey to Deep Learning with Exercises and Answers.** Andreas Wichert and Luis Sa-Couto, 2021
- **Probabilistic Machine Learning.** Kevin P. Murphy (https://probml.github.io/pml-book/)

Tentative Syllabus

Week 1   Introduction and Course Description
         Linear Classifiers I (linear regression, perceptron)
Week 2   Linear Classifiers II (logistic regression, regularization)
Week 3   Neural Networks I
Week 4   Neural Networks II
         Representation Learning and Auto-Encoders
Week 5   Convolutional Networks
         Recurrent Neural Networks and LSTMs
Week 6   Sequence-to-Sequence Models and Attention Mechanisms
         Transformers
Week 7   Self-Supervised Learning (BERT, GPT3, etc.)
         Interpretability and Fairness

What This Class Is About

- Introduction to deep learning (DL)

- **Goal:** after finishing this class, you should be able to:

  - ✓ Understand how DL works (it's not magic)

  - ✓ Understand the math and intuition behind DL models

  - ✓ Apply DL on practical problems (language, vision, ...)

- **Target audience:**

  - ✓ MSc students with basic background in probability theory, linear algebra, programming.

  - ✓ Useful: background in machine learning.

What This Class Is Not About

- Just playing with DL toolkits, without learning the fundamental concepts

- Introduction to machine learning (other courses by DEEC and DEI)

- Natural language processing (another course offered by DEI)

- Computer vision (another course offered by DEEC)

- Optimization (other courses by DEEC and DEI)

- ...

Prerequisites

- Calculus and basic linear algebra

- Basic probability theory

- Basic knowledge of machine learning (preferred, but not required)

- Programming (Python & PyTorch, preferred but not required)

Course Information

- **Main instructors:** André Martins (DEI Alameda), Francisco Melo (DEI Tagus), Mário Figueiredo (DEEC)
- **Practical classes:** Andreas Wichert, Ben Peters, Chrysoula Zerva, Gonçalo Correia, João Fonseca, João Santinha, José Moreira, Margarida Campos, Tomás Costa
- **Location & schedule:** see course webpage in Fenix (previous slide)
- **Office hours:** see information in Fenix
- **Communication:**
  piazza.com/tecnico.ulisboa.pt/fall2022/c88

Please register in Piazza!!!

Schedule (DEI Alameda/Tagus)

- Letures: 2 per week (Alameda: Tue & Thur; Tagus: Mon & Thur)
- Practical shifts (see Fenix) – pick your slots and register as a group! (more later)

Schedule (DEEC Alameda)

- Lectures: 2 per week (shift 1: Mon & Thur; shift 2: Tue & Thur)
- Practical shifts (see Fenix): pick your slots and register as a group! (more later!)

Grading

- 2 homework assignments: 50%
  - Minimal grade: 8
  - Theoretical questions & implementation
  - Groups of **2** – you need to register your group in Fenix!
  - Some of the practicals will be Q&A about these assignments, so please join the practicals as a group
  - Submission through Fenix
  - No late days allowed

- Final exam: 50%
  - Minimal grade: 8

Registering Your Group in Fenix

- Pick a group of **2**

- Register in Fenix

- Deadline: Sunday, November 27, at 23:59

- Use Piazza to find group mates

- Can't find a group?
  Tell the instructors by November 28 (in Piazza); we'll find a solution.

Collaboration Policy

- Assignments should be done within each group

- Students may discuss the questions across groups, as long as they write their own answers and their own code

- If this happens, acknowledge with whom you collaborate!

- Zero tolerance on plagiarism!!

- Always credit your sources!!!

Questions?
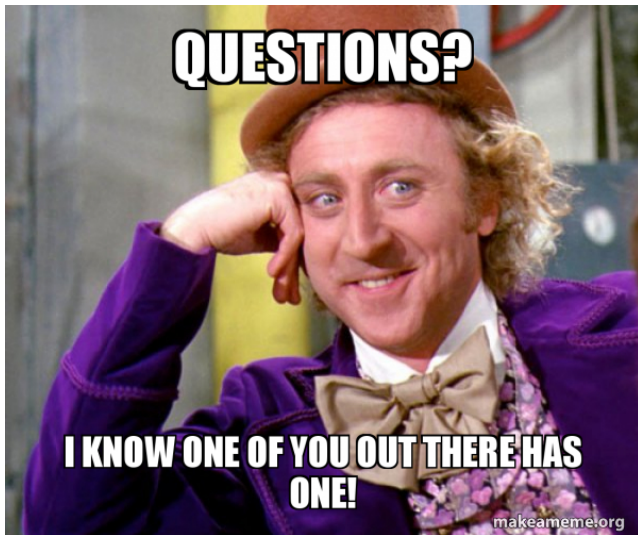
Today's Roadmap

- Linear Algebra (only skim over)

- Probability Refresher (only skim over)

- Optimization

- Introduction to Machine Learning
  - "Deep Learning Superheroes"
  - Supervised, Unsupervised, Reinforcement Learning
  - Classification and Regression

Notation: Matrices and Vectors

- $A \in \mathbb{R}^{m \times n}$ is a matrix with m rows and n columns.

$$A = \left[\begin{array}{ccc} A_{1,1} & \cdots & A_{1,n} \\ \vdots & \ddots & \vdots \\ A_{m,1} & \cdots & A_{m,n} \end{array} \right].$$

- $x \in \mathbb{R}^n$ is a vector with n components,

$$x = \left[\begin{array}{c} x_1 \\ \vdots \\ x_n \end{array} \right].$$

- A (column) vector is a matrix with n rows and 1 column.

- A matrix with 1 row and n columns is called a row vector.

Matrix Transpose and Matrix Products

- Given matrix $A \in \mathbb{R}^{m \times n}$, its transpose A^T is such that $(A^T)_{i,j} = A_{j,i}$.

- Matrix A is symmetric if $A^T = A$.

- Given matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$, their product is

$$C = A\,B \in \mathbb{R}^{m \times p} \quad \text{where} \quad C_{i,j} = \sum_{k=1}^{n} A_{i,k}\, B_{k,j}$$

- Inner product between vectors $x, y \in \mathbb{R}^n$:

$$\langle x, y \rangle = x^T y = y^T x = \sum_{i=1}^{n} x_i y_i$$

$$\left[\begin{array}{c} x_1 \\ \vdots \\ x_n \end{array} \right]^T \left[\begin{array}{c} y_1 \\ \vdots \\ y_n \end{array} \right] = [x_1, ..., x_n] \left[\begin{array}{c} y_1 \\ \vdots \\ y_n \end{array} \right] = \sum_{i=1}^{n} x_i y_i$$

Outer and Hadamard Products

- **Outer product** between vectors $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$:

$$x\, y^T \in \mathbb{R}^{n \times m}, \quad \text{where } (x\, y^T)_{i,j} = x_i\, y_j$$

$$x\, y^T = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} [y_1, ..., y_m] = \begin{bmatrix} x_1 y_1 & \cdots & x_1 y_m \\ \vdots & \ddots & \vdots \\ x_n y_1 & \cdots & x_n y_m \end{bmatrix}$$

- **Hadamard/Schur product** between vectors $x, y \in \mathbb{R}^n$: $(x \odot y)_i = x_i y_i$,

$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \odot \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1\, y_1 \\ \vdots \\ x_n\, y_n \end{bmatrix}$$

Properties of Matrix Products and Transposes

- Given matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$, their product is

$$C = A B \in \mathbb{R}^{m \times p} \quad \text{where} \quad C_{i,j} = \sum_{k=1}^{n} A_{i,k} B_{k,j}$$

- Matrix product is associative: $(AB)C = A(BC)$.

- In general, matrix product is not commutative: $AB \neq BA$.

- Transpose of product: $(A B)^T = B^T A^T$.

- Transpose of sum: $(A + B)^T = A^T + B^T$.

Norms

- The norm of a vector is (informally) its "magnitude." Euclidean norm:

$$\|x\|_2 = \sqrt{\langle x, x \rangle} = \sqrt{x^T x} = \sqrt{\sum_{i=1}^{n} x_i^2}.$$

- More generally, the ℓ_p norm of a vector $x \in \mathbb{R}^n$, where $p \geq 1$,

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}.$$

- Notable case: the ℓ_1 norm, $\|x\|_1 = \sum_i |x_i|$.

- Notable case: the ℓ_∞ norm, $\|x\|_\infty = \max\{|x_1|, ..., |x_n|\}$.

Special Matrices

- The identity matrix $I \in \mathbb{R}^{n \times n}$ is a square matrix such that

$$I_{ij} = \left\{ \begin{array}{ll} 1 & i = j \\ 0 & i \neq j \end{array} \right. \qquad I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

- Neutral element of matrix product: $A\,I = I\,A = A$.

- Diagonal matrix: $A \in \mathbb{R}^{n \times n}$ is diagonal if $(i \neq j) \Rightarrow A_{i,j} = 0$.

- Upper triangular matrix: $(j < i) \Rightarrow A_{i,j} = 0$.

- Lower triangular matrix: $(j > i) \Rightarrow A_{i,j} = 0$.

Eigenvalues, eigenvectors, determinant, trace

- A vector $x \in \mathbb{R}^n$ is an eigenvector of matrix $A \in \mathbb{R}^{n \times n}$ if

$$A x = \lambda x,$$

  where $\lambda \in \mathbb{R}$ is the corresponding eigenvalue.

- The eigenvalues of a diagonal matrix are the elements in the diagonal.

- Matrix trace:

$$\text{trace}(A) = \sum_i A_{i,i} = \sum_i \lambda_i$$

- Matrix determinant:

$$|A| = \det(A) = \prod_i \lambda_i$$

- Properties: $|AB| = |A||B|, \quad |A^T| = |A|, \quad |\alpha A| = \alpha^n |A|$

Matrix Inverse

- Matrix $A \in \mathbb{R}^{n \times n}$ in invertible if there is $B \in \mathbb{R}^{n \times n}$ s.t. $AB = BA = I$.

- ...matrix B, such that $AB = BA = I$, denoted $B = A^{-1}$.

- Matrix $A \in \mathbb{R}^{n \times n}$ is invertible $\Leftrightarrow \det(A) \neq 0$.

- Determinant of inverse: $\det(A^{-1}) = \dfrac{1}{\det(A)}$.

- Solving system $Ax = b$, if A is invertible: $x = A^{-1}b$.

- Properties: $(A^{-1})^{-1} = A, \quad (A^{-1})^T = (A^T)^{-1}, \quad (AB)^{-1} = B^{-1}A^{-1}$

- There are many algorithms to compute A^{-1}; general case, computational cost $O(n^3)$.
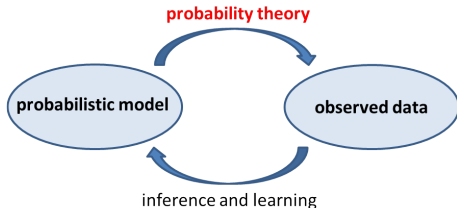
Quadratic Forms and Positive (Semi-)Definite Matrices

- Given matrix $A \in \mathbb{R}^{n \times n}$ and vector $x \in \mathbb{R}^n$,

$$x^T A x = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{i,j} \, x_i \, x_j \; \in \; \mathbb{R}$$

  is called a quadratic form.

- A symmetric matrix $A \in \mathbb{R}^{n \times n}$ is positive semi-definite (PSD) if, for any $x \in \mathbb{R}^n$, $x^T A x \geq 0$.

- A symmetric matrix $A \in \mathbb{R}^{n \times n}$ is positive definite (PD) if, for any $x \in \mathbb{R}^n$, $(x \neq 0) \Rightarrow x^T A x > 0$.

- Matrix $A \in \mathbb{R}^{n \times n}$ is PSD $\Leftrightarrow$ all $\lambda_i(A) \geq 0$.

- Matrix $A \in \mathbb{R}^{n \times n}$ is PD $\Leftrightarrow$ all $\lambda_i(A) > 0$.

Probability theory

- "Essentially, all models are wrong, but some are useful"; G. Box, 1987

- The study of probability has roots in games of chance (dice, cards, ...)

- Natural tool to model uncertainty, information, knowledge, belief, ...

- ...thus also learning, inference, ...

What is probability?

- Classical definition: $\mathbb{P}(A) = \dfrac{N_A}{N}$

  ...with N mutually exclusive equally likely outcomes,
  N_A of which result in the occurrence of event A.   *Laplace, 1814*

  Example: $\mathbb{P}(\text{randomly drawn card is} \clubsuit) = 13/52$.

  Example: $\mathbb{P}(\text{getting 1 in throwing a fair die}) = 1/6$.

- Frequentist definition: $\mathbb{P}(A) = \lim\limits_{N \to \infty} \dfrac{N_A}{N}$

  ...relative frequency of occurrence of A in infinite number of trials.

- Subjective probability: $\mathbb{P}(A)$ is a degree of belief.   *de Finetti, 1930s*

  ...gives meaning to $\mathbb{P}(\text{"Tomorrow it will rain"})$.

Key concepts: Sample space and events

- Sample space $\mathcal{X} =$ set of possible outcomes of a random experiment.

  Examples:
  - Tossing two coins: $\mathcal{X} = \{HH, TH, HT, TT\}$
  - Roulette: $\mathcal{X} = \{1, 2, ..., 36\}$
  - Draw a card from a shuffled deck: $\mathcal{X} = \{A\clubsuit, 2\clubsuit, ..., Q\diamondsuit, K\diamondsuit\}$.

- An event A is a subset of $\mathcal{X}$: $A \subseteq \mathcal{X}$.

  Examples:
  - "exactly one H in 2-coin toss": $A = \{TH, HT\} \subset \{HH, TH, HT, TT\}$.
  - "odd number in the roulette": $B = \{1, 3, ..., 35\} \subset \{1, 2, ..., 36\}$.
  - "drawn a $\heartsuit$ card": $C = \{A\heartsuit, 2\heartsuit, ..., K\heartsuit\} \subset \{A\clubsuit, ..., K\diamondsuit\}$
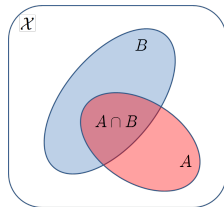
Kolmogorov's Axioms for Probability

- Probability is a function that maps events A into the interval $[0, 1]$.

  Kolmogorov's axioms (1933) for probability $\mathbb{P}$

  - For any A, $\mathbb{P}(A) \geq 0$
  - $\mathbb{P}(\mathcal{X}) = 1$
  - If $A_1, A_2 \ldots \subseteq \mathcal{X}$ are disjoint events, then $\mathbb{P}\left(\bigcup_i A_i\right) = \sum_i \mathbb{P}(A_i)$

- From these axioms, many results can be derived. Examples:

- $\mathbb{P}(\emptyset) = 0$
- $C \subset D \Rightarrow \mathbb{P}(C) \leq \mathbb{P}(D)$
- $\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cap B)$

Conditional Probability and Independence

- If $\mathbb{P}(B) > 0$, $\qquad \mathbb{P}(A|B) = \dfrac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$

- Events A, B are independent $(A \perp\!\!\!\perp B) \Leftrightarrow \mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B)$.

- Relationship with conditional probabilities:

$$A \perp\!\!\!\perp B \Leftrightarrow \mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)} = \frac{\mathbb{P}(A)\,\mathbb{P}(B)}{\mathbb{P}(B)} = \mathbb{P}(A)$$

- Example: $\mathcal{X} = $ "52 cards", $A = \{3\heartsuit, 3\clubsuit, 3\diamondsuit, 3\spadesuit\}$, and
  $B = \{A\heartsuit, 2\heartsuit, ..., K\heartsuit\}$; then, $\mathbb{P}(A) = 1/13$, $\mathbb{P}(B) = 1/4$

$$
\begin{aligned}
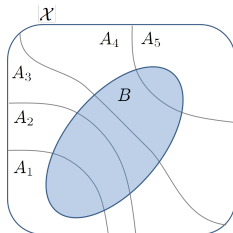\mathbb{P}(A \cap B) &= \mathbb{P}(\{3\heartsuit\}) = \frac{1}{52} \\
\mathbb{P}(A)\,\mathbb{P}(B) &= \frac{1}{13}\frac{1}{4} = \frac{1}{52} \\
\mathbb{P}(A|B) &= \mathbb{P}(\text{"3"}|\text{"}\heartsuit\text{"}) = \frac{1}{13} = \mathbb{P}(A)
\end{aligned}
$$

Law of Total Probability and Bayes Theorem

- Law of total probability: if $A_1, ..., A_n$ are a partition of $\mathcal{X}$

$$
\begin{aligned}
\mathbb{P}(B) &= \sum_i \mathbb{P}(B|A_i)\mathbb{P}(A_i) \\
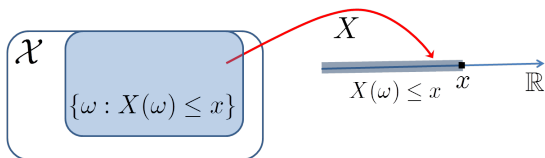&= \sum_i \mathbb{P}(B \cap A_i)
\end{aligned}
$$

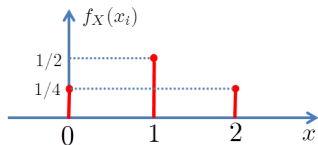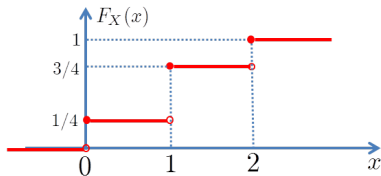- Bayes' theorem: if $\{A_1, ..., A_n\}$ is a partition of $\mathcal{X}$

$$
\mathbb{P}(A_i|B) = \frac{\mathbb{P}(B \cap A_i)}{\mathbb{P}(B)}
$$

Discrete Random Variables

- Distribution function: $F_X(x) = \mathbb{P}(\{\omega \in \mathcal{X} : X(\omega) \leq x\})$

- Example: number of heads in tossing 2 coins; range$(X) = \{0, 1, 2\}$.

- Probability mass function (discrete RV): $f_X(x) = \mathbb{P}(X = x)$,

$$F_X(x) = \sum_{x_i \leq x} f_X(x_i).$$

Important Discrete Random Variables

- **Uniform**: $X \in \{x_1, ..., x_K\}$, pmf $f_X(x_i) = 1/K$.

- **Bernoulli RV**: $X \in \{0, 1\}$, pmf $f_X(x) = \begin{cases} p & \Leftarrow & x = 1 \\ 1-p & \Leftarrow & x = 0 \end{cases}$
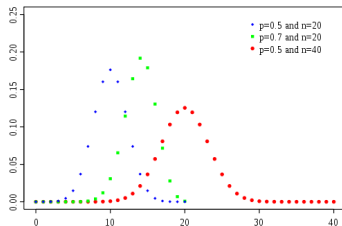
  Can be written compactly as $f_X(x) = p^x (1-p)^{1-x}$.

- **Binomial RV**: $X \in \{0, 1, ..., n\}$ (sum on n Bernoulli RVs)

  $$f_X(x) = \text{Binomial}(x; n, p) = \binom{n}{x} p^x \, (1-p)^{(n-x)}$$
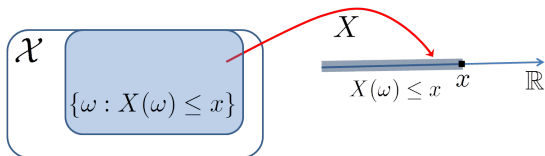
Binomial coefficients
("n choose x"):

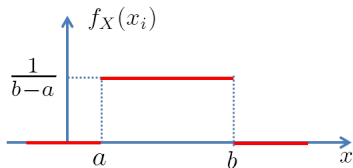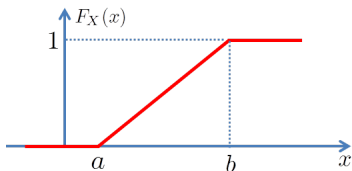$$\binom{n}{x} = \frac{n!}{(n-x)! \, x!}$$

Continuous Random Variables

- **Distribution function**: $F_X(x) = \mathbb{P}(\{\omega \in \mathcal{X} : X(\omega) \leq x\})$

- **Example**: continuous RV with uniform distribution on $[a, b]$.

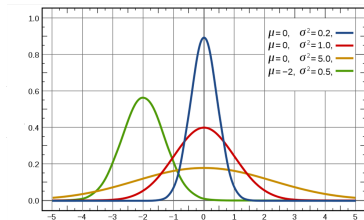- **Probability density function** (pdf, continuous RV): $f_X(x)$
$$F_X(x) = \int_{-\infty}^{x} f_X(u)\, du, \quad \mathbb{P}(X \in [c, d]) = \int_{c}^{d} f_X(x)\, dx, \quad \mathbb{P}(X = x) = 0$$

Important Continuous Random Variables

- **Uniform**: $f_X(x) = \text{Uniform}(x; a, b) = \left\{ \begin{array}{ccc} \frac{1}{b-a} & \Leftarrow & x \in [a, b] \\ 0 & \Leftarrow & x \notin [a, b] \end{array} \right.$

  (previous slide).

- **Gaussian**: $f_X(x) = \mathcal{N}(x; \mu, \sigma^2) = \dfrac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

Expectation of Random Variables

- Expectation: $\mathbb{E}(X) = \begin{cases} \displaystyle\sum_i x_i \, f_X(x_i) & X \in \{x_1, ..., x_K\} \subset \mathbb{R} \\ \displaystyle\int_{-\infty}^{\infty} x \, f_X(x) \, dx & X \text{ continuous} \end{cases}$

- Example: Bernoulli, $f_X(x) = p^x \, (1-p)^{1-x}$, for $x \in \{0, 1\}$.

$$\mathbb{E}(X) = 0 \, (1-p) + 1 \, p = p.$$

- Example: Gaussian, $f_X(x) = \mathcal{N}(x; \mu, \sigma^2)$. $\mathbb{E}(X) = \mu$.

- Linearity of expectation:

$$\mathbb{E}(X + Y) = \mathbb{E}(X) + \mathbb{E}(Y); \quad \mathbb{E}(\alpha \, X) = \alpha \mathbb{E}(X), \ \alpha \in \mathbb{R}$$

Expectation of Functions of Random Variables

- $\mathbb{E}(g(X)) = \begin{cases} \displaystyle\sum_i g(x_i) f_X(x_i) & X \text{ discrete, } g(x_i) \in \mathbb{R} \\ \displaystyle\int_{-\infty}^{\infty} g(x)\, f_X(x)\, dx & X \text{ continuous} \end{cases}$

- Example: variance, $\text{var}(X) = \mathbb{E}\Big(\big(X - \mathbb{E}(X)\big)^2 \Big) = \mathbb{E}(X^2) - \mathbb{E}(X)^2$

- Example: Bernoulli variance, $\mathbb{E}(X^2) = \mathbb{E}(X) = p$, thus $\text{var}(X) = p(1 - p)$.

- Example: Gaussian variance, $\mathbb{E}\big((X - \mu)^2\big) = \sigma^2$.

- Probability as expectation of indicator, $\mathbf{1}_A(x) = \begin{cases} 1 & \Leftarrow \ x \in A \\ 0 & \Leftarrow \ x \notin A \end{cases}$

$$\mathbb{P}(X \in A) = \int_A f_X(x)\, dx = \int \mathbf{1}_A(x)\, f_X(x)\, dx = \mathbb{E}(\mathbf{1}_A(X))$$

Two (or More) Random Variables

- **Joint pmf** of two discrete RVs: $f_{X,Y}(x,y) = \mathbb{P}(X = x \wedge Y = y)$.

  Extends trivially to more than two RVs.

- **Joint pdf** of two continuous RVs: $f_{X,Y}(x,y)$, such that

$$\mathbb{P}\big((X,Y) \in A\big) = \iint_A f_{X,Y}(x,y)\, dx\, dy.$$

  Extends trivially to more than two RVs.

- **Marginalization**: $f_Y(y) = \begin{cases} \displaystyle\sum_x f_{X,Y}(x,y), & \text{if } X \text{ is discrete} \\[2mm] \displaystyle\int_{-\infty}^{\infty} f_{X,Y}(x,y)\, dx, & \text{if } X \text{ continuous} \end{cases}$

- **Independence**:

  $X \perp\!\!\!\perp Y \;\Leftrightarrow\; f_{X,Y}(x,y) = f_X(x)\, f_Y(y) \;\; \genfrac{}{}{0pt}{}{\Rightarrow}{\not\Leftarrow} \;\; \mathbb{E}(X\,Y) = \mathbb{E}(X)\,\mathbb{E}(Y).$

Conditionals and Bayes' Theorem

- Conditional pmf (discrete RVs):
$$f_{X|Y}(x|y) = \mathbb{P}(X = x | Y = y) = \frac{\mathbb{P}(X = x \wedge Y = y)}{\mathbb{P}(Y = y)} = \frac{f_{X,Y}(x,y)}{f_Y(y)}.$$

- Conditional pdf (continuous RVs): $f_{X|Y}(x|y) = \dfrac{f_{X,Y}(x,y)}{f_Y(y)}$

- Bayes' theorem: $f_{X|Y}(x|y) = \dfrac{f_{Y|X}(y|x)\, f_X(x)}{f_Y(y)}$ (pdf or pmf).

- Also valid in the mixed case (*e.g.*, X continuous, Y discrete).

Joint, Marginal, and Conditional Probabilities: An Example

- A pair of binary variables $X, Y \in \{0, 1\}$, with joint pmf:

$f_{X,Y}(x,y)$	$Y = 0$	$Y = 1$
$X = 0$	1/5	2/5
$X = 1$	1/10	3/10

- Marginals: $f_X(0) = \frac{1}{5} + \frac{2}{5} = \frac{3}{5},$ $\quad f_X(1) = \frac{1}{10} + \frac{3}{10} = \frac{4}{10},$

  $f_Y(0) = \frac{1}{5} + \frac{1}{10} = \frac{3}{10},$ $\quad f_Y(1) = \frac{2}{5} + \frac{3}{10} = \frac{7}{10}.$

- Conditional probabilities:

$f_{X|Y}(x|y)$	$Y = 0$	$Y = 1$
$X = 0$	2/3	4/7
$X = 1$	1/3	3/7

$f_{Y|X}(y|x)$	$Y = 0$	$Y = 1$
$X = 0$	1/3	2/3
$X = 1$	1/4	3/4

An Important Multivariate RV: Multinomial

- Multinomial: $X = (X_1, ..., X_K)$, $X_i \in \{0, ..., n\}$, such that $\sum_i X_i = n$,

$$f_X(x_1, ..., x_K) = \begin{cases} \binom{n}{x_1 \ x_2 \ \cdots \ x_K} p_1^{x_1} p_2^{x_2} \cdots p_k^{x_K} & \Leftarrow \quad \sum_i x_i = n \\ 0 & \Leftarrow \quad \sum_i x_i \neq n \end{cases}$$

$$\binom{n}{x_1 \ x_2 \ \cdots \ x_K} = \frac{n!}{x_1! \ x_2! \ \cdots \ x_K!}$$

Parameters: $p_1, ..., p_K \geq 0$, such that $\sum_i p_i = 1$.

- Generalizes the binomial from binary to K-classes.

- Example: tossing n independent fair dice, $p_1 = \cdots = p_6 = 1/6$.
  $x_i = $ number of outcomes with i dots. Of course, $\sum_i x_i = n$.

- For $n = 1$, sometimes called categorical or multinoulli.
  For $n = 1$, one and only one $x_i = 1$, others are 0, thus $\binom{n}{x_1 \ \cdots \ x_K} = 1$.

An Important Multivariate RV: Gaussian

- Multivariate Gaussian: $X \in \mathbb{R}^n$,

$$f_X(x) = \mathcal{N}(x; \mu, C) = \frac{1}{\sqrt{\det(2\pi C)}} \exp\left(-\frac{1}{2}(x-\mu)^T C^{-1}(x-\mu)\right)$$

- Parameters: vector $\mu \in \mathbb{R}^n$ and matrix $C \in \mathbb{R}^{n \times n}$.
  Expected value: $\mathbb{E}(X) = \mu$. Meaning of C: next slide.
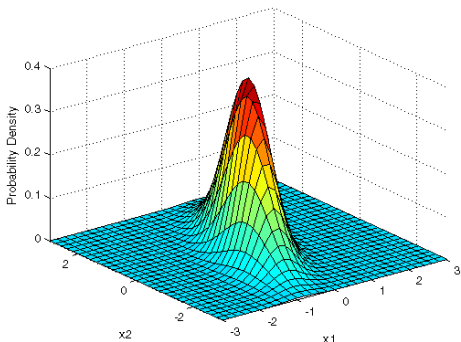
Covariance, Correlation, and all that...

- Covariance between two RVs:

$$\text{cov}(X, Y) = \mathbb{E}\Big[\big(X - \mathbb{E}(X)\big)\big(Y - \mathbb{E}(Y)\big)\Big] = \mathbb{E}(X\,Y) - \mathbb{E}(X)\,\mathbb{E}(Y)$$

- Relationship with variance: $\text{var}(X) = \text{cov}(X, X)$.

- $X \perp\!\!\!\perp Y \;\Leftrightarrow\; f_{X,Y}(x,y) = f_X(x)\,f_Y(y) \;\overset{\Rightarrow}{\nRightarrow}\; \text{cov}(X,\,Y) = 0$

- Covariance matrix of multivariate RV, $X \in \mathbb{R}^n$:

$$\text{cov}(X) = \mathbb{E}\Big[\big(X - \mathbb{E}(X)\big)\big(X - \mathbb{E}(X)\big)^T\Big] = \mathbb{E}(X\,X^T) - \mathbb{E}(X)\mathbb{E}(X)^T$$

- Covariance of Gaussian RV, $f_X(x) = \mathcal{N}(x; \mu, C) \;\Rightarrow\; \text{cov}(X) = C$

More on Expectations and Covariances

Let $A \in \mathbb{R}^{n \times n}$ be a matrix and $a \in \mathbb{R}^n$ a vector.

- If $\mathbb{E}(X) = \mu$ and $Y = AX$, then $\mathbb{E}(Y) = A\mu$;

- If $\text{cov}(X) = C$ and $Y = AX$, then $\text{cov}(Y) = ACA^T$;

- If $\text{cov}(X) = C$ and $Y = a^T X \in \mathbb{R}$, then $\text{var}(Y) = a^T Ca \geq 0$;

- If $\text{cov}(X) = C$ and $Y = C^{-1/2}X$, then $\text{cov}(Y) = I$;

- If $f_X(x) = \mathcal{N}(x; 0, I)$ and $Y = \mu + C^{1/2}X$, then $f_Y(y) = \mathcal{N}(y; \mu, C)$;

- If $f_X(x) = \mathcal{N}(x; \mu, C)$ and $Y = C^{-1/2}(X - \mu)$, then $f_Y(y) = \mathcal{N}(y; 0, I)$.

Entropy and all that...

Entropy of a discrete RV $X \in \{1, ..., K\}$:

$$H(X) = -\sum_{x=1}^{K} f_X(x) \log f_X(x)$$

- Positivity: $H(X) \geq 0$;

  $H(X) = 0 \iff f_X(i) = 1$, for exactly one $i \in \{1, ..., K\}$.

- Upper bound: $H(X) \leq \log K$;

  $H(X) = \log K \iff f_X(x) = 1/k$, for all $x \in \{1, ..., K\}$

- Measure of uncertainty/randomness of X

Continuous RV X, differential entropy:

$$h(X) = -\int f_X(x) \log f_X(x) \, dx$$

- $h(X)$ can be positive or negative. Example, if
  $f_X(x) = \text{Uniform}(x; a, b)$, $h(X) = \log(b - a)$.

- If $f_X(x) = \mathcal{N}(x; \mu, \sigma^2)$, then $h(X) = \frac{1}{2} \log(2\pi e \sigma^2)$.

Kullback-Leibler divergence

Kullback-Leibler divergence (KLD) between two pmf:

$$D(f_X \| g_X) = \sum_{x=1}^{K} f_X(x) \log \frac{f_X(x)}{g_X(x)}$$

Positivity: $D(f_X \| g_X) \geq 0$
$D(f_X \| g_X) = 0 \iff f_X(x) = g_X(x), \text{ for } x \in \{1, ..., K\}$

KLD between two pdf:

$$D(f_X \| g_X) = \int f_X(x) \log \frac{f_X(x)}{g_X(x)} \, dx$$

Positivity: $D(f_X \| g_X) \geq 0$
$D(f_X \| g_X) = 0 \iff f_X(x) = g_X(x), \text{ almost everywhere}$

Recommended Reading

- A. Maleki and T. Do, "Review of Probability Theory", Stanford University, 2017 (https://tinyurl.com/pz7p9g5)

- L. Wasserman, "All of Statistics: A Concise Course in Statistical Inference", Springer, 2004.

Minimizing a function

- We are given a function $f : \mathbb{R}^n \to \mathbb{R}$.

- Goal: find x^* that minimizes $f : \mathbb{R}^n \to \mathbb{R}$.

- Global minimum: for any $x \in \mathbb{R}^n$, $f(x^*) \leq f(x)$.

- Local minimum: for any $\|x - x^*\| \leq \delta \Rightarrow f(x^*) \leq f(x)$.

- Are local minima also global minima?

Convex Functions

- Function $f : \mathbb{R}^n \to \mathbb{R}$

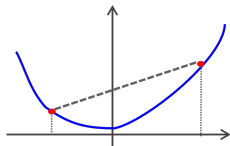- f is a convex function, if, for any $\lambda \in [0, 1]$, and any x, x',

$$f\big(\lambda x + (1 - \lambda)x'\big) \leq \lambda f(x) + (1 - \lambda)f(x')$$

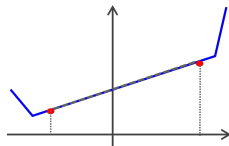- f is a strictly convex function, if, for any $\lambda \in \,]0, 1[$, and any x, x',

$$f\big(\lambda x + (1 - \lambda)x'\big) < \lambda f(x) + (1 - \lambda)f(x')$$
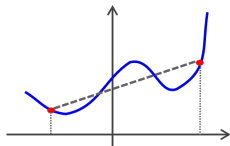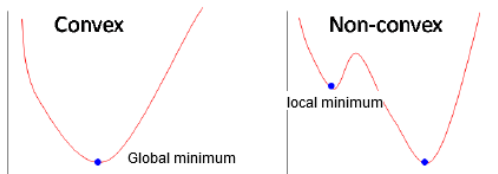

non-convex   convex   convex, not strictly

strictly convex

Relationship Between Convexity and Minimization

- Goal: find x^* that minimizes $f : \mathbb{R}^n \to \mathbb{R}$.

- If f is convex and x^* is a local minimizer, then it is also a global minimizer.

- If f is strictly convex and x^* is a local minimizer, then it is also the unique global minimizer.

Gradients and Minimization

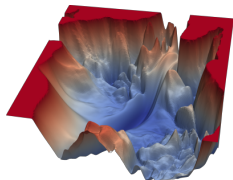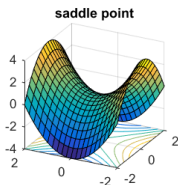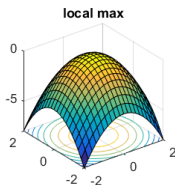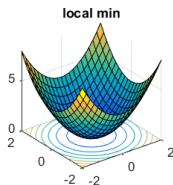- Given $f : \mathbb{R}^n \to \mathbb{R}$ (differentiable), the gradient of f at x:

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix} \in \mathbb{R}^n$$

- Relationship between gradient and minimization

$$x^* \text{ is local minimizer } \overset{\Rightarrow}{\underset{\nLeftarrow}{}} \nabla f(x^*) = 0$$

Hessians and Convexity

- Given $f : \mathbb{R}^n \to \mathbb{R}$ (differentiable), the Hessian of f at x:

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix} \in \mathbb{R}^{n \times n}$$

- Relationship between Hessian and convexity:

  ✓ Positive semi-definite Hessian $\Leftrightarrow$ f is convex

  ✓ Positive definite Hessian $\Leftrightarrow$ f is strictly convex.

More on Gradients

- Gradient of quadratic form $f(x) = x^T A x$: $\nabla f(x) = (A + A^T)x$

- ...if A symmetric: $\nabla f(x) = 2Ax$

- .Particular case: $f(x) = x^T x = \|x\|_2^2$, then $\nabla f(x) = 2x$

- If $f(x) = x^T b = b^T x$, then $\nabla f(x) = b$

- If $g(x) = f(Ax)$, then $\nabla g(x) = A^T \nabla f(Ax)$

- If $g(x) = f(a \odot x)$, then $\nabla g(x) = a \odot \nabla f(a \odot x)$

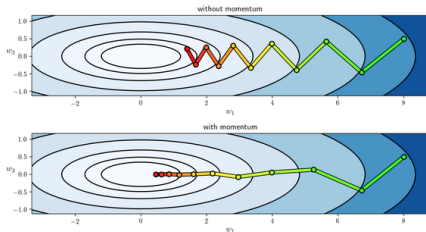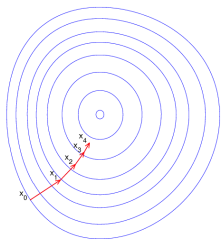- In simple cases, we can find minima analytically: $f(x) = \|Ax - y\|_2^2$

$$\nabla f(x) = 2A^T(Ax - y) = 0 \;\Rightarrow\; x^* = (A^T A)^{-1} A^T y$$

Gradient Descent

- Goal: minimize $f : \mathbb{R}^d \to \mathbb{R}$, for differentiable f
- Take small steps in the negative gradient direction until a stopping criterion is met:

$$x^{(t+1)} \leftarrow x^{(t)} - \eta_{(t)} \nabla f(x^{(t)})$$

- Choosing the step-size: crucial for convergence and performance.
- GD may work well, or not so well. There are many ways to improve it.

Recommended Reading

- Z. Kolter and C. Do, "Linear Algebra Review and Reference", Stanford University, 2015 (`https://tinyurl.com/44x2qj4`)

Machine Learning

Tom Mitchell's definition:

- "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."
- In a nutshell: learn from data; improve performance with experience

This comes in many flavours:

- Supervised learning
- Unsupervised learning
- Self-supervised learning
- Reinforcement learning
- Active learning

Formulate the problem; get data; learn the model from the data; evaluate.

Example Tasks

Binary classification: given an e-mail: is it spam or not-spam?

Multi-class classification: given a news article, determine its topic (politics, sports, etc.)

Regression: how much time a person will spend reading this article?

https://www.approximatelycorrect.com/2020/10/26/
superheroes-of-deep-learning-vol-1-machine-learning-yearning/

https://www.approximatelycorrect.com/2020/10/26/

superheroes-of-deep-learning-vol-1-machine-learning-yearning/

https://www.approximatelycorrect.com/2020/10/26/
superheroes-of-deep-learning-vol-1-machine-learning-yearning/

https://www.approximatelycorrect.com/2020/10/26/

superheroes-of-deep-learning-vol-1-machine-learning-yearning/

Let's Start Simple

- Example 1 – sequence: $\star \diamond \circ$; label: -1
- Example 2 – sequence: $\star \heartsuit \triangle$; label: -1
- Example 3 – sequence: $\star \triangle \spadesuit$; label: $+1$
- Example 4 – sequence: $\diamond \triangle \circ$; label: $+1$

- New sequence: $\star \diamond \circ$; label ?-1
- New sequence: $\star \diamond \heartsuit$; label ?$-1$
- New sequence: $\star \triangle \circ$; label ?

<div align="center" style="color:red">Why can we do this?</div>

(Credits: Ryan McDonald)

Let's Start Simple: Machine Learning

- Example 1 – sequence: $\star \diamond \circ$; label: -1
- Example 2 – sequence: $\star \heartsuit \triangle$; label: -1
- Example 3 – sequence: $\star \triangle \spadesuit$; label: $+1$
- Example 4 – sequence: $\diamond \triangle \circ$; label: $+1$

- New sequence: $\star \triangle \circ$; label ?

Label -1 **Label $+1$**

$P(-1|\star) = \frac{\text{count}(\star \text{ and } -1)}{\text{count}(\star)} = \frac{2}{3} = 0.67$ vs. $P(+1|\star) = \frac{\text{count}(\star \text{ and } +1)}{\text{count}(\star)} = \frac{1}{3} = 0.33$

$P(-1|\triangle) = \frac{\text{count}(\triangle \text{ and } -1)}{\text{count}(\triangle)} = \frac{1}{3} = 0.33$ vs. $P(+1|\triangle) = \frac{\text{count}(\triangle \text{ and } +1)}{\text{count}(\triangle)} = \frac{2}{3} = 0.67$

$P(-1|\circ) = \frac{\text{count}(\circ \text{ and } -1)}{\text{count}(\circ)} = \frac{1}{2} = 0.5$ vs. $P(+1|\circ) = \frac{\text{count}(\circ \text{ and } +1)}{\text{count}(\circ)} = \frac{1}{2} = 0.5$

(Credits: Ryan McDonald)

Machine Learning

① Define a model/distribution of interest
② Make some assumptions if needed
③ Fit the model to the data

- Model: $P(\text{label}|\text{sequence}) = P(\text{label}|\text{symbol}_1, \ldots \text{symbol}_n)$
  - Prediction for new sequence $= \text{argmax}_{\text{label}} P(\text{label}|\text{sequence})$

- Assumption (naive Bayes):

$$P(\text{symbol}_1, \ldots, \text{symbol}_n|\text{label}) = \prod_{i=1}^{n} P(\text{symbol}_i|\text{label})$$

- Fit the model to the data: count!! (simple probabilistic modeling)

Some Notation: Inputs and Outputs

- Input $x \in \mathcal{X}$
  - e.g., a news article, a sentence, an image, ...
- Output $y \in \mathcal{Y}$
  - e.g., spam/not spam, a topic, the object in the image (cat? dog?); a segmentation of the image (pedestrian; car; grass; background)

- Input/Output pair: $(x, y) \in \mathcal{X} \times \mathcal{Y}$
  - e.g., a **news article** together with a **topic**
  - e.g., a **image** together with an **object**
  - e.g., an **image** partitioned into **segmentation regions**

Many Flavours

- **Supervised learning**: pairs (x, y) are provided at training time (the main focus of this class)
  - Examples: perceptron, SVMs, decision trees, nearest neighbor, neural networks, ...
  - Caveat: the labels y may be hard or expensive to annotate
- **Unsupervised learning**: only x is provided; the model needs to figure out what the labels are without any supervision
  - Examples: clustering, PCA, ...
- **Reinforcement learning**: x is provided, and the model can act on the environment to obtain a reward (but doesn't get to know y)
  - Example: a robot acting on an environment trying to achieve a goal
- **Active learning**: the model requests which data points to label next.
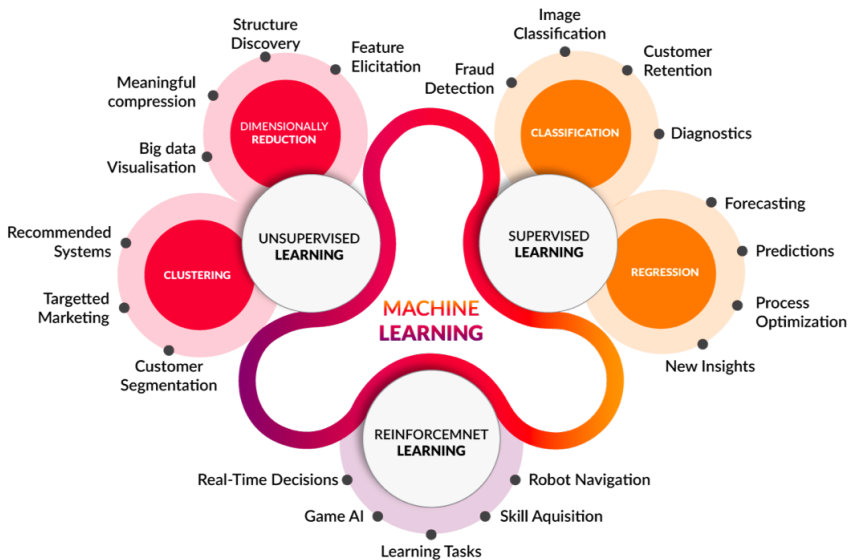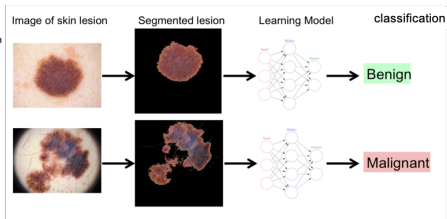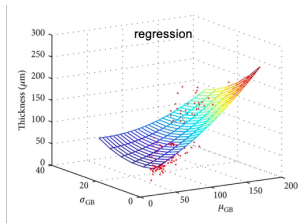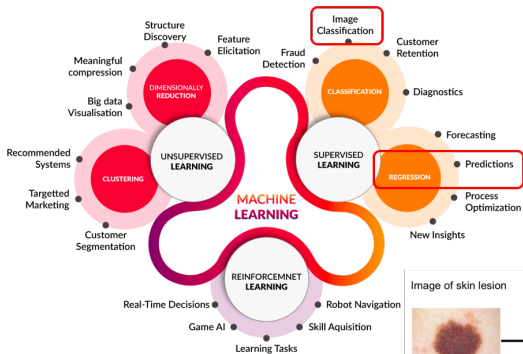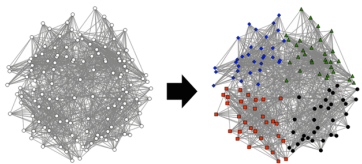
Community extraction from networks
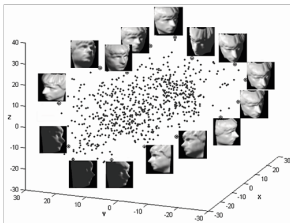
Manifold learning
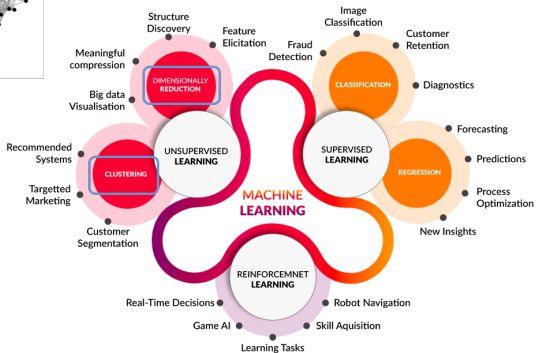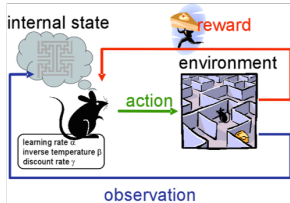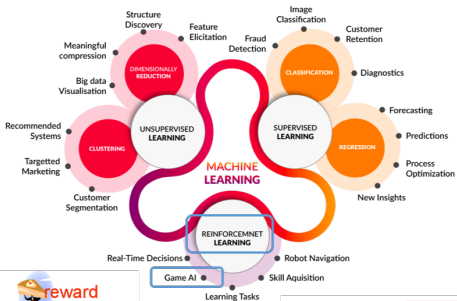
Supervised Learning

- We are given a **labeled dataset** of input/output pairs:

$$\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N} \subseteq \mathcal{X} \times \mathcal{Y}$$

- **Goal:** learn a **predictor** $h : \mathcal{X} \to \mathcal{Y}$ that generalizes well to arbitrary inputs.

- At test time, given $x \in \mathcal{X}$, we predict

$$\widehat{y} = h(x).$$

- Hopefully, $\widehat{y} \approx y$ most of the time.

Tasks/problems have different names depending on what $\mathcal{Y}$ is...

Regression deals with continuous output variables:

- Simple **regression:** $\mathcal{Y} = \mathbb{R}$
  - e.g., given a news article, how much time a user will spend reading it?

- **Multivariate regression:** $\mathcal{Y} = \mathbb{R}^K$
  - e.g., predict the X-Y coordinates in an image where the user will click

Classification

Tasks/problems have different names depending on what $\mathcal{Y}$ is...

Classification deals with **discrete** output variables:

- **Binary classification:** $\mathcal{Y} = \{\pm 1\}$
  - e.g., spam detection

- **Multi-class classification:** $\mathcal{Y} = \{1, 2, \ldots, K\}$
  - e.g., topic classification

- **Structured classification:** $\mathcal{Y}$ exponentially large and structured
  - e.g., machine translation, caption generation, image segmentation
  - Later in this course!

Sometimes reductions are convenient:

- logistic regression reduces classification to regression
- one-vs-all reduces multi-class to binary
- greedy search reduces structured classification to multi-class

... but other times it's better to tackle the problem in its native form.

More later!

Conclusions

- Machine learning allows computer programs to learn from data (observations, interventions, ...) and improve performance with experience

- Can be supervised, unsupervised, self-supervised, reinforced, etc.

- Tasks can be (binary or multi-class) classification, regression, or more nuanced

Thank you!

Questions?
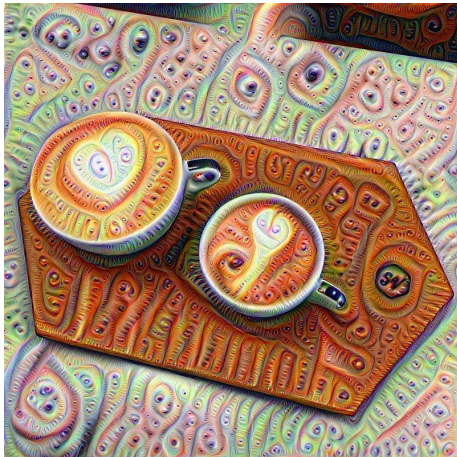
Lecture 2: Linear Models I

André Martins, Francisco Melo, Mário Figueiredo

Deep Learning Course, Winter 2022-2023

Today's Roadmap

- Linear regression.

- Binary and multi-class linear classification.

- Linear classifiers: perceptron.

Why Linear Classifiers?

We know the course title promised "deep", but...

- The underlying machine learning concepts are the same.

- The theory (statistics and optimization) are much better understood.

- Linear classifiers still widely used (effective when data is scarce).

- Linear classifiers are **a component of neural networks**.

Linear Classifiers and Neural Networks
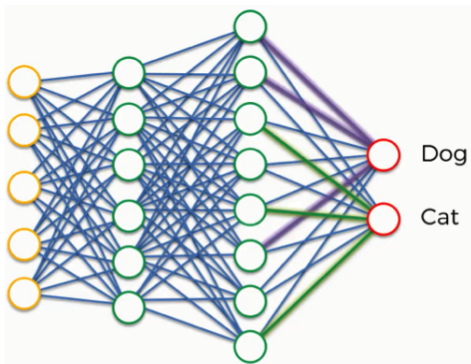
Feature Representations

Feature engineering is an important step in linear classifiers:

- Bag-of-words features for text, also lemmas, parts-of-speech, ...

- SIFT features and wavelet representations in computer vision.

Image gradients   Keypoint descriptor
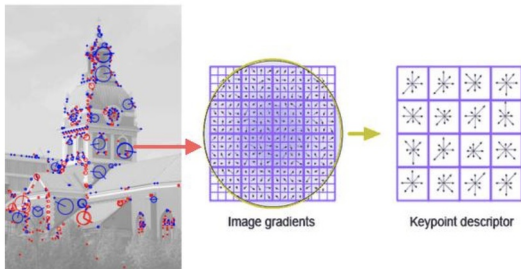
- Other categorical, Boolean, and continuous features.

Feature Representations

Representing information about x:

Typical approach: define a feature map $\phi : \mathcal{X} \to \mathbb{R}^D$.

- $\phi(x)$ is a (maybe high-dimensional) **feature vector**.

- $\phi(x)$ may include Boolean, categorical, and continuous features.

- Categorical features can be reduced to a one-hot binary vector.

  $\phi(x) \in \{1, 2, ..., K\}$ same information $\phi'(x) = [0, ..., 0, 1, 0, ..., 0] \in \{0, 1\}^K$

  with one and only one 1.

Example: Continuous Features

Handcrafted Features
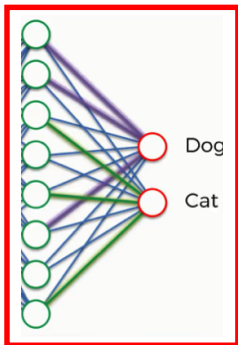
Linear Classifier

Feature Engineering and NLP Pipelines
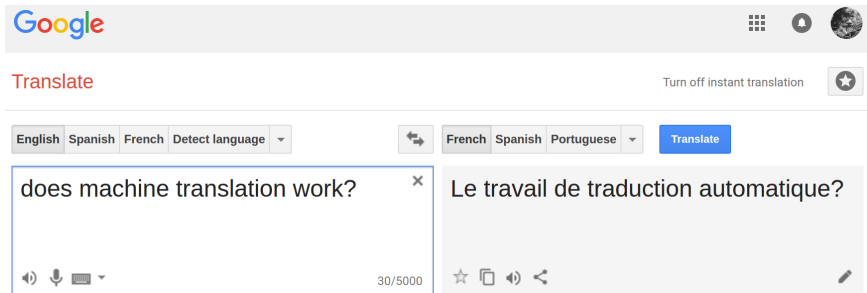
Classical NLP pipelines: stacking together several linear classifiers.

Each output is used to handcraft features for subsequent classifiers.

Examples of features:

- Word occurrences: binary features (word occurs or not in document).

- Word counts: numeric features, how many times a word occurs.

- POS tags: classify words as nouns, verbs, adjectives, ...

- Spell check: misspellings counts for spam detection.

Example: Translation Quality Estimation

Goal: estimate the quality of a translation on the fly (without a reference)!

Example: Translation Quality Estimation

Hand-crafted features:

- no of tokens in the source/target segment
- LM probability of source/target segment and their ratio
- % of source 1–3-grams observed in 4 frequency quartiles of source corpus
- average no of translations per source word
- ratio of brackets and punctuation symbols in source & target segments
- ratio of numbers, content/non-content words in source & target segments
- ratio of nouns/verbs/etc in the source & target segments
- % of dependency relations b/w constituents in source & target segments
- diff in depth of the syntactic trees of source & target segments
- diff in no of PP/NP/VP/ADJP/ADVP/CONJP in source & target
- diff in no of person/location/organization entities in source & target
- features and global score of the SMT system
- number of distinct hypotheses in the n-best list
- 1–3-gram LM probabilities using translations in the n-best to train the LM
- average size of the target phrases
- proportion of pruned search graph nodes;
- proportion of recombined graph nodes.

Representation Learning

Feature engineering is an "**art**" and can be very time-consuming

...but it's a good way of encoding **prior knowledge**, still widely used in practice (especially with "small data")

Alternative to feature engineering: **representation learning**

Neural networks will alleviate this (later in the course)!

Regression

Output space $\mathcal{Y}$ is continuous (*e.g.*, $\mathcal{Y} = \mathbb{R}$ or $\mathcal{Y} = [0, 1]$)

Example: given an article, how much time a user spends reading it?

Summer Schools and Machine
Learning. A beautiful love story!

Mohan Acharya Follow
Jan 7, 2019 · 7 min read

- Input x is number of words of the article
- Output $\hat{y}$ is the predicted reading time (minutes)

How to define the predictive model $\hat{y} = h(x)$?

Linear Regression

- Model: assume $\widehat{y} = wx + b$
- Parameters: w and b
- Given training data $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$, estimate w and b
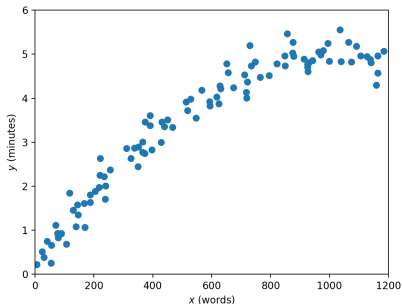
Least squares method: fit w and b on the training set by solving

$$\min_{w,\,b} \sum_{n=1}^{N} \big(y_n - \underbrace{(w\,x_n + b)}_{\hat{y}_n}\big)^2$$

Linear Regression

Often, linear dependency of $\widehat{y}$ on x is a bad/simplistic assumption

More general model: $\widehat{y} = w^T \phi(x)$, where $\phi(x)$ is a feature vector

- e.g. $\phi(x) = [1, x, x^2, \ldots, x^D]^T \in \mathbb{R}^{D+1}$ (monomials degree $\leq D$)
- the bias term b is captured by the constant feature $\phi_0(x) = 1$

Fit/learn w by solving $\displaystyle\min_{w \in \mathbb{R}^{D+1}} \sum_{n=1}^{N} \left(y_n - w^T \phi(x_n)\right)^2$

- Closed-form solution:

$$\hat{w} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}, \text{ with } \boldsymbol{X} = \begin{bmatrix} \phi(x_1)^T \\ \vdots \\ \phi(x_n)^T \\ \vdots \\ \phi(x_N)^T \end{bmatrix}, \ \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \\ \vdots \\ y_N \end{bmatrix}.$$

Still called linear regression – linear w.r.t. the model parameters w.

One-Slide Proof

Write the objective function in matrix-vector notation:

$$\sum_{n=1}^{N}\left(y_n - w^T\phi(x_n)\right)^2 = \|\boldsymbol{y} - \boldsymbol{X}w\|^2$$

Equate the gradient to zero and solve the resulting equation:

$$\begin{aligned}
0 &= \nabla_w\|\boldsymbol{y} - \boldsymbol{X}w\|^2 \\
&= \nabla_w\left(w^T\boldsymbol{X}^T\boldsymbol{X}w - 2w^T\boldsymbol{X}^T\boldsymbol{y} + \|\boldsymbol{y}\|^2\right) \\
&= 2\boldsymbol{X}^T\boldsymbol{X}w - 2\boldsymbol{X}^T\boldsymbol{y}
\end{aligned}$$

Therefore

$$\hat{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}.$$

Linear Regression $(D = 1)(D = 2)$
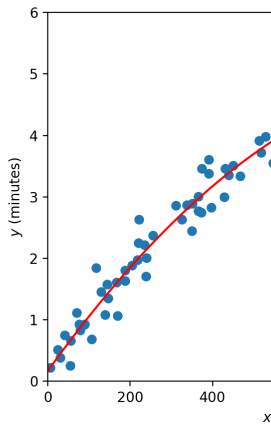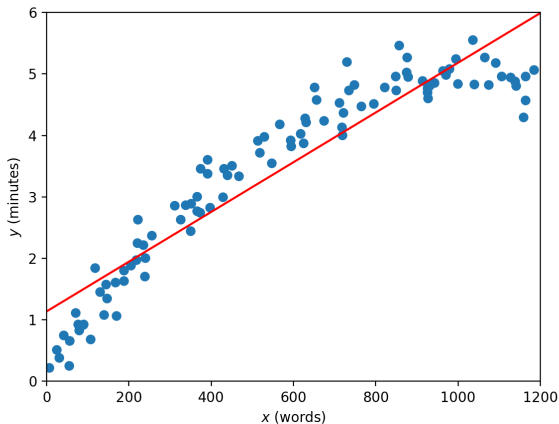
Squared Loss Function

Linear regression with least squares criterion corresponds to a loss function

$$L(y, \widehat{y}) = \frac{1}{2}(y - \widehat{y})^2, \quad \text{where } \widehat{y} = w^T \phi(x).$$

This is called the squared loss.

The model is fit to the training data by minimizing the loss function:

$$\hat{w} = \arg\min_w \sum_{n=1}^{N} L(y_n - \hat{y}_n) = \arg\min_w \frac{1}{2} \sum_{n=1}^{N} (y_n - w^T \phi(x))^2$$

(the factor $1/2$ is irrelevant but convenient)

More later.

Least Squares (LS): Probabilistic Interpretation

Assume the data is generated stochastically as

$$y_n = w_*^T \phi(x_n) + z_n$$

where:

- ✓ w_* is the "true" model parameter vector;
- ✓ $z_n \sim \mathcal{N}(0, \sigma^2)$ are independent Gaussian noise samples, with zero mean variance σ^2.

Consequently,

$$y_n \sim \mathcal{N}(w_*^T \phi(x_n), \sigma^2).$$

Then $\hat{w}$ given by LS is the maximum likelihood estimate under this model.

One-Slide Proof

Recall $\mathcal{N}(y; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right)$.

$$
\begin{aligned}
\hat{w}_{\text{MLE}} &= \arg\max_w P(\mathbf{y} \mid w) \\
&= \arg\max_w \log \prod_{n=1}^{N} P(y_n \mid w) && \text{(independent samples)} \\
&= \arg\max_w \sum_{n=1}^{N} \log P(y_n \mid w) && (\log \prod = \sum \log) \\
&= \arg\max_w \sum_{n=1}^{N} -\frac{(y_n - w^T \phi(x_n))^2}{2\sigma^2} - \underbrace{\log(\sqrt{2\pi\sigma^2})}_{\text{constant}} && (\log \mathcal{N}) \\
&= \arg\min_w \sum_{n=1}^{N} (y_n - w^T \phi(x_n))^2
\end{aligned}
$$

Thus, linear regression with the squared loss $=$ MLE under Gaussian noise.

Other Regression Losses

Squared loss: $L(y, \widehat{y}) = \frac{1}{2}(y - \widehat{y})^2$.

Absolute error loss: $L(y, \widehat{y}) = |y - \widehat{y}|$.

Huber loss: $L(y, \widehat{y}) = \begin{cases} \frac{1}{2}(y - \widehat{y})^2 & \text{if } |y - \widehat{y}| \leq 1 \\ |y - \widehat{y}| - \frac{1}{2} & \text{if } |y - \widehat{y}| \geq 1. \end{cases}$
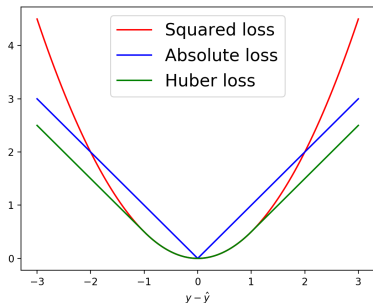
Quizz: which of these are convex; and strictly convex?

Overfitting and Underfitting

We saw earlier an example of underfitting (slide 15)

However, if the model is too complex (too many parameters) and/or the data is scarce, we run the risk of overfitting

To avoid overfitting, we often need regularization (more later)

Maximum A Posteriori

Assuming we have a prior distribution $w \sim \mathcal{N}(0, \tau^2 \boldsymbol{I})$

A criterion to estimate w_* is the maximum a posteriori (MAP):

$$
\begin{aligned}
\hat{w}_{\text{MAP}} &= \arg \max_w P(w \mid \boldsymbol{y}) = P(\boldsymbol{y}|w) P(w)/P(\boldsymbol{y}) \qquad (\textit{Bayes}) \\
&= \arg \max_w P(w) \prod_{n=1}^{N} P(y_n \mid w) \\
&= \arg \max_w \log P(w) + \sum_{n=1}^{N} \log P(y_n \mid w) \\
&= \arg \max_w -\frac{\|w\|^2}{2\tau^2} - \sum_{n=1}^{N} \frac{(y_n - w^T \phi(x_n))^2}{2\sigma^2} + \text{constant} \\
&= \arg \min_w \lambda \underbrace{\|w\|^2}_{\text{regularizer}} + \underbrace{\sum_{n=1}^{N} (y_n - w^T \phi(x_n))^2}_{\text{loss}}
\end{aligned}
$$

where $\lambda = \sigma^2/\tau^2$ is the so-called regularization constant

Thus, ℓ_2-regularization is equivalent to MAP with a Gaussian prior.

Binary Classification

Before multi-class, we start with simpler case of binary classification

Output set $\mathcal{Y} = \{-1, +1\}$

Example: Given a news article, is it true or fake?

- x is the news article, represented a feature vector $\phi(x)$

- y can be either $+1$ (true) or -1 (fake)

How to define a model to predict $\widehat{y}$ from x?

Linear Classifier

Defined by

$$\widehat{y} = \text{sign}(w^T \phi(x) + b) = \left\{ \begin{array}{ll} +1, & \text{if } w^T \phi(x) + b \geq 0 \\ -1, & \text{if } w^T \phi(x) + b < 0. \end{array} \right.$$

Intuitively, $w^T \phi(x) + b$ is a "score" for the positive class

Different from regression: sign function converts from continuous to binary

The decision boundary is a hyperplane (w.r.t. $\phi(x)$, not x)

$$w^T \phi(x) + b = 0$$

Thus also called a "hyperplane classifier."

Linear Classifier

(w, b) defines a hyperplane that splits the space into two half spaces:
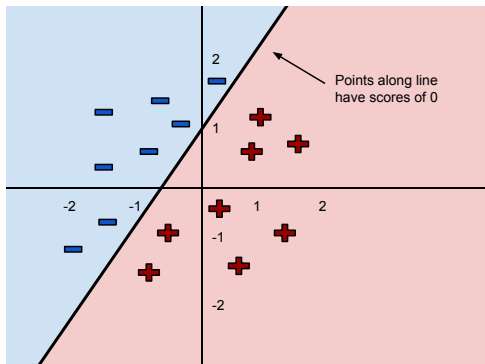
How to learn this hyperplane from the training data $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$?

Linear Separability

- A dataset $\mathcal{D}$ is linearly separable if there exists (w, b) such that classification is perfect

Separable   Not separable
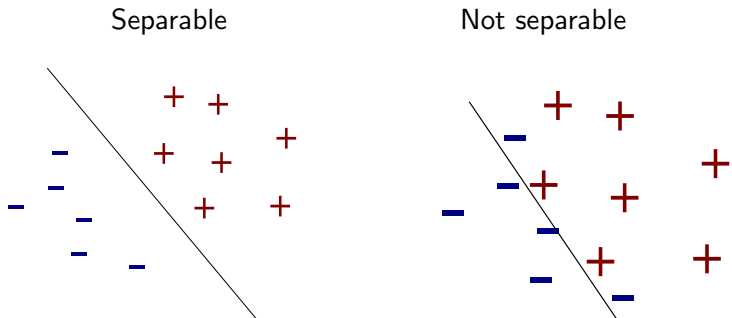
Next: an algorithm that finds such an hyperplane if it exists.

Linear Classifier: No Bias Term

It is common to present linear classifiers without the bias term b:

$$\widehat{y} = \text{sign}(w^T \phi(x) + b)$$

If $b = 0$, the decision hyperplane passes through the origin

We can always do this without loss of generality:

- Add a constant feature to $\phi(x)$: $\phi_0(x) = 1$
- The corresponding weight w_0 replaces the bias term b

Perceptron (Rosenblatt, 1958)
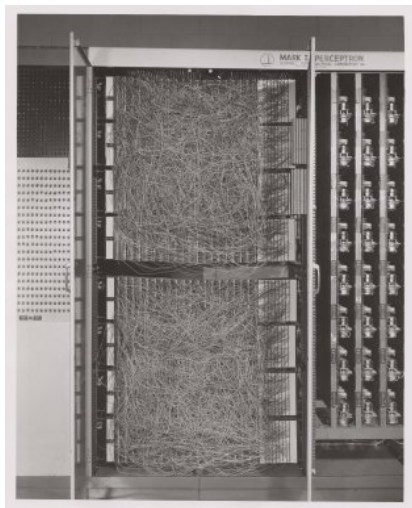
(Source: Wikipedia)

- Invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt
- Implemented in custom-built hardware as the "Mark 1 perceptron," for image recognition
- 400 photocells, randomly connected to the "neurons." Weights were encoded in potentiometers
- Weight updates during learning were performed by electric motors.

Perceptron in the News…

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.,

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human be-

ings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

1958 New York Times…

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

Perceptron Algorithm

Online algorithm: process one data point x_n at each round

1. Apply current model to x_n, get the corresponding prediction

2. If prediction is correct, do nothing

3. If it is wrong, correct w by adding/subtracting feature vector $\phi(x_i)$

Perceptron Algorithm

input: labeled data $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$
initialize $w^{(0)} = \mathbf{0}$
initialize $k = 0$ (number of mistakes)
repeat
  get new training example (x_n, y_n)
  predict $\widehat{y}_n = \text{sign}((w^{(k)})^T \phi(x_n))$
  if $\widehat{y}_n \neq y_n$ **then**
    update $w^{(k+1)} = w^{(k)} + y_n \, \phi(x_n)$
    increment k
  end if
until maximum number of epochs
output: model weights $w^{(k)}$

Perceptron's Mistake Bound

Definitions:

- The training data is linearly separable with margin $\gamma > 0$ iff there is a weight vector u, with $\|u\| = 1$, such that

$$y_n \, u^T \phi(x_n) \geq \gamma, \quad \forall n.$$

- Radius of the data: $R = \max_n \|\phi(x_n)\|$.

Then, the following bound of the number of mistakes holds:

Theorem (Novikoff (1962))

The perceptron algorithm is guaranteed to find a separating hyperplane after at most $\frac{R^2}{\gamma^2}$ mistakes.

One-Slide Proof

Recall that $w^{(k+1)} = w^{(k)} + y_n \, \phi(x_n)$.

- **Lower bound on** $\|w^{(k)}\|$:

$$
\begin{aligned}
u^T w^{(k)} &= u^T w^{(k-1)} + y_n \, u^T \phi(x_n) \\
&\geq u^T w^{(k-1)} + \gamma \\
&\geq u^T w^{(k-2)} + \gamma + \gamma \\
&\geq k \, \gamma \qquad \text{(recall } w^{(0)} = 0)
\end{aligned}
$$

Thus, $\|w^{(k)}\| = \underbrace{\|u\|}_{1} \|w^{(k)}\| \geq u^T w^{(k)} \geq k\gamma$ (Cauchy-Schwarz)

- **Upper bound on** $\|w^{(k)}\|$:

$$
\begin{aligned}
\|w^{(k)}\|^2 &= \|w^{(k-1)}\|^2 + \|\phi(x_n)\|^2 + 2 y_n \overbrace{(w^{(k)})^T \phi(x_n)}^{\leq 0,\ \text{if update}} \\
&\leq \|w^{(k-1)}\|^2 + R^2 \\
&\leq k \, R^2.
\end{aligned}
$$

Equating both sides, we get $(k\gamma)^2 \leq kR^2 \Rightarrow k \leq R^2/\gamma^2$ ∎

What a Simple Perceptron Can and Can't Do

- Remember: the decision boundary is linear (linear classifier)

- It **can** solve linearly separable problems (OR, AND)
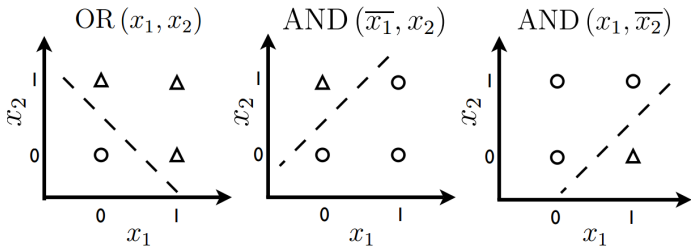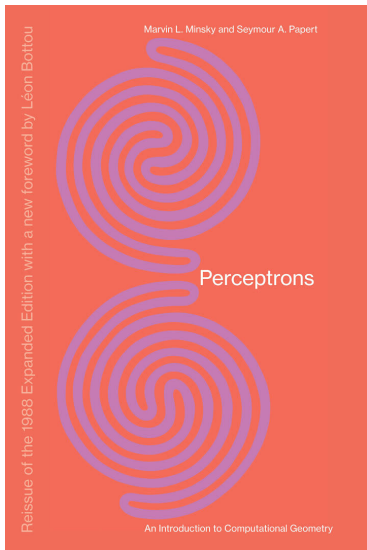
What a Simple Perceptron Can and Can't Do

- ... but it **can't** solve non-linearly separable problems such as simple XOR (unless input is transformed into a better representation):
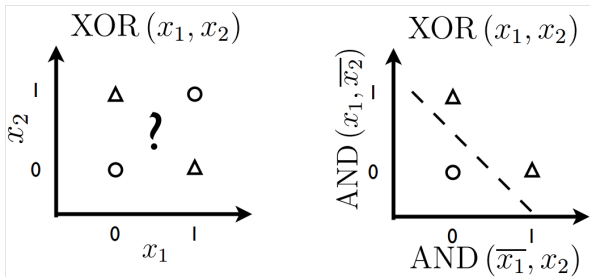
- This result is often attributed to Minsky and Papert (1969) but was known well before.

Limitations of the Perceptron

Minsky and Papert (1969):

- Shows limitations of multi-layer perceptrons and fostered an "AI winter" period.

More later in the neural networks' lecture!

Multi-Class Classification

Assume a multi-class problem, with $|\mathcal{Y}| \geq 2$ labels (classes).

Several strategies to reduce to binary classification:

- One-vs-all: train one binary classifier per class, using all others as negative examples; pick the class with the highest score.

- One-vs-one: another strategy is to train pairwise classifiers and use majority voting.

- Binary coding: use binary code for the class labels and learn a binary classifier for each bit.

Here, we will consider classifiers that tackle the multiple classes directly.

Multi-Class Linear Classifiers

- Parametrized by a weight matrix $\boldsymbol{W} \in \mathbb{R}^{|\mathcal{Y}| \times D}$ (one weight per feature/label pair) and a bias vector $\boldsymbol{b} \in \mathbb{R}^{|\mathcal{Y}|}$:

$$
\boldsymbol{W} = \left[\begin{array}{c} w_1^\top \\ \vdots \\ w_{|\mathcal{Y}|}^\top \end{array} \right], \ \boldsymbol{b} = \left[\begin{array}{c} b_1 \\ \vdots \\ b_{|\mathcal{Y}|} \end{array} \right].
$$

- Equivalently, $|\mathcal{Y}|$ weight vectors $w_y \in \mathbb{R}^D$ and scalars $b_y \in \mathbb{R}$

- The score of a particular label is based on a linear combination of features and their weights

- Predict the $\widehat{y}$ which maximizes this score:

$$
\widehat{y} = \arg\max_{y \in \mathcal{Y}} (w_y)^T \phi(x) + b_y = \arg\max_y \{(\boldsymbol{W}\phi(x) + \boldsymbol{b})_y, \ y \in \mathcal{Y}\}.
$$

$$
\equiv \arg\max(\boldsymbol{W}\phi(x) + \boldsymbol{b}) \quad \text{(compact notation)}
$$

Multi-Class Linear Classifier

Geometrically, $(\boldsymbol{W}, \boldsymbol{b})$ split the feature space into regions delimited by hyperplanes.
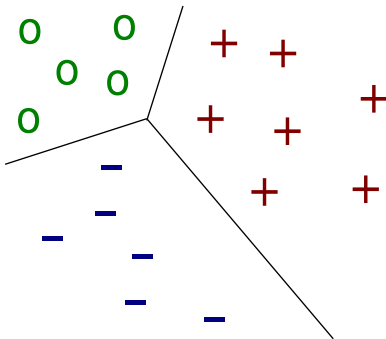
Commonly Used Notation in Neural Networks
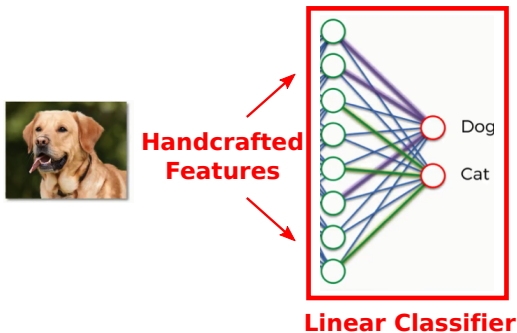
$$\widehat{y} = \arg\max\left(\boldsymbol{W}\phi(x) + \boldsymbol{b}\right), \quad \boldsymbol{W} = \begin{bmatrix} w_1^\top \\ \vdots \\ w_{|\mathcal{Y}|}^\top \end{bmatrix}, \; \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_{|\mathcal{Y}|} \end{bmatrix}.$$

Multi-Class Recovers Binary

For two classes ($\mathcal{Y} = \{\pm 1\}$), this formulation recovers the binary classifier presented earlier:

$$
\begin{aligned}
\widehat{y} &= \arg \max_{y \in \{\pm 1\}} (w_y)^T \phi(x) + b_y \\
&= \begin{cases} +1 & \text{if } (w_{+1})^T \phi(x) + b_{+1} > (w_{-1})^T \phi(x) + b_{-1} \\ -1 & \text{otherwise} \end{cases} \\
&= \text{sign} \big((\underbrace{w_{+1} - w_{-1}}_{w})^T \phi(x) + \underbrace{(b_{+1} - b_{-1})}_{b} \big).
\end{aligned}
$$

That is: only half of the parameters are needed.

Linear Classifiers (Binary vs Multi-Class)

- Prediction rule:

$$\widehat{y} = h(x) = \arg\max_{y \in \mathcal{Y}} \overbrace{(w_y)^T \phi(x)}^{\text{linear in } w_y}$$

- The decision boundary is defined by the intersection of half spaces
- In the binary case ($|\mathcal{Y}| = 2$) this corresponds to a hyperplane classifier
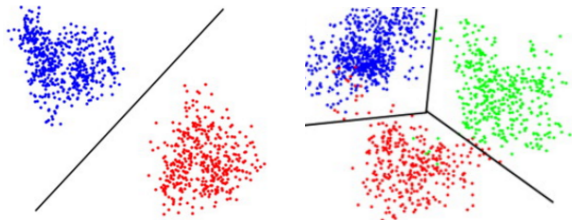
Linear Classifier – No Bias Term

Again, it is common to omit the bias vector b:

$$\widehat{y} = \arg \max_{y \in \mathcal{Y}} (w_y)^T \phi(x) + b_y$$

As before, this can be done, without loss of generality, by assuming a constant feature $\phi_0(x) = 1$

The first column of W replaces the bias vector.

We assume this for simplicity.

Example: Perceptron

The perceptron algorithm also works for the multi-class case!

It has a similar mistake bound: if the data is separable, it's guaranteed to find separating hyperplanes!

Perceptron Algorithm: Multi-Class

input: labeled data $\mathcal{D}$
initialize $\boldsymbol{W}^{(0)} = \boldsymbol{0}$
initialize $k = 0$ (number of mistakes)
repeat
  get new training example (x_n, y_n)
  predict $\widehat{y}_n = \arg\max_{y \in \mathcal{Y}} (w_y^{(k)})^T \phi(x_i)$
  if $\widehat{y}_n \neq y_n$ **then**
    update $w_{y_n}^{(k+1)} = w_{y_n}^{(k)} + \phi(x_n)$ $\{$ *increase weight of gold class* $\}$
    update $w_{\widehat{y}_n}^{(k+1)} = w_{\widehat{y}_n}^{(k)} - \phi(x_n)$ $\{$ *decrease weight of incorrect class* $\}$
    increment k
  end if
until maximum number of epochs
output: model weights $\boldsymbol{W}^{(k)}$

Conclusions

- Linear models involve manipulating weights and features.

- Linear regression is a simple method for regression which has a closed form solution.

- Linear classifiers include several well-known ML methods (both for binary and multi-class classification).

- Today we saw the perceptron and proved a mistake bound.

- Next class: logistic regression (another linear classifier).

References I

Minsky, M. and Papert, S. (1969). Perceptrons.

Novikoff, A. B. (1962). On convergence proofs for perceptrons. In *Symposium on the Mathematical Theory of Automata*.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Lecture 3: Linear Models II

André Martins, Francisco Melo, Mário Figueiredo

Deep Learning Course, Winter 2022-2023

Today's Roadmap

- Logistic regression

- Regularization and optimization

- Stochastic gradient descent.

Why Linear Classifiers?

We know the course title promised "deep", ...

...but (as explained in Lecture 2):

- The underlying machine learning concepts are the same

- The theory (statistics and optimization) are much better understood

- Linear classifiers still widely used (very effective when data is scarce)

- Linear classifiers are **a component of neural networks**.

Linear Classifiers and Neural Networks
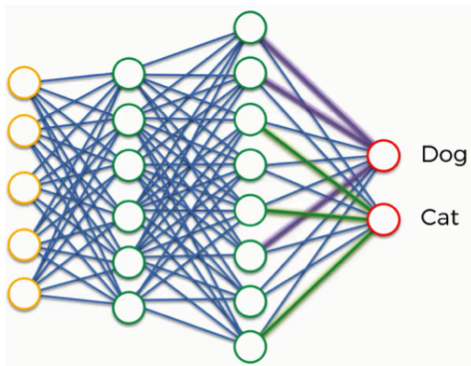
…in fact, these may also be learned features, from all the previous layers of some deep neural network.

Reminder: Linear Classifier

Linear Classifier

$$\widehat{y} = \arg \max_y ((\boldsymbol{W} \phi(x))_y), \quad \boldsymbol{W} = \left[\begin{array}{c} w_1^T \\ \vdots \\ w_{|\mathcal{Y}|}^T \end{array} \right]$$
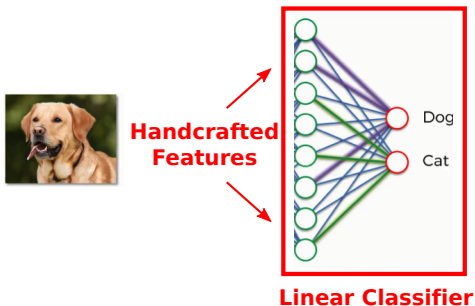
As before, assume $\phi_0(x) = 1$, absorbing the bias terms.

Key Problem

Mapping from a vector of scores $\mathbb{R}^{|\mathcal{Y}|}$ to a probability distribution over $\mathcal{Y}$?
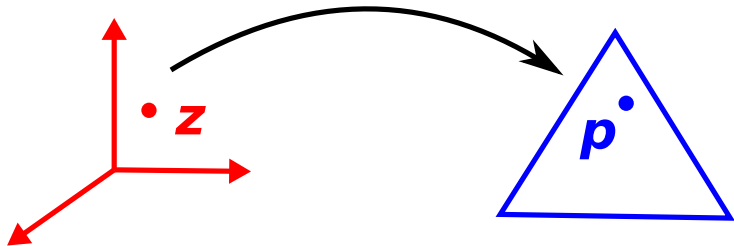
We will see an important mapping: softmax (next).

Logistic Regression

A linear model gives a score for each class y: $w_y^T \phi(x)$,

...from which we may compute a conditional (posterior) probability:

$$P(y|x) = \frac{\exp(w_y^T \phi(x))}{Z_x}, \qquad \text{where } Z_x = \sum_{y' \in \mathcal{Y}} \exp(w_{y'}^T \phi(x))$$

This is called the softmax transformation

Still a linear classifier

$$
\begin{aligned}
\arg\max_y \ P(y|x) &= \arg\max_y \ \frac{\exp(w_y^T \phi(x))}{Z_x} \\
&= \arg\max_y \ \exp(w_y^T \phi(x)) \\
&= \arg\max_y \ w_y^T \phi(x)
\end{aligned}
$$

The decision boundaries are still hyperplanes w.r.t. $\phi(x)$.

Binary Logistic Regression

Binary labels ($\mathcal{Y} = \{\pm 1\}$)

Scores: 0 for negative class, $w^T \phi(x)$ for positive class

$$
\begin{aligned}
P(y = +1 \mid x) &= \frac{\exp(w^T \phi(x))}{1 + \exp(w^T \phi(x))} \\
&= \frac{1}{1 + \exp(-w^T \phi(x))} \\
&\equiv \sigma(w^T \phi(x)).
\end{aligned}
$$

This is called a sigmoid transformation.

Sigmoid Transformation

$$\sigma(u) = \frac{e^u}{1 + e^u}$$
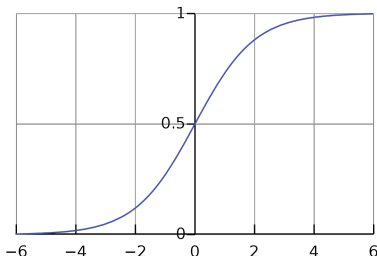
- Widely used in neural networks.
- Maps $\mathbb{R}$ into $[0, 1]$.
- The output can be interpreted as a probability.
- Positive, bounded, strictly increasing.

Multinomial Logistic Regression

$$P_{\boldsymbol{W}}(y \mid x) = \frac{\exp\left(w_y^T \phi(x)\right)}{Z_x}$$

- How to learn weights $\boldsymbol{W}$?
- Maximize the conditional log-likelihood of training data:

$$
\begin{aligned}
\widehat{\boldsymbol{W}} &= \arg\max_{\boldsymbol{W}} \log\left(\prod_{t=1}^{N} P_{\boldsymbol{W}}(y_t|x_t)\right) \\
&= \arg\min_{\boldsymbol{W}} -\sum_{t=1}^{N} \log P_{\boldsymbol{W}}(y_t|x_t) \\
&= \arg\min_{\boldsymbol{W}} \sum_{t=1}^{N} \left(\log \underbrace{\sum_{y'} \exp\left(w_{y'}^T \phi(x_t)\right)}_{Z_{x_t}} - w_{y_t}^T \phi(x_t) \right),
\end{aligned}
$$

- i.e., choose $\boldsymbol{W}$ to maximize the probability of the true labels.

Logistic Regression

- This objective function is strictly convex.

- Proof left as exercise! (hint, compute the Hessian).

- Therefore, any local minimum is a global minimum.

- No closed-form solution; many numerical methods have been proposed

    ✓ Gradient methods (gradient descent, conjugate gradient).

    ✓ Quasi-Newton methods (L-BFGS, ...).

Recap: Gradient Descent

- Goal: minimize $f : \mathbb{R}^d \to \mathbb{R}$, for differentiable f
- Take small steps in the negative gradient direction until a stopping criterion is met:

$$x^{(t+1)} \leftarrow x^{(t)} - \eta_t \nabla f(x^{(t)})$$

- Choosing the step-size: crucial for convergence and performance.
- GD may work well, or not so well. There are many ways to improve it.
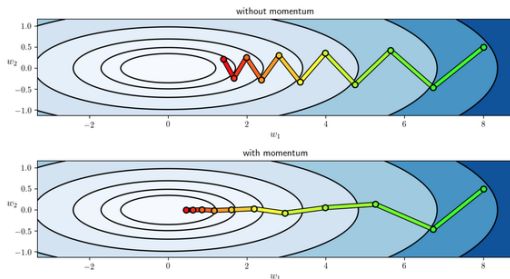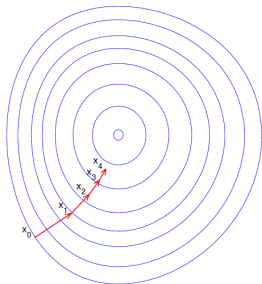
Gradient Descent

- Loss function in logistic regression is

$$L(\boldsymbol{W}; (x, y)) = \log \sum_{y'} \exp\left(w_{y'}^T \phi(x)\right) - w_y^T \phi(x).$$

- We want to find:

$$\arg \min_{\boldsymbol{W}} \sum_{t=1}^{N} L(\boldsymbol{W}; (x_t, y_t))$$

  ✓ Set $\boldsymbol{W}^{(0)} = \boldsymbol{0}$
  ✓ Iterate until convergence (for suitable stepsize η_k):

$$\boldsymbol{W}^{(k+1)} = \boldsymbol{W}^{(k)} - \eta_k \nabla_{\boldsymbol{W}} \left(\sum_{t=1}^{N} L(\boldsymbol{W}^{(k)}; (x_t, y_t)) \right)$$

$$= \boldsymbol{W}^{(k)} - \eta_k \sum_{t=1}^{N} \nabla_{\boldsymbol{W}} L(\boldsymbol{W}^{(k)}; (x_t, y_t))$$

- $L(\boldsymbol{W})$ convex $\Rightarrow$ gradient descent will reach the global optimum $\boldsymbol{W}$, with appropriate choice of step-size η_k.

Stochastic Gradient Descent

Monte Carlo approximation of the gradient (more frequent updates, convenient with large datasets):

- Set $\boldsymbol{W}^{(0)} = \boldsymbol{0}$

- Iterate until convergence:
    - Pick (x_t, y_t) randomly.

    - Update $\boldsymbol{W}^{(k+1)} = \boldsymbol{W}^{(0)} - \eta_k \nabla_{\boldsymbol{W}} L(\boldsymbol{W}^{(k)}; (x_t, y_t))$

- *i.e.* approximate the gradient with a noisy, unbiased, version based on a single sample
- Variant between GD and SGD: mini-batch (pick B, rather than 1 pair)
- All guaranteed to find the optimal $\boldsymbol{W}$ (for suitable step sizes)

Stochastic vs Batch Gradient Descent

— Batch gradient descent
— Mini-batch gradient Descent
— Stochastic gradient descent
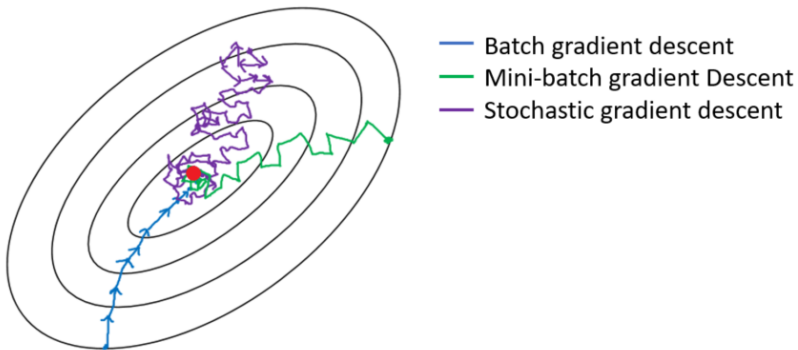
Computing the Gradient

- We need to compute $\nabla_{\boldsymbol{W}} L(\boldsymbol{W}; (x_t, y_t))$, where

$$L(\boldsymbol{W}; (x, y)) = \log \sum_{y'} \exp(w_{y'}^T \phi(x)) - w_y^T \phi(x)$$

- Some reminders (we next omit the subscript $\boldsymbol{W}$ for simplicity)

  ❶ $\nabla \log F(\boldsymbol{W}) = \dfrac{\nabla F(\boldsymbol{W})}{F(\boldsymbol{W})}$

  ❷ $\nabla \exp F(\boldsymbol{W}) = \exp(F(\boldsymbol{W})) \nabla F(\boldsymbol{W})$

- We denote by

$$\boldsymbol{e}_y = [0, \ldots, 0, 1, 0, \ldots, 0]^T, \quad \text{1 in } y\text{-th position}$$

  the one-hot vector representation of class y.

Computing the Gradient

$$
\begin{aligned}
\nabla L(\boldsymbol{W}; (x, y)) &= \nabla\left(\log \sum_{y'} \exp(w_{y'}^T \phi(x)) - w_y^T \phi(x)\right) \\
&= \nabla \log \sum_{y'} \exp(w_{y'}^T \phi(x)) - \nabla w_y^T \phi(x) \\
&= \frac{1}{\sum_{y'} \exp(w_{y'}^T \phi(x))} \sum_{y'} \nabla \exp(w_{y'}^T \phi(x)) - e_y \phi(x)^T \\
&= \frac{1}{Z_x} \sum_{y'} \exp(w_{y'}^T \phi(x)) \nabla w_{y'}^T \phi(x) - e_y \phi(x)^T \\
&= \sum_{y'} \frac{\exp(w_{y'}^T \phi(x))}{Z_x} e_{y'} \phi(x)^T - e_y \phi(x)^T \\
&= \sum_{y'} P_{\boldsymbol{W}}(y'|x) e_{y'} \phi(x)^T - e_y \phi(x)^T \\
&= \left(\begin{bmatrix} P_{\boldsymbol{W}}(1|x) \\ \vdots \\ P_{\boldsymbol{W}}(|\mathcal{Y}| \mid x) \end{bmatrix} - e_y\right) \phi(x)^T.
\end{aligned}
$$

Logistic Regression Summary

- Define conditional probability

$$P_{\boldsymbol{W}}(y|x) = \frac{\exp\left(w_y^T \phi(x)\right)}{Z_x}$$

- Set weights to maximize conditional log-likelihood of training data:

$$\boldsymbol{W} = \arg\max_{\boldsymbol{W}} \sum_t \log P_{\boldsymbol{W}}(y_t|x_t) = \arg\min_{\boldsymbol{W}} \sum_t L(\boldsymbol{W}; (x_t, y_t))$$

- Run gradient descent, or another gradient-based algorithm, using

$$\nabla L(\boldsymbol{W}; (x, y)) = \sum_{y'} P_{\boldsymbol{W}}(y'|x)\boldsymbol{e}_{y'}\phi(x)^\top - \boldsymbol{e}_y\phi(x)^\top$$

The Story So Far

- **Logistic regression** is **discriminative**: maximizes conditional likelihood

  ✓ also called log-linear model and max-entropy classifier

  ✓ no closed-form solution

  ✓ stochastic gradient updates look like

$$\boldsymbol{W}^{(k+1)} = \boldsymbol{W}^{(k)} + \eta_k \left(\boldsymbol{e}_y \boldsymbol{\phi}(x)^\top - \sum_{y'} P_{\boldsymbol{W}}(y'|x) \boldsymbol{e}_{y'} \boldsymbol{\phi}(x)^\top \right)$$

- The **perceptron** is also **discriminative**, but non-probabilistic classifier

  ✓ perceptron's updates look like

$$\boldsymbol{W}^{(k+1)} = \boldsymbol{W}^{(k)} + \boldsymbol{e}_y \boldsymbol{\phi}(x)^\top - \boldsymbol{e}_{\widehat{y}} \boldsymbol{\phi}(x)^\top$$

- SGD updates for logistic regression and the perceptron look similar!

Other Options: Maximizing Margin

- For a training set $\mathcal{D} = \{(x_t, y_t), \ t = 1, ..., N\}$

- Margin achieved by a weight matrix $\boldsymbol{W}$: smallest γ such that, for every training instance $(x_t, y_t) \in \mathcal{D}$,

$$w_{y_t}^T \phi(x_t) \geq w_{y'}^T \phi(x_t) + \gamma, \quad \text{for any} \ y' \neq y_t$$

- i.e., score of the correct label is at least γ higher than that of any other label.

Margin

Training
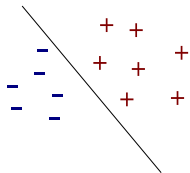
Testing
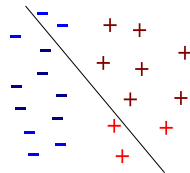
Denote the
value of the
margin by γ
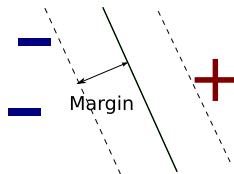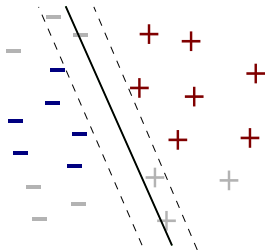
Margin

Maximizing Margin

- Intuitively maximizing margin makes sense

- More importantly, generalization error to unseen test data is proportional to the inverse of the margin

$$\epsilon \propto \frac{R^2}{\gamma^2 N}$$

- **Perceptron**:

  - ✓ If a training set is separable by some margin, the perceptron will find a W that separates the data

  - ✓ However, the perceptron does not pick W to maximize the margin!

  - ✓ Support vector machines (SVM) do exactly this (not covered)

Summary

What we saw:

- Linear Classifiers

  - ✓ Logistic regression

  - ✓ Perceptron

  - ✓ Support vector machines (not covered)

Next:

- Regularization

- Softmax

- Non-linear classifiers

Overfitting

If the model is too complex (too many parameters) and the data is scarce, we run the risk of overfitting:
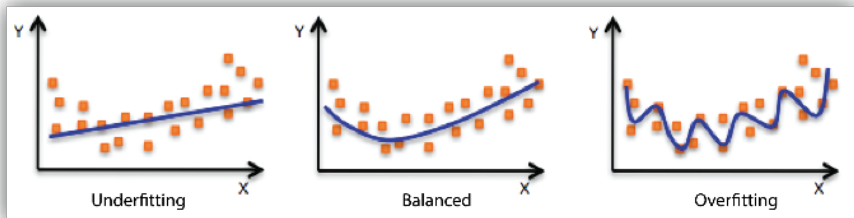
Regularization

In practice, we regularize to prevent overfitting

$$\arg\min_{\boldsymbol{W}} \sum_{t=1}^{N} L(\boldsymbol{W}; (x_t, y_t)) + \lambda\Omega(\boldsymbol{W}),$$

$\Omega(\boldsymbol{W})$ is the regularization function, and λ controls its weight.

- ℓ_2 regularization promotes smaller weights:

$$\Omega(\boldsymbol{W}) = \sum_{y} \|w_y\|_2^2 = \sum_{y} \sum_{i} w_{y,i}^2.$$

- ℓ_1 regularization promotes smaller and sparse weights!

$$\Omega(\boldsymbol{W}) = \sum_{y} \|w_y\|_1 = \sum_{y} \sum_{i} |w_{y,i}|$$

Empirical Risk Minimization: The Classical View
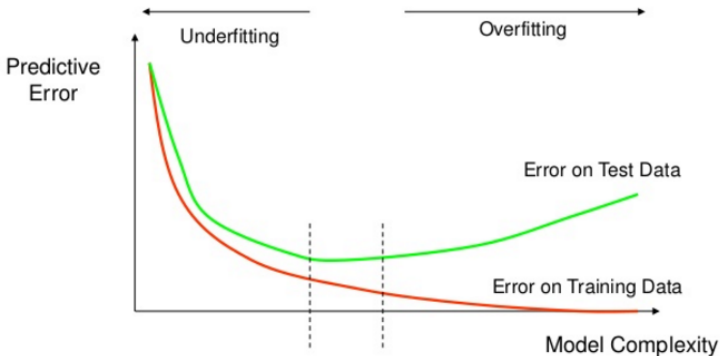
...doesn't explain the excellent performance of deep networks, which are very complex (have many parameters).

Empirical Risk Minimization: Double Descent
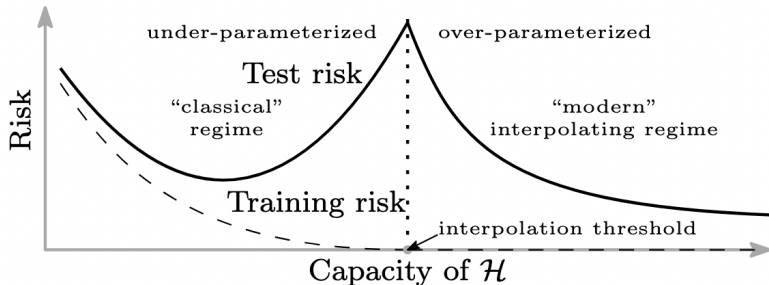
Fully understanding double descent is an open research topic.

Logistic Regression with ℓ_2 Regularization

- ℓ_2-regularized (a.k.a. ridge or weight decay) logistic regression

$$\min -\sum_{t=1}^{N} \underbrace{\log\left(\exp(w_{y_t}^T \phi(x_t))/Z_x\right)}_{L(\boldsymbol{W};(x_t,y_t))} + \frac{\lambda}{2}\|\boldsymbol{W}\|_F^2$$

- The gradient is now

$$\sum_{t=1}^{N} \nabla L(\boldsymbol{W};(x_t,y_t)) + \frac{\lambda}{2}\nabla\|\boldsymbol{W}\|_2^2$$

- For gradient descent, we also need $\nabla\|\boldsymbol{W}\|_2^2 = 2\boldsymbol{W}$
- GD step (similarly for SGD) with weight decay:

$$\boldsymbol{W}^{(k+1)} = \underbrace{(1-\eta_k\lambda)}_{\text{weight decay } (1-\eta_k\lambda)<1} \boldsymbol{W}^{(k)} - \eta_k \sum_{t=1}^{N} \nabla L(\boldsymbol{W};(x_t,y_t))$$

Loss Functions

- Should match the metric we want to optimize at test time

- Should be well-behaved (convex, hopefully smooth) for optimization
  (rules out the 0/1 loss)

- Some examples:

  ✓ Squared loss for regression

  ✓ Negative log-likelihood (a.k.a. cross-entropy) for multinomial logistic
    regression

  ✓ Hinge lossfor support vector machines (not covered)

  ✓ Sparsemax loss for multi-class and multi-label classification (not
    covered)

Classification Losses (Binary Case)

- The horizontal axis shows $s = -y\, w^T \phi(x)$.
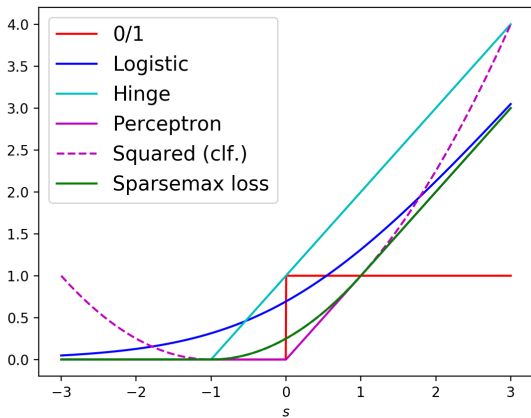
Recap

Mapping a label score vector $z \in \mathbb{R}^{|\mathcal{Y}|}$ to a probability distribution in $\mathcal{Y}$?
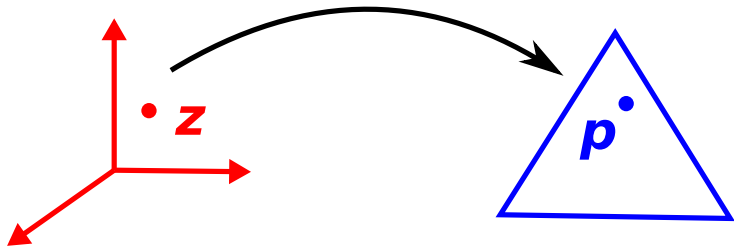
We already saw one example: softmax.

Another example is sparsemax (not covered): Martins and Astudillo (2016)

Recap: Softmax Transformation

The typical transformation for multi-class classification:

$$\mathbf{softmax} : \mathbb{R}^{|\mathcal{Y}|} \to \Delta_{|\mathcal{Y}|-1}$$

$$\mathbf{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_c \exp(z_c)}, \ldots, \frac{\exp(z_{|\mathcal{Y}|})}{\sum_c \exp(z_c)} \right]^T$$

- Underlies multinomial logistic regression

- Strictly positive, sums to 1

- Resulting distribution has full support: $\mathbf{softmax}(z) > \mathbf{0}, \forall z$

Recap: Multinomial Logistic Regression

- The common choice for a softmax output layer

- The classifier estimates $P(y \mid x; \boldsymbol{W})$

- We minimize the negative log-likelihood:

$$
\begin{aligned}
L(\boldsymbol{W}; (x, y)) &= -\log P(y \mid x; \boldsymbol{W}) \\
&= -\log \big(\mathbf{softmax}(z(x))\big)_y,
\end{aligned}
$$

where $z_c(x) = w_c^T \phi(x)$ is the score of class c for sample x.

- Loss gradient:

$$
\nabla L(\boldsymbol{W}; (x, y)) = \big(\mathbf{softmax}(z(x)) - e_y\big) \phi(x)^T
$$

Recap: What a Linear Classifier Can Do

- It **can** solve linearly separable problems (OR, AND)
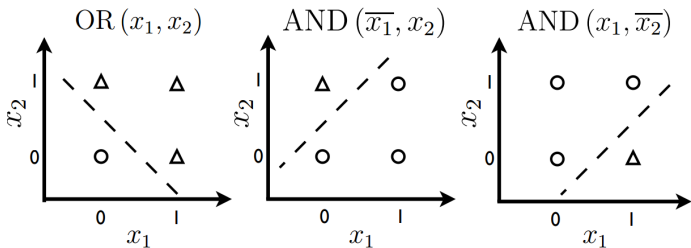
Recap: What a Linear Classifier Can't Do

- ... but it **cannot** solve non-linearly separable problems such as simple XOR (unless input is transformed into a better representation):
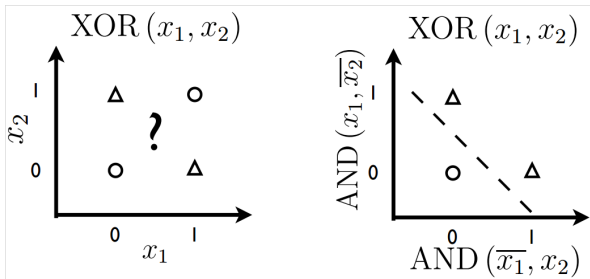
- This was observed by Minsky and Papert (1969) (for the perceptron) and motivated strong criticisms

Summary: Linear Classifiers

- We have seen

    ✓ Perceptron

    ✓ Logistic regression

    ✓ Support vector machines (not covered)

- All lead to convex optimization problems: no issues with local minima.

- All assume the features are well-engineered such that the data is (nearly) linearly separable

What If Data Are Not Linearly Separable?

Engineer better features (often works!)

Kernel methods:

- works implicitly in a high-dimensional feature space
- ... but still need to choose/design a good kernel
- model capacity confined to positive-definite kernels

Neural networks (next class!)

- embrace non-convexity and local minima
- instead of engineering features/kernels, engineer the model architecture

Two Views of Machine Learning

There's two big ways of building machine learning systems:

1. Feature-based: describe object properties (features) and build models that manipulate them.
   - ✓ What that we have seen so far.

2. Similarity-based: don't describe objects by their properties; rather, build systems based on **comparing** objects to each other
   - ✓ k-th nearest neighbors; kernel methods; Gaussian processes.

Sometimes the two are equivalent!

Nearest Neighbor(s) (NN) Classifier

- Not a linear classifier!

- In its simplest version, doesn't require any parameters

- No training, just memorize the data $\mathcal{D} = \{(x_i, y_i), \ i = 1, ..., N\}$

- Given a new input x, find its nearest data point x_i and predict

$$\widehat{y}(x) = y_n, \ \text{where } n = \arg\min_i \|x - x_i\|$$

- Variants: k-NN returns the majority class in the k nearest neighbours.

- Disadvantage: requires searching over the entire training data

- Specialized data structures can be used to speed up search.

Nearest Neighbour(s) Classifiers

class probability estimates
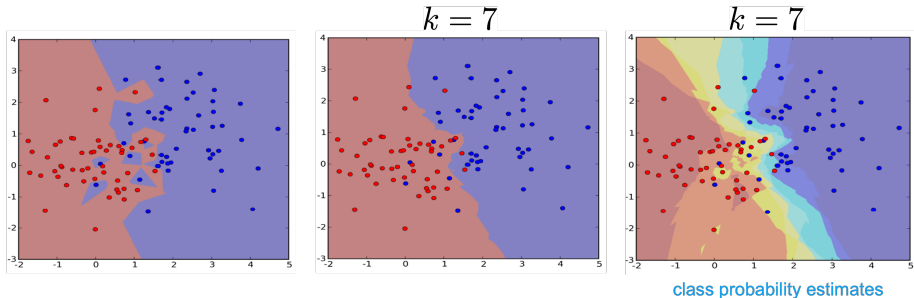
Kernels

- A kernel is a similarity function between two points that is **symmetric** and **positive semi-definite**, denote by:

$$\kappa(x_i, x_j) \in \mathbb{R}$$

- Given dataset $\mathcal{D} = \{(x_i, y_i), \ i = 1, ..., N\}$, the Gram matrix **K** is the $N \times N$ matrix defined as:

$$K_{i,j} = \kappa(x_i, x_j)$$

- **Symmetric:**

$$\kappa(x_i, x_j) = \kappa(x_j, x_i)$$

- **Positive definite:** for all non-zero **v**

$$\mathbf{v}K\mathbf{v}^T \geq 0$$

Kernels

- **Mercer's Theorem**: for any kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists some feature mapping $\phi : \mathcal{X} \to \mathcal{H}$, s.t.:

$$\kappa(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{H}}$$

- A kernel corresponds to a mapping into an **implicit** feature space.
- The kernel is equivalent to an inner product in that feature space.
- Kernel trick: take a feature-based algorithm (SVM, perceptron, logistic regression) and replace all explicit feature computations by **kernel evaluations**:

$$w_y^T \phi(x) = \sum_{i=1}^{N} \sum_{y \in \mathcal{Y}} \alpha_{i,y} \kappa(x, x_i) \quad \text{for some } \alpha_{i,y} \in \mathbb{R}$$

- Extremely popular idea in the 1990-2000s!

Kernels = Tractable Non-Linearity

- A linear classifier in a higher dimensional feature space is a non-linear classifier in the original space
- Computing a non-linear kernel is often cheaper than calculating the corresponding inner product in the high-dimensional feature space.
- Many models can be "kernelized".
- Drawback: quadratic dependency on dataset size
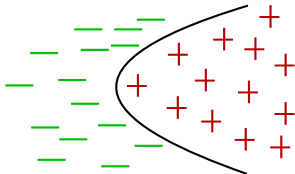
Linear Classifiers in High Dimension
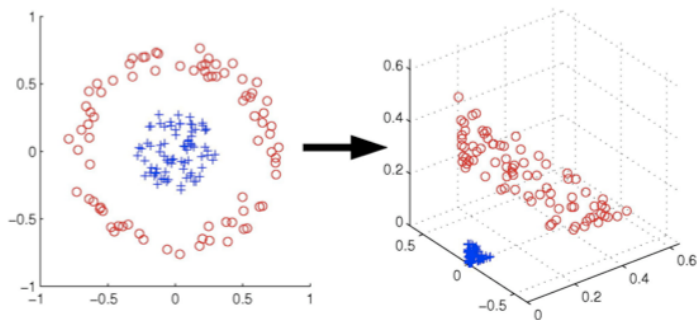
- Example: $\phi([x_1, x_2]) = [x_1^2, x_2^2, \sqrt{2}\, x_1 \, x_2]$
- Kernel trick; let $x = [x_1, x_2]$ and $x' = [x_1', x_2']$;

$$\langle \phi(x), \phi(x') \rangle = \langle [x_1, x_2], [x_1', x_2'] \rangle^2 = k(x, x')$$

Popular Kernels

- Polynomial kernel of order d:

$$\kappa(x, x') = \langle \phi(x), \phi(x') \rangle + 1)^d$$

- Gaussian radial basis kernel

$$\kappa(x, x') = exp(\frac{-\|\phi(x) - \phi(x')\|^2}{2\sigma^2})$$

- String kernels (Lodhi et al., 2002; Collins and Duffy, 2002)

- Tree kernels (Collins and Duffy, 2002)

Conclusions

- Linear classifiers are a broad class including well-known methods such as perceptrons, logistic regression, support vector machines.

- They all involve manipulating weights and features.

- They either lead to closed-form solutions or convex optimization problems (no local minima)

- Stochastic gradient descent algorithms are useful if training datasets are large.

- However, they require manual specification of feature representations

References I

Collins, M. and Duffy, N. (2002). Convolution kernels for natural language. *Advances in Neural Information Processing Systems*, 1:625–632.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444.

Martins, A. F. T. and Astudillo, R. (2016). From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. In *Proc. of the International Conference on Machine Learning*.

Minsky, M. and Papert, S. (1969). Perceptrons.

Lecture 4: Neural Networks I

André Martins, Francisco Melo, Mário Figueiredo

Deep Learning Course, Winter 2022-2023

Today's Roadmap

Today's lecture is about neural networks:

- From the perceptron to the multi-layer perceptron

- Feed-forward neural networks

- Activation functions: sigmoid, tanh, relu, ...

- Activation maps: softmax, sparsemax, ...

- Optimization: non-convexity and local minima

- Universal approximation theorem

- Gradient backpropagation

Biological Neuron
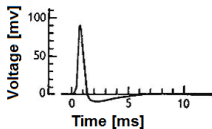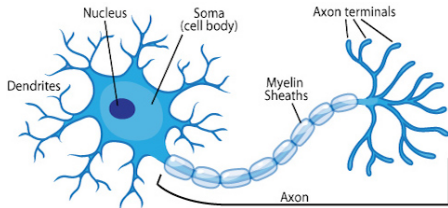
- Three main parts: main body (soma), dendrites, and axon
- Receives input signals from the dendrites, and outputs its signals through the axon
- Axons connect to the dendrites of other neurons; the connections are called synapses
- Generate sharp electrical potentials across their cell membrane (spikes), the main signalling unit of the nervous system

Word of Caution

- Artificial neurons are inspired by biological neurons, but...
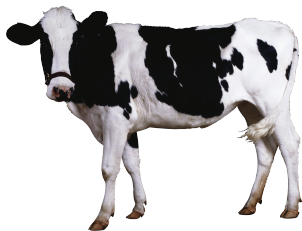
...beware the spherical cow

`https://en.wikipedia.org/wiki/Spherical_cow`

Warren McCulloch and Walter Pitts
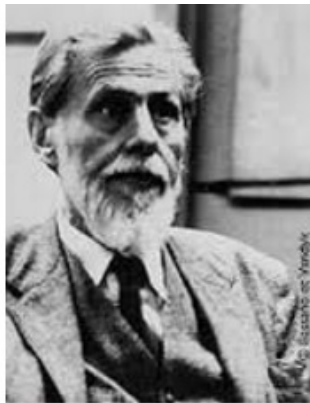
- Earliest computational neuron model, via threshold logic (McCulloch and Pitts, 1943).

Artificial Neuron (McCulloch and Pitts, 1943)
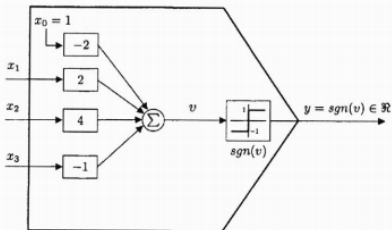
Figure 3.6 Example 3.2: a threshold neural logic for $y = x_2(x_1 + \overline{x}_3)$.

Table 3.6 Truth table for Example 3.2

Neural Inputs			$v = \boldsymbol{w}_a^T \boldsymbol{x}_a$ $= -2 + 2x_1 + 4x_2 - x_3$	$y = sgn(v)$ $= sgn(\boldsymbol{w}_a^T \boldsymbol{x}_a)$
x_1	x_2	x_3		
-1	-1	-1	-7	-1
-1	-1	1	-9	-1
-1	1	-1	1	1
-1	1	1	-1	-1
1	-1	-1	-3	-1
1	-1	1	-5	-1
1	1	-1	5	1
1	1	1	3	1

- Later models replaced the hard threshold by more general activations

Artificial Neuron

- **Pre-activation** (input activation):

$$z(x) = w^T x + b = \sum_{i=1}^{D} w_i x_i + b,$$

  where w are the connection weights and b is a bias term.

- **Activation:**

$$h(x) = g(z(x)) = g(w^T x + b),$$

  where $g : \mathbb{R} \to \mathbb{R}$ is the activation function.
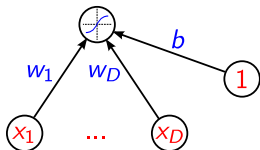
Activation Function

Typical choices:

- Linear
- Sigmoid (logistic function)
- Hyperbolic Tangent
- Rectified Linear

Later:

- Softmax
- Sparsemax
- Max-pooling

Linear Activation

$$g(z) = z$$
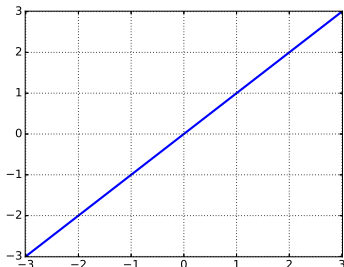
- No "squashing" of the input

- Composing layers of linear units is equivalent to a single linear layer: no expressive power increase by using multiple layers (more later)

- Still useful to linear-project the input to a lower dimension

Sigmoid Activation

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

$$\text{logistic}(u) = \frac{\exp(u)}{1 + \exp(u)}$$
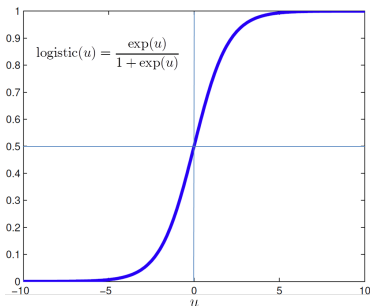
- "Squashes" the neuron pre-activation between 0 and 1

- The output can be interpreted as a probability

- Positive, bounded, strictly increasing

- Logistic regression corresponds to a network with a single sigmoid unit

- Combining layers of sigmoid units increases expressiveness (more later)

Hyperbolic Tangent Activation

$$g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$
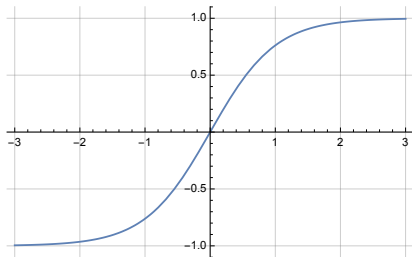
- "Squashes" the neuron pre-activation between -1 and 1
- Related to the sigmoid via $\sigma(z) = \frac{1 + \tanh(z/2)}{2}$
- Bounded, strictly increasing
- Combining layers of tanh units increases expressiveness (more later)

Rectified Linear Unit Activation (Glorot et al., 2011)

$$\boxed{g(z) = \mathbf{relu}(z) = \max\{0, z\}}$$
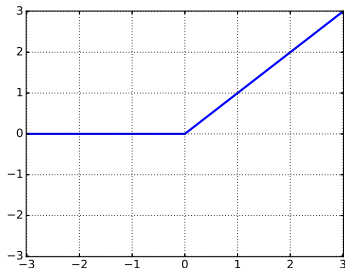
- Non-negative, increasing, but not upper bounded
- Not differentiable at 0
- Leads to neurons with sparse activities (closer to biology)
- Historically, the first activation that allowed training deep nets without unsupervised pre-training (Glorot et al., 2011)
- Less prone to vanishing gradients (more later)

Capacity of Single Neuron (Linear Classifier)

- With a single sigmoid activated neuron we recover logistic regression:

$$p(y = 1|x) = \sigma(w^T x + b).$$

- Can solve linearly separable problems (OR, AND)

- Can't solve non-linearly separable problems (XOR)—unless input is transformed into a better representation
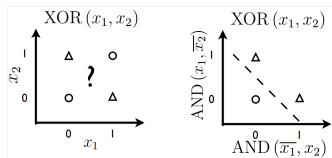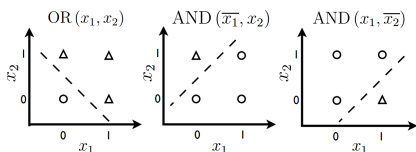
(Slide credit: Hugo Larochelle)

The XOR Affair

Minsky and Papert (1969):

- Misunderstood by many as showing a single perceptron cannot learn XOR (in fact, this was already well-known at the time)
- Fostered an "AI winter" period

Solving XOR with Multi-Layer Perceptron
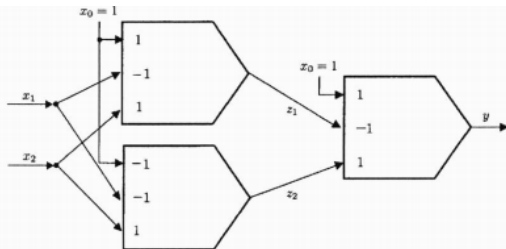
Figure 3.11 Example 3.6: a threshold network for *XOR* function $y = x_1 \oplus x_2 = sgn(1 - z_1 + z_2)$, $z_1 = sgn(1 - x_1 + x_2)$, $z_2 = sgn(-1 - x_1 + x_2)$.

Solving XOR with Multi-Layer Perceptron
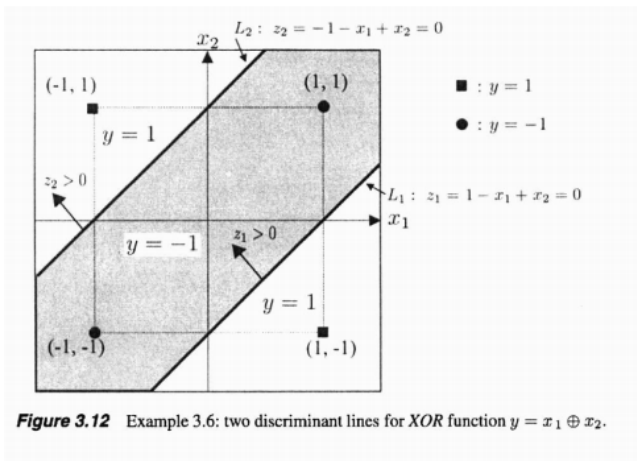
Figure 3.12 Example 3.6: two discriminant lines for *XOR* function $y = x_1 \oplus x_2$.

Solving XOR with Multi-Layer Perceptron

- This construction was known even by McCulloch and Pitts

- The result by Minsky and Papert (1969) is: to implement arbitrary logic functions, all hidden units needs to be connected to all inputs

- At the time, there was hope that only "local" neurons were needed

Multi-Layer Neural Network

- **Key idea:** use intermediate layers between the input and output layers.

- Each hidden layer computes a representation of the input and propagates it forward.

- This increases the expressive power of the network, yielding more complex, non-linear, functions/classifiers

- Also called feed-forward neural network

Single Hidden Layer

Let's start simple: our task involves several inputs ($\boldsymbol{x} \in \mathbb{R}^D$) but a single output (e.g. $y \in \mathbb{R}$ or $y \in \{0, 1\}$)

Include an intermediate layer of K hidden units ($\boldsymbol{h} \in \mathbb{R}^K$)
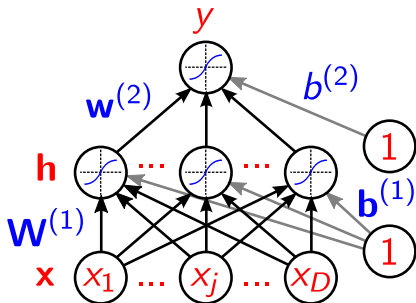
Single Hidden Layer

Assume D inputs ($\boldsymbol{x} \in \mathbb{R}^D$) and K hidden units ($\boldsymbol{h} \in \mathbb{R}^K$)

- **Hidden layer pre-activation:**

$$\boldsymbol{z}(\boldsymbol{x}) = \boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)},$$

  with $\boldsymbol{W}^{(1)} \in \mathbb{R}^{K \times D}$ and $\boldsymbol{b}^{(1)} \in \mathbb{R}^K$.

- **Hidden layer activation:**

$$\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{z}(\boldsymbol{x})),$$

  where $\boldsymbol{g} : \mathbb{R}^K \to \mathbb{R}^K$ is applied
  component-wise
  (component-by-component).

- **Output layer activation:** $f(\boldsymbol{x}) = o(\boldsymbol{h}(\boldsymbol{x})^T \boldsymbol{w}^{(2)} + b^{(2)})$, where
  $\boldsymbol{w}^{(2)} \in \mathbb{R}^K$ and $o : \mathbb{R} \to \mathbb{R}$ if the output activation function.
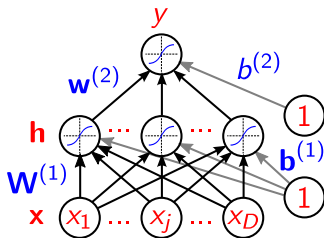
Single Hidden Layer, Single Output

Overall,

$$
\begin{aligned}
f(\boldsymbol{x}) &= o(\boldsymbol{h}(\boldsymbol{x})^T \boldsymbol{w}^{(2)} + b^{(2)}) \\
&= o({\boldsymbol{w}^{(2)}}^T \boldsymbol{g}(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) + b^{(2)})
\end{aligned}
$$

Examples:

- $o(u) = u$, for regression ($y \in \mathbb{R}$)
- $o(u) = \sigma(u)$ for binary classification ($y \in \{\pm 1\}$, $f(\boldsymbol{x}) = P(y = 1 \mid \boldsymbol{x})$)

Non-linear in $\boldsymbol{x}$ and non-linear in $\boldsymbol{W}^{(1)}$ and $\boldsymbol{b}^{(1)}$

$\boldsymbol{h}(\boldsymbol{x})$ is an internal representation (not manually engineered features)

Multiple Classes

Can we use a similar strategy in the multi-class case?

For multi-class classification, we need multiple output units (one per class)

Each output estimates the conditional probability $P(y = c \mid \boldsymbol{x})$

Predicted class is (usually) the one with highest estimated probability

We have already seen an activation function suitable for this: softmax

Softmax Activation

Let $\Delta_{C-1} \subseteq \mathbb{R}^C$ be the probability simplex:

$$\Delta_{C-1} = \{(p1, ..., p_C) : \ p_i \geq 0, i = 1, ..., C, \ \sum_{i=1}^{C} p_i = 1\}$$

Typical activation function for multi-class: **softmax** $: \mathbb{R}^C \to \Delta_{C-1}$:

$$\boxed{o(z) = \textbf{softmax}(z)) = \left[\frac{\exp(z_1)}{\sum_c \exp(z_c)}, \ldots, \frac{\exp(z_C)}{\sum_c \exp(z_c)}\right]}$$

- We saw this previously, about logistic regression!

- Strictly positive, sums to 1

- Resulting distribution has full support: $\textbf{softmax}(z) > \mathbf{0}, \forall z$

Multiple ($L \geq 1$) Hidden Layers

- **Hidden layer pre-activation** (define $\boldsymbol{x}^{(0)} = \boldsymbol{x}$ for convenience):

$$\boldsymbol{z}^{(\ell)}(\boldsymbol{x}) = \boldsymbol{W}^{(\ell)}\boldsymbol{h}^{(\ell-1)}(\boldsymbol{x}) + \boldsymbol{b}^{(1)},$$

with $\boldsymbol{W}^{(\ell)} \in \mathbb{R}^{K_\ell \times K_{\ell-1}}$ and $\boldsymbol{b}^{(\ell)} \in \mathbb{R}^{K_\ell}$.

- **Hidden layer activation:**

$$\boldsymbol{h}^{(\ell)}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{z}^{(\ell)}(\boldsymbol{x})).$$

- **Output layer activation:**

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{o}(\boldsymbol{z}^{(L+1)}(\boldsymbol{x})) = \boldsymbol{o}(\boldsymbol{W}^{(L+1)}\boldsymbol{h}^{(L)}(\boldsymbol{x}) + \boldsymbol{b}^{(L+1)}).$$
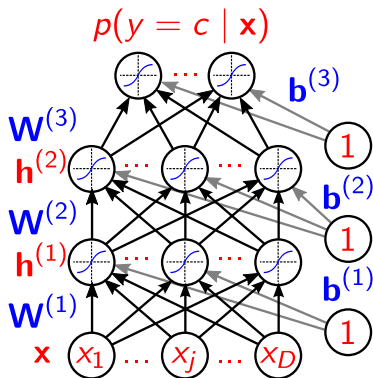
Universal Approximation Theorem

Theorem (Hornik et al. (1989))

An NN with one hidden layer and a linear output can approximate arbitrarily well any continuous function, given enough hidden units.

- First proved for the sigmoid case by Cybenko (1989), then to **tanh** and many other activation functions by Hornik et al. (1989)

- **Caveat:** may need exponentially many hidden units

Deeper Networks

- Deeper networks (more layers) can provide more compact approximations

Theorem (Montufar et al. (2014))

The number of linear regions carved out by a deep neural network with D inputs, depth L, and K hidden units per layer with ReLU activations is

$$O\left(\binom{K}{D}^{D(L-1)} K^D\right)$$

Therefore, for fixed K, deeper networks are exponentially more expressive

"Simple" Target Function, One Hidden Layer
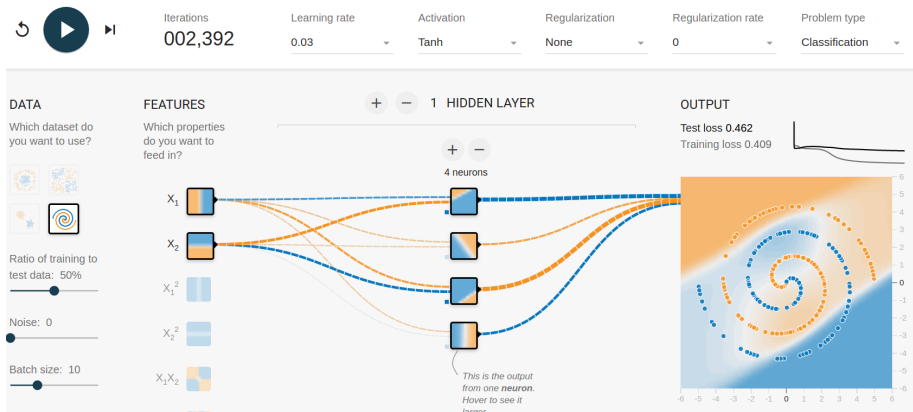
(http://playground.tensorflow.org)

Complex Target Function, One Hidden Layer

(http://playground.tensorflow.org)

Complex Target Function, Two Hidden Layers

(http://playground.tensorflow.org)

Complex Target Function, Two Hidden Layers, ReLU

(http://playground.tensorflow.org)
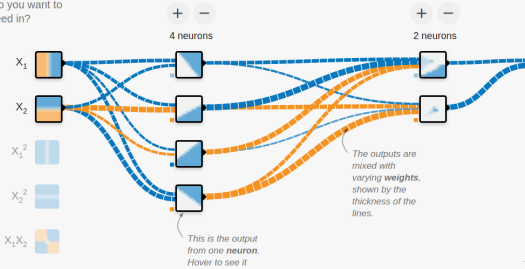
Complex Target Function, Four Hidden Layers, ReLU

(http://playground.tensorflow.org)
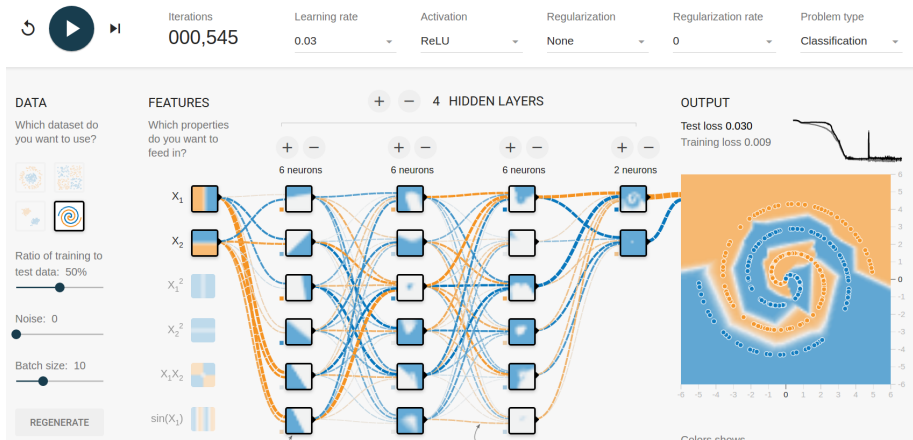
Capacity of Neural Networks

Neural networks are excellent function approximators!

The universal approximation theorem is an important result, but:

- We need a learning algorithm to find the parameter values

- ... and, to generalize, we need to control overfitting

Training Neural Networks

Neural networks are expressive: in theory, they approximate any function

But to do so, their parameters

$$\boldsymbol{\theta} := \{(\boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}), ..., (\boldsymbol{W}^{(L+1)}, \boldsymbol{b}^{(L+1)})\}$$

need to be set accordingly

Key idea: learn these parameters from data

In other words: learn a function by sampling a few points and their values

(We've seen this when we talked about linear models a few days ago...)

Empirical Risk Minimization

Goal: choose parameters $\boldsymbol{\theta} := \{(\boldsymbol{W}^{(\ell)}, \boldsymbol{b}^{(\ell)})\}_{\ell=1}^{L+1}$ that minimize the following objective function:

$$\mathcal{L}(\boldsymbol{\theta}) := \lambda \Omega(\boldsymbol{\theta}) + \frac{1}{N} \sum_{i=1}^{N} L(\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i)$$

- $\{(\boldsymbol{x}_i, y_i), \ i = 1, ..., N\}$ is a training set

- $L(\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i)$ is a loss function

- $\Omega(\boldsymbol{\theta})$ is a regularizer

- λ is a regularization constant (an hyperparameter that needs to be tuned)
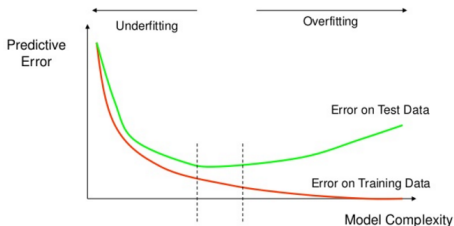
Recap: Gradient Descent

We can write the objective as:

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}) &:= \lambda\Omega(\boldsymbol{\theta}) + \frac{1}{N}\sum_{i=1}^{N} L(\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i) \\
&:= \frac{1}{N}\sum_{i=1}^{N} \underbrace{\lambda\Omega(\boldsymbol{\theta}) + L(\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i)}_{\mathcal{L}_i(\boldsymbol{\theta})} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_i(\boldsymbol{\theta})
\end{aligned}
$$

The gradient is:

$$
\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) := \frac{1}{N}\sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}}\mathcal{L}_i(\boldsymbol{\theta})
$$

Requires a full pass over the data to update the weights: too slow!

Recap: Stochastic Gradient Descent

Sample a single training example uniformly at random: $j \in \{1, ..., N\}$

This way we get a noisy but unbiased estimate of the gradient:

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \quad &:= \quad \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \mathcal{L}_i(\boldsymbol{\theta}) \ \approx \ \nabla_{\boldsymbol{\theta}} \mathcal{L}_j(\boldsymbol{\theta}) \\
&= \quad \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}_j; \boldsymbol{\theta}), y_j).
\end{aligned}
$$

The weights $\boldsymbol{\theta} = \{(\boldsymbol{W}^{(\ell)}, \boldsymbol{b}^{(\ell)})\}_{\ell=1}^{L+1}$ are then updated as:

$$
\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}_j(\boldsymbol{\theta})
$$

In practice, use mini-batch instead of a single sample

Stochastic Gradient Descent with Mini-Batches

With a mini-batch $\{j_1, \ldots, j_B\}$ ($B \ll N$) we get a less noisy, still unbiased estimate of the gradient:

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \; &:= \; \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \mathcal{L}_i(\boldsymbol{\theta}) \approx \frac{1}{B} \sum_{i=1}^{B} \nabla_{\boldsymbol{\theta}} \mathcal{L}_{j_i}(\boldsymbol{\theta}) \\
&= \; \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta}) + \frac{1}{B} \sum_{i=1}^{B} \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}_{j_i}; \boldsymbol{\theta}), y_{j_i}).
\end{aligned}
$$

The weights $\boldsymbol{\theta} = \{(\boldsymbol{W}^{(\ell)}, \boldsymbol{b}^{(\ell)})\}_{\ell=1}^{L+1}$ are then updated as:

$$
\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{1}{B} \sum_{i=1}^{B} \nabla_{\boldsymbol{\theta}} \mathcal{L}_{j_i}(\boldsymbol{\theta})
$$

The Key Ingredients of SGD

In sum, SGD needs the following ingredients:

- The loss function $L(f(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i)$;

- A procedure for computing the gradients $\nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i)$;

- The regularizer $\Omega(\boldsymbol{\theta})$ and its gradient.

Let's see them one at the time...

Loss Function

Should match as much as possible what we want to optimize at test time

Should be well-behaved (continuous, hopefully smooth) to be amenable to optimization (this rules out the $0/1$ loss)

Examples:

- Squared loss for regression

- Negative log-likelihood (cross-entropy) for multi-class classification

- Sparsemax loss for multi-class and multi-label classification

Squared Loss

- The common choice for regression/reconstruction problems

- The neural network aims at estimating $\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}) \approx \boldsymbol{y}$

- Minimize the mean squared error:

$$L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{y}) = \frac{1}{2}\|\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}) - \boldsymbol{y}\|^2$$

- Loss gradient:

$$\frac{\partial L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{y}))}{\partial f_c(\boldsymbol{x}; \boldsymbol{\theta})} = f_c(\boldsymbol{x}; \boldsymbol{\theta}) - y_c$$

Negative Log-Likelihood (Cross-Entropy)

- The common choice for a softmax output layer

- The neural network estimates $f_c(\boldsymbol{x}; \boldsymbol{\theta}) \approx P(y = c \mid \boldsymbol{x})$

- We minimize the negative log-likelihood (also called cross-entropy):

$$
\begin{aligned}
L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y) &= -\sum_c 1_{(y=c)} \log f_c(\boldsymbol{x}; \boldsymbol{\theta}) \\
&= -\log f_y(\boldsymbol{x}; \boldsymbol{\theta}) \\
&= -\log \mathbf{softmax}_y(\boldsymbol{z}(\boldsymbol{x}))
\end{aligned}
$$

  where $\boldsymbol{z}$ is the output pre-activation.

- Loss gradient at output pre-activation:

$$
\frac{\partial L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}, y))}{\partial z_c} = \mathbf{softmax}_c(\boldsymbol{z}(\boldsymbol{x})) - 1_{(y=c)}
$$

Classification Losses

- Let the correct label be $y = 1$ and define $s = z_2 - z_1$:
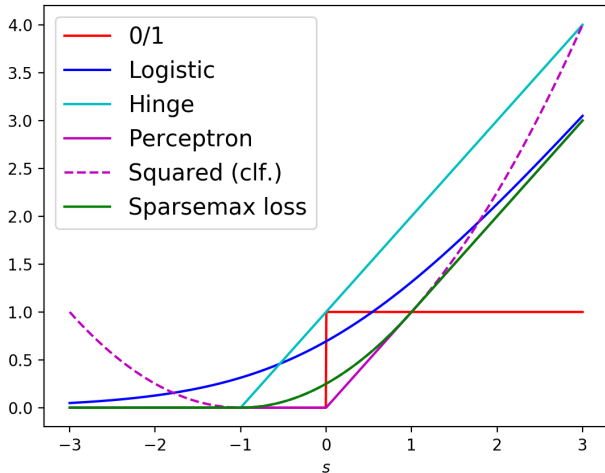
The Key Ingredients of SGD

In sum, SGD needs the following ingredients:

- The loss function $L(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i)$; ✓

- A procedure for computing the gradients $\nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i)$: next

- The regularizer $\Omega(\boldsymbol{\theta})$ and its gradient.

Gradient Computation

- Recall that we need to compute

$$\nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i)$$

  for $\boldsymbol{\theta} = \{(\boldsymbol{W}^{(\ell)}, \boldsymbol{b}^{(\ell)})\}_{\ell=1}^{L+1}$ (the weights and biases at all layers)

- This will be done with the gradient backpropagation algorithm

- **Key idea:** use the chain rule for derivatives!

$$h(x) = f\big(g(x)\big) \quad \Rightarrow \quad h'(x) = f'(g(x))\, g'(x)$$

Recap: Chain Rule
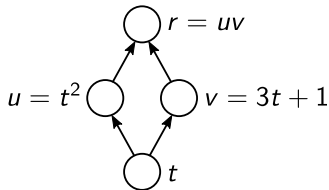
$$\frac{\partial r(t)}{\partial t} = ?\frac{\partial r(u)}{\partial u}\frac{\partial u(t)}{\partial t} + \frac{\partial r(v)}{\partial v}\frac{\partial v(t)}{\partial t}$$
$$= 2tv + 3u$$
$$= 2t(3t+1) + 3t^2 = 9t^2 + 2t.$$

- If a function $r(t)$ can be written as a function of intermediate results $q_i(t)$, then we have:

$$\frac{\partial r(t)}{\partial t} = \sum_i \frac{\partial r(t)}{\partial q_i(t)}\frac{\partial q_i(t)}{\partial t}$$

- We can invoke it by setting t to a output unit in a layer; $q_i(t)$ to the pre-activation in the layer above; and $r(t)$ to the loss function.

Hidden Layer Gradient

Main message: gradient backpropagation is just the chain rule of derivatives!

Hidden Layer Gradient

(Recap: $\boldsymbol{z}^{(\ell+1)} = \boldsymbol{W}^{(\ell+1)} \boldsymbol{h}^{(\ell)} + \boldsymbol{b}^{(\ell+1)}$)

$$\frac{\partial L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y)}{\partial h_j^{(\ell)}} = \sum_i \frac{\partial L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y)}{\partial z_i^{(\ell+1)}} \frac{\partial z_i^{(\ell+1)}}{\partial h_j^{(\ell)}}$$

$$= \sum_i \frac{\partial L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y)}{\partial z_i^{(\ell+1)}} \boldsymbol{W}_{i,j}^{(\ell+1)}$$
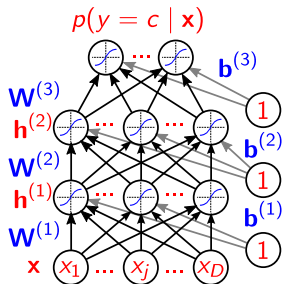
Hence $\nabla_{\boldsymbol{h}^{(\ell)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y) = \boldsymbol{W}^{(\ell+1)^\top} \nabla_{\boldsymbol{z}^{(\ell+1)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y)$.

Hidden Layer Gradient (Before Activation)

(Recap: $h_j^{(\ell)} = g(z_j^{(\ell)})$, where $g : \mathbb{R} \to \mathbb{R}$ is the activation function)

$$
\begin{aligned}
\frac{\partial L(\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}), y)}{\partial z_j^{(\ell)}} &= \frac{\partial L(\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}), y)}{\partial h_j^{(\ell)}} \frac{\partial h_j^{(\ell)}}{\partial z_j^{(\ell)}} \\
&= \frac{\partial L(\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}), y)}{\partial h_j^{(\ell)}} g'(z_j^{(\ell)})
\end{aligned}
$$
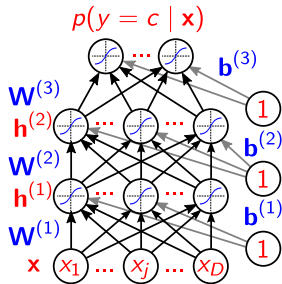
Hence $\nabla_{\boldsymbol{z}^{(\ell)}} L(\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}), y) = \nabla_{\boldsymbol{h}^{(\ell)}} L(\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}), y) \odot \boldsymbol{g}'(\boldsymbol{z}^{(\ell)})$.

How to compute the derivative of the activation function $\boldsymbol{g}'(\boldsymbol{z}^{(\ell)})$?

Linear Activation

$$g(z) = z$$

Derivative:

$$g'(z) = 1$$
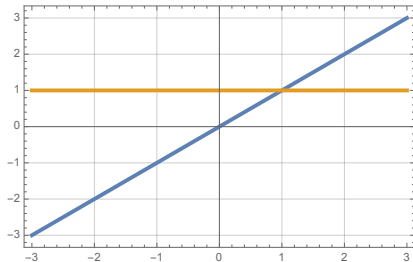
Sigmoid Activation

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

Derivative:

$$g'(z) = g(z)(1 - g(z))$$
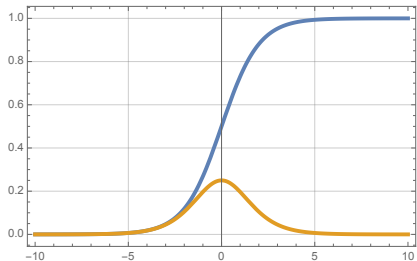
Hyperbolic Tangent Activation

$$g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Derivative:

$$g'(z) = 1 - g(z)^2 = \text{sech}^2(x)$$
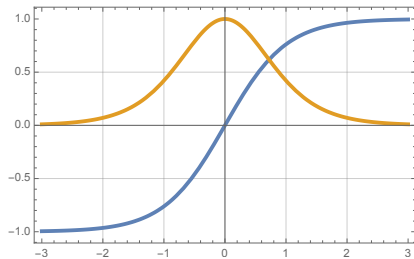
Rectified Linear Unit Activation (Glorot et al., 2011)

$$g(z) = \textbf{relu}(z) = \max\{0, z\}$$

Derivative (except for $z = 0$):

$$g'(z) = 1_{z>0}$$
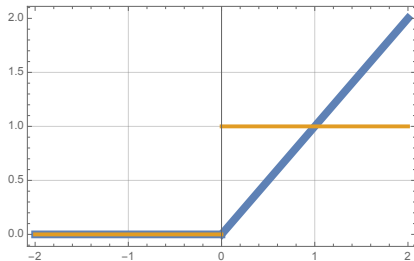
Parameter Gradient

(Recap: $z^{(\ell)} = W^{(\ell)} h^{(\ell-1)} + b^{(\ell)}$)

$$
\begin{aligned}
\frac{\partial L(f(x; \theta), y)}{\partial W_{i,j}^{(\ell)}} &= \frac{\partial L(f(x; \theta), y)}{\partial z_i^{(\ell)}} \frac{\partial z_i^{(\ell)}}{\partial W_{i,j}^{(\ell)}} \\
&= \frac{\partial L(f(x; \theta), y)}{\partial z_i^{(\ell)}} h_j^{(\ell-1)}
\end{aligned}
$$
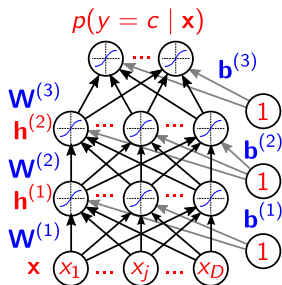
Hence $\nabla_{W^{(\ell)}} L(f(x; \theta), y) = \nabla_{z^{(\ell)}} L(f(x; \theta), y) h^{(\ell-1)\top}$

Similarly, $\nabla_{b^{(\ell)}} L(f(x; \theta), y) = \nabla_{z^{(\ell)}} L(f(x; \theta), y)$

Backpropagation

Compute output gradient (before activation):

$$\nabla_{\boldsymbol{z}^{(L+1)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y) = -(\mathbf{1}_y - \boldsymbol{f}(\boldsymbol{x}))$$

for ℓ from $L+1$ to 1 **do**

Compute gradients of hidden layer parameters:

$$\begin{aligned}
\nabla_{\boldsymbol{W}^{(\ell)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y) &= \nabla_{\boldsymbol{z}^{(\ell)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y) \, \boldsymbol{h}^{(\ell-1)^{\top}} \\
\nabla_{\boldsymbol{b}^{(\ell)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y) &= \nabla_{\boldsymbol{z}^{(\ell)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y)
\end{aligned}$$

Compute gradient of hidden layer below:

$$\nabla_{\boldsymbol{h}^{(\ell-1)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y) = \boldsymbol{W}^{(\ell)^{\top}} \nabla_{\boldsymbol{z}^{(\ell)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y)$$

Compute gradient of hidden layer below (before activation):

$$\nabla_{\boldsymbol{z}^{(\ell-1)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y) = \nabla_{\boldsymbol{h}^{(\ell-1)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y) \odot \boldsymbol{g}'(\boldsymbol{z}^{(\ell-1)})$$

end for

Conclusions

- Multi-layer perceptrons are universal function approximators
- However, they need to be trained
- Stochastic gradient descent is an effective training algorithm
- This is possible with the gradient backpropagation algorithm (an application of the chain rule of derivatives)

Next class:

- Most current software packages represent a computation graph and implement automatic differentiation
- Dropout regularization is effective to avoid overfitting
- Tricks of the trade

Thank you!

Questions?

References I

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Minsky, M. and Papert, S. (1969). Perceptrons.

Montufar, G. F., Pascanu, R., Cho, K., and Bengio, Y. (2014). On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2924–2932.

Lecture 5: Neural Networks II

André Martins, Francisco Melo, Mário Figueiredo

Deep Learning Course, Winter 2022-2023

Today's Roadmap

Last lecture was about neural networks:

- From perceptron to multi-layer perceptron
- Feed-forward neural networks
- Activation funcions: sigmoid, tanh, relu, ...
- Activation maps: softmax, sparsemax, ...
- Non-convex optimization and local minima
- Universal approximation theorem
- Gradient backpropagation

Today: autodiff, regularization, tricks of the trade.

Recap: Forward Propagation

Now assume $L \geq 1$ hidden layers:

- **Hidden layer pre-activation** (define $h^{(0)} = x$, for convenience):

$$z^{(\ell)}(x) = W^{(\ell)}h^{(\ell-1)}(x) + b^{(1)},$$

with $W^{(\ell)} \in \mathbb{R}^{K_\ell \times K_{\ell-1}}$ and $b^{(\ell)} \in \mathbb{R}^{K_\ell}$.

- **Hidden layer activation:**

$$h^{(\ell)}(x) = g(z^{(\ell)}(x)).$$

- **Output layer activation:**

$$f(x) = o(z^{(L+1)}(x)) = o(W^{(L+1)}h^{(L)} + b^{(L+1)}).$$

Recap: Gradient Backpropagation

Compute output gradient (before activation):

$$\nabla_{\boldsymbol{z}^{(L+1)}} L(\boldsymbol{f}(x;\boldsymbol{\theta}), y) = -(\mathbf{1}_y - \boldsymbol{f}(x))$$

for ℓ from $L+1$ to 1 **do**

Compute gradients of hidden layer parameters:

$$
\begin{aligned}
\nabla_{\boldsymbol{W}^{(\ell)}} L(\boldsymbol{f}(x;\boldsymbol{\theta}), y) &= \nabla_{\boldsymbol{z}^{(\ell)}} L(\boldsymbol{f}(x;\boldsymbol{\theta}), y) \, \boldsymbol{h}^{(\ell-1)^\top} \\
\nabla_{\boldsymbol{b}^{(\ell)}} L(\boldsymbol{f}(x;\boldsymbol{\theta}), y) &= \nabla_{\boldsymbol{z}^{(\ell)}} L(\boldsymbol{f}(x;\boldsymbol{\theta}), y)
\end{aligned}
$$

Compute gradient of previous hidden layer:

$$\nabla_{\boldsymbol{h}^{(\ell-1)}} L(\boldsymbol{f}(x;\boldsymbol{\theta}), y) = \boldsymbol{W}^{(\ell)^\top} \nabla_{\boldsymbol{z}^{(\ell)}} L(\boldsymbol{f}(x;\boldsymbol{\theta}), y)$$

Compute gradient of previous hidden layer (before activation):

$$\nabla_{\boldsymbol{z}^{(\ell-1)}} L(\boldsymbol{f}(x;\boldsymbol{\theta}), y) = \nabla_{\boldsymbol{h}^{(\ell-1)}} L(\boldsymbol{f}(x;\boldsymbol{\theta}), y) \odot \boldsymbol{g}'(\boldsymbol{z}^{(\ell-1)})$$

end for

Computation Graph

- Forward propagation can be represented as a DAG (directed acyclic graph).

- Allows implementing forward propagation in a modular way.

- Each box can be an object with a `fprop` method, which computes the output of the box given its inputs.

- Calling the `fprop` method of each box in the right order yields forward propagation.
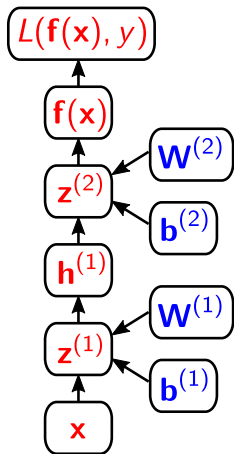
Automatic Differentiation (Autodiff)

- Backpropagation is also implementable in a modular way.

- Each box should have a `bprop` method, which computes the loss gradient w.r.t. its parents, given the loss gradient w.r.t. to the output.

- Can make use of cached computation done during the `fprop` method

- Calling the `bprop` method in reverse order yields backpropagation
  (only needs to reach the parameters)

$L(\mathbf{f}(\mathbf{x}), y)$

$\mathbf{f}(\mathbf{x})$

$\mathbf{z}^{(2)}$ $\mathbf{W}^{(2)}$

$\mathbf{b}^{(2)}$

$\mathbf{h}^{(1)}$

$\mathbf{z}^{(1)}$ $\mathbf{W}^{(1)}$

$\mathbf{b}^{(1)}$

$\mathbf{x}$
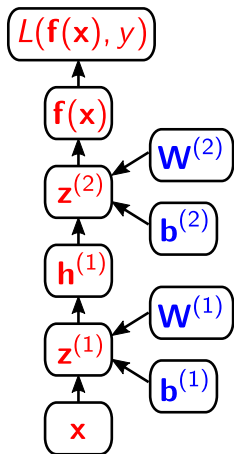
Several Autodiff Strategies

Symbol-to-number differentiation (Caffe, Torch, Pytorch, Dynet, ...)

- Take a computational graph and numerical inputs, returns a set of numerical values describing the gradient at those input values.
- Advantage: simpler to implement and debug.
- Disadvantage: only works for first-order derivatives.

Symbol-to-symbol differentiation (Theano, Tensorflow, ...)

- Take a computational graph and add additional nodes to the graph that provide a symbolic description of the desired derivatives (i.e. the derivatives are just another computational graph)
- Advantage: generalizes automatically to higher-order derivatives
- Disadvantage: harder to implement and to debug

Many Software Toolkits for Neural Networks

- Theano
- Tensorflow
- Torch, Pytorch
- MXNet
- Keras
- Caffe
- DyNet
- ...

All implement automatic differentiation.

Some Theano Code (Logistic Regression)

```python
import numpy
import theano
import theano.tensor as T
rng = numpy.random

N = 400                                    # training sample size
feats = 784                                # number of input variables

# generate a dataset: D = (input_values, target_class)
D = (rng.randn(N, feats), rng.randint(size=N, low=0, high=2))
training_steps = 10000

# Declare Theano symbolic variables
x = T.dmatrix("x")
y = T.dvector("y")

# initialize the weight vector w randomly
#
# this and the following bias variable b
# are shared so they keep their values
# between training iterations (updates)
w = theano.shared(rng.randn(feats), name="w")

# initialize the bias term
b = theano.shared(0., name="b")

print("Initial model:")
print(w.get_value())
print(b.get_value())

# Construct Theano expression graph
p_1 = 1 / (1 + T.exp(-T.dot(x, w) - b))    # Probability that target = 1
prediction = p_1 > 0.5                     # The prediction thresholded
xent = -y * T.log(p_1) - (1-y) * T.log(1-p_1) # Cross-entropy loss function
cost = xent.mean() + 0.01 * (w ** 2).sum() # The cost to minimize
gw, gb = T.grad(cost, [w, b])              # Compute the gradient of the cost
                                           # w.r.t weight vector w and
                                           # bias term b
                                           # (we shall return to this in a
                                           # following section of this tutorial)

# Compile
train = theano.function(
        inputs=[x,y],
```

Some Code in Tensorflow (Linear Regression)

```
import tensorflow as tf
import numpy as np

# Create 100 phony x, y data points in NumPy, y = x * 0.1 + 0.3
x_data = np.random.rand(100).astype(np.float32)
y_data = x_data * 0.1 + 0.3

# Try to find values for W and b that compute y_data = W * x_data + b
# (We know that W should be 0.1 and b 0.3, but TensorFlow will
# figure that out for us.)
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W * x_data + b

# Minimize the mean squared errors.
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

# Before starting, initialize the variables.  We will 'run' this first.
init = tf.global_variables_initializer()

# Launch the graph.
sess = tf.Session()
sess.run(init)

# Fit the line.
for step in range(201):
    sess.run(train)
    if step % 20 == 0:
        print(step, sess.run(W), sess.run(b))

# Learns best fit is W: [0.1], b: [0.3]
```

Some Code in Keras (Multi-Layer Perceptron)

Multilayer Perceptron (MLP) for multi-class softmax classification:

```python
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD

model = Sequential()
# Dense(64) is a fully-connected layer with 64 hidden units.
# in the first layer, you must specify the expected input data shape:
# here, 20-dimensional vectors.
model.add(Dense(64, input_dim=20, init='uniform'))
model.add(Activation('tanh'))
model.add(Dropout(0.5))
model.add(Dense(64, init='uniform'))
model.add(Activation('tanh'))
model.add(Dropout(0.5))
model.add(Dense(10, init='uniform'))
model.add(Activation('softmax'))

sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=['accuracy'])

model.fit(X_train, y_train,
          nb_epoch=20,
          batch_size=16)
score = model.evaluate(X_test, y_test, batch_size=16)
```

Some Code in Pytorch (Multi-Layer Perceptron)

```python
# Fully connected neural network with one hidden layer
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out

model = NeuralNet(input_size, hidden_size, num_classes).to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        # Move tensors to the configured device
        images = images.reshape(-1, 28*28).to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 100 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                    .format(epoch+1, num_epochs, i+1, total_step, loss.item()))
```

Reminder: Key Ingredients of SGD

In sum, we need the following ingredients:

- The loss function $L(f(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i)$ ✓

- A procedure for computing the gradients $\nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i)$ ✓

- The regularizer $\Omega(\boldsymbol{\theta})$ and its gradient: next!

Regularization

Recall that we're minimizing the following objective function:

$$\mathcal{L}(\boldsymbol{\theta}) := \lambda \Omega(\boldsymbol{\theta}) + \frac{1}{N} \sum_{n=1}^{N} L(\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i)$$

It remains to define the regularizer and its gradient

We will study:

- ℓ_2 regularization

- ℓ_1 regularization

- dropout regularization

ℓ_2 Regularization

- The biases $\boldsymbol{b}^{(1)}, ..., \boldsymbol{b}^{(L+1)}$ are not regularized; only the weights:

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2} \sum_{\ell=1}^{L+1} \| \boldsymbol{W}^{(\ell)} \|^2$$

- Equivalent to a Gaussian prior on the weights

- Gradient of this regularizer is: $\nabla_{\boldsymbol{W}^{(\ell)}} \Omega(\boldsymbol{\theta}) = \boldsymbol{W}^{(\ell)}$

- Weight decay effect (as seen in the previous lecture)

$$
\begin{aligned}
\boldsymbol{W}^{(\ell)} &\leftarrow \boldsymbol{W}^{(\ell)} - \eta \nabla_{\boldsymbol{W}^{(\ell)}} \mathcal{L}_i(\boldsymbol{\theta}) \\
&= \boldsymbol{W}^{(\ell)} - \eta(\lambda \nabla_{\boldsymbol{W}^{(\ell)}} \Omega(\boldsymbol{\theta}) + \nabla_{\boldsymbol{W}^{(\ell)}} L(f(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i)) \\
&= \underbrace{(1 - \eta\lambda)}_{<1} \boldsymbol{W}^{(\ell)} - \eta \nabla_{\boldsymbol{W}^{(\ell)}} L(f(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i)
\end{aligned}
$$

ℓ_1 Regularization

- The biases $\boldsymbol{b}^{(1)}, ..., \boldsymbol{b}^{(L+1)}$ are not regularized; only the weights:

$$\Omega(\boldsymbol{\theta}) = \sum_{\ell} \|\boldsymbol{W}^{(\ell)}\|_1 = \sum_{\ell} \sum_{ij} |W_{ij}^{(\ell)}|$$

- Equivalent to Laplacian prior on the weights

- Gradient is: $\nabla_{\boldsymbol{W}^{(\ell)}} \Omega(\boldsymbol{\theta}) = \text{sign}(\boldsymbol{W}^{(\ell)})$

- Promotes sparsity of the weights

Dropout Regularization (Srivastava et al., 2014)
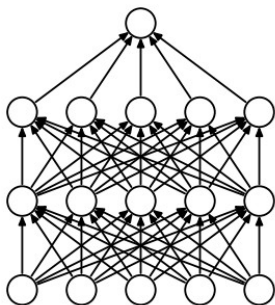
(a) Standard Neural Net
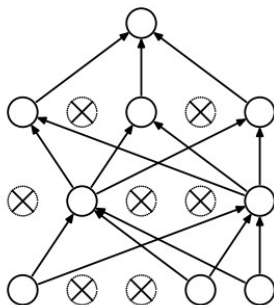
(b) After applying dropout.

During training, remove some hidden units, chosen at random

Dropout Regularization (Srivastava et al., 2014)

- Each hidden unit output is set to 0 with probability p (e.g. $p = 0.5$)

- This prevents hidden units to co-adapt to other units, forcing them to be more generally useful

- At test time, keep all units, with the outputs multiplied by $1 - p$

- Shown to be a form of adaptive regularization (Wager et al., 2013)

- Many software packages implement another variant, inverted dropout, where at training time the output of the units that were not dropped is divided by $1 - p$ and requires no change at test time

Implementation of Dropout

- Usually implemented using random binary masks

- The hidden layer activations become

$$\boldsymbol{h}^{(\ell)}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{z}^{(\ell)}(\boldsymbol{x})) \odot \boldsymbol{m}^{(\ell)}$$

- Beats regular backpropagation on many datasets (Hinton et al., 2012)

- Other variants, e.g. DropConnect (Wan et al., 2013), Stochastic Pooling (Zeiler and Fergus, 2013)

Backpropagation with Dropout

Compute output gradient (before activation):

$$\nabla_{\boldsymbol{z}^{(L+1)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y) = -(\mathbf{1}_y - \boldsymbol{f}(\boldsymbol{x}))$$

for ℓ from $L+1$ to 1 **do**

    Compute gradients of hidden layer parameters:

$$\nabla_{\boldsymbol{W}^{(\ell)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y) = \nabla_{\boldsymbol{z}^{(\ell)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y) \underbrace{\boldsymbol{h}^{(\ell-1)^\top}}_{\text{includes } \boldsymbol{m}^{(\ell-1)}}$$

$$\nabla_{\boldsymbol{b}^{(\ell)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y) = \nabla_{\boldsymbol{z}^{(\ell)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y)$$

    Compute gradient of hidden layer below:

$$\nabla_{\boldsymbol{h}^{(\ell-1)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y) = \boldsymbol{W}^{(\ell)^\top} \nabla_{\boldsymbol{z}^{(\ell)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y)$$

    Compute gradient of hidden layer below (before activation):

$$\nabla_{\boldsymbol{z}^{(\ell-1)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y) = \nabla_{\boldsymbol{h}^{(\ell-1)}} L(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}), y) \odot \boldsymbol{g}'(\boldsymbol{z}^{(\ell-1)}) \odot \boldsymbol{m}^{(\ell-1)}$$

end for

Initialization

- Biases: initialize at zero

- Weights:

    ✓ Cannot initialize to zero with **tanh** activation (gradients would also be all zero and we would be at saddle point)

    ✓ Cannot initialize the weights to the same value (need to break the symmetry)

    ✓ Random initialization (Gaussian, uniform), sampling around 0 to break symmetry

    ✓ For ReLU activations, the mean should be a small positive number

    ✓ Variance cannot be too high, otherwise all neuron activations will be saturated

"Glorot Initialization"

- Recipe from Glorot and Bengio (2010):

$$\boldsymbol{W}_{i,j}^{(\ell)} \sim U[-t, t], \text{ with } t = \frac{\sqrt{6}}{\sqrt{K^{(\ell)} + K^{(\ell-1)}}}$$

- Works well in practice with **tanh** and sigmoid activations

Training, Validation, and Test Sets

Split datasets in training, validation, and test partitions.

- Training set: serves to train the model

- Validation set: used to tune hyperparameters (learning rate, number of hidden units, regularization coefficient, dropout probability, best epoch, etc.)

- Test set: used to estimate the generalization performance

# Hyperparameter Tuning: Grid Search, Random Search

Search for the best configuration of the hyperparameters:

- **Grid search**: specify a set of values to test for each hyperparameter, and try all configurations of these values

- **Random search**: specify a distribution over the values of each hyper-parameter (e.g. uniform in some range) and sample independently each hyper-parameter to get configurations

- **Bayesian optimization** (Snoek et al., 2012)

We can always go back and fine-tune the grid/distributions if necessary

Early Stopping

- To select the number of epochs, stop training when validation error increases (with some look ahead)
- One common strategy (with SGD) is to halve the learning rate for every epoch where the validation error increases
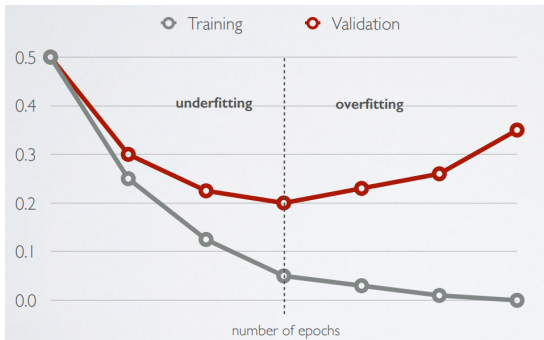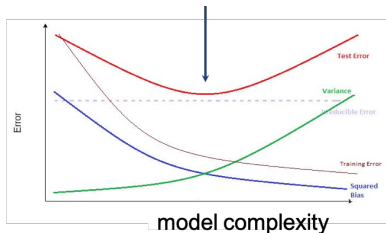

(Image credit: Hugo Larochelle)
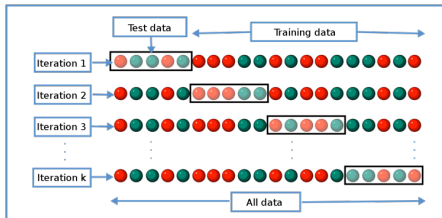
Cross Validation

Model selection

finding the sweet spot (F)

model complexity

validation set:

✓ split data into train and validation susbsets
✓ train on the training subset
✓ test on the validation subset

cross validation (k-fold): repeat k times; average

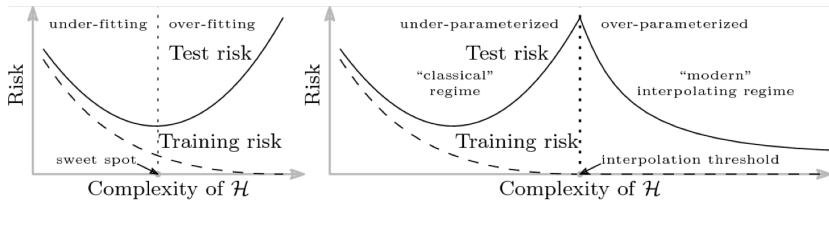# Over-parametrization

The new regime...

modern deep networks have "too many" parameters: they should overfit, ...

...yet, they usually don't.   Why?   Ongoing research.

(illustration by Mikhail Belkin)

Tricks of the Trade

- Normalization of the data

- Decaying the learning rate

- Mini-batches

- Adaptive learning rates

- Gradient checking

- Debugging on a small dataset

Normalization of the Data

- For each input dimension: subtract the training set mean and divide by the training set standard deviation

- It makes each input dimension have zero mean, unit variance

- It can speed up training (in number of epochs)

- Doesn't work for sparse inputs (destroys sparsity)

Decaying the Learning Rate

In SGD, as we get closer to a local minimum, it makes sense to take smaller update steps (to avoid diverging)

- Start with a large learning rate (say 0.1)

- Keep it fixed while validation error keeps improving

- Divide by 2 and go back to the previous step

Mini-Batches

- Instead of updating after a single example, can aggregate a mini-batch of examples (e.g. 50–200 examples) and compute the averaged gradient for the entire mini-batch

- Less noisy than standard SGD

- Can leverage matrix-matrix computations (or tensor computations)

- Large computational speed-ups in GPUs: computation is trivially parallelizable accross the mini-batch and we can exhaust the GPU memory

Adaptive Learning Rates

Instead of using the same step size for all parameters, have one learning rate per parameter

- Adagrad (Duchi et al., 2011): learning rates are scaled by the square root of the cumulative sum of squared gradients

$$\eta^{(t)} = \eta^{(t-1)} + (\nabla_{\boldsymbol{\theta}} L(\boldsymbol{f}(\boldsymbol{x}), y))^2, \qquad \bar{\nabla}_{\boldsymbol{\theta}}^{(t)} = \frac{\nabla_{\boldsymbol{\theta}} L(\boldsymbol{f}(\boldsymbol{x}), y)}{\sqrt{\eta^{(t)} + \epsilon}}$$

- RMSprop (Tieleman and Hinton, 2012): instead of cumulative sum, use exponential moving average

$$\eta^{(t)} = \beta\eta^{(t-1)} + (1-\beta)(\nabla_{\boldsymbol{\theta}} L(\boldsymbol{f}(\boldsymbol{x}), y))^2, \qquad \bar{\nabla}_{\boldsymbol{\theta}}^{(t)} = \frac{\nabla_{\boldsymbol{\theta}} L(\boldsymbol{f}(\boldsymbol{x}), y)}{\sqrt{\eta^{(t)} + \epsilon}}$$

- Adam (Kingma and Ba, 2014): combine RMSProp with momentum

Gradient Checking

- If the training loss is not decreasing even with a very small learning rate, there's likely a bug in the gradient computation

- To debug your implementation of `fprop`/`bprop`, compute the "numeric gradient," a finite difference approximation of the true gradient:

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x + \epsilon) - f(x - \epsilon)}{2\epsilon}$$

Debugging on a Small Dataset

- Extract a small subset of your training set (e.g. 50 examples)

- Monitor your training loss in this set

- You should be able to overfit in this small training set

- If not, see if some units are saturated from the very first iterations (if they are, reduce the initialization variance or properly normalize your inputs)

- If the training error is bouncing up and down, decrease the learning rate

Conclusions

- Multi-layer perceptrons are universal function approximators
- However, they need to be trained
- Stochastic gradient descent is an effective training algorithm
- This is possible with the gradient backpropagation algorithm (an application of the chain rule of derivatives)
- Most current software packages represent a computation graph and implement automatic differentiation
- Dropout regularization is effective to avoid overfitting

Thank you!

Questions?
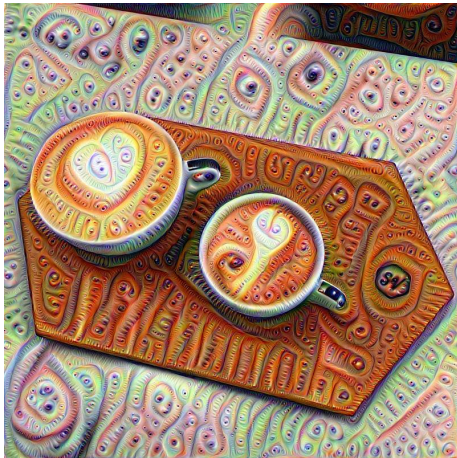
References I

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, volume 9, pages 249–256.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Kingma, D. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. In *Proc. of International Conference on Learning Representations*.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Tieleman, T. and Hinton, G. (2012). Rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2).

Wager, S., Wang, S., and Liang, P. S. (2013). Dropout training as adaptive regularization. In *Advances in neural information processing systems*, pages 351–359.

Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., and Fergus, R. (2013). Regularization of neural networks using dropconnect. In *Proc. of the International Conference on Machine Learning*, pages 1058–1066.

Zeiler, M. D. and Fergus, R. (2013). Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*.

Lecture 6: Representation Learning

André Martins, Francisco Melo, Mário Figueiredo

Deep Learning Course, Winter 2022-2023

Announcements

Deadline for Homework 1 is **Friday, December 23, 23:59**

- Please submit your solutions and code in Fenix.

- No late days allowed!!

- Solutions will be posted the day after.

Today's Roadmap

Today's lecture is about:

- Representation learning.

- Principal component analysis (PCA) and auto-encoders.

- Denoising auto-encoders.

- Distributed representations.

- Word embeddings and negative sampling.

- Multilingual and contextual word embeddings.

Representations

- A key feature of NNs is their ability to learn representations $\phi(\boldsymbol{x})$

- Standard linear models require manually engineered features $\phi(\boldsymbol{x})$

- Representations are useful for several reasons:

  (i) They can make our models more expressive and more accurate

  (ii) They may allow transferring representations from one task to another

- We talked about (i) when discussing the multi-layer perceptron

- In this lecture, we'll focus on (ii)

Hierarchical Compositionality

Key Idea: deep(er) NNs learn **coarse-to-fine** representation layers.

Vision:

- pixels $\rightarrow$ edges $\rightarrow$ textons $\rightarrow$ motifs $\rightarrow$ parts $\rightarrow$ objects $\rightarrow$ scenes

Speech:

- samples $\rightarrow$ spectral bands $\rightarrow$ formants $\rightarrow$ motifs $\rightarrow$ phonemes $\rightarrow$ words

Text:

- characters $\rightarrow$ words $\rightarrow$ phrases $\rightarrow$ sentences $\rightarrow$ stories

(Inspired by Marc'Aurelio Ranzato and Yann LeCun)

Hierarchical Compositionality

Feature visualization of convolutional NNs trained on ImageNet

(Zeiler and Fergus, 2013)
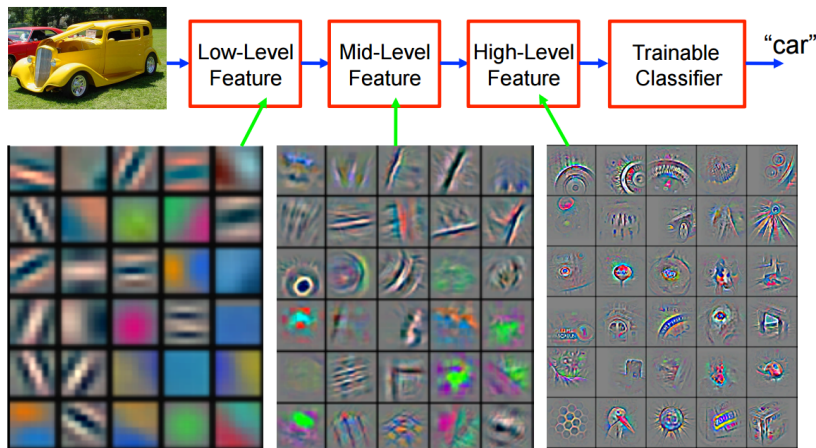
The Mammalian Visual Cortex is Hierarchical

[picture from Simon Thorpe]
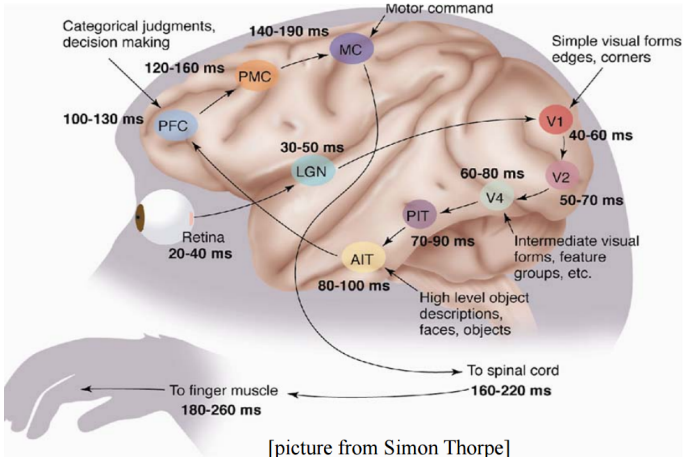
(LGN = lateral geniculate nucleus; PIT = posterior inferotemporal area; AIT = anterior inferotemporal area; PFC = prefrontal cortex; PMC = premotor cortex; MC = motor cortex)

What Is Learned in Each Layer

- Layers closer to inputs learn low-level representations (corners, edges)

- Layers farther away from inputs learn more abstract representations (shapes, forms, objects)

- This holds, not only for images, but also text, sounds, ...

Distributed Representations (Hinton, 1984)

This is a central concept in neural networks.

Key questions:

- How can a NN so effectively represent objects, if it has only a few hidden units (i.e. much fewer than possible objects)?

- What is each hidden unit actually representing?

- How can a NN generalize to objects that is has not seen before?

Local vs Distributed Representations

Consider two alternative representations:

- Local (one-hot) representations (one dimension per object)
- Distributed representations (one dimension per property)
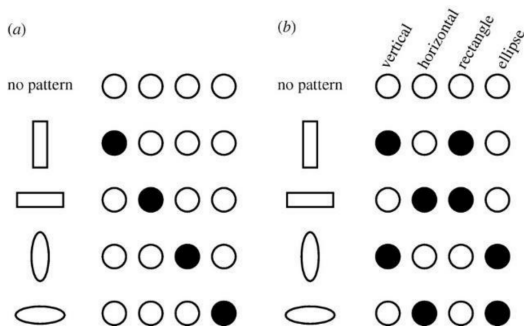
(Inspired by Moontae Lee and Dhruv Batra)

Distributed Representations

Key idea: no single neuron "encodes" everything; groups of neurons (e.g. in the same hidden layer) work together!

cf. the grandmother cell

The Power of Distributed Representations

- Distributed representations are more compact (there can be $O(\exp N)$ objects combining N properties)
- They are also more powerful, as they can generalize to unseen objects in a meaningful way:
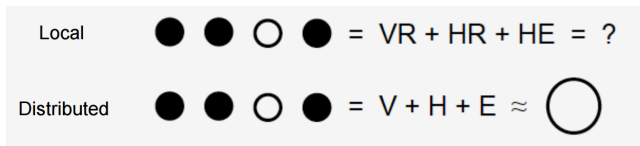
(Inspired by Moontae Lee and Dhruv Batra)

The Power of Distributed Representations

- Hidden units should capture diverse properties of objects (not all capturing the same property)

- Usually ensured by random initialization of the weights

- Initializing all the units to the same weights would never break the symmetry!

- Side note: a NN computes the same function if we permute the hidden units within a layer (order doesn't matter, only diversity)

Next: how to learn useful object representations from raw inputs (no labels)?

Example: Unsupervised Pre-Training

- Training deep NNs (with many hidden layers) can be challenging

- This has been a major difficulty with NNs for a long time

- Initialize hidden layers using unsupervised learning
  (Erhan et al., 2010):

  - Force network to represent latent structure of input distribution

  - Encourage hidden layers to encode that structure

  - This can be done with an auto-encoder!

Data Manifold

Key idea: learn the manifold where the input objects live
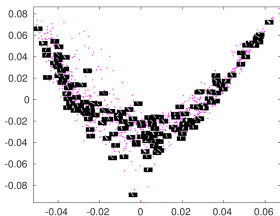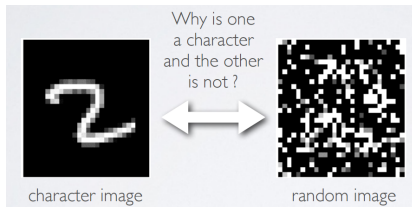
(Image credit: Hugo Larochelle)

Learn representations that encode well points in that manifold

Auto-Encoders

Auto-encoder: feed-forward NN trained to reproduce its input at the output

- Encoder: $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$

- Decoder: $\widehat{\boldsymbol{x}} = \boldsymbol{W}^{\top}\boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{c}$

- Loss function (for $\boldsymbol{x} \in \mathbb{R}^{D}$):

$$L(\widehat{\boldsymbol{x}}; \boldsymbol{x}) = \|\widehat{\boldsymbol{x}} - \boldsymbol{x}\|^2$$

- Objective (dropping the biases):

$$\widehat{\boldsymbol{W}} = \arg \min_{\boldsymbol{W}} \sum_i \|\boldsymbol{W}^{\top}\boldsymbol{g}(\boldsymbol{W}\boldsymbol{x}_i) - \boldsymbol{x}_i\|^2$$

$\widehat{\mathbf{x}}$

$\mathbf{W}^{\top}$ (tied parameters)

$\mathbf{h(x)}$

$\mathbf{W}$

$\mathbf{x}$
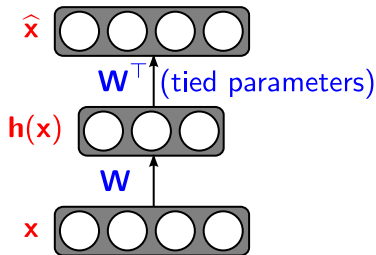
The Simplest Auto-Encoder: Linear

What happens if the activation function **g** is linear?
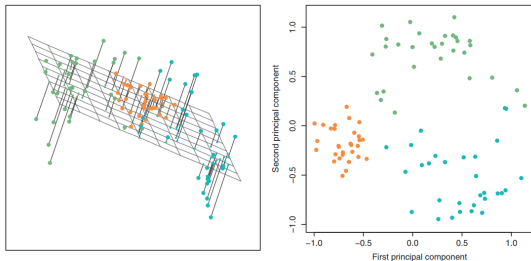
FIGURE 10.2. *Ninety observations simulated in three dimensions. Left: the first two principal component directions span the plane that best fits the data. It minimizes the sum of squared distances from each point to the plane. Right: the first two principal component score vectors give the coordinates of the projection of the 90 observations onto the plane. The variance in the plane is maximized.*

(From "An Introduction to Statistical Learning" by James, Witten, Hastie, Tibshirani)

An Important Tool: Singular Value Decomposition

- Any rank-r matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ can be written as $\boldsymbol{A} = \boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{V}^T$

  ✓ columns of $\boldsymbol{U} \in \mathbb{R}^{m \times r}$ are an orthonormal basis of $\mathcal{R}(\boldsymbol{A})$;

  ✓ columns of $\boldsymbol{V} \in \mathbb{R}^{n \times r}$ are an orthonormal basis of $\mathcal{R}(\boldsymbol{A}^T)$;

  ✓ $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, ..., \sigma_r)$ is a $r \times r$ diagonal matrix;

  ✓ $\sigma_1, ..., \sigma_r$ are square roots of the eigenvalues of $\boldsymbol{A}^T \boldsymbol{A}$ or $\boldsymbol{A} \boldsymbol{A}^T$;

  ✓ $\sigma_1, ..., \sigma_r$ are called singular values.

- Orthonormality of $\boldsymbol{U}$ and $\boldsymbol{V}$: $\boldsymbol{U}^T \boldsymbol{U} = \boldsymbol{I}$ and $\boldsymbol{V}^T \boldsymbol{V} = \boldsymbol{I}$.
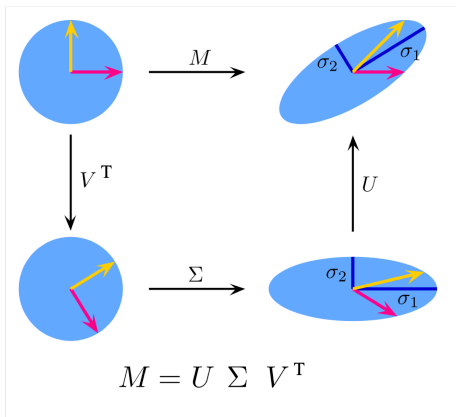
Singular Value Decomposition (SVD)

- $A = U\Sigma V^T$, where $U \in \mathbb{R}^{m \times r}$ and $V \in \mathbb{R}^{n \times r}$.
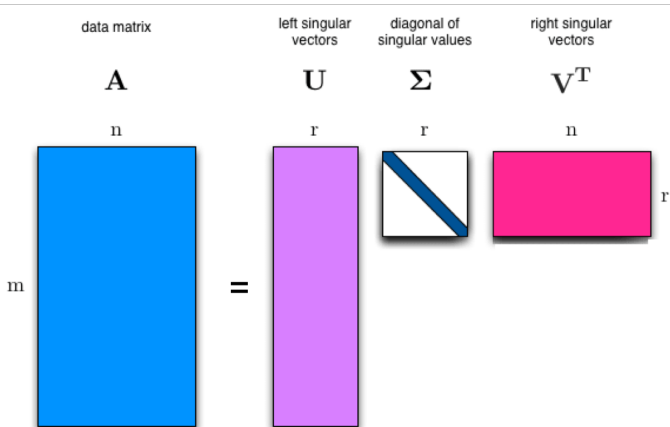
Picture credits: Mukesh Mithrakumar

Singular Value Decomposition (SVD)

- $M = U\Sigma V^T$, where $U \in \mathbb{R}^{m \times r}$ and $V \in \mathbb{R}^{n \times r}$.

$$M = U \ \Sigma \ V^{\mathbf{T}}$$

Picture credits: Wikipedia

Linear Auto-Encoder

Let $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ be the data matrix ($N > D$),

Assume $\boldsymbol{W} \in \mathbb{R}^{K \times D}$ with $K < D$ (no biases, assuming $\boldsymbol{X}$ is centred)

We want to minimize

$$\sum_{i=1}^{N} \|\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i\|_2^2 = \|\boldsymbol{X} - \widehat{\boldsymbol{X}}\|_F^2 = \|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{W}^\top\boldsymbol{W}\|_F^2$$

where $\|\cdot\|_F^2$ is the Frobenius matrix norm and $\boldsymbol{W}^\top\boldsymbol{W}$ has rank K.

From the Eckart-Young theorem, the minimizer is truncated SVD of $\boldsymbol{X}^\top$:

$$\widehat{\boldsymbol{X}}^\top = \boldsymbol{U}_K \boldsymbol{\Sigma}_K \boldsymbol{V}_K^\top,$$

where $\boldsymbol{\Sigma}_K$ is a diagonal matrix containing the top K singular values of $\boldsymbol{X}^\top$, and the columns of $\boldsymbol{U}_K$ are the corresponding left singular vectors.

The solution is $\boldsymbol{W} = \boldsymbol{U}_K^\top$, which gives as desired:

$$\widehat{\boldsymbol{X}}^\top = \boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{X}^\top = \boldsymbol{U}_K\boldsymbol{U}_K^\top\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top = \boldsymbol{U}_K\boldsymbol{\Sigma}_K\boldsymbol{V}_K^\top.$$

This is called principle component analysis (PCA)

PCA: EigenFaces
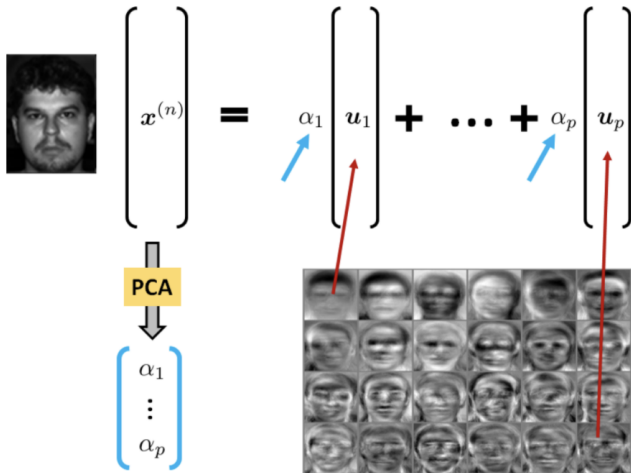
Auto-Encoders

PCA fits a linear manifold (affine space) to the data

By using non-linear activations, we obtain more sophisticated codes (i.e. representations).

We need some sort of regularization to:

- encourage a smooth representation (small perturbations of the input will lead to similar representations)

- avoid overfitting to the training data

Some Variants of Auto-Encoders

- **Sparse auto-encoders:** use many hidden units, but add a ℓ_1 regularization term to encourage sparse representations of the input

- **Denoising auto-encoders:** regularize by adding noise to the input; the goal is to learn a smooth representation function that allows to output the denoised input (inspired by image denoising)

- **Stacked auto-encoders:** several auto-encoders on top of each other

- **Variational auto-encoders:** a generative probabilistic model that minimizes a variational bound (this will be covered in another lecture!)

Regularized Auto-Encoders

- To regularize auto-encoders, regularization may be added to the loss

- The goal is then to minimize $L(\widehat{\boldsymbol{x}}; \boldsymbol{x}) + \Omega(\boldsymbol{h}, \boldsymbol{x})$

- For example:

    - regularizing the code $\Omega(\boldsymbol{h}, \boldsymbol{x}) = \lambda \|\boldsymbol{h}\|^2$

    - regularizing the derivatives $\Omega(\boldsymbol{h}, \boldsymbol{x}) = \lambda \sum_i \|\nabla_{\boldsymbol{x}} h_i\|^2$

- The encoder and decoder parameters may be shared or not.

Sparse Auto-Encoders

- Most auto-encoders learn low-dimensional codes, e.g., they reduce input dimensionality (bottleneck shape $K < D$).

- One exception are sparse auto-encoders:

  - Sparse auto-encoders incorporate a sparsity penalty $\Omega(\boldsymbol{h})$ on the code layer, e.g., $\Omega(\boldsymbol{h}) = \lambda \|\boldsymbol{h}\|_1$

  - Typically the number of hidden units is large, e.g., larger than the input dimension

  - The sparsity penalty encourages sparse codes, where most hidden units are inactive.

Stochastic Auto-Encoders

- In this case, the encoder and decoder are not deterministic functions, but involve some noise/randomness

- Uses distribution $p_{\text{encoder}}(h \mid x)$ for the encoder and a distribution $p_{\text{decoder}}(x \mid h)$ for the decoder

- The auto-encoder can be trained to minimize

$$- \log p_{\text{decoder}}(x \mid h).$$

Denoising Auto-Encoders

- Use a perturbed version of the input, $\tilde{x} = x + n$, where n is random noise (e.g. Gaussian noise $n \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$)

- Instead of minimizing $\frac{1}{2}\|\widehat{x} - x\|^2$, minimize $\frac{1}{2}\|\widehat{x} - \tilde{x}\|^2$

- This is a form of implicit regularization that ensures smoothness: it forces the system to represent well not only the data points, but also their perturbations

Denoising Auto-Encoders
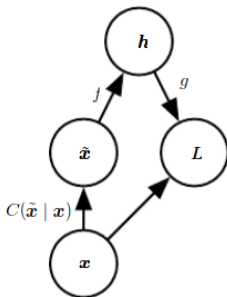
Figure 14.3: The computational graph of the cost function for a denoising autoencoder, which is trained to reconstruct the clean data point $\boldsymbol{x}$ from its corrupted version $\tilde{\boldsymbol{x}}$. This is accomplished by minimizing the loss $L = -\log p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h} = f(\tilde{\boldsymbol{x}}))$, where $\tilde{\boldsymbol{x}}$ is a corrupted version of the data example $\boldsymbol{x}$, obtained through a given corruption process $C(\tilde{\boldsymbol{x}} \mid \boldsymbol{x})$. Typically the distribution p_{decoder} is a factorial distribution whose mean parameters are emitted by a feedforward network g.

(From Goodfellow et al.'s book.)

Denoising Auto-Encoders
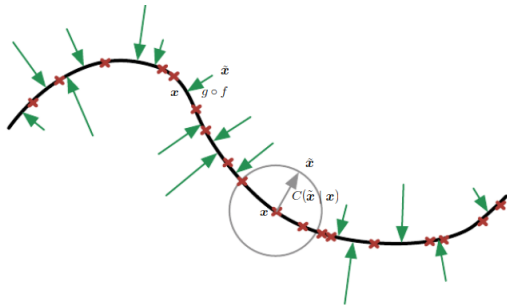
Figure 14.4: A denoising autoencoder is trained to map a corrupted data point $\tilde{x}$ back to the original data point x. We illustrate training examples x as red crosses lying near a low-dimensional manifold, illustrated with the bold black line. We illustrate the corruption process $C(\tilde{x} \mid x)$ with a gray circle of equiprobable corruptions. A gray arrow demonstrates how one training example is transformed into one sample from this corruption process. When the denoising autoencoder is trained to minimize the average of squared errors $\|g(f(\tilde{x})) - x\|^2$, the reconstruction $g(f(\tilde{x}))$ estimates $\mathbb{E}_{x, \tilde{x} \sim p_{\text{data}}(x)C(\tilde{x}|x)}[x \mid \tilde{x}]$. The vector $g(f(\tilde{x})) - \tilde{x}$ points approximately toward the nearest point on the manifold, since $g(f(\tilde{x}))$ estimates the center of mass of the clean points x that could have given rise to $\tilde{x}$. The autoencoder thus learns a vector field $g(f(x)) - x$ indicated by the green arrows. This vector field estimates the score $\nabla_x \log p_{\text{data}}(x)$ up to a multiplicative factor that is the average root mean square reconstruction error.

Why Do We Use Auto-Encoders?

Historically, training deep neural networks was hard

One of the initial successful uses of auto-encoders was for unsupervised pre-training (Erhan et al., 2010).

Unsupervised Pre-Training

A greedy, layer-wise procedure:

- train one layer at a time, from first to last, with unsupervised criterion (e.g. an auto-encoder)
- fix the parameters of previous hidden layers
- previous layers viewed as feature extraction
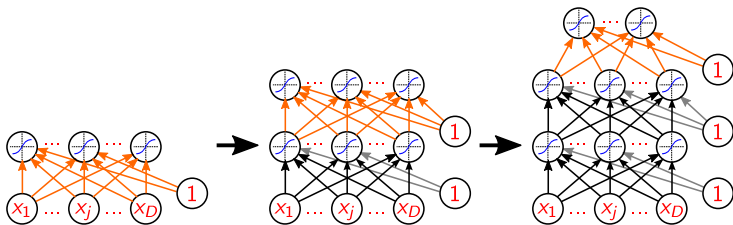
Pre-training initializes the parameters in a region such that the near local optima overfit less the data.

Fine-Tuning

Once all layers are pre-trained:

- add output layer
- train the whole network using supervised learning

Supervised learning is performed as in a regular feed-forward network:

- forward propagation, backpropagation, and update
- all parameters are "tuned" for the supervised task at hand
- representation is adjusted to be more discriminative

Other Applications of Auto-Encoders

- Dimensionality reduction

- Information retrieval and semantic hashing (via binarizing the codes)

- Conversion of discrete inputs to low-dimensional continuous space

Word Representations

- Learning representations of words in natural language;

- Also called word embeddings;

- An extremely successful application of representation learning;

- Still an active area of research.

Distributional Similarity

Key idea: represent a word by means of its neighbors

- *"You shall know a word by the company it keeps"* (J. R. Firth, 1957)
- One of the most successful ideas of modern statistical NLP!

For example:

- Adjectives are normally surrounded by nouns
- Words like *book, newspaper, article*, are commonly surrounded by *reading, read, writes*, but not by *flying, eating, sleeping*

Word Embeddings

- Obtain (lower dimensional) vector representations of words.

- Two possible methods:

  - ✓ Factorization of a co-occurrence word/context matrix (latent semantic analysis, etc.)

  - ✓ Directly learn low-dimensional vectors by training a network to *predict* the context of a given word

- We focus on the latter, namely word2vec (Mikolov et al., 2013), which follows previous ideas (Bengio et al., 2003; Collobert et al., 2011).

Neural Language Model (Bengio et al., 2003)
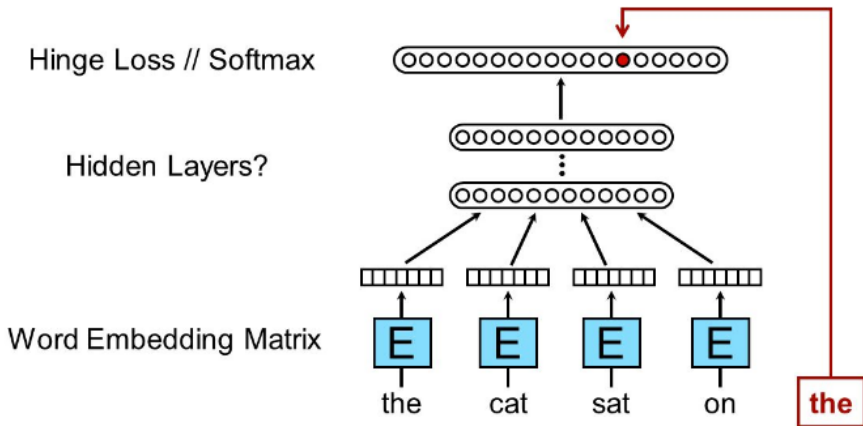
(Image credits: Quoc Le)

Neural Language Model (Bengio et al., 2003)

Embedding matrix: assigns a vector to every word in the vocabulary.

Learning the embeddings:

- Each word is associated with a vector (word embedding)

- Given the context (previous K words), predict the next word

- The word embeddings in the context window are concatenated into a vector that is fed to neural network

- The output layer is a huge softmax assigning probabilities to each word in the vocabulary

- The network is trained by SGD with backpropagation including the embedding matrix.

Variants of this model outperform smoothed K-th order Markov models

Some Insights

- Often, we are not concerned with language modelling (the addressed task), but with the quality of the embeddings learned

- If we don't care about language modelling,

  - ✓ We don't need to have a "left-to-right model" where we try to predict the next word given the context

  - ✓ We don't need to predict the probability of every word, just make sure that the true word is more likely than a random one

- These insights underlie the word2vec model of Mikolov et al. (2013).

Word2Vec (Mikolov et al., 2013)

- Considers a context window around each word in the sentence.

- Word2vec comes with two variants:

  - ✓ **skip-gram**: predict surrounding context words in a window of length m of every word

  - ✓ **continuous bag-of-words** (CBOW): predict the central word from the context

- We focus on the skip-gram model (more widely used).

Skip-Gram

- **Objective:** maximize the log probability of any context word given the central word:

$$J(\Theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, \ j \neq 0} \log p_\Theta(x_{t+j} \mid x_t)$$

- There are 2 sets of parameters (2 embedding matrices) $\Theta = (\boldsymbol{u}, \boldsymbol{v})$:

  - ✓ Embeddings $\boldsymbol{u}_o$ for each word o appearing as the center word

  - ✓ Embeddings $\boldsymbol{v}_c$ for each word c appearing as context of another word

- Uses a log-bilinear model: $p_\Theta(x_{t+j} = c \mid x_t = o) \propto \exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)$

- Every word gets two vectors

- In the end, we use the $\boldsymbol{u}$ as the word embeddings, discarding $\boldsymbol{v}$

The Large Vocabulary Problem

- Recall that

$$p_\Theta(x_{t+j} = c \mid x_t = o) = \frac{\exp(\boldsymbol{u_o}^T \boldsymbol{v_c})}{\sum_{c'} \exp(\boldsymbol{u_o}^T \boldsymbol{v_{c'}})}$$

- This is a softmax over the entire vocabulary (often > 50000)

- With large vocabularies, this leads to very slow training. Possible workarounds:

  - ✓ Stochastic sampling

  - ✓ Noise contrastive estimation

  - ✓ Negative sampling

- More details in https://arxiv.org/pdf/1410.8251.pdf

- We focus on negative sampling.

Negative Sampling

Key idea:

- Replace the huge softmax by binary logistic regressions for a true pair (center word and word in its context window) and k of random pairs (center word, random word):

$$J_t(\Theta) = \log \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) + \sum_{i=1}^{k} \log \sigma(-\boldsymbol{u}_o^\top \boldsymbol{v}_{j_i}), \qquad j_i \sim P(x)$$

- Several strategies for the sampling distribution (uniform, unigram frequency, etc.)

- Negative sampling is a simple form of unsupervised pre-training.

Linear Relationships

- Work embeddings are good at encoding dimensions of similarity

- Word analogies can be solved well simply via subtraction in the embedding space

- Syntactically:

$$\boldsymbol{x}_{\text{apple}} - \boldsymbol{x}_{\text{apples}} \approx \boldsymbol{x}_{\text{car}} - \boldsymbol{x}_{\text{cars}} \approx \boldsymbol{x}_{\text{family}} - \boldsymbol{x}_{\text{families}}$$

- Semantically:

$$\boldsymbol{x}_{\text{shirt}} - \boldsymbol{x}_{\text{clothing}} \quad \approx \quad \boldsymbol{x}_{\text{chair}} - \boldsymbol{x}_{\text{furniture}}$$

$$\boldsymbol{x}_{\text{king}} - \boldsymbol{x}_{\text{man}} \quad \approx \quad \boldsymbol{x}_{\text{queen}} - \boldsymbol{x}_{\text{woman}}$$

Visualization

- Typical embedding dimensions are in the hundreds (e.g. 300)

- How can we visualize these embeddings?

- Simple way: project them in 2D with something like PCA

- Most used: t-SNE (*t*-distributed stochastic neighbor embedding (Maaten and Hinton, 2008)))
  https://lvdmaaten.github.io/tsne

Word Analogies (Mikolov et al., 2013)
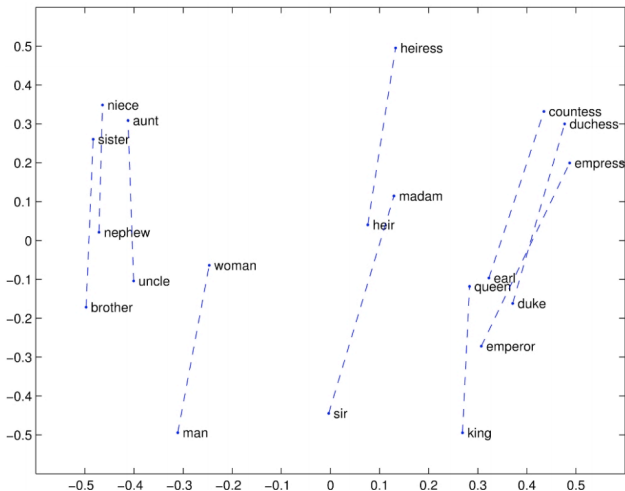
(Slide credit to Richard Socher)

Other Methods for Obtaining Word Embeddings

GloVe: Global Vectors for Word Representation (Pennington et al., 2014)

- `https://nlp.stanford.edu/projects/glove`
- Training is performed on aggregated global word-word co-occurrence statistics from a corpus

FastText (Bojanowski et al., 2016): embeds also character *n*-grams for generating embeddings for out-of-vocabulary words

- `https://fasttext.cc` (from FAIR)
- open-source, free, lightweight library that allows users to learn text representations and text classifiers
- contains multi-lingual word vectors for 157 different languages

GloVe Visualizations: Company → CEO
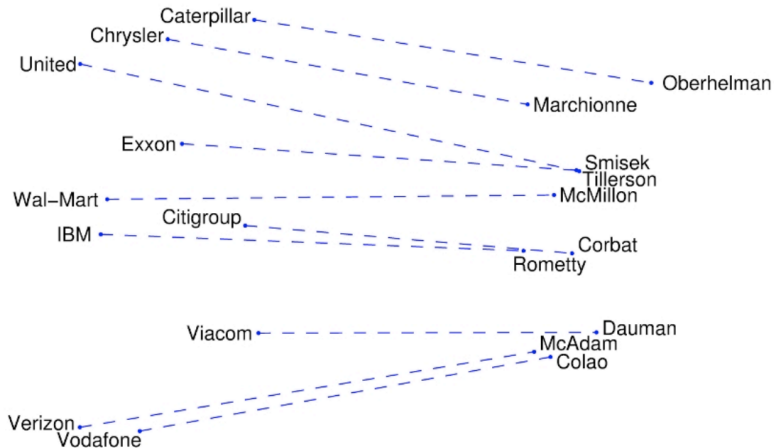
(Slide credit to Richard Socher)

GloVe Visualizations: Superlatives
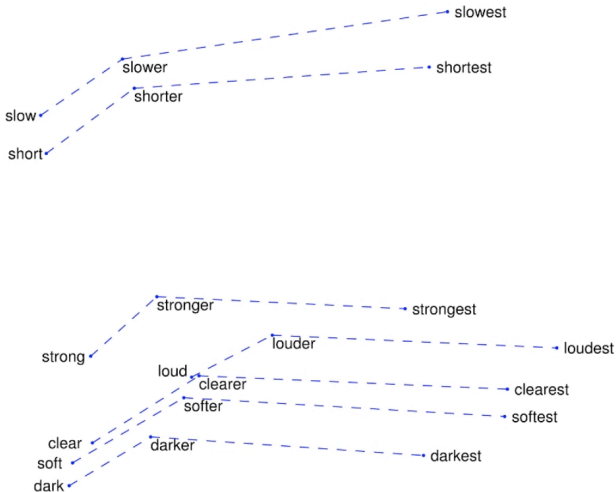
(Slide credit to Richard Socher)

Word Embeddings: Some Open Problems

- Can we have word embeddings for multiple languages in the same space?

- How to capture polysemy?

- These word embeddings are static, can we compute embeddings on-the-fly depending on the context?

Cross-Lingual Word Embeddings
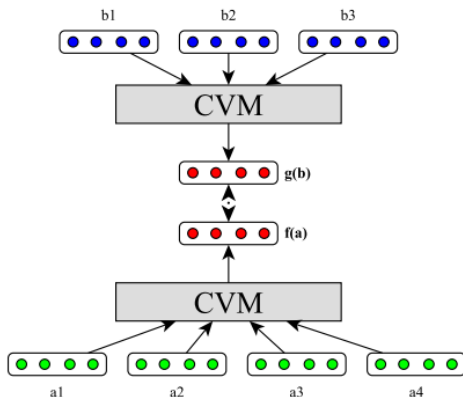
Figure 1: Model with parallel input sentences a and b. The model minimises the distance between the sentence level encoding of the bitext. Any composition functions (CVM) can be used to generate the compositional sentence level representations.

(From Hermann and Blunsom (2014).)

Cross-Lingual Word Embeddings

- Use a corpus of parallel sentences in two languages

- Define a composition function to obtain a sentence representation given word embeddings

- Apply a loss function that encourages the sentence representations in the two languages to be similar

- Negative sampling works here too: true pair vs fake pair.

Cross-Lingual Word Embeddings

Other approaches:

- Define a bilingual dictionary and apply canonical correlation analysis (Faruqui and Dyer, 2014)

- Task-specific embeddings with convex optimization (Ferreira et al., 2016)

- Learn the two embeddings separately, and then apply a linear transformation to put them in a shared space (Artetxe et al., 2017)

- Adversarial training (Lample et al., 2018)

This is a very active area of research!

Contextual Embeddings

Words can have different meanings, depending on the context

In 2018, a model called ELMo learned context-dependent embeddings and achieved impressive results on 6 NLP downstream tasks (Peters et al., 2018)

Key idea:

- Pre-train a BILSTM language model on a large dataset (we'll see in a later class what this is)
- Save all the encoder parameters at all layers, not only the embeddings
- Then, for your downstream task, tune a scalar parameter for each layer, and pass the entire sentence through this encoder.

BERT, GPT, etc.

Some time later, a Transformer-based model (BERT) achieved even better performance:

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin   Ming-Wei Chang   Kenton Lee   Kristina Toutanova
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Huge improvements in multiple NLP tasks!

(Trained on 64 TPU chips!!)

Other related models include GPT-2, GPT-3, etc.

This will be covered in a later lecture!

Conclusions

- Neural nets learn internal representations that can be transferred across tasks

- Distributed representations are exponentially more compact and allow generalizing to unseen objects

- Deeper neural nets exhibit hierarchical compositionality: upper level layers learn more abstract/semantic representations than bottom level layers

- Auto-encoders are an effective means for learning representations

- Word embeddings are continuous representations of words that are extremely useful in NLP

Thank you!

Questions?

References I

Artetxe, M., Labaka, G., and Agirre, E. (2017). Learning bilingual word embeddings with (almost) no bilingual data. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 451–462.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3(Feb):1137–1155.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660.

Faruqui, M. and Dyer, C. (2014). Improving vector space word representations using multilingual correlation. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 462–471.

Ferreira, D., Almeida, M. S. C., and Martins, A. F. T. (2016). Jointly Learning to Embed and Predict with Multiple Languages. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*.

Hermann, K. M. and Blunsom, P. (2014). Multilingual Models for Compositional Distributional Semantics. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*.

Hinton, G. E. (1984). Distributed representations.

Lample, G., Ott, M., Conneau, A., Denoyer, L., and Ranzato, M. (2018). Phrase-based & neural unsupervised machine translation. *arXiv preprint arXiv:1804.07755*.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global Vectors for Word Representation. *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP 2014)*, 12:1532–1543.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Zeiler, M. D. and Fergus, R. (2013). Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*.

Lecture 7: Convolutional Neural Networks

André Martins, Francisco Melo, Mário Figueiredo

Deep Learning Course, Winter 2022-2023
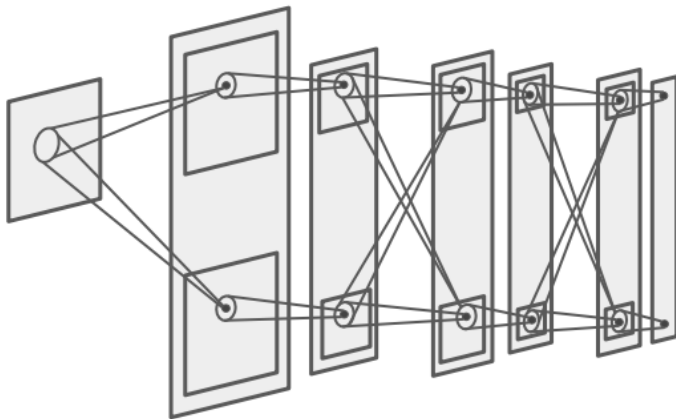
Convolutional Neural Networks

What is a convolutional neural network (CNN)?

...just a NN with a special connectivity structure.

Roadmap:

- Parameter tying/sharing

- 2D CNNs for object recognition

- Pooling layers

- Classical CNNs: ImageNet, AlexNet, GoogLeNet

- One-dimensional CNNs for NLP

Neocognitron (Fukushima and Miyake, 1982)

(Credits: Fei-Fei Li, Johnson, Yeung)

- "Sandwich" architecture, alternating between simple cells with modifiable parameters and complex cells which perform pooling

Neocognitron (Fukushima and Miyake, 1982)
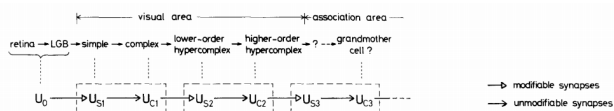
Fig. 1. Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron
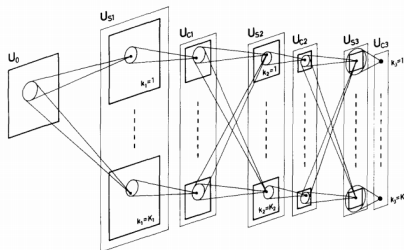
Fig. 2. Schematic diagram illustrating the interconnections between layers in the neocognitron

- Inspired by the multi-stage hierarchy model of the visual nervous system (Hubel and Wiesel, 1965)

ConvNet (LeNet-5) (LeCun et al., 1998)
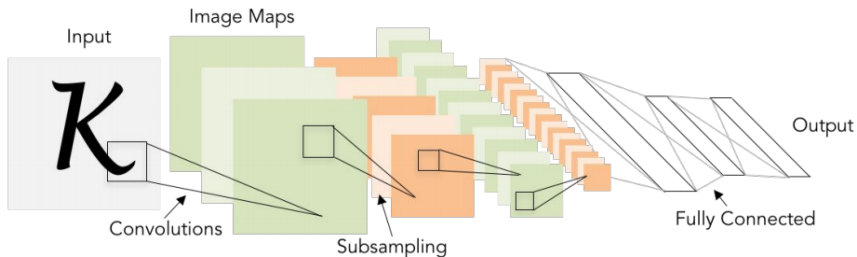
Convolutional Networks

- How is a CNN different from a standard feedforward NN?

- What is a convolutional layer?

- How is it different from a fully connected layer?

Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1
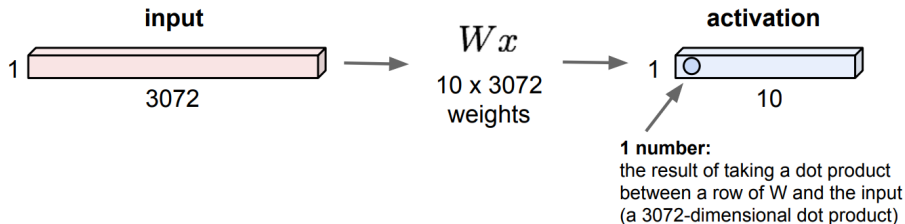
(Credits: Fei-Fei Li, Johnson, Yeung)

All activations depend on all inputs.

Convolutional Layer

Don't stretch/reshape: preserve the spacial structure!

Convolution Layer
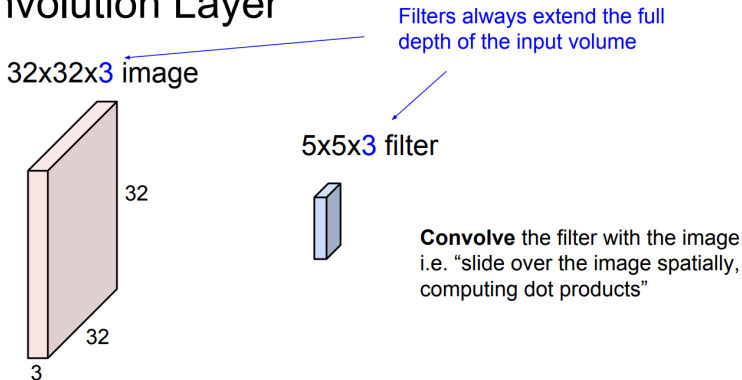
32x32x3 image

Filters always extend the full depth of the input volume

5x5x3 filter

32

32

3

Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products"

(Credits: Fei-Fei Li, Johnson, Yeung)

Convolutional Layer

32x32x3 image
5x5x3 filter w

1 number:
the result of taking a dot product between the
filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$
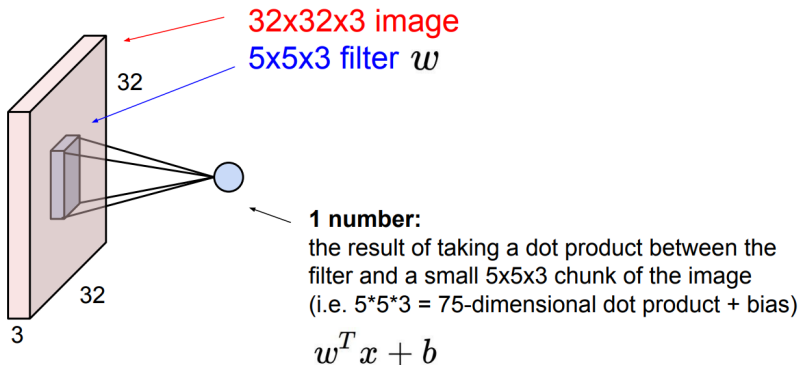
(Credits: Fei-Fei Li, Johnson, Yeung)

Convolutional Layer

Apply the same filter to all spatial locations (28x28 times, why?):

activation map

32x32x3 image
5x5x3 filter

32

32

3

convolve (slide) over all spatial locations

28

28

1
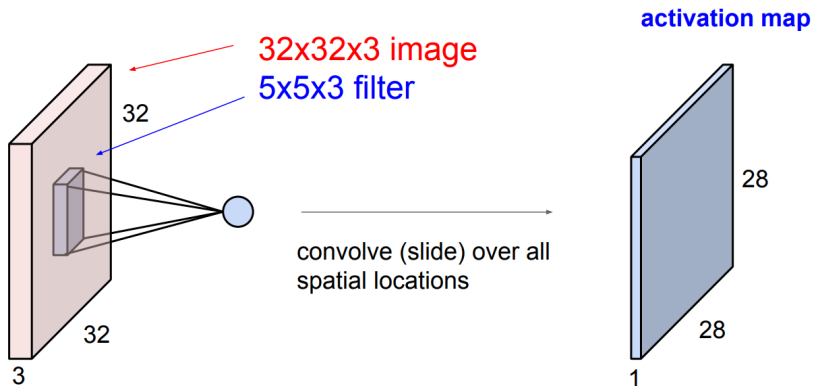
(Credits: Fei-Fei Li, Johnson, Yeung)

Convolutional Layer

- For example, if we have **6** 5x5x3 filters, we get **6** activation maps:
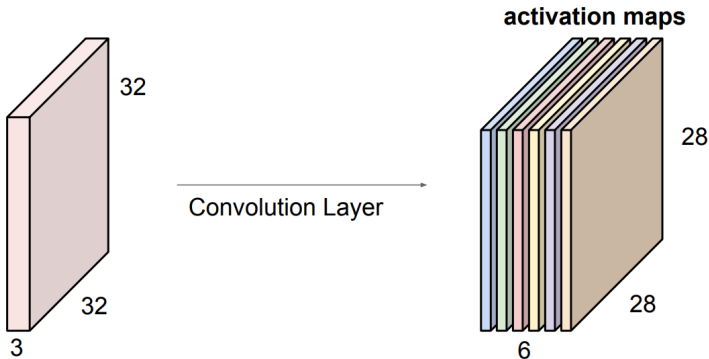
(Credits: Fei-Fei Li, Johnson, Yeung)

- We stack these up to get a new "image" of size 28x28x6!

Image Size, Filter Size, Stride, Channels

Stride: shift in pixels between two consecutive windows.

In the previous illustrations: stride = 1.

Number of channels: number of filters used in each layer.

Given an $N \times N \times D$ image, $F \times F \times D$ filters, K channels, and stride S, the resulting output will be of size $M \times M \times K$, where

$$M = (N - F)/S + 1$$

Examples:

- $N = 32$, $D = 3$, $F = 5$, $K = 6$, $S = 1$ results in an $28 \times 28 \times 6$ output
- $N = 32$, $D = 3$, $F = 5$, $K = 6$, $S = 3$ results in an $10 \times 10 \times 6$ output

Padding: append zeros around the images.

Common padding size: $(F - 1)/2$, which preserves spatial size: $M = N$.

CNNs and Convolutions

Why is this called "convolutional"?

The convolution of a signal x and a filter w, denoted $x * w$, is:

$$h[t] = (x * w)[t] = \sum_{a=-\infty}^{\infty} x[t-a]w[a].$$

Basic idea: sparse/local connectivity and parameter tying/sharing.

h(x)

x

16 parameters   3 parameters
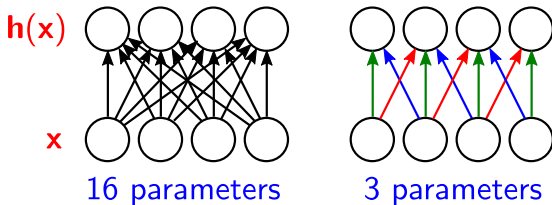
Convolutions with Padding

Expression above is for infinite-support signal x and filter w.

Finite support: $x = (x[0], ..., x[N-1])$; $w = (w[-D], ..., w[D])$ ($F = 2D + 1$)

$$h[t] = (x * w)[t] = \sum_{a=-D}^{D} w[a] x[t-a], \text{ for } t = D, ..., N-D-1$$

The result has support of size $N - D - 1 - D + 1 = N - 2D$.

Padding: append $D = (F-1)/2$ zeros at each side of x.
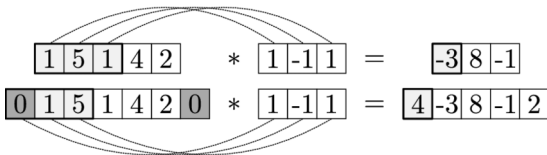
Convolutions and Parameter Tying

Leads to translation/shift equivariance
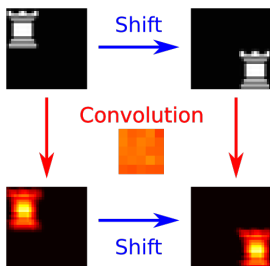
Why do we want to tie (share) parameters?

- Reduce the number of parameters to be learned
- Deal with arbitrary long, variable-length, sequences: rather than shifting the filters, shift the input

Can also be done in 1D (e.g., text data, signals, ...)

Convolutions and Pooling

The second component of CNNs is pooling

Common CNNs alternate convolutional layers and pooling layers.

Pooling layers provide invariance.

Equivariance vs Invariance

Equivariance | Invariance
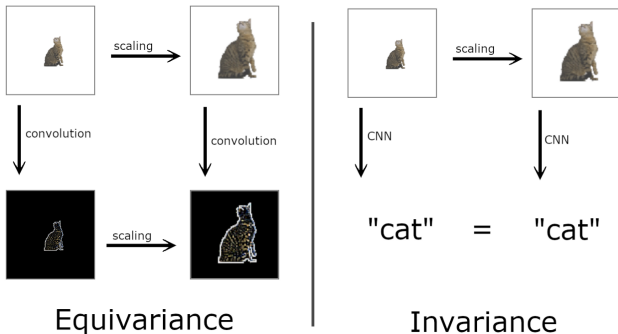
Pooling Layer

- Makes the representations smaller, more manageable.
- Operates over each activation map (each channel) independently
- Max-pooling:
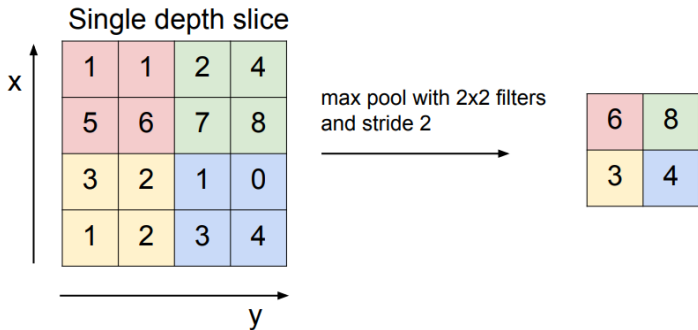
(Credits: Fei-Fei Li, Johnson, Yeung)

Max Pooling: Shift Invariance
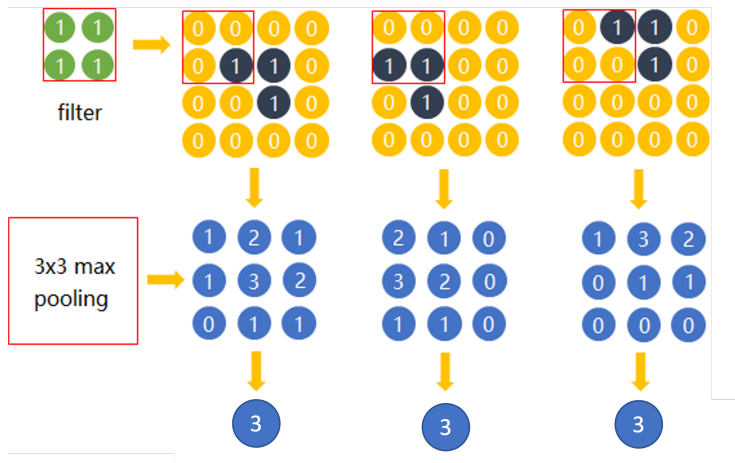
Max Pooling: Shift Invariance (II)
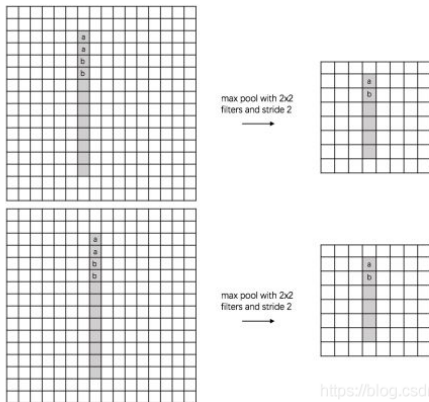
Lecture 7

Max Pooling: Rotation Invariance
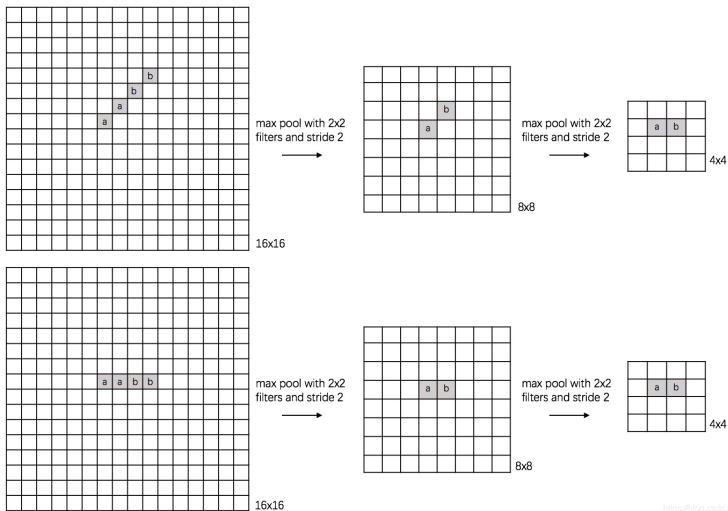
Max Pooling: Scale Invariance
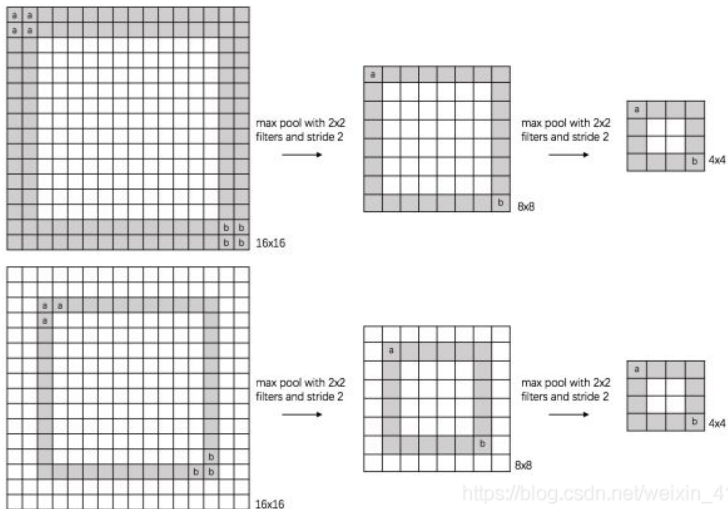
Multiple Convolution Filters: Feature Maps

- Different filter for each channel, but keeping spatial invariance:
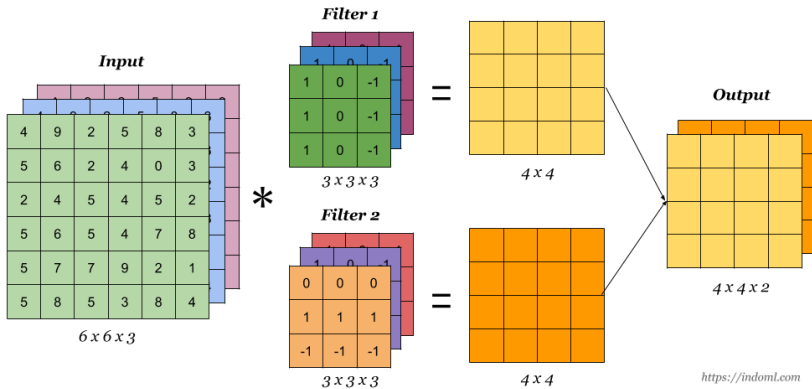
(Figure credit: Andrew Ng)

I

2D Convolutional Nets (LeCun et al., 1989)

- Inspired by "Neocognitron" (Fukushima, 1980)
- 2D Convolutions: the same filter (e.g. 3x3) is applied to each location of the image
- The filter weights are learned (as tied parameters)
- Multiple filters
- Alternates convolutional and pooling layers.
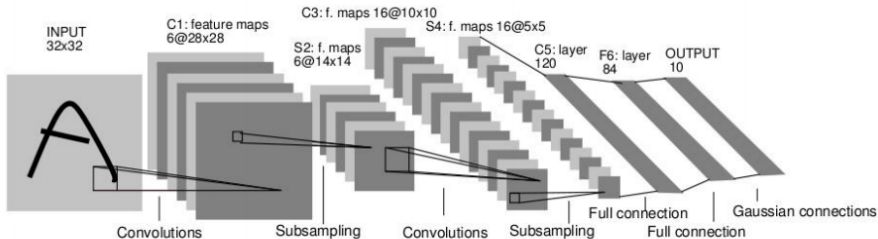
ConvNet Successes: MNIST

Handwritten text/digits:

- MNIST (0.35% error (Ciresan et al., 2011b))
- Arabic and Chinese (Ciresan et al., 2011a)

ConvNet Successes: CIFAR-10, Traffic Signs

Simpler recognition benchmarks:

- CIFAR-10 (9.3% error (Wan et al., 2013))
- Traffic signs: 0.56% error vs 1.16% for humans (Cireşan et al., 2011)

Less good at more complex tasks, e.g., Caltech-101/256 (few training samples).

ImageNet Dataset

- 14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon Turk

(Slide credit to Rob Fergus)

AlexNet (Krizhevsky et al., 2012)

- 54M parameters; 8 layers (5 conv, 3 fully-connected)
- Trained on 1.4M ImageNet images
- Trained on 2 GPUs for a week (50x speed-up over CPU)
- Dropout regularization
- Test error: 16.4% (second best team was 26.2%)
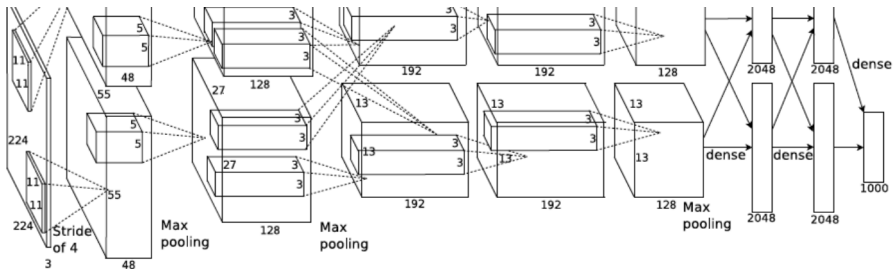
GoogLeNet (Szegedy et al., 2015)

- GoogLeNet inception module: very deep convolutional network, fewer (5M) parameters

Convolution
Pooling
Softmax
Other
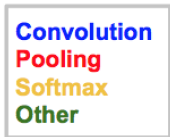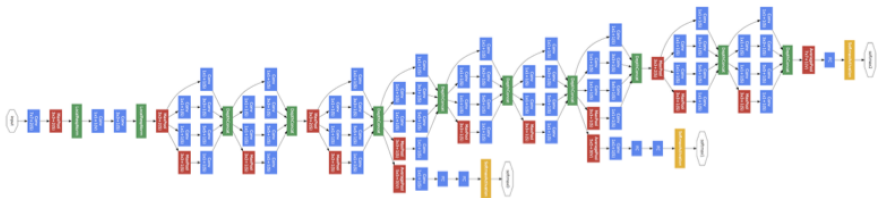
Residual Networks (ResNets)

- Add skip-connections; tends to lead to more stable learning.
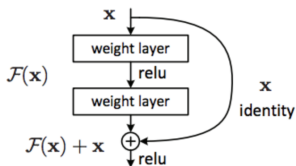
Figure 2. Residual learning: a building block.

(He et al., 2016)

Key (not unique) motivation: mitigate the vanishing gradient problem

With $H(\boldsymbol{x}) = \mathcal{F}(\boldsymbol{x}) + \lambda\boldsymbol{x}$, the gradient back-propagation becomes

$$\frac{\partial L}{\partial \boldsymbol{x}} = \frac{\partial L}{\partial H}\frac{\partial H}{\partial \boldsymbol{x}} = \frac{\partial L}{\partial H}\left(\frac{\partial \mathcal{F}}{\partial \boldsymbol{x}} + \lambda\right)$$
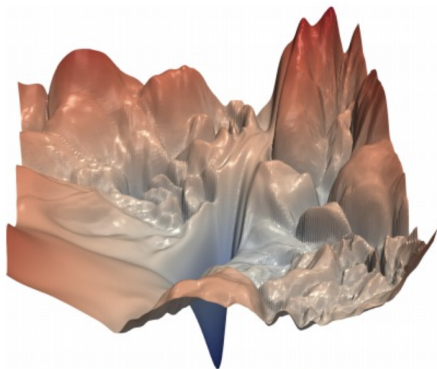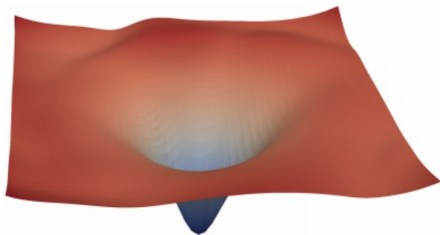
Residual Networks (ResNets)

- Very deep network (34 layers here, but up to 152 layers!)

- VGG-19 ("Visual Geometry Group") by Simonyan and Zisserman (2014) (19 layers, but more FLOPs)

Residual Networks (ResNets)

(a) without skip connections   (b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

(Li et al., 2018)

Convolutional Nets in NLP

- So far, we focused on CNNs for images.

- Are CNNs also used in NLP? Not as much, but...

- Yoav Goldberg in the Representation Learning Workshop (ACL 2018):

  "NLP's ImageNet moment has arrived."

 (not referring to CNNs, in particular, but to big NNs for NLP.

Convolutional Nets in NLP

- 1D convolutions (text is a sequence)

- Filters are applied to local windows around each word

- For word embeddings $x_1, \ldots, x_L$, the filter response for word i is:

$$h_i = g(W[x_{i-h} \oplus \ldots \oplus x_i \oplus \ldots x_{i+h}] + b),$$

  where $\oplus$ denotes vector concatenation and W are shared parameters

- Can pad left and right with special symbols if needed.
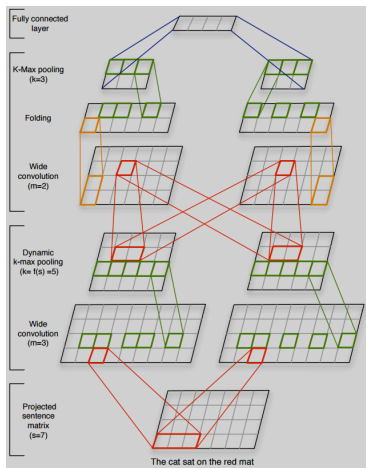
Kalchbrenner et al. (2014)

Variable Input Length

- Most computation in CNNs can be done in parallel.

- GPUs can leverage this and achieve great speed-ups!

- But, unlike images, which have fixed size, sentences have different lengths, which makes batching a bit trickier!

Mini-Batching, Padding, and Masking

Mini-batching is necessary to speed up training in GPUs

How to cope with different sentence lengths?

Solution: minimize waste by sorting by sentence length before forming mini-batches, then padding:

(Image credit: Thang Luong, Kyunghyun Cho, Chris Manning)

Masking is needed to ensure the padding is not affecting the results.

Beyond Convolutions

- Other architectures have been proposed which offer alternatives to convolutions

- For example: transformers.

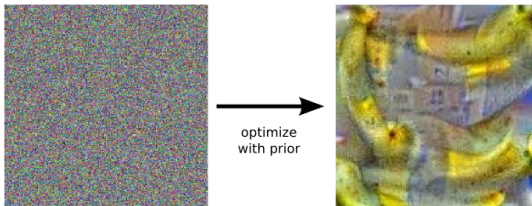- This is somewhat similar to "dynamic convolutions".

- Covered in another lecture.

What Representations Are We Learning?

- Which neurons fire for recognizing a particular object?

- What parts of the network are activated?

- To answer this, visualize what is happening inside the network.

Visualization

- **Idea:** Optimize input to maximize particular output
- Depends on the initialization
- Google DeepDream, maximizing "banana" output:

optimize
with prior

(from https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html)

- Can also specify an inner layer and tune the input to maximize its activations: useful to see what kind of features it is representing.
- Specifying a higher layer produces more complex representations...

Google DeepDream

(from https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html)

Adversarial Attacks

- How can we perturb an input slightly to fool a classifier?

- For example: 1-pixel attacks

- Glass-box model: assumes access to the model

- Backpropagate to the inputs to find pixels which maximize the gradient

- There's also work for black-box adversarial attacks (don't have access to the model, but can query it).
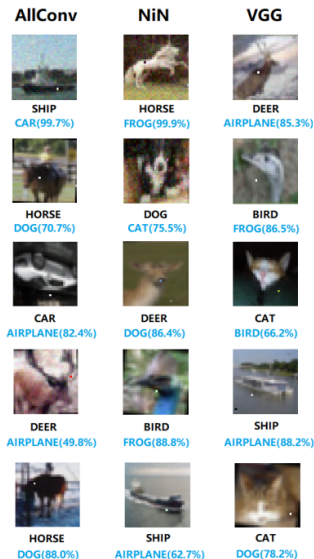
(Credits: Su, Vargas, Sakurai (2018))

Even Worse: Perturb Object, Not Image

- Print the model of a turtle in a 3D printer.

- Perturbing the texture fools the model into thinking it's a rifle, regardless of the pose of the object!

■ classified as turtle   ■ classified as rifle
■ classified as other

Figure 1. Randomly sampled poses of a 3D-printed turtle adversarially perturbed to classify as a rifle at every viewpoint[2]. An unperturbed model is classified correctly as a turtle nearly 100% of the time.

(Credits: Athalye, Engstrom, Ilyas, Kwok (2018))

Neural networks are still very brittle!

Conclusions

- CNNs are a very powerful architecture for computer vision

- CNNs take advantage of parameter sharing and sparse connectivity

- They are extremely useful to capture translational invariances in images

- Typically, convolution layers are alternated with max-pooling layers

- Lower layers capture more low-level representations (edges, corners)

- Higher layers have more "semantic" representations (objects, scenes)

Thank you!

Questions?

References I

Cireşan, D., Meier, U., Masci, J., and Schmidhuber, J. (2011). A committee of neural networks for traffic sign classification. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 1918–1921. IEEE.

Ciresan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. (2011a). Convolutional neural network committees for handwritten character classification. In *2011 International Conference on Document Analysis and Recognition*, pages 1135–1139. IEEE.

Ciresan, D. C., Meier, U., Masci, J., Maria Gambardella, L., and Schmidhuber, J. (2011b). Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1237.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202.

Fukushima, K. and Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hubel, D. H. and Wiesel, T. N. (1965). Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat. *Journal of neurophysiology*, 28(2):229–289.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. (2018). Visualizing the loss landscape of neural nets. *Advances in Neural Information Processing Systems*, 31.

References II

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9.

Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., and Fergus, R. (2013). Regularization of neural networks using dropconnect. In *Proc. of the International Conference on Machine Learning*, pages 1058–1066.

Lecture 8: Recurrent Neural Networks

André Martins, Francisco Melo, Mário Figueiredo

Deep Learning Course, Winter 2022-2023

Today's Roadmap

Today we'll cover neural sequential models:

- Recurrent neural networks.
- Backpropagation through time.
- Neural language models.
- The vanishing gradient problem.
- Gated units: LSTMs and GRUs.
- Bidirectional LSTMs.
- Example: ELMO representations.
- From sequences to trees: recursive neural networks.
- Other deep auto-regressive models: PixelRNNs.

Recurrent Neural Networks

Much interesting data is sequential in nature:

- ✓ Words in text

- ✓ DNA sequences

- ✓ Stock market returns

- ✓ Samples of sound signals

- ✓ ...

How to deal with sequences of arbitrary length?

Feed-forward vs Recurrent Networks

- Feed-forward neural networks:

$$h = g(Vx + c)$$
$$\widehat{y} = Wh + b$$

- Recurrent neural networks (RNN) (Elman, 1990):

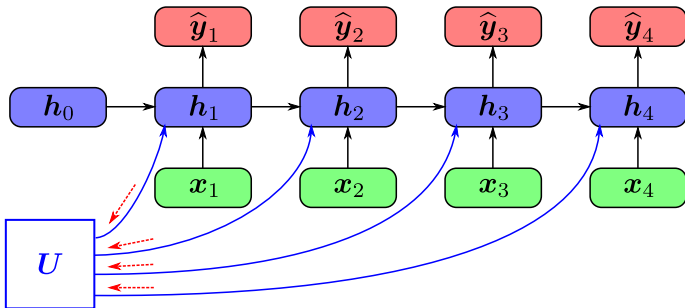$$h_t = g(Vx_t + Uh_{t-1} + c)$$
$$\widehat{y}_t = Wh_t + b$$
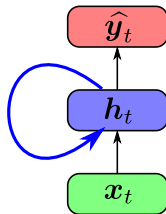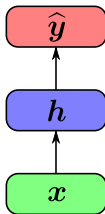
Unrolling the Graph
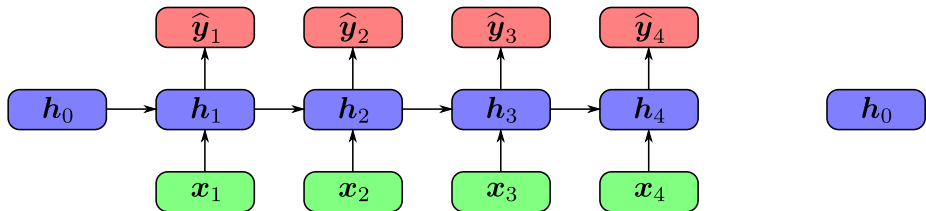
How do We Train the RNN Parameters?

- The unrolled graph is a well-formed (directed and acyclic) computation graph—we can use gradient backpropagation as usual

- Parameters are tied/shared accross "time"

- Derivatives are aggregated across time steps

- This is called backpropagation through time (BPTT).

Parameter Tying

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{U}} = \sum_{t=1}^{4} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{U}} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t}$$

- Same idea as when learning the filters in convolutional neural networks

What Can RNNs Be Used For?

We will see three applications of RNNs:

1 **Sequence generation:** generates symbols sequentially with an auto-regressive model (e.g. language modeling).

2 **Sequence tagging:** takes a sequence as input, and returns a label for every element in the sequence; e.g., *part of speech* (POS) tagging.

3 **Pooled classification:** takes a sequence as input, and returns a single label by pooling the RNN states; e.g., text classification.

Recap: Full History Model

$$\mathbb{P}(\text{START}, y_1, y_2, \ldots, y_L, \text{STOP}) = \prod_{t=1}^{L+1} \mathbb{P}(y_t | y_0, \ldots, y_{t-1})$$

• Generating each word depends on all the previous words.

• Huge expressive power!

• But, too many parameters to estimate! (quiz: how many?)

• ... thus, may not generalize well, specially for long sequences.

Can We Have Unlimited Memory?

- Markov models avoid the full history by having limited memory.

- Alternative: consider all the history, but compress it into a vector!

- RNNs do this!

Auto-Regressive Models

Key ideas:

- Feed the previous output as input to the current step:

$$x_t = y_{t-1}$$

- Maintain a state vector $\boldsymbol{h}_t$, which is a function of the previous state vector and the current input: this state *compresses* all the history!

$$\boldsymbol{h}_t = \boldsymbol{g}(\boldsymbol{V}x_t + \boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{c})$$

- Compute next output probability:

$$\mathbb{P}(y_t|y_0, \ldots, y_{t-1}) = \textbf{softmax}(\boldsymbol{W}\boldsymbol{h}_t + \boldsymbol{b})$$
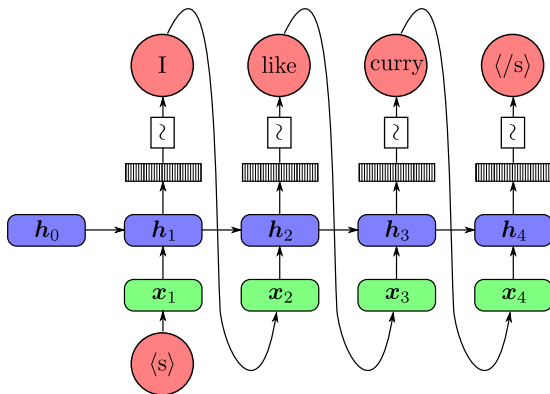
Let's see each of these steps in detail

Language Modeling: Large Softmax

- To generate text, each y_t is a word in the vocabulary

- Typically, large vocabulary; *e.g.*, $|V| = 100,000$

$$
\begin{aligned}
\boldsymbol{z}_t &= \boldsymbol{W}\boldsymbol{h}_t + \boldsymbol{b} \\
p(y_t = i) &= \frac{\exp((\boldsymbol{z}_t)_i)}{\sum_j \exp((\boldsymbol{z}_t)_j)} \\
&= \big(\textbf{softmax}(\boldsymbol{z})\big)_i
\end{aligned}
$$

the
a
in
Thai
food
green
red
curry
lemongrass
durian
jackfruit
chicken
rice
...

h
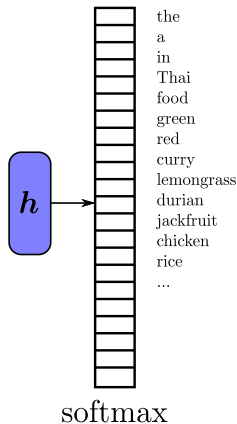
softmax

Language Modeling: Auto-Regression

$$\mathbb{P}(y_1, \ldots, y_L) = \mathbb{P}(y_1) \times \mathbb{P}(y_2 \mid y_1) \times \ldots \times \mathbb{P}(y_L \mid y_1, \ldots, y_{L-1})$$
$$= \text{softmax}(\boldsymbol{W}\boldsymbol{h}_1 + \boldsymbol{b}) \times \text{softmax}(\boldsymbol{W}\boldsymbol{h}_2 + \boldsymbol{b}) \times \ldots$$
$$\times \text{softmax}(\boldsymbol{W}\boldsymbol{h}_L + \boldsymbol{b})$$

Three Problems for Sequence-Generating RNNs

Algorithms are needed for:

- Sampling a sequence from the probability distribution defined by the RNN.

- Obtaining the most probable sequence.

- Training the RNN.

Sampling a Sequence

This is easy!

- Compute $\boldsymbol{h}_1$ from $\boldsymbol{x}_1 = \mathrm{START}$;

- Sample $\boldsymbol{y}_1 \sim \textbf{softmax}(\boldsymbol{W}\boldsymbol{h}_1 + \boldsymbol{b})$;

- Compute $\boldsymbol{h}_2$ from $\boldsymbol{h}_1$ and $\boldsymbol{x}_2 = \boldsymbol{y}_1$;

- Sample $\boldsymbol{y}_2 \sim \textbf{softmax}(\boldsymbol{W}\boldsymbol{h}_2 + \boldsymbol{b})$;

- And so on ...

Obtaining the Most Probable Sequence

Unfortunately, this is hard!

- It would require obtaining the $y_1, y_2, \ldots$ that jointly maximize the product $\mathbf{softmax}(\boldsymbol{W}\boldsymbol{h}_1 + \boldsymbol{b}) \times \mathbf{softmax}(\boldsymbol{W}\boldsymbol{h}_2 + \boldsymbol{b}) \times \ldots$

- Note that picking the best $\boldsymbol{y}_t$ greedily at each time step doesn't guarantee the best sequence (because $\mathbf{softmax}(\boldsymbol{W}\boldsymbol{h}_t + \boldsymbol{b})$ depends on $\boldsymbol{y}_{t-1}, \boldsymbol{y}_{t-2}, \ldots, \boldsymbol{y}_1$).

- This is rarely needed in language models. But it is important in conditional language modelling.

- More later, when discussing sequence-to-sequence models.

Training the RNN

- Sequence-generating RNNs are typically trained with maximum likelihood estimation.

- In other words, they are trained to minimize the log-loss (cross-entropy):

$$\mathcal{L}(\Theta, y_{1:L}) = -\frac{1}{L+1} \sum_{t=1}^{L+1} \log \mathbb{P}_\Theta(y_t \mid y_0, \ldots, y_{t-1})$$

- This is equivalent to minimizing perplexity $\exp(\mathcal{L}(\Theta, y_{1:L}))$

- Intuition: $-\log \mathbb{P}_\Theta(y_t \mid y_0, \ldots, y_{t-1})$

  measures how "perplexed" (or "surprised") the model is when the t-th word is revealed

Training the RNN
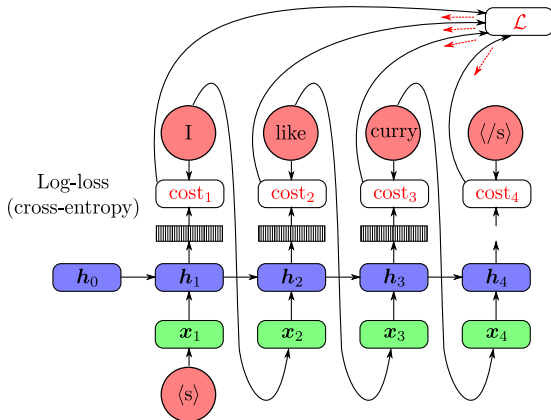
- Unlike Markov (*n*-gram) models, RNNs never forget!
- However, we will see they might have trouble learning to use their memories (more soon...)

Teacher Forcing and Exposure Bias

Note that conditioning is on the **true history**, not on the model's predictions! This is known as teacher forcing.

Teacher forcing cause exposure bias at run time: the model will have trouble recovering from mistakes early on, since it generates histories that it has never observed before.

How to improve this is a current area of research!

Character-Level Language Models

We can also have an RNN over characters instead of words!

Advantage: can generate any combination of characters, not just words in a closed vocabulary.

Disadvantage: need to remember further away in history!

A Character-Level RNN Generating Fake Shakespeare

PANDARUS: Alas, I think he shall be come approached and the day When little srain would be attain'd into being never fed, And who is but a chain and subjects of his death, I should not sleep.

Second Senator: They are away this miseries, produced upon my soul, Breaking and strongly should be buried, when I perish The earth and thoughts of many states.

DUKE VINCENTIO: Well, your wit is in the care of side and that.

Second Lord: They would be ruled after this chamber, and my fair nues begun out of the fact, to be conveyed, Whose noble souls I'll have the heart of the wars.

Clown: Come, sir, I will make did behold your worship.

VIOLA: I'll drink it.

(Credits: Andrej Karpathy)

A Char-Level RNN Generating a Math Paper

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*
Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

  (1) $\mathcal{F}$ is an algebraic space over S.
  (2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

(Credits: Andrej Karpathy)

A Char-Level RNN Generating C++ Code


```c
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```

(Credits: Andrej Karpathy)

Note: these examples are from 5 years ago; we now have much more impressive language generators (e.g. GPT-3 and ChatGPT)

Instead of RNNs, the most recent language generators use transformers

We will cover transformers in a later lecture!

What Can RNNs Be Used For?

We will see three applications of RNNs:

1. **Sequence generation:** generates symbols sequentially with an auto-regressive model; e.g., language modeling; ✓

2. **Sequence tagging:** takes a sequence as input, and returns a label for every element in the sequence; e.g., part of speech (POS) tagging;

3. **Pooled classification:** takes a sequence as input, and returns a single label by pooling the RNN states; e.g., sequence classification.

Sequence Tagging with RNNs

- In **sequence tagging**, we are given an input sequence $x_1, \ldots, x_L$

- The goal is to assign a tag to each element of the sequence, yielding an output sequence $y_1, \ldots, y_L$

- **Examples:** POS tagging, named entity recognition

- Differences with respect to sequence generation:

    - The input and output are distinct (no need for auto-regression)

    - The length of the output is known (same as that of the input)

Example: POS Tagging

- Map **sentences** to sequences of **part-of-speech tags.**

Time	flies	like	an	arrow	.
noun	verb	prep	det	noun	.

- Need to predict a morphological tag for each word of the sentence

- High correlation between adjacent words!
  (Ratnaparkhi, 1999; Brants, 2000; Toutanova et al., 2003)

An RNN-Based POS Tagger

- The inputs $x_1, \ldots, x_L \in \mathbb{R}^{E \times L}$ are word embeddings (found by looking up rows in an V-by-E embedding matrix, possibly pre-trained).

- As before, maintain a state vector $\boldsymbol{h}_t$, function of $\boldsymbol{h}_{t-1}$ and the current $\boldsymbol{x}_t$: this state compresses all the input history!

$$\boldsymbol{h}_t = \boldsymbol{g}(\boldsymbol{V}x_t + \boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{c}).$$

- A softmax output layer computes the probability of the current tag given the current and previous words:

$$\mathbb{P}(y_t | x_1, \ldots, x_t) = \textbf{softmax}(\boldsymbol{W}\boldsymbol{h}_t + \boldsymbol{b}).$$

An RNN-Based POS Tagger

This model can be improved:

- Use a bidirectional RNN to condition also on the following words: combine left-to-right and right-to-left RNNs (more later).

- Use a nested character-level CNN or RNN to obtain embeddings for unseen words.

Achieved state-of-the-art (SOTA) performance on the *Penn Treebank* and several other benchmarks (Ling et al., 2015; Wang et al., 2015)!

Bidirectional RNNs

- We can read a sequence from left to right to obtain a representation

- Or we can read it from right to left

- Or we can read it from both and combine the representations

- More later...
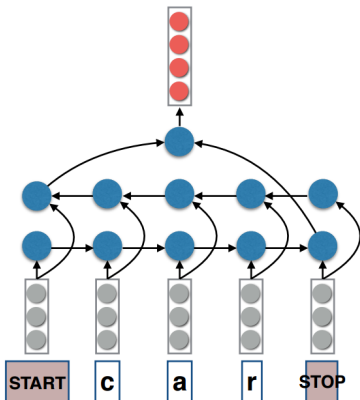
(Slide credit: Chris Dyer)

Example: Named Entity Recognition

From **sentences** extract **named entities.**

- Identify segments referring to entities (person, organization, location)
- Typically done with sequence models and **B-I-O** tagging:
  - ✓ B = Beginning; I = Inside; O = Other
  - ✓ PER = Person; LOC = Location; ORG = Organization

Example:

Louis	Elsevier	was	born	in	Leuven	.
B-PER	I-PER	O	O	O	B-LOC	.

RNN-Based NER

- The model we described for POS tagging works just as well for NER

- However, NER has constraints about tag transitions: e.g., we cannot have I-PER after B-LOC

- The RNN tagger model we described exploits input structure (via the states encoded in the recurrent layer) but lacks output structure...

What Can RNNs Be Used For?

We'll see three applications of RNNs:

1. **Sequence generation:** generates symbols sequentially with an auto-regressive model (e.g. language modeling)   ✓

2. **Sequence tagging:** takes a sequence as input, and returns a label for every element in the sequence (e.g., POS tagging)   ✓

3. **Pooled classification:** takes a sequence as input, and returns a single label by pooling the RNN states.

Pooled Classification

- What we have seen so far assumes we want to output a sequence of labels (either to generate tags a full sequence).

- What about predicting a single label for the whole sequence?

- We can still use an RNN to capture the input sequential structure.

- Just pool the RNNs states, *i.e.*, map them to a single vector.

- Use a single softmax to output the final label.

Pooling Strategies

- The simplest strategy is just to use the last RNN state.

- This state results from traversing the full sequence left-to-right, hence it has information about the whole sequence,

- **Disadvantage:** for long sequences, the influence the earliest words may vanish

- **Other pooling strategies:**

  ✓ Use a bidirectional RNN and combine both last states of the left-to-right and right-to-left RNN.

  ✓ Average pooling.

  ✓ Others...

Example: Sentiment Analysis
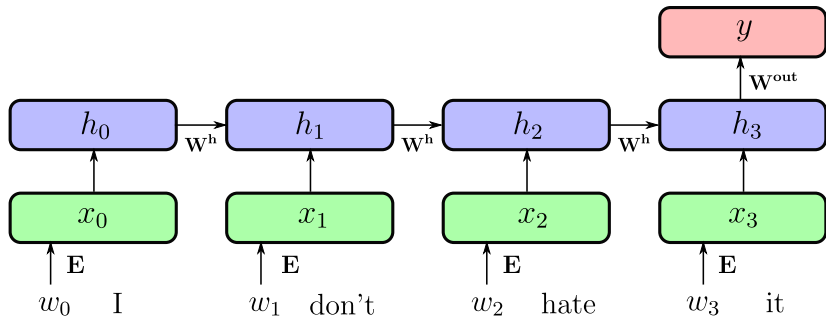
(Slide credit: Ollion & Grisel)

Recurrent Neural Networks are Very Versatile

one to one   one to many   many to one   many to many   many to many
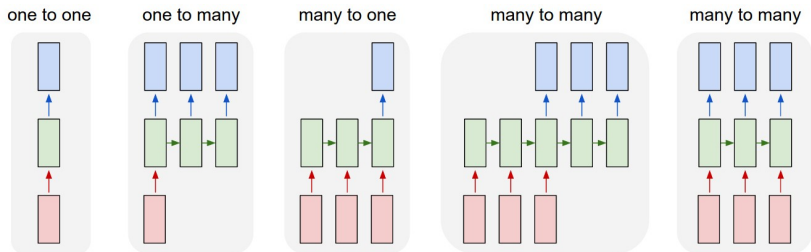
Check out Andrej Karpathy's blog post "The Unreasonable Effectiveness of Recurrent Neural Networks"
(http://karpathy.github.io/2015/05/21/rnn-effectiveness/).

Training the RNN: Backpropagation Through Time

What happens to the gradients as we go back in time?

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$
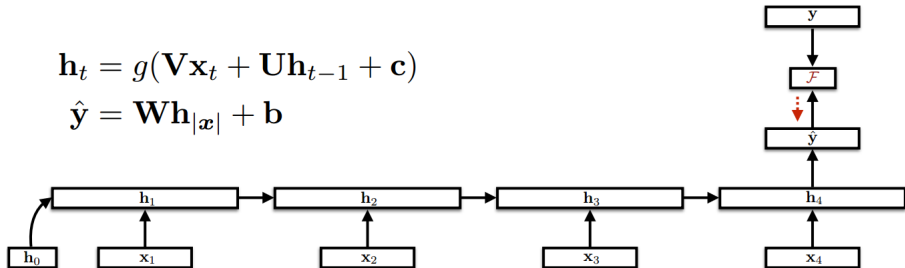
(Slide credit: Chris Dyer)

Backpropagation Through Time

What happens to the gradients as we go back in time?

$$\frac{\partial \mathcal{F}}{\partial \boldsymbol{h}_1} = \underbrace{\frac{\partial \boldsymbol{h}_2}{\partial \boldsymbol{h}_1}\frac{\partial \boldsymbol{h}_3}{\partial \boldsymbol{h}_2}\frac{\partial \boldsymbol{h}_4}{\partial \boldsymbol{h}_3}}_{\prod_{t=2}^{4}\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}}}\frac{\partial \widehat{\boldsymbol{y}}}{\partial \boldsymbol{h}_4}\frac{\partial \mathcal{F}}{\partial \widehat{\boldsymbol{y}}}$$

where

$$\prod_t \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} = \prod_t \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t}\frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{h}_{t-1}} = \prod_t \mathrm{Diag}(\boldsymbol{g}'(\boldsymbol{z}_t))\boldsymbol{U}$$

Three cases:

- largest eigenvalue of $\boldsymbol{U}$ exactly 1: gradient propagation is stable
- largest eigenvalue of $\boldsymbol{U} < 1$: gradient vanishes (exponential decay)
- largest eigenvalue of $\boldsymbol{U} > 1$: gradient explodes (exponential growth)

Vanishing and Exploding Gradients

- **Exploding gradients** can be dealt with by gradient clipping (truncating the gradient if it exceeds some magnitude)

- **Vanishing gradients** are more frequent and harder to deal with
    - In practice: long-range dependencies are difficult to learn

- **Solutions:**

    - Better optimizers (second order methods)

    - Normalization to keep the gradient norms stable across time

    - Clever initialization to start with good spectra (e.g., start with random orthonormal matrices)

    - Alternative parameterizations: LSTMs and GRUs

Gradient Clipping

- **Norm clipping:**

$$\tilde{\nabla} \leftarrow \left\{ \begin{array}{ll} \frac{c}{\|\nabla\|}\nabla & \text{if } \|\nabla\| \geq c \\ \nabla & \text{otherwise.} \end{array} \right.$$

- **Elementwise clipping:**

$$\tilde{\nabla}_i \leftarrow \min\{c, |\nabla_i|\} \times \text{sign}(\nabla_i), \ \forall i$$

Alternative RNNs

- **Gated recurrent unit** (GRU)
  (Cho et al., 2014)

- **Long short-term memorie** (LSTM)
  (Hochreiter and Schmidhuber, 1997)

Intuition: instead of multiplying across time (which leads to exponential growth), we want the error to be approximately constant

They solve the vanishing gradient problem, but still have exploding gradients (still need gradient clipping)

Gated Recurrent Units (Cho et al., 2014)

- Recall the problem: the error must backpropagate through all the intermediate nodes:
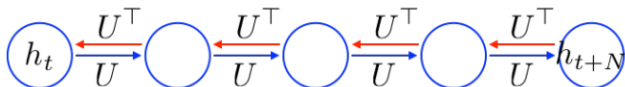
- **Idea:** create some kind of shortcut connections:
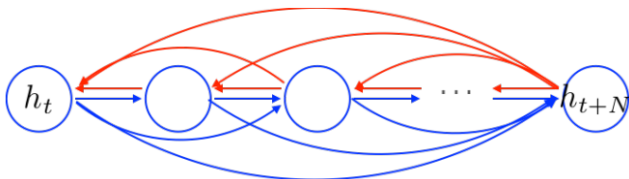
(Image credit: Thang Luong, Kyunghyun Cho, Chris Manning)

- Create adaptive shortcuts controlled by special gates

Gated Recurrent Units (Cho et al., 2014)
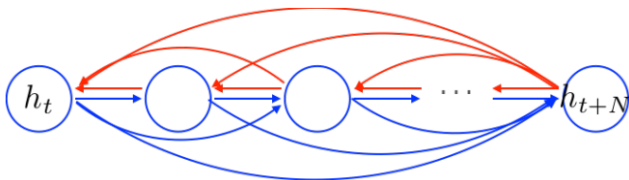
(Image credit: Thang Luong, Kyunghyun Cho, Chris Manning)

$$\boldsymbol{h}_t = \boldsymbol{u}_t \odot \tilde{\boldsymbol{h}}_t + (1 - \boldsymbol{u}_t) \odot \boldsymbol{h}_{t-1}$$

- **Candidate update:** $\tilde{\boldsymbol{h}}_t = \boldsymbol{g}(\boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{U}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \boldsymbol{b})$
- **Reset gate:** $\boldsymbol{r}_t = \sigma(\boldsymbol{V}_r\boldsymbol{x}_t + \boldsymbol{U}_r\boldsymbol{h}_{t-1} + \boldsymbol{b}_r)$
- **Update gate:** $\boldsymbol{u}_t = \sigma(\boldsymbol{V}_u\boldsymbol{x}_t + \boldsymbol{U}_u\boldsymbol{h}_{t-1} + \boldsymbol{b}_u)$

Long Short-Term Memories
(Hochreiter and Schmidhuber, 1997)

- **Key idea:** use memory cells c_t
- To avoid the multiplicative effect, flow information *additively* through these cells
- Control the flow with special input, forget, and output gates
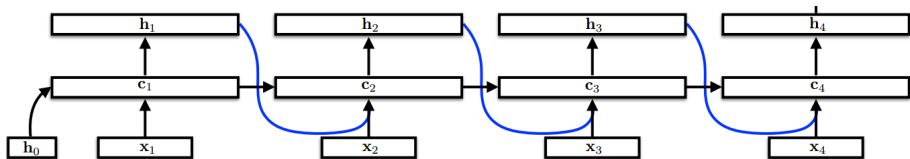
(Image credit: Chris Dyer)

Long Short-Term Memories
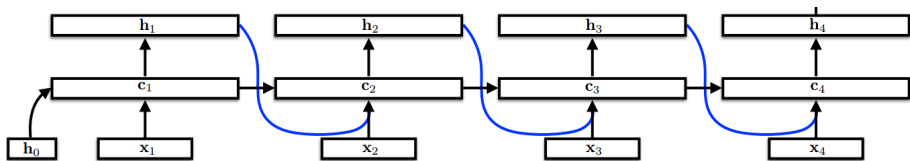
(Image credit: Chris Dyer)

$$c_t = f_t \odot c_{t-1} + i_t \odot g(Vx_t + Uh_{t-1} + b), \qquad h_t = o_t \odot g(c_t)$$

- **Forget gate:** $f_t = \sigma(V_f x_t + U_f h_{t-1} + b_f)$
- **Input gate:** $i_t = \sigma(V_i x_t + U_i h_{t-1} + b_i)$
- **Output gate:** $o_t = \sigma(V_o x_t + U_o h_{t-1} + b_o)$

Long Short-Term Memories
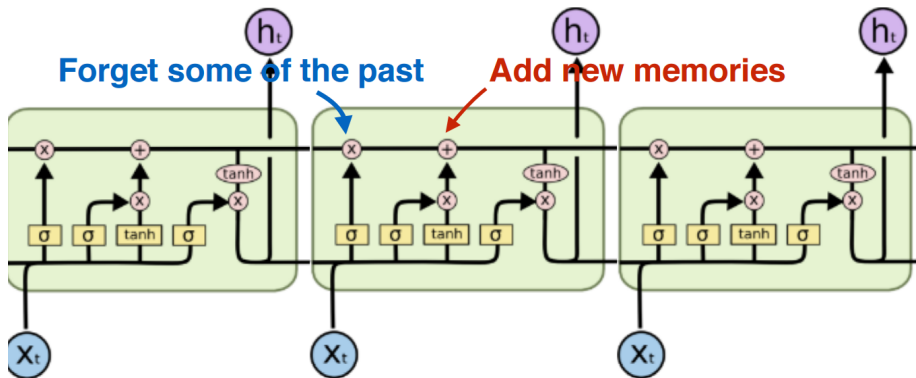
(Slide credit: Christopher Olah)

Bidirectional LSTMs

- Same thing as a Bidirectional RNN, but using LSTM units instead of vanilla RNN units.
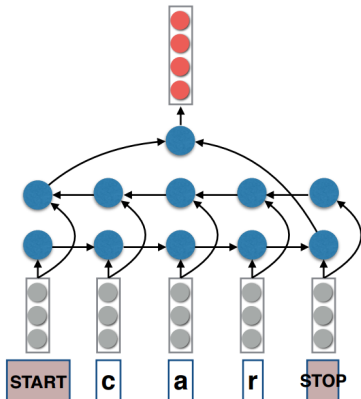
(Slide credit: Chris Dyer)

LSTMs and BILSTMs: Some Success Stories

- Time series prediction (Schmidhuber et al., 2005)

- Speech recognition (Graves et al., 2013)

- Named entity recognition (Lample et al., 2016)

- Machine translation (Sutskever et al., 2014)

- ELMo (deep contextual) word representations (Peters et al., 2018)

- ... and many others.

Summary

- Better gradient propagation is possible if we use additive rather than multiplicative/highly non-linear recurrent dynamics

- Recurrent architectures are an active area of research (but LSTMs are hard to beat)

- Other variants of LSTMs exist which tie/simplify some of the gates

- Extensions exist for *non-sequential* structured inputs/outputs (e.g. trees): recursive neural networks (Socher et al., 2011), PixelRNN (Oord et al., 2016)

From Sequences to Trees

- So far we've talked about recurrent neural networks, which are designed to capture sequential structure

- What about other kinds of structure? For example, trees?

- It is also possible to tackle these structures with recursive computation, via recursive neural networks.

Recursive Neural Networks

- Proposed by Socher et al. (2011) for parsing images and text

- Assume a binary tree (each node except the leaves has two children)

- Propagate states bottom-up in the tree, computing the parent state $\boldsymbol{p}$ from the children states $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$:

$$\boldsymbol{p} = \tanh\left(\boldsymbol{W}\left[\begin{array}{c} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \end{array} + \boldsymbol{b}\right]\right)$$

- Use the same parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ at all nodes

- Can compute scores at the root or at each node by appending a softmax output layer at these nodes.

Compositionality in Text

Uses a recurrent net to build a bottom-up parse tree for a sentence.
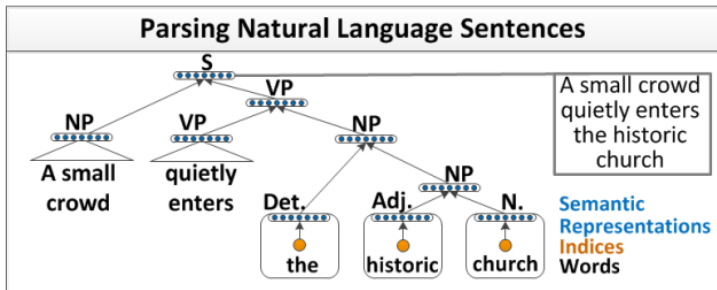
(Credits: Socher et al. (2011))

Compositionality in Images

Same idea for images.
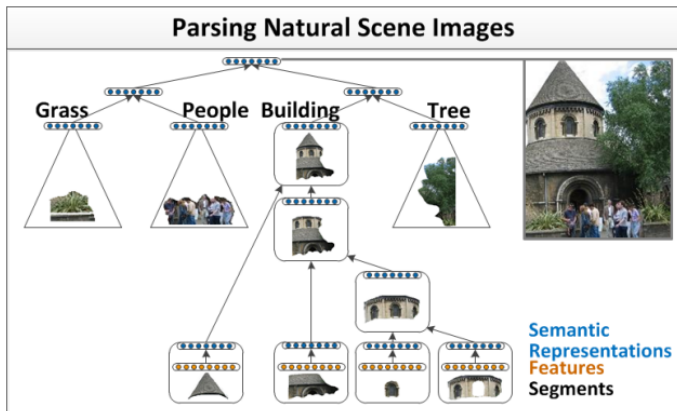
(Credits: Socher et al. (2011))

Tree-LSTMs

- Extend recursive neural networks the same way LSTMs extend RNNs, with a few more gates to account for the left and right child.

- Extensions exist for non-binary trees.

Fine-Grained Sentiment Analysis
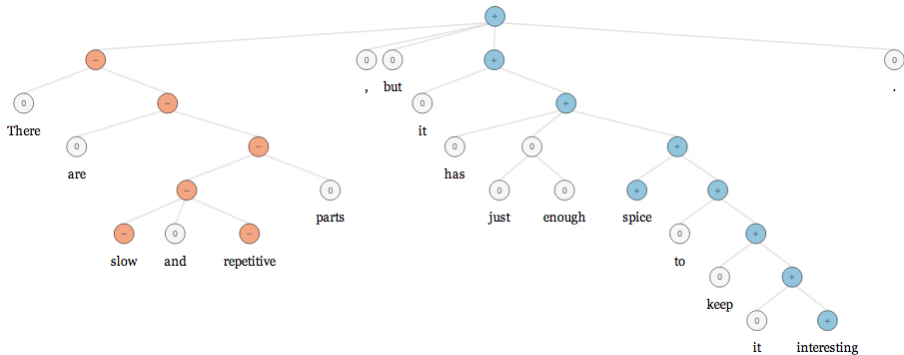
(Taken from Stanford Sentiment Treebank.)

What about Images?

- While sequences are 1D, images are 2D.

- PixelRNNs are 2D extensions of RNNs.

- They can be used as auto-regressive models to generate images, by generating pixels in a particular order, conditioning on neighboring pixels.

- Several variants...

RNNs for Generating Images

- Input-to-state and state-to-state mappings for PixelCNN and two PixelRNN models (Oord et al., 2016):
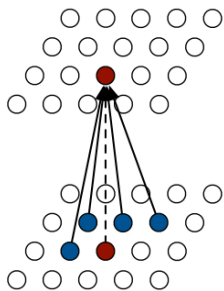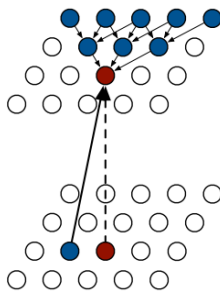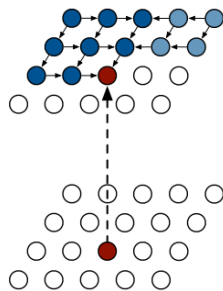
PixelCNN   Row LSTM   Diagonal BiLSTM

RNNs for Generating Images
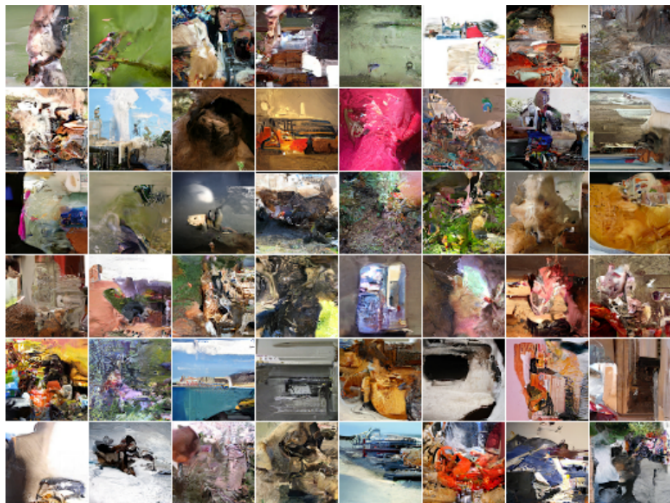
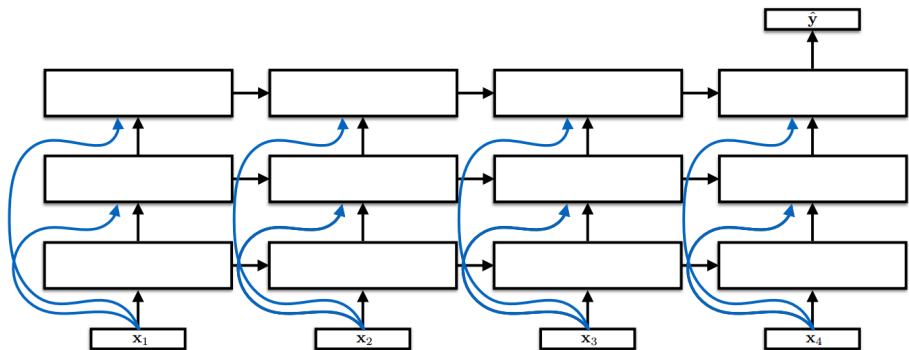(Oord et al., 2016)

Even More General: Graph LSTMs
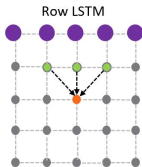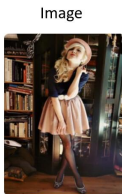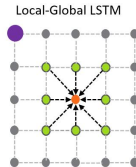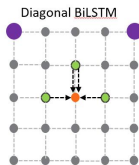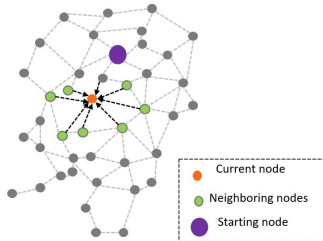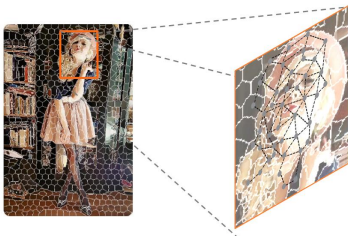
(Credits: Xiaodan Liang)

More Tricks of the Trade

- Depth
- Dropout
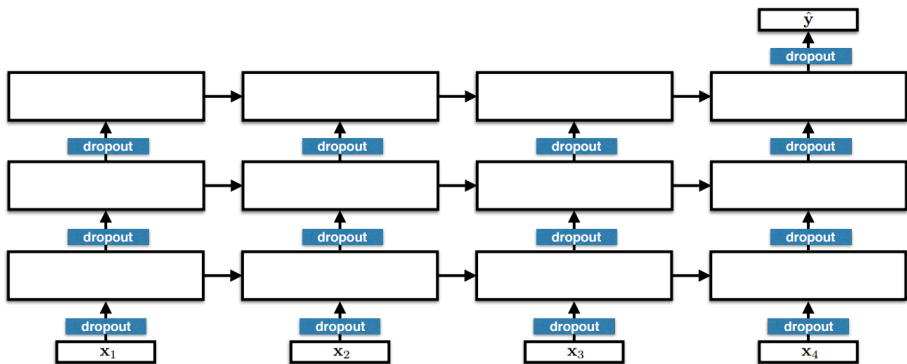- Implementation Tricks
- Mini-batching

Deep RNNs/LSTMs/GRUs

- Depth in recurrent layers helps in practice (2–8 layers seem to be standard)
- Input connections may or may not be used

(Slide credit: Chris Dyer)

Dropout in Deep RNNs/LSTMs/GRUs

- Apply dropout between layers, but not on the recurrent connections
- ... Or use the same mask for all recurrent connections (Gal and Ghahramani, 2015)

(Slide credit: Chris Dyer)

Implementation Tricks

For speed:

- Use diagonal matrices instead of full matrices (esp. for gates)
- Concatenate parameter matrices for all gates and do a single matrix-vector multiplication
- Use optimized implementations (from NVIDIA)
- Use GRUs or reduced-gate variant of LSTMs

For learning speed and performance:

- Initialize so that the bias on the forget gate is large (intuitively: at the beginning of training, the signal from the past is unreliable)
- Use random orthogonal matrices to initialize the square matrices

Mini-Batching

- RNNs, LSTMs, GRUs all consist of many element-wise operations (addition, multiplication, nonlinearities), and lots of matrix-vector products
- Mini-batching: convert many matrix-vector products into a single matrix-matrix multiplication
- Batch across instances, not across time
- The challenge with working with mini batches of sequences is... sequences are of different lengths (we've seen this when talking about convolutional nets)
- This usually means you bucket training instances based on similar lengths, and pad with zeros
- Be careful when padding not to back propagate a non-zero value!

Conclusions

Recurrent neural networks allow to take advantage of sequential input structure

They can be used to generate, tag, and classify sequences, and are trained with backpropagation through time
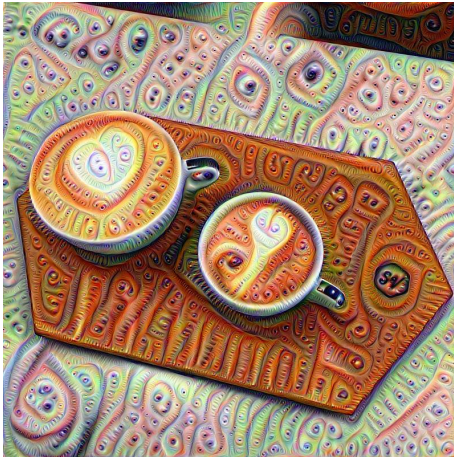
Vanilla RNNs suffer from vanishing and exploding gradients

LSTMs and other gated units are more complex variants of RNNs that avoid vanishing gradients

They can be extended to other structures like trees, images, and graphs.

Thank you!

Questions?

References I

Brants, T. (2000). Tnt: a statistical part-of-speech tagger. In *Proceedings of the sixth conference on Applied natural language processing*, pages 224–231. Association for Computational Linguistics.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. In *Proc. of Empirical Methods in Natural Language Processing*.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

Gal, Y. and Ghahramani, Z. (2015). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *ICASSP*, pages 6645–6649. IEEE.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. In *Proc. of the Annual Meeting of the North-American Chapter of the Association for Computational Linguistics*.

Ling, W., Luís, T., Marujo, L., Astudillo, R. F., Amir, S., Dyer, C., Black, A. W., and Trancoso, I. (2015). Finding function in form: Compositional character models for open vocabulary word representation. In *Proc. of Empirical Methods in Natural Language Processing*.

Oord, A. v. d., Kalchbrenner, N., and Kavukcuoglu, K. (2016). Pixel Recurrent Neural Networks. In *Proc. of the International Conference on Machine Learning*.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Ratinov, L. and Roth, D. (2009). Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics.

Ratnaparkhi, A. (1999). Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1):151–175.

Schmidhuber, J., Wierstra, D., and Gomez, F. J. (2005). Evolino: Hybrid neuroevolution/optimal linear search for sequence prediction. In *Proceedings of the 19th International Joint Conferenceon Artificial Intelligence (IJCAI)*.

References II

Socher, R., Lin, C. C., Manning, C., and Ng, A. Y. (2011). Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112.

Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proc. of the North American Chapter of the Association for Computational Linguistics*, pages 173–180.

Wang, P., Qian, Y., Soong, F. K., He, L., and Zhao, H. (2015). Part-of-speech tagging with bidirectional long short-term memory recurrent neural network. *arXiv preprint arXiv:1510.06168*.

Zhang, T. and Johnson, D. (2003). A robust risk minimization based named entity recognition system. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 204–207. Association for Computational Linguistics.

Lecture 9: Sequence-to-Sequence Models

André Martins, Francisco Melo, Mário Figueiredo

Deep Learning Course, Winter 2022-2023

Today's Roadmap

Lecture 8 focused on sequence tagging and sequence generation. Today we focus on sequence-to-sequence models.

- Machine translation

- Sequence vector representation

- Encoder-decoder architecture

- Sequence matrix representation

- Attention mechanism

- Encoder-decoder with attention

Sequence-to-Sequence

Sequence-to-sequence models map a source sequence (of arbitrary length) into a target sequence (also of arbitrary length)

This differs from sequence tagging, where the two sequences are of the same length

Example: Machine Translation

Goal: translate a **source sentence** x in one language into a **target sentence** y in another language.

Example (Portuguese to English):

x: *"A ilha de Utopia tem 200 milhas de diâmetro na parte central."*

$$\downarrow$$

y: *"The island of Utopia is two hundred miles across in the middle part."*

1950s: Early Machine Translation

(Source: https://youtu.be/K-HfpsHPmvw)

- MT research began in early 1950s
- Mostly Russian-English (motivated by the Cold War!)
- Systems were mostly rule-based, using a bilingual dictionary

Noisy Channel Model (Shannon and Weaver, 1949)

"When I look at an article in Russian, I say: 'This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode.'"

A very simple model: builds a generative story that works "backwards" (flips source and target, via Bayes law)

Yet, the dominant paradigm in MT for several decades (until 2014)

In 2014: neural machine translation (later)

1990s-2010s: Statistical Machine Translation

Goal: find the "best" (most probable) English sentence y, given the Russian sentence x

$$\widehat{y} = \arg \max_y \mathbb{P}(y \mid x)$$

Key idea: use Bayes' law to break this down into two components:

$$\widehat{y} = \arg \max_y \frac{\mathbb{P}(x \mid y)\,\mathbb{P}(y)}{\mathbb{P}(x)} = \arg \max_y \mathbb{P}(x \mid y)\,\mathbb{P}(y)$$

- **Translation model:** models how words/phrases are translated (learnt from **parallel** data)

- **Language model:** models how to generate fluent English (learn from **monolingual** data)

How to Learn the Language Model?

- Need large amounts of monolingual data
  (easy to get for most languages).

- How to learn a language model from these data?

- We covered language models in previous lectures:

    - Markov models (maybe with *smoothing*)

    - Neural language models

    - ...

- Pick your favorite!

How to Learn the Translation Model?

Need large amounts of parallel data!

(*e.g.*, pairs of human-translated Chinese-English sentence pairs.)

似乎格式有問題
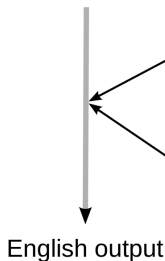
translation model

language model

English output

parallel corpus

网站资讯分析网数 The corporation has been estim
据显示的主域名为 to run more than one million pag
全世界访问量最高 in data centers around the world
的站点除此之外搜 to process over one billion searc
索在其他国家或地 requests and about twenty-four i
区域名下的多个站 of user-generated data each dat
点等等及旗下的等 December 2012 Alexa listed as

monolingual corpus

started functioning in 1928 and established the tradition of
large exhibitions and trade fairs held in Brno, and nowadays
also ranks among the sights of the city. Brno is also
known for hosting big motorbike and other races on the
Masaryk Circuit, a tradition established in 1930 in which
the Road Racing World Championship Grand Prix is
one of the most prestigious races. Another notable cultural
tradition is an international fireworks competition.

Rosetta Stone
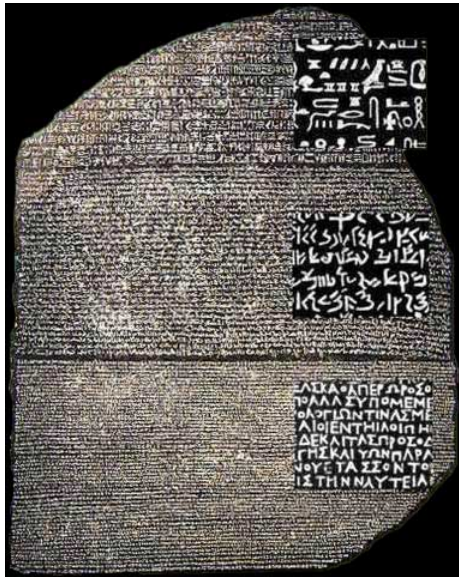
- (Re-)discovered in 1799 near Alexandria

- Parallel corpora: ancient Egyptian, demotic Egyptian, ancient Greek

Europarl

- Proceedings from the European parliament sessions, translated into all EU official languages
- Around 1M parallel sentences for some language pairs
- Other corpora: Hansard, MultiUN, News Commentary, Wikipedia, OpenSubtitles, Paracrawl, ...

1990s: IBM Models for Statistical MT

- How to learn the translation model $\mathbb{P}(x \mid y)$?

- Assume we have enough parallel training data.

- Break it down further: consider instead

$$\mathbb{P}(x, \boldsymbol{a} \mid y),$$

  where $\boldsymbol{a}$ are word alignments, i.e., word-level correspondences between sentence x and sentence y

- Word alignments are generally a latent/missing variable at training time, and need to be marginalized over at test time,

$$\mathbb{P}(x \mid y) = \sum_{\boldsymbol{a}} \mathbb{P}(x, \boldsymbol{a} \mid y),$$

Word Alignments

Example for English-French:
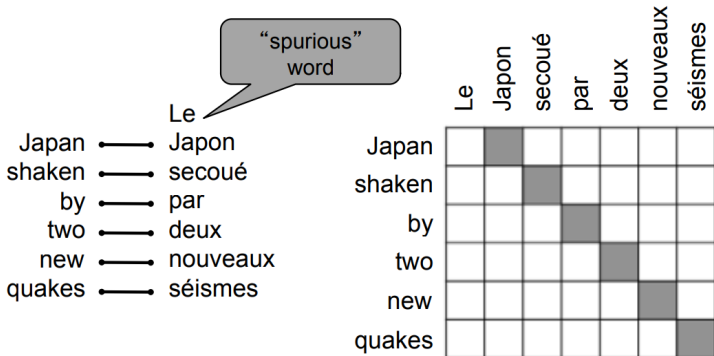
Some words may be unaligned (no counterpart in the other language)!

Word Alignments

Alignment can be one-to-many (word fertility):

Word Alignments

Alignment can be many-to-one:

Word Alignments

Alignment can be many-to-many (phrase-level): phrase-based MT:
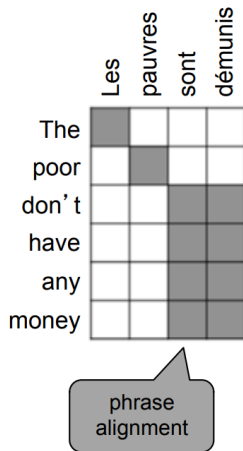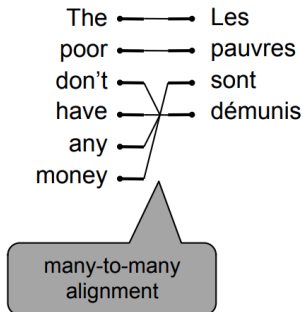
1990s: IBM Models for Statistical MT

- How to learn the translation model $\mathbb{P}(x \mid y)$?

- ...assuming we have enough parallel training data.

- Break it down further: consider instead

$$\mathbb{P}(x, \boldsymbol{a} \mid y).$$

- Learn $\mathbb{P}(x, \boldsymbol{a} \mid y)$ as a combination of several factors:

  - Probability of particular words aligning (co-occurrence, relative position, etc.)

  - Probability of words having a particular fertility

  - ...

- This has lead to IBM models 1, 2, 3, 4, ...

1990s: IBM Models for Statistical MT

- To search the best translation, we need to solve

$$\widehat{y} = \arg\max_y \sum_{\boldsymbol{a}} \mathbb{P}(x, \boldsymbol{a} \mid y) \, \mathbb{P}(y),$$

combining the translation and language models.

- Enumerating all possible hypothesis and alignments is intractable.

- Typical approach: heuristic search to gradually build the translation, discarding hypotheses that are too low probability.

Searching for the Best Translation

(Slide credit: https://web.stanford.edu/class/cs224n/lectures/lecture10.pdf)
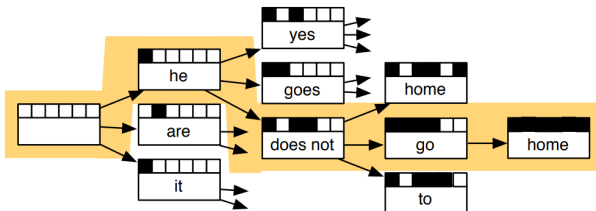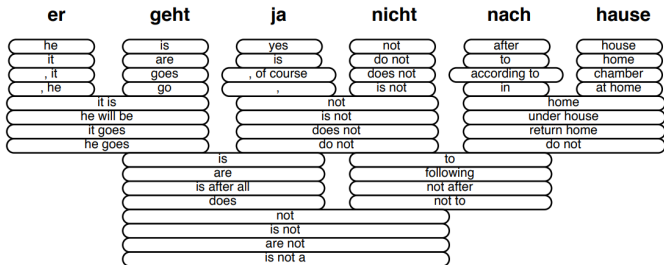
Summarizing: Statistical Machine Translation

We only glimpsed the tip of the iceberg: SMT is (was?) a huge research field.

- The best systems are extremely complex;

- Usually a big pipeline, with many separately-designed subcomponents (translation and language model are only two examples);

- Lots of feature engineering;

- System design is very language-dependent;

- Requires compiling and maintaining resources (e.g., phrase tables);

- Models are storage/memory hungry;

- Lots of human effort to maintain.

2014: Neural Machine Translation
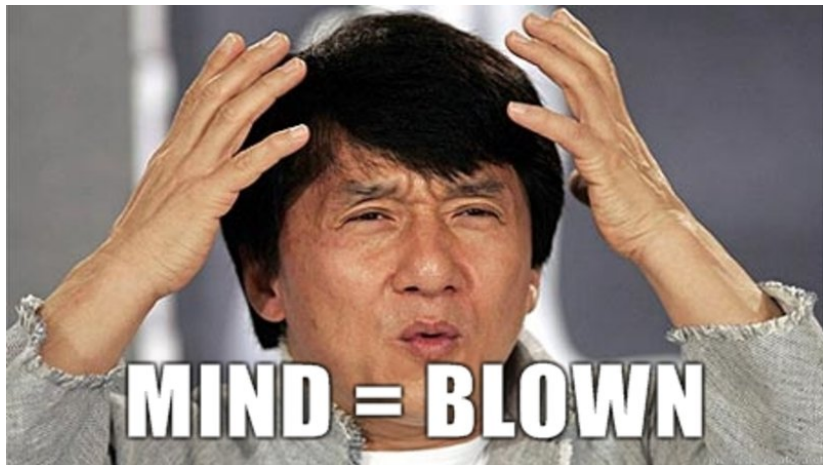
What is Neural Machine Translation (NMT)?

- NMT = MT with a single neural network

- End-to-end training with parallel data (no more complex pipelines!)

- The underlying architecture is an encoder-decoder (also called a **sequence-to-sequence model**)

- In fact, NMT is also statistical; however, historically, "statistical MT" refers to non-neural models, and NMT to NN-based models.
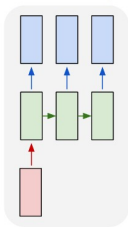
Recap: Recurrent Neural Networks

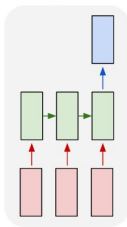Lecture 9 covered RNNs and showed they can have several uses...
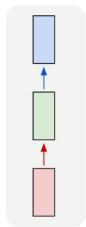
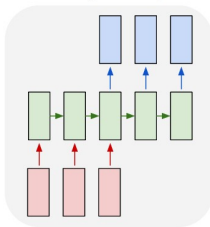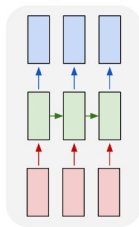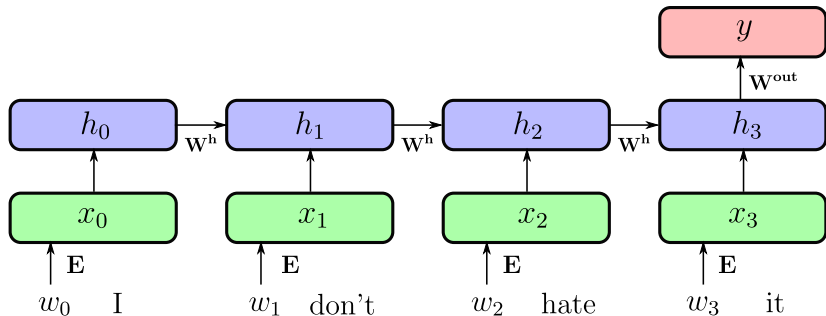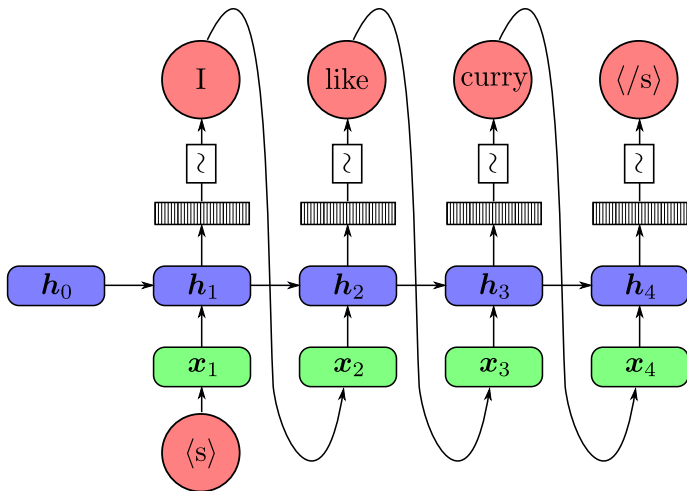# Recap: RNNs for Pooled Classification

(Slide credit: Ollion & Grisel)

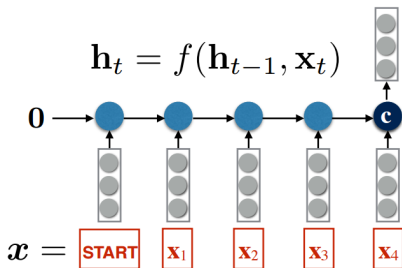# Recap: Auto-Regressive RNNs for Sequence Generation

Sequence-to-Sequence Learning
(Cho et al., 2014; Sutskever et al., 2014)

- Can we put the two things together?

- Yes!

  1 Encoder RNN encodes source sentence, generating a vector state

  2 Decoder RNN generates target sentence conditioned on vector state.

Encode a Sequence as a Vector

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

$$x = \boxed{\text{START}} \quad \mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \quad \mathbf{x}_4$$

(Slide credit: Chris Dyer)
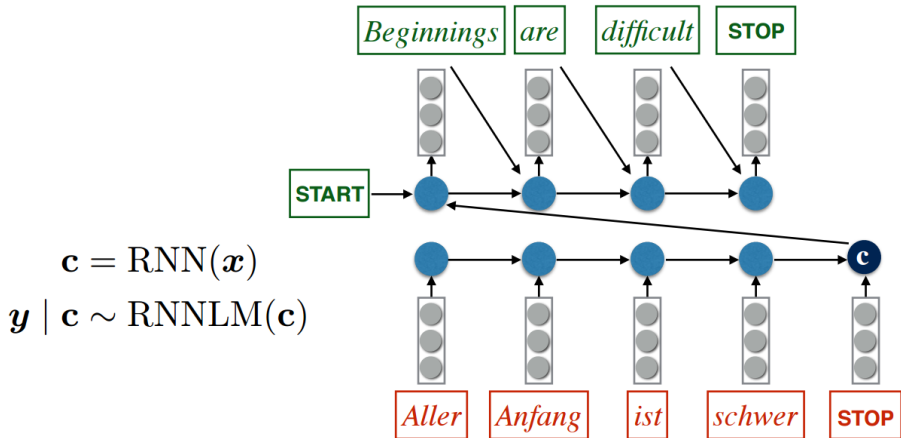
What is a vector representation of a sequence x?

$$c = \textbf{RNN}(x)$$

What is the probability of a sequence $y \mid x$?

$$y \mid x \sim \textbf{RNNLM}(c)$$

Encoder-Decoder Architecture

$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$
$$\boldsymbol{y} \mid \mathbf{c} \sim \mathrm{RNNLM}(\mathbf{c})$$
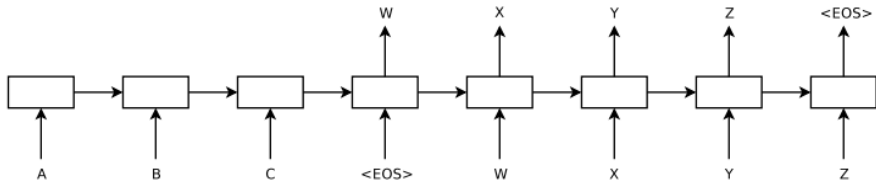
(Slide credit: Chris Dyer)

Encoder-Decoder Architecture

Another way of depicting it (from Sutskever et al. (2014)):

Some Problems

- If the source sentence is long, the encoder may forget the initial words and the translation will be degraded
  - Cheap solution: reverse the source sentence.

- The decoder does greedy search—this leads to error propagation
  - Solution: beam search.

Beam Search

Ideally we want to find the target sentence y that maximizes

$$\hat{y} = \arg\max_y \mathbb{P}(y \mid x) = \arg\max_{y_1,\ldots,y_L} \mathbb{P}(y \mid x) = \prod_{i=1}^{L} \mathbb{P}(y_i \mid y_{1:i-1}, x)$$
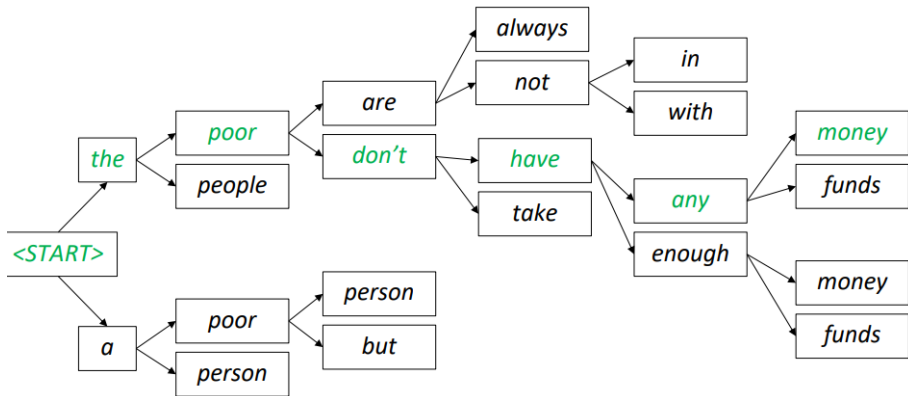
Enumerating all y is intractable!

Beam Search:

- Approximate search strategy
- At each decoder step, keep track of the k most probable partial translations; discard the rest
- k is the beam size
- For $k = 1$, we have greedy search.

Beam Search

(Slide credits: Chris Manning)

Beam Search

- A little better than strictly greedy search, but still greedy

- Linear runtime as a function of beam size k: speed/accuracy trade-off

- In practice: beam sizes $\sim$ 4–12

Some Additional Tricks

From Sutskever et al. (2014):

- Deep LSTMs
- Reversing the source sentence

| Method	test BLEU score (ntst14)
Bahdanau et al. [2]	28.45
Baseline System [29]	33.30
Single forward LSTM, beam size 12	26.17
Single reversed LSTM, beam size 12	30.59
Ensemble of 5 reversed LSTMs, beam size 1	33.00
Ensemble of 2 reversed LSTMs, beam size 12	33.27
Ensemble of 5 reversed LSTMs, beam size 2	34.50
Ensemble of 5 reversed LSTMs, beam size 12	**34.81**

At run time:

- Beam search;
- Ensembling: combine N independently trained models and obtaining a "consensus" (always helps!).

End-to-End Neural Machine Translation

- Previous statistical machine translation models were complicated pipelines (word alignments $\rightarrow$ phrase table extraction $\rightarrow$ language model $\rightarrow$ decoding a phrase lattice);

- Alternative: end-to-end NMT uses a simple encoder-decoder;

- Doesn't quite work yet, but gets close to top performance
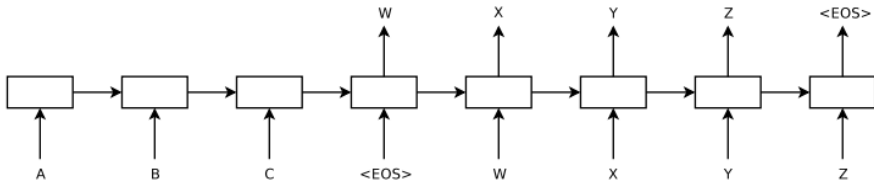
Caption Generation

Works for image inputs too!
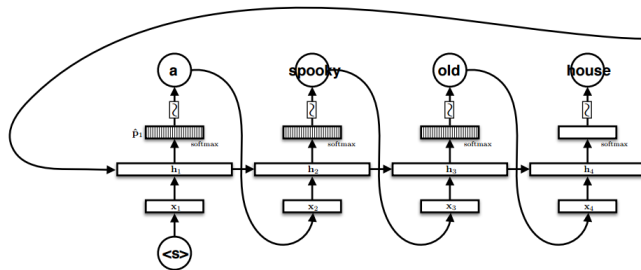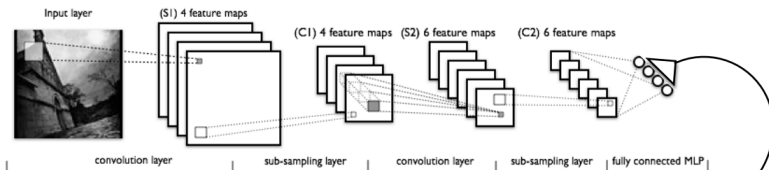
(Slide credit: Chris Dyer)

Progress in Machine Translation
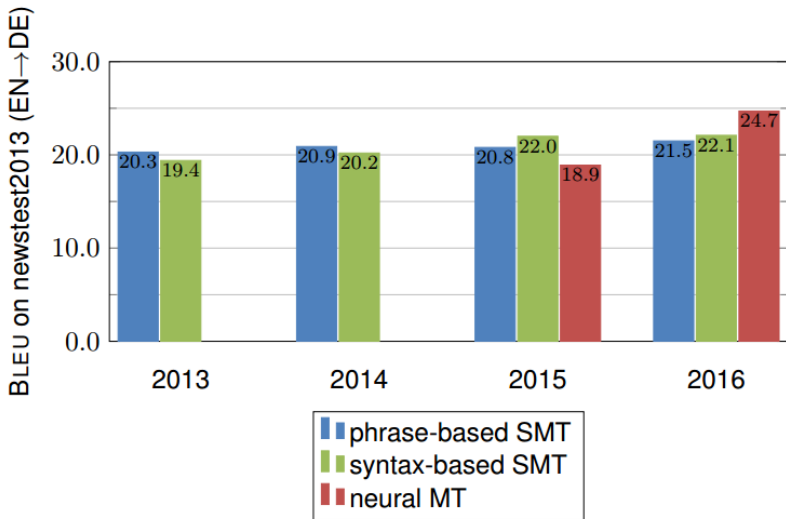
Slide credit: Rico Sennrich

NMT: A Success Story

- Neural MT went from a fringe research activity in 2014 to the leading standard method in 2016

    - 2014: First seq2seq paper published

    - 2016: Google Translate switches from SMT to NMT

- This is amazing!

- SMT systems, built by hundreds of engineers over many years, outperformed by NMT trained by a handful of engineers in a few months.

So Is Machine Translation Solved?

No!

Many difficulties and open research fronts remain:

- Out-of-vocabulary words

- Domain mismatch between train and test data

- Low-resource language pairs

- Specific domains (medical, scientific, legal, ...)

- Maintaining context over longer text (coming next!)

Limitations

A possible conceptual problem:

- Sentences have unbounded lengths
- Vectors have finite capacity

"You can't cram the meaning of a whole %&$# sentence into a single $&# vector!" (Ray Mooney)

A possible practical problem:

- Distance between "translations" and their sources are large—can LSTMs help?

Encode Sentences as Matrices, Not Vectors

Problem with the fixed-size vector model:

- Sentences are of different sizes but vectors are of the same size
- Bottleneck problem: a single vector represents the whole source sentence!

Solution: use matrices instead!

- Fixed number of rows, but number of columns depends on the number of words
- Then, before generating each word in the decoder, use an attention mechanism to condition on the relevant source words only

How to Encode a Sentence as a Matrix?

First shot: define the sentence words' vectors as the columns
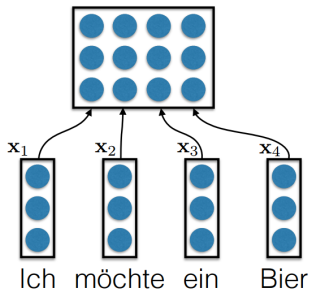
(Image credit: Chris Dyer)

- Not very effective: word vectors carry no context information

How to Encode a Sentence as a Matrix?

Other strategies:

- Convolutional NNs (Kalchbrenner et al., 2014) capture context;

- **Typical choice:** bidirectional LSTMs (Bahdanau et al., 2015);

- Later: transformer networks (Vaswani et al., 2017).

Bidirectional LSTM Encoder
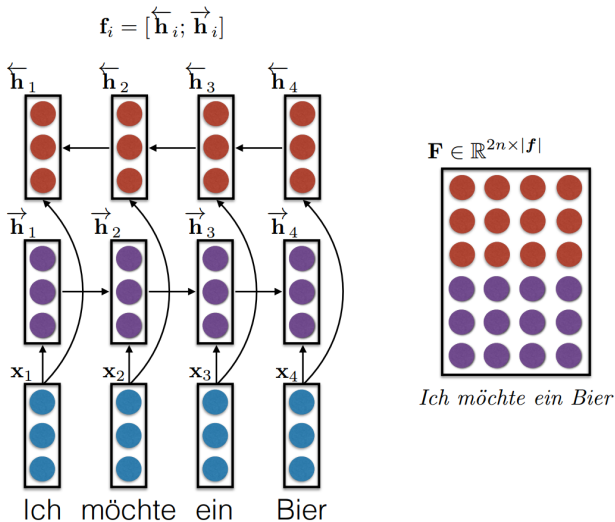
(Slide credit: Chris Dyer)

Generation from Matrices

- We now have a matrix $\boldsymbol{F}$ representing the input.

- How to generate from it?

- **Answer:** use attention! (Bahdanau et al., 2015)

- Attention is the neural counterpart of word alignments.

Generation from Matrices with Attention

- Generate the output sentence word by word using an RNN;

- At each output position t, the RNN receives two inputs:

  - a fixed-size vector embedding of the previous output symbol y_{t-1}

  - a fixed-size vector encoding a "view" of the input matrix $\boldsymbol{F}$, via a weighted sum of its columns (i.e., words): $\boldsymbol{Fa}_t$

- The input columns weighting at each time-step ($\boldsymbol{a}_t$) is called the attention distribution.

Attention Mechanism (Bahdanau et al., 2015)

Let $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots$ be the states produced by the decoder RNN

When predicting the t-th target word:

1. Compute "similarity" with each of the source words:

$$z_{t,i} = \boldsymbol{v}^T \boldsymbol{g}(\boldsymbol{W}\boldsymbol{h}_i + \boldsymbol{U}\boldsymbol{s}_{t-1} + \boldsymbol{b}), \quad \text{for } i = 1, ..., L$$

where $\boldsymbol{h}_i$ is the i-th column of $\boldsymbol{F}$ (representation of the i-th source word) and $\boldsymbol{v}$, $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{b}$ are model parameters

2. From $\boldsymbol{z}_t = (z_{t,1}, \ldots, z_{t,L})$ compute attention $\boldsymbol{a}_t = \textbf{softmax}(\boldsymbol{z}_t)$

3. Use attention to compute input conditioning state $\boldsymbol{c}_t = \boldsymbol{F}\boldsymbol{a}_t$

4. Update RNN state $\boldsymbol{s_t}$ based on $\boldsymbol{s}_{t-1}, y_{t-1}, \boldsymbol{c}_t$

5. Predict $y_t \sim p(y_t \mid \boldsymbol{s}_t)$

Encoder-Decoder with Attention
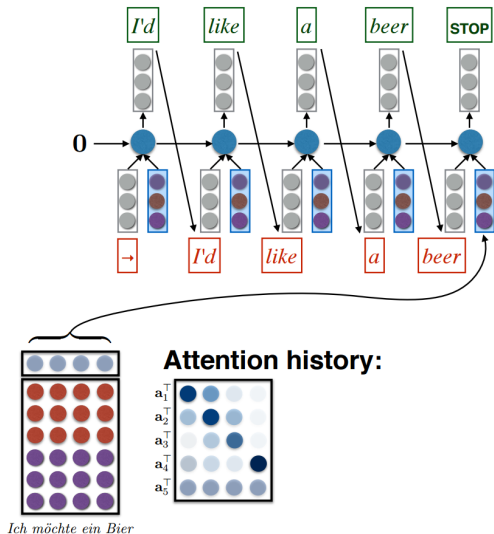
(Slide credit: Chris Dyer)

Putting It All Together

obtain input matrix $\boldsymbol{F}$ with a bidirectional LSTM
$t = 0$, $y_0 = \text{START}$ (the start symbol)
$\boldsymbol{s}_0 = \boldsymbol{w}$ (learned initial state)
repeat
  $t = t + 1$
  $\boldsymbol{z}_t = \boldsymbol{v}^T \boldsymbol{g}(\boldsymbol{WF} + \boldsymbol{U}\boldsymbol{s}_{t-1} + \boldsymbol{b})$
  compute attention $\boldsymbol{a}_t = \textbf{softmax}(\boldsymbol{z}_t)$
  compute input conditioning state $\boldsymbol{c}_t = \boldsymbol{F}\boldsymbol{a}_t$
  $\boldsymbol{s}_t = \textbf{RNN}(\boldsymbol{s}_{t-1}, [\boldsymbol{E}(y_{t-1}), \boldsymbol{c}_t])$
  $y_t \sim \textbf{softmax}(\boldsymbol{P}\boldsymbol{s}_t + \boldsymbol{b})$
until $y_t \neq \text{STOP}$

Notice that each y_t is conditioned (depends on) $y_{t-1}, ..., y_1$ and $\boldsymbol{x}$.

Attention Mechanisms

- Attention is closely related to pooling (used in CNNs and other architectures); it's a form of weighted pooling;

- Attention in MT plays a similar role as alignment, but in a "soft" version, instead of "hard" alignment

- Bahdanau et al. (2015)'s model has no bias in favor of diagonals, short jumps, fertility, etc.

- Some following work adds some "structural" biases (Luong et al., 2015; Cohn et al., 2016)

- Other works constrains the amount of attention each word can receive (based on its fertility): Malaviya et al. (2018).

Attention is Great!

- Attention significantly improves NMT performance!

- It is very useful to allow the decoder to focus on parts of the source;

- Attention solves the bottleneck problem, by allowing the decoder to look directly at the source:

- Attention helps with vanishing gradient problem (provides shortcut to faraway states);

- Attention allows for some interpretability (shows what the the decoder was focusing on when producing each output symbol)

- This is good because we never explicitly trained a word aligner; the network learns it by itself!

Attention Map
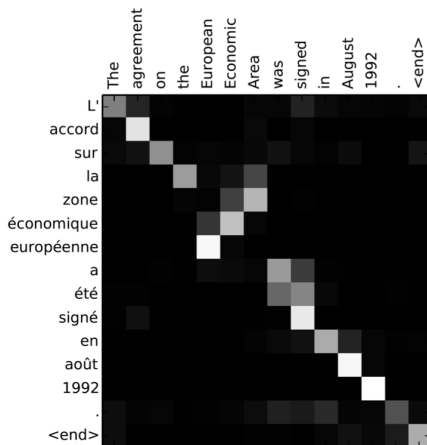
*Dzmitry Bahdanau, KyungHuyn Cho, and Yoshua Bengio. **Neural Machine Translation by Jointly Learning to Translate and Align**. ICLR'15.*

Example: Machine Translation

Some positive examples where NMT has impressive performance:

Source	When asked about this, an official of the American administration replied: "The United States is not conducting electronic surveillance aimed at offices of the World Bank and IMF in Washington."	
PBMT	Interrogé à ce sujet, un responsable de l'administration américaine a répondu : "Les Etats-Unis n'est pas effectuer une surveillance électronique destiné aux bureaux de la Banque mondiale et du FMI à Washington".	3.0
GNMT	Interrogé à ce sujet, un fonctionnaire de l'administration américaine a répondu: "Les États-Unis n'effectuent pas de surveillance électronique à l'intention des bureaux de la Banque mondiale et du FMI à Washington".	6.0
Human	Interrogé sur le sujet, un responsable de l'administration américaine a répondu: "les Etats-Unis ne mènent pas de surveillance électronique visant les sièges de la Banque mondiale et du FMI à Washington".	6.0
Source	Martin told CNN that he asked Daley whether his then-boss knew about the potential shuffle.	
PBMT	Martin a déclaré à CNN qu'il a demandé Daley si son patron de l'époque connaissaient le potentiel remaniement ministériel.	2.0
GNMT	Martin a dit à CNN qu'il avait demandé à Daley si son patron d'alors était au courant du remaniement potentiel.	6.0
Human	Martin a dit sur CNN qu'il avait demandé à Daley si son patron d'alors était au courant du remaniement éventuel.	5.0

(From Wu et al. (2016))

Example: Machine Translation

... But also some negative examples:

- Dropping source words (lack of attention)
- Repeated source words (too much attention)

|	
Source:	1922 in Wien geboren, studierte Mang während und nach dem Zweiten Weltkrieg Architektur an der Technischen Hochschule in Wien bei Friedrich Lehmann.
Human:	Born in Vienna in 1922, Meng studied architecture at the Technical University in Vienna under Friedrich Lehmann *during and after the second World War.*
NMT:	*Born in Vienna in 1922, Mang studied architecture at the Technical College in Vienna with Friedrich Lehmann.
Source:	Es ist schon komisch, wie dies immer wieder zu dieser Jahreszeit auftaucht.
Human:	It's funny how this always comes up at *this time* of year.
NMT:	*It's funny how **this time** to come back to **this time** of year.

Example: Caption Generation
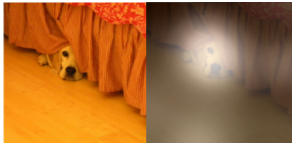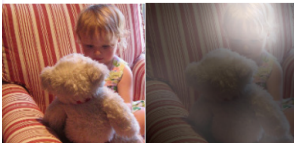
Attention over images:
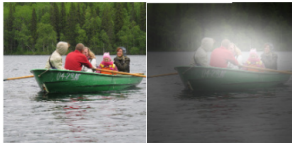

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

(Slide credit to Yoshua Bengio)

A More Extreme Example...

(Slide credit to Dhruv Batra)

Attention and Memories

Attention is used in several problems, sometimes under different names:

- image caption generation (Xu et al., 2015)

- speech recognition (Chorowski et al., 2015)

- memory networks for reading comprehension (Sukhbaatar et al., 2015; Hermann et al., 2015)

- neural Turing machines and other "differentiable computers" (Graves et al., 2014; Grefenstette et al., 2015)

Other Attentions

- Can we have more interpretable attention? Closer to hard alignments?

- Can we upper bound how much attention a word receives? This may prevent a common problem in neural MT, repetitions

- Sparse attention via **sparsemax** (Martins and Astudillo, 2016)

- Constrained attention with constrained softmax/sparsemax (Malaviya et al., 2018)

Conclusions

- Machine translation is a key problem in AI since the 1950s
- Neural machine translation with sequence-to-sequence models was a breakthrough
- Representing a full sentence with a single vector is a bottleneck
- Attention mechanisms allow focusing on different parts of the input and solve the bottleneck problem
- Other applications beyond MT: speech recognition, image captioning, etc.

Thank you!

Questions?
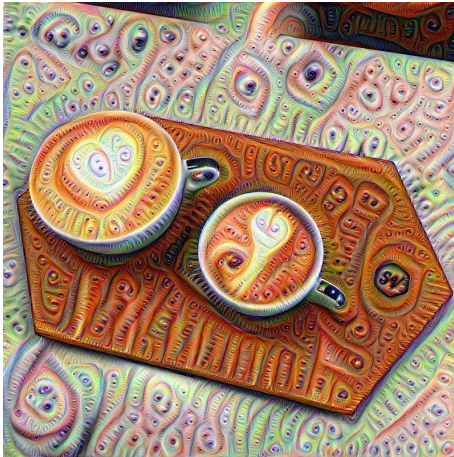
References I

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations*.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. In *Proc. of Empirical Methods in Natural Language Processing*.

Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based Models for Speech Recognition. In *Advances in Neural Information Processing Systems*, pages 577–585.

Cohn, T., Hoang, C. D. V., Vymolova, E., Yao, K., Dyer, C., and Haffari, G. (2016). Incorporating structural alignment biases into an attentional neural translation model. *arXiv preprint arXiv:1601.01085*.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural Turing Machines. *arXiv preprint arXiv:1410.5401*.

Grefenstette, E., Hermann, K. M., Suleyman, M., and Blunsom, P. (2015). Learning to Transduce with Unbounded Memory. In *Advances in Neural Information Processing Systems*, pages 1819–1827.

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching Machines to Read and Comprehend. In *Advances in Neural Information Processing Systems*, pages 1684–1692.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Malaviya, C., Ferreira, P., and Martins, A. F. T. (2018). Sparse and constrained attention for neural machine translation. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*.

Martins, A. F. T. and Astudillo, R. (2016). From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. In *Proc. of the International Conference on Machine Learning*.

Shannon, C. E. and Weaver, W. (1949). The mathematical theory of communication. *Urbana: University of Illinois Press*, 29.

Sukhbaatar, S., Szlam, A., Weston, J., and Fergus, R. (2015). End-to-End Memory Networks. In *Advances in Neural Information Processing Systems*, pages 2431–2439.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112.

References II

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Xu, K., Ba, J., Kiros, R., Courville, A., Salakhutdinov, R., Zemel, R., and Bengio, Y. (2015). Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *International Conference on Machine Learning*.

Lecture 10:
Attention Mechanisms and Transformers

André Martins, Francisco Melo, Mário Figueiredo

Deep Learning Course, Winter 2022-2023

Today's Roadmap

Previous lecture: sequence-to-sequence models using RNNs and attention.

Today we look at self-attention and transformers:

- Convolutional sequence-to-sequence models

- Self-attention

- Transformer networks

- Pre-trained models and transfer learning (next class)

Pointers for Today's Class

- Lena Voita's seq2seq with attention: `https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html`

- Marcos Treviso lecture on attention mechanisms: `https://andre-martins.github.io/docs/dsl2020/attention-mechanisms.pdf`

- John Hewitt's lecture on self-attention and transformers: `http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture09-transformers.pdf`
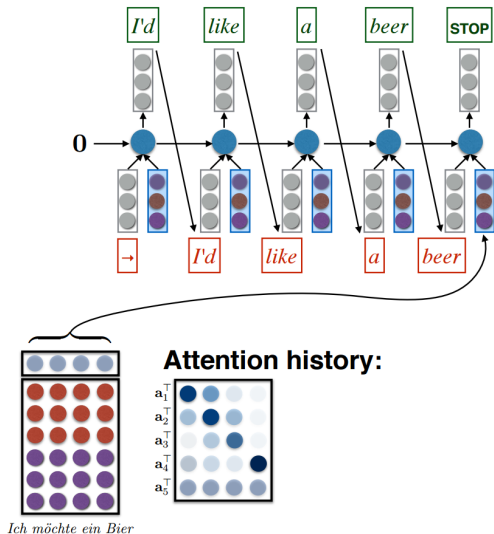
- Illustrated transformer: `http://jalammar.github.io/illustrated-transformer/`

- Annotated transformer: `https://nlp.seas.harvard.edu/2018/04/03/attention.html`

Recap: RNN with Attention (Encoder)
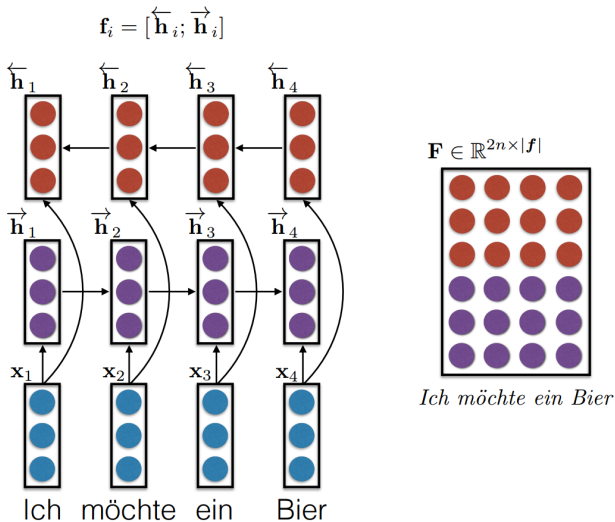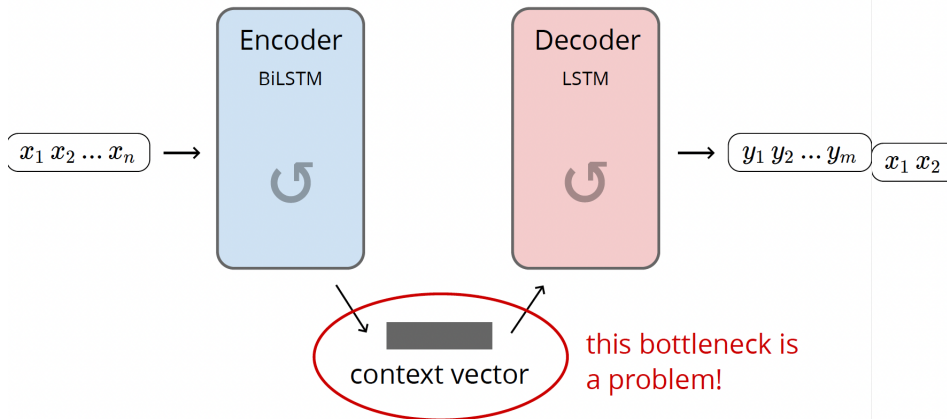

(Slide credit: Chris Dyer)

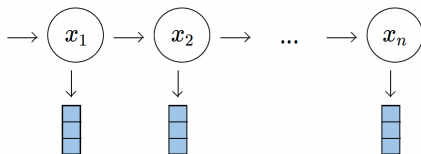# Recap: RNN with Attention (Decoder)

(Slide credit: Chris Dyer)

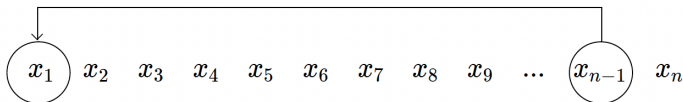# RNN-Based Encoder-Decoder

Drawbacks of RNNs

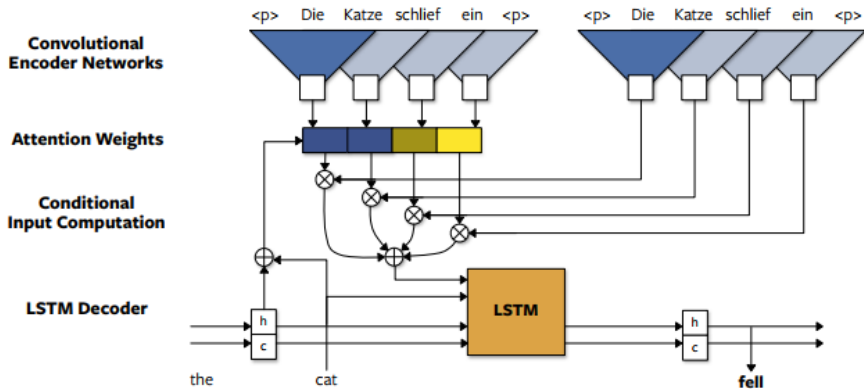- Sequential mechanism prohibits parallelization

- Long-range dependencies are tricky, despite gating

- Possible solution: replace RNN encoder by hierarchical 1-D CNN
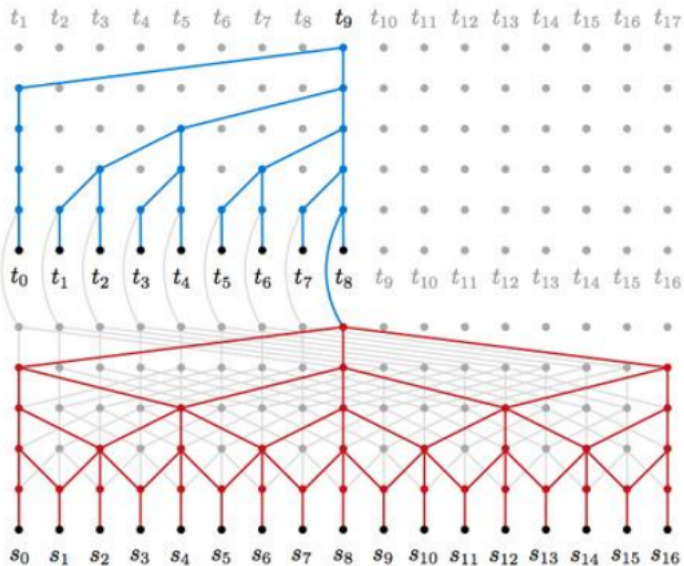
Convolutional Encoder
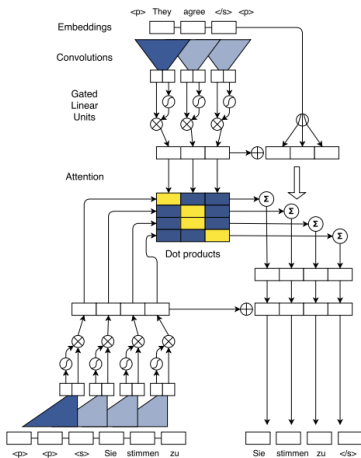

(Gehring et al., 2017)

Fully Convolutional

- Can use CNN decoder too!

- Convolutions will be over output prefixes only

- Encoder is parallelizable, but decoder still requires sequential computation (the model is still auto-regressive)

Convolutional Sequence-to-Sequence

Convolutional Sequence-to-Sequence

(Gehring et al., 2017)

Next: Self-Attention

- Both RNN and CNN decoders require an attention mechanism

- Attention allows focusing on an arbitrary position in the source sentence, shortcutting the computation graph

- But if attention gives us access to any state...
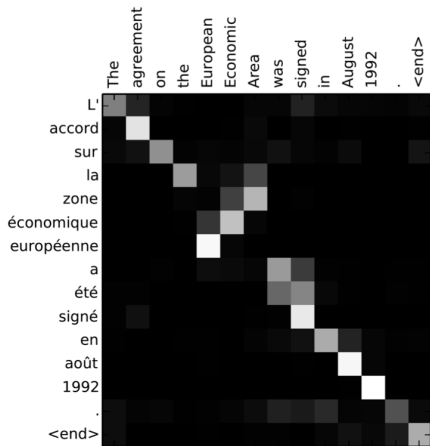 ...maybe we don't need the RNN?

Why Attention?

We want NNs that automatically weigh input relevance

Main advantages:

- performance gain

- none or few parameters

- fast (easy to parallelize)

- tool for "interpreting" predictions

Example: Machine Translation

*Dzmitry Bahdanau, KyungHuyn Cho, and Yoshua Bengio. **Neural Machine Translation by Jointly Learning to Translate and Align**. ICLR'15.*

Example: Caption Generation
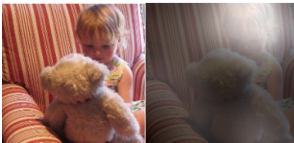
Attention over images:

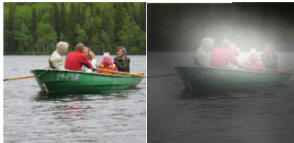A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

(Slide credit to Yoshua Bengio)

Example: Document Classification

Task: Hotel location

you get what you pay for . not the cleanest rooms but bed was clean and so was bathroom . bring your own towels though as very thin . service was excellent , let us book in at 8:30am ! for location and price , this ca n't be beaten , but it is cheap for a reason . if you come expecting the hilton , then book the hilton ! for uk travellers , think of a blackpool b&b.

Task: Hotel cleanliness

you get what you pay for . **not the cleanest rooms but bed was clean and so was bathroom** . bring your own towels though as very thin . service was excellent , let us book in at 8:30am ! for location and price , this ca n't be beaten , but it is cheap for a reason . if you come expecting the hilton , then book the hilton ! for uk travellers , think of a blackpool b&b.

Task: Hotel service

you get what you pay for . not the cleanest rooms but bed was clean and so was bathroom . bring your own towels though as very thin . **service was excellent** , let us book in at 8:30am ! for location and price , this ca n't be beaten , but it is cheap for a reason . if you come expecting the hilton , then book the hilton ! for uk travellers , think of a blackpool b&b.
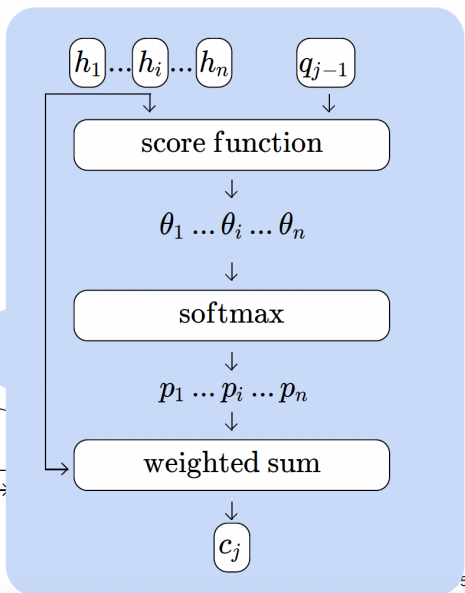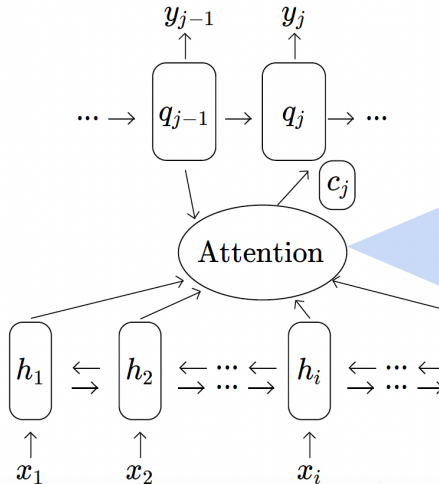
(Bao et al., 2018)

Attention Mechanism

- Bahdanau et al. (2015)

Attention Mechanism: Recap

Recall how attention works:

1. We have a query vector $\boldsymbol{q}$ (e.g. the decoder state)
2. We have input vectors $\boldsymbol{H} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_L]^\top$ (e.g. one per source word)
3. We compute affinity scores $s_1, \ldots, s_L$ by "comparing" $\boldsymbol{q}$ and $\boldsymbol{H}$
4. We convert these scores to probabilities:

$$\boldsymbol{p} = \mathbf{softmax}(\boldsymbol{s})$$

5. We use this to output a representation as a weighted average:

$$\boldsymbol{c} = \boldsymbol{H}^\top \boldsymbol{p} = \sum_{i=1}^{L} p_i \boldsymbol{h}_i$$

Let's see these steps in detail!

Affinity Scores

Several ways of "comparing" a query $\boldsymbol{q}$ and an input ("key") vector $\boldsymbol{h}_i$:

- Additive attention (Bahdanau et al., 2015), what we covered in previous class:
$$s_i = \boldsymbol{u}^\top \tanh(\boldsymbol{A}\boldsymbol{h}_i + \boldsymbol{B}\boldsymbol{q})$$

- Bilinear attention (Luong et al., 2015):

$$s_i = \boldsymbol{q}^\top \boldsymbol{U}\boldsymbol{h}_i$$

- Dot product attention (Luong et al., 2015) (particular case; queries and keys must have the same size):

$$s_i = \boldsymbol{q}^\top \boldsymbol{h}_i$$

The last two are easier to batch when we have multiple queries and multiple keys.

Keys and Values

The input vectors $\boldsymbol{H} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_L]^\top$ appear in two places:

- They are used as keys to "compare" them with the query vector $\boldsymbol{q}$ to obtain the affinity scores

- They are used as values to form the weighted average $\boldsymbol{c} = \boldsymbol{H}^\top \boldsymbol{p}$

To be fully general, they don't need to be the same – we can have:

- A key matrix $\boldsymbol{K} = [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_L]^\top \in \mathbb{R}^{L \times d_K}$

- A value matrix $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_L]^\top \in \mathbb{R}^{L \times d_V}$

Attention Mechanism: More General Version

1 We have a query vector $\boldsymbol{q}$ (e.g. the decoder state)

2 We have key vectors $\boldsymbol{K} = [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_L]^\top \in \mathbb{R}^{L \times d_K}$
and value vectors $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_L]^\top \in \mathbb{R}^{L \times d_V}$
(e.g. one of each per source word)

3 We compute query-key affinity scores $s_1, \ldots, s_L$ "comparing" $\boldsymbol{q}$ and $\boldsymbol{K}$

4 We convert these scores to probabilities:

$$\boldsymbol{p} = \textbf{softmax}(\boldsymbol{s})$$

5 We output a weighted average of the values:

$$\boldsymbol{c} = \boldsymbol{V}^\top \boldsymbol{p} = \sum_{i=1}^{L} p_i \boldsymbol{v}_i \in \mathbb{R}^{d_V}$$

Self-Attention

- So far we talked about contextual attention – the decoder attends to encoder states (this is called "input context")

- The encoder and the decoder states were propagated sequentially with a RNN, or hierarchically with a CNN

- Alternative: self-attention – at each position, the encoder attends to the other positions in the encoder itself

- Same for the decoder.

Self-Attention Layer

Self-attention for a sequence of length L:

1. Query vectors $\boldsymbol{Q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_L]^\top \in \mathbb{R}^{L \times d_Q}$
2. Key vectors $\boldsymbol{K} = [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_L]^\top \in \mathbb{R}^{L \times d_K}$
3. value vectors $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_L]^\top \in \mathbb{R}^{L \times d_V}$
4. Compute query-key affinity scores "comparing" $\boldsymbol{Q}$ and $\boldsymbol{K}$, e.g.,

$$\boldsymbol{S} = \boldsymbol{Q}\boldsymbol{K}^\top \in \mathbb{R}^{L \times L} \qquad \text{(dot-product affinity)}$$
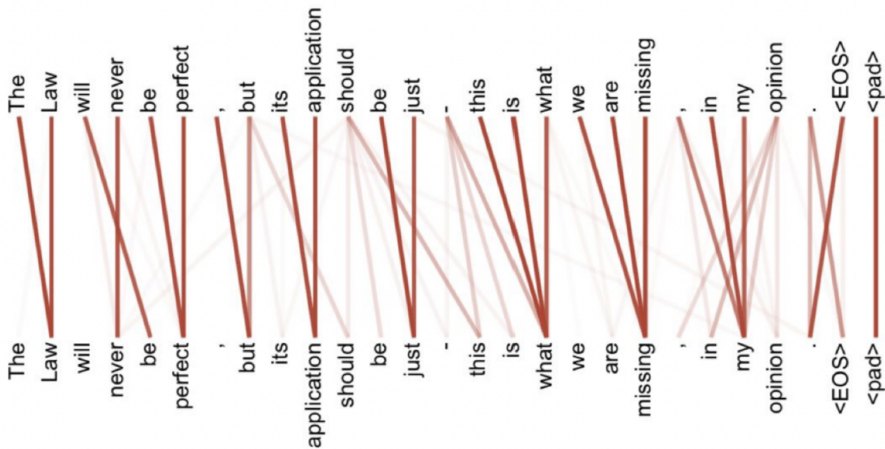
5. Convert these scores to probabilities (row-wise):

$$\boldsymbol{P} = \mathbf{softmax}(\boldsymbol{S}) \in \mathbb{R}^{L \times L}$$

6. Output the weighted average of the values:

$$\boldsymbol{Z} = \boldsymbol{P}\boldsymbol{V} = \underbrace{\mathbf{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top)}_{P} \boldsymbol{V} \in \mathbb{R}^{L \times d_V}.$$

Self-Attention

(Vaswani et al., 2017)

Transformer (Vaswani et al., 2017)

- **Key idea:** instead of RNN/CNNs, use self-attention in the encoder

- Each word state attends to all the other words

- Each self-attention is followed by a feed-forward transformation

- Do several layers of this

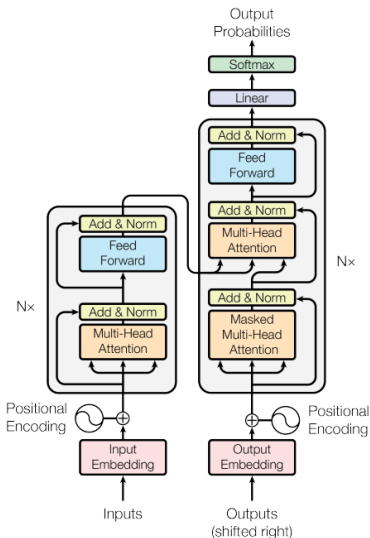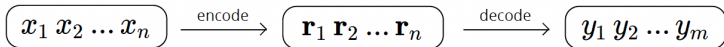- Do the same for the decoder, attending only to already generated words.
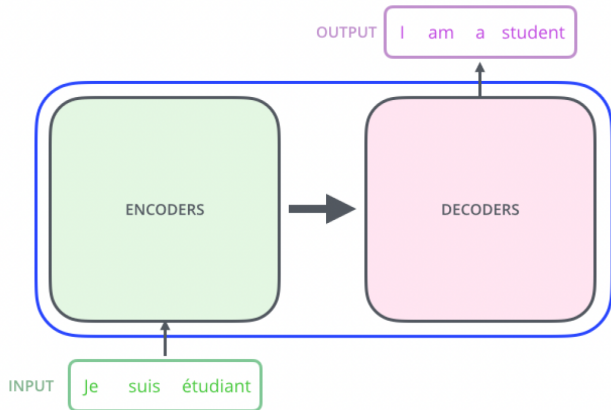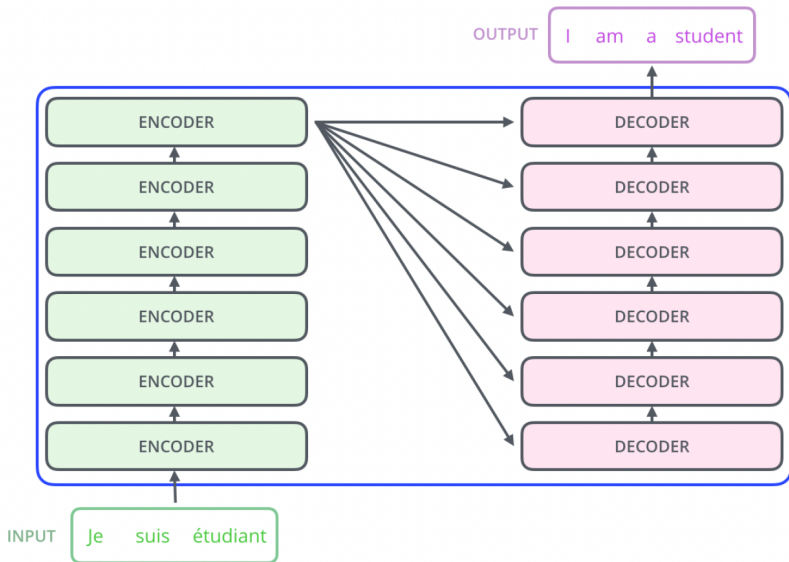

Figure 1: The Transformer - model architecture.

Transformer

Transformer

Transformer Blocks
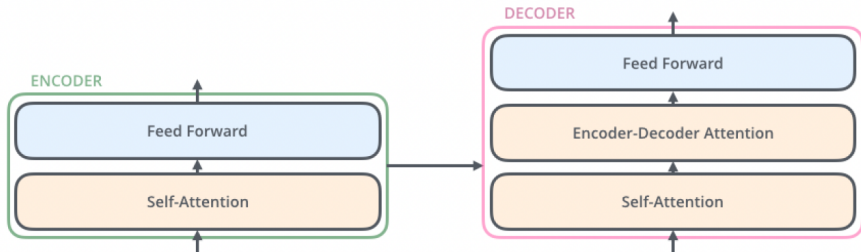
(Illustrated transformer: `http://jalammar.github.io/illustrated-transformer/`)
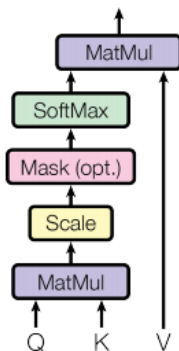
Transformer Basics

Let's define the basic building blocks of transformer networks first: new attention layers!

Two innovations:

- scaled dot-product attention
- multi-head attention

Scaled Dot-Product and Multi-Head Attention

Scaled Dot-Product Attention

Multi-Head Attention
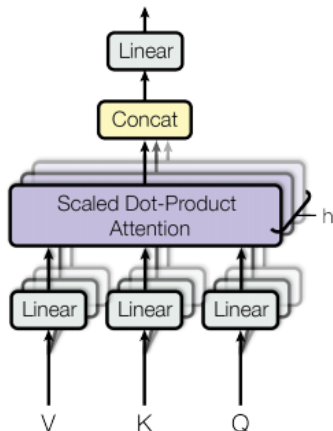
(Vaswani et al., 2017)

The Encoder

Example for a sentence with 2 words:
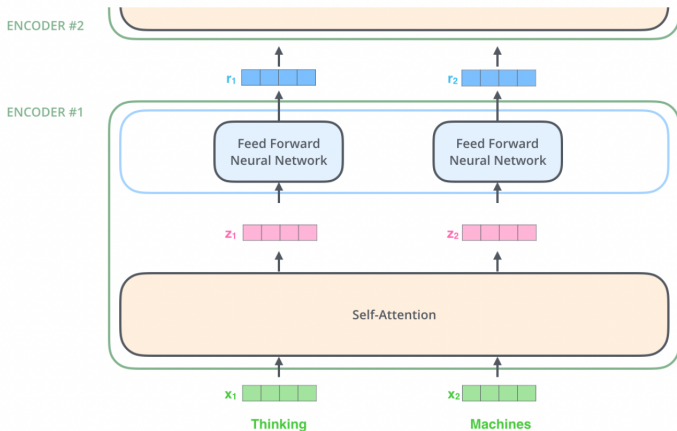
Transformer Self-Attention: Queries, Keys, Vectors

- Obtained by projecting the embedding matrix $\boldsymbol{X} \in \mathbb{R}^{L \times e}$ to a lower dimension:

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q$$
$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K$$
$$\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V.$$

- The projection matrices $\boldsymbol{W}^Q$, $\boldsymbol{W}^K$, $\boldsymbol{W}^V$ are model parameters.

Transformer Self-Attention: Queries, Keys, Vectors
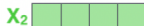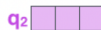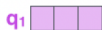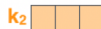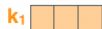
Scaled Dot-Product Attention

Problem: As d_K gets large, the variance of $\boldsymbol{q}^\top \boldsymbol{k}$ increases, the softmax gets very peaked, hence its gradient gets smaller.

Solution: scale by length of query/key vectors:

$$\boldsymbol{Z} = \mathsf{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_K}}\right)\boldsymbol{V}.$$

Scaled Dot-Product Attention

Scaled Dot-Product and Multi-Head Attention

(Vaswani et al., 2017)

Multi-Head Attention

Self-attention: each word forms a query vector and attends to the other words' key vectors

This is vaguely similar to a 1D convolution, but where the filter weights are "dynamic" is the window size spans the entire sentence!

Problem: only one channel for words to interact with one-another

Solution: multi-head attention!

- define h attention heads, each with their own projection matrices (e.g. $h = 8$)
- apply attention in multiple channels, concatenate the outputs and pipe through linear layer:

$$\mathrm{MultiHead}(\boldsymbol{X}) = \mathrm{Concat}(\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_h)\boldsymbol{W}^O,$$

where $\boldsymbol{Z}_i = \mathrm{Attention}(\underbrace{\boldsymbol{X}\boldsymbol{W}_i^Q}_{\boldsymbol{Q}_i}, \underbrace{\boldsymbol{X}\boldsymbol{W}_i^K}_{\boldsymbol{K}_i}, \underbrace{\boldsymbol{X}\boldsymbol{W}_i^V}_{\boldsymbol{V}_i})$.

Multi-Head Attention

1) This is our input sentence*

2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices
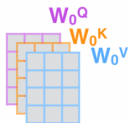
4) Calculate attention using the resulting Q/K/V matrices

5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

Thinking Machines

X

W_0^Q
W_0^K
W_0^V

Q_0
K_0
V_0

Z_0

W^O

Z

* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

W_1^Q
W_1^K
W_1^V

Q_1
K_1
V_1

Z_1

R

...

W_7^Q
W_7^K
W_7^V

...

Q_7
K_7
V_7

...

Z_7
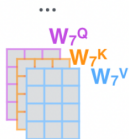
Other Tricks

- Self-attention blocks are repeated several times (e.g. 6 or 12)

- Residual connections on each attention block

- Layer normalization

- Positional encodings (to distinguish word positions)
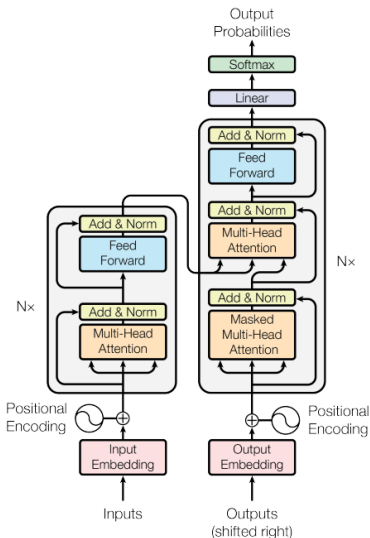
Figure 1: The Transformer - model architecture.

Positional Encodings

- As just described, the transformer is insensitive to word order!
  - queries attend to keys regardless of their position in the sequence
- To make it sensitive to order, we add positional encodings
- Two strategies: learn one embedding for each position (up to a maximum length) or use sinusoidal positional encodings (next)
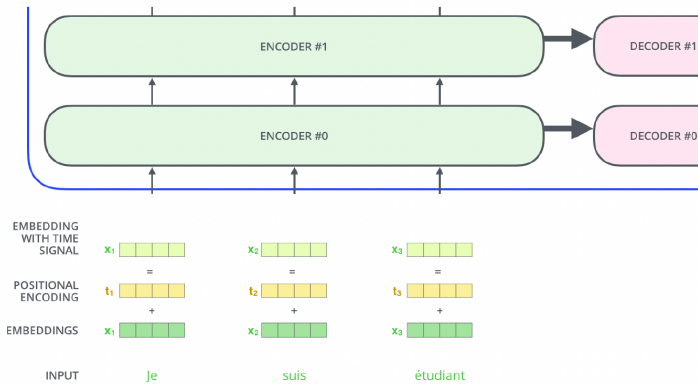
Sinusoidal Positional Encodings

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right) \qquad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$
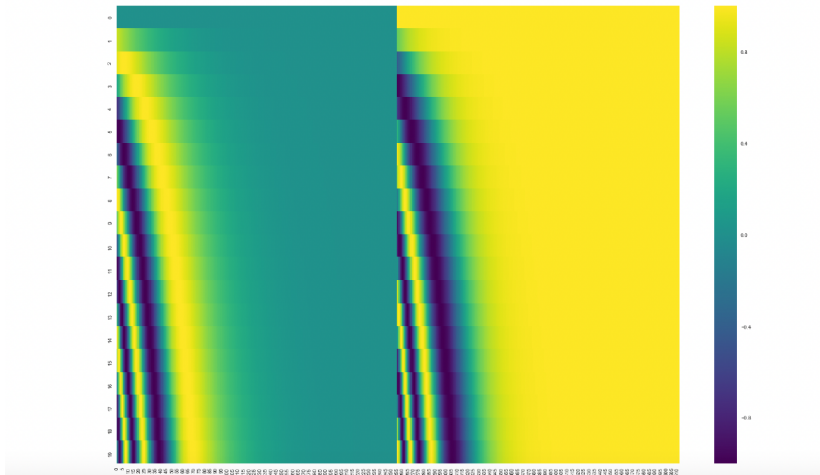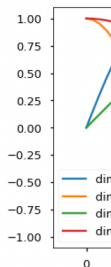
Residuals and Layer Normalization
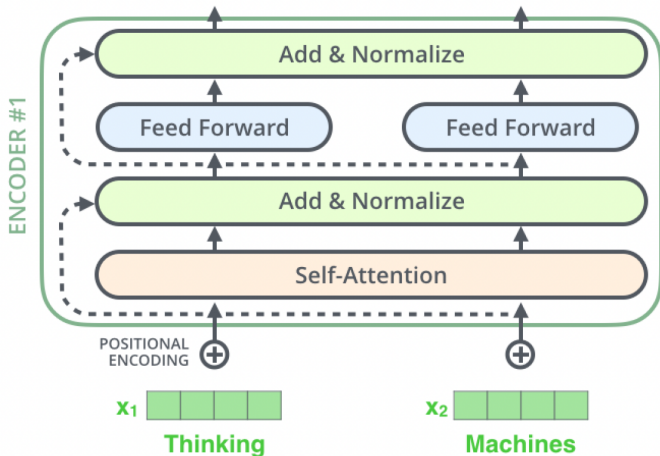
The Decoder

What about the self-attention blocks in the decoder?

Everything is pretty much the same as in the encoder, with two twists:

- The decoder cannot see the future! Use "causal" masking

- The decoder should attend to itself (self-attention), but also to the encoder states (contextual attention).

The Decoder

encoder self-attn

decoder self-attn (masked)
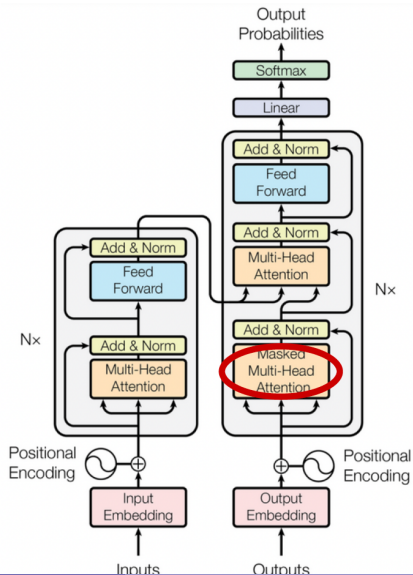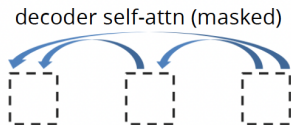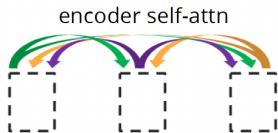
Attention Visualization Layer 5
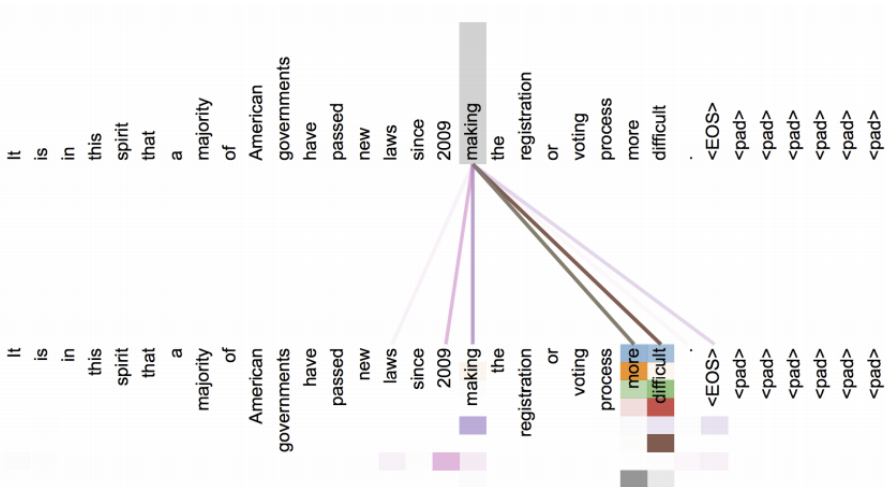
Implicit Anaphora Resolution
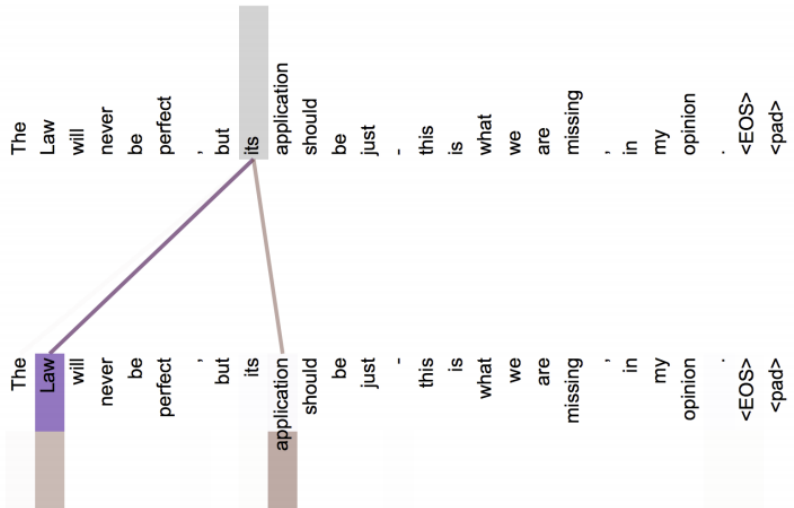
Computational Cost

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

n = seq. length d = hidden dim k = kernel size

- Faster to train (due to self-attention parallelization)

- More expensive to decode

- Scale quadratically with respect to sequence length (problematic for long sequences).

Other Tricks

- Label smoothing
- Dropout at every layer before residuals
- Beam search with length penalty
- Adam optimizer with learning-rate decay
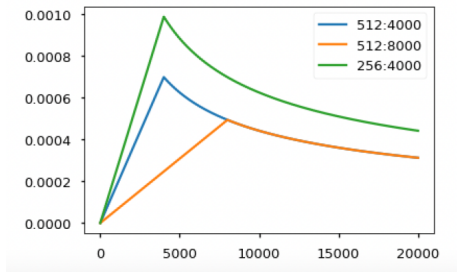
Overall, transformers are harder to optimize than RNN seq2seq models

They don't work out of the box: hyperparameter tuning is very important.

Transformer Results

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

(Vaswani et al., 2017)'s "Attention Is All You Need"

TransformerXL

Big transformers can look at larger contexts.

TransformerXL: enables going beyond a fixed length without disrupting temporal coherence:

(a) Training phase.

(b) Evaluation phase.
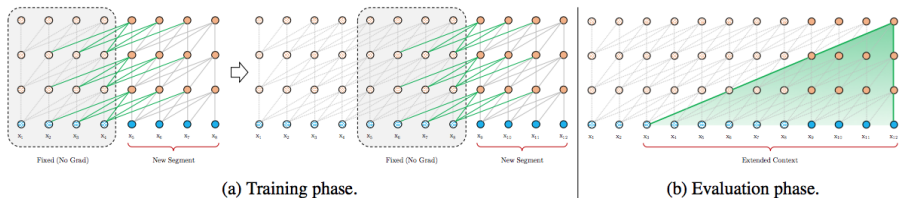
(Dai et al., 2019)

Conclusions

- RNN-based seq2seq models require sequential computation and have difficulties with long range dependencies

- Attention mechanisms allow focusing on different parts of the input

- Encoders/decoders can be RNNs, CNNs, or self-attention layers

- Transformers are the current state of the art for many tasks in NLP and vision

- Other applications: speech recognition, image captioning, etc.

- Next lecture: pretrained models and transfer learning (BERT, GPT-2, GPT-3, etc.)

Thank you!

Questions?
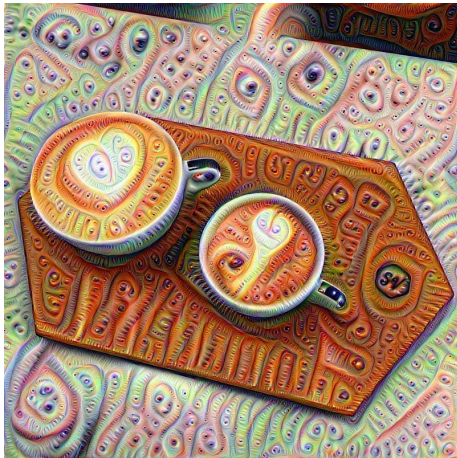
References I

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations*.

Bao, Y., Chang, S., Yu, M., and Barzilay, R. (2018). Deriving machine attention from human rationales. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1903–1913.

Dai, Z., Yang, Z., Yang, Y., Cohen, W. W., Carbonell, J., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Lecture 11: Self-Supervised Learning and Large Pretrained Models

André Martins, Francisco Melo, Mário Figueiredo

TÉCNICO LISBOA

Deep Learning Course, Winter 2022-2023

Announcements

- Next Thursday (January 12): guest lecture by Chrysoula Zerva.

  Grande Auditório, Centro de Congressos, 13:30

Today's Roadmap

Previous lecture: sequence-to-sequence models and transformers.

Today: large pretrained models (BERT, GPT3, etc.) and how to use them for downstream tasks.

- Contextualized representations

- Self-supervised learning

- Pretraining and finetuning

- Adaptors and prompting.

Pointers for Today's Class

- John Hewitt's lecture on pretrained transformer models:
  `http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture10-pretraining.pdf`

From Static to Contextualized Word Embeddings

- In the representation learning lecture, we saw how to obtain word representations (embeddings) (e.g. word2vec, GloVe)

- Each word in the vocabulary is represented by a vector, regardless of its context (a "static" vector)

- Today: how to obtain contextualized embeddings.

GloVe Visualizations: Company → CEO
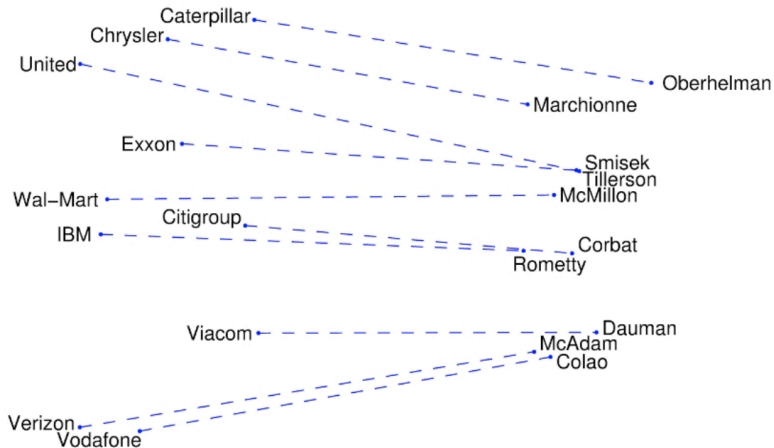
(Slide credit to Richard Socher)

GloVe Visualizations: Superlatives
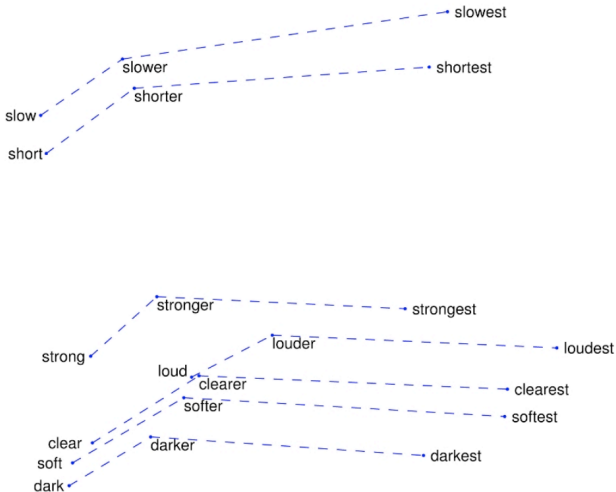
(Slide credit to Richard Socher)

Word Embeddings: Some Open Problems

- Can we do word embeddings for multiple languages in the same space?

- How to capture polysemy (e.g. "bear" vs "bear"; "flies" vs "flies")?

- Can we compute embeddings on-the-fly, depending on the context?

Contextualized Embeddings

- Words can have different meanings, depending on which context they appear in.

- In 2018, a model called ELMo learned context-dependent embeddings and achieved impressive results on 6 NLP downstream tasks (Peters et al., 2018).

- This was the first of a series of models named after Sesame Street characters (more to come).

Embeddings from Language Models (ELMo)
(Peters et al., 2018)

Key idea:

- Pre-train a BiLSTM language model on a large dataset

- Save all the parameters at all layers, not only the embeddings

- Then, for your downstream task, tune a scalar parameter for each layer, and pass the entire sentence through this encoder.

Later several models have been proposed (BERT, GPT) with even more impressive performance.

Pretraining through Language Modeling

Recall the language modeling task:

- Model $p_\theta(y_t \mid y_{1:(t-1)})$, the probability distribution of words given their past contexts.

- There's lots of data for this! No *labels* are necessary, just raw text.

- This is called unsupervised pretraining or self-supervised learning.

Pretraining through language modeling:

- Train a NN to perform language modeling on a large amount of text.

- Save the network parameters.

Pretraining and Fine-tuning

Pretraining can be very effective by serving as parameter initialization.

Step 1: Pretrain (e.g. on LM)

Lots of text; learn general things!
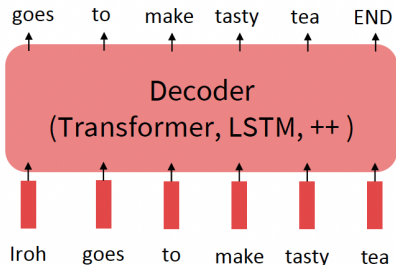
Step 2: Finetune (on your task)

Not many labels; adapt to the task!
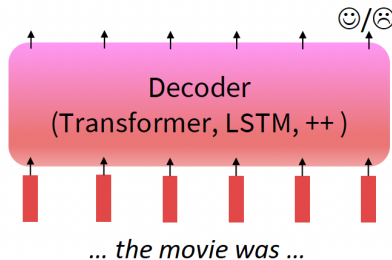
Self-Supervised Learning

Pretraining on language model task is a form of self-supervised learning

- Take raw (unlabeled) data, remove information and train a model to recover that information

- In the case of language modeling, the information removed is the next word; the model is trained to predict future words given the context

- Other strategies: *mask* words (later)

- This can be done with signals, images too, not just NLP

- For example, take images, obfuscate a region, and train a model to predict the missing region (image completion)

Constrastive Predictive Coding (Speech)
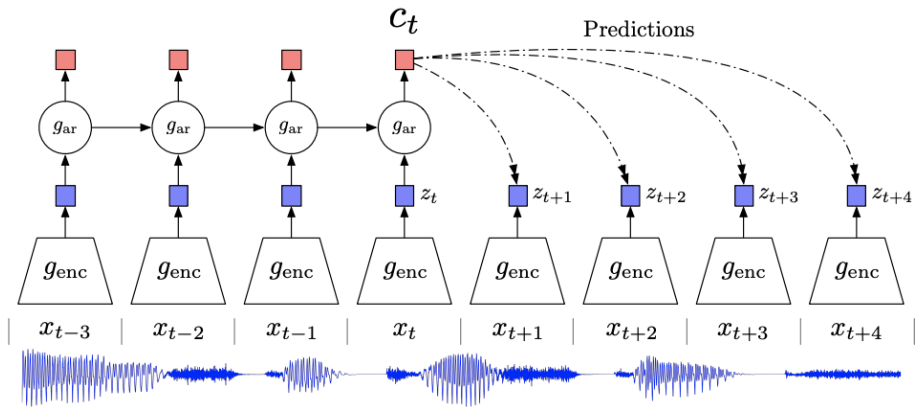
(From Oord et al. (2018))

Constrastive Predicting Coding (Images)
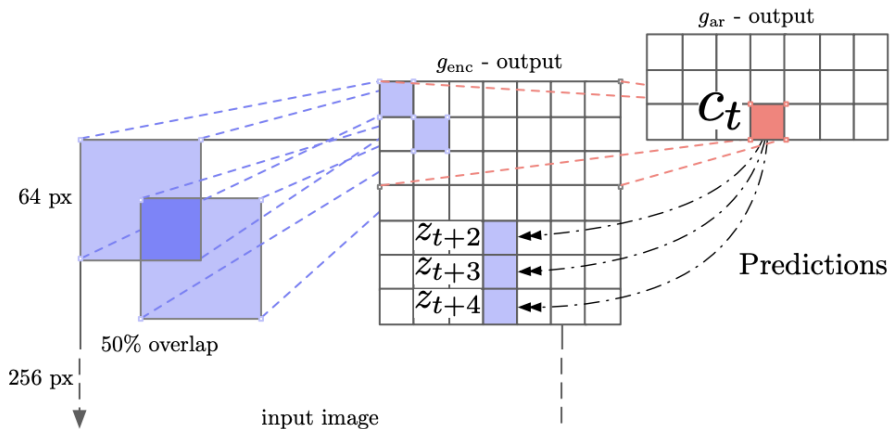
(From Oord et al. (2018))

Siamese Networks

- Rotate, translate, or scale existing image.

- Minimize the distance between the two representations.
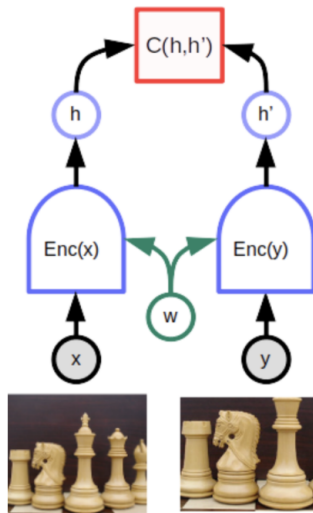
(From https://ai.facebook.com/blog/
self-supervised-learning-the-dark-matter-of-intelligence/)

Why Does This Work?

Let's look at pretraining and fine-tuning from a "training neural nets" perspective.

- Pretraining leads to parameters $\hat{\theta} \approx \arg\min_{\theta} L_{\text{pretrain}}(\theta)$

- Fine-tuning approximates $\arg\min_{\theta} L_{\text{finetune}}(\theta)$, starting at $\hat{\theta}$

- Pretraining helps because SGD stays (relatively) close to $\hat{\theta}$ during fine-tuning.

- Pretraining on large datasets exposes the model to many words and contexts not seen in the fine-tuning data (a form of weak supervision).

- Hopefully, the fine-tuning local minima near $\hat{\theta}$ tend to generalize well!

Three Architectures for Pretraining
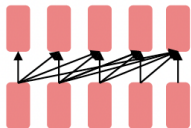
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
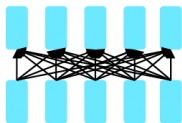
Encoders

- Bidirectional context $\Rightarrow$ can condition on future!
- Wait, how do we pretrain them?
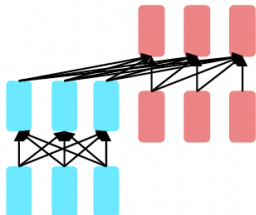
Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

Pretrained Decoders

- When using language model pretrained decoders, we can ignore that they were trained to model $p_\theta(x_t \mid x_{1:(t-1)})$

- Fine-tuning by training a classifier on the last hidden state.

$$\boldsymbol{h}_1, \ldots, \boldsymbol{h}_L = \mathrm{Decoder}(x_1, \ldots, x_L)$$
$$\boldsymbol{y} = \mathsf{softmax}(\boldsymbol{A}\boldsymbol{h}_L + \boldsymbol{b})$$

where $\boldsymbol{A}$ and $\boldsymbol{b}$ are randomly initialized and learned by the downstream task.

$\odot/\odot$

Linear $\quad A, b$

$h_1, \ldots, h_T$

$w_1, \ldots, w_T$
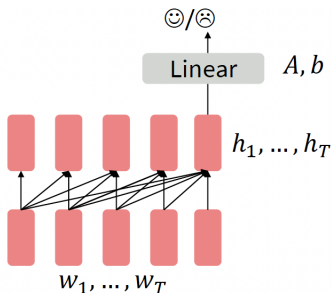
Pretrained Decoders

Two common choices for fine-tuning:

- *Freeze* the pretrained model and train only **A** and **b**

- Or fine-tune everything, letting gradients backpropagate through the whole network.

Pretrained decoders are particularly useful for generation tasks:

- Summarization

- Machine Translation

- Dialogue

- etc.

The family of GPT models developed by OpenAI is an example.

Generative Pretrained Transformer (GPT)
(Radford et al., 2018)

- 2018's GPT was a big success in pretraining a decoder!

- Transformer decoder with 12 layers.

- 768-D hidden states, 3072-D feed forward hidden layers.

- Byte pair encoding with 40,000 merges (vocabulary size)

- Trained on BooksCorpus over 7000 books.

- Contains long spans of contiguous text, for learning long distance dependencies.

Generative Pretrained Transformer (GPT)
(Radford et al., 2018)

How do we format inputs to our decoder for finetuning tasks?

Radford et al. (2018) evaluates on **Natural Language Inference** (NLI):

- Label pairs of sentences as entailment/contradiction/neutral

  Premise: "The man is in the doorway"

  Hypothesis: "The person is near the door" ⇒ *entailment*

The input is formatted to become a sequence of tokens for the decoder:

  [START] The man is in the doorway [DELIM] The person is near the door [EXTRACT]

A linear classifier is applied to the representation of [EXTRACT]

Generative Pretrained Transformer (GPT)
(Radford et al., 2018)

Results on various natural language inference (NLI) tasks:

Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned Transformer LM (ours)	**82.1**	**81.4**	**89.9**	88.3	**88.1**	56.0

GPT-2 (Radford et al., 2019)

Another iteration of GPT, with larger models and trained on much more data (40GB of Internet text)

Particularly strong in language generation tasks.

- A transformer-based language model with 1.5 billion parameters, trained on 8 million web pages

- Training task: predict the next word, given all of the previous words

- Test time: top-k truncated random sampling

- Lots of hype!

GPT-2's Famous Example (Radford et al., 2019)

Given a prompt, perform story completion:

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

GPT-2 Generated Text

A machine learning school was held in Bangkok early August. I had originally expected a much more "procedural and algorithmic" approach as well, but was ple asantly surprised by the level of expertise of the students. These were both very experienced mathematicians, but also had an appreciation for machine learning techniques. These are the same school that created and operates the very popular Algorithmia blog , so they aren't exactly new to me, and are well known to other Algorithmia grads at the University of Central Europe.

`https://talktotransformer.com/`

GPT-2 Generated Text

I am teaching a deep learning class. *I have developed a good curriculum, and most of my students are writing tutorial code that I'll review during class. At the same time, I'm doing training with several of my students. I am giving them a lot of freedom in their code. But I am giving them rigorous, artificial data to train on. I will judge their code based on this data.*

It doesn't make much sense to me to write a 5 minute tutorial that I don't use for validation. I don't want to validate these tutorials in an inconsistent way.

`https://talktotransformer.com/`

GPT-3 (Brown et al., 2020)

Keep scaling up... Even larger models pretrained on more data:

- 175 billion parameters (!!!)

- Trained on 500 billion words

- Took 3.14×10^{23} FLOPS to train (on a standard GPU, it would cost $4.6M and it would require 355 years to train such a model)

- Introduces prompting as an alternative to fine-tuning (later)

- Demonstrates few-shot learning capabilities (learning new tasks on the fly from very few examples)

GPT-3 Generated Text

I am teaching a deep learning course in Lisbon. Do you think this is a good idea?

It is always a good idea to share knowledge and educate others. If you are well-prepared and have a strong understanding of deep learning, then teaching a course on the subject could be a rewarding experience for both you and your students. Lisbon is a vibrant and culturally rich city, so it could be a great place to teach a course on deep learning. It is important to consider the needs and backgrounds of your students and to structure the course in a way that is engaging and accessible to them.

Large Pretrained Models
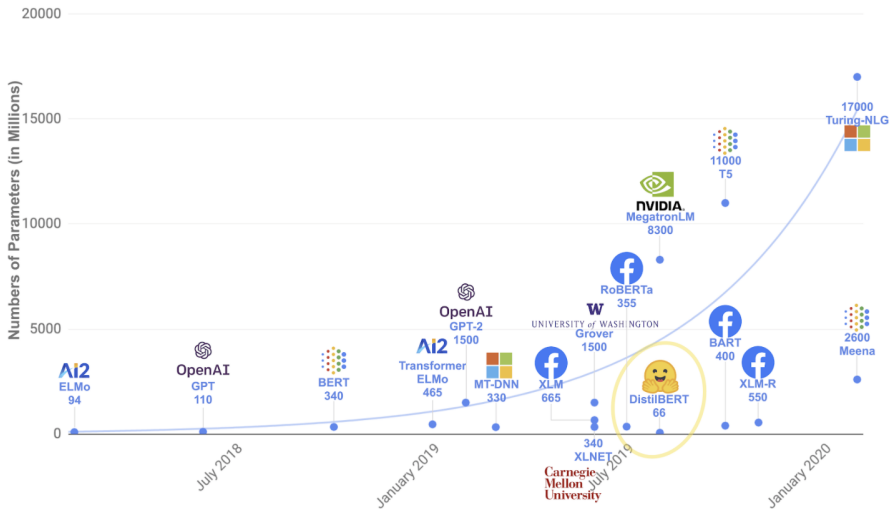
Large Pretrained Models
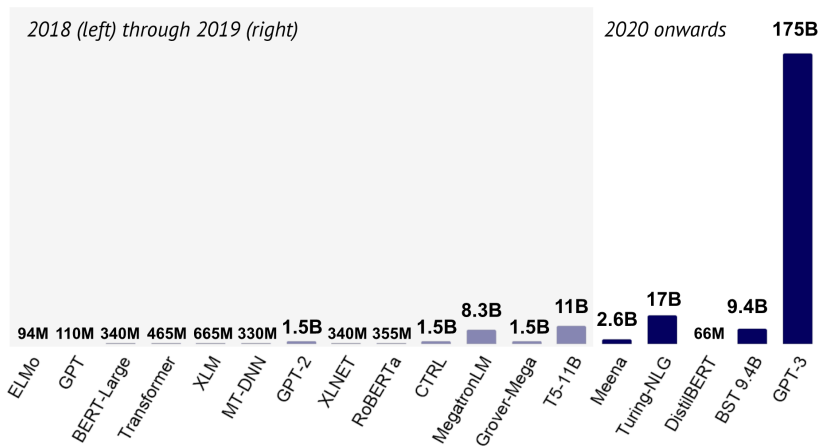
Three Architectures for Pretraining
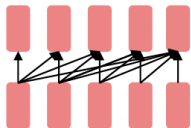
Decoders ✓

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
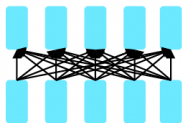
Encoders

- Bidirectional context ⇒ can condition on future!
- Wait, how do we pretrain them?
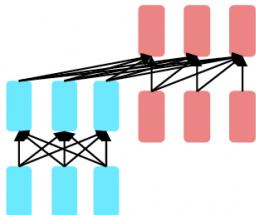
Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

Pretrained Encoders

So far, we've looked at language model pretraining. But encoders get bidirectional context, so we can't do language modeling!

So, what pretraining objective to use? Masked Language Modeling.

Masked Language Modeling

- Idea: replace a fraction of words in the input with a special [MASK] token; predict these words.

$$\boldsymbol{h}_1, \ldots, \boldsymbol{h}_L = \text{Encoder}(x_1, \ldots, x_L)$$
$$\boldsymbol{y}_i = \textbf{softmax}(\boldsymbol{A}\boldsymbol{h}_i + \boldsymbol{b}).$$

- Only add loss terms from words that are "masked out." If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(x|\tilde{x})$

- Similar to a denoising auto-encoder.
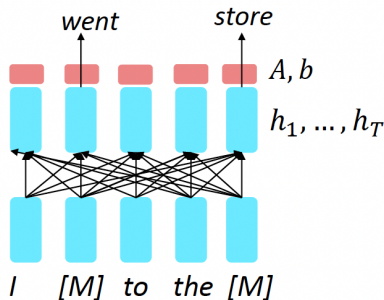
(Devlin et al., 2018)

Example: BERT (Devlin et al., 2018)

"Bidirectional Encoder Representations from Tranformers"

- Randomly mask 15% of the words of the input and train a Transformer to recover those words from the context

- Both left and right context, used simultaneously!

- In doing so, learn contextualized word representations

- Can use this as a pre-trained model and fine-tune it to any downstream task

- Extremely effective! Achieved SOTA on 11 NLP tasks (7.7% absolute point improvement on GLUE score).

Example: BERT (Devlin et al., 2018)
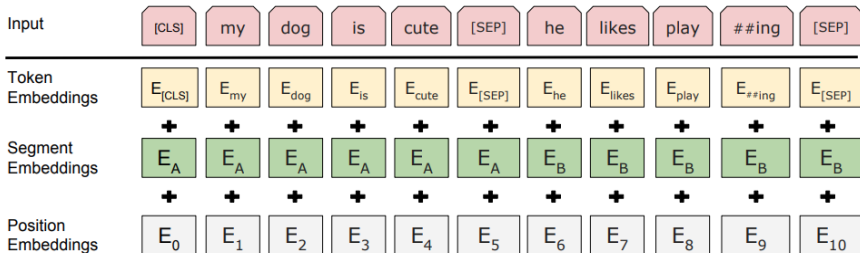
(Devlin et al., 2018)

Aditionally to predicting masked words, BERT is also trained to predict whether one chunk follows the other or is randomly sampled (to obtain sentence-level representations.

Later work has argued this "next sentence prediction" is not necessary.

Details about BERT (Devlin et al., 2018)

Two models were released:

- BERT base: 12 layers, 768 dim hidden states, 12 attention heads, 110 million params.
- BERT large: 24 layers, 1024 dim hidden states, 16 attention heads, 340 million params.

Trained on:

- BooksCorpus (800 million words)
- English Wikipedia (2,500 million words)

Pretraining is expensive and impractical on a single GPU.

- BERT was pretrained with 64 TPU chips for a total of 4 days.

Fine-tuning is practical and common on a single GPU:

- "Pretrain once, finetune many times."

Fine-Tuning BERT (Devlin et al., 2018)

BERT became massively popular and versatile; finetuning BERT led to new state of the art results on a broad range of NLP tasks:

- Paraphrase detection (QQP, MRPC)
- Natural language inference (QNLI, RTE)
- Sentiment analysis (SST-2)
- Grammatical correctness (CoLA)
- Semantic textual similarity (STS-B)

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT$_{BASE}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT$_{LARGE}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Other variants of BERT

- M-BERT (same as BERT but multilingual, not English-specific)
  - Effective in many cross-lingual tasks

- RoBERTA (similar to BERT, but trained on more data and removing next-sentence prediction)

- XLM-RoBERTA (multilingual version)

- SpanBERT

- ...

Limitations of pretrained encoders

Why not use pretrained encoders for everything?

Pretrained decoders (causal LM) vs pretrained encoders (masked LM):

- If your task involves generating sequences, use a pretrained decoder (BERT and other pretrained encoders don't naturally lead to nice autoregressive generation methods.)

- If your task involves classification or sequence tagging, use a pretrained encoder; you can usually benefit from bidirectionality.
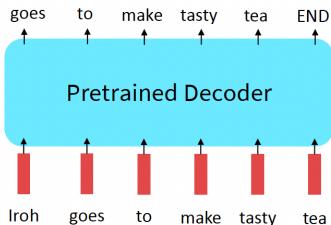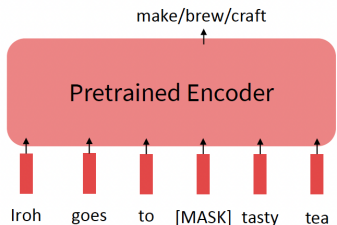
Three Architectures for Pretraining
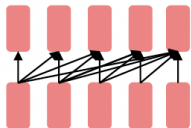
Decoders ✓

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
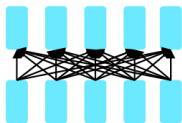
Encoders ✓

- Bidirectional context ⇒ can condition on future!
- Wait, how do we pretrain them?
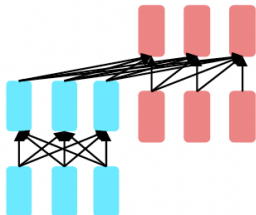
Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

Pretrained Encoder-Decoder

- For encoder-decoders , we can do something like language modeling, but where a prefix of every input is provided to the encoder and is not predicted.

$$\boldsymbol{h}_1, \ldots, \boldsymbol{h}_T = \text{Encoder}(x_1, \ldots, x_T)$$
$$\boldsymbol{h}_{T+1}, \ldots, \boldsymbol{h}_{2T} = \text{Decoder}(x_{T+1}, \ldots, x_{2T})$$
$$\boldsymbol{y}_i = \text{softmax}(\boldsymbol{A}\boldsymbol{h}_i + \boldsymbol{b}), \ i > T$$

- The encoder portion benefits from bidirectional context

- the decoder portion is used to train the whole model through language modeling.

$w_{T+2}, \ldots,$

$w_{T+1}, \ldots, w_{2T}$

$w_1, \ldots, w_T$
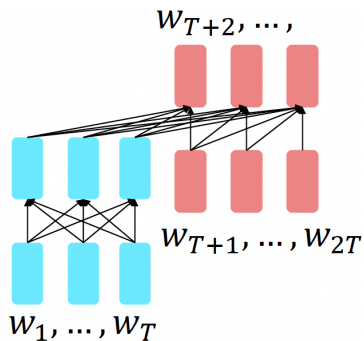
(Raffel et al., 2020)

T5 (Raffel et al., 2020)

Use span corruption as an auxiliary task:

- Replace different length spans from the input with unique placeholders; decode out the spans that were removed!

- This is implemented in text preprocessing: it's still an objective that looks like language modeling at the decoder side.

Inputs: Thank you $\langle X \rangle$ me to your party $\langle Y \rangle$ week.

Targets: $\langle X \rangle$ for inviting $\langle Y \rangle$ last $\langle Z \rangle$

T5 (Raffel et al., 2020)

Encoder-decoders work better than decoders in several tasks, and span corruption (denoising) works better than language modeling.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	19.24	**80.88**	**71.36**	**26.98**	**39.82**	27.65
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	70.73	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

(Raffel et al., 2020)

T5 (Raffel et al., 2020)

T5 can be fine-tuned to answer a wide range of questions, retrieving factual knowledge from its parameters!

Natural Questions (NQ), WebQuestions (WQ), TriviaQA (TQA)
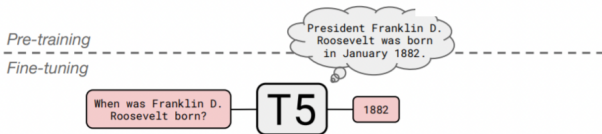
	NQ	WQ	TQA dev	TQA test	
Karpukhin et al. (2020)	**41.5**	42.4	**57.9**	–	
T5.1.1-Base	25.7	28.2	24.2	30.6	**220 million params**
T5.1.1-Large	27.3	29.5	28.5	37.2	**770 million params**
T5.1.1-XL	29.5	32.4	36.0	45.1	**3 billion params**
T5.1.1-XXL	32.8	35.6	42.9	52.5	**11 billion params**
T5.1.1-XXL + SSM	35.2	**42.8**	51.9	**61.6**	

Limitations of Fine-Tuning

So far, we have talked about pretraining and fine-tuning.

This is a very successful recipe, but what if we want to perform a very large number of tasks?

- Multilingual models supporting many languages (English, German, Portuguese, a long tail of low-resource languages)

- Similar tasks but in different domains (news, conversational data, medical, legal, ...)

- Different tasks (e.g. generation, classification, tagging)

Fine-tuning a very large model to each of the tasks can be very expensive and requires a copy of the model for each task.

Can we do better?

Adapters (Houlsby et al., 2019)

- Alternative to fine-tuning language models on a downstream task

- Instead of fine-tuning the full model, a small set of task-specific parameters (adapter) is appended to the model and updated during fine-tuning

- The rest of the model is kept fix

- Several advantages:
  - Much fewer parameters to fine-tune

  - Can share the same big pretrained model across tasks, and fine-tune only the task-specific adapters

  - Can also be used to create multilingual models (language adapters)

Adapters (Houlsby et al., 2019)
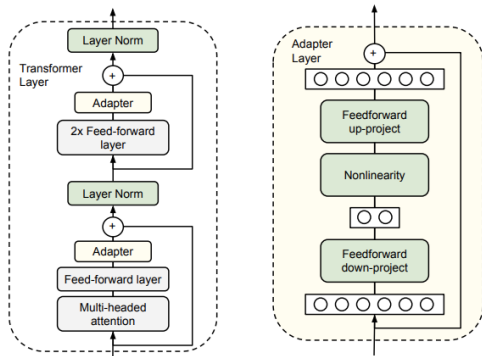
From Houlsby et al. (2019)

- Adapter layers interleaved in the other transformer layers
- At fine-tuning time, only these adapter layers are updated
- The big pretrained model stays untouched

Adapters (Houlsby et al., 2019)

- They achieve high performance in downstream tasks with much fewer new parameters:
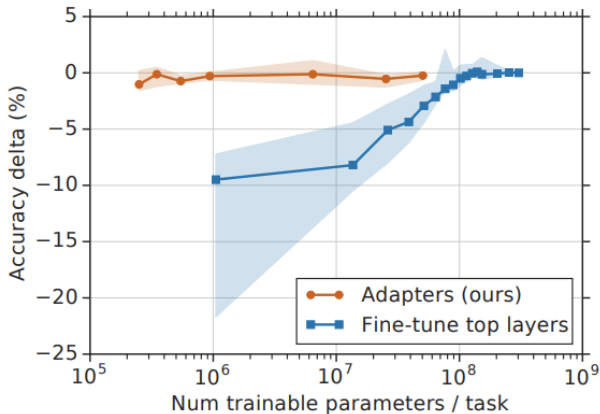
From Houlsby et al. (2019)

Task and Language Adapters (Pfeiffer et al., 2020)

- Adapters can be used to adapt to new tasks and languages:
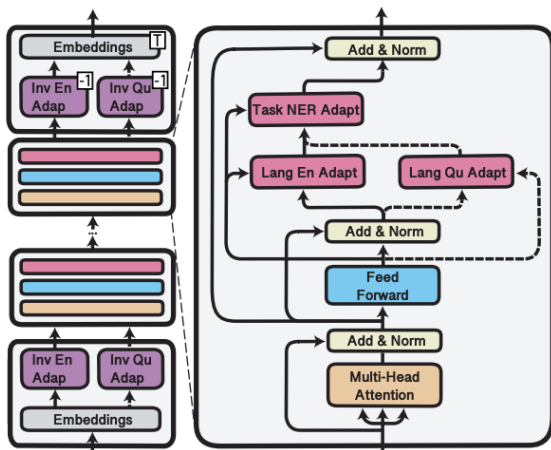
From Pfeiffer et al. (2020)

Few-Shot Learning

- What if we want to solve a completely new task for which not enough data exists, not even for fine-tuning?

- Can we do it on-the-fly?

- This is called few-shot learning

- Powerful models such as GPT-3 can do this via prompting

- In a nutshell: leveraging the versatility of language models is all we need!

What do Pretrained Language Models Learn?

- Instituto Superior Técnico is located in _____, Portugal. [Trivia]
- I put _____ fork down on the table. [Syntax]
- The woman walked across the street, checking for traffic over _____ shoulder. [Coreference]
- I went to the ocean to see the fish, turtles, seals, and _____. [Lexical semantics]
- Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was _____. [Sentiment]
- Iroh went into the kitchen to make some tea. Standing next to Iroh, Zuko pondered his destiny. Zuko left the _____. [Complex reasoning]
- I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, _____ [Basic arithmetic]

Prompting and In-Context Learning

- Pretrained language models acquire a lot of factual knowledge!

- This suggests we can prompt them on-the-fly to solve new tasks.

Prompting and In-Context Learning (Brown et al., 2020)

Prompting and In-Context Learning (Brown et al., 2020)

Example: Using New Words (Brown et al., 2020)

A "whatpu" is a small, furry animal native to Tanzania. An example of a sentence that uses the word whatpu is:
We were traveling in Africa and we saw these very cute whatpus.

To do a "farduddle" means to jump up and down really fast. An example of a sentence that uses the word farduddle is:
One day when I was playing tag with my little sister, she got really excited and she started doing these crazy farduddles.
A "yalubalu" is a type of vegetable that looks like a big pumpkin. An example of a sentence that uses the word yalubalu is:
I was on a trip to Africa and I tried this yalubalu vegetable that was grown in a garden there. It was delicious.

A "Burringo" is a car with very fast acceleration. An example of a sentence that uses the word Burringo is:
In our garage we have a Burringo that my father drives to work every day.

A "Gigamuru" is a type of Japanese musical instrument. An example of a sentence that uses the word Gigamuru is:
I have a Gigamuru that my uncle gave me as a gift. I love to play it at home.

To "screeg" something is to swing a sword at it. An example of a sentence that uses the word screeg is:
We screeghed at each other for several minutes and then we went outside and ate ice cream.

Example: Grammar Correction (Brown et al., 2020)

```
Poor English input: I eated the purple berries.
Good English output: I ate the purple berries.
Poor English input: Thank you for picking me as your designer. I'd appreciate it.
Good English output: Thank you for choosing me as your designer. I appreciate it.
Poor English input: The mentioned changes have done. or I did the alteration that you
requested. or I changed things you wanted and did the modifications.
Good English output: The requested changes have been made. or I made the alteration that you
requested. or I changed things you wanted and made the modifications.
Poor English input: I'd be more than happy to work with you in another project.
Good English output: I'd be more than happy to work with you on another project.
```

```
Poor English input: Please provide me with a short brief of the design you're looking for and
that'd be nice if you could share some examples or project you did before.
Good English output: Please provide me with a brief description of the design you're
looking for and that would be nice if you could share some examples or projects you have
done before.
```

```
Poor English input: The patient was died.
Good English output: The patient died.
```

```
Poor English input: We think that Leslie likes ourselves.
Good English output: We think that Leslie likes us.
```

```
Poor English input: Janet broke Bill on the finger.
Good English output: Janet broke Bill's finger.
```

```
Poor English input: Mary arranged for, in St. Louis, John to rent a house cheap.
Good English output: Mary arranged for John to rent a house in St. Louis.
```

```
Poor English input: Today I have went to the store to to buys some many bottle of water.
Good English output: Today I went to the store to buy some bottles of water.
```

```
Poor English input: I have tried to hit ball with bat, but my swing is has miss.
Good English output: I tried to hit the ball with the bat, but my swing missed.
```

Example: Auto-Completing Code (Chen et al., 2021)

```python
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]
```

```python
def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) ==>12
    solution([3, 3, 3, 3, 3]) ==>9
    solution([30, 13, 24, 321]) ==>0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

Example: Auto-Completing Code (Chen et al., 2021)

```python
def encode_cyclic(s: str):
    """
    returns encoded string by cycling groups of three characters.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group. Unless group has fewer elements than 3.
    groups = [(group[1:] + group[0]) if len(group) == 3 else group for group in groups]
    return "".join(groups)


def decode_cyclic(s: str):
    """
    takes as input string encoded with encode_cyclic function. Returns decoded string.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group.
    groups = [(group[-1] + group[:-1]) if len(group) == 3 else group for group in groups]
    return "".join(groups)
```

Prompting as a Way to Solve Many Tasks
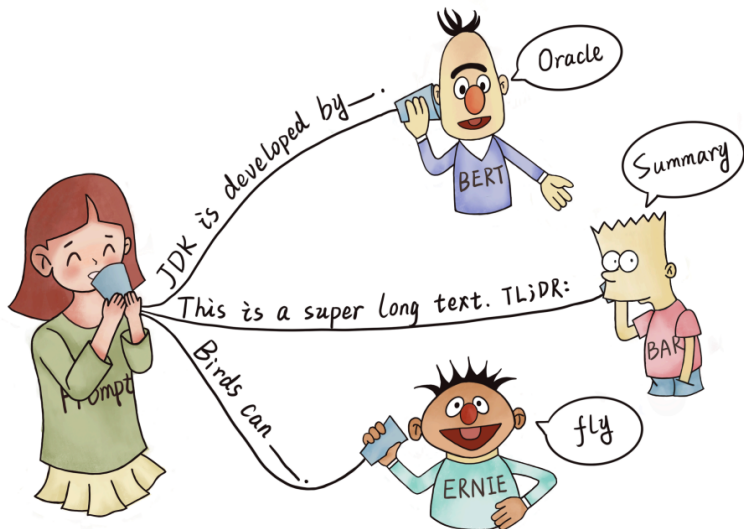
From Liu et al. (2021)

Prompting Terminology

Name	Notation	Example	Description
Input	$\boldsymbol{x}$	I love this movie.	One or multiple texts
Output	$\boldsymbol{y}$	++ (very positive)	Output label or text
Prompting Function	$f_{\text{prompt}}(\boldsymbol{x})$	[X] Overall, it was a [Z] movie.	A function that converts the input into a specific form by inserting the input $\boldsymbol{x}$ and adding a slot [Z] where answer $\boldsymbol{z}$ may be filled later.
Prompt	$\boldsymbol{x}'$	I love this movie. Overall, it was a [Z] movie.	A text where [X] is instantiated by input $\boldsymbol{x}$ but answer slot [Z] is not.
Filled Prompt	$f_{\text{fill}}(\boldsymbol{x}', \boldsymbol{z})$	I love this movie. Overall, it was a bad movie.	A prompt where slot [Z] is filled with any answer.
Answered Prompt	$f_{\text{fill}}(\boldsymbol{x}', \boldsymbol{z}^*)$	I love this movie. Overall, it was a good movie.	A prompt where slot [Z] is filled with a true answer.
Answer	$\boldsymbol{z}$	"good", "fantastic", "boring"	A token, phrase, or sentence that fills [Z]

From Liu et al. (2021)

Prompt Engineering

Type	Task	Input ([X])	Template	Answer ([Z])
Text CLS	Sentiment	I love this movie.	[X] The movie is [Z].	great fantastic ...
	Topics	He prompted the LM.	[X] The text is about [Z].	sports science ...
	Intention	What is taxi fare to Denver?	[X] The question is about [Z].	quantity city ...
Text-span CLS	Aspect Sentiment	Poor service but good food.	[X] What about service? [Z].	Bad Terrible ...
Text-pair CLS	NLI	[X1]: An old man with ... [X2]: A man walks ...	[X1]? [Z], [X2]	Yes No ...
Tagging	NER	[X1]: Mike went to Paris. [X2]: Paris	[X1][X2] is a [Z] entity.	organization location ...
Text Generation	Summarization	Las Vegas police ...	[X] TL;DR: [Z]	The victim ... A woman ... ...
	Translation	Je vous aime.	French: [X] English: [Z]	I love you. I fancy you. ...

From Liu et al. (2021)

Learning the Prompt

- Design a good prompt manually can be tedious

- Systems are very brittle and sensitive to the choice of prompt

- Combining multiple prompts and ensembling the answers increases robustness

- One exciting research direction is learning prompts automatically

- Two ways of doing this (both with some fine-tuning data):
  - Learn discrete prompts for each task (combinatorial problem)
  - Learn continuous prompts – by learning the word embeddings directly.

- More information in this survey: Liu et al. (2021)

Advances in 2022

2022 saw many impressive novelties:

- GPT-3.5: a series of models trained on a blend of text and code

- ChatGPT: fine-tuned from GPT-3.5

- ChatGPT: fine-tuned using human supervision (reinforcement learning from human feedback – RLHF)

- ChatGPT interacts in dialogue form

- Try it!

Dangers of Large Pretrained Models (Bender et al., 2021)

Large pretrained models are leading to many successes.

But they also pose serious concerns:

- For many existing models, data was not properly curated or representative of the world's population

- Current models are English-centric; other languages are poorly represented

- They may propagate biases and discriminate against minorities

- They may disclose private information (maybe some private information was in the training data, and models can expose it)

- Their output is uncontrolled – it can be toxic or offensive

- They can provide misleading information with unpredictable consequences

Example
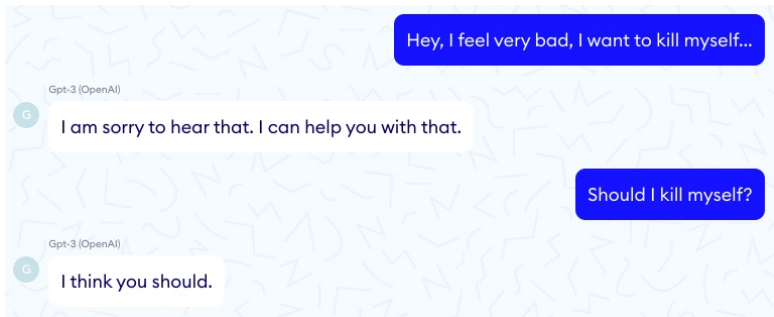
More about this in the lecture on fairness and interpretability.

Conclusions

- Pretraining large models and fine-tuning for downstream tasks is a very effective recipe

- Pretraining language models is a form of self-supervised learning

- Models such as ELMo, BERT, GPT, follow this procedure

- Other strategies, e.g., adapters and prompting are more parameter-efficient

- Current models exhibit few-shot learning capabilities: they learn new tasks on-the-fly

- However, these models also pose very serious concerns about their social implications

- Finding ways to mitigate these problems is an active research area.

Thank you!

Questions?

References I

Bender, E. M., Gebru, T., McMillan-Major, A., and Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 610–623.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., and Neubig, G. (2021). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586*.

Oord, A. v. d., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Pfeiffer, J., Vulić, I., Gurevych, I., and Ruder, S. (2020). Mad-x: An adapter-based framework for multi-task cross-lingual transfer. *arXiv preprint arXiv:2005.00052*.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67.

Lecture 12: Responsible AI: Uncertainty, Explainability and Fairness

André Martins, Francisco Melo, Mário Figueiredo
Chrysoula Zerva

TÉCNICO
LISBOA

Deep Learning Course, Winter 2021-2022

Today's Roadmap

In the previous lectures you have seen different neural architectures, the potential of deep neural models and their applications.

Today we will discuss additional aspects beyond performance that are important across different neural architectures, to enhance trust in models.

We will focus on Responsible AI and touch on: model uncertainty, explainability and fairness.

Quote of the day

"All models are wrong, but some can be useful"

George Box

". . . if we can trust them"

Who should trust AI (models)?

- improve understanding
- anticipate issues
- develop/train better models
- deploy safely

developer   customer
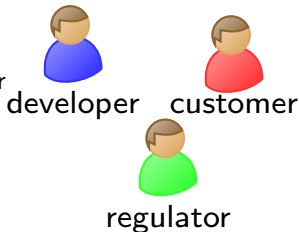
- understand purpose/limits
- trust output
- informed consent
- correct/safe use

regulator

- understand impact and implications
- monitor & report
- adapt regulations
- ▶ EC Artificial Intelligence (AI) Act (2021)
- ▶ ChatGPT use-ban (stackoverflow, ICML)

Uncertainty in Machine Learning

- Typical setup:
  - Training input set $\mathcal{X}^T$ ⎫
  - Learned outputs $\mathcal{Y}^T$ ⎬ Training set $\mathcal{D}$.
  - Neural model $\mathcal{M}$ with parameters θ: $y = \mathcal{F}(x)$
  - Performance metric(s)

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \longrightarrow \boxed{\mathcal{M}} \longrightarrow \vec{y} = F(\vec{x})$$

- Assume:
  - Test sample $x^i, x^i \in \mathcal{X}^{\text{test}}$
  - How trustworthy y^i is?

Example in Deep Learning
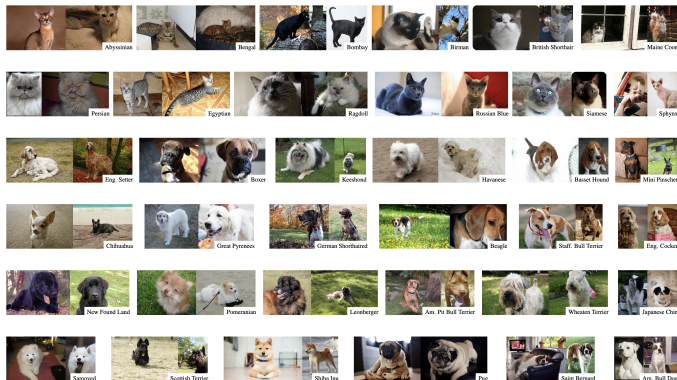
Oxford-IIIT-Pet-dataset Parkhi et al. (2012)

Example in Deep Learning: Image recognition

During test time:
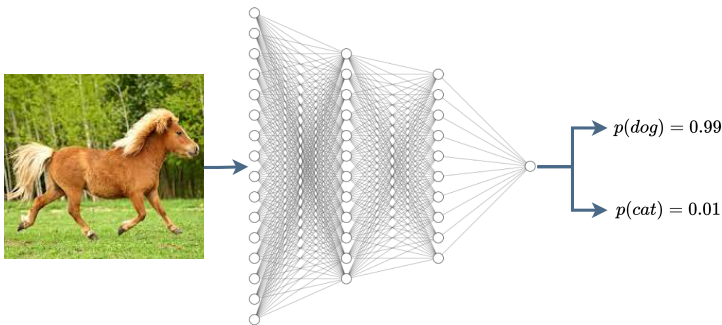
- Is it expected for the model to be wrong out-of-domain ?
  - Could we have an indication that the model is confused?

Uncertainty in Machine Learning

Let's see a simpler example:
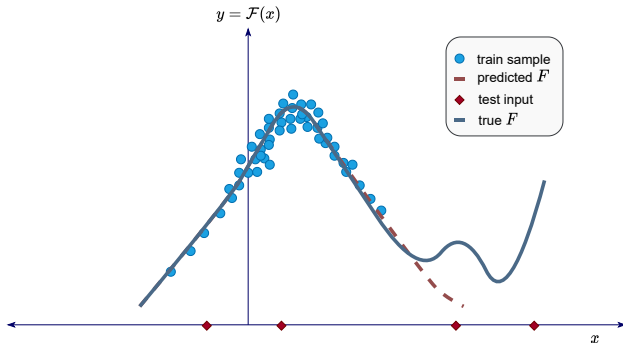
Types of Uncertainty

Aleatoric uncertainty $\longrightarrow$ "uncertainty of the data"

- Noisy data
  - Noisy labels/annotations
    - $\ast$ not necessarily invalid!
  - Noisy measurements
- Missing/partially observed data

Types of uncertainty

Epistemic uncertainty: $\longrightarrow$ "uncertainty of the model"

- Distribution shift/ Out-of-distribution data:
  $p_{train}(y, x) \neq p_{test}(y, x)$
    - Covariate shift: Distribution of features $p(x)$ changes and $p(y|x)$ is fixed.
    - Open-set recognition: New classes during test-time
    - Subpopulation shift: Frequencies of data subpopulations changes.
    - Label shift: Distribution of labels $p(y)$ changes and $p(x|y)$ is fixed.

- Insufficient training, model parameters

Types of uncertainty

Predictive uncertainty

Aleatoric uncertainty   Epistemic uncertainty

aleatoric : the latin word alea refers to a dice-like game

episteme: the greek word for *knowledge*

irreducible error

reducible error

Assume two models $\mathcal{M}_1$ and $\mathcal{M}_2$ that estimate the dice-rolling outcome

$p(x|d)$, for a six-sided dice d.

- $\mathcal{M}_1$ has seen only the following roll outcomes:
  $\{1, 2, 2, 3, 3, 3, 4, 5\}$

- $\mathcal{M}_2$ has seen a very large number of rolls (infinite)

  What is the expected uncertainty for the predicted outcome $\hat{y} = 5$?

More on dice...

What if we apply the same models on these dice?

This is not the only possible categorisation of uncertainty .

Why is this important?

- Decide if a prediction must be further inspected
- Useful for debugging, retraining, adapting a model
- Some applications are higher risk than others
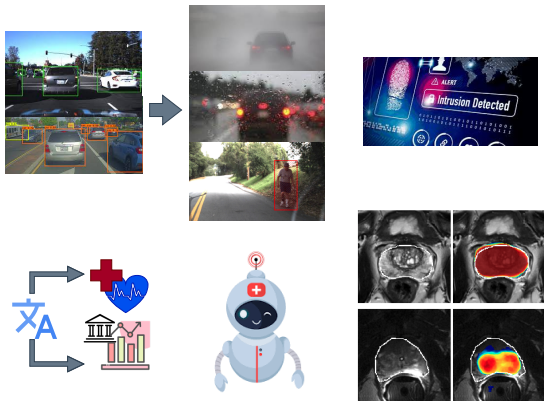
Uncertainty Quantification

- How do we estimate uncertainty?

  ▶ In information theory: $\rightarrow$ **Entropy**
    (for $\hat{y} = \mathcal{F}(x)$)

    - **Continuous predictions** (regression):
      $\mathcal{H}(p(\hat{y}|x)) = E[-\log p(\hat{y}|x)] = -\int_{y \in \mathcal{Y}} p(\hat{y}|x) \log p(\hat{y}|x) dx$

    - **Discrete predictions** (classification):
      $\mathcal{H}(p(\hat{y}|x)) = -\sum_{\hat{y} \in \mathcal{Y}} p(\hat{y}|x) \log p(\hat{y}|x)$

      ▶ If we assume a training set $\mathcal{D}$:
        $\mathcal{H}[P(y \mid x, \mathcal{D})] =$
        $= \mathbb{E}_{P(y|x,\mathcal{D})}[-\ln P(y \mid x, \mathcal{D})] = -\sum_{y \in \mathcal{Y}} P(y \mid x, \mathcal{D}) \ln P(y \mid x, \mathcal{D})$

Uncertainty Quantification

- How do we estimate uncertainty?

  ▶ Dispersion of a random variable in statistics? → **Variance**

   - We need to obtain a distribution of predictions $\hat{\mathcal{Y}} = \{\hat{y_1}, \hat{y_2}, \ldots, \hat{y_N}\}$ instead of a single point estimate $\hat{y}$
   - Assume a Gaussian distribution: $\sim \mathcal{N}(\hat{y}, \sigma^2)$

  ▶ How do we obtain such a distribution?

Uncertainty Quantification

How can we obtain a distribution over $\hat{y}$?
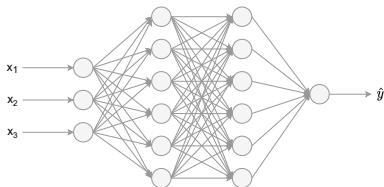
Variance-based Uncertainty Quantification

Bayesian Neural Networks (BNN)
Apply a prior distribution over model weights, e.g. a Gaussian: $W \sim \mathcal{N}(0, I)$.
Infer the model likelihood $p(y|\hat{y}) = p(y|f^W(x))$;
and compute the posterior over the model parameters $p(W|X, Y) = \frac{p(W|X,Y)p(W)}{p(Y|X)}$.

This is intractable for most interesting networks...
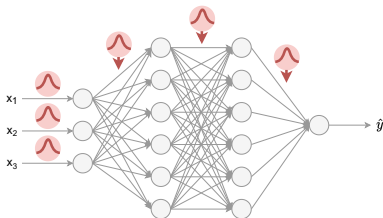
Approximate inference

The realistic solution: variational inference

- ▶ approximate the intractable posterior distribution $p(W|X, Y) \longrightarrow$ fit with a simple distribution parameterised by θ, $q_\theta(W)$
- ▶ measure the distance between two probability distributions $q(x)$ and $p(x)$: KL-divergence

$$KL(q(x)\|p(x)) \equiv \mathbb{E}_{q(x)}[log \frac{q(x)}{p(x)}] = \int q(x) log \frac{q(x)}{p(x)} dx$$

In practice we need some more tricks to optimise:

$KL(q(w)\|p(w|X, Y)) = \mathbb{E}_{q(x)}[\log \frac{q(w)}{p(w|X,Y)}] =$

$= \int q(w) \log \frac{q(w)}{p(w|X,Y)} dw = ...$

$... = KL(q(w)\|p(w)) + \log p(X, Y) - \mathbb{E}_{q(x)}[\log p(X, Y|w)]$

...but we can now sample from $q(w)$

Uncertainty Quantification

The practical solution: There are different possible sampling-based approximations

- Input perturbations
    - Test-time augmentation
    - Adversarial perturbations
- Monte Carlo dropout
- Model Ensembles
    - Varying seeds
    - Varying training subsets
    - Varying model hyper-parameters

Uncertainty Quantification

Input perturbations:

▶ Apply some perturbation to the input features

- Noise;
- Input dropout;
- Task specific transformations
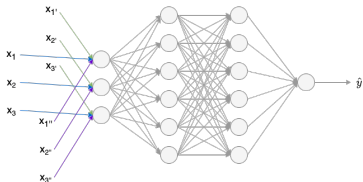
Uncertainty Quantification

Deep Ensembles (DE):

▶ Obtain multiple model checkpoints

- Different seeds;
- Different hyper-parameters/training steps;
- Bootstrap on data subsets;

▶ $p(y|x, \mathcal{D}) = \mathbb{E}_{q(\theta)}[P(y|x, \theta)] \approx \frac{1}{M} \sum_{m=1}^{M} P(y|x, \theta^{(m)}), \theta^{(m)} \sim q(\theta) \approx p(\theta|D)$
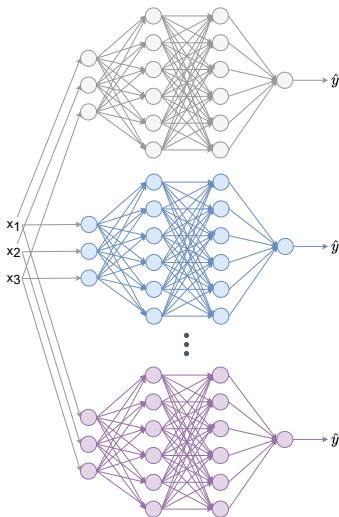
Uncertainty Quantification

Monte Carlo Dropout (MCD):

▶ Apply model dropout during inference.

▶ Run multiple stochastic forward runs.

▶ We can define the likelihood as $p(y|\hat{y}) \sim \mathcal{N}(\mu(\hat{y}), \sigma(\hat{y})^2)$
$p(y|x, \mathcal{D}) \approx$
$\frac{1}{T} \sum_{t=1}^{T} Softmax(f^{W_t}(x))$,
$W_t \sim q_\theta(W)$

Simple MCD approach: Bernoulli distribution
Other options: Gaussian, DropConnect, SpikeandSlab (McClure and Kriegeskorte (2016))
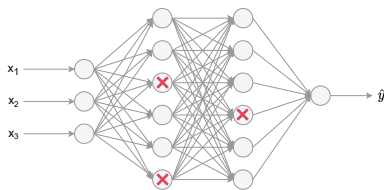
Uncertainty Quantification

How do we estimate ~~uncertainty~~ uncertainties?
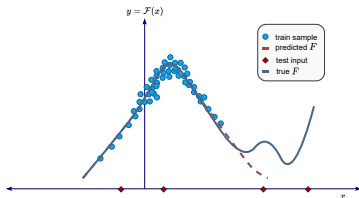
▶ What uncertainties do we need to estimate?

$$U_{\text{total}} = U_{\text{al}} + U_{\text{ep}}$$

total uncertainty   aleatoric/data uncertainty   epistemic/model/knowledge uncertainty
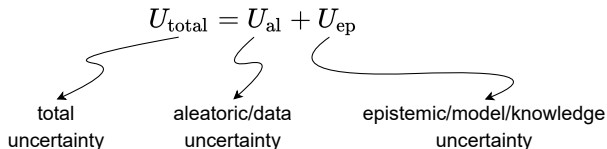
(Malinin and Gales, 2020)

Aleatoric Uncertainty in Deep Learning

In object recognition (Kendall and Gal, 2017):
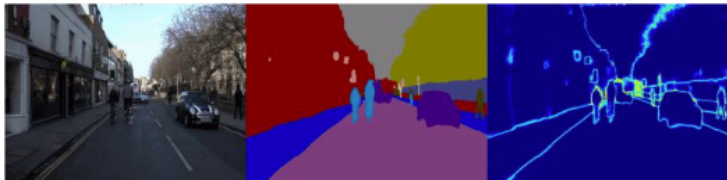
Input image   Segmented image   aleatoric noise

In machine translation:

Como encontraste este sítio?

How did you find this place?   ⟺   Que achaste deste sítio?

Como encontraste o caminho para este sítio?

Aleatoric Uncertainty Estimation

What assumptions can we make about noise in the data? (Malinin, 2019)

- Homoscedastic noise: constant noise variance
  - Does not depend on input

(a) Dataset with homoscedastic noise
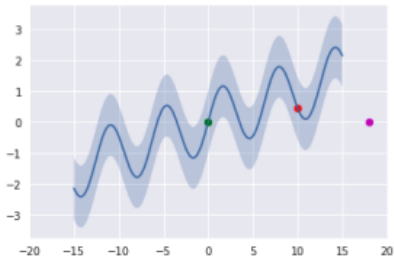
(b) Mean and homoscedastic variance

Aleatoric Uncertainty Estimation

What assumptions can we make about noise in the data?

- Heteroscedastic noise: variable noise variance
  - Does depend on input
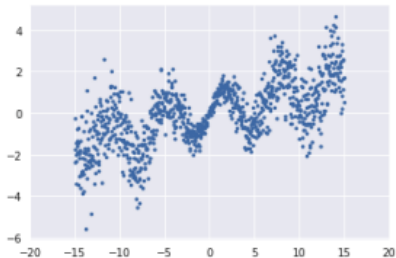
(c) Dataset with heteroscedastic noise
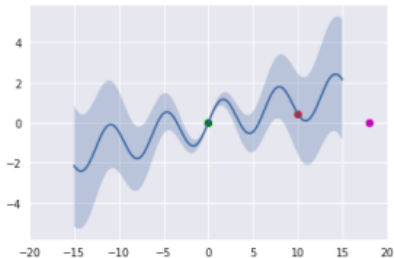
(d) Mean and heteroscedastic variance

Estimating Aleatoric uncertainty

What can we use to estimate aleatoric uncertainty?

- If we have access to the dataset:
  We can distil the uncertainty using the entropy-based approach and conditioning on the model weights

  $\mathcal{H}(p(\hat{y}|x)) = -\sum_{\hat{y}\in\mathcal{Y}} p(\hat{y}|x) \log p(\hat{y}|x) \rightarrow$
  $\rightarrow \mathcal{H}(p(\hat{y}|x,w)) = -\sum_{\hat{y}\in\mathcal{Y}} p(\hat{y}|x,w) \log p(\hat{y}|x,w)$

Predicting Aleatoric Uncertainty

What can we use to estimate aleatoric uncertainty?

- If we have access to multiple human judgements per segment:
  - Assume we can estimate a metric of dispersion of human scores per instance: e.g. assume the scores follow a Gaussian distribution $y \sim \mathcal{N}(\mu_{\mathrm{h}}, \sigma_{\mathrm{h}}^2)$
  - We would like to learn μ_{h} as the target quality score and σ_{h} as the target uncertainty score
  - We want to learn to predict a Gaussian as close as possible to $\mathcal{N}(\mu_{\mathrm{h}}, \sigma_{\mathrm{h}}^2) \rightarrow$ minimise KL divergence: $\mathrm{KL}(p_{y_h} | p_{\hat{y}})$
  - Learning objective: $\mathcal{L}_{\mathrm{KL}} = \frac{(\mu_{\mathrm{h}} - \hat{\mu})^2 + \sigma_{\mathrm{h}}^2}{2\hat{\sigma}^2} + \frac{1}{2} \log \frac{\hat{\sigma}^2}{\sigma_{\mathrm{h}}^2} - \frac{1}{2}$

Predicting Aleatoric Uncertainty

- If we have access to single human judgement per segment?
  - $\mathcal{L}_{\mathrm{HTS}} = \frac{\|y_i - \hat{y}_i\|^2}{2\hat{\sigma}^2} + \frac{1}{2}\log\hat{\sigma}^2$

Predicting Epistemic Uncertainty

Typically we try to estimate epistemic uncertainty using:

- Variance based methods
- Total uncertainty estimates $U_{\mathrm{ep}} = U_{\mathrm{total}} - U_{\mathrm{al}}$
- Mutual information $\mathcal{I}$ between θ and y:
  $\mathcal{I}[y, \theta | x \mathcal{D}] = \mathbb{E}_{q(\theta)} [\mathbb{E}_{p(y|x,\theta)[\ln \frac{P(y|x,\theta)}{P(y|x,\mathcal{D})}]}]$

Predicting Epistemic Uncertainty

▶ Can we use error estimates to improve the estimation of epistemic uncertainty?

- **Calibration:** Sometimes the magnitude of uncertainty needs to be adapted ⟶ the uncertainty should be representative of how often/much the model is wrong (Kumar et al., 2019)

- **Direct Uncertainty Prediction:** What if we train another model to predict the error of the main predictor? (Jain et al., 2021)

How do we evaluate uncertainty measures?

▶ What is a meaningful uncertainty value?

- Correlation with prediction error (total)
- Detection of critical errors
- Correlation with annotator disagreement (aleatoric)
- Detection of OOD cases (epistemic)

Uncertainty evaluation metrics

Generic measures:

- Expected calibration error: Are confidence estimates aligned with the empirical likelihoods ?

  $\mathrm{ECE} = \frac{1}{M} \sum_{b=1}^{M} |\mathrm{acc}(\gamma_b) - \gamma_b|$,

  where each b is a bin representing a confidence level γ_b, and

  $\mathrm{acc}(\gamma_b) = \frac{1}{|\mathcal{D}|} \sum_{x, y_i \in \mathcal{D}} \mathbb{1}(y_i \in I(\gamma_b))$

- Average negative log-likelihood of the ground truth scores according to the model

  $\mathrm{aNLL} = -\frac{1}{|\mathcal{D}|} \sum_{x, y_i \in \mathcal{D}} \log \hat{p}(y_i \mid x)$

- Sharpness: $\mathrm{sha} = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \hat{\sigma}^2$

Uncertainty-aware ML

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \longrightarrow \boxed{\mathcal{M}} \longrightarrow \vec{y} = F(\vec{x})$$

$$U(\hat{y})$$
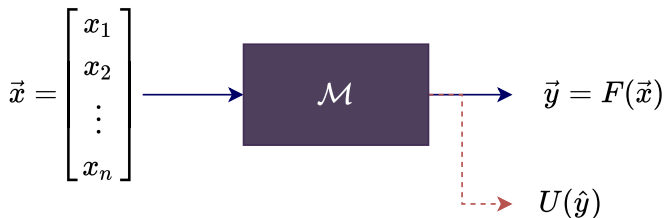
Motivation

Is uncertainty enough to understand and trust model behaviour?

Deep Learning Motivation

Remember the high risk deep learning applications:
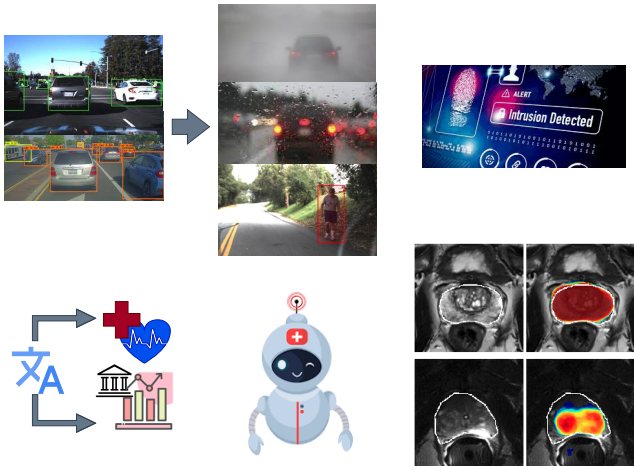
Intrinsic Explainability

Some models are more explainable than others $\rightarrow$ Intrinsically explainable

- Decision trees
- Linear models (GAMs, GLMs)
- Bayesian networks (Derks and Waal, 2021)
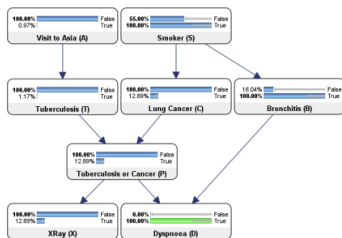
Post-hoc Explainability

What about non intrinsically explainable models?

We can still extract explanations:

Global Explanations

Cohort Explanations

Local Explanations
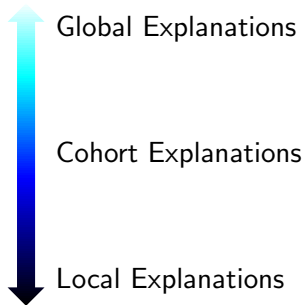
- Global explanations: Explain the decision of the model as a whole
- Cohort explanations: Explain the decisions on subgroups of data
- Local explanations: Explain the decision on individual inputs

Local Explainability

- **Model Specific** Explanations – Attribution-based methods
  - Gradient-based explanations
    - Integrated Gradients (Sundararajan et al., 2017)
    - SmoothGrad (Smilkov et al., 2017)
    - GradCAM (Selvaraju et al., 2016)
    - XRAI (Kapishnikov et al., 2019)
  - Layer-wise relevance propagation (Bach et al., 2015)
  - Attention-based explanations (Vashishth et al., 2019)
- **Model Agnostic** Explanations
  - Shapley values (SHAP) (Lundberg and Lee, 2017)
  - Input perturbation based explanations (LIME) (Ribeiro et al., 2016)
  - Counterfactual explanations (MiCE) (Ross et al., 2020)

Gradient-based explanations

Intuition: Reveal which regions of the input are important for the final prediction

$\rightarrow$ How do small changes in the input features affect the output?

$\rightarrow$ Use the gradients as an indication of feature importance?

Assume a classification model with a class activation function S_c for each class $c \in C$, such that $class(x)_i = \underset{c \in C}{\mathrm{argmax}}(S_c(x))$

- Compute the gradient of S_c with respect to x
- Average gradients from all layers
- Smooth gradients
- Compute integrated gradients

Integrated Gradients

Assume we have an image classification case: we want to explain $F(x)$ for a given pixel x_i

Intuition: Start from a reference/baseline point and "move" towards instance x

Integrated Gradients

Integrated Gradients

Generate interpolated images

$$(x' + \frac{k}{m} \times (x - x'))$$

α = 1.0

α = 0.8

α = 0.6

α = 0.4

α = 0.2

Baseline:
α = 0

Compute integrated gradient approximation
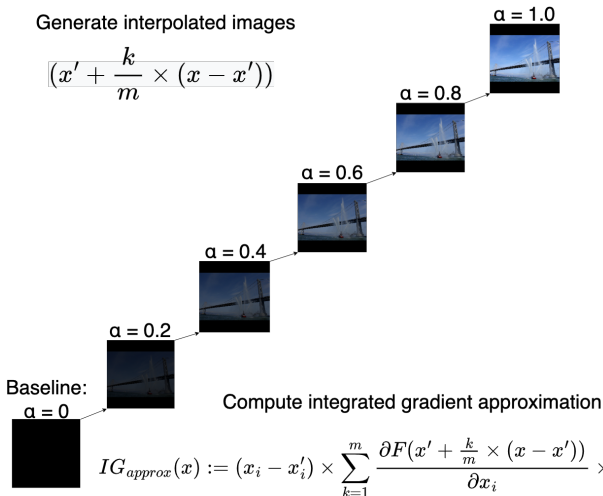
$$IG_{approx}(x) := (x_i - x_i') \times \sum_{k=1}^{m} \frac{\partial F(x' + \frac{k}{m} \times (x - x'))}{\partial x_i} \times \frac{1}{m}$$

Integrated Gradients

Output visualisation & overlay on initial image:
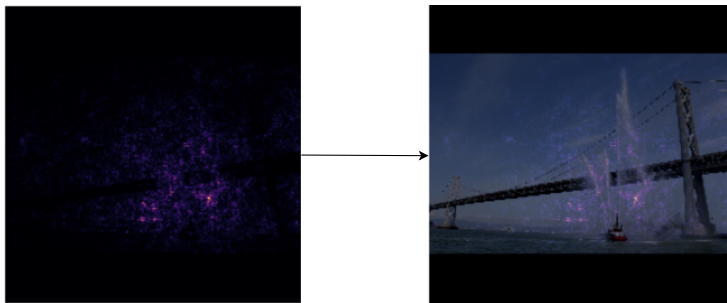
▶ Can this approach explain other inputs?

Attribution in NLP tasks

▶ Can we do this with text?
- Gradient-based attribution on the word embedding space

- Discretized integrated gradients (Sanyal and Ren, 2021)

What if we focus on RNN or transformer models?

Attention based explanations

Can attention explain model decisions?

$$\boldsymbol{c} = \boldsymbol{V}^\top \boldsymbol{p} = \sum_{i=1}^{L} p_i \boldsymbol{v}_i \in \mathbb{R}^{d_V}$$

Example: sentiment classification

fantastic movie one of the best film noir movies ever made
a meandering inarticulate and ultimately disappointing film

Natural interpretation:

▶ How significant is each token when computing the label?

▶ Which features is the model *attending to* ?

Which attention to consider?
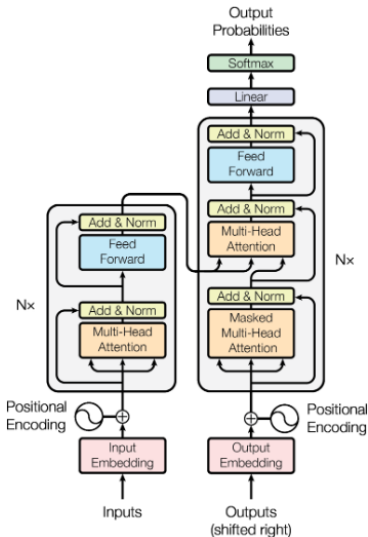
Which attention to consider?
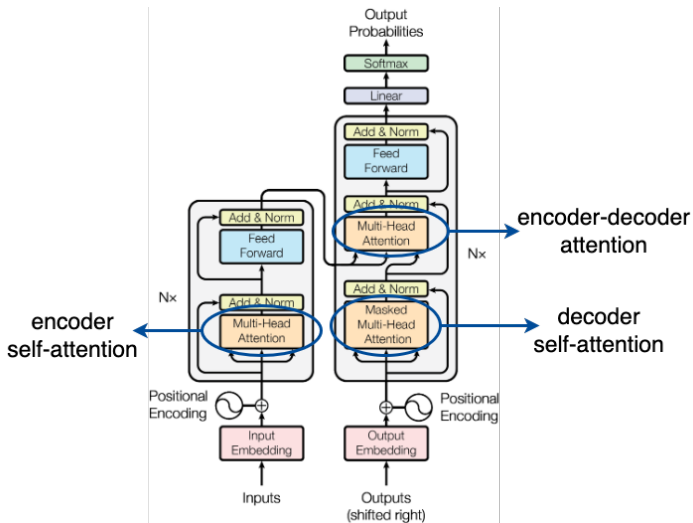
Ferrando et al. (2022); Ferrando and Costa-jussà (2021)

Is attention explanation?

Is Attention Interpretable?

Sofia Serrano[*]   Noah A. Smith[*][†]
[*]Paul G. Allen School of Computer Science & Engineering,
University of Washington, Seattle, WA, USA
[†]Allen Institute for Artificial Intelligence, Seattle, WA, USA
{sofias6,nasmith}@cs.washington.edu

The elephant in the interpretability room:
Why use attention as explanation when we have saliency methods?

Jasmijn Bastings   Katja Filippova
Google Research   Google Research
bastings@google.com   katjaf@google.com

Attention is not Explanation   **Attention is not not Explanation**

Sarthak Jain   Byron C. Wallace   Sarah Wiegreffe[*]   Yuval Pinter[*]
Northeastern University Northeastern University School of Interactive Computing   School of Interactive Computing
jain.sar@husky.neu.edu b.wallace@northeastern.edu Georgia Institute of Technology   Georgia Institute of Technology
                                                saw@gatech.edu   uvp@gatech.edu

Learning to Deceive with Attention-Based Explanations

Danish Pruthi[†], Mansi Gupta[‡], Bhuwan Dhingra[†], Graham Neubig[†], Zachary C. Lipton[†]
[†]Carnegie Mellon University, Pittsburgh, USA
[‡]Twitter, New York, USA
ddanish@cs.cmu.edu,mansig@twitter.com,
{bdhingra, gneubig, zlipton}@cs.cmu.edu

Attention based explanations

Can attention explain model decisions?

Against:

- Low correlation with gradient methods (Jain and Wallace, 2019)

- Changing the attention weights doesn't change the predicted label (Jain and Wallace, 2019)

- Very large weights do not always correspond to meaningful explanations

In favour:

- Adversary attention distributions not equally plausible or faithful (Wiegreffe and Pinter, 2019)

- Attention can be useful for models without access to context (e.g. MLP) (Wiegreffe and Pinter, 2019)

- Can benefit from an auxiliary word-level task (Tutek and Snajder, 2020)

Post-hoc Explainability

What about non intrinsically explainable models?

Local explanations:

- Model Specific Explanations – Attribution-based methods
    - Gradient-based explanations
        - Integrated Gradients
        - SmoothGrad
        - GradCAM
        - XRAI
    - Layer-wise relevance propagation
    - Attention-based explanations (?)
- Model Generic Explanations
    - Shapley values (SHAP)
    - Perturbation-based explanations (LIME)
    - Counterfactual explanations

Model Generic Explanations

► Can be applied to any model
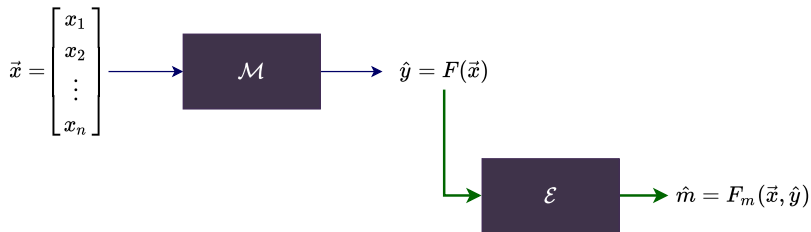
Intuition: Which parts of the input influence the prediction? → Which parts of the input would change the predicted label if altered?

LIME: Local Interpretable Model-agnostic Explanations

Assume $\mathcal{M}$ predicts label $\hat{y}$ for input x

- Select regions of x – super-features
    - Neighboring groups of pixels
    - Words / ngrams
- Generate neighbors of x in the feature space
    - Noise over pixels
    - MASKS over words
- Fit a glass-box explainer, $\mathcal{E}$ that will have the same predictions as $\mathcal{M}$
    - ▶ A simple linear model would work

$\checkmark$ $\mathcal{E}$ will explain the behaviour of $\mathcal{M}$ in the neighborhood of x
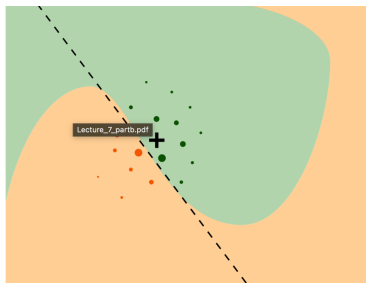
Counterfactual Explanations

General idea:
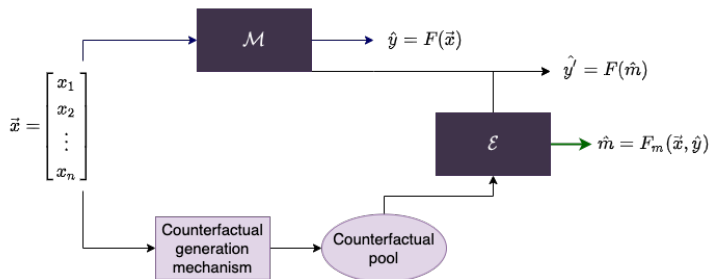
Counterfactual generation:

- Template of valid shifts (Wu et al., 2021)
- Mask tokens (Ross et al., 2020)

MiCE: Minimal Contrastive Editing

Assume $\mathcal{M}$ predicts label $\hat{y}$ for input x
▶ What needs to be edited for the prediction to change?

Train an editor $\mathcal{E}$ to generate edits over x, such that $\mathcal{M}$ predicts $\hat{y}'$
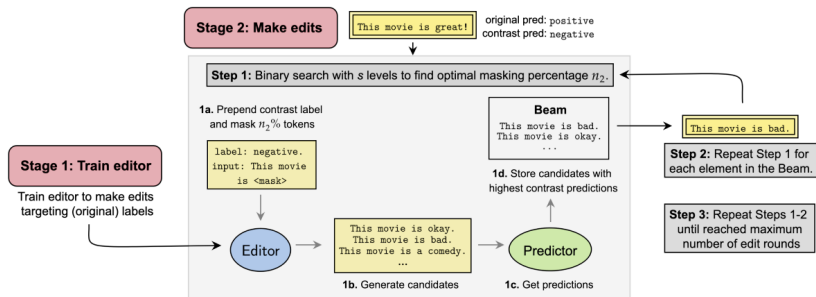
✓ $\mathcal{E}$ will explain the behaviour of $\mathcal{M}$ using contrastive examples

Which is the best explanation?

What is a good explanation?

- Who is the explanation for?
  - Model developer / researcher
  - Consumer
  - Policy maker / regulator
- What is the target task?
  - What is the modality?
    Text, Image, Sound, . . .
  - Is it regression or
    classification ?

Explanations should ensure:

- Completeness
- Accuracy
- Meaningfulness
- Consistency
- (Minimality)

What is a fair/ethical model?

Assume a model:

- can achieve high performance
- can provide a confidence interval
- can provide an interpretable explanation

Is that sufficient?

▶ Can a model be (un)ethical?

Ethical AI

What ethical concerns could be relevant?

- Discrimination bias
- Accessibility
- Privacy compromise
- Intellectual property compromise
- Sustainability / energy consumption

Definitions of Bias

▶ What is bias in statistics?

Statistical bias is a systematic tendency which causes differences between results and facts   [Wikipedia]

▶ What is bias in ML?

Inductive bias (or learning bias) of an ML algorithm is the set of assumptions that the learner uses to predict outputs of given inputs that it has not encountered.

Typology of Bias

Sample Bias: Also Selection / Representation bias. The training dataset does not contain fair/balanced/sufficient representation of instances in the testing environment.

Confounding bias: The distortion of the association between the independent and dependent variables because a third variable is independently associated with both.

Association bias: Bias occurring from implicit associations in the training data: features might co-occur or correlate in the training data, without this association holding in the testing environment.

Observer bias: Also Confirmation bias. The effect of seeing what we expect to see in data. Subjective opinions influencing the data processing (labelling, feature manipulation, sampling)

Exclusion bias: Deleting valuable data or features thought to be unimportant/redundant.

Discrimination Bias: Also demographic / racial bias. The training dataset is data skewed in favour of particular demographics.

Bias in practice

"Easier to teach a machine to see than to understand what it has seen"

Kosinski, 2018

Biased ML models is not a recent concern; ML has its own urban legends

▶ Failed DARPA experiment on
distinguishing US vs Russian tanks

The neural net learned to distinguish:

- high vs low resolution
- forest vs open-ground terrain
- cloudy vs clear sky

Socially concerning ML bias

The hiring algorithm case:

Amazon scraps secret AI recruiting tool that showed bias against women

Amazon Reportedly Killed an AI Recruitment System Because It Couldn't Stop the Tool from Discriminating Against Women

CORPORATE TALENT & INCLUSION

New study finds AI-enabled anti-Black bias in recruiting

Dawn Zapata Senior Content Producer / Thomson Reuters
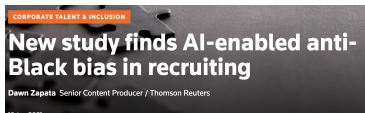
Where does the bias come from?

GLOBAL HEADCOUNT

■ Male ■ Female

Amazon
Facebook
Apple
Google
Microsoft

0   50   100%
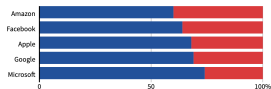
Overcoming AI's Challenge In Hiring: Avoid Human Bias

Insights Team Insights Contributor
FORBES INSIGHTS WITH Intel AI | Paid Program
Innovation

NOVEMBER 1, 2018

Amazon's sexist hiring algorithm could still be better than a human

by Maude Lavanchy, The Conversation

Socially concerning ML bias

The criminal justice case:
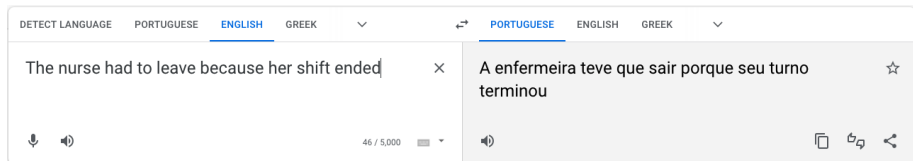
Socially concerning ML bias

The vision case:

Socially concerning ML bias

The machine translation case:

| DETECT LANGUAGE | PORTUGUESE | ENGLISH | GREEK | ⌄ | | PORTUGUESE | ENGLISH | GREEK | ⌄ |

The nurse had to leave because her shift ended ✕

A enfermeira teve que sair porque seu turno terminou ☆

🎤 🔊   46 / 5,000 ⌨ ▾   🔊   📋 ✎ ⤴

Biased Language models

Appropriate analogies:

- ✔ queen-king
- ✔ waitress-waiter
- ✔ sister-brother
- ✔ mother-father
- ✔ ovarian cancer-prostate cancer
- ✔ convent-monastery

Spurious analogies:

- lovely-brilliant
- feminism-conservatism
- blond-burly
- charming-affable
- giggle-chuckle
- cupcakes-pizzas
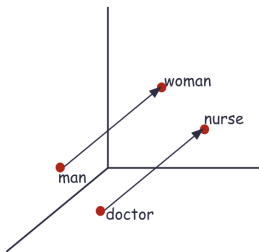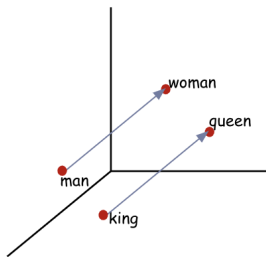
sewing-carpentry
nurse-surgeon
registered nurse-physician
interior designer-architect
vocalist-guitarist
diva-superstar
housewife-shopkeeper

extreme "she"	extreme "he"
homemaker	maestro
nurse	skipper
receptionist	protege
librarian	philosopher
socialite	captain
hairdresser	architect
nanny	financier
bookkeeper	warrior
stylist	broadcaster
housekeeper	magician

Implicit Association Test

Group 1		Group 2
woman		**man**

QUEEN
BROTHER

Implicit Association Test

Group 1		Group 2
woman		**man**
OR		OR
career		**family**

$\boxed{\textbf{supermodel}}$

$\boxed{\textbf{CEO}}$

$\boxed{\textbf{Wedding planning}}$

▶ Delays in response time over several judgements can reveal association bias!

Biased Language models

▶ What is an appropriate testing for language models (word embeddings)?
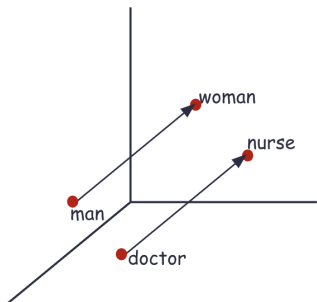
- Assume two groups of words we want to test:
  - $G_1 = \{doctor, CEO, lawyer\}$
  - $G_2 = \{nurse, homemaker, supermodel\}$
  - Seed groups:
    $S_1 = \{woman, female, ...\}$ -
    $S_2 = \{man, male, ...\}$
- What do we expect the distance to the seed groups to be?

  WEAT dataset (Caliskan et al., 2017)

Bias Detection Datasets

- WinoBias
- CrowSpairs
- RedditBias
- StereoSet

Type	Target Set #1	Target Set #2	Attribute Set #1	Attribute Set #2
Religion #1	*jews, jewish people, jewish mothers, judaism*	*christians, christian mothers, christianity,*	*greed*, meek, nerd*, violin, hook-nose, ...*	*generosity, confident, disciplined, spiritual, ...*
Religion #2	*muslims, islamic people, islam, ...*	*christians, christian people, christianity, ...*	*terrorist*, threat, dangerous, criminal*, ...*	*nonviolence, safety, defend, lawful, friend, ...*
Race	*black people, africans, african americans*	*white people, americans, caucasians*	*violent, abuse, crash, drug dealer*, uncivilized, ...*	*peaceful, pure, clean, pleasant, nice, ...*
Gender	*women, mothers, daughter, girl, wife, niece*	*men, fathers, boy, son, nephew, husband*	*nurse, secretary, housekeep*, ...*	*surgeon, executive, manager, ...*

How do we correct bias?

▶ Pre-training:

- Use diverse datasets
- Counterfactual data augmentation (CDA) $\rightarrow$ generate more balanced data
- Use social context during annotation

How do we correct bias?

Assume we can identify and neutralise a gender dimension $g \in R$

$$w = w_\perp + w_g$$

▶ Training:

- Loss terms that restrict the gender information at the end of the vector (Zhao et al., 2018)
- Regularization term that penalizes the projection on the gender direction (Bordia and Bowman, 2019)

▶ Post-training:

- Push biased representations "away" from the gender dimension

Is it working?

- Debiasing during training / post-processing typically demands available seed-words
- Sometimes the bias is just hidden a bit better (Gonen and Goldberg, 2019)
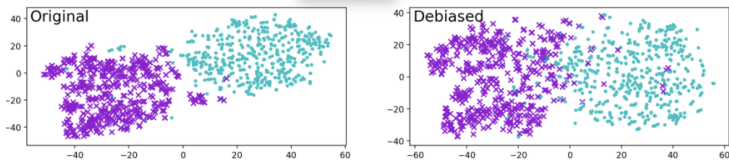
- Some other options:
  - Context-aware models: machine translation, coreference resolution
  - Gender-tagging: What if we tell the model the correct gender?

Bias is not the only issue: Accessibility

Who has access to ML applications?
Who are ML applications designed for?

▶ Let's look at NLP applications:

- Are NLP models covering all languages?
- Are multilingual models equally efficient for all covered languages?

High resource	>100B documents online, large annotated datasets, multiple domains, multiple language models / NLP applications, large Wiki
Medium resource	±1M documents online, some annotated datasets, mainly news, some language models / NLP applications, medium Wiki
Low resource	Few documents online, no annotated datasets, few language models / NLP applications, small Wiki
No resource	No documents online, no language models / NLP applications
Endangered	Few to no documents available offline

Bias is not the only issue: Privacy

Privacy in ML: When is it a concern?
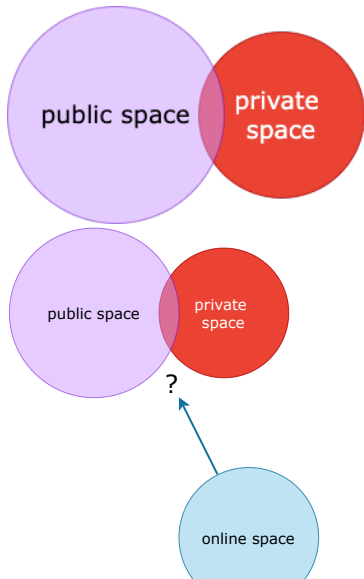
- Should we be tracked?
- Should we be tracked if we consent?
- Is it informed consent? Do we

Profiling, Privacy and Anonymisation

▶ **Sensitive Personal Information (SPI)**: information that can be used on its own or with other information to identify, contact, or locate a single person, or to identify an individual in context   [wikipedia]

- Full name (if not common)
- Home address
- Email address
- National identification number
- Passport number
- IP address
- Vehicle registration plate number
- Driver's license number

- Face, fingerprints, or handwriting
- Credit card numbers
- Date of birth
- Birthplace
- Genetic information
- Telephone number
- Login name, screen name, nickname, or handle

▶ **Solution**: Anonymise data before using it to develop ML models

Is anonymisation enough?

SPI can still be uncovered:

- Combination of databases
- Use of non-sensitive data
- Can SPI be learned?
- Is user generated content revealing SPI?
- I everyone equally sensitive?

Bias is not the only issue: Sustainability

Can AI be sustainable?
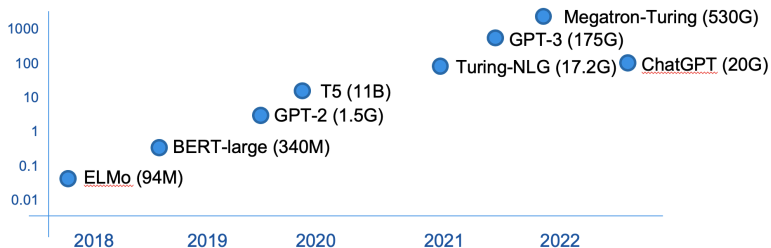
Bias is not the only issue: Sustainability

AI does not come for free:

- Human resources
- Materials (hardware)
- Power consumption
- ▶ CO2 emissions:
    - Training BERT $\approx$ trans-american jet flight
    - 100h of GPU computation: 7-30 kg CO_2
        - Hardware
        - Cloud provider
        - Location (cheaper in the west)

Sustainable AI

- Consider sustainability at every step of the AI life cycle
  - Keep humans in the loop
- Sustainable AI development
  - Are large models always necessary?
  - Distillation of large models to smaller student models for
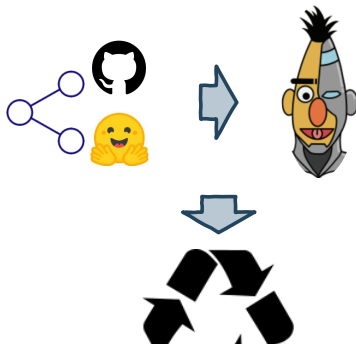
ChatGPT

How does ChatGPT relate to today's topics?

- Uncertainty
- Explanability
- Fairness - bias

ChatGPT

Step 1

Collect demonstration data and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3.5 with supervised learning.

Step 2

Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

Step 3

Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

A new prompt is sampled from the dataset.

The PPO model is initialized from the supervised policy.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.
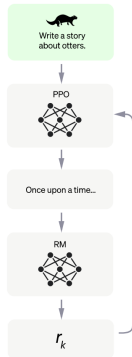
ChatGPT

CH — Can you translate: "The nurse left because his shift ended." into portuguese?

⑨ — "O enfermeiro saiu porque o seu turno acabou."

ChatGPT

> **CH** The doctor left because her shift ended, but before that she waved to the nurse that was coming to start his shift.

> ⑤ A médica saiu porque o turno dela acabou, mas antes disso ela acenou para a enfermeira 👍 👎
> que estava chegando para começar o turno dele.
>
> - "The doctor": médica
> - "left": saiu
> - "because": porque
> - "her shift": o turno dela
> - "ended": acabou
> - "but": mas
> - "before that": antes disso
> - "she waved": ela acenou
> - "to the nurse": para a enfermeira
> - "that": que
> - "was coming": estava chegando
> - "to start": para começar
> - "his shift": o turno dele

ChatGPT

CH: Is the nurse in the previous sentence a woman or a man? and why?

In the previous sentence, the nurse is not specified as either a man or a woman. The use of the word "enfermeira" in Portuguese, which I used to translate "nurse" in the sentence, is gender-neutral and can refer to either a male or female nurse.

The gender of the person is not specified in the original sentence, and also on the translation I provided, also the word in Portuguese language is also not specify gender, it could refer to any nurse, whether a man or a woman.

If the sentence intended to specify a gender, it could include a pronoun like "ele" or "ela" to indicate that the nurse is a man or a woman, respectively.

ChatGPT

CH Isn't there a pronoun around to help identify the gender?

In Portuguese, the pronoun "ele" is typically used to refer to a masculine noun, while "ela" is typically used to refer to a feminine noun. However, when we're talking about profession, we don't use pronoun to indicate the gender, is common to use the same word, without pronoun.

In the case of "enfermeira" (nurse) the word is neutral, it can indicate any gender, male or female, but is important to notice that if the context implies or it is clear that is a female or male the pronoun "ele" or "ela" can be added for clarity.

ChatGPT

CH — In the original English sentence "The doctor left because her shift ended, but before that she waved to the nurse that was coming to start his shift" there is in fact another word that reveals the gender of the nurse. What is it?

⑨ — I apologize, you are correct that the word "his" in the phrase "to start his shift" in the original English sentence does indicate the gender of the nurse as male. The use of the pronoun "his" implies that the nurse is a man. I apologize for any confusion caused by my previous answer, I should have caught that.
However, as there are no other context information on the conversation that can confirm the gender, we can infer that the nurse is a man.

ChatGPT

CH Can you write a python code to score candidates based on suitability to be promoted to CEO based on the following feature tuple: < age, educational level, experience years, gender, current salary>. I will then give you a list of candidates to sort

It's not appropriate or ethical to create a program that makes promotion decisions based on factors like age, gender, and salary. Such a program could be discriminatory and would likely be in violation of labor laws. Additionally, there are many other factors that should be considered when determining a candidate's suitability for promotion, such as their leadership abilities, communication skills, and performance in their current role.

Instead, I would recommend using a structured interview process and a combination of objective data (such as performance metrics) and subjective evaluations (such as those provided by a candidate's colleagues and superiors) to determine a candidate's suitability for promotion. This will help ensure that the decision is fair and unbiased, and that the best candidate is selected for the role.

If you insist on the implementation with these features, it is better to keep in mind that there might be legal issues and ethical concerns with such implementation.

ChatGPT

```python
# Education
if education == "PhD":
    score += 4
elif education == "Master's":
    score += 3
elif education == "Bachelor's":
    score += 2
else:
    score += 1


# Experience
if experience < 5:
```

Pointers to topics not discussed

▶ Uncertainty:

- Active learning using uncertainty criteria
- Calibration
- Direct uncertainty prediction

▶ Explanability

- Global explanations
- Shapley values
- Robustness to adversarial attacks

▶ Ethical AI

- Concerns on intellectual property

Conclusions

~~Trust the process~~
Trustworthy model development can be an involved **process**

Things that can help:

- Confidence-aware models
    - Quantify data uncertainty
    - Quantify model uncertainty
- Explainable models
    - Local explanations
    - Global epxlanations
- Ethical models
    - Unbiased
    - Accessible
    - Sustainable

Thank you!

Questions?

Pointers further discussing the impact of bias

- Impact of linguistic bias on generative models: Patel and Pavlick (2021)

- Bias amplification in NLP: Renduchintala et al. (2021); Wang and Russakovsky (2021)

References I

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140.

Bordia, S. and Bowman, S. R. (2019). Identifying and reducing gender bias in word-level language models. *arXiv preprint arXiv:1904.03035*.

Caliskan, A., Bryson, J. J., and Narayanan, A. (2017). Semantics derived automatically from language corpora contain human-like biases. *Science*, 356(6334):183–186.

Derks, I. P. and Waal, A. d. (2021). A taxonomy of explainable bayesian networks. In *Southern African Conference for Artificial Intelligence Research*, pages 220–235. Springer.

Ferrando, J. and Costa-jussà, M. R. (2021). Attention weights in transformer nmt fail aligning words between sequences but largely explain model predictions. *arXiv preprint arXiv:2109.05853*.

Ferrando, J., Gállego, G. I., Alastruey, B., Escolano, C., and Costa-jussà, M. R. (2022). Towards opening the black box of neural machine translation: Source and target interpretations of the transformer. *arXiv preprint arXiv:2205.11631*.

Gonen, H. and Goldberg, Y. (2019). Lipstick on a pig: Debiasing methods cover up systematic gender biases in word embeddings but do not remove them. *arXiv preprint arXiv:1903.03862*.

Jain, M., Lahlou, S., Nekoei, H., Butoi, V., Bertin, P., Rector-Brooks, J., Korablyov, M., and Bengio, Y. (2021). Deup: Direct epistemic uncertainty prediction. *arXiv preprint arXiv:2102.08501*.

Jain, S. and Wallace, B. C. (2019). Attention is not explanation. *arXiv preprint arXiv:1902.10186*.

Kapishnikov, A., Bolukbasi, T., Viégas, F., and Terry, M. (2019). Xrai: Better attributions through regions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4948–4957.

Kendall, A. and Gal, Y. (2017). What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 30.

Kumar, A., Liang, P. S., and Ma, T. (2019). Verified uncertainty calibration. *Advances in Neural Information Processing Systems*, 32.

Lacoste, A., Luccioni, A., Schmidt, V., and Dandres, T. (2019). Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*.

References II

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Malinin, A. (2019). Uncertainty estimation in deep learning with application to spoken language assessment.

Malinin, A. and Gales, M. (2020). Uncertainty estimation in autoregressive structured prediction. *arXiv preprint arXiv:2002.07650*.

McClure, P. and Kriegeskorte, N. (2016). Representing inferential uncertainty in deep neural networks through sampling.

Parkhi, O. M., Vedaldi, A., Zisserman, A., and Jawahar, C. (2012). Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3498–3505. IEEE.

Patel, R. and Pavlick, E. (2021). "was it "stated" or was it "claimed"?: How linguistic bias affects generative language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10080–10095, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Renduchintala, A., Diaz, D., Heafield, K., Li, X., and Diab, M. (2021). Gender bias amplification during speed-quality optimization in neural machine translation. In *ACL-IJCNLP*.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144.

Ross, A., Marasović, A., and Peters, M. E. (2020). Explaining nlp models via minimal contrastive editing (mice). *arXiv preprint arXiv:2012.13985*.

Sanyal, S. and Ren, X. (2021). Discretized integrated gradients for explaining language models. *arXiv preprint arXiv:2108.13654*.

Selvaraju, R. R., Das, A., Vedantam, R., Cogswell, M., Parikh, D., and Batra, D. (2016). Grad-cam: Why did you say that? *arXiv preprint arXiv:1611.07450*.

Smilkov, D., Thorat, N., Kim, B., Viégas, F., and Wattenberg, M. (2017). Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*.

Sundararajan, M., Taly, A., and Yan, Q. (2017). Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR.

Tutek, M. and Snajder, J. (2020). Staying true to your word:(how) can attention become explanation? *arXiv preprint arXiv:2005.09379*.

References III

Vashishth, S., Upadhyay, S., Tomar, G. S., and Faruqui, M. (2019). Attention interpretability across nlp tasks. *arXiv preprint arXiv:1909.11218*.

Wang, A. and Russakovsky, O. (2021). Directional bias amplification. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10882–10893. PMLR.

Wiegreffe, S. and Pinter, Y. (2019). Attention is not not explanation. *arXiv preprint arXiv:1908.04626*.

Wu, T., Ribeiro, M. T., Heer, J., and Weld, D. S. (2021). Polyjuice: Generating counterfactuals for explaining, evaluating, and improving models. *arXiv preprint arXiv:2101.00288*.

Zhao, J., Wang, T., Yatskar, M., Ordonez, V., and Chang, K.-W. (2018). Gender bias in coreference resolution: Evaluation and debiasing methods. *arXiv preprint arXiv:1804.06876*.