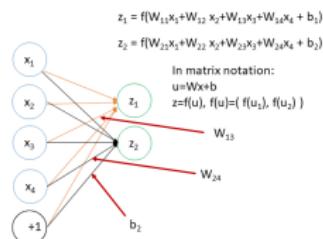


Chapter 1: Introduction to Machine Learning – Overview



Christian Heumann
(largely based on slides from Nina Poerner, Benjamin Roth, Marina Speranskaya)

CIS LMU München, Department of Statistics LMU München

November 2020

Course mode

- 60 - 70 minutes of video material per chapter (split into smaller videos of approx. 15 - 20 minutes)
- 30 minutes discussion of the videos during the regular lecture time
- 45 minutes (live) discussion of the exercise sheet's solution afterwards
- Examination: 15 minutes Oral Exam (Date: 17.02.2021)

References:

- Deep learning: ▶ Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep learning*
- Statistical learning:
 - ▶ Trevor Hastie, Robert Tibshirani and Jerome Friedman. *The Elements of Statistical Learning*
- Introduction to Machine Learning (online course):
 - ▶ Bernd Bischl, Fabian Scheipl, Heidi Seibold, Christoph Molnar and Daniel Schalk. *I2ML*

Why Machine Learning is Useful Today

- Humans get lost in the lake of ever-growing volumes and varieties of available data ▶ **Big Data**, thus automatic methods for analyzing the data or making decisions based on the data are necessary.
 - Unstructured data (e.g. text-heavy data) ▶ **Unstructured Data** often needs a lot of preprocessing before it can be further used. For humans this is usually an awkward task and is better delegated to a "machine".
 - Scientific laws (e.g. laws of classical mechanics) can most often not be applied to the data at hand, we can only learn something (usually called a *model*) from the observed *example data*.
 - *Accurate model building* can be a time-consuming and frustrating process, especially for complex data and has to be done again and again for different data. Machine learning promises to reduce the use of human resources, also leading to more accurate models.



Disadvantages of Machine Learning

- The resulting models are often black boxes. That makes it often difficult to evaluate the reliability and robustness of a model. Since there is a zoo of available ML algorithms, *model agnostic* methods are needed.
- It may be difficult to explain and interpret a model. For example, a credit scoring model should be transparent why a certain customer is getting a credit but another not.
- The model can only learn from the data which is available to *train* it. If the *training data* does not reflect or cover the *target population* well or contains *stereotypes* (e.g. gender stereotypes in text data), its application may lead to so-called *biased* results.
- The computational resources needed are often very high (e.g. GPUs are still expensive).
- Depending on the complexity of the problem, a high number of examples (statistically: a high sample size) is needed to build accurate models.

A Formal 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 ."
(Mitchell 1997)

- Learning: Attaining the ability to perform a task T .
- A set of examples ("experience") represents a more general task.
- Examples are (usually) described by *features*. The features are often represented by numeric vectors $x \in \mathbb{R}^m$ (sometimes also by matrices, tensors).

Data (Experience)

Typical rectangular data sets are defined as follows.

- Dataset: collection of examples
- Data is stored in a *design matrix*:

$$\mathbf{X} \in \mathbb{R}^{m \times n}$$

- m : number of examples
- n : number of features
- Examples: $X_{i,j}$ is the count of feature j (e.g. a stem form) in document i or the intensity of the j 'th pixel in image i

Characterization of the Learning Process

An often used characterization is *supervised* versus *unsupervised* learning.

- Unsupervised learning:
 - ▶ Model the data in \mathbf{X} , or find interesting properties of \mathbf{X} .
 - ▶ Example: Clustering (find groups of similar images/documents)
 - ▶ Training data: only \mathbf{X} .
- Supervised learning:
 - ▶ *Predict specific* additional properties from \mathbf{X}
 - ▶ E.g., sentiment classification: Predict sentiment (1–5) of amazon reviews
 - ▶ In the training data, additional *labels* (the *outputs*) are available for each example, i.e. an additional label vector $\mathbf{y} \in \mathbb{R}^m$ together with the *input* \mathbf{X} can be used for training the model.
 - ▶ The task is then to predict the labels of new data, where only the features \mathbf{X} are known.
 - ▶ The *performance* of this prediction is measured by a *loss function* and the task is usually solved by an optimization which minimizes the loss.

Supervised and Unsupervised Learning: data generation process (DGP)

- Unsupervised learning: Learn interesting properties, such as the probability distribution $p(\mathbf{x})$, i.e. estimate the distribution
- Supervised learning: learn mapping from \mathbf{x} to y , typically by estimating the conditional distribution $p(y|\mathbf{x})$. The supervised learning task is often formulated as the problem of estimating (or approximating) a function f , such that

$$y = f(\mathbf{x}) + \varepsilon ,$$

where ε is a random component (*error term*) describing the probabilistic nature of the task.

- Supervised learning in an unsupervised way (Bayes theorem):

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}, y)}{\sum_{y'} p(\mathbf{x}, y')} = \frac{p(\mathbf{x}, y)}{p(\mathbf{x})}$$

Note: if y is continuous, the sum (in the denominator) must be replaced by an integral.

Machine Learning Tasks

Types of Tasks:

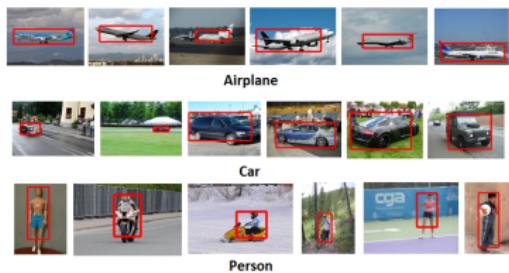
- Classification
- Regression
- Structured Prediction
- Anomaly Detection
- Synthesis and sampling
- Imputation of missing values
- Denoising
- Clustering
- Reinforcement learning
- ...

Task: Classification

- Which of k classes does an example belong to?

$$f : \mathbb{R}^n \rightarrow \{1 \dots k\}$$

- Typical example: Categorize image patches
 - ▶ Feature vector: color intensities for each pixel; derived features.
 - ▶ Output categories: Predefined set of labels



- Typical example: Spam Classification

- ▶ Feature vector: High-dimensional, sparse vector.
Each dimension indicates occurrence of a particular word, or other email-specific information.
- ▶ Output categories: “spam” vs. “ham”

Task: Classification

Identifying civilians killed by police with distantly supervised entity-event extraction

Katherine A. Keith, Abram Handler, Michael Pinkham,
Cara Maglozzi, Joshua McDuffie, and Brendan O'Connor
College of Information and Computer Sciences
University of Massachusetts Amherst

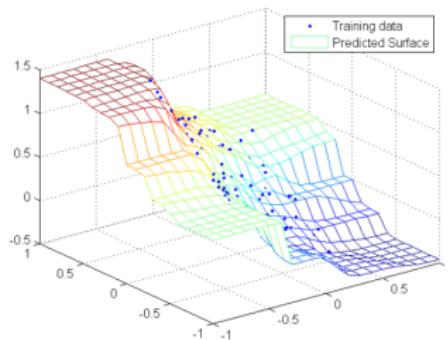
- EMNLP 2017: Given a **person name** in a sentence that contains **keywords** related to **police** ("officer", "police" ...) and to **killing** ("killed", "shot"), was the person a civilian killed by police?

Text	Person killed by police?
Alton Sterling was killed by police.	True
Officers shot and killed Philando Castile.	True
Officer Andrew Hanson was shot.	False
Police report Megan Short was fatally shot in apparent murder-suicide.	False

Task: Regression

- Predict a numerical value given some input.

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$



- Typical examples:
 - Predict the risk of an insurance customer.
 - Predict the value of a stock.

Task: Regression

Towards an Automatic Turing Test: Learning to Evaluate Dialogue Responses

Ryan Lowe^{♡*}

Michael Noseworthy^{♡*}

Iulian V. Serban[◊]

Nicolas A.-Gontier[♡]

Yoshua Bengio^{◊‡}

Joelle Pineau^{♡‡}

► arXiv link

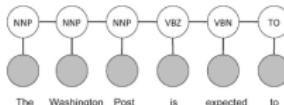
- ACL 2017: Given a response in a multi-turn dialogue, predict the value (on a scale from 1 to 5) how natural a response is.

Context	Reference response	Model responses	Human score	Adem score
photo to see my television debut go to - some. some on <url> - hehe <url> → it really was you? i thought ppl were recognizing someone who looked like you! were the oysters worth the wait?	yeah it was me . haha i 'd kinda forgotten about it it was filmed a while ago	1) i'm not sure. i just don't know what to do with it. 2) you heard the horsepower productions remix of lee scratch perry's 'exercising' off his 'mighty upsetter' album? 3) you wont chug a fuzzy peach navel 4) they were!	3 1 1 5	1.602 1.513 1.744 3.274
just beat call of duty!! → want a cookie? → yes!! → come get it	im in kenmore at the moment	1) i'm gonna get a new phone some moro 2) no way man. 3) wow i just got a free pizza coupon! get yours before theres no more! <url> 4) i'm going to go to the mall.	1 5 1 1	1.848 4.265 0.921 2.634
am i out of twitter jail yet? testing → yeah. i posted bail → thanks. i am a right chatter tweetbox on sundays. same happened last sunday lol	any news on meeting our user ? i go to the us on friday and i don 't want to miss anything arranged	1) i'm not sure if i'm going to be able to get it. 2) good to see another mac user in the leadership ranks 3) awww poor baby hope u get to feeling better soon. maybe some many work days at piedmont 4) did you tweet too much?	3 4 2 5	1.912 1.417 1.123 2.539

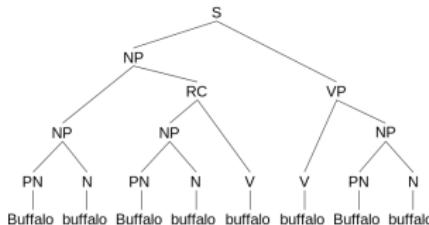


Task: Structured Prediction

- Predict a multi-valued output with special inter-dependencies and constraints.
- Typical examples:
 - ▶ Part-of-speech tagging



- ▶ Syntactic parsing



▶ Wikipedia

- ▶ Machine Translation
- Often involves search and problem-specific algorithms.

Task: Structured Prediction

Joint Extraction of Entities and Relations Based on a Novel Tagging Scheme

Suncong Zheng, Feng Wang, Hongyun Bao, Yuexing Hao, Peng Zhou, Bo Xu
Institute of Automation, Chinese Academy of Sciences, 100190, Beijing, P.R. China

- ACL 2017: jointly find all relations of interest in a sentence by tagging arguments and combining them. [Paper](#)

Input Sentence: The United States President Trump will visit the Apple Inc founded by Steven Paul Jobs

Tags: O B-CP-1 E-CP-1 O S-CP-2 O O O B-CF-1 E-CF-1 O O B-CF-2 I-CF-2 E-CF-2

Final Results:

{United States, Country-President, Trump}

{Apple Inc, Company-Founder, Steven Paul Jobs}

Task: Reinforcement Learning

- In **reinforcement learning**, the model (also called **agent**) needs to select a series of actions, but only observes the outcome (**reward**) at the end.
- The goal is to predict actions that will maximize the outcome.

Deal or No Deal? End-to-End Learning for Negotiation Dialogues

Mike Lewis¹, Denis Yarats¹, Yann N. Dauphin¹, Devi Parikh^{2,1} and Dhruv Batra^{2,1}
¹Facebook AI Research ²Georgia Institute of Technology

- EMNLP 2017: The computer negotiates with humans in natural language in order to maximize its points in a game. [▶ Paper](#)

Divide these objects between you
and another Turker. Try hard to get
as many points as you can!

Send a message now, or enter the agreed deal!



Items	Value	Number You Get
	8	<input type="text" value="1"/>
	1	<input type="text" value="1"/>
	0	<input type="text" value="0"/>

Mark Deal Agreed

Fellow Turker: I'd like all the balls

You: Ok, if I get everything else

Fellow Turker: If I get the book then you have a deal

You: No way - you can have one hat and all the balls

Fellow Turker: Ok deal

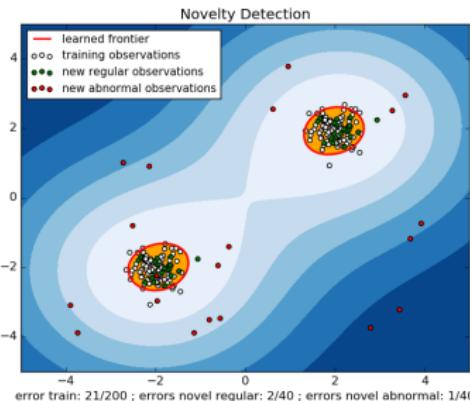
Type Message Here:

Message

Send

Task: Anomaly Detection

- Detect atypical items or events.
- Common approach: Estimate density and identify items that have low probability.



- Examples:
 - ▶ Quality assurance
 - ▶ Detection of criminal activity
- Often items categorized as outliers are sent to humans for further scrutiny.

Task: Anomaly Detection

Using Automated Metaphor Identification to Aid in Detection and Prediction of First-Episode Schizophrenia

E. Darío Gutiérrez¹ Philip R. Corlett² Cheryl M. Corcoran³ Guillermo A. Cecchi¹

- ACL 2017: Schizophrenia patients can be detected by their non-standard use of metaphors, and more extreme sentiment expressions. [▶ Paper](#)

Frequently Used Models and Algorithms in supervised ML

Supervised	Label	Task
linear regression	numeric	regression
logistic regression	binary	regression, classification
decision trees (e.g. C 4.5) ¹	class	classification
CART ²	class, numeric	classification, regression
random forest ³	class, numeric	classification, regression
Boosting, e.g. ▶ XGBoost	class, numeric	classification, regression
(artificial) neural networks	(see later)	(see later)

Note: Class variables are often called categorical variables in statistics.

¹Quinlan, J.R. (1992). C4.5 Programs for Machine Learning, San Mateo, CA: Morgan Kaufmann.

²L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone: CART: Classification and Regression Trees. Wadsworth: Belmont, CA, 1984.

³Breiman L., Random forests. In: Machine Learning, 2001, 45(1), Seiten 5–32, doi:10.1023/A:1010933404324

Some Algorithms for unsupervised ML

Unsupervised	Task
Clustering (k -means)	group similar objects
Clustering (partition around medoids (PAM))	group similar objects
PCA (principal components analysis)	dimension reduction
k -nearest neighbor	outlier detection
Isolation forest ⁴	anomaly detection
Autoencoders	representation, data compression

⁴Liu, F. T., Ting, K. M., and Zhou, Z.-H. 2008a. Isolation Forest. In ICDM '08: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining. IEEE Computer Society, 413–422.

Performance Measures

"A computer program is said to learn [...] with respect to some [...] **performance measure P** , if its performance [...] **as measured by P** , improves [...]"

- Quantitative measure of algorithm performance.
- Task-specific.

Discrete vs. Continuous Loss Functions

- **Discrete Loss Functions**
 - ▶ Accuracy (how many samples were correctly labeled?)
 - ▶ Error Rate ($1 - \text{accuracy}$)
 - ▶ Precision / Recall
 - ▶ Accuracy may be inappropriate for skewed label distributions, where relevant category is rare. Often used is the F1-Score (harmonic mean of precision and recall):

$$\text{F1-score} = \frac{2 \cdot \text{Prec} \cdot \text{Rec}}{\text{Prec} + \text{Rec}}$$

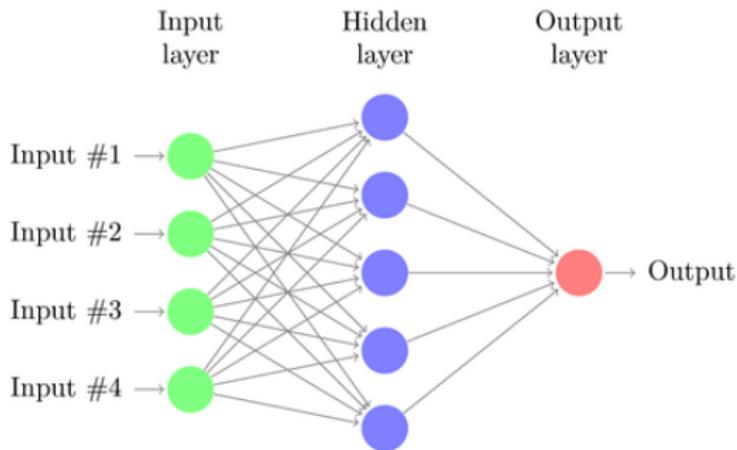
- ▶ F1-Score does not take into account the true negatives. Proposed alternative: Matthews correlation coefficient (MCC), $-1 \leq \text{MCC} \leq 1$.
- Discrete loss functions cannot indicate **how wrong** a wrong decision is.
- They are not differentiable (hard to optimize)
- Often algorithms are optimized using a continuous loss (e.g. hinge loss) and evaluated using another loss (e.g. F1-Score).

Examples for Continuous Loss Functions

- Squared error (regression): $(y - f(\mathbf{x}))^2$
- Hinge loss (classification):
 - ▶ $\max(0, 1 - f(\mathbf{x}) \cdot y)$
 - ▶ (assume that $y \in \{-1, 1\}$)
- ...
- These loss functions are differentiable. So we can use them for gradient descent (more on that later).

Deep Learning

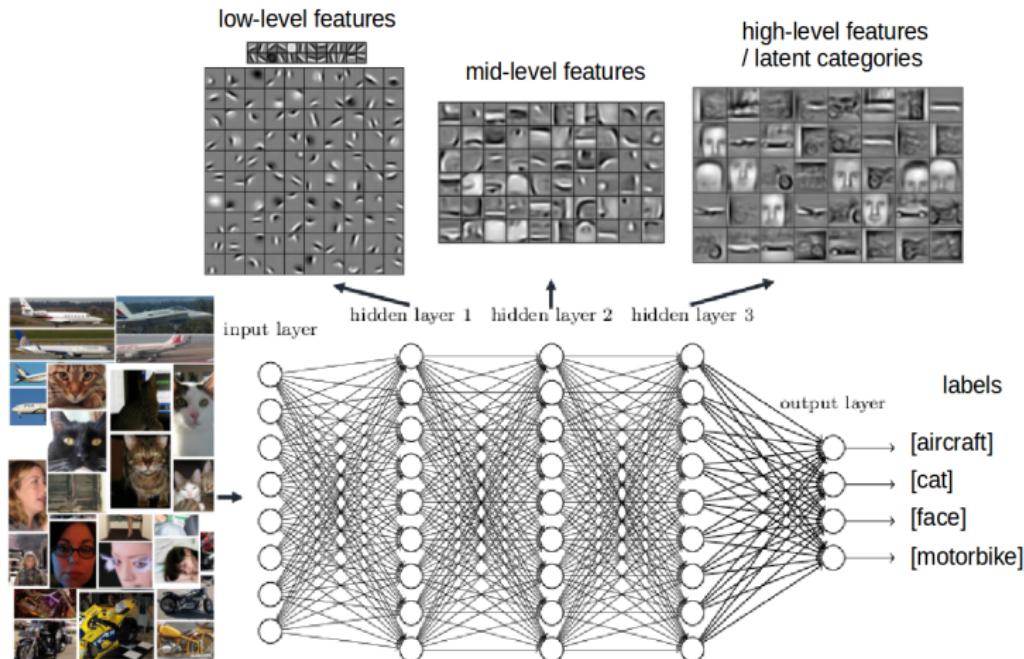
- Learn complex functions, that are (recursively) composed of simpler functions.
- Many parameters have to be estimated.



Deep Learning

- Main Advantage: Feature learning

- ▶ Models learn to capture *most essential* properties of data (according to some performance measure) as intermediate representations.
- ▶ No need to hand-craft feature extraction algorithms



Neural Networks

- First training methods for deep nonlinear NNs appeared in the 1960s (Ivakhnenko and others).
- Increasing interest in NN technology (again) since around 10 years ago ("Neural Network Renaissance"):
Orders of magnitude more data and faster computers now.
- Many successes:
 - ▶ Image recognition and captioning
 - ▶ Speech recognition
 - ▶ NLP and Machine translation
 - ▶ Game playing (AlphaGO)
 - ▶ ...

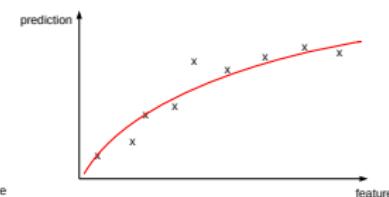
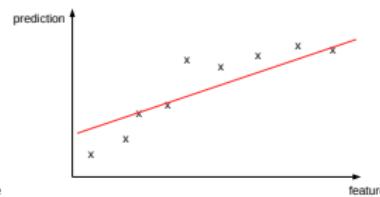
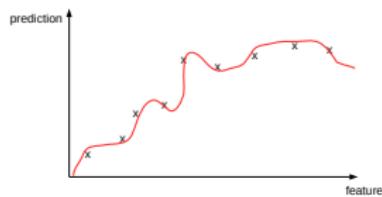
Machine Learning

- Deep Learning builds on general Machine Learning concepts

$$\operatorname{argmin}_{\theta \in \mathcal{H}} \sum_{i=1}^m \mathcal{L}(f(\mathbf{x}_i; \theta), y_i)$$

\mathcal{L} is called *loss function* or *cost function* (we will discuss it further in the next chapter).

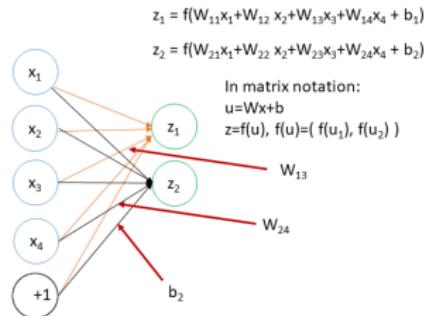
- Fitting data vs. generalizing from data



Summary

- Machine learning definition
 - ▶ Data
 - ▶ Task
 - ▶ Cost function
- Machine learning tasks
 - ▶ Classification
 - ▶ Regression
 - ▶ ...
- Deep Learning
 - ▶ many successes in recent years
 - ▶ feature learning instead of feature engineering
 - ▶ builds on general machine learning concepts

Chapter 2: Introduction to Machine Learning – ML Basics



Christian Heumann

November 2020

Overview

- Linear Algebra
- Probabilities in NLP
- Train-test-Splits
- Loss functions
- Performance measures
- Useful functions

Scalars, vectors, matrices and tensors I

- A scalar is a single number, e.g. a real number $x \in \mathbb{R}$, or a natural number $n \in \mathbb{N}$
- A vector is an array of (real) numbers. We define vectors as column vectors. A vector x of dimension n , $x \in \mathbb{R}^n$ is written as

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

x_i returns the element of the vector at row i .

Scalars, vectors, matrices and tensors II

- A matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$ is a two-dimensional array (of real numbers) of dimension $n \times p$. We use usually capital letters, i.e.

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{12} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix}$$

$\mathbf{X}_{i,j} = x_{ij}$ is the element in row i and column j . $\mathbf{X}_{:,j}$ returns the whole column j , $\mathbf{X}_{i,:}$ the i -th row of \mathbf{X} .

- A tensor is a general multi-dimensional array, e.g. a three-dimensional $n \times p \times m$ array $X \in \mathbb{R}^{n \times p \times m}$. The element in cell (i, j, k) is denoted by $x_{i,j,k}$.

Transpose

- The transpose of a matrix is denoted by T and exchanges rows and columns, e.g.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

That means that $\mathbf{X}_{i,j}^T = \mathbf{X}_{j,i} = x_{ji}$.

Addition

- Two matrices \mathbf{A} and \mathbf{B} can usually only be added if they have the same dimensions. The addition $\mathbf{C} = \mathbf{A} + \mathbf{B}$ is done element by element, i.e. $c_{ij} = a_{ij} + b_{ij}$. Example:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}$$

- If b is a scalar, $\mathbf{C} = \mathbf{A} + b$ means that b is added to each element of \mathbf{A} , i.e. $\mathbf{C}_{i,j} = c_{ij} = a_{ij} + b$.

Multiplication I

- Two matrices \mathbf{A} and \mathbf{B} can only be multiplied if the number of columns of \mathbf{A} is equal to the number of rows of \mathbf{B} , i.e. if \mathbf{A} is $n \times p$ and \mathbf{B} is $p \times m$. The multiplication $\mathbf{C} = \mathbf{AB}$ is done as

$$\mathbf{C}_{i,j} = c_{ij} = \sum_{k=1}^p a_{ik} b_{kj}$$

i.e. row i of \mathbf{A} is multiplied by column j of \mathbf{B} . \mathbf{C} is of dimension $n \times m$. Example:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 \end{bmatrix} = \begin{bmatrix} 74 & 80 & 86 & 92 \\ 173 & 188 & 203 & 218 \end{bmatrix}$$

- The dot product of two vectors \mathbf{x} and \mathbf{y} of same length n is defined as the scalar value $\mathbf{x}^T \mathbf{y} = \sum_{i=1}^n x_i y_i$. Therefore \mathbf{C}_{ij} is also the dot product of row i of \mathbf{A} and column j of \mathbf{B} .

Multiplication II

- If b is a scalar, $\mathbf{C} = b\mathbf{A}$ means that each element of \mathbf{A} is multiplied by b , i.e. $C_{ij} = c_{ij} = ba_{ij}$.
- If two matrices have the same dimensions, the element by element multiplication is called Hadamard product

$$\mathbf{C} = \mathbf{A} \odot \mathbf{B}$$

i.e. $c_{ij} = a_{ij}b_{ij}$.

- The Kronecker product of \mathbf{A} ($n \times p$) and \mathbf{B} returns a matrix \mathbf{C} where each (scalar) element of \mathbf{A} is multiplied by the matrix \mathbf{B} , i.e. $c_{ij} = a_{ij}\mathbf{B}$ so that

$$\mathbf{C} = \mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \dots & a_{1p}\mathbf{B} \\ \vdots & \vdots & & \vdots \\ a_{n1}\mathbf{B} & a_{n2}\mathbf{B} & \dots & a_{np}\mathbf{B} \end{bmatrix}$$

Calculation rules

- $C(A + B) = CA + CB$
- $C(BA) = (CB)A$
- $(AB)^T = B^T A^T$
- For n vectors x, y : $x^T y = (x^T y)^T = y^T x$
- Trace of a quadratic $n \times n$ matrix X :

$$\text{tr}X = \sum_{i=1}^n x_{ii}$$

What is a trace?

There are also useful rules, e.g. $\text{tr}(A + B) = \text{tr}A + \text{tr}B$ or $\text{tr}(AB) = \text{tr}(BA)$.

Special matrices

?

- Inverse of a quadratic $n \times n$ matrix \mathbf{A} : \mathbf{A}^{-1} , such that $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}_n$, where \mathbf{I}_n is the (quadratic) n -dimensional identity matrix with 1's on the diagonal and 0's elsewhere.
- Inverse exists if all rows (columns) are linearly independent.
- If \mathbf{A} is symmetric and positive definite, the inverse exists, all eigenvalues are positive and real, and the following decomposition exists:

$$\mathbf{A} = \mathbf{\Gamma} \mathbf{\Lambda} \mathbf{\Gamma}^T$$

where the columns of $\mathbf{\Gamma}$ are the orthogonal eigenvectors of \mathbf{A} and $\mathbf{\Lambda}$ is a diagonal matrix with the eigenvalues on the diagonal. All matrices are real valued in that case.

Further, $\mathbf{A}^{-1} = \mathbf{\Gamma} \mathbf{\Lambda}^{-1} \mathbf{\Gamma}^T$.

Norms

- L_p norm of an n vector \mathbf{x} :

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

Special cases: L_1 norm, L_2 norm (Euclidean norm)

- Max norm: $\|\mathbf{x}\|_\infty = \max_i |x_i|$
- Frobenius norm of a $n \times p$ matrix:

$$\|\mathbf{X}\|_F = \sqrt{\sum_{i=1}^n \sum_{j=1}^p x_{ij}^2}$$

- Applications: L_1 distance, L_2 distance of two vectors, Frobenius distance of two matrices
- $\mathbf{x}^T \mathbf{x} = (\|\mathbf{x}\|_2)^2 = \|\mathbf{x}\|_2^2$
- $\mathbf{x}^T \mathbf{y} = \|\mathbf{x}\|_2 \|\mathbf{y}\|_2 \cos(\phi)$, where ϕ is the angle between \mathbf{x} and \mathbf{y} .

Derivations I

Formulas for derivations of expressions wrt a vector (or a matrix) are useful tools for optimizations. The gradient notation $\nabla_{\mathbf{w}}$ is used for the derivative of a scalar wrt a vector \mathbf{w} .

- Derivative of a scalar product two vectors wrt one vector (we always assume that the dimensions are compatible):

$$\frac{\partial}{\partial \mathbf{w}} \mathbf{w}^T \mathbf{x} = \nabla_{\mathbf{w}} \mathbf{w}^T \mathbf{x} = \mathbf{x}$$

This can be shown as follows:

$$\frac{\partial}{\partial \mathbf{w}} \mathbf{w}^T \mathbf{x} = \begin{bmatrix} \frac{\partial}{\partial w_1} (w_1 x_1 + w_2 x_2 + \dots + w_n x_n) \\ \frac{\partial}{\partial w_2} (w_1 x_1 + w_2 x_2 + \dots + w_n x_n) \\ \vdots \\ \frac{\partial}{\partial w_n} (w_1 x_1 + w_2 x_2 + \dots + w_n x_n) \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \mathbf{x}$$

Derivations II

- Let \mathbf{A} symmetric. Then

$$\nabla_{\mathbf{w}} \mathbf{w}^T \mathbf{A} \mathbf{w} + \mathbf{b}^T \mathbf{w} = 2\mathbf{A}\mathbf{w} + \mathbf{b}$$

This is later applied in the linear (regression) model.

Derivations III

With the previous rule, it suffices to show that $\nabla_{\mathbf{w}} \mathbf{w}^T \mathbf{A} \mathbf{w} = 2\mathbf{A} \mathbf{w}$.

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}} \mathbf{w}^T \mathbf{A} \mathbf{w} &= \left[\begin{array}{c} \frac{\partial}{\partial w_1} (\sum_{i=1} w_i^2 a_{ii} + 2 \sum_{i < j} w_i w_j a_{ij}) \\ \frac{\partial}{\partial w_2} (\sum_{i=1} w_i^2 a_{ii} + 2 \sum_{i < j} w_i w_j a_{ij}) \\ \vdots \\ \frac{\partial}{\partial w_n} (\sum_{i=1} w_i^2 a_{ii} + 2 \sum_{i < j} w_i w_j a_{ij}) \end{array} \right] \\ &= \left[\begin{array}{c} 2w_1 a_{11} + 2 \sum_{j \neq 1} w_j a_{1j} \\ 2w_2 a_{22} + 2 \sum_{j \neq 2} w_j a_{2j} \\ \vdots \\ 2w_n a_{nn} + 2 \sum_{j \neq n} w_j a_{nj} \end{array} \right] \quad \text{unclar} \\ &= \left[\begin{array}{c} 2 \sum_{j=1}^n w_j a_{1j} \\ 2 \sum_{j=1}^n w_j a_{2j} \\ \vdots \\ 2 \sum_{j=1}^n w_j a_{nj} \end{array} \right] = 2\mathbf{A} \mathbf{w}\end{aligned}$$

Derivations IV

- Further example: Let $f : \mathbb{R} \rightarrow \mathbb{R}$ a scalar (non-linear) function.

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

and $\mathbf{z} = f(\mathbf{Wx} + \mathbf{b})$, i.e. f is applied elementwise:

$$u_1 = W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + W_{14}x_4 + b_1 = \mathbf{W}_{1,:}\mathbf{x}$$

$$u_2 = W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + W_{24}x_4 + b_2 = \mathbf{W}_{2,:}\mathbf{x}$$

$$z_1 = f(u_1) = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + W_{14}x_4 + b_1) = f(\mathbf{W}_{1,:}\mathbf{x})$$

$$z_2 = f(u_2) = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + W_{24}x_4 + b_2) = f(\mathbf{W}_{2,:}\mathbf{x})$$

Derivations V

Then, using the chain rule (note, that z_1 only depends on u_1 , z_2 only on u_2):

$$\begin{aligned}\frac{\partial z_i}{\partial W_{ij}} &= \frac{\partial z_i}{\partial u_i} \frac{\partial u_i}{\partial W_{ij}} && \text{chain rule} \\ &= \frac{\partial f(u_i)}{\partial u_i} x_j = f'(u_i) x_j\end{aligned}$$

Further,

$$\begin{aligned}\frac{\partial z_i}{\partial b_i} &= \frac{\partial z_i}{\partial u_i} \frac{\partial u_i}{\partial b_i} \\ &= \frac{\partial f(u_i)}{\partial u_i} = f'(u_i)\end{aligned}$$

Derivations VI

Assume, that we have an additional (non-linear) function g :

$$\begin{aligned}t &= h_1 z_1 + h_2 z_2 \\v &= g(t)\end{aligned}$$

also chain rule

then ($j = 1, 2, 3, 4$)

$$\begin{aligned}\frac{\partial v}{\partial x_j} &= \frac{\partial v}{\partial z_1} \frac{\partial z_1}{\partial x_j} + \frac{\partial v}{\partial z_2} \frac{\partial z_2}{\partial x_j} \\&= \frac{\partial v}{\partial z_1} \frac{\partial z_1}{\partial u_1} \frac{\partial u_1}{\partial x_j} + \frac{\partial v}{\partial z_2} \frac{\partial z_2}{\partial u_2} \frac{\partial u_2}{\partial x_j} \\&= [g'(t)h_1]f'(u_1)W_{1j} + [g'(t)h_2]f'(u_2)W_{2j}\end{aligned}$$

Probabilities in NLP

- In NLP, probabilities are put on the words of a corpus or sequences of words (e.g. a sentence)
- These distributions are discrete because each word or each sequence of words is treated as a possible outcome
- Language models are based on probabilities and maximization of the log-likelihood (or minimization of the negative log-likelihood), which is also based on (log-)probabilities.

Discrete random variables

- In NLP, the words may be treated as discrete random variables W .
- The number of words $|V|$ in a corpus is finite, but may be huge (in the millions)
- A realization of W is a certain word w , e.g. *house*
- The words in a sentence occur not independently from each other, i.e. the words are correlated. Therefore, we have to consider sequences of words as multivariate random variables (W_1, W_2, \dots, W_n) .
- A realization is a certain sequence, e.g.
 $(W_1 = \text{The}, W_2 = \text{dog}, W_3 = \text{is}, W_4 = \text{barking})$.

Probability distributions in NLP

- Joint distribution of a sequence of words w_1, w_2, \dots, w_n , n may be any number:

$$P(W_1 = w_1, W_2 = w_2, \dots, W_n = w_n) = P(w_1, w_2, \dots, w_n)$$

The right hand side of the equation is a short form of the left hand side. Example: $P(\text{The}, \text{dog}, \text{is}, \text{barking})$.

- Special choices of n :

- ▶ $n = 1$: Unigrams
- ▶ $n = 2$: Bigrams

Generally: n -grams

- Conditional distributions, e.g. the probability of the next word given a sequence of previous words:

$$P(w_{i+1} | w_i, w_{i-1}, \dots, w_1)$$

The sequence of previous words $(w_i, w_{i-1}, \dots, w_1)$ is also called context of w_{i+1} .

Factorization of the joint distribution

- Sometimes called chain rule for conditional probabilities
- The joint distribution can be written as a product of conditional distributions

$$\begin{aligned} P(w_1, w_2, \dots, w_n) &= P(w_1)P(w_2|w_1)P(w_3|w_2, w_1)\cdots P(w_n|w_{n-1}, \dots, w_1) \\ &= P(w_1) \prod_{i=2}^n P(w_i|w_{i-1}, \dots, w_1) \end{aligned}$$

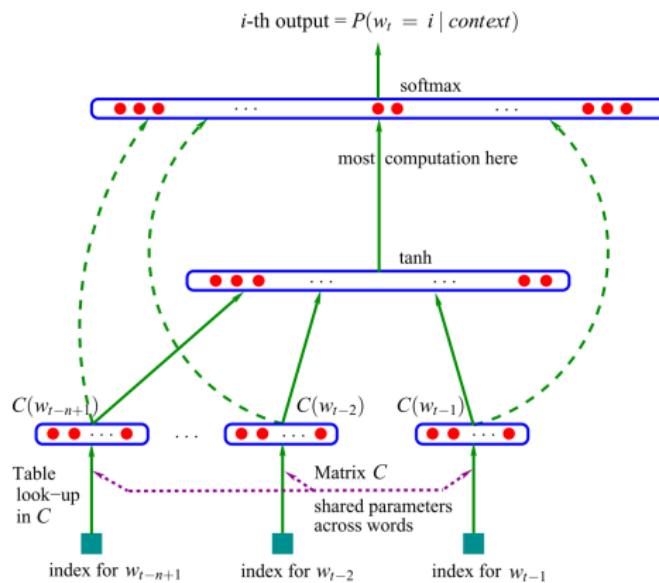
Markov assumptions

- The context of a word cannot be arbitrary long
- A fixed (or maximum) size for the context is usually assumed
- Under the Markov assumptions, the joint distribution can be factorized by conditional probabilities with a fixed context size.
- Example: Assume that $P(w_i|w_{i-1}, \dots, w_1) = P(w_i|w_{i-1})$ (i.e., we only consider bigrams). Then

$$P(w_1, w_2, \dots, w_n) = P(w_1) \prod_{i=2}^n P(w_i|w_{i-1})$$

Bengio's et al. neural network language model (NNLM)

- In 2003, Bengio et al. introduced the NNLM which uses the introduced factorization with the Markov property



More on random numbers and probabilities

- Discrete random variables also occur in other contexts than NLP.
Examples:
 - ▶ Binary random variables (Bernoulli experiments) are a model for experiments with two possible outcomes (0 or 1), e.g. coin tossing.
 - ▶ Poisson random numbers are often for counts, e.g. the number of car accidents during a certain interval (month, year) in a city.
- Random numbers can also be continuous, e.g. Gaussian random numbers, i.e., X is Gaussian, if $X \sim N(\mu, \sigma^2)$ where μ and σ^2 are two parameters which have usually to be estimated from the data. The distribution of random numbers is often given in terms of a *probability density* $f(x)$ which is positive and integrates to 1. For a Gaussian, the density is

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(x - \mu)^2\right\} .$$

Expectation and variance

- If g is a function of a random variable X with density function $p(x)$, the expectation or expected value of g is the mean value or average over all possible values of X :

$$E(g(X)) = \int g(x)p(x)dx .$$

If f is discrete, the integral is replaced by a sum.

- The variance of g is then

$$\text{Var}[g(X)] = E\{[g(X) - E(g(X))]^2\} = \int [g(x) - E(g(X))]^2 p(x)dx .$$

Theorem of Bayes

- The theorem of Bayes can be stated for probabilities and densities
 - ▶ For probabilities, it solves the following problem: if it is known that an event B has occurred (e.g. that the thrown number on top of a dice is odd) than the probability of another event A (e.g. a 3 has been thrown) can be calculated. This is usually denoted as

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

E.g.: $P(\text{odd number} = \frac{1}{2})$ and

$$P(3|\text{odd number}) = \frac{\text{odd number and } 3}{P(\text{odd number})} = \frac{\frac{1}{6}}{\frac{1}{2}} = \frac{1}{3}$$

- ▶ For densities: let $p(x, y)$ the joint density of two random variables, $p(x)$ the marginal and $p(y|x)$ the conditional density. Then

$$p(y|x) = \frac{p(x, y)}{p(x)} .$$

This holds also, if x is an n -dimensional random vector .

Train-/Dev-/Test-Splits I

- To train a machine learning model, several strategies are possible. One strategy is to split the whole available training data into two or three parts. If the algorithm requires hyperparameter tuning, a split in three parts is necessary:



The *training set* is used to generate the ML model (given that certain hyperparameters are fixed at some values), the *development set* is used to compare different hyperparameter values and to select (in best case) the optimal ones or at least those who give the best evaluation on the dev set. The test set is finally used to measure the performance of the ML model with the optimal hyperparameters.

- Another strategy is to use cross validation.

Loss function I

- A loss function \mathcal{L} is a function into the positive real numbers (including 0).
- Let y the actual value and \hat{y} a prediction then the loss function is given by $\mathcal{L} = \mathcal{L}(y, \hat{y})$.
- Loss functions can be discrete or continuous
- For optimization purposes, continuous loss functions are preferable
- Supervised learning tasks try to minimize the *expected* loss $EL = E_{(x,y)}(\mathcal{L}(Y, \hat{Y}))$. Usually, the prediction \hat{Y} is estimated as a prediction function $f(x)$, depending on the features x ,

$$\hat{Y} = f(x) .$$

The expected loss is then $EL = E_{(x,y)}(\mathcal{L}(Y, f(x)))$.

Loss function II

- If one chooses a quadratic loss function $\mathcal{L}(Y, f(\mathbf{X})) = [Y - f(\mathbf{X})]^2$, the expected loss (or prediction error) is

$$E(\mathcal{L}(Y, f(\mathbf{X}))) = \int [y - f(\mathbf{x})]^2 P(dy, d\mathbf{x}).$$

Factorizing $P(Y, \mathbf{X}) = P(Y|\mathbf{X})P(\mathbf{X})$, this can be written as

$$E(\mathcal{L}(Y, f(\mathbf{X}))) = E_{\mathbf{X}} E_{Y|\mathbf{X}}([Y - f(\mathbf{X})]^2 | \mathbf{X})$$

A pointwise minimization leads to

$$f(\mathbf{x}) = \operatorname{argmin}_c E_{Y|\mathbf{X}}([Y - c]^2 | \mathbf{X} = \mathbf{x})$$

This is called the regression function (conditional expectation)
 $f(\mathbf{x}) = E(Y|\mathbf{X} = \mathbf{x})$.

Loss function III

- Discrete case: let C the observed class and $\hat{C}(\mathbf{X})$ the function which assigns each \mathbf{X} a certain class C from \mathcal{C} . Let the classes denoted as $1, 2, \dots, K$. A possible loss function is the 0-1-loss. A wrong assignment is punished by 1:

$$\mathcal{L}(C, C(\mathbf{X})) = \begin{cases} 0 & \text{if } C = \hat{C}(\mathbf{X}) \\ 1 & \text{if } C \neq \hat{C}(\mathbf{X}) \end{cases}$$

It holds $EL = E_{\mathbf{X}}[\sum_{k=1}^K \mathcal{L}(k, \hat{C}(\mathbf{X}))P(k|\mathbf{X})]$. A pointwise minimization leads to

$$\hat{C}(\mathbf{x}) = \operatorname{argmin}_{c \in \mathcal{C}} \sum_{k=1}^K \mathcal{L}(k, c)P(k|\mathbf{X} = \mathbf{x}) = \operatorname{argmin}_{c \in \mathcal{C}} (1 - P(c|\mathbf{X} = \mathbf{x}))$$

With the 0-1-loss one gets

$$\hat{C}(\mathbf{x}) = c_0, \quad \text{if } P(c_0|\mathbf{X} = \mathbf{x}) = \max_{c \in \mathcal{C}} P(c|\mathbf{X} = \mathbf{x}) .$$

Loss function IV

This is the so-called Bayes-classifier which assigns the class which has the highest conditional probability.

- Note that the expected loss is estimated from test data using the test samples (empirical loss). Therefore the distributions are not necessary to do the calculation of the empirical loss. We simply calculate the average over all validation or test samples (which is a consistent estimate for the expected loss):

$$\widehat{\mathcal{L}}(Y, f(\mathbf{X})) = \frac{1}{m'} \sum_{j=1}^{m'} \mathcal{L}(Y_j, \hat{f}(\mathbf{x}_j))$$

where $\hat{Y}_j = \hat{f}(\mathbf{x}_j)$ is the prediction of the j th sample and m' is the number of samples.

- The negative log-likelihood is often used as loss function (we will see an example later).

Performance measures (supervised learning) I

- While the loss functions are used for optimization purposes, performance measures are usually used to evaluate a machine learning model.
- Many supervised tasks are classification tasks, often with two classes (binary task), e.g. sentiment analysis (positive or negative sentiment of a text), but in principle a finite number of classes is possible. In this case, many measures have been proposed which are based on the *confusion matrix*. Example: consider a classifier for images showing either a donkey or a horse:

		observed class	
		donkey	horse
predicted class	donkey	4 (<i>TP</i>)	1 (<i>FP</i>)
	horse	2 (<i>FN</i>)	3 (<i>TN</i>)

The data behind this table can be summarized as

$y = (1, 1, 1, 1, 1, 0, 0, 0, 0)$, where 1 is used for representing a donkey (a "positive" case) and 0 is used for a horse (a "negative")

Performance measures (supervised learning) II

case). Then we call the prediction of a donkey as donkey as *true positive (TP)*, the prediction of a horse as a horse as *true negative (TN)*, the prediction of a donkey as a horse as *false negative (FN)* and the prediction of a horse as a donkey as *false positive (FP)*. All performance measures build on these four numbers:

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$F1 = 2 \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

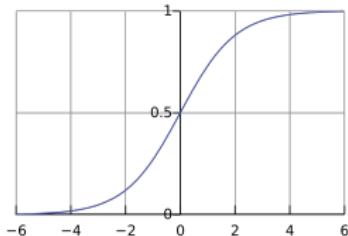
$$MC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(FP + TN)(FN + TN)}}$$

MC: Matthews correlation coefficient

Performance measures (supervised learning) III

- In multiclass problems, a generalization can be obtained by two approaches. One approach considers all correct predictions versus all wrong predictions (e.g. Micro F1 score). The second focuses on one class and pools all other classes together. Then, binary measures can be calculated for each class separately and then summarized into one measure (e.g. Macro F1 score).

Sigmoid function



- The sigmoid function (also used in logistic regression) is defined as

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

Note that $0 \leq \sigma(z) \leq 1$ and $1 - \sigma(z) = \frac{1}{1 + e^z}$.

- The derivative is

$$\begin{aligned}\frac{d}{dz} \sigma(z) &= \frac{(1 + e^z)e^z - e^z e^z}{(1 + e^z)^2} = \frac{e^z}{(1 + e^z)^2} \\&= \frac{e^z}{1 + e^z} \frac{1}{1 + e^z} \\&= \sigma(z)(1 - \sigma(z))\end{aligned}$$

Softmax function

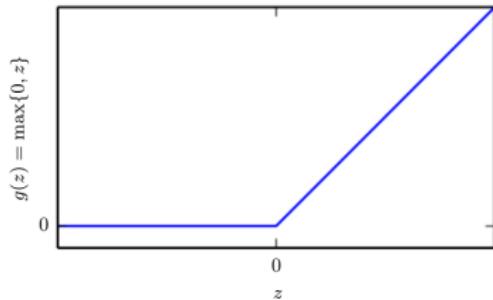
- Let $\mathbf{z} \in \mathbb{R}^K$ a real vector $\mathbf{z} = (z_1, \dots, z_K)$. Then the value of the softmax function is also a vector of dimension K with elements

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}, j = 1, \dots, K$$

- It holds $0 \leq \sigma(\mathbf{z})_j \leq 1$ and $\sum_{k=1}^K \sigma(\mathbf{z})_j = 1$.
- This normalization property makes it useful for modeling discrete probability distributions
- It is also used in the multinomial logistic regression model.
- The derivative is

$$\frac{\partial}{\partial z_k} \sigma(\mathbf{z})_j = \text{exercise!}$$

Rectified linear unit (ReLU)

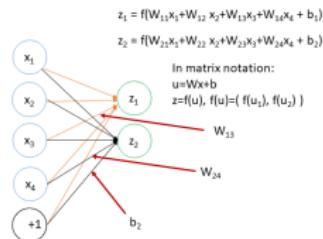


- The ReLu is defined as

$$g(z) = \max(0, z) = z^+$$

- Piecewise linear function
- Derivative is either 0 ($z < 0$) or 1 ($z > 0$) or undefined ($z = 0$).
- Many variants exist

Chapter 3: Introduction to Machine Learning – Linear and Logistic Regression



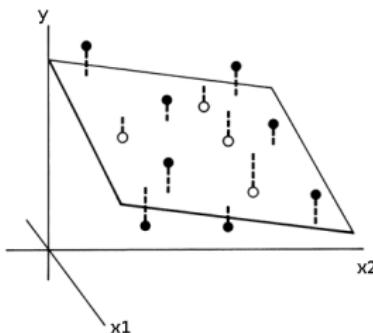
Christian Heumann

(largely based on slides from Nina Poerner, Benjamin Roth, Marina Speranskaya)

CIS LMU München, Department of Statistics LMU München

November 2020

Linear Regression



- In linear regression, the prediction \hat{y} of a label y is a linear function:

$$\hat{y} = \mathbf{w}^T \mathbf{x} = \sum_{j=1}^n w_j x_j;$$

- Note, that in this notation, the parameter vector $\mathbf{w} \in \mathbb{R}^n$ stands already for the *estimated* parameter vector.
- Weight w_j decides if value of feature x_j increases or decreases prediction \hat{y} .

Matrix notation I

- m samples
- Prediction for one sample:

$$\hat{y}_i = \mathbf{w}^T \mathbf{x}_i$$

- Error for one sample (residual):

$$e_i = (y_i - \hat{y}_i)$$

- Squared error for one sample:

$$e_i^2 = (y_i - \hat{y}_i)^2 = (\hat{y}_i - y_i)^2$$

Matrix notation II

- Matrix notation:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \quad \hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_m \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_m^T \end{bmatrix} \quad \hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$$

- \mathbf{X} in detail: typically, first column contains all **1** for the intercept (bias, shift) term.

$$\mathbf{X} = \begin{bmatrix} 1 & x_{12} & x_{13} & \dots & x_{1n} \\ 1 & x_{22} & x_{23} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m2} & x_{m3} & \dots & x_{mn} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

- Estimate parameters using $\mathbf{X}^{(train)}$ and $\mathbf{y}^{(train)}$.
- Make high-level decisions (which features...) using $\mathbf{X}^{(dev)}$ and $\mathbf{y}^{(dev)}$.
- Evaluate resulting model using $\mathbf{X}^{(test)}$ and $\mathbf{y}^{(test)}$.

Simple Example: Housing Prices

- Predict property prices (in 1K Euros) from just one feature: Square feet of property.

$$\mathbf{X} = \begin{bmatrix} 1 & 450 \\ 1 & 900 \\ 1 & 1350 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 730 \\ 1300 \\ 1700 \end{bmatrix}$$

- Prediction is:

$$\hat{\mathbf{y}} = \begin{bmatrix} w_1 + 450w_2 \\ w_1 + 900w_2 \\ w_1 + 1350w_2 \end{bmatrix} = \begin{bmatrix} 1 & 450 \\ 1 & 900 \\ 1 & 1350 \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \mathbf{X}\mathbf{w}$$

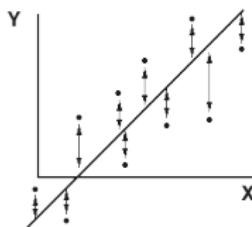
- w_1 will contain costs incurred in any property acquisition
- w_2 will contain remaining average price per square feet.
- Optimal parameters are for the above case:

$$\mathbf{w} = \begin{bmatrix} 273.3 \\ 1.08 \end{bmatrix} \quad \hat{\mathbf{y}} = \begin{bmatrix} 759.1 \\ 1245.1 \\ 1731.1 \end{bmatrix}$$

Linear Regression: Mean Squared Error

- Mean squared error of training (or test) data set is the sum of squared differences between the predictions and labels of all m instances.

$$MSE := \frac{1}{m} \sum_{i=1}^m e_i^2 = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2$$



- In matrix notation:

$$MSE := \frac{1}{m} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2$$

$$= \frac{1}{m} \|\mathbf{X} \mathbf{w} - \mathbf{y}\|_2^2$$

Learning: Improving on MSE

- Gradient of a function f : Vector whose components are the n partial derivatives of f wrt to the parameters, here \mathbf{w} .

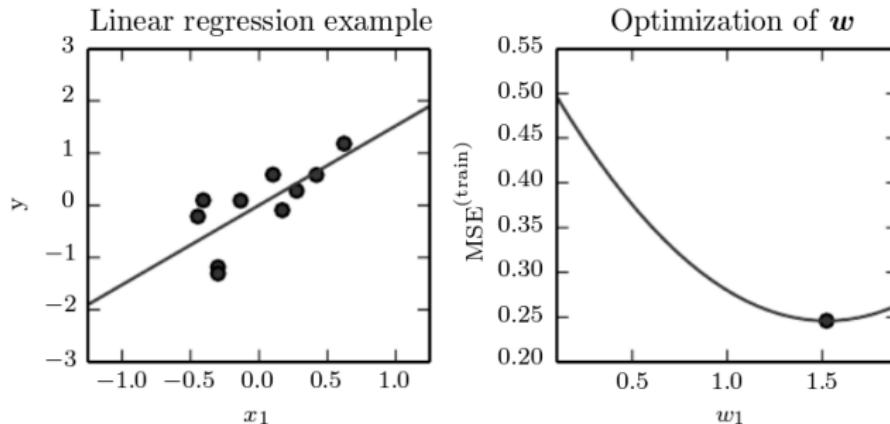
$$\nabla_{\mathbf{w}} f(\mathbf{w}) = \begin{bmatrix} \frac{\partial f(\mathbf{w})}{\partial w_1} \\ \frac{\partial f(\mathbf{w})}{\partial w_2} \\ \vdots \\ \frac{\partial f(\mathbf{w})}{\partial w_n} \end{bmatrix}$$

- View MSE as a function $f(\mathbf{w})$ of \mathbf{w}
- Minimum is where gradient is $\mathbf{0}$:

$$\nabla_{\mathbf{w}} MSE \stackrel{!}{=} \mathbf{0}$$

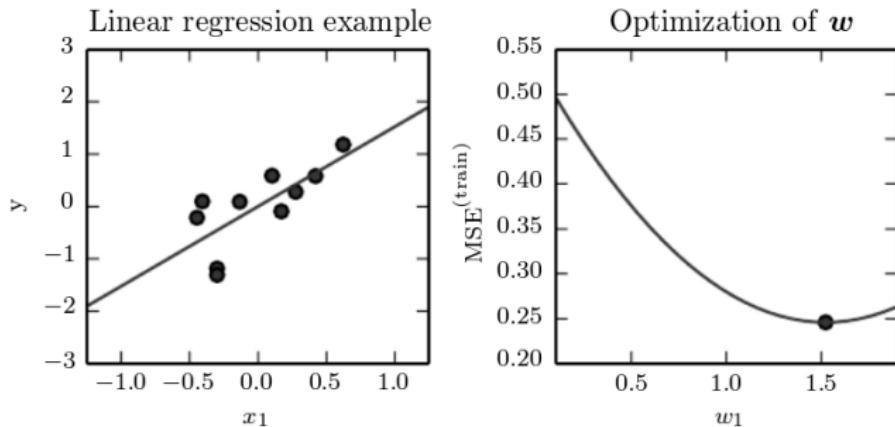
Learning: Improving on MSE

- View MSE as a function of w



- Minimum is where gradient $\nabla_w MSE = \mathbf{0}$.
- Why minimum and not maximum or saddle point?
 - Because it is a quadratic function...
 - Check convexity for 1 dimensional function: Second derivative > 0 .
 - Check for vector valued function: Hessian is positive-semidefinite.

Second Derivative Test



Second derivative of Mean Squared Error for Linear model with only one feature:

$$\frac{d^2}{dw^2} \sum_{i=1}^m (x^{(i)}w - y^{(i)})^2 = \frac{d^2}{dw^2} \sum_{i=1}^m (x^{(i)2}w^2 - 2x^{(i)}w + y^{(i)2}) = 2 \sum_{i=1}^m x^{(i)2} > 0$$

Solving for w

- We now know that minimum is where gradient is $\mathbf{0}$.

$$\nabla_w MSE = \mathbf{0}$$

$$\Rightarrow \nabla_w \frac{1}{m} \|\mathbf{X}w - \mathbf{y}\|_2^2 = \mathbf{0}$$

- Solve for w :

$$w = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

(Normal Equation)

- The inverse $(\mathbf{X}^T \mathbf{X})^{-1}$ exists, if X has full column rank (i.e. rank n).

Deriving the Normal Equation

- Function to minimize:

$$\begin{aligned} & \| \mathbf{Xw} - \mathbf{y} \|_2^2 \\ &= (\mathbf{Xw} - \mathbf{y})^T (\mathbf{Xw} - \mathbf{y}) \\ &= \mathbf{w}^T \mathbf{X}^T \mathbf{Xw} - \mathbf{w}^T \mathbf{X}^T \mathbf{y} - \mathbf{y}^T \mathbf{Xw} + \mathbf{y}^T \mathbf{y} \\ &= \mathbf{w}^T \mathbf{X}^T \mathbf{Xw} - 2\mathbf{w}^T \mathbf{X}^T \mathbf{y} + \mathbf{y}^T \mathbf{y} \end{aligned}$$

- Take the gradient¹ w.r.t. \mathbf{w} and set equal to $\mathbf{0}$:

$$\begin{aligned} & 2\mathbf{X}^T \mathbf{Xw} - 2\mathbf{X}^T \mathbf{y} = \mathbf{0} \\ & \Rightarrow \mathbf{X}^T \mathbf{Xw} = \mathbf{X}^T \mathbf{y} \\ & \Rightarrow (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Xw} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \end{aligned}$$

¹[Matrix Cookbook. Petersen and Pedersen, 2012]:

$$\nabla_{\mathbf{w}} \mathbf{w}^T \mathbf{a} = \mathbf{a}$$

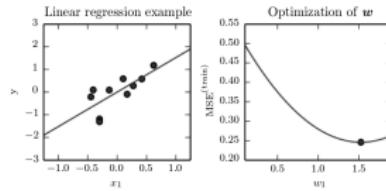
$$\nabla_{\mathbf{w}} \mathbf{w}^T \mathbf{Bw} = 2\mathbf{Bw} \text{ for symmetric } \mathbf{B}$$

Practical Remarks

What if $\mathbf{X}^T \mathbf{X}$ does not have an inverse?

- This can happen if there are infinitely many solutions:
 - ▶ one feature is the exact multiple of another
 - ▶ there are more features than training examples
 - ▶ \Rightarrow a Moore-Penrose pseudoinverse picks solution with smallest Euclidean norm.
 - ▶ \Rightarrow adding a regularization term makes the system of equations non-singular (uniquely solvable).
 - ★ Normal Equation becomes: $\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$
 - ★ Also called *Ridge Regression*.
- In general: **Avoid computing the matrix inverse when implementing least squares (slow/unstable)!**
- There are usually special routines for solving linear least squares and ridge regression.

Second Derivative Test for Multi-Dimensional Case



- Second derivative test for multi-dimensional $\mathbf{w} \in \mathbb{R}^n$
Is Hessian positive-semidefinite? ($\mathbf{z}^T \mathbb{H} \mathbf{z} > 0$ for $\mathbf{z} \neq \mathbf{0}$)
 \Leftrightarrow Function is convex.
- Hessian: Matrix of second-order partial derivatives.

$$\mathbb{H}_{j,k} = \frac{\partial^2 f(\mathbf{w})}{\partial w_j \partial w_k}$$

- Matrix algebra² gives:

$$\mathbb{H} = 2\mathbf{X}^T \mathbf{X} = 2 \sum_{i=1}^m \mathbf{x}^{(i)} \mathbf{x}^{(i)\top}; \quad \mathbf{z}^T \mathbb{H} \mathbf{z} = 2 \sum_{i=1}^m (\mathbf{z}^T \mathbf{x}^{(i)})^2 > 0$$

²[Matrix Cookbook. Petersen and Pedersen, 2012]

Linear Regression: Summary

- Linear regression models simple linear relationships between X and y
- The Mean squared error is a quadratic function of the parameter vector w , and has a unique minimum.
- Normal equations: Find the minimum by setting the gradient to zero and solving for w .
- Linear algebra packages have special routines for solving least squares linear regression.

Maximum Likelihood Estimation

- Machine learning models are often more interpretable if they are stated in a probabilistic way.
- Performance measure: What is the probability of the training data given the model parameters? Works only well for discrete data! More general: use densities instead of probabilities
- Likelihood: Probability density of data as a function of model parameters. Generally, likelihood is a function proportional to the density.
- ⇒ Maximum Likelihood Estimation
- Many models can be formulated in a probabilistic way!

Probability density of a data set

- Data:

- ▶ Set of m examples $\mathbf{X} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$
- ▶ Sometimes written as design matrix:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^{(1)T} \\ \vdots \\ \mathbf{x}^{(m)T} \end{bmatrix}$$

- Probability density of a data set \mathbf{X} , parametrized by θ :

$$p_{model}(\mathbf{X}; \theta)$$

Probability density of a data set

- Data points are assumed to be (stochastically) independent (and sometimes identically distributed) random variables (i.d. or i.i.d.)
 - ▶ Assumption made by many ML models.
 - ▶ Identically distributed: Examples come from same distribution.
 - ▶ Independent: Value of one example doesn't influence other example.
 - ▶ \Rightarrow Probability density of the data set is the product of example probability densities.

$$p_{model}(\mathbf{X}; \boldsymbol{\theta}) = \prod_{i=1}^m p_{model}(\mathbf{x}^{(i)}; \boldsymbol{\theta})$$

Maximum Likelihood Estimation

- Likelihood: Probability density of data viewed as function of parameters θ
- (Negative) Log-Likelihood (NLL):
 - ▶ Logarithm is monotonically increasing
 - ★ Maximum of function stays the same
 - ★ Easier to do arithmetic with (sums vs. products)
 - ▶ Optimization is often formulated as minimization \Rightarrow take negative of function.
- Maximum likelihood estimator for θ :

$$\theta_{ML} = \operatorname{argmax}_{\theta} p_{model}(\mathbf{X}; \theta)$$

$$= \operatorname{argmax}_{\theta} \prod_{i=1}^m p_{model}(\mathbf{x}^{(i)}; \theta)$$

$$= \operatorname{argmax}_{\theta} \sum_{i=1}^m \log p_{model}(\mathbf{x}^{(i)}; \theta)$$

Conditional Log-Likelihood

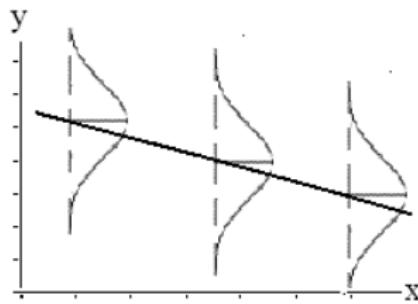
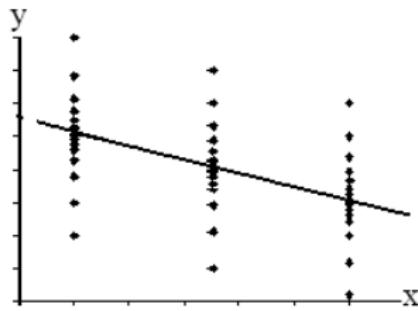
- Log-likelihood can be stated for supervised and unsupervised tasks.
- Unsupervised learning (e.g. density estimation).
 - ▶ Task: model $p_{model}(\mathbf{X}; \theta)$ (as before)
 - ▶ $\mathbf{X} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$
- Supervised learning (Predictive modeling):
 - ▶ Task: model $p_{model}(y|\mathbf{X}; \theta)$
 - ▶ $\mathbf{X} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}, \quad \mathbf{y} = \{y^{(1)}, \dots, y^{(m)}\}$
- Maximum likelihood estimation for the supervised case (independent examples):

$$\theta_{ML} = \operatorname{argmax}_{\theta} P(\mathbf{y}|\mathbf{X}; \theta)$$

$$= \operatorname{argmax}_{\theta} \sum_{i=1}^m \log P(y^{(i)}|\mathbf{x}^{(i)}; \theta)$$

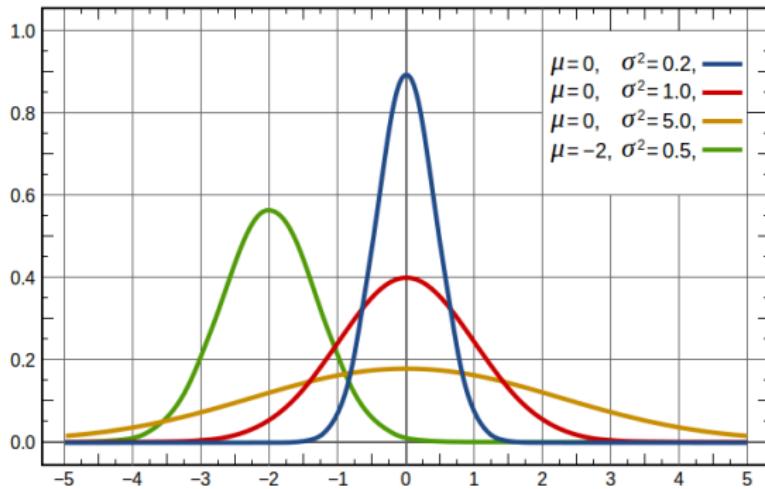
Linear Regression as Maximum Likelihood

- Instead of predicting one value \hat{y} for an input x , model probability distribution $p(y|x)$.
- For the same value of x , different values of y may occur (with different probability).



Gaussian Distribution

- Gaussian distribution: $N(y|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(y-\mu)^2}{2\sigma^2}\right]$
 - ▶ Quadratic function as negative exponent, scaled by variance
 - ▶ Normalization factor $\frac{1}{\sigma\sqrt{2\pi}}$



Linear Regression as Maximum Likelihood

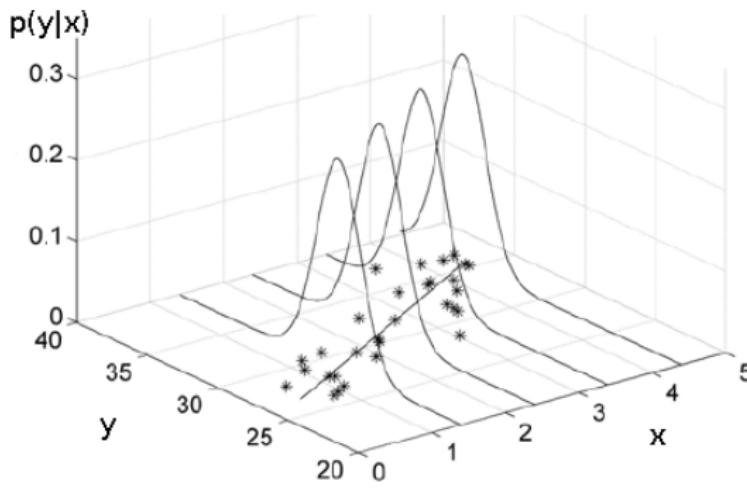
- Assume label y is distributed by a Gaussian, depending on features x

$$p(y|x) = N(y|\mu, \sigma^2)$$

where the mean is determined by the linear transformation

$$\mu = \theta^T x$$

and σ is a constant.



Linear Regression as Maximum Likelihood

- Gaussian distribution: $N(y|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(y-\mu)^2}{2\sigma^2}\right]$
 - ▶ Taking the log makes it a quadratic function!
- Conditional negative log-likelihood:

$$\begin{aligned} & -\log P(\mathbf{y}|\mathbf{X}; \boldsymbol{\theta}) \\ &= -\sum_{i=1}^m \log p(y^{(i)}|\mathbf{x}^{(i)}; \boldsymbol{\theta}) \\ &= m \log \sigma + \frac{m}{2} \log(2\pi) + \sum_{i=1}^m \frac{(y^{(i)} - \boldsymbol{\theta}^T \mathbf{x}^{(i)})^2}{2\sigma^2} \\ &= \text{const} + \frac{1}{2\sigma^2} \sum_{i=1}^m (y^{(i)} - \boldsymbol{\theta}^T \mathbf{x}^{(i)})^2 \end{aligned}$$

- What is the optimal value for $\boldsymbol{\theta}$?

Linear Regression as Maximum Likelihood

- Conditional negative log-likelihood:

$$NLL(\theta) = \text{const} + \frac{1}{2\sigma^2} \sum_{i=1}^m (y^{(i)} - \theta^T \mathbf{x}^{(i)})^2$$

- Compare to previous result:

$$MSE(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^n (y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)})^2$$

- Minimizing NLL under these assumptions is equivalent to minimizing MSE!
- The result of the minimization of the NLL is the MLE $\hat{\theta}$.

A Caveat: Maximum Likelihood is not *Bayesian*

- MLE is simple: Identify distribution and parameters, then maximize!
- MLE accounts for (some) uncertainty: Instead of predicting a single value \hat{y} , treat y as a random variable.
- However, what about the uncertainty about our parameters θ ?
- Shouldn't θ be treated as a random variable, too?
(Instead of a point estimate $\hat{\theta}$?)

A Caveat: Maximum Likelihood is not *Bayesian*

- Uncertainty about θ influenced by:
 - ▶ Our assumptions what reasonable values for θ look like.
 - ▶ The amount of training data.
 - ▶ The properties (variance ...) of the training data.
- *Bayesian inference* is all about modelling randomness of θ in a sound way.
- Why is it called *Bayesian*?

$$p(\theta|\mathbf{X}) = \frac{p(\mathbf{X}|\theta)p(\theta)}{\int p(\mathbf{X}|\theta)p(\theta)d\theta}$$

- The Bayesian approach is theoretically more appealing than working with point estimates.
- In practical terms: Sometimes going the Bayesian way pays off, sometimes it doesn't.

Maximum Likelihood: Summary

- Many machine learning problems can be stated in a probabilistic way.
- Mean squared error linear regression can be stated as a probabilistic model that allows for Gaussian random noise around the predicted value \hat{y} .
- A straightforward optimization is to maximize the likelihood of the training data.
- Maximum likelihood is not Bayesian, and may give undesirable results (e.g. if there is only little training data).
- In practice, MLE and point estimates are often used to solve machine learning problems.

Linear regression and mean squared error of prediction

- The linear model is given by (Y is a random variable, y its realization)

$$Y = \theta^T x + \varepsilon$$

with $\varepsilon \sim N(0, \sigma^2)$. Thus the *expectation* of Y is $E(Y) = \theta^T x$.

- Using the estimated parameter (Least Squares or MLE) $\hat{\theta}$, the predicted value \hat{y} is given by

$$\hat{y} = \hat{\theta}^T x$$

which is also the *estimated expectation* of y , $\widehat{E(Y)}$.

- The theoretical quantity, which is estimated by the introduced MSE, is the so-called MSEP (*mean squared error of prediction*), which is also an expectation, based on a *new* observation Y_{m+1} (we ignore the index in the following) and its prediction $\hat{y} = \hat{\theta}^T x_{m+1}$:

$$MSEP = E\{(Y - \hat{y})^2\} = E\{(Y - E(Y) + E(Y) - \hat{y})^2\} = Var(Y) + MSE(\hat{y}).$$

Note, that some terms are zero, because of stochastic independence of Y_{m+1} and \hat{y} . Note, that $MSEP > Var(Y) = \sigma^2$.

From Regression to Classification

- So far, linear regression:
 - ▶ A simple linear model.
 - ▶ Probabilistic interpretation.
 - ▶ Find optimal parameters using Maximum Likelihood Estimation.
- Can we do something similar for classification?
- ⇒ Logistic Regression (*... it's not actually used for regression ...*)

Logistic Regression

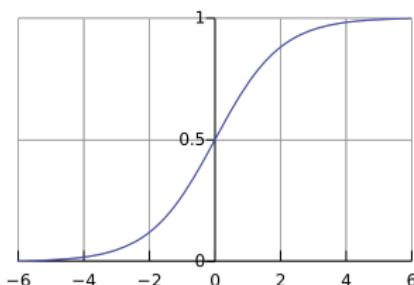
- Binary logistic model:

Estimate the probability of a binary response $y \in \{0, 1\}$ based on features x .

- Logistic Regression is a *Generalized Linear Model*:

Linear model is related to the response variable via a **link function**.

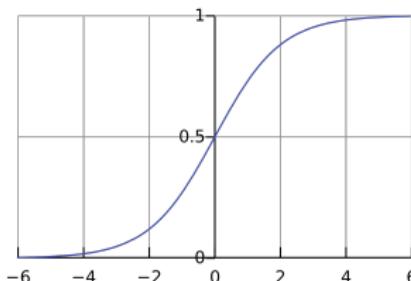
$$p(Y = 1|x; \theta) = f(\theta^T x)$$



(Note: Y denotes a random variable, whereas $y, y^{(i)}, 0, 1$ denote values that the random variable can take on. If the random variable is obvious from the context, it may be omitted.)

Logistic Regression

- Recall linear regression: $p(y|x; \theta) = N(y; \theta^T x, \sigma^2 I)$
 - ▶ Predicts $y \in \mathbb{R}$
- Classification: Outcome (per example) 0 or 1
 - ▶ Logistic sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$



- ▶ Logistic Regression: Linear function + logistic sigmoid

$$p(Y=1|x; \theta) = \sigma(\theta^T x)$$

Binary Logistic Regression

Probability of different outcomes (for one example):

- Probability of positive outcome:

$$p(Y = 1|x; \theta) = \frac{1}{1 + e^{-\theta^T x}}$$

- Probability of negative outcome:

$$p(Y = 0|x; \theta) = 1 - p(Y = 1|x; \theta)$$

Probability of a Training Example

- Probability for actual label $y^{(i)}$ given features $\mathbf{x}^{(i)}$
- Can be written for both labels (0 and 1) without case distinction
- Label exponentiation trick: use $x^0 = 1$

$$\begin{aligned} p(Y = y^{(i)} | \mathbf{x}^{(i)}; \boldsymbol{\theta}) \\ &= \begin{cases} p(Y = 1 | \mathbf{x}^{(i)}; \boldsymbol{\theta}) & \text{if } y^{(i)} = 1 \\ p(Y = 0 | \mathbf{x}^{(i)}; \boldsymbol{\theta}) & \text{if } y^{(i)} = 0 \end{cases} \\ &= \begin{cases} p(Y = 1 | \mathbf{x}^{(i)}; \boldsymbol{\theta})^1 p(Y = 0 | \mathbf{x}^{(i)}; \boldsymbol{\theta})^0 & \text{if } y^{(i)} = 1 \\ p(Y = 1 | \mathbf{x}^{(i)}; \boldsymbol{\theta})^0 p(Y = 0 | \mathbf{x}^{(i)}; \boldsymbol{\theta})^1 & \text{if } y^{(i)} = 0 \end{cases} \\ &= p(Y = 1 | \mathbf{x}^{(i)}; \boldsymbol{\theta})^{y^{(i)}} p(Y = 0 | \mathbf{x}^{(i)}; \boldsymbol{\theta})^{1-y^{(i)}} \end{aligned}$$

Binary Logistic Regression

- Conditional Negative Log-Likelihood (NLL):

$$\begin{aligned} NLL(\theta) &= -\log p(\mathbf{y}|\mathbf{X}; \theta) \\ &= -\log \prod_{i=1}^m p(Y = y^{(i)} | \mathbf{x}^{(i)}; \theta) \\ &= -\log \prod_{i=1}^m p(Y = 1 | \mathbf{x}^{(i)}; \theta)^{y^{(i)}} (1 - p(Y = 1 | \mathbf{x}^{(i)}; \theta))^{1-y^{(i)}} \\ &= -\sum_{i=1}^m y^{(i)} \log \sigma(\boldsymbol{\theta}^T \mathbf{x}^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(\boldsymbol{\theta}^T \mathbf{x}^{(i)})) \end{aligned}$$

- No closed form solution for minimum
- Use numerical / iterative methods.
- LBFGS, Gradient descent ...

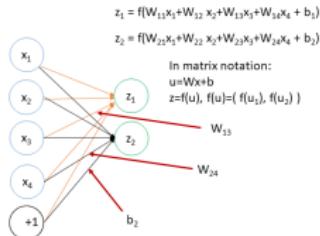
Logistic Regression

- Logistic regression: Logistic sigmoid function applied to a weighted linear combination of feature values.
- To be interpreted as the probability that the label for a specific example equals 1.
- Applying the model on test data: Predict $y^{(i)} = 1$ if

$$p(Y = 1 | \mathbf{x}^{(i)}; \boldsymbol{\theta}) > 0.5$$

- No closed form solution for maximizing NLL, iterative methods necessary.

Chapter 4: Introduction to Machine Learning – Optimization, Deep Feed Forward Networks, Backpropagation, Regularization



Christian Heumann

(largely based on slides from Nina Poerner, Benjamin Roth, Marina Speranskaya)

CIS LMU München, Department of Statistics LMU München

November 2020

Optimization

- Optimization: Minimize some function $J(\theta)$ by altering θ .
- Maximize $f(\theta)$ by minimizing $J(\theta) = -f(\theta)$
- $J(\theta)$:
 - ▶ “criterion”, “objective function”, “cost function”, “loss function”, “error function”
 - ▶ In a probabilistic machine learning setting often (conditional) negative log-likelihood:

$$-\log p(\mathbf{X}; \theta)$$

or

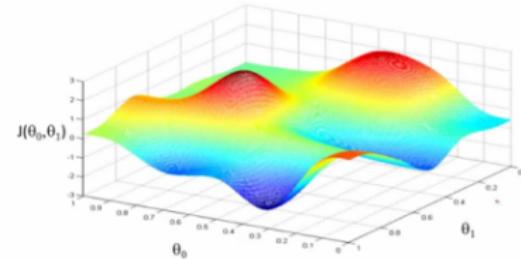
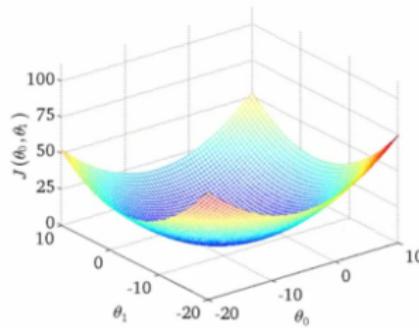
$$-\log p(\mathbf{y}|\mathbf{X}; \theta)$$

as a function of θ

- ▶ $\theta^* = \arg \min_{\theta} J(\theta)$

Optimization

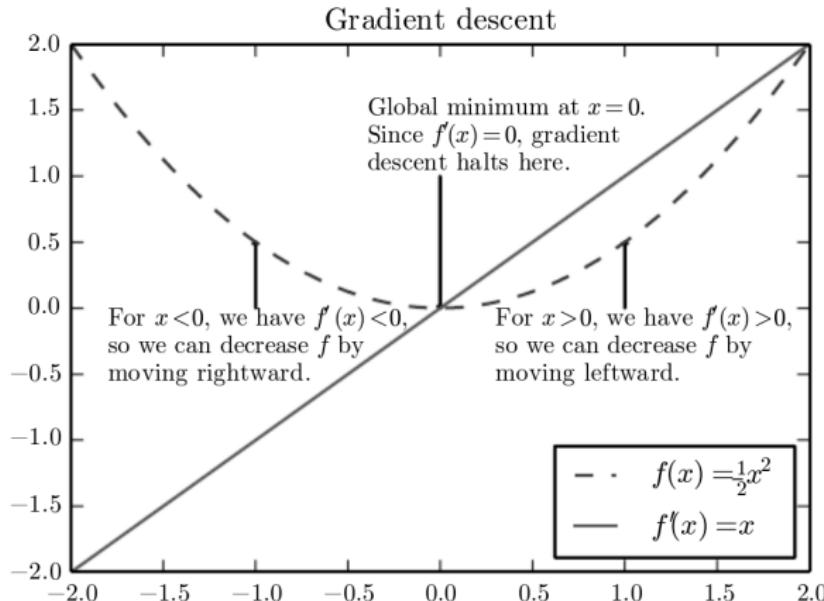
- If $J(\theta)$ is convex, it is minimized where $\nabla_{\theta} J(\theta) = \mathbf{0}$
- If $J(\theta)$ is not convex, the gradient can help us to improve our objective nevertheless (and find a local optimum).
- Many optimization techniques were originally developed for convex objective functions, but are found to be working well for non-convex functions too.
- Use the fact that gradient indicates the slope of the function in the direction of steepest increase.



Gradient-Based Optimization

- Derivative: Given a small change in input, what is the corresponding change in output?

$$f(x + \epsilon) \approx f(x) + \epsilon f'(x)$$



- $f(x - \epsilon \operatorname{sign} f'(x)) < f(x)$ for small enough ϵ

Gradient Descent

- For $J(\theta) : \mathbb{R}^n \rightarrow \mathbb{R}$
- If partial derivative $\frac{\partial J(\theta)}{\partial \theta_j} > 0$, $J(\theta)$ will increase for small increases of θ_j
⇒ go in opposite direction of gradient (since we want to minimize)
- Steepest descent: iterate

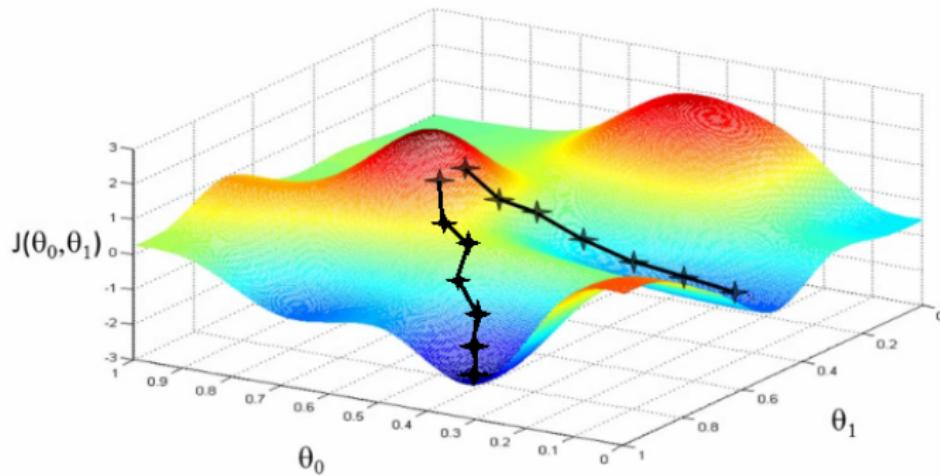
$$\theta_{t+1} \leftarrow \theta_t - \eta \nabla_{\theta} J(\theta_t)$$

where θ_t is the actual parameter, $J(\theta_t)$ is the objective function evaluated at θ_t and θ_{t+1} is the updated parameter.

- η is the learning rate (set to small positive constant).
- Converges if $\nabla_{\theta} J(\theta)$ is (close to) **0**

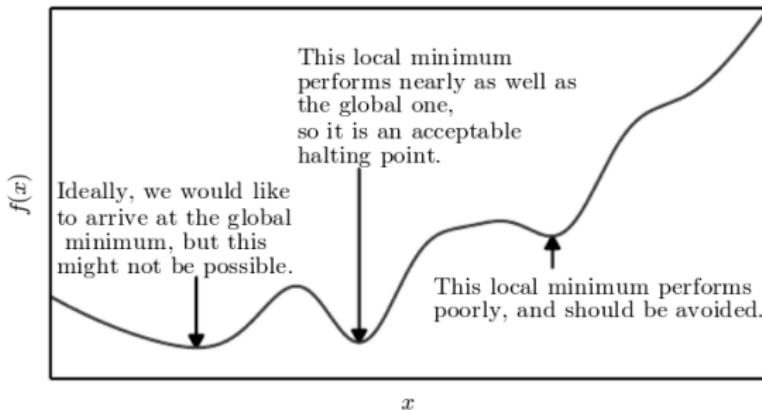
Local Minima

- If the function is non-convex, different results can be obtained at convergence, depending on initialization of θ .

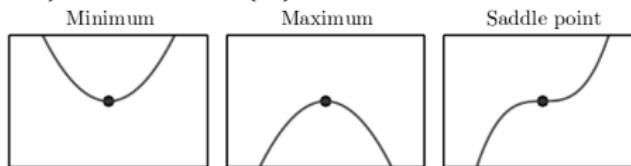


Local Minima

- Minima can be global or local:



- Critical (stationary) points: $f'(x) = 0$



- For neural networks, only good (not perfect) parameter values can be found.

Gradient Descent for Logistic Regression

$$\begin{aligned}\nabla_{\theta} NLL(\theta) &= -\nabla_{\theta} \sum_{i=1}^m y^{(i)} \log \sigma(\theta^T x^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(\theta^T x^{(i)})) \\ &= -\sum_{i=1}^m (y^{(i)} - \sigma(\theta^T x^{(i)})) x^{(i)}\end{aligned}$$

- The gradient descent update becomes:

$$\theta_{t+1} := \theta_t + \eta \sum_{i=1}^m (y^{(i)} - \sigma(\theta_t^T x^{(i)})) x^{(i)}$$

- Note: Which feature weights are increased, which are decreased?

Derivation of Gradient for Logistic Regression

This is a great exercise! Use the following facts:

Gradient	$(\nabla_{\theta} f(\theta))_j = \frac{\partial f(\theta)}{\partial \theta_j}$
Derivative of a sum	$\frac{d}{dz} \sum_i f_i(z) = \sum_i \frac{df_i(z)}{dz}$
Chain rule	$F(z) = f(g(z)) \Rightarrow F'(z) = f'(g(z))g'(z)$
Derivative of logarithm	$\frac{d \log z}{dz} = 1/z$
D. of logistic sigmoid	$\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$
Partial d. of dot-product	$\frac{\partial \theta^T x}{\partial \theta_j} = x_j$

Gradient Descent: Summary

- Iterative method for function minimization.
- Gradient indicates rate of change in objective function, given a local change to feature weights.
- Subtract the gradient:
 - ▶ **decrease** parameters that (locally) have **positive** correlation with objective
 - ▶ **increase** parameters that (locally) have **negative** correlation with objective
- Gradient updates only have the desired properties in a small region around previous parameters θ_t . Control locality by the learning rate η .
- Gradient descent is slow: For relatively small step in the right direction, all of training data has to be processed.
- This version of gradient descent is often also called *batch gradient descent*.

Stochastic Gradient Descent (SGD)

- *Batch gradient descent* is slow: For relatively small step in the right direction, all of training data has to be processed.

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \eta \nabla_{\boldsymbol{\theta}} \sum_{i=1}^m \log p(y_i | \mathbf{x}_i; \boldsymbol{\theta})$$

- *Stochastic gradient descent* in a nutshell:

- ▶ For each update, only use random sample \mathbb{B}_t of training data (mini-batch).

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \eta \nabla_{\boldsymbol{\theta}} \sum_{i \in \mathbb{B}_t} \log p(y_i | \mathbf{x}_i; \boldsymbol{\theta})$$

- ▶ Mini-batch size can also just be 1.

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \eta \nabla_{\boldsymbol{\theta}} \log p(y_t | \mathbf{x}_t; \boldsymbol{\theta})$$

- ⇒ More frequent updates.

Stochastic Gradient Descent (SGD)

- The actual gradient is *approximated* using only a sub-sample of the data.
- For objective functions that are highly non-convex, the random deviations of these approximations may even help to escape local minima.
- Treat batch size and learning rate as hyper-parameters.

Deep Feedforward Networks

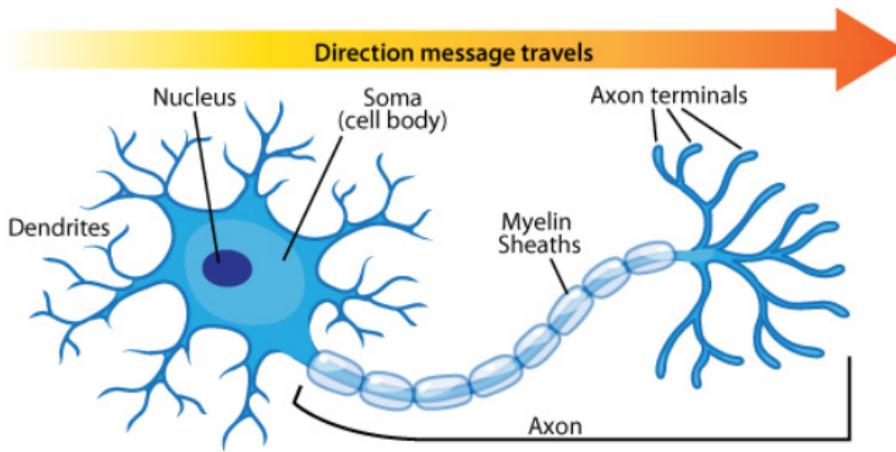
- Function approximation: find good mapping $\hat{y} = f(x; \theta)$ (or more exactly $f(x; \hat{\theta})$, but we omit the hat in future).
- Network: Composition of functions $f^{(1)}, f^{(2)}, f^{(3)}$ with multi-dimensional input and output
- Each $f^{(i)}$ represents one *layer* $f(x) = f^{(1)}(f^{(2)}(f^{(3)}(x)))$
- *Feedforward*:
 - ▶ Input → intermediate representation → output
 - ▶ No feedback connections
 - ▶ Cf. *recurrent* networks

Deep Feedforward Networks: Training

- Loss function defined on output layer, e.g. $(y - f(\mathbf{x}; \theta))^2$
- Quality criterion on other layers not directly defined.
- Training algorithm must decide how to use those layers most effectively (w.r.t. loss on output layer)
- Non-output layers can be viewed as providing a feature function $\phi(\mathbf{x})$ of the input, that is to be learned.

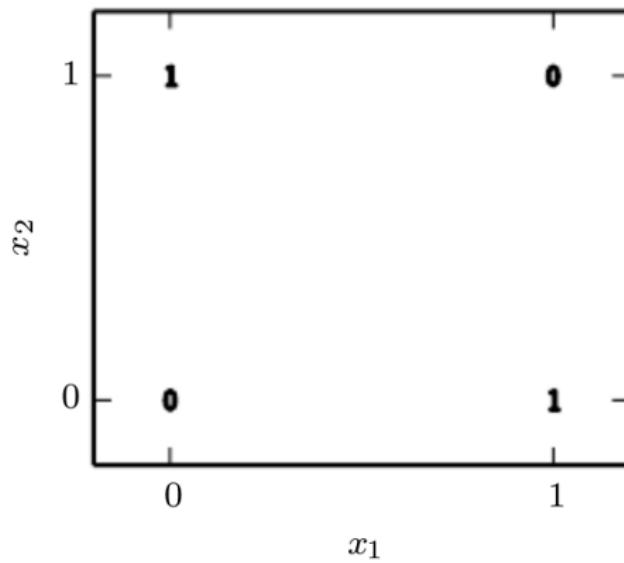
"Neural" Networks

- Inspired by biological neurons (nerve cells)
- Neurons are connected to each other, and receive and send electrical pulses.
- *"If the [input] voltage changes by a large enough amount, an all-or-none electrochemical pulse called an action potential is generated, which travels rapidly along the cell's axon, and activates synaptic connections with other cells when it arrives."* (Wikipedia)



Activation Functions with Non-Linearities

- Linear Functions are limited in what they can express.
- Famous example: XOR
- Simple layered non-linear functions can represent XOR.



Design Choices for Output Units

- Can typically be interpreted as probabilities.
 - ▶ Logistic sigmoid
 - ▶ Softmax
 - ▶ mean and variance of a Gaussian, ...
- Trained with negative log-likelihood.

Softmax

- Logistic sigmoid
 - ▶ Vector y of binary outcomes, with no constraints on how many can be 1.
 - ▶ Bernoulli distribution.
- Softmax
 - ▶ Exactly one element of y is 1.
 - ▶ Multinomial (categorical) distribution.

$$p(Y = i|\phi(x))$$

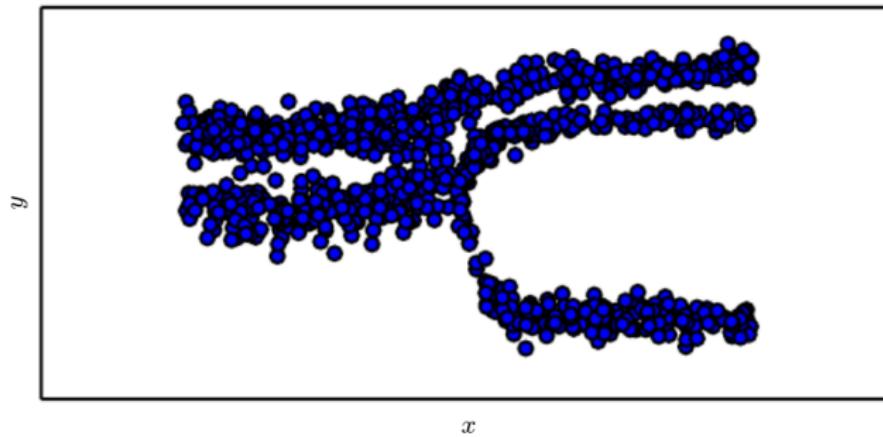
$$\sum_i p(Y = i|\phi(x)) = 1$$

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Parametrizing a Gaussian Distribution

- Use final layer to predict parameters of Gaussian mixture model.
- Weight of mixture component: softmax.
- Means: no non-linearity.
- Precisions ($\frac{1}{\sigma^2}$) need to be positive: softplus

$$\text{softplus}(z) = \ln(1 + \exp(z))$$

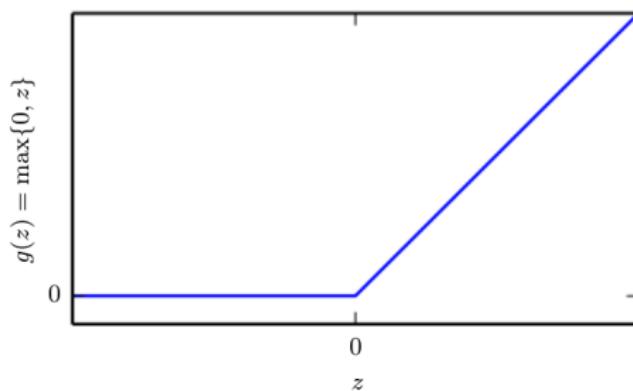


Rectified Linear Units

- Rectified Linear Unit:

$$relu(z) = \max(0, z)$$

$$z = \mathbf{x}^T \mathbf{w} + b$$



- Consistent gradient of 1 when unit is *active* (i.e. if there is an error to propagate).
- Default choice for hidden units.

A Simple ReLU Network to Solve XOR

$$f(\mathbf{x}; \mathbf{W}, \mathbf{c}, \mathbf{w}) = \mathbf{w}^T \max(0, \mathbf{W}^T \mathbf{x} + \mathbf{c})$$

$$\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

Other Choices for Hidden Units

- A good activation function aids learning, and provides large gradients.
- Sigmoidal functions (logistic sigmoid)
 - ▶ have only a small region before they flatten out in either direction.
 - ▶ Practice shows that this seems to be ok in conjunction with Log-loss objective.
 - ▶ But they don't work as well as hidden units.
 - ▶ ReLU are better alternative since gradient stays constant.
- Other hidden unit functions:
 - ▶ maxout: take maximum of several values in previous layer.
 - ▶ purely linear: can serve as low-rank approximation.

- Forward propagation: Input information x propagates through network to produce output \hat{y} (and cost $J(\theta)$ in training)
- Back-propagation:
 - ▶ compute gradient w.r.t. model parameters
 - ▶ Cost gradient propagates backwards through the network
- Back-propagation is part of learning procedure (e.g. stochastic gradient descent), not learning procedure in itself.

Chain Rule of Calculus: Real Functions

- Let

$$x, y, z \in \mathbb{R}$$

$$f, g : \mathbb{R} \rightarrow \mathbb{R}$$

$$y = g(x)$$

$$z = f(g(x)) = f(y)$$

- Then

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

Chain Rule of Calculus: Multivariate Functions

- Let

$$\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n, z \in \mathbb{R}$$

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

$$g : \mathbb{R}^m \rightarrow \mathbb{R}^n$$

$$\mathbf{y} = g(\mathbf{x})$$

$$z = f(g(\mathbf{x})) = f(\mathbf{y})$$

- Then

$$\frac{\partial z}{\partial x_i} = \frac{\partial z}{\partial y_1} \frac{\partial y_1}{\partial x_i} + \frac{\partial z}{\partial y_2} \frac{\partial y_2}{\partial x_i} + \dots + \frac{\partial z}{\partial y_n} \frac{\partial y_n}{\partial x_i} = \sum_{j=1}^n \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

- In order to write this in vector notation, we need to define the Jacobian matrix.

Jacobian

- The Jacobian matrix is the matrix of all first-order partial derivatives of a vector-valued function.

$$\mathbf{J} = \frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial g(\mathbf{x})_1}{\partial x_1} & \dots & \frac{\partial g(\mathbf{x})_1}{\partial x_m} \\ \frac{\partial g(\mathbf{x})_2}{\partial x_1} & & \frac{\partial g(\mathbf{x})_2}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial g(\mathbf{x})_n}{\partial x_1} & \dots & \frac{\partial g(\mathbf{x})_n}{\partial x_m} \end{bmatrix}$$

- How to write in terms of gradients?
- We can write the chain rule as:

$$\nabla_{\mathbf{x}} z =$$

Jacobian

- The Jacobian matrix is the matrix of all first-order partial derivatives of a vector-valued function.

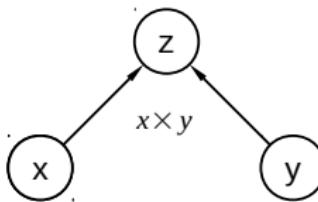
$$\mathbf{J} = \frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial g(\mathbf{x})_1}{\partial x_1} & \dots & \frac{\partial g(\mathbf{x})_1}{\partial x_m} \\ \frac{\partial g(\mathbf{x})_2}{\partial x_1} & & \frac{\partial g(\mathbf{x})_2}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial g(\mathbf{x})_n}{\partial x_1} & \dots & \frac{\partial g(\mathbf{x})_n}{\partial x_m} \end{bmatrix}$$

- How to write in terms of gradients?
- We can write the chain rule as:

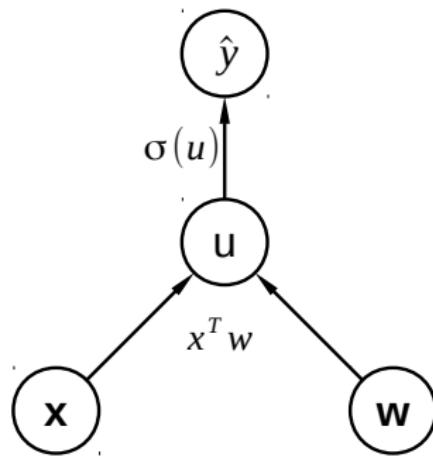
$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right)^T \nabla_{\mathbf{y}} z$$

Viewing the Network as a Graph

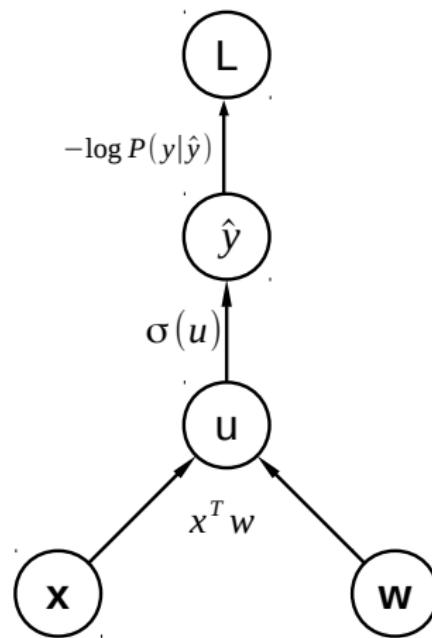
- Nodes are function outputs (can be scalar or vector valued)
- Arrows are inputs
- Example: Scalar multiplication $z = xy$.



Which Function?

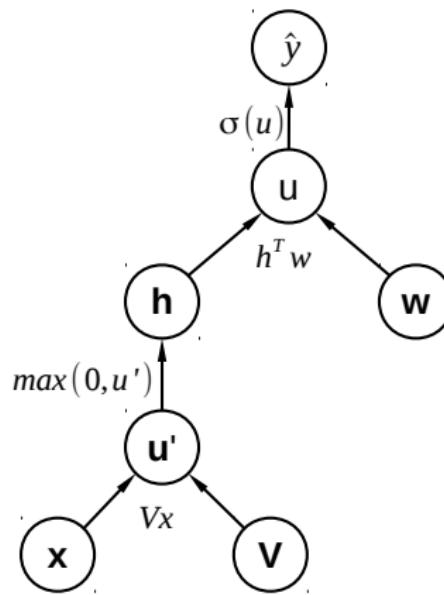


Graph with Cost



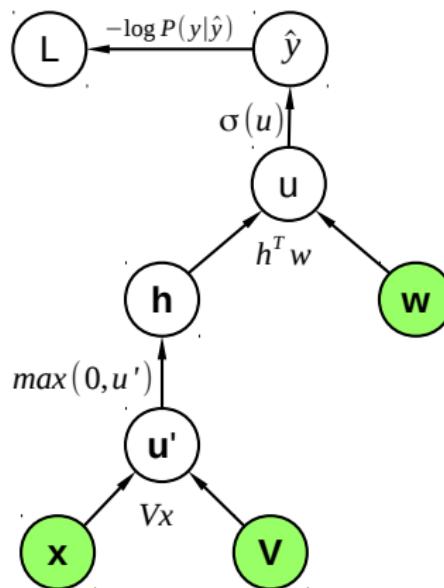
Which Function?

- Parameter vectors can be converted to matrix as needed.



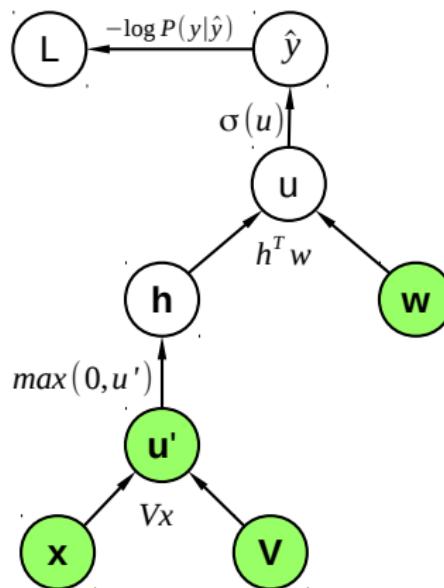
Forward Pass

- Green: known or computed.



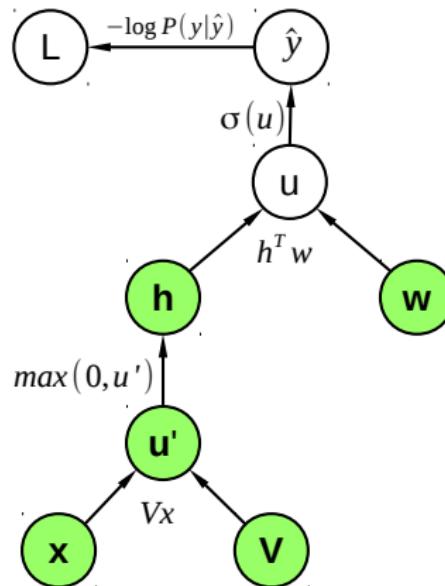
Forward Pass

- Green: known or computed.



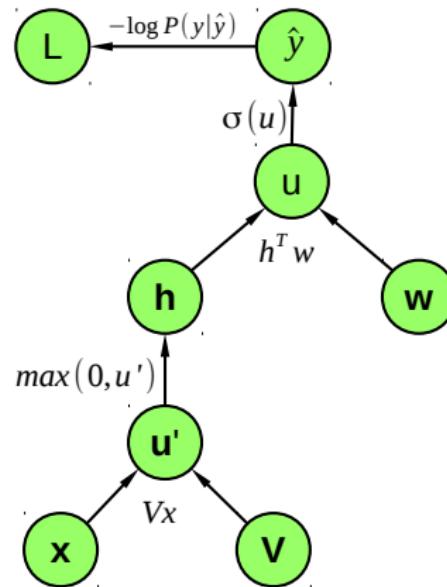
Forward Pass

- Green: known or computed.



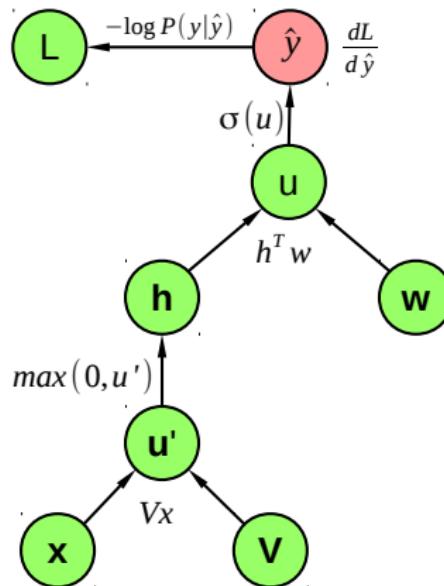
Forward Pass

- End of forward pass (some steps skipped).



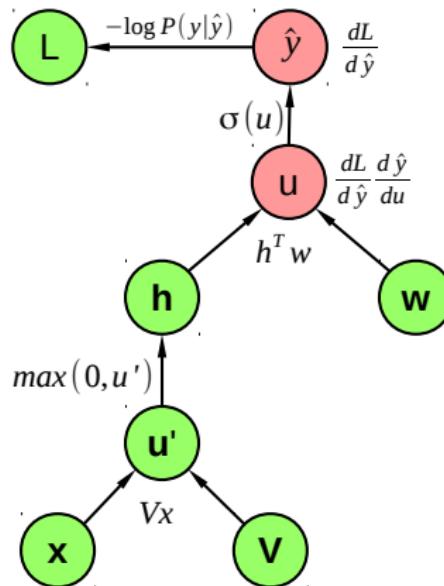
Backward Pass

- Red: gradient of cost computed for node.



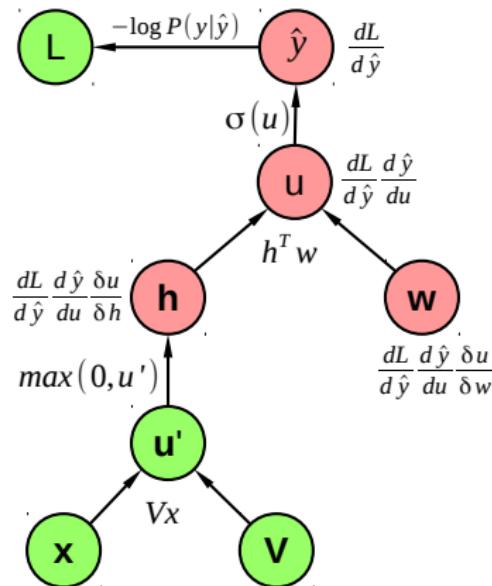
Backward Pass

- Red: gradient of cost computed for node.



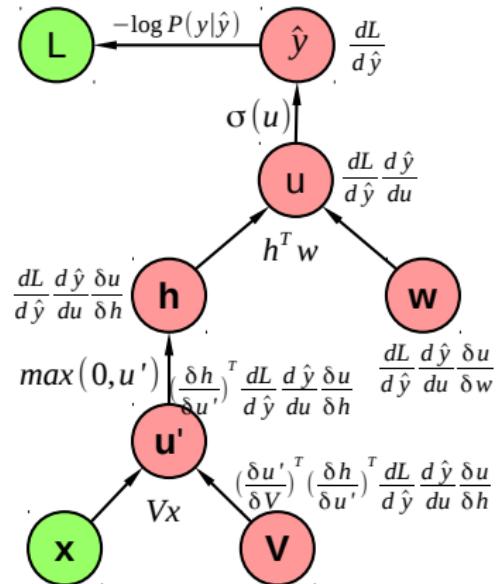
Backward Pass

- Red: gradient of cost computed for node.



End of Backward Pass

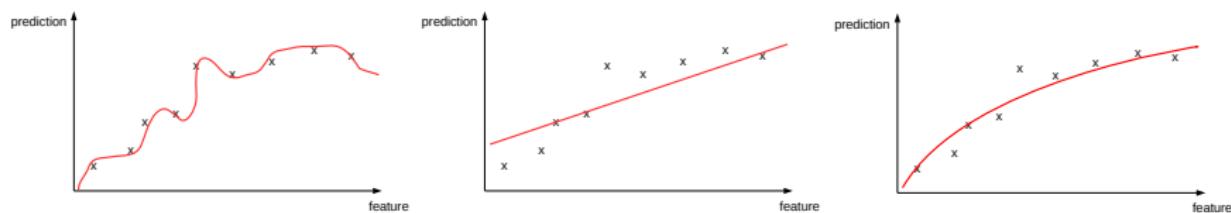
- We have the gradients for all parameters, let's use them for SGD.



Summary

- Gradient descent: Minimize loss by iteratively subtracting gradient from parameter vector.
- Stochastic gradient descent: Approximate gradient by considering small subsets of examples.
- Regularization: penalize large parameter values, e.g. by adding L2-norm of parameter vector.
- Feedforward networks: layers of (non-linear) function compositions.
- Output non-linearities: interpreted as probability densities (logistic sigmoid, softmax, Gaussian)
- Hidden layers: Rectified linear units ($\max(0, z)$)
- Backpropagation: Compute gradient of cost w.r.t. parameters using chain rule.

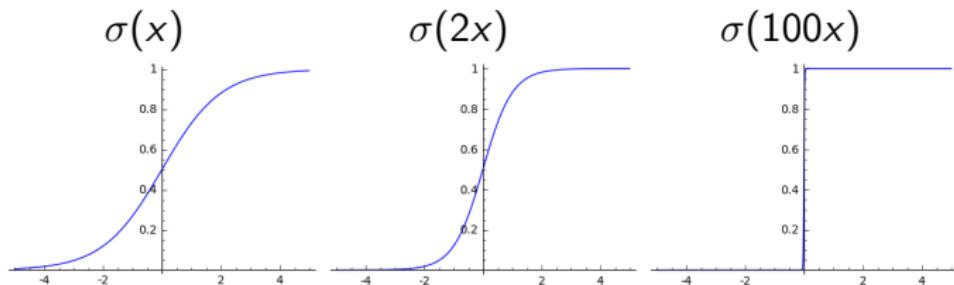
Regularization



- Overfitting vs. underfitting
- Regularization: Any modification to a learning algorithm for reducing its generalization error but not its training error
- Solution space is still the same

L2-Regularization

- Large parameters → overfitting



- Prefer models with smaller feature weights.
- Popular regularizers:
 - ▶ Penalize large L2 norm.
 - ▶ Penalize large L1 norm (aka LASSO, induces sparsity)

Regularization

- Add term that penalizes large L2 norm.
- The amount of penalty is controlled by a parameter λ
 - ▶ Linear regression:

$$J(\theta) = MSE(\theta) + \frac{\lambda}{2} \theta^T \theta$$

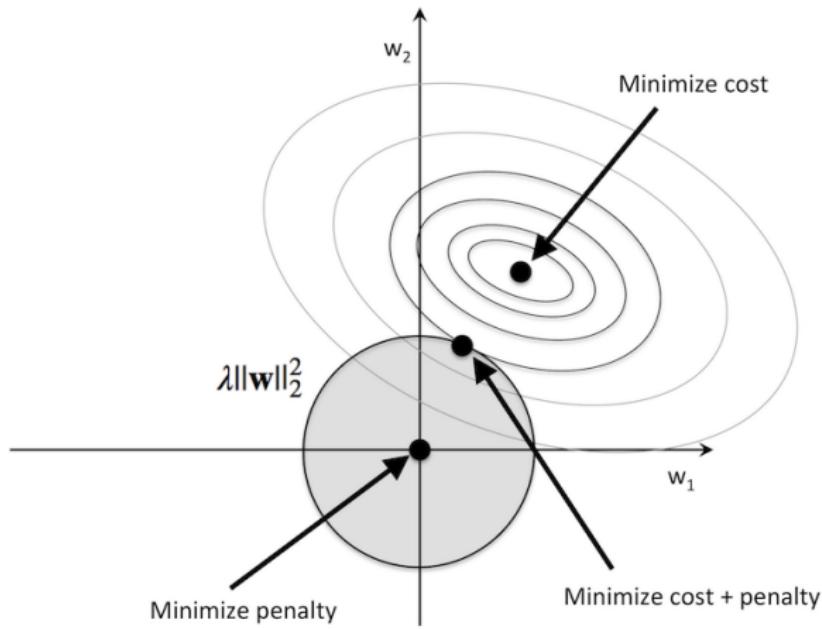
- ▶ Logistic regression:

$$J(\theta) = NLL(\theta) + \frac{\lambda}{2} \theta^T \theta$$

- From a Bayesian perspective, L2-regularization corresponds to a Gaussian prior on the parameters.

L2-Regularization

- The surface of the objective function is now a combination of the original cost, and the regularization penalty.



L2-Regularization

- Gradient of regularization term:

$$\nabla_{\theta} \frac{\lambda}{2} \theta^T \theta = \lambda \theta$$

- Gradient descent for regularized cost function:

$$\theta_{t+1} := \theta_t - \eta \nabla_{\theta} (NLL(\theta_t) + \lambda \theta_t^T \theta_t)$$

\Leftrightarrow

$$\theta_{t+1} := (1 - \eta \lambda) \theta_t - \eta \nabla_{\theta} NLL(\theta_t)$$

Word Embeddings and Word2Vec

Deep Learning for NLP: Lecture 5

Centrum für Informations- und Sprachverarbeitung
Ludwig-Maximilians-Universität München
poerner@cis.uni-muenchen.de

December 02, 2020

Outline

- ① Motivation for word embeddings
- ② Word2Vec
 - Skipgram task
 - CBOW task
 - Naive softmax model
 - Hierarchical softmax model
 - Negative sampling model
 - FastText
- ③ Applications of pretrained word embeddings
 - Initialization of word embedding layer
 - Word analogies
 - Cross-lingual embedding spaces

Outline

1 Motivation for word embeddings

2 Word2Vec

- Skipgram task
- CBOW task
- Naive softmax model
- Hierarchical softmax model
- Negative sampling model
- FastText

3 Applications of pretrained word embeddings

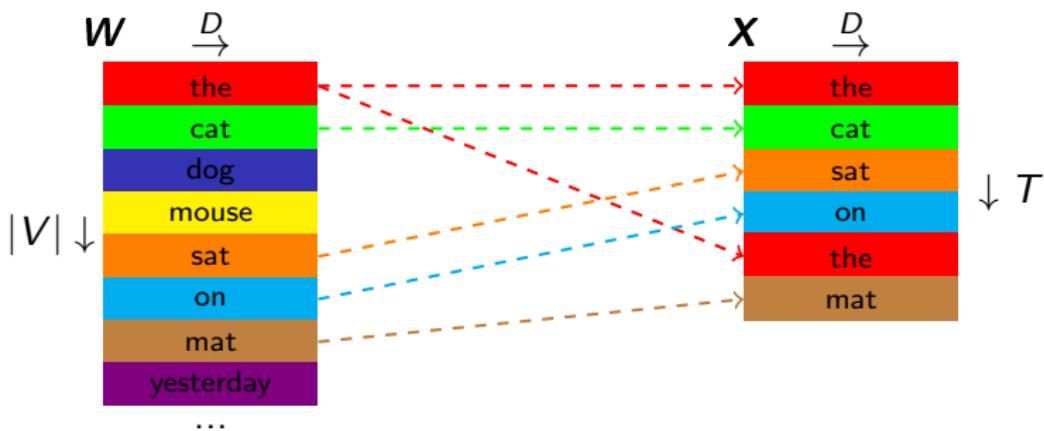
- Initialization of word embedding layer
- Word analogies
- Cross-lingual embedding spaces

Motivation for word embeddings

- Deep Learning models understand numerical representations (vectors, matrices...)
- Texts are sequences of symbolic units (words, ngrams, ...)
- Step 1 in any NLP deep learning model: Turn symbols (e.g., words) into vectors

Word vector lookup

- For now, assume that we have finite word vocabulary V e.g.
 - $V = \{\text{the, cat, dog, mouse, sat, on, mat, yesterday, ...}\}$ with a bijective indexing function $\mathcal{I}_V : V \rightarrow \{1 \dots |V|\}$
- Let $\mathbf{W} \in \mathbb{R}^{|V| \times D}$ be a matrix whose rows are word vectors
- Represent word $x \in V$ as $\mathbf{w}^{(x)} = \mathbf{w}_{\mathcal{I}_V(x)}$
- Represent sentence $X = (x_1 \dots x_T)$ as matrix \mathbf{X} with $x_t = \mathbf{w}^{(x_t)}$.
- Then use \mathbf{X} as input to downstream model (CNN, RNN ...)



Motivation for word embeddings

- So where do we get \mathbf{W} from?
- Simple solution: Identity matrix, with $w_i^{(x)} = 1$ if $i = \mathcal{I}_V(x)$ and $w_i^{(x)} = 0$ otherwise

$$\mathbf{w}^{(\text{the})} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \quad \mathbf{w}^{(\text{cat})} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \quad \mathbf{w}^{(\text{dog})} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

- All vectors are orthogonal to each other \rightarrow no notion of word similarity
- That's not what we want...

Motivation for word embeddings

- Word embeddings are dense (\neq sparse), trainable word vectors
- $\mathbf{w}^{(i)} \in \mathbb{R}^D$, where typically $50 \leq D \leq 1000 \ll |V|$
- Allows us to calculate similarities between words, e.g., with cosine similarity:

$$\text{sim}(\mathbf{w}^{(i)}, \mathbf{w}^{(j)}) = \frac{\mathbf{w}^{(i)T} \mathbf{w}^{(j)}}{\|\mathbf{w}^{(i)}\|_2 \cdot \|\mathbf{w}^{(j)}\|_2}$$

Word embeddings from scratch

- Word embeddings can be trained from scratch on supervised data:
 - ▶ Initialize \mathbf{W} randomly
 - ▶ Use as input layer to some model (CNN, RNN, ...)
 - ▶ During training, back-propagate gradients from the model into the embedding layer, and update \mathbf{W}
- Result: Words that play similar roles in the training task get similar embeddings.
- For example: If our training task is sentiment analysis, we might expect

$$\text{sim}(\mathbf{w}^{(\text{great})}, \mathbf{w}^{(\text{awesome})}) \approx 1$$

Motivation for pretrained word embeddings

- Supervised training sets tend to be small (labeled data is expensive)
- Many words that are relevant at test time will be infrequent or unseen at training time
- The embeddings of infrequent words may have low quality, unseen words have no embeddings at all.

- We have more unlabeled than labeled data.
- So let's pretrain embeddings on the unlabeled data first.

Questions?

Take a moment to write down any questions you have for the QA session!

Outline

1 Motivation for word embeddings

2 Word2Vec

- Skipgram task
- CBOW task
- Naive softmax model
- Hierarchical softmax model
- Negative sampling model
- FastText

3 Applications of pretrained word embeddings

- Initialization of word embedding layer
- Word analogies
- Cross-lingual embedding spaces

Word2Vec

- Proposed in Mikolov et al. (2013): Efficient estimation of word representations in vector space.
- The basis of many embedding pretraining models (including Word2Vec) is the **distributional hypothesis**:
- “a word is characterized by the company it keeps” (Firth, 1957)

Outline

1 Motivation for word embeddings

2 Word2Vec

- Skipgram task

- CBOW task

- Naive softmax model

- Hierarchical softmax model

- Negative sampling model

- FastText

3 Applications of pretrained word embeddings

- Initialization of word embedding layer

- Word analogies

- Cross-lingual embedding spaces

The skipgram task

- Let X be an unlabeled text (sequence of words), and let M be a hyperparameter
- Let x_t be the t 'th word, and let $C_t = (x_{t-M} \dots x_{t-1}, x_{t+1} \dots x_{t+M})$ be a context window around x_t .
- Example:

$$M = 1, X = (\text{the, cat, sat, on, the, mat})$$

- $x_t = \text{the}, C_t = (\text{cat})$
- $x_t = \text{cat}, C_t = (\text{the, sat})$
- $x_t = \text{sat}, C_t = (\text{cat, on})$
- $x_t = \text{on}, C_t = (\text{sat, the})$
- ... and so on.

Skipgram task

- The skipgram task is to maximize the likelihood of the context words, given their center word:

$$\operatorname{argmax}_{x_t, C_t} \prod_{x_t' \in C_t} P(x_t' | x_t)$$

- Expressed as a negative log likelihood loss:

$$\mathcal{J} = \sum_{x_t, C_t} \sum_{x_t' \in C_t} -\log(P(x_t' | x_t))$$

Outline

1 Motivation for word embeddings

2 Word2Vec

- Skipgram task
- CBOW task
- Naive softmax model
- Hierarchical softmax model
- Negative sampling model
- FastText

3 Applications of pretrained word embeddings

- Initialization of word embedding layer
- Word analogies
- Cross-lingual embedding spaces

CBOW task

- The continuous bag of words (CBOW) task is to maximize the likelihood of the center words, given their context words:

$$\operatorname{argmax} \prod_{x_t, C_t} P(x_t | C_t)$$

- Expressed as a negative log likelihood loss:

$$\mathcal{J} = \sum_{x_t, C_t} -\log(P(x_t | C_t))$$

Skipgram and CBOW

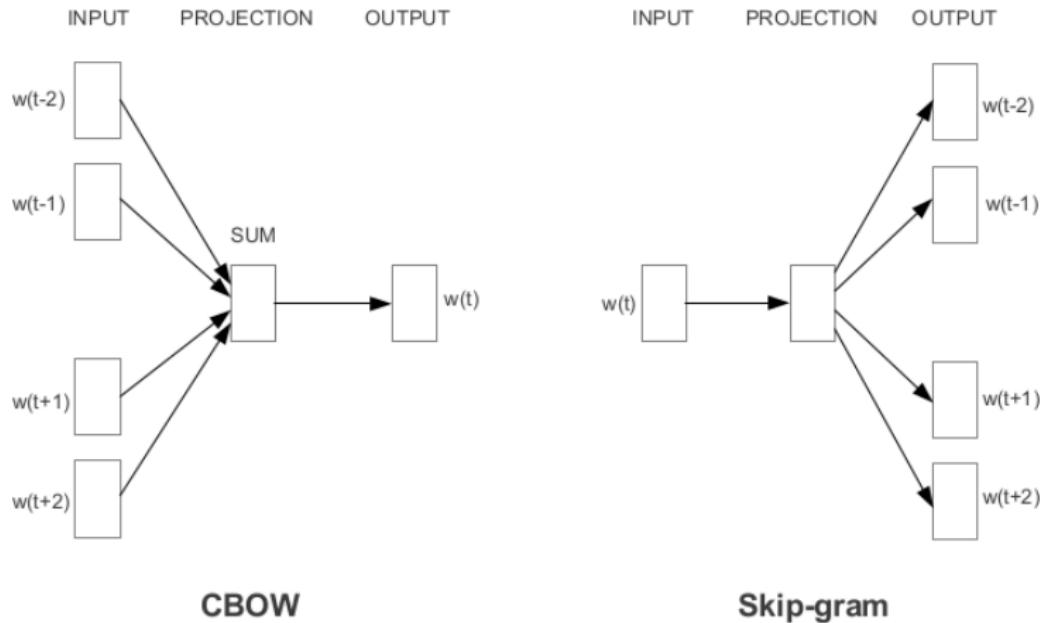


Figure from Mikolov et al. (2013): Efficient Estimation of Word Representations in Vector Space

Outline

1 Motivation for word embeddings

2 Word2Vec

- Skipgram task
- CBOW task
- **Naive softmax model**
- Hierarchical softmax model
- Negative sampling model
- FastText

3 Applications of pretrained word embeddings

- Initialization of word embedding layer
- Word analogies
- Cross-lingual embedding spaces

Naive softmax model

- Let all words be represented as two types of vectors:
 - $\mathbf{v}^{(*)} \in \mathbb{R}^D$ (if they are being predicted)
 - $\mathbf{w}^{(*)} \in \mathbb{R}^D$ (if they are being conditioned on)
 - Total number of parameters: $2 \cdot |V| \cdot D$
- Skipgram:

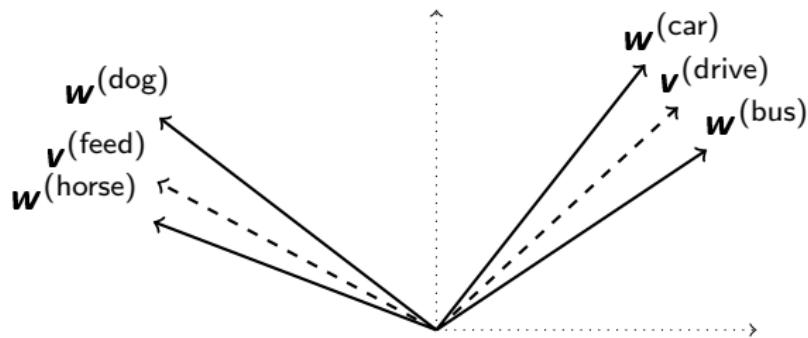
$$P(x_{t'}|x_t) = \text{softmax}(\mathbf{V}\mathbf{w}^{(x_t)})_{\mathcal{I}_V(x_{t'})} = \frac{\exp(\mathbf{w}^{(x_t)T} \mathbf{v}^{(x_{t'})})}{\sum_{\bar{x} \in V} \exp(\mathbf{w}^{(x_t)T} \mathbf{v}^{(\bar{x})})}$$

- CBOW:
 - Let $\mathbf{w}^{(C_t)}$ be shorthand for $\sum_{x_{t'} \in C_t} \mathbf{w}^{(x_{t'})}$. Then:

$$P(x_t|C_t) = \text{softmax}(\mathbf{V}\mathbf{w}^{(C_t)})_{\mathcal{I}_V(x_t)} = \frac{\exp(\mathbf{w}^{(C_t)T} \mathbf{v}^{(x_t)})}{\sum_{\bar{x} \in V} \exp(\mathbf{w}^{(C_t)T} \mathbf{v}^{(\bar{x})})}$$

Outcome

- $w^{(*)}$ and $v^{(*)}$ vectors express syntagmatic relations (e.g., “car” and “drive”).
- $w^{(*)}$ vectors express paradigmatic relations (e.g., “car” and “bus”)
- Usually, $w^{(*)}$ vectors are used for downstream applications



Naive softmax model: Complexity

- **Question:** How many dot products do we need in order to predict a single probability $P(n|m)$?
 - ▶ For both CBOW and skipgram, the softmax is defined over $|V|$ dot products.
 - ▶ So when V is big (which it usually is), then prediction (and training) is slow.
- Solution: Replace naive softmax with hierarchical softmax or negative sampling

Outline

1 Motivation for word embeddings

2 Word2Vec

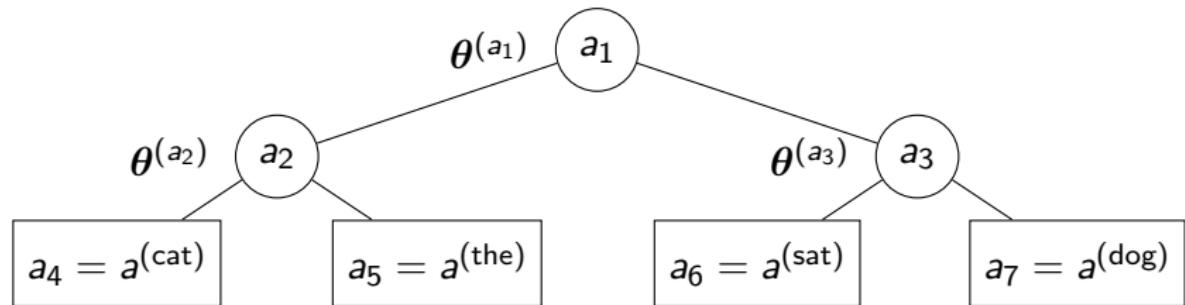
- Skipgram task
- CBOW task
- Naive softmax model
- **Hierarchical softmax model**

- Negative sampling model
- FastText

3 Applications of pretrained word embeddings

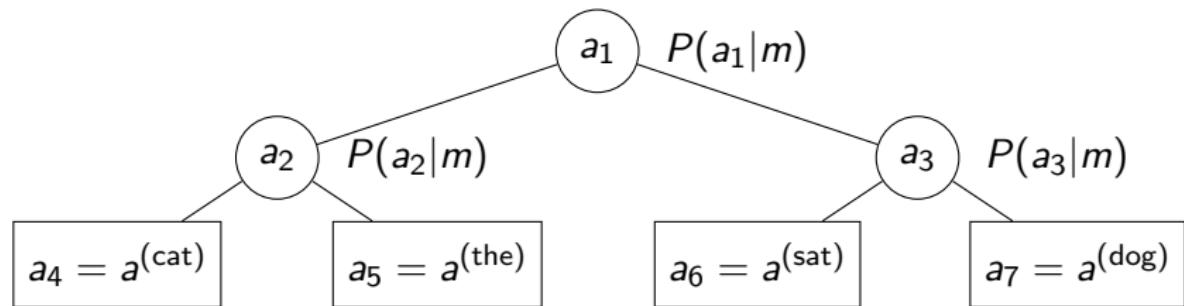
- Initialization of word embedding layer
- Word analogies
- Cross-lingual embedding spaces

Hierarchical softmax model



- $w^{(*)}$ vectors are defined like before
- $v^{(*)}$ vectors are replaced by a binary tree with nodes $A = \{a_1, a_2 \dots\}$
- Let $A' \subset A$ be the set of leaf nodes (rectangles). Every $a' \in A'$ corresponds to a unique word in V .
- Every non-leaf node (circles) has a parameter vector $\theta^{(a)} \in \mathbb{R}^D$.
- We denote the parent of some node a as $q(a)$, its left child as $l(a)$ and its right child as $r(a)$, e.g., $a^{(2)} = l(a^{(1)}) = q(a^{(4)}) = q(a^{(5)})$

Hierarchical softmax model



$$P(\text{cat}|m) = P(a^{(\text{cat})}|m) \quad P(\text{the}|m) = P(a^{(\text{the})}|m) \quad P(\text{sat}|m) = P(a^{(\text{sat})}|m) \quad P(\text{dog}|m) = P(a^{(\text{dog})}|m)$$

- Let m be the thing that we are conditioning on, and let $\mathbf{w}^{(m)}$ be its vector.
 - For skipgram, m is the center word x_t , and $\mathbf{w}^{(m)} = \mathbf{w}^{(x_t)}$
 - For CBOW, m is the context C_t , and $\mathbf{w}^{(m)} = \mathbf{w}^{(C_t)} = \sum_{x_{t'} \in C_t} \mathbf{w}^{(x_{t'})}$
- Every node has a conditional probability $P(a|m)$
- Let n be the word that we are trying to predict. Then we need to calculate the probability of the corresponding leaf node, given m :
$$P(n|m) = P(a^{(n)}|m)$$

Hierarchical softmax model

- The conditional probability of the root node is always 1.
- The conditional probability of some node $a \in A$ is defined through recursion:

$$P(a|m) = P(q(a)|m)P(a|q(a), m)$$

- The probability of selecting the left child:

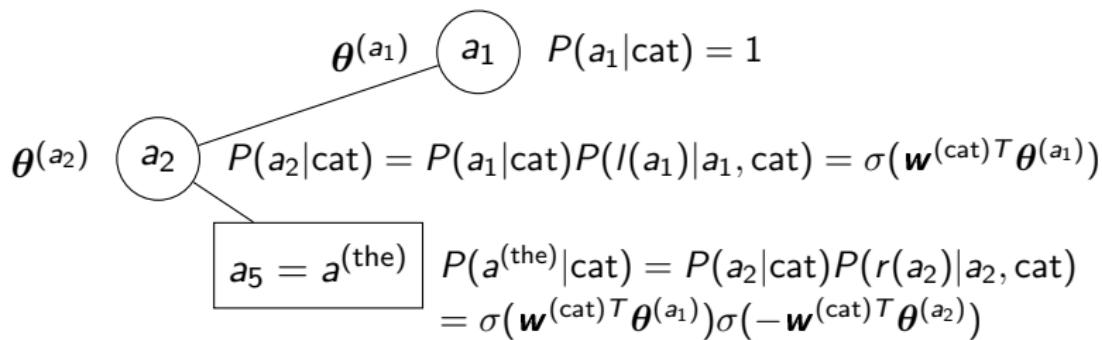
$$P(l(a)|a, m) = \sigma(\mathbf{w}^{(m)T}\boldsymbol{\theta}^{(a)})$$

- The probability of selecting the right child:

$$P(r(a)|a, m) = 1 - P(l(a)|a, m) = \sigma(-\mathbf{w}^{(m)T}\boldsymbol{\theta}^{(a)})$$

Hierarchical softmax model

Example: Let's calculate $P(\text{the}|\text{cat})$



Hierarchical softmax model: Complexity

- **Question:** How many dot products do we need in order to compute $P(n|m)$ (assuming that the tree is balanced)? How does this compare to the naive softmax?
 - ▶ Depth of the tree, i.e., $\log_2(|A'|) = \log_2(|V|)$
 - ▶ $\log_2(|V|) \ll |V|$ (e.g., $\log_2(1000000) \approx 20$)

- **Question:** Show that $\sum_{n \in V} P(n|m) = 1$.
- There is a one-to-one correspondence between V and the set of leaf nodes A' , so we must show that $\sum_{a' \in A'} P(a'|m) = 1$
- Imagine re-building the tree top-down. Let A'_j be the set of leaf nodes at step j , and let $A'_0 = \{a_1\}$. Let J be the number of steps required to finish the tree, i.e., $A' = A'_J$.
- Building procedure:
 - ▶ For every step $1 \leq j \leq J$, we select a leaf node $a'_j \in A'_{j-1}$ and split it into two new leaf nodes $I(a'_j), r(a'_j)$. Of course, a'_j stops being a leaf node at this point.
 - ▶ Formally: $A'_j = A'_{j-1} \cup \{I(a'_j), r(a'_j)\} \setminus \{a'_j\}$

- Base case:

$$\sum_{a' \in A'_0} P(a'|m) = P(a_1|m) = 1$$

- Induction step: We must show that:

$$\sum_{a' \in A'_{j-1}} P(a'|m) = 1 \implies \sum_{a' \in A'_j} P(a'|m) = 1$$

- Let's try:

$$\begin{aligned} \sum_{a' \in A'_j} P(a'|m) &= \sum_{a' \in A'_{j-1} \cup \{l(a'_j), r(a'_j)\} \setminus \{a'_j\}} P(a'|m) \\ &= \left(\sum_{a' \in A'_{j-1}} P(a'|m) \right) + P(l(a'_j)|m) + P(r(a'_j)|m) - P(a'_j|m) \end{aligned}$$

- If we could show that $P(l(a'_j)|m) + P(r(a'_j)|m) - P(a'_j|m) = 0$, it would follow that $\sum_{a' \in A'_j} P(a'|m) = \sum_{a' \in A'_{j-1}} P(a'|m)$. This would fulfill the induction step.

- Recall from our model definition:

$$P(a|m) = P(q(a)|m)P(a|q(a), m)$$
$$P(r(a)|a, m) = 1 - P(I(a)|a, m)$$

- Also, keep in mind that $q(I(a)) = q(r(a)) = a$.
- Then:

$$P(I(a)|a, m) + P(r(a)|a, m) = 1$$
$$\iff P(a|m)P(I(a)|a, m) + P(a|m)P(r(a)|a, m) = P(a|m)$$
$$\iff P(q(I(a))|m)P(I(a)|a, m) + P(q(r(a))|m)P(r(a)|a, m) = P(a|m)$$
$$\iff P(I(a)|m) + P(r(a)|m) = P(a|m)$$
$$\iff P(I(a)|m) + P(r(a)|m) - P(a|m) = 0$$

Questions?

Take a moment to write down any questions you have for the QA session!

Outline

1 Motivation for word embeddings

2 Word2Vec

- Skipgram task
- CBOW task
- Naive softmax model
- Hierarchical softmax model
- **Negative sampling model**

• FastText

3 Applications of pretrained word embeddings

- Initialization of word embedding layer
- Word analogies
- Cross-lingual embedding spaces

Negative sampling model

- Another trick: **negative sampling**
- Instead of predicting a probability distribution over whole vocabulary, predict binary probabilities for a small number of word pairs.
- This is not a language model anymore...
- ... but since we only care about the word vectors, and not the skip gram/CBOW predictions, that's not really a problem.

Negative sampling model

- Let B be a hyperparameter.
- For every positive (observed) center-context (skipgram) or context-center (CBOW) pair (m, n) , create B negative pairs $((m, n'_1) \dots (m, n'_B))$, where $n'_b \sim V$ is a random word
 - Detail: We sample as a function of word frequency, with undersampling for very frequent words
- Binary classification: predict whether a pair is positive or negative.
- As negative log likelihood:

$$\mathcal{J} = \sum_{m,n} \left(-\log(P(\text{pos}|m, n)) - \sum_{b=1}^B \log(P(\text{neg}|m, n'_b)) \right)$$

- Can be rewritten as:

$$\mathcal{J} = \sum_{m,n,y} \left(-y \log(P(\text{pos}|m, n)) - (1-y) \log(P(\text{neg}|m, n)) \right)$$

- where $y = 1$ if (m, n) is a positive pair and $y = 0$ if (m, n) is a negative pair.

Negative sampling model

- Parameters:

- ▶ $\mathbf{v}^{(*)} \in \mathbb{R}^D$
- ▶ $\mathbf{w}^{(*)} \in \mathbb{R}^D$

$$P(\text{pos}|m, n) = \sigma(\mathbf{w}^{(m)T} \mathbf{v}^{(n)})$$

$$P(\text{neg}|m, n) = 1 - P(\text{pos}|m, n) = 1 - \sigma(\mathbf{w}^{(m)T} \mathbf{v}^{(n)}) = \sigma(-\mathbf{w}^{(m)T} \mathbf{v}^{(n)})$$

- So the loss function becomes:

$$\mathcal{J} = \sum_{m,n,y} \left(-y \log(\sigma(\mathbf{w}^{(m)T} \mathbf{v}^{(n)})) - (1-y) \log(\sigma(-\mathbf{w}^{(m)T} \mathbf{v}^{(n)})) \right)$$

- On the next slides, we will show how to perform a gradient update for one tuple (m, n, y) .

Negative sampling model: Gradients

$$J = -y \log(\sigma(\mathbf{w}^{(m)T} \mathbf{v}^{(n)})) - (1-y) \log(\sigma(-\mathbf{w}^{(m)T} \mathbf{v}^{(n)}))$$

$$\begin{aligned}\nabla_{\mathbf{w}^{(m)}} J &= -y \frac{1}{\sigma(\mathbf{w}^{(m)T} \mathbf{v}^{(n)})} \sigma(\mathbf{w}^{(m)T} \mathbf{v}^{(n)}) \sigma(-\mathbf{w}^{(m)T} \mathbf{v}^{(n)}) \mathbf{v}^{(n)} \\ &\quad - (1-y) \frac{1}{\sigma(-\mathbf{w}^{(m)T} \mathbf{v}^{(n)})} \sigma(-\mathbf{w}^{(m)T} \mathbf{v}^{(n)}) \sigma(\mathbf{w}^{(m)T} \mathbf{v}^{(n)}) (-1) \mathbf{v}^{(n)} \\ &= -y(1 - \sigma(\mathbf{w}^{(m)T} \mathbf{v}^{(n)})) \mathbf{v}^{(n)} + (1-y) \sigma(\mathbf{w}^{(m)T} \mathbf{v}^{(n)}) \mathbf{v}^{(n)} \\ &= -y \mathbf{v}^{(n)} + y \sigma(\mathbf{w}^{(m)T} \mathbf{v}^{(n)}) \mathbf{v}^{(n)} + \sigma(\mathbf{w}^{(m)T} \mathbf{v}^{(n)}) \mathbf{v}^{(n)} - y \sigma(\mathbf{w}^{(m)T} \mathbf{v}^{(n)}) \mathbf{v}^{(n)} \\ &= \sigma(\mathbf{w}^{(m)T} \mathbf{v}^{(n)}) \mathbf{v}^{(n)} - y \mathbf{v}^{(n)} = (\sigma(\mathbf{w}^{(m)T} \mathbf{v}^{(n)}) - y) \mathbf{v}^{(n)} \\ &= (P(\text{pos}|m, n) - y) \mathbf{v}^{(n)}\end{aligned}$$

Similarly:

$$\nabla_{\mathbf{v}^{(n)}} J = (P(\text{pos}|n, m) - y) \mathbf{w}^{(m)}$$

Derivation cheat sheet:

$$\frac{dcx}{dx} = c \quad \frac{d\log(x)}{dx} = \frac{1}{x} \quad \frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x)) = \sigma(x)\sigma(-x) \quad \frac{\partial x^T y}{\partial x_i} = y_i$$

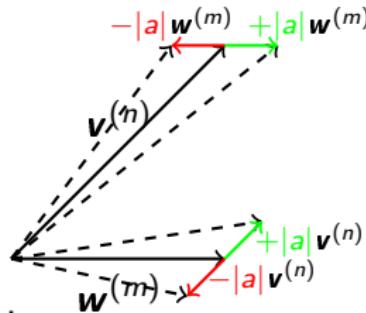
Negative sampling model: Gradient update

$$\mathbf{w}_{updated}^{(m)} = \mathbf{w}^{(m)} - \eta \nabla_{\mathbf{w}^{(m)}} J = \mathbf{w}^{(m)} + (-1)\eta(P(\text{pos}|m, n) - y)\mathbf{v}^{(n)}$$

$$\mathbf{v}_{updated}^{(n)} = \mathbf{v}^{(n)} - \eta \nabla_{\mathbf{v}^{(n)}} J = \mathbf{v}^{(n)} + (-1)\eta(P(\text{pos}|m, n) - y)\mathbf{w}^{(m)}$$

Intuition:

- Let $a = (-1)\eta(P(\text{pos}|m, n) - y)$
- True word pair: $y = 1 \wedge \eta > 0 \implies a \geq 0 \implies a = +|a|$
 - $|a|\mathbf{v}^{(n)}$ is added to $\mathbf{w}^{(m)}$ and vice versa \rightarrow vectors become more similar.
- Random word pair: $y = 0 \wedge \eta > 0 \implies a \leq 0 \implies a = -|a|$
 - $|a|\mathbf{v}^{(n)}$ is subtracted from $\mathbf{w}^{(m)}$ and vice versa \rightarrow vectors become less similar.



Negative sampling model: Complexity

- **Question:** How many dot products do we need to calculate per positive pair (m, n) ? How does this compare to the naive and hierarchical softmax?
 - ▶ $B + 1$ (B negative pairs and 1 positive pair)
 - ▶ $B + 1 \approx \log_2(|V|) \ll |V|$ (for $B = 20, |V| = 1000000$)

Questions?

Take a moment to write down any questions you have for the QA session!

Outline

1 Motivation for word embeddings

2 Word2Vec

- Skipgram task
- CBOW task
- Naive softmax model
- Hierarchical softmax model
- Negative sampling model
- **FastText**

3 Applications of pretrained word embeddings

- Initialization of word embedding layer
- Word analogies
- Cross-lingual embedding spaces

FastText

- Even if we train Word2Vec on a very large corpus, we will still encounter unknown words at test time
- Subwords can often help us:
- $w^{(\text{remuneration})}$ should be similar to
 - ▶ $w^{(\text{remunerate})}$ (same stem)
 - ▶ $w^{(\text{iteration})}$, $w^{(\text{consideration})}$... (same suffix → may be same part of speech)

FastText

- Extension of Word2Vec by Bojanowski et al. (2017): Enriching Word Vectors with Subword Information.
- Model:
 - ▶ No $\mathbf{w}^{(*)}$ vectors
 - ▶ Instead: $\mathbf{u}^{(*)} \in \mathbb{R}^D$ vectors, one vector per word in $x \in V$, and one for every character n-gram (typically 3- to 6-grams)
 - ▶ Function that enumerates the character n-grams of a word:

$$\mathcal{S}(\text{remuneration}) = \{\#\text{re}, \text{rem}, \#\text{rem}, \dots, \text{ration}, \text{ation}\$\}$$

$$\mathbf{w}^{(m)} = \begin{cases} \frac{1}{|\mathcal{S}(m)|+1} \left[\mathbf{u}^{(m)} + \sum_{m' \in \mathcal{S}(m)} \mathbf{u}^{(m')} \right] & \text{if } m \in V \\ \frac{1}{|\mathcal{S}(m)|} \sum_{m' \in \mathcal{S}(m)} \mathbf{u}^{(m')} & \text{otherwise} \end{cases}$$

- Plug new definition of $\mathbf{w}^{(m)}$ into one of the Word2Vec models.
- During training, gradients of loss w.r.t. $\mathbf{w}^{(m)}$ are evenly distributed to $\mathbf{u}^{(m)}$ and its subword vectors $\mathbf{u}^{(m')}$

Questions?

Take a moment to write down any questions you have for the QA session!

Outline

- ① Motivation for word embeddings
- ② Word2Vec
 - Skipgram task
 - CBOW task
 - Naive softmax model
 - Hierarchical softmax model
 - Negative sampling model
 - FastText
- ③ Applications of pretrained word embeddings
 - Initialization of word embedding layer
 - Word analogies
 - Cross-lingual embedding spaces

Outline

1 Motivation for word embeddings

2 Word2Vec

- Skipgram task
- CBOW task
- Naive softmax model
- Hierarchical softmax model
- Negative sampling model
- FastText

3 Applications of pretrained word embeddings

- Initialization of word embedding layer
- Word analogies
- Cross-lingual embedding spaces

Initialization of word embedding layers

- Use pretrained $w^{(*)}$ vectors to initialize the word embedding layer of a CNN, RNN ... (see first slides)
- Design choice: Fine-tune embeddings on task or freeze them?
 - ▶ Pro fine-tuning: Can learn/strengthen features that are important for the task
 - ▶ Pro freezing: We might overfit on training set and mess up the relationships between seen and unseen words
 - ▶ Both options are popular

Model		MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	(randomly initialized)	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	(pretrained+frozen)	81.0	45.5	86.8	93.0	92.8	84.7	89.6
CNN-non-static	(pretrained+fine-tuned)	81.5	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	(combination)	81.1	47.4	88.1	93.2	92.2	85.0	89.4

Table from Kim 2014: Convolutional Neural Networks for Sentence Classification.

Outline

- ① Motivation for word embeddings
- ② Word2Vec
 - Skipgram task
 - CBOW task
 - Naive softmax model
 - Hierarchical softmax model
 - Negative sampling model
 - FastText
- ③ Applications of pretrained word embeddings
 - Initialization of word embedding layer
 - Word analogies
 - Cross-lingual embedding spaces

Word analogies

$$\mathbf{w}^{(a)} - \mathbf{w}^{(b)} \approx \mathbf{w}^{(c)} - \mathbf{w}^{(d)}$$

$$\hat{d} = \operatorname{argmax}_d \sim(\mathbf{w}^{(d)}, \mathbf{w}^{(c)} - \mathbf{w}^{(a)} + \mathbf{w}^{(b)})$$

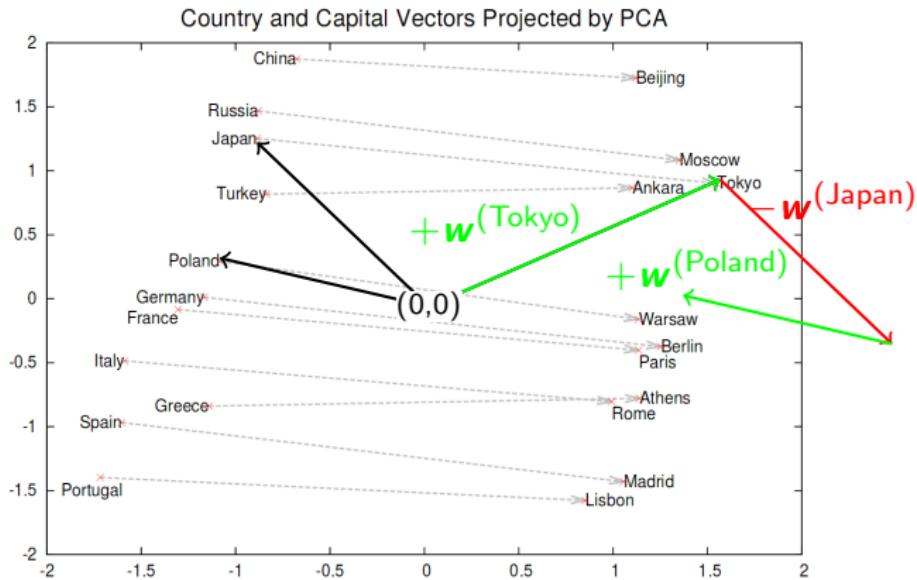


Figure from Mikolov et al. (2013): Distributed Representations of Words and Phrases and their Compositionality

Word analogies

country-capital

$$\mathbf{w}^{(\text{Tokio})} - \mathbf{w}^{(\text{Japan})} + \mathbf{w}^{(\text{Poland})} \approx \mathbf{w}^{(\text{Warsaw})}$$

opposite

$$\mathbf{w}^{(\text{unacceptable})} - \mathbf{w}^{(\text{acceptable})} + \mathbf{w}^{(\text{logical})} \approx \mathbf{w}^{(\text{illogical})}$$

Nationality-adjective

$$\mathbf{w}^{(\text{Australian})} - \mathbf{w}^{(\text{Australia})} + \mathbf{w}^{(\text{Switzerland})} \approx \mathbf{w}^{(\text{Swiss})}$$

Outline

- ① Motivation for word embeddings
- ② Word2Vec
 - Skipgram task
 - CBOW task
 - Naive softmax model
 - Hierarchical softmax model
 - Negative sampling model
 - FastText
- ③ Applications of pretrained word embeddings
 - Initialization of word embedding layer
 - Word analogies
 - Cross-lingual embedding spaces

Cross-lingual Embedding Spaces

- Monolingual embedding spaces of two languages: \mathbf{W}_{L1} , \mathbf{W}_{L2}
- Dictionary of a few translations: $D = ((\text{cat}, \text{katze}), (\text{dog}, \text{hund}), \dots)$
- Learn function f with parameters θ that minimizes:

$$\operatorname{argmin}_{\theta} \sum_{i,j \in D} \|f(\mathbf{w}_{L1}^{(i)}; \theta) - \mathbf{w}_{L2}^{(j)}\|_2^2$$

- ▶ e.g., linear transformation with parameter matrix \mathbf{A} :
$$f(\mathbf{w}_{L1}^{(i)}; \mathbf{A}) = \mathbf{A}\mathbf{w}_{L1}^{(i)}$$
- Given a word k in L1 with unknown translation, translate as:

$$\operatorname{argmax}_{\bar{j}} \sim(\mathbf{w}_{L2}^{(\bar{j})}, f(\mathbf{w}_{L1}^{(k)}))$$

Cross-lingual Embedding Spaces

Spanish word	Computed English Translations	Dictionary Entry
emociones	emotions emotion feelings	emotions
protegida	wetland undevlopable protected	protected
imperio	dictatorship imperialism tyranny	empire
determinante	crucial key important	determinant
preparada	prepared ready prepare	prepared
millas	kilometers kilometres miles	miles
hablamos	talking talked talk	talk

Table from Mikolov et al. 2013: Exploiting Similarities among Languages for Machine Translation

Questions?

Take a moment to write down any questions you have for the QA session!

Introduction to numpy

Deep Learning for NLP: Lecture 5(b)

Centrum für Informations- und Sprachverarbeitung
Ludwig-Maximilians-Universität München
poerner@cis.uni-muenchen.de

December 02, 2020

What is numpy?

- Acronym for “Numeric Python”
- Open source library for python for efficient computation of arrays (vectors, matrices, higher-order arrays)
- Used by many scientific computing and machine learning packages, such as scipy or sklearn.
- Not a specialized library for deep learning (unlike tensorflow or pytorch). But the APIs of deep learning libraries are often similar to numpy.
- Useful for data handling and preprocessing.
- www.numpy.org

Core python lists vs. numpy

- Using python nested lists as “arrays”:
 - + can contain any type of object, or mix of data types (e.g., ints and floats)
 - + dynamic sizing, cheap insertion/deletion
 - array operations are not very straightforward (e.g., taking the “dot product” of two lists)
 - no built-in checks for shapes and data types
 - operations are generally not very efficient
- Numpy arrays:
 - no mixing of data types
 - data type and shape of array must be declared beforehand
 - + commonly used operations are functions or overloaded operators
 - + built-in checks for shapes and types
 - + array operations are efficient

A simple numpy example

- Declaring a “vector” \mathbf{y} and a “matrix” \mathbf{X} as nested python lists:

```
>>> l_y = [25.0, 33.3]
>>> l_X = [[6.3, 5.6], [4.5, 4.4], [8.0, 1.0]]
>>> l_y
[25.0, 33.3]
```

- Calculating $\mathbf{X}\mathbf{y}$ with core python is possible, but annoying:

```
>>> [sum([a*b for a,b in zip(l_y, row)]) for row in l_X]
[343.9799999999996, 259.02, 233.3]
```

- No shape check: If $\mathbf{l}_\mathbf{y}$ had more dimensions than the rows, extra elements would just be ignored by the `zip` function
- To convert the lists into numpy arrays, use `np.array`

```
>>> import numpy as np # Convention: use np for numpy
>>> n_y = np.array(l_y)
>>> n_X = np.array(l_X)
>>> n_y
array([25., 33.3])
```

- Then you can simply calculate $\mathbf{X}\mathbf{y}$ as:

```
>>> n_X.dot(n_Y)
array([343.98, 259.02, 233.3])
```

Numpy arrays: dtypes

- An array has exactly one datatype (dtype), e.g., np.int32, np.float32, np.float64, np.bool...
- When you create an array from a nested list, numpy will try to infer the dtype. If a list mixes several dtypes, it will resolve it by casting (e.g., int to float):

```
>>> np.array([[1, 2, 3], [10, 2, 1]]) # ints only  
array([[ 1,  2,  3],  
       [10,  2,  1]])  
>>> np.array([[1.0, 2.0, 3.0], [10, 2, 1]]) # ints and floats  
array([[ 1.,  2.,  3.],  
       [10.,  2.,  1.]])
```

- Or you can specify the dtype explicitly:..

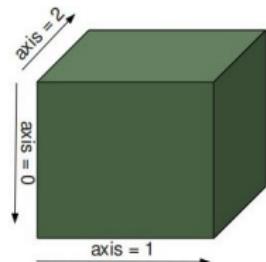
```
>>> np.array([0,0,1], dtype=np.bool)  
array([False, False, True])  
>>> np.array([0,0,1], dtype=np.float64)  
array([0., 0., 1.])
```

Numpy arrays: shapes

- Every numpy array has a shape (tuple of integers)
- The length of the shape denotes the number of axes:
 - ▶ 0 = scalar, 1 = vector, 2 = matrix, 3 or more = higher-dimensional array (tensor)

axis = 1		
0 = size		
67	63	87
77	69	59
85	87	99
79	72	71
63	89	93
68	92	78

← matrix 3D tensor →



- The integers in the tuple denote the number of dimensions per axis
- When you create an array from a nested python list, numpy tries to infer the shape:

```
>>> np.array([[1.0, 2.0, 3.0], [10.0, 2.0, 1.0]]).shape  
(2, 3)
```

- Careful: If the nested list is not a valid shape (e.g., different row lengths), you get an “object” array instead of a numerical array:

```
>>> np.array([[1.0, 2.0, 3.0], [10.0, 2.0]])  
array([list([1.0, 2.0, 3.0]), list([10.0, 2.0])], dtype=object)
```

Numpy arrays: shapes

- When declaring an array from scratch, you usually specify its shape:

```
>>> np.zeros(shape=()) # scalar  
0.0  
>>> np.zeros(shape=(2,)) # vector  
array([0., 0.])  
>>> np.zeros(shape=(2,3)) # matrix  
array([[0., 0., 0.],  
       [0., 0., 0.]])  
>>> np.zeros(shape=(2,3,4)) # tensor (3 axes)  
array([[[0., 0., 0., 0.],  
        [0., 0., 0., 0.],  
        [0., 0., 0., 0.]],  
      [[0., 0., 0., 0.],  
        [0., 0., 0., 0.],  
        [0., 0., 0., 0.]]])
```

- Careful: An axis with one dimension is not the same as no axis:

```
>>> np.zeros(shape=(1,1)) # matrix  
array([[0.]])  
>>> np.zeros(shape=(1,)) # vector  
array([0.])  
>>> np.zeros(shape=()) # scalar  
array(0.)
```

Exercise

- Given a shape and dtype, think of an example of what the array might represent.

shape	dtype	example
(100,)	bool	yes/no votes by 100 voters
(32, 2,)	int	coordinates of 32 chess pieces on a chess board (8x8 grid)
(1080, 720,)	float	picture with height 1080, width 720, grayscale (0.0-1.0)
(240, 1080, 720, 3)	float	movie with 240 frames, height 1080, width 720, three channels per pixel (red, green, blue)

Some useful numpy functionalities

- This is not meant to be an exhaustive guide!
- For more info, check out
<https://numpy.org/devdocs/user/quickstart.html>
- Or look on stack overflow!

Zeros and ones

- To create an array of all-zeros/all-ones:

```
>>> zeros = np.zeros(shape=(2,3))  
>>> zeros  
array([[0., 0., 0.],  
       [0., 0., 0.]])  
>>> ones = np.ones(shape=(2,3))
```

- You can also create arrays that have the same shape and dtype as a different array, but are filled with zeros or ones:

```
>>> np.zeros_like(ones)  
array([[0., 0., 0.],  
       [0., 0., 0.]])  
>>> np.ones_like(zeros)
```

- To create an identity matrix of shape NxN:

```
>>> np.eye(N=3)  
array([[1., 0., 0.],  
       [0., 1., 0.],  
       [0., 0., 1.]])
```

- Padding with zeros:

```
>>> a = np.ones((3,))  
>>> np.pad(a, (2,))  
array([0., 0., 1., 1., 1., 0., 0.])
```

Ranges in numpy

- Numpy has an equivalent of the core python range function. The default dtype is np.int64.

```
>>> np.arange(start=2, stop=10, step=2)
array([2, 4, 6, 8])
```

- To create a vector of N evenly spaced floats between two points:

```
>>> np.linspace(start=2, stop=10, num=6)
array([ 2. ,  3.6,  5.2,  6.8,  8.4, 10. ])
```

Changing the shape

- Changing shape value of an existing array:

```
>>> a = np.arange(12)
>>> a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
>>> a.shape = (2,6)
>>> a
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]])
```

- reshape creates new array:

```
>>> b = a.reshape((3,4))
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

- Obviously, the product of the dimensions must match number of elements in the array.
- If a dimension is specified as -1, numpy tries to derive it from the other dimensions:

```
>>> b.reshape((2,2,-1)).shape
(2, 2, 3)
```

Transposing an array

- `a.transpose(...)` takes a tuple of indices, indicating which axis of the old (input) array is used for each axis of the new (output) array.
- Note: numpy counts axes starting from 0

```
>>> a = np.zeros((2,3,4))
>>> a.shape
(2, 3, 4)
>>> b=a.transpose((2,0,1))
>>> b.shape
(4, 2, 3)
```

- Interpretation: Axis 0 of *a* becomes axis 1 of *b*, axis 1 of *a* becomes axis 2 of *b*, and axis 2 of *a* becomes axis 0 of *b*.
- To reverse the order of the axes, you can also use:

```
>>> a = np.zeros((2,3,4))
>>> b = a.T
>>> b.shape
(4, 3, 2)
```

Basic Operations

- Usually, operators on arrays apply *elementwise*:

```
>>> a = np.array([20, 30, 40, 50])
>>> b = np.array([0, 1, 2, 3])
>>> a-b
array([20, 29, 38, 47])
```

- In particular, the *elementwise multiplication* ...

```
>>> a * b
array([ 20,  60, 120, 200])
```

- ... should not be confused with the *dot product*:

```
>>> a.dot(b)
400
```

- Numpy implements many standard unary (elementwise) functions:

```
>>> np.exp(b)
>>> np.sqrt(b)
>>> np.log(b)
```

Reduction operators over axes

- Some functions reduce the array, by taking the sum (or mean, maximum, ...) over a specified axis.
- If no axis is specified, the entire array is reduced to a scalar.

```
>>> b = np.arange(12).reshape(3,4)
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

>>> b.sum(axis=0)
array([12, 15, 18, 21])
>>> b.sum(axis=1)
array([ 6, 22, 38])
>>> b.sum() # sum over all axes
66
>>> b.mean(axis=0)
>>> b.max(axis=1)
>>> b.min()
```

- To keep the reduced axis as a 1D axis:

```
>>> b.sum(axis=0, keepdims=True)
array([[12, 15, 18, 21]])
```

Indexing elements

- Indexing single elements:

```
>>> b = np.array([[ [111, 112], [121, 122]],
   ...                 [[211, 212], [221, 222]],
   ...                 [[311, 312], [321, 322]]])
>>> b[2,1,0] # this would also work: b[2][1][0]
321
>>> b[-1,-2,-1] # indexing from the end
312
```

- Indexing a sub-array:

```
>>> b[1]
array([[211, 212],
       [221, 222]])
```

- Assignment by index:

```
>>> b[2,1,0] = -100
>>> b
np.array([[ [111, 112],
   ...                 [121, 122]],
   ...                 [[211, 212],
   ...                 [221, 222]],
   ...                 [[311, 312],
   ...                 [-100, 322]]])
```

Indexing with arrays

- You can index several elements at once with an array / list of indices:

```
>>> a = np.arange(12)**2
>>> i = np.array( [ 1,1,3,8,5 ] ) # or: i = [ 1,1,3,8,5 ]
>>> a[i]
array([ 1,   1,   9,  64,  25])
```

- To index a matrix or higher, use a tuple of index arrays (one array per axis):

```
>>> a.shape = (3,4)
>>> a
array([[ 0,   1,   4,   9],
       [ 16,  25,  36,  49],
       [ 64,  81, 100, 121]])
>>> i = (np.array([0,0,0,2,1]), np.array([1,1,3,0,1]))
>>> a[i]
array([ 1,   1,   9,  64,  25])
```

Boolean indexing

- Boolean indexing is done with a boolean array of the *same shape* as the indexed array.

```
>>> a = np.arange(12).reshape((3,4))
>>> a
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> b = a > 4
>>> b
array([[False, False, False, False],
       [False,  True,  True,  True],
       [ True,  True,  True,  True]])
>>> a[b]
array([ 5,  6,  7,  8,  9, 10, 11])
```

Slicing

- Syntax for slicing lists and tuples (start:stop:step) can be applied to multiple axes of arrays.
- Example with one axis:

```
>>> a = np.arange(12)
>>> a[3:8:2]
array([3, 5, 7])
>>> a[:4]
array([0, 1, 2, 3])
>>> a[4:]
array([4, 5, 6, 7, 8, 9, 10, 11])
>>> a[:] # 'dummy' slice
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])
```

- Example with several axes:

```
>>> b = a.reshape((4,3))
>>> b
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
>>> b[:,2]
array([[ 2],
       [ 5],
       [ 8],
       [11]])
```

Slicing: Caveat

- Slicing only creates a new **view**: the underlying data is shared with the original array.

```
>>> a = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> b = a[2:6]
>>> b[0] = -10000
>>> b[1] = -10000
>>> a
array([ 0,  1, -10000, -10000,  4,  5,  6,  7,  8,  9])
```

- If you want a copy, use:

```
>>> a = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> b = a[2:6].copy()
>>> b[0] = -10000
>>> b[1] = -10000
>>> a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9])
```

Creating random arrays

- Array of floats uniformly drawn from the interval $[0, 1)$:

```
>>> np.random.rand(shape=(2,3))
array([[ 0.53604809,  0.54046081,  0.84399025],
       [ 0.59992296,  0.51895053,  0.09988041]])
```

- Generate floats drawn from standard normal distribution $\mathcal{N}(0, 1)$:

```
>>> np.random.randn(shape=(2,3))
array([-1.28520219, -1.02882158, -0.20196267],
      [ 0.48258382, -0.2077209 , -2.03846176]])
```

- For reproducibility, initialize the random seed at the beginning of your script:

```
>>> np.random.seed(0)
```

- Otherwise, it will be initialized differently at every run (from system clock), leading to different results each time.

Iterating over arrays

- You can iterate over arrays the way you would iterate over a nested list:

```
>>> b = np.arange(9).reshape(3,3)
>>> for row in b:
...     print(row)
...     for elem in row:
...         print(elem)
```

```
[0 1 2]
```

```
0
```

```
1
```

```
2
```

```
[3 4 5]
```

```
3
```

```
4
```

```
5
```

```
[6 7 8]
```

```
6
```

```
7
```

```
8
```

- For efficiency, use array operations instead of iterating, whenever you can.

Finding, sorting and reversing

- To find the indices of an array where a condition holds true:

```
>>> a = np.arange(10, 20).reshape(2,5)
>>> a
array([[10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
>>> indices = np.where(a % 2 == 0)
>>> indices
(array([0, 0, 0, 1, 1]), array([0, 2, 4, 1, 3]))
>>> a[indices]
array([10, 12, 14, 16, 18])
```

- You can sort arrays over a specified axis:

```
>>> a = np.random.random((3,3))
>>> a.sort(axis=1) # sort inside rows
>>> a
array([[0.60930752, 0.85081993, 0.95468601],
       [0.27631831, 0.57972567, 0.88009649],
       [0.28681042, 0.65898285, 0.69914403]])
```

- ... and reverse the order of axes:

```
>>> a[::-1] # reverse the order of the rows
array([[0.28681042, 0.65898285, 0.69914403],
       [0.27631831, 0.57972567, 0.88009649],
       [0.60930752, 0.85081993, 0.95468601]])
```

Concatenating arrays

- Two or more arrays can be concatenated:

```
>>> a = np.array ([[1,2],[3,4]])
>>> b = np.array ([[11,22],[33,44]])
>>> c = np.array ([[ -1, -2]])
>>> np.concatenate ((a,b,c), axis=0)
array ([[ 1,  2],
       [ 3,  4],
       [11, 22],
       [33, 44],
       [-1, -2]])
>>> np.concatenate ((a,b), axis=1)
array ([[ 1,  2, 11, 22],
       [ 3,  4, 33, 44]])
```

- Note that concatenation requires that (a) all arrays have the same number of axes and (b) all axes (except the one used for concatenation) have the same dimensionality.

```
>>> np.concatenate ((a,b,c), axis=1)
ValueError: all the input array dimensions for the concatenation
```

Stacking arrays

- Stacking arrays means treating them as sub-arrays of a new array. This requires that all stacked arrays have the same shape. With stacking, a new axis is introduced.

```
>>> np.stack((a,b), axis=0) # add new axis before first
array([[[ 1,  2],
       [ 3,  4]],

       [[11, 22],
        [33, 44]]])
>>> np.stack((a,b), axis=-1) # add new axis after last
array([[[[ 1, 11],
          [ 2, 22]],

          [[ 3, 33],
           [ 4, 44]]]])
```

Inserting and deleting dummy axes

- For broadcasting (see next slide), it can be useful to add “dummy axes” to our arrays:

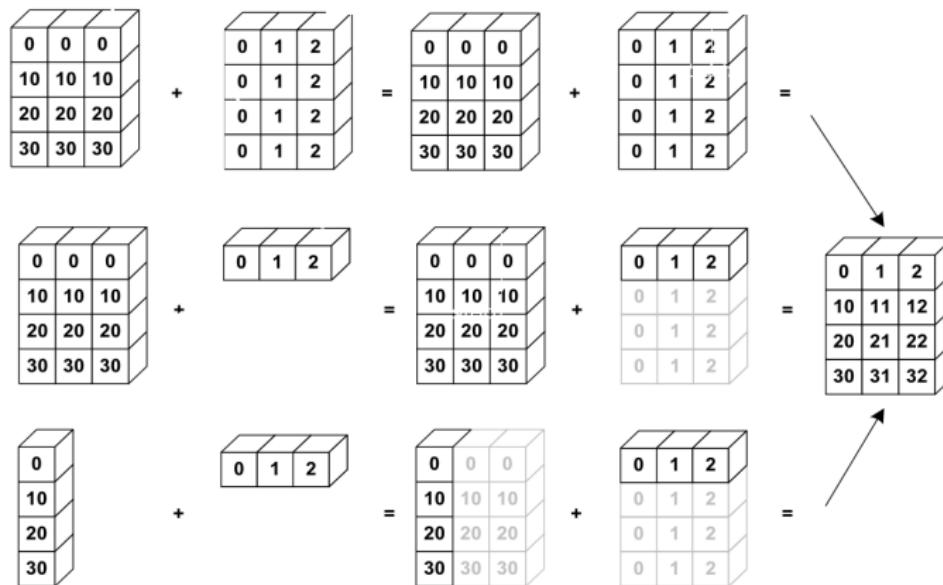
```
>>> a = np.array([5, 6, 7])
>>> a
array([5, 6, 7])
>>> a.shape
(3,)
>>> b = np.expand_dims(a, axis=0)
>>> b
array([[5, 6, 7]])
>>> b.shape
(1, 3)
>>> c = np.expand_dims(a, axis=1)
>>> c.shape
(3, 1)
```

- To delete a dummy axis:

```
>>> b.squeeze(axis=0)
array([5, 6, 7])
```

Broadcasting

Let $\mathbf{X} \in \mathbb{R}^{m \times n}$ and $\mathbf{Y} \in \mathbb{R}^{m \times 1}$. In mathematical notation, the elementwise sum $\mathbf{X} + \mathbf{Y}$ is not legal, because their shapes don't match. But in numpy, it is possible through *broadcasting*.



- This is useful for the outer vector product:

```
>>> x = np.arange(3)
>>> y = np.arange(4)*10
>>> x
array([0, 1, 2])
>>> y
array([ 0, 10, 20, 30])
>>> np.expand_dims(x, axis=0) * np.expand_dims(y, axis=1)
array([[ 0,  0,  0],
       [ 0, 10, 20],
       [ 0, 20, 40],
       [ 0, 30, 60]])
```

Convolutional Neural Networks (CNNs)

Deep Learning for NLP: Lecture 6

Centrum für Informations- und Sprachverarbeitung
Ludwig-Maximilians-Universität München
poerner@cis.uni-muenchen.de

December 09, 2020

Outline

1 Neural network architectures

2 Convolutional layers

- Example: 1-D convolution
- Backpropagation for 1-D convolution
- Some additional details
- Convolution with more than one axis
- Convolution with more than one feature
- Convolution with a filter bank
- Assumptions behind convolution

3 Pooling layers

4 CNNs in NLP

Outline

1 Neural network architectures

2 Convolutional layers

- Example: 1-D convolution
- Backpropagation for 1-D convolution
- Some additional details
- Convolution with more than one axis
- Convolution with more than one feature
- Convolution with a filter bank
- Assumptions behind convolution

3 Pooling layers

4 CNNs in NLP

What do we mean by “neural network architecture”?

- An architecture is an abstract design for a neural network
- Loosely speaking: Which nodes connect to which nodes? Which connections share parameters?
- Examples of architectures:
 - ▶ Fully Connected Neural Networks
 - ▶ Convolutional Neural Networks (CNNs)
 - ▶ Recurrent Neural Networks (RNNs)
 - ★ Subclasses: Vanilla RNNs, LSTMs, GRUs, QRNNs, ...
 - ▶ Transformers (self-attention)
 - ▶ ...
- The choice of architecture is often informed by assumptions about the data, based on our domain knowledge

Assumption of locality

- Computer vision: Pixels that make up a meaningful objects tend to be located in a coherent (“local”) area:



- Not always true for words in NLP:
 - ▶ *sie sollen sich heute bei ihm im büro vorstellen*
 - ▶ **stellen** *sie sich bitte heute bei ihm im büro vor*

Assumption of translation invariance

- The features that make up a meaningful object do not depend on that object's absolute position in the input.
- Computer vision:



- NLP:
 - ▶ **[the yellow house]** *[rest of sentence]* → noun phrase
 - ▶ *[rest of sentence]* **[the yellow house]** → still a noun phrase

Assumption of sequentiality

- NLP: Sentences should be processed left-to-right or right-to-left. This one is falling out of fashion, since people are replacing recurrent neural networks with self-attention networks.

Are these assumptions (always) true?

- Of course not.
- But they are a good way of thinking about why certain architectures are popular for certain problems.
- Also, for limiting the search space when deciding on an architecture for a given project.

Outline

1 Neural network architectures

2 Convolutional layers

- Example: 1-D convolution
- Backpropagation for 1-D convolution
- Some additional details
- Convolution with more than one axis
- Convolution with more than one feature
- Convolution with a filter bank
- Assumptions behind convolution

3 Pooling layers

4 CNNs in NLP

Convolutional layers

- Technique from Computer Vision (esp. object recognition), by LeCun et al., 1998
- Adapted for NLP (e.g., Kalchbrenner et al., 2014)
- Filter banks with trainable filters
- So what is a filter? What is a filter bank?

Example: 7 day rolling average filter



<https://www.nytimes.com/>, December 07, 2020

Example: 7 day rolling average filter

- Let $\mathbf{x} \in \mathbb{R}^J$ be a vector that represents a data series (e.g., number of positive tests per day, over J consecutive days)
- Let $\mathbf{f} \in \mathbb{R}^K; \mathbf{f} = [\frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}]^T$ be our “7 day rolling average” filter vector
- Filtering \mathbf{x} with \mathbf{f} (written $\mathbf{f} * \mathbf{x}$) yields a new vector \mathbf{h} , with:

$$h_j = \sum_{k=1}^7 f_k x_{(j+k-7)} = \frac{1}{7}(x_{j-6} + \dots + x_j)$$

- (For now, let's not worry about the edge cases where $j + k - 7 \leq 0$.)

1-D convolution

- Let's say that we want to train a linear regression model to predict some variable of interest from each datapoint.
- Since the raw data is noisy, it makes sense to smooth it with our rolling average filter first:

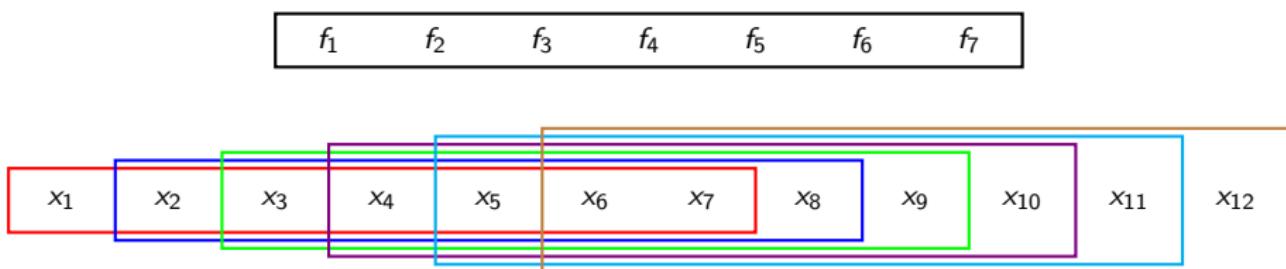
$$\hat{y}_j = w h_j + b; \quad h_j = (\mathbf{f} * \mathbf{x})_j$$

- But maybe we should weight the datapoints in our 7-day window differently? Maybe we should give a higher weight to datapoints close to the current day j ?
- We can manually engineer a filter that we think is better, for example:
 - $\mathbf{f}' = [\frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}]^T$
 - Alternative: Let the *model* choose the optimal filter parameters, based on the training data.
 - How? Gradient descent!

Backpropagation for 1-D convolution

$$\frac{\partial \text{loss}}{\partial h_7} \quad \frac{\partial \text{loss}}{\partial h_8} \quad \frac{\partial \text{loss}}{\partial h_9} \quad \frac{\partial \text{loss}}{\partial h_{10}} \quad \frac{\partial \text{loss}}{\partial h_{11}} \quad \frac{\partial \text{loss}}{\partial h_{12}}$$

h_7 h_8 h_9 h_{10} h_{11} h_{12}



Convolution as dot products

Filter f

(same f for all time steps)

$$h_7 = f^T \bar{x}_7$$

f_7
f_6
f_5
f_4
f_3
f_2
f_1

$$h_8 = f^T \bar{x}_8$$

f_7
f_6
f_5
f_4
f_3
f_2
f_1

$$h_9 = f^T \bar{x}_9$$

f_7
f_6
f_5
f_4
f_3
f_2
f_1

$$h_{10} = f^T \bar{x}_{10}$$

f_7
f_6
f_5
f_4
f_3
f_2
f_1

$$h_{11} = f^T \bar{x}_{11}$$

f_7
f_6
f_5
f_4
f_3
f_2
f_1

$$h_{12} = f^T \bar{x}_{12}$$

f_7
f_6
f_5
f_4
f_3
f_2
f_1

Shifted by 0:

...

x_7

Shifted by -1:

...

x_6

Shifted by -2:

...

x_5

Shifted by -3:

...

x_4

Shifted by -4:

...

x_3

Shifted by -5:

...

x_2

Shifted by -6:

...

x_1

\bar{x}_7

\bar{x}_8

\bar{x}_9

\bar{x}_{10}

\bar{x}_{11}

\bar{x}_{12}

Backpropagation for 1-D convolution

- If we pretend that there is a separate filter parameter \mathbf{f}_j for each $\bar{\mathbf{x}}_j$, then the gradient of the loss w.r.t. that filter would simply be:

$$\nabla_{\mathbf{f}_j} \text{loss} = \frac{\partial h_j}{\partial \mathbf{f}_j} \frac{\partial \text{loss}}{\partial h_j} = \bar{\mathbf{x}}_j \frac{\partial \text{loss}}{\partial h_j}$$

- But of course, there is only one filter. So we add up the gradients for all time steps:

$$\nabla_{\mathbf{f}} \text{loss} = \sum_j \nabla_{\mathbf{f}_j} \text{loss} = \sum_j \bar{\mathbf{x}}_j \frac{\partial \text{loss}}{\partial h_j}$$

Questions?

Take a moment to write down any questions you have for the QA session!

Bias, nonlinearities, padding and stride

- **Bias and nonlinearities:**

- ▶ In a real CNN, we usually add a trainable scalar bias b , and we apply a nonlinearity g (such as the rectified linear unit):

$$h_j = g\left(b + \sum_{k=1}^K f_k x_{(j+k-K)}\right)$$

- **Edge cases:**

- ▶ Possible strategies for when the filter overlaps with the edges (beginning/end) of \mathbf{x} :
 - ★ Outputs where filter overlaps with edge are undefined, i.e., \mathbf{h} has fewer dimensions than \mathbf{x} ($K - 1$ fewer, to be exact)
 - ★ Pad \mathbf{x} with zeros before applying the filter
 - ★ Pad \mathbf{x} with some other relevant value, such as the overall average, the first/last value, ...

- **Stride:**

- ▶ The stride of a convolutional layer is the “step size” with which the filter is moved over the input
- ▶ We apply \mathbf{f} to the j 'th window of \mathbf{x} only if j is divisible by the stride.
- ▶ In NLP, the stride is usually 1.

Convolution with more than one axis

- Extending the convolution operation to tensors with more than one axis is straightforward.
- Let $\mathbf{X} \in \mathbb{R}^{J_1 \times \dots \times J_L}$ be a tensor that has L axes and let $\mathbf{F} \in \mathbb{R}^{K_1 \times \dots \times K_L}$ be a filter.
 - The dimensionalities of the filter axes are called filter sizes or kernel sizes (“kernel width”, “kernel height”, etc.)
 - From now on, we assume that the filter is applied to a symmetric window around position j , not just to positions to the left of j . (The rolling average was a special scenario, because we assume that future data points $x_{j'}$ with $j' > j$ are not available on day j .)
- Then the output $\mathbf{H} = \mathbf{F} * \mathbf{X}$ is a tensor with L axes, where:

$$h_{(j_1, \dots, j_L)} = g\left(b + \sum_{k_1=1}^{K_1} \dots \sum_{k_L=1}^{K_L} f_{(k_1, \dots, k_L)} x_{(j_1+k_1-\lceil \frac{K_1}{2} \rceil, \dots, j_L+k_L-\lceil \frac{K_L}{2} \rceil)}\right)$$

Example: 2D convolution

Filter \mathbf{F} with $K_1 = K_2 = 3$

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 0 & 1 & 2 \end{bmatrix}$$

Input \mathbf{X} with $J_1 = J_2 = 4$ (padded)

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 2 & 3 & 2 & 0 \\ 0 & 2 & 1 & 2 & 1 & 0 \\ 0 & 2 & 1 & 2 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Output \mathbf{H} with $J_1 = J_2 = 4$
(without bias or nonlinearity)

$$\begin{array}{c} \textcolor{red}{16} \quad ? \quad ? \quad ? \\ ? \quad ? \quad \textcolor{blue}{26} \quad ? \\ ? \quad ? \quad ? \quad ? \\ ? \quad ? \quad ? \quad \textcolor{green}{5} \end{array}$$

Channels

- So far, we have assumed that each position in the input (each day or each pixel) contains a single scalar.
- Now, we assume that our input has M features (“channels”) per position, i.e., there is an additional feature axis: $\mathbf{X} \in \mathbb{R}^{J_1 \times \dots \times J_L \times M}$
- Example:
 - ▶ $M = 3$ channels per pixel in an RGB (red/green/blue) image
 - ▶ In NLP: dimensionality of pretrained word embeddings
- Then our filter gets an additional feature axis, which has the same dimensionality as the feature axis of \mathbf{X} :

$$\mathbf{F} \in \mathbb{R}^{K_1 \times \dots \times K_L \times M}$$

- During convolution, we simply sum over this new axis as well:

$$h_{(j_1, \dots, j_L)} = g\left(b + \sum_{k_1=1}^{K_1} \dots \sum_{k_L=1}^{K_L} \sum_{m=1}^M f_{(k_1, \dots, k_L, m)} x_{(j_1+k_1 - \lceil \frac{K_1}{2} \rceil, \dots, j_L+k_L - \lceil \frac{K_L}{2} \rceil, m)}\right)$$

Filter banks

- A filter bank is a tensor that consists of N filters, which each have the same shape.
- The filters of a filter bank are applied to \mathbf{X} independently, and their outputs are stacked (they form a new axis in the output).
- So let this be our input and filter bank:

$$\mathbf{X} \in \mathbb{R}^{J_1 \times \dots \times J_L \times M}$$

$$\mathbf{F} \in \mathbb{R}^{K_1 \times \dots \times K_L \times M \times N}$$

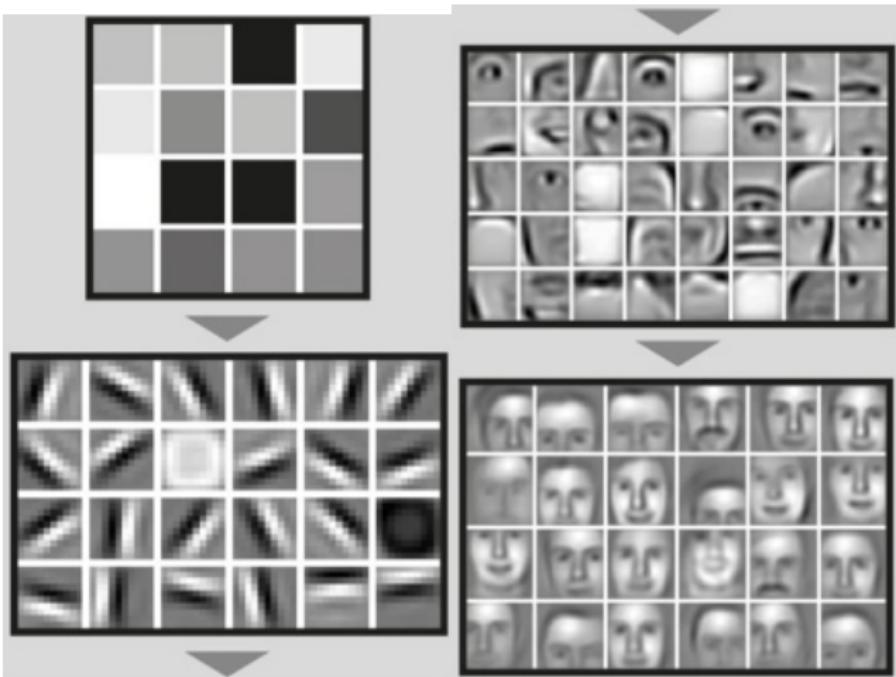
- Then our output is a tensor of shape $\mathbf{H} \in \mathbb{R}^{J_1 \times \dots \times J_L, N}$
 - ▶ (assuming that we are padding the first L axes of \mathbf{X} to preserve their dimensionality in \mathbf{H})
- where:

$$h_{(j_1, \dots, j_L, n)} = g\left(b + \sum_{k_1=1}^{K_1} \dots \sum_{k_L=L}^{K_L} \sum_{m=1}^M f_{(k_1, \dots, k_L, m, n)} X_{(j_1+k_1-\lceil \frac{K_1}{2} \rceil, \dots, j_L+k_L-\lceil \frac{K_L}{2} \rceil, m)}\right)$$

Assumptions behind convolution

- Translation invariance: Same object in different positions.
 - ▶ Parameter sharing: Filters are shared between all positions.
- Locality: Meaningful objects form coherent areas
 - ▶ In a single convolutional layer, information can travel no further than a few positions (depending on filter size)
- Hierarchy of features from simple to complex:
 - ▶ In computer vision, we often apply many convolutional layers one after another
 - ▶ With every layer, the information travels further, and the feature vectors become more complex
 - ▶ Pixels → edges → shapes → small objects → bigger objects

Convolution



Source: Computer science: The learning machines. Nature (2014).

Questions?

Take a moment to write down any questions you have for the QA session!

Outline

1 Neural network architectures

2 Convolutional layers

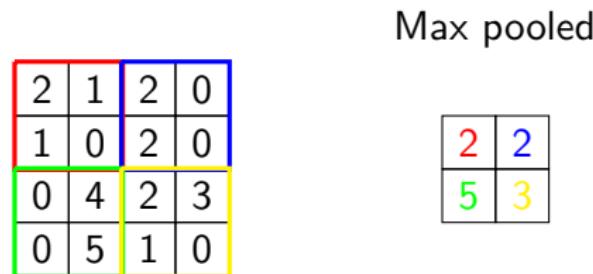
- Example: 1-D convolution
- Backpropagation for 1-D convolution
- Some additional details
- Convolution with more than one axis
- Convolution with more than one feature
- Convolution with a filter bank
- Assumptions behind convolution

3 Pooling layers

4 CNNs in NLP

Pooling layers

- Pooling layers are parameter-less
- Divide axes of \mathbf{H} (excluding the filter/feature axis) into a coarse-grained “grid”
- Aggregate each grid cell with some operator, such as the average or maximum
- Example (shown without a feature axis):

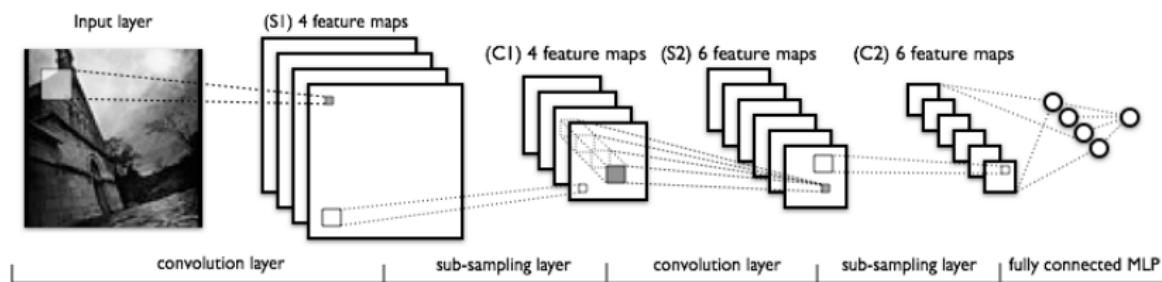


Pooling layers

- When pooling, we lose fine-grained information about exact positions
- In return, pooling allows us to aggregate features from a local neighborhood into a single feature.
 - ▶ For example, if we have a neighborhood where many “dog features” were detected (meaning, shapes that look like parts of a dog), we want to aggregate that information into a single “dog neuron”

Convolution and Pooling: LeNet

- In computer vision, we often apply pooling between convolutional layers.
- Repeated pooling has the effect of reducing tensor sizes.



LeCun et al. (1998). Gradient-based learning applied to document recognition.

Outline

1 Neural network architectures

2 Convolutional layers

- Example: 1-D convolution
- Backpropagation for 1-D convolution
- Some additional details
- Convolution with more than one axis
- Convolution with more than one feature
- Convolution with a filter bank
- Assumptions behind convolution

3 Pooling layers

4 CNNs in NLP

CNNs in NLP

- Images are 2D, but text is a 1D sequence (of words, n-grams, etc).
- Words are usually represented by M -dimensional word embeddings (e.g., from Word2Vec)
- So on text, we do 1-D convolution with M input features:
 - ▶ Input matrix: $\mathbf{X} \in \mathbb{R}^{J \times M}$
 - ▶ Filter bank of N filters: $\mathbf{F} \in \mathbb{R}^{K \times M \times N}; K < J$
 - ▶ Output matrix: $\mathbf{H} \in \mathbb{R}^{J \times N}$
 - ★ (assuming that we padded \mathbf{X} with zeros along its first axis)
- Usually, CNNs in NLP are not as deep as CNNs in Computer Vision – often just one convolutional layer.

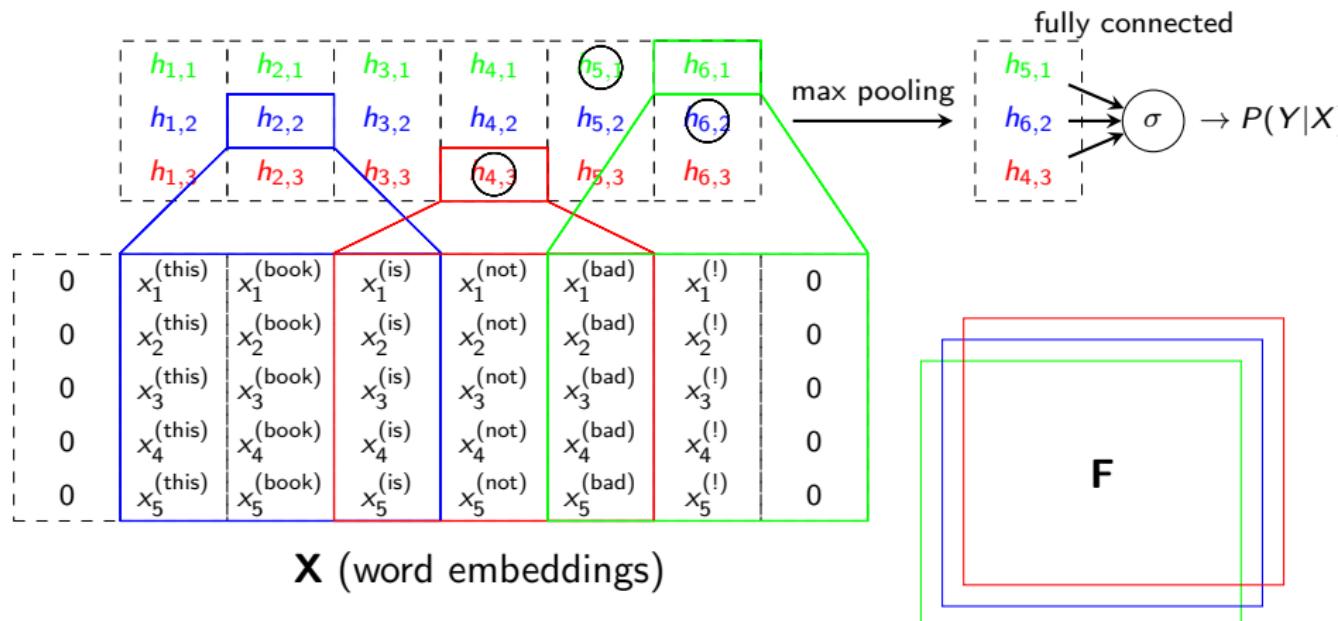
Pooling in NLP

- Pooling is less frequently used in NLP.
- If the task is a word-level task (i.e., we need one output vector per word), we can simply use the output of the final convolutional layer – no pooling needed.
- If the task is some sentence-level task (i.e., we need a single vector to represent the entire sentence), we usually do a “global” pooling step over the entire sentence *after* the last convolutional layer:

$$\mathbf{h}_{\text{avgpool}} = \frac{1}{J} \sum_{j=1}^J \mathbf{h}_j$$

$$\mathbf{h}_{\text{maxpool}} = \begin{bmatrix} \max_j h_{j,1} \\ \vdots \\ \max_j h_{j,N} \end{bmatrix}$$

CNNs in NLP



$\mathbf{X} = \text{"this book is not bad !"}, \mathbf{Y} = \text{"positive"}$

- How many parameters in the filter bank? $3 \times 5 \times 3$ (assuming no bias)

Questions?

Take a moment to write down any questions you have for the QA session!

Introduction to pytorch

Deep Learning for NLP: Lecture 6(b)

Centrum für Informations- und Sprachverarbeitung
Ludwig-Maximilians-Universität München
poerner@cis.uni-muenchen.de

December 09, 2020

Outline

- 1 Why pytorch?
- 2 Tensors and operations
- 3 Automatic differentiation
- 4 Modules
- 5 Loss functions
- 6 Optimizers
- 7 Datasets and data loaders
- 8 Implementing your own model
- 9 Using pytorch on a GPU
- 10 Putting it all together

Outline

- 1 Why pytorch?
- 2 Tensors and operations
- 3 Automatic differentiation
- 4 Modules
- 5 Loss functions
- 6 Optimizers
- 7 Datasets and data loaders
- 8 Implementing your own model
- 9 Using pytorch on a GPU
- 10 Putting it all together

Can we implement neural networks with numpy?

- Sure we can, but...
 - ▶ ... we would have to implement many functionalities from scratch
 - ▶ ... we would have to derive all of our gradients by hand → error-prone and annoying
 - ▶ ... numpy has no inherent way of grouping functions and their parameters → no modularization
 - ▶ ... numpy does not support calculations on GPU

Enter pytorch!

- Open source library for deep learning with Python, developed by Facebook
- Documentation: <http://pytorch.org/docs/master/torch.html>
- 60 minute tutorial: https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
- Popular alternatives are *tensorflow* by Google, *CNTK* by Microsoft

pytorch...

- ... many pre-implemented layers, loss functions and optimizers
- ... can do gradients and backpropagation automatically
- ... object-oriented grouping of functions and parameters into modules
- ... supports calculations on GPU

Outline

- 1 Why pytorch?
- 2 Tensors and operations
- 3 Automatic differentiation
- 4 Modules
- 5 Loss functions
- 6 Optimizers
- 7 Datasets and data loaders
- 8 Implementing your own model
- 9 Using pytorch on a GPU
- 10 Putting it all together

Tensors

- `torch.tensor`: Numerical objects (scalars, vectors, matrices...)
- Every tensor has a data type (`dtype`) and a shape (which is called `size`)
- Tensors can be manipulated and combined via operations (addition, matrix multiplication, concatenation, etc.)
- The result of a tensor operation is a new tensor
- Think about `torch.tensor` as the pytorch equivalent of `numpy.array` (but with some additional abilities related to gradients)

Creating tensors

```
>>> import torch
>>> # You can create tensors from nested Python lists:
>>> torch.tensor([[1.0, 1.3], [7.7, 8.0]], dtype=torch.float32)
tensor([[1.0000, 1.3000],
       [7.7000, 8.0000]])
>>> # ... or from numpy arrays:
>>> torch.from_numpy(np.arange(3))
tensor([0, 1, 2])
>>> # ... or from all-zeros / all-ones:
>>> torch.ones(size=(2,3))
>>> torch.zeros(size=(2,3))
>>> # ... or through random initialization:
>>> y = torch.rand(size=(2,), dtype=torch.float32)
>>> # You can easily convert tensors back to numpy arrays:
>>> y.detach().numpy()
array([0.5634323, 0.8529429], dtype=float32)
```

Tensor operations

- Operations often work similar to numpy (with some exceptions):

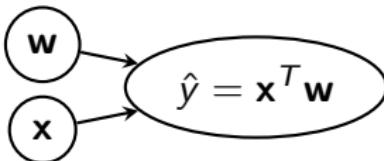
```
>>> vector = torch.tensor([1.0, 1.3])
>>> vector2 = torch.tensor([-0.5, -0.7])
>>> matrix = torch.ones(size=(2,3))
>>> # elementwise addition, subtraction, multiplication, etc.
>>> vector + vector2
>>> vector * vector2
>>> # elementwise unary functions (log, exp, etc.)
>>> torch.log(vector)
>>> torch.exp(vector)
>>> # dot product
>>> vector.dot(vector2)
>>> # matrix multiplication
>>> vector.matmul(matrix)
```

- When you are looking for a specific operation, try googling “pytorch equivalent of numpy _____”

Outline

- 1 Why pytorch?
- 2 Tensors and operations
- 3 Automatic differentiation
- 4 Modules
- 5 Loss functions
- 6 Optimizers
- 7 Datasets and data loaders
- 8 Implementing your own model
- 9 Using pytorch on a GPU
- 10 Putting it all together

Automatic differentiation



$$\nabla_w \hat{y} = \mathbf{x}; \quad \nabla_x \hat{y} = \mathbf{w}$$

- When we apply operations to tensors, pytorch implicitly builds a computation graph
- When backpropagation is invoked at a scalar tensor, the gradients of that tensor are computed via automatic differentiation
- The gradients are stored in the `grad` attribute of the input tensors:

```
>>> x = torch.tensor([1., 2., 3.], requires_grad=True) # see note below
>>> w = torch.tensor([0., 0., 1.], requires_grad=True)
>>> y_hat = x.dot(w)
>>> y_hat
tensor(3., grad_fn=<DotBackward>)
>>> y_hat.backward()
>>> (w.grad, x.grad)
(tensor([1., 2., 3.]), tensor([0., 0., 1.]))
```

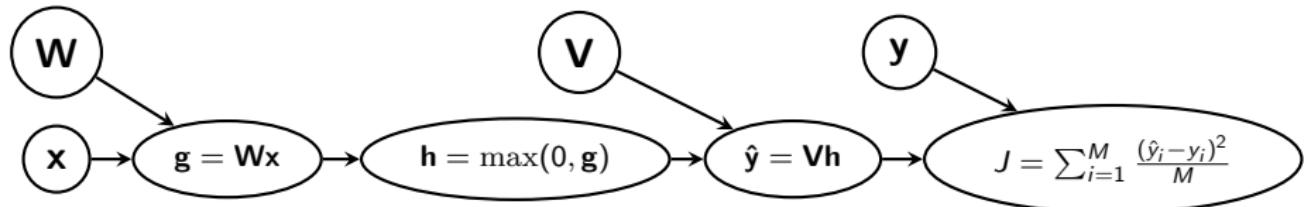
*Note: We are using single-char variable names to highlight the relation with the computation graph.
This is not best practice, and you should use more expressive variable names in your own code.

Disabling automatic differentiation

- Automatic differentiation is expensive:
 - ▶ We have to store the content of all intermediate tensors
 - ▶ After backpropagation, we have to store the gradients
- Disabling automatic differentiation:
 - ▶ Use `requires_grad=False` for tensors that do not require gradients (e.g., “frozen” parameters)
 - ▶ When you don’t intend to do gradient updates (e.g., during evaluation), use `torch.no_grad()` context

```
>>> x = torch.tensor([1., 2., 3.], requires_grad=False)
>>> w = torch.tensor([0., 0., 1.], requires_grad=True)
>>> with torch.no_grad():
>>>     y_hat = x.dot(w)
>>>     y_hat.backward()  # no_grad context -> no backpropagation
RuntimeError
>>> y_hat = x.dot(w)
>>> y_hat.backward()
>>> (w.grad, x.grad) # gradients only for tensors with requires_grad=True
(torch.tensor([1., 2., 3.]), None)
```

Automatic differentiation via chain rule: Feed-Forward Net



```
>>> x = torch.tensor([1., 2., 3.], requires_grad=False) # inputs
>>> y = torch.tensor([.7, .8], requires_grad=False) # targets
>>> W = torch.rand(size=(3,2), requires_grad=True) # param 1
>>> V = torch.rand(size=(2,2), requires_grad=True) # param 2
>>> g = x.matmul(W) # hidden vector
>>> h = torch.relu(g) # hidden vector after nonlinearity
>>> y_hat = h.matmul(V) # predicted vector
>>> J = torch.mean((y_hat - y)**2) # MSE loss
>>> J.backward()
>>> W.grad
tensor([[2.1843, 1.4898],
        [4.3687, 2.9796],
        [6.5530, 4.4694]])
>>> V.grad
tensor([[3.7324, 4.8193],
        [7.0224, 9.0673]])
```

Automatic differentiation

- No matter how complicated our neural network becomes: As long as every function is a (differentiable) pytorch function, pytorch can derive the gradients via the chain rule.
- No more manual backpropagation!



Outline

- 1 Why pytorch?
- 2 Tensors and operations
- 3 Automatic differentiation
- 4 Modules
- 5 Loss functions
- 6 Optimizers
- 7 Datasets and data loaders
- 8 Implementing your own model
- 9 Using pytorch on a GPU
- 10 Putting it all together

Modules

- Pytorch groups functions and their parameters into *modules* (also called *layers*)
- `torch.nn` contains many predefined module classes:

```
>>> import torch.nn as nn  
>>> nn.Linear(in_features, out_features) # simple linear layer  
>>> nn.Conv2d(in_channels, out_channels, kernel_size) # 2D CNN  
>>> nn.Embedding(num_embeddings, embedding_dim) # lookup layer  
>>> # and many more
```

- Here, we instantiate a linear layer with input size 3 and output size 2:

```
>>> torch.random.manual_seed(0)  
>>> linear = nn.Linear(in_features=3, out_features=2, bias=True)
```

Modules: parameters

- Most modules have parameters
- For example, our linear layer has a parameter matrix and a bias vector
- The parameters are randomly initialized when the module is created
- We can inspect the parameters:

```
>>> list(linear.parameters())
[Parameter containing:
tensor([[-0.0043,  0.3097, -0.4752],
       [-0.4249, -0.2224,  0.1548]], requires_grad=True),
 Parameter containing:
tensor([-0.0114,  0.4578], requires_grad=True)]
```

- Note that `requires_grad` is set to `True`, because pytorch expects us to train these parameters.

Modules: forward function

- Every module class has a `forward` function, which handles the forward-pass logic.
- In the case of the linear layer, the forward-pass logic is simply:

$$f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

```
# from https://pytorch.org/docs/stable/_modules/torch
class Linear(Module):
    # ...
    def forward(self, input: Tensor) -> Tensor:
        return F.linear(input, self.weight, self.bias)

# F.linear:
def linear(input, weight, bias=None):
    # ...
    output = input.matmul(weight.t())
    if bias is not None:
        output += bias
    ret = output
    return ret
```

Modules: forward() function

- “Calling” a module means calling its forward function:

```
>>> inputs = torch.tensor([[1., 2., 3.], [4., 5., 6.]])  
>>> predictions = linear(inputs)  
>>> predictions  
tensor([-0.8219,  0.0526],  
      [-1.3313, -1.4247]), grad_fn=<AddmmBackward>  
>>> linear.forward(inputs) # this is equivalent  
tensor([-0.8219,  0.0526],  
      [-1.3313, -1.4247]), grad_fn=<AddmmBackward>
```

- Typically, modules expect their inputs to have a batch axis. So in reality, the input is a tensor of shape (N, \dots) , where N is the number of datapoints per batch (here: 2), and \dots stands for the remaining axes (here: one 3-dimensional feature axis). The layer is then applied to every datapoint independently.

Outline

- 1 Why pytorch?
- 2 Tensors and operations
- 3 Automatic differentiation
- 4 Modules
- 5 Loss functions
- 6 Optimizers
- 7 Datasets and data loaders
- 8 Implementing your own model
- 9 Using pytorch on a GPU
- 10 Putting it all together

Loss functions

- Many popular loss functions are pre-implemented in `torch.nn`:

```
>>> nn.MSELoss() # mean squared error
>>> nn.BCELoss() # Negative Log Likelihood (binary)
>>> nn.NLLLoss() # Negative Log Likelihood (multi-class)
>>> # and many more
```

- A loss function behaves like a module, except that it has no parameters, and its forward function usually takes two inputs (the prediction and the target):

```
>>> loss_function = nn.MSELoss()
>>> targets = torch.tensor([[.1, .2], [-.3, -.4]])
>>> predictions = linear(inputs)
>>> predictions
tensor([[-0.8219,  0.0526],
       [-1.3313, -1.4247]], grad_fn=<AddmmBackward>)
>>> loss = loss_function(predictions, targets)
>>> loss
tensor(0.7463, grad_fn=<MseLossBackward>)
```

Backpropagation

- Now we can backpropagate the loss:

```
>>> loss.backward()
```

- Let's look at the gradients of the loss w.r.t. the parameters. There is one gradient tensor per parameter (one for the matrix and one for the bias). These gradients were calculated when we called `loss.backward()`, and they are stored in the `grad` attribute:

```
>>> [param.grad for param in linear.parameters()]
[tensor([-2.5235, -3.5000, -4.4766],
       [-2.1231, -2.7092, -3.2953]),
 tensor([-0.9766, -0.5861])]
```

Outline

- 1 Why pytorch?
- 2 Tensors and operations
- 3 Automatic differentiation
- 4 Modules
- 5 Loss functions
- 6 Optimizers
- 7 Datasets and data loaders
- 8 Implementing your own model
- 9 Using pytorch on a GPU
- 10 Putting it all together

Gradient update

- Now that we have the gradients, we could update the parameters manually:

```
>>> learning_rate = 0.01
>>> with torch.no_grad():
...     for param in linear.parameters():
...         param.sub_(learning_rate*param.grad)
```

- But that's too much work.
- Instead, we use an optimizer from the `torch.optim` package

Optimizers

- Optimizers are objects that take care of gradient updates.
- When instantiating an optimizer, you pass the model parameters to it. That way, the optimizer knows what it has to update:

```
>>> import torch.optim as optim  
>>> optimizer = optim.SGD(linear.parameters(), lr=0.01)
```

- Pytorch has many complex pre-implemented optimizers
- Complex optimizers can do things like adapt the momentum of the learning rate, use different learning rates for different parameters, etc.

```
>>> optim.Adam(linear.parameters(), lr=0.01)  
>>> optim.RMSprop(linear.parameters(), lr=0.01)  
>>> # ...and many more. Let's use SGD for now.
```

Optimizers

- With an optimizer, the gradient update step is simply:

```
>>> optimizer.step() # update all parameters with their gradients  
>>> optimizer.zero_grad() # zero out all gradients
```

- Careful: The call to `optimizer.zero_grad()` is easy to forget. But it is important.
 - If we don't zero out gradients, they sum up inside the grad tensor. As a result, the "gradient" at training step 1000 would be $\sum_{i=1}^{1000} \nabla_{\theta} J(\theta; x_i, y_i)$ instead of $\nabla_{\theta} J(\theta; x_{1000}, y_{1000})$ (where x_i, y_i is the i 'th training batch)

Outline

- 1 Why pytorch?
- 2 Tensors and operations
- 3 Automatic differentiation
- 4 Modules
- 5 Loss functions
- 6 Optimizers
- 7 Datasets and data loaders
- 8 Implementing your own model
- 9 Using pytorch on a GPU
- 10 Putting it all together

Handling data

- What we could do:

```
>>> all_inputs, all_targets = load_data() # data as tensors
>>> all_inputs.shape, all_targets.shape
(torch.Size([100, 3]), torch.Size([100, 2]))
>>>
>>> custom_shuffling_function(all_inputs, all_targets)
>>> for i in range(0, all_inputs.shape[0], batch_size):
>>>     inputs = all_inputs[i:i+batch_size]
>>>     targets = all_targets[i:i+batch_size]
```

- Runs in same thread as model

Datasets and Dataloaders

- Datasets store and offer access to data
- DataLoaders take care of iteration, batching, shuffling, ...
- multithreading → we don't waste GPU time waiting for the next batch

```
>>> import torch.utils.data as data
>>> dataset = data.TensorDataset(all_inputs, all_targets)
>>> len(dataset)
100
>>> dataloader = data.DataLoader(dataset, batch_size=3, shuffle=True,
                                 num_workers=4)
>>> len(dataloader) # ceil(len(dataset) / batch_size)
34
>>> for inputs, targets in dataloader:
...     print(inputs)
...     print(targets)
...     break
...
tensor([[0.7601, 0.2101, 0.3622],
        [0.6240, 0.2828, 0.1898],
        [0.7679, 0.9298, 0.4364]])
tensor([[0.4980, 0.3000],
        [0.8256, 0.0253],
        [0.0634, 0.7476]])
```

Outline

- 1 Why pytorch?
- 2 Tensors and operations
- 3 Automatic differentiation
- 4 Modules
- 5 Loss functions
- 6 Optimizers
- 7 Datasets and data loaders
- 8 **Implementing your own model**
- 9 Using pytorch on a GPU
- 10 Putting it all together

Implementing your own model

- When you implement a model, you will normally define a module class, which inherits from `nn.Module`
- Normally, you will build the model from simpler sub-modules
- To implement the feed-forward network from a few slides ago:

```
>>> import torch.nn.functional as F
>>> class FFN(nn.Module):
...     def __init__(self, input_size, hidden_size, output_size):
...         super(FFN, self).__init__() # mandatory: invoke parent init
...
...         # register sub-modules as attributes
...         self.linear1 = nn.Linear(input_size, hidden_size)
...         self.linear2 = nn.Linear(hidden_size, output_size)
...
...     def forward(self, inputs):
...         hidden_vectors = F.relu(self.linear1(inputs))
...         predictions = self.linear2(hidden_vectors)
...         return predictions
```

Implementing your own model

- Our model can then be instantiated and “called” like any other module from `torch.nn`:

```
>>> model = FFN(input_size=3, hidden_size=2, output_size=2)
>>> predictions = model(inputs)
```

- The parameters of our model are the parameters of its sub-modules:

```
>>> list(model.parameters()) # 2 matrices, 2 bias vectors
[Parameter containing:
tensor([[ 0.4607, -0.0698,  0.0371],
        [ 0.5576,  0.0167,  0.5723]], requires_grad=True),
 Parameter containing:
tensor([0.2204, 0.4270], requires_grad=True),
 Parameter containing:
tensor([[-0.1111, -0.6049],
        [ 0.0504,  0.3004]], requires_grad=True),
 Parameter containing:
tensor([0.0817, 0.1186], requires_grad=True)]
```

Outline

- 1 Why pytorch?
- 2 Tensors and operations
- 3 Automatic differentiation
- 4 Modules
- 5 Loss functions
- 6 Optimizers
- 7 Datasets and data loaders
- 8 Implementing your own model
- 9 Using pytorch on a GPU
- 10 Putting it all together

Using pytorch on a GPU

- To use pytorch on a GPU, you need a compatible GPU (duh!) and CUDA
 - ▶ see <https://pytorch.org/get-started/locally/>
- Alternatively, you can use a cloud GPU via Google Colab (see demo at the end of Lecture 5(b)).
- Note: The coding exercises are designed in such a way that you can solve them on a CPU or GPU. So if you can't make them work on the GPU, don't worry.
- (But if you are thinking about doing a deep learning project for your Master's thesis: This is a good time to get some practice.)

Using pytorch on a GPU

- First, check if CUDA is available on your machine:

```
>>> torch.cuda.is_available()  
True
```

- Then, move your model (your `nn.Module` object) to the GPU.

```
>>> model = model.to(device='cuda')
```

- If you have several GPUs, you can select them by their index:

```
>>> model = model.to(device='cuda:2') # indices start at 0
```

- Inputs have to be on the same device as the model:

```
>>> inputs = inputs.to(device='cuda') # or whatever device the model is on  
>>> predictions = model(inputs)
```

- Tensors that were computed by your model (here: `predictions`) live on the same device as the model. To move them back to the CPU:

```
>>> predictions.to(device='cpu')
```

Outline

- 1 Why pytorch?
- 2 Tensors and operations
- 3 Automatic differentiation
- 4 Modules
- 5 Loss functions
- 6 Optimizers
- 7 Datasets and data loaders
- 8 Implementing your own model
- 9 Using pytorch on a GPU
- 10 Putting it all together

Putting it all together: The training loop

```
>>> device = 'cuda' if torch.cuda.is_available() else 'cpu'  
>>> model = MODELCLASS(...) # e.g., model = FFN(3, 2, 2)  
>>> model = model.to(device)  
>>> optimizer = OPTIMCLASS(model.parameters(), ...) # e.g., optim.SGD(...)  
>>> loss_function = LOSSCLASS(...) # e.g., nn.MSELoss()  
>>> X, Y = load_data() # load training data tensors from somewhere  
>>> dataset = data.TensorDataset(X, Y)  
>>> dataloader = data.DataLoader(dataset, shuffle=True, batch_size=16)  
>>>  
>>> for epoch in range(10):  
>>>     loss_sum = 0.0  
>>>     for inputs, targets in dataloader:  
>>>         inputs = inputs.to(device)  
>>>         targets = targets.to(device)  
>>>         predictions = model(inputs)  
>>>         loss = loss_function(predictions, targets)  
>>>         loss.backward()  
>>>         optimizer.step()  
>>>         optimizer.zero_grad()  
>>>         loss_sum += loss.to('cpu').detach().numpy()  
>>>         print('Avg loss for epoch', epoch, ':', loss_sum/len(dataloader))
```

Recurrent Neural Networks (RNNs)

Deep Learning for NLP: Lecture 7

Centrum für Informations- und Sprachverarbeitung
Ludwig-Maximilians-Universität München
poerner@cis.uni-muenchen.de

December 16, 2020

Outline

- 1 What are RNNs?
- 2 Vanilla RNN
- 3 Training RNNs
 - Backpropagation through time
 - The vanishing gradients problem
- 4 Gated RNNs
 - LSTM
 - GRU
 - Comparison
- 5 RNN applications
- 6 Extensions: Multi-layer RNNs, bidirectionality
- 7 RNNs with attention

Outline

- 1 What are RNNs?
- 2 Vanilla RNN
- 3 Training RNNs
 - Backpropagation through time
 - The vanishing gradients problem
- 4 Gated RNNs
 - LSTM
 - GRU
 - Comparison
- 5 RNN applications
- 6 Extensions: Multi-layer RNNs, bidirectionality
- 7 RNNs with attention

What are RNNs?

- Assumption: Text is written sequentially, so our model should read it sequentially
- “RNN”: class of Neural Network architectures that process text sequentially (left-to-right or right-to-left)
- Generally speaking:
 - ▶ Internal “state” \mathbf{h}
 - ▶ RNN consumes one input $\mathbf{x}^{(j)}$ per time step j
 - ▶ Update function: $\mathbf{h}^{(j)} = f(\mathbf{x}^{(j)}, \mathbf{h}^{(j-1)}; \theta)$
 - ★ where parameters θ are shared across all time steps

Outline

- 1 What are RNNs?
- 2 Vanilla RNN
- 3 Training RNNs
 - Backpropagation through time
 - The vanishing gradients problem
- 4 Gated RNNs
 - LSTM
 - GRU
 - Comparison
- 5 RNN applications
- 6 Extensions: Multi-layer RNNs, bidirectionality
- 7 RNNs with attention

Vanilla RNN

- Let $\mathbf{x}^{(1)} \dots \mathbf{x}^{(J)}$, with $\mathbf{x}^{(j)} \in \mathbb{R}^{d'}$ be our input (e.g., a sequence of d' -dimensional word embeddings)
- Let $\mathbf{h}^{(0)} = \{0\}^d$ be our initial state
- Let $\theta = \{\mathbf{W} \in \mathbb{R}^{d \times d'}, \mathbf{V} \in \mathbb{R}^{d \times d}, \mathbf{b} \in \mathbb{R}^d\}$ be our parameters
- Then the j 'th update is defined as:

$$\mathbf{h}^{(j)} = \tanh(\mathbf{W}\mathbf{x}^{(j)} + \mathbf{V}\mathbf{h}^{(j-1)} + \mathbf{b})$$

Vanilla RNN

- Sentence: “the cat sat”
- $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}) = (\mathbf{x}^{(\text{the})}, \mathbf{x}^{(\text{cat})}, \mathbf{x}^{(\text{sat})})$

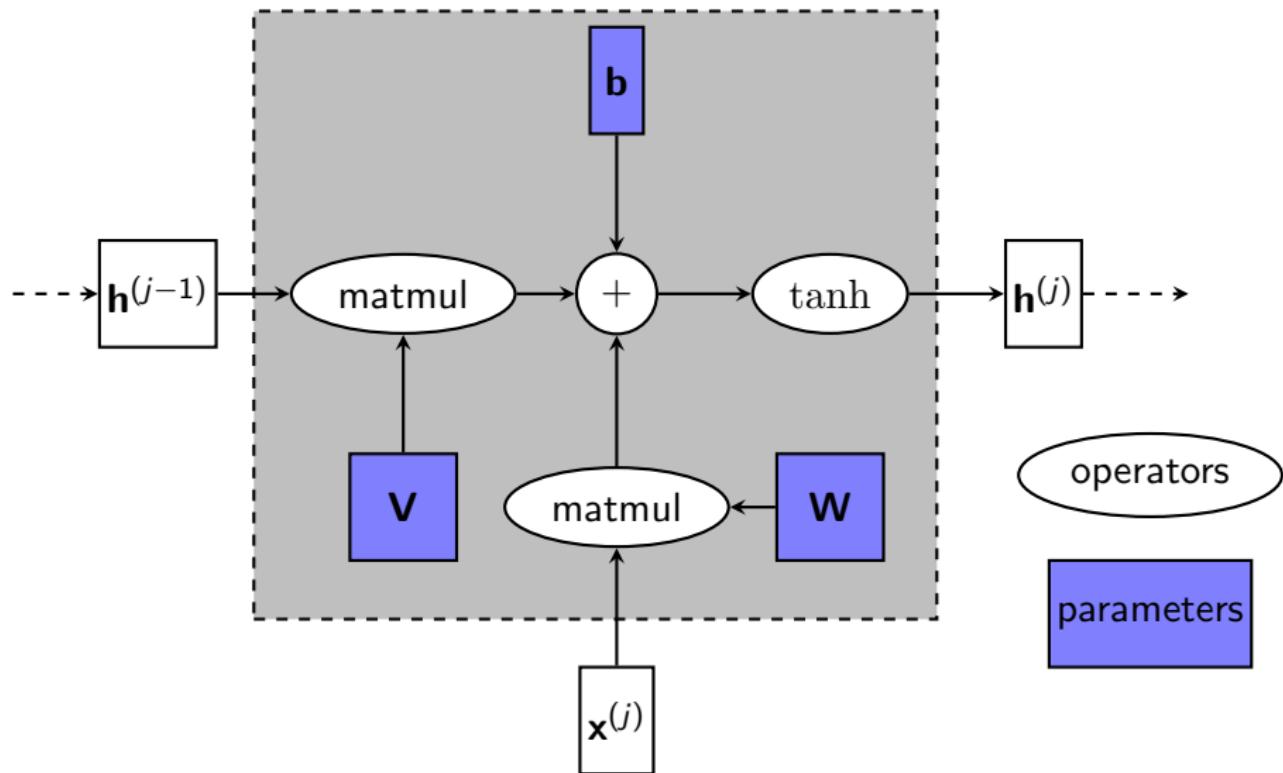
$$\mathbf{h}^{(1)} = \tanh(\mathbf{Wx}^{(\text{the})} + \mathbf{Vh}^{(0)} + \mathbf{b}) = \tanh(\mathbf{Wx}^{(\text{the})} + \mathbf{b})$$

$$\mathbf{h}^{(2)} = \tanh(\mathbf{Wx}^{(\text{cat})} + \mathbf{Vh}^{(1)} + \mathbf{b}) = \tanh(\mathbf{Wx}^{(\text{cat})} + \mathbf{V}\tanh(\mathbf{Wx}^{(\text{the})} + \mathbf{b}) + \mathbf{b})$$

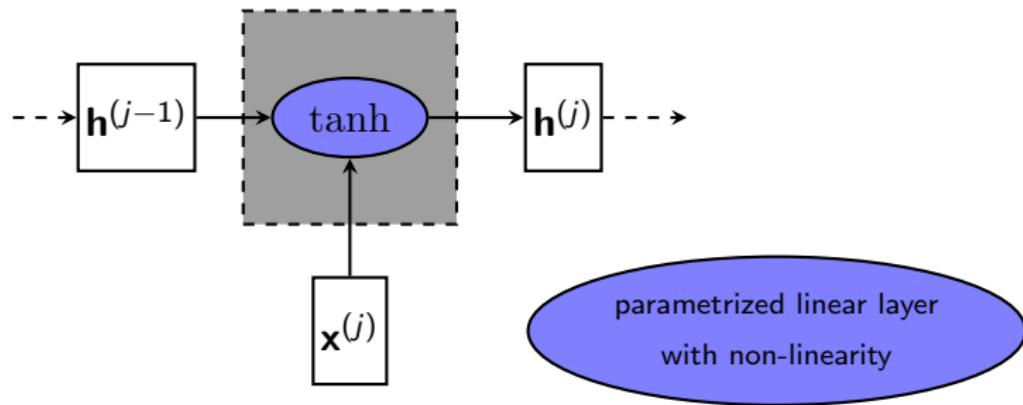
$$\mathbf{h}^{(3)} = \tanh(\mathbf{Wx}^{(\text{sat})} + \mathbf{Vh}^{(2)} + \mathbf{b}) = \dots$$

- **Question:** Which word(s) does $\mathbf{h}^{(1)}$ depend on?
 - ▶ “the”
- **Question:** Which word(s) does $\mathbf{h}^{(2)}$ depend on?
 - ▶ “the cat”
- **Question:** Which word(s) does $\mathbf{h}^{(3)}$ depend on?
 - ▶ “the cat sat”

Vanilla RNN cell



Vanilla RNN cell (simplified)



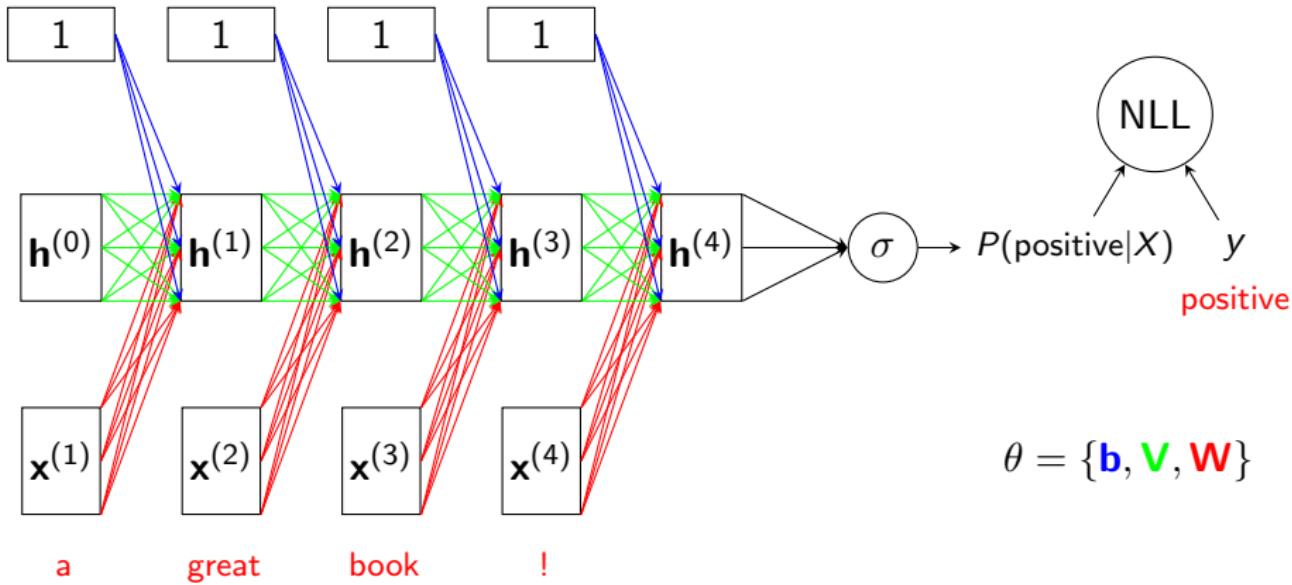
Questions?

Take a moment to write down any questions you have for the QA session!

Outline

- 1 What are RNNs?
- 2 Vanilla RNN
- 3 Training RNNs
 - Backpropagation through time
 - The vanishing gradients problem
- 4 Gated RNNs
 - LSTM
 - GRU
 - Comparison
- 5 RNN applications
- 6 Extensions: Multi-layer RNNs, bidirectionality
- 7 RNNs with attention

Example: Binary sentiment classifier

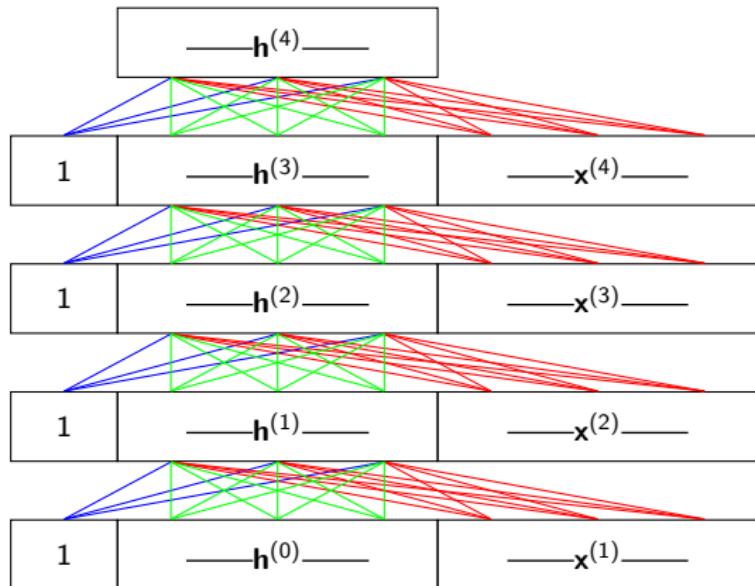


(Non-linearity omitted for readability.)

Backpropagation through time

- RNNs are trained via “backpropagation through time”
- To understand how this works, imagine the RNN as a Feed-Forward Net (FFN), whose depth is equal to the sentence length
- For now, let’s pretend that every time step (every layer) has its own dummy parameters $\theta^{(j)}$, which are identical copies of θ

Vanilla RNN as Feed-Forward Net



$$\theta^{(4)} = \{\mathbf{b}^{(4)}, \mathbf{V}^{(4)}, \mathbf{W}^{(4)}\}$$

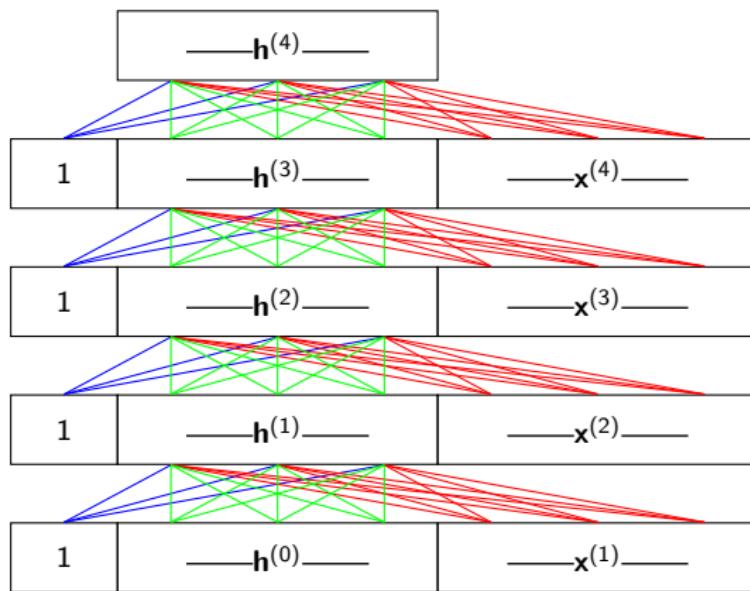
$$\theta^{(3)} = \{\mathbf{b}^{(3)}, \mathbf{V}^{(3)}, \mathbf{W}^{(3)}\}$$

$$\theta^{(2)} = \{\mathbf{b}^{(2)}, \mathbf{V}^{(2)}, \mathbf{W}^{(2)}\}$$

$$\theta^{(1)} = \{\mathbf{b}^{(1)}, \mathbf{V}^{(1)}, \mathbf{W}^{(1)}\}$$

Backpropagating through time

- Assume that we have calculated derivatives of loss L w.r.t. last state:
 $\frac{\partial L}{\partial \mathbf{h}^{(J)}}$
- Then we can apply the normal backpropagation algorithm for FFNs to calculate $\nabla_{\mathbf{W}^{(i)}} L, \nabla_{\mathbf{V}^{(i)}} L, \nabla_{\mathbf{b}^{(i)}} L$ for all dummy parameters.



$$\{\nabla_{\mathbf{b}^{(4)}} L, \nabla_{\mathbf{V}^{(4)}} L, \nabla_{\mathbf{W}^{(4)}} L\}$$

$$\{\nabla_{\mathbf{b}^{(3)}} L, \nabla_{\mathbf{V}^{(3)}} L, \nabla_{\mathbf{W}^{(3)}} L\}$$

$$\{\nabla_{\mathbf{b}^{(2)}} L, \nabla_{\mathbf{V}^{(2)}} L, \nabla_{\mathbf{W}^{(2)}} L\}$$

$$\{\nabla_{\mathbf{b}^{(1)}} L, \nabla_{\mathbf{V}^{(1)}} L, \nabla_{\mathbf{W}^{(1)}} L\}$$

Backpropagating through time

- But of course, there is only one set of parameters.
- So the actual gradients are:

$$\nabla_{\mathbf{W}} L = \sum_{j=1}^J \nabla_{\mathbf{W}^{(j)}} L$$

$$\nabla_{\mathbf{V}} L = \sum_{j=1}^J \nabla_{\mathbf{V}^{(j)}} L$$

$$\nabla_{\mathbf{b}} L = \sum_{j=1}^J \nabla_{\mathbf{b}^{(j)}} L$$

- (In reality, we are of course using pytorch to calculate the gradients.)

Vanishing gradients

- On long inputs, Vanilla RNNs suffer from “vanishing” gradients
- Vanishing gradients mean that the impact that an input has on the gradient of the loss becomes smaller when it is further away from the loss in the computation graph.

Vanishing gradients

- We assume that we have backpropagated the partial derivatives of the loss to $\mathbf{h}^{(J)}$:

$$\frac{\partial L}{\partial \mathbf{h}^{(J)}}$$

- Backpropagate to $\mathbf{h}^{(j)}$ via chain rule:

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{h}^{(j)}} &= \left[\prod_{j'=j+1}^J \left(\frac{\partial \mathbf{h}^{(j')}}{\partial \mathbf{h}^{(j'-1)}} \right)^T \right] \frac{\partial L}{\partial \mathbf{h}^{(J)}} \\ &= \left[\prod_{j'=j+1}^J \left(\frac{\partial \mathbf{Vh}^{(j'-1)}}{\partial \mathbf{h}^{(j'-1)}} \right)^T \frac{\partial \tanh(\mathbf{Vh}^{(j'-1)} + \dots)}{\partial \mathbf{Vh}^{(j'-1)}} \right] \frac{\partial L}{\partial \mathbf{h}^{(J)}}\end{aligned}$$

Vanishing gradients

$$\frac{\partial L}{\partial \mathbf{h}^{(j)}} = \left[\prod_{j'=j+1}^J \left(\frac{\partial \mathbf{V} \mathbf{h}^{(j'-1)}}{\partial \mathbf{h}^{(j'-1)}} \right)^T \frac{\partial \tanh(\mathbf{V} \mathbf{h}^{(j'-1)} + \dots)}{\partial \mathbf{V} \mathbf{h}^{(j'-1)}} \right] \frac{\partial L}{\partial \mathbf{h}^{(J)}}$$

- What happens to $\frac{\partial L}{\partial \mathbf{h}^{(j)}}$ when the distance $J - j$ grows?
 - ▶ Remember that \tanh is applied elementwise, and that the derivative of \tanh is between 0 and 1. So the **red Jacobian matrix** is a diagonal matrix with entries between 0 and 1. The product of many such matrices approaches zero.
 - ▶ Furthermore, the **blue Jacobian matrix** is just \mathbf{V} . When initialized with small enough values, $\prod \mathbf{V}$ will approach zero as well.
 - ▶ As a result, $\frac{\partial L}{\partial \mathbf{h}^{(j)}}$ approaches zero (“vanishes”)
- What does this mean?
 - ▶ Since the “dummy parameter gradients” of step j , $\nabla_{\theta^{(j)}} L$ are upstream from $\frac{\partial L}{\partial \mathbf{h}^{(j)}}$, they approach zero too, i.e., their effect on the “dummy gradient sum” is negligible.
 - ▶ This means that if the words that your RNN should be paying attention to are far from the loss, the network will not (or slowly) adjust its weights to those words

Exploding gradients

- So why don't we just use a nonlinearity with a derivative larger than 1, or initialize \mathbf{V} differently?
 - ▶ $\left\| \frac{\partial L}{\partial \mathbf{h}^{(j)}} \right\|$ would become very large ("explode"). This is even worse than vanishing gradients, because it leads to non-convergence of gradient descent.
 - ▶ So vanishing gradients is the lesser of two evils.

Questions?

Take a moment to write down any questions you have for the QA session!

Outline

- 1 What are RNNs?
- 2 Vanilla RNN
- 3 Training RNNs
 - Backpropagation through time
 - The vanishing gradients problem
- 4 Gated RNNs
 - LSTM
 - GRU
 - Comparison
- 5 RNN applications
- 6 Extensions: Multi-layer RNNs, bidirectionality
- 7 RNNs with attention

Long-Short Term Memory Network (LSTM)

- Proposed in Hochreiter and Schmidhuber, 1997
 - Became popular around 2010 for handwriting recognition, speech recognition, and many NLP problems
 - Addresses vanishing gradients by changing the architecture of the RNN cell

Long-Short Term Memory Network (LSTM)

- Two states: \mathbf{h} ("short-term memory") and \mathbf{c} ("long-term memory")
- Candidate state $\bar{\mathbf{h}} \in \mathbb{R}^d$ corresponds to \mathbf{h} in the Vanilla RNN
- Interactions are mediated by "gates" $\in (0, 1)^d$, which apply elementwise:
 - ▶ Forget gate \mathbf{f} decides what information from \mathbf{c} should be forgotten
 - ▶ Input gate \mathbf{i} decides what information from $\bar{\mathbf{h}}$ should be added to \mathbf{c}
 - ▶ Output gate \mathbf{o} decides what information from \mathbf{c} should be exposed to \mathbf{h}
- Each gate and the candidate state have their own parameters $\theta^{(i)}, \theta^{(f)}, \theta^{(o)}, \theta^{(\bar{h})}$
- "Gradient highway" from $\mathbf{c}^{(j)}$ to $\mathbf{c}^{(j-1)}$, with no non-linearities or matrix multiplications

LSTM definition

$$\mathbf{h}^{(0)} = \mathbf{c}^{(0)} = \{0\}^d$$

$$\mathbf{f}^{(j)} = \sigma(\mathbf{W}^{(f)} \mathbf{x}^{(j)} + \mathbf{V}^{(f)} \mathbf{h}^{(j-1)} + \mathbf{b}^{(f)})$$

$$\mathbf{i}^{(j)} = \sigma(\mathbf{W}^{(i)} \mathbf{x}^{(j)} + \mathbf{V}^{(i)} \mathbf{h}^{(j-1)} + \mathbf{b}^{(i)})$$

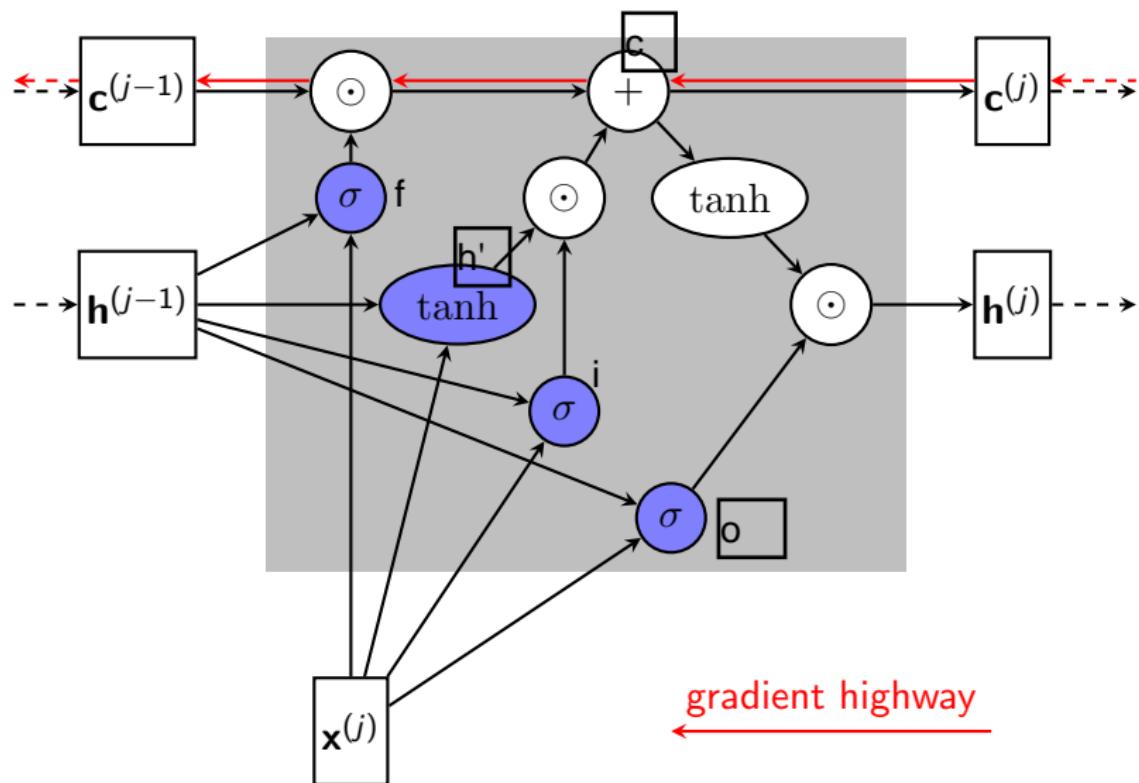
$$\mathbf{o}^{(j)} = \sigma(\mathbf{W}^{(o)} \mathbf{x}^{(j)} + \mathbf{V}^{(o)} \mathbf{h}^{(j-1)} + \mathbf{b}^{(o)})$$

$$\bar{\mathbf{h}}^{(j)} = \tanh(\mathbf{W}^{(\bar{h})} \mathbf{x}^{(j)} + \mathbf{V}^{(\bar{h})} \mathbf{h}^{(j-1)} + \mathbf{b}^{(\bar{h})})$$

$$\mathbf{c}^{(j)} = \mathbf{f}^{(j)} \odot \mathbf{c}^{(j-1)} + \mathbf{i}^{(j)} \odot \bar{\mathbf{h}}^{(j)}$$

$$\mathbf{h}^{(j)} = \mathbf{o}^{(j)} \odot \tanh(\mathbf{c}^{(j)})$$

LSTM cell



Gated Recurrent Unit (GRU)

- Proposed by Cho et al. (2014)
- Lightweight alternative to LSTM, with only one state and three sets of parameters
- State \mathbf{h} is a dynamic “interpolation” of long and short term memory
- Reset gate $\mathbf{r} \in (0, 1)^d$ controls what information passes from \mathbf{h} to candidate state $\bar{\mathbf{h}}$
- Update gate $\mathbf{z} \in (0, 1)^d$ interpolates between \mathbf{h} and $\bar{\mathbf{h}}$
- Separate set of parameters $\theta^{(r)}, \theta^{(z)}, \theta^{(\bar{h})}$ for each gate and the candidate state.

GRU definition

$$\mathbf{h}^{(0)} = \{0\}^d$$

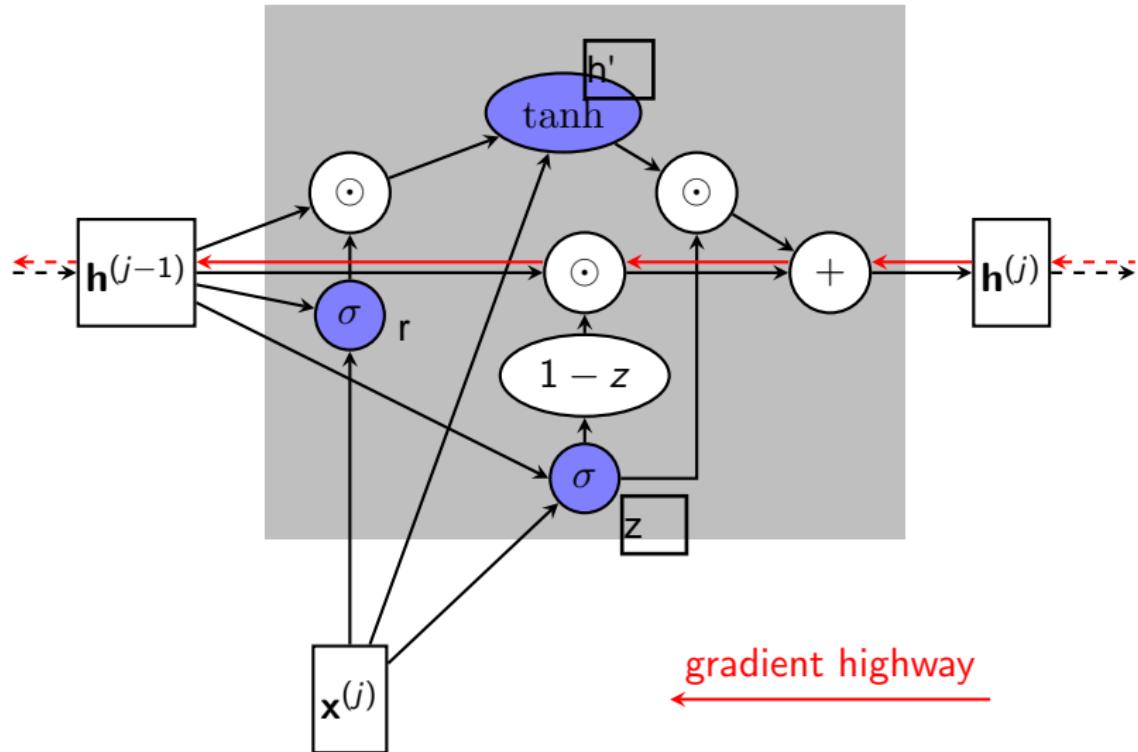
$$\mathbf{r}^{(j)} = \sigma(\mathbf{W}^{(r)} \mathbf{x}^{(j)} + \mathbf{V}^{(r)} \mathbf{h}^{(j-1)} + \mathbf{b}^{(r)})$$

$$\mathbf{z}^{(j)} = \sigma(\mathbf{W}^{(z)} \mathbf{x}^{(j)} + \mathbf{V}^{(z)} \mathbf{h}^{(j-1)} + \mathbf{b}^{(z)})$$

$$\bar{\mathbf{h}}^{(j)} = \tanh(\mathbf{W}^{(\bar{h})} \mathbf{x}^{(j)} + \mathbf{V}^{(\bar{h})} (\mathbf{r}^{(j)} \odot \mathbf{h}^{(j-1)}) + \mathbf{b}^{(\bar{h})})$$

$$\mathbf{h}^{(j)} = (1 - \mathbf{z}^{(j)}) \odot \mathbf{h}^{(j-1)} + \mathbf{z}^{(j)} \odot \bar{\mathbf{h}}^{(j)}$$

GRU cell



Vanilla vs. GRU vs. LSTM

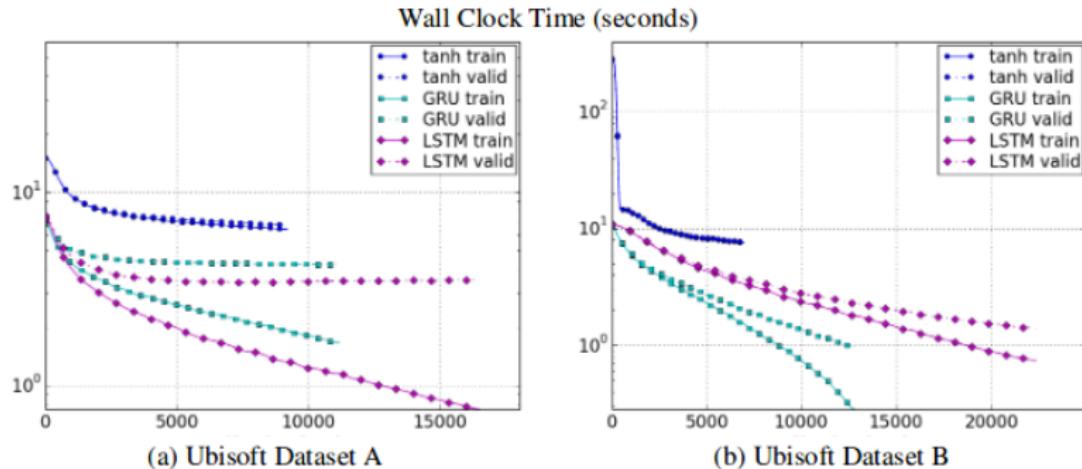


Figure from Chung et al. 2014: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling

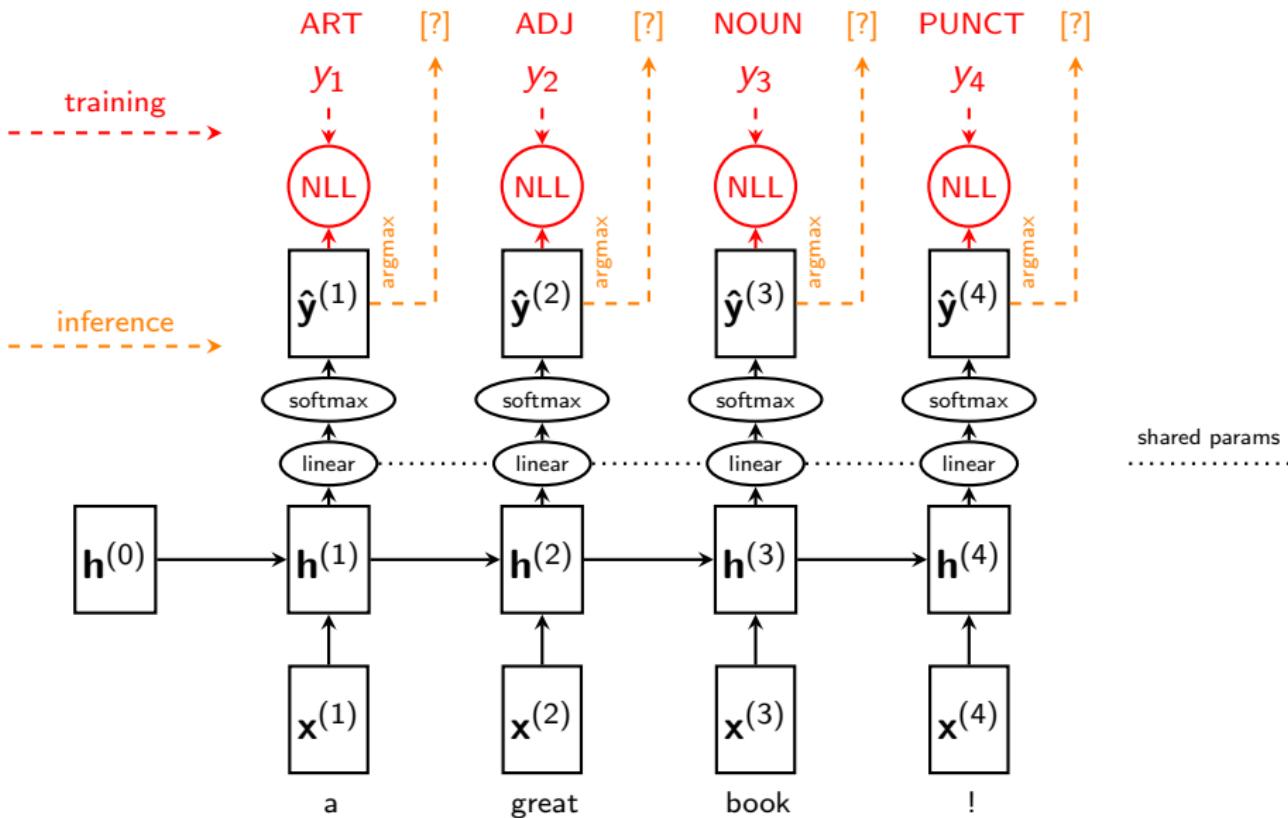
Questions?

Take a moment to write down any questions you have for the QA session!

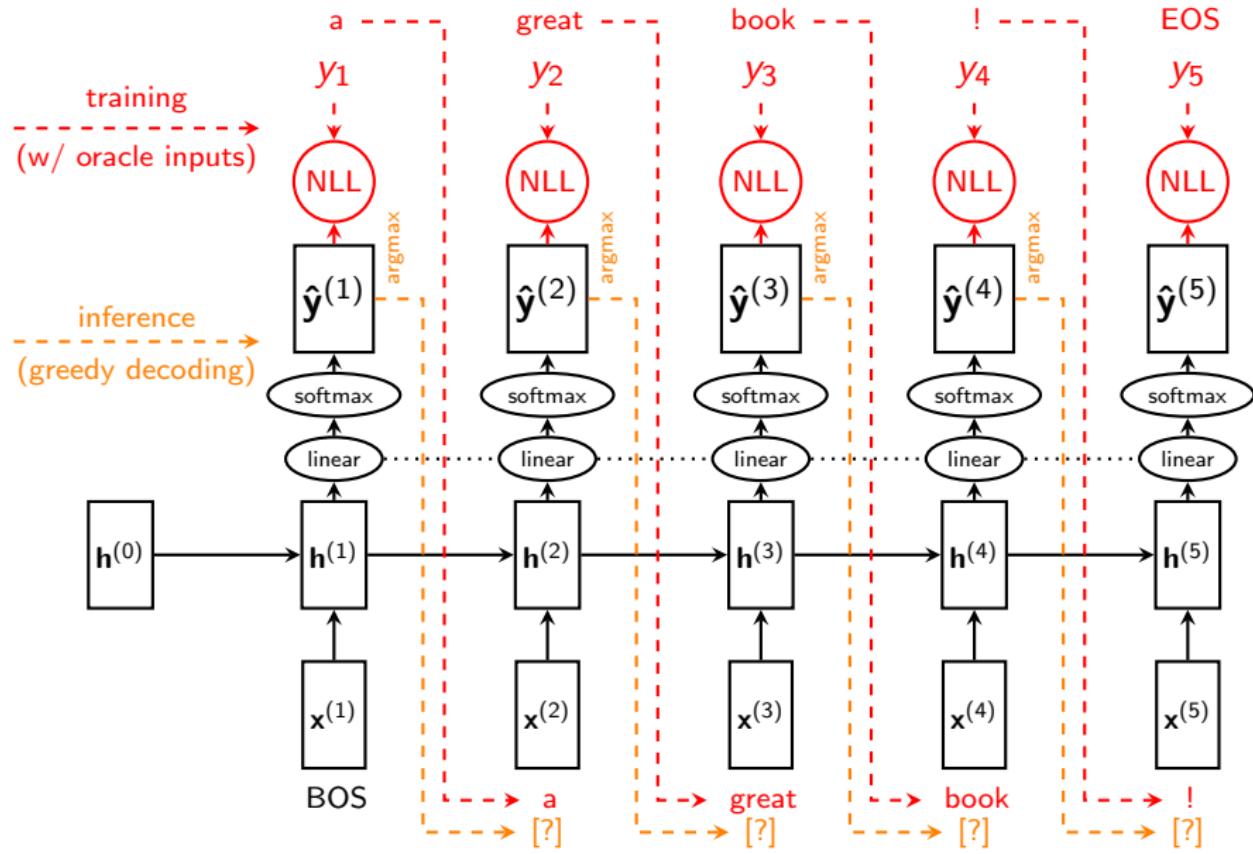
Outline

- 1 What are RNNs?
- 2 Vanilla RNN
- 3 Training RNNs
 - Backpropagation through time
 - The vanishing gradients problem
- 4 Gated RNNs
 - LSTM
 - GRU
 - Comparison
- 5 RNN applications
- 6 Extensions: Multi-layer RNNs, bidirectionality
- 7 RNNs with attention

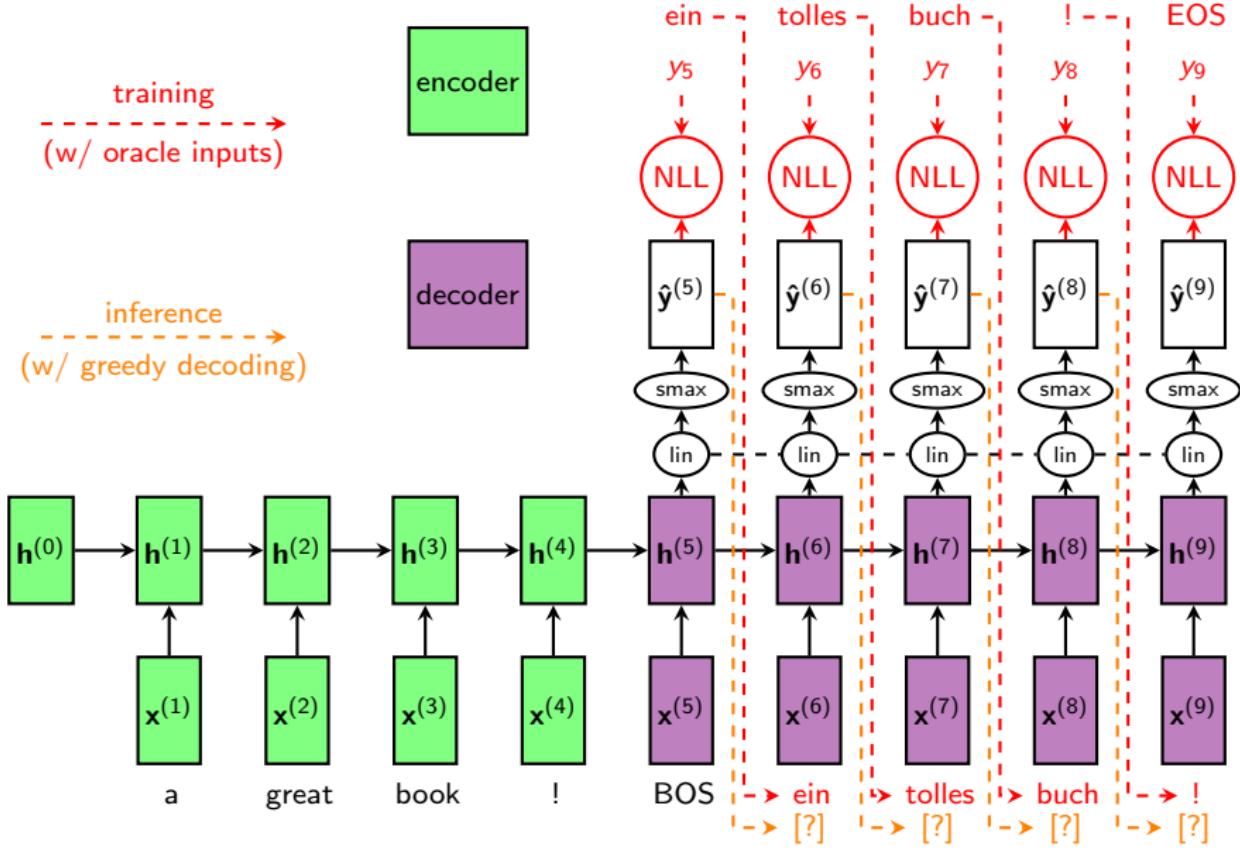
Tagging (example: Part-Of-Speech)



Autoregressive Language Modeling



Sequence-to-sequence (example: Machine Translation)



Questions?

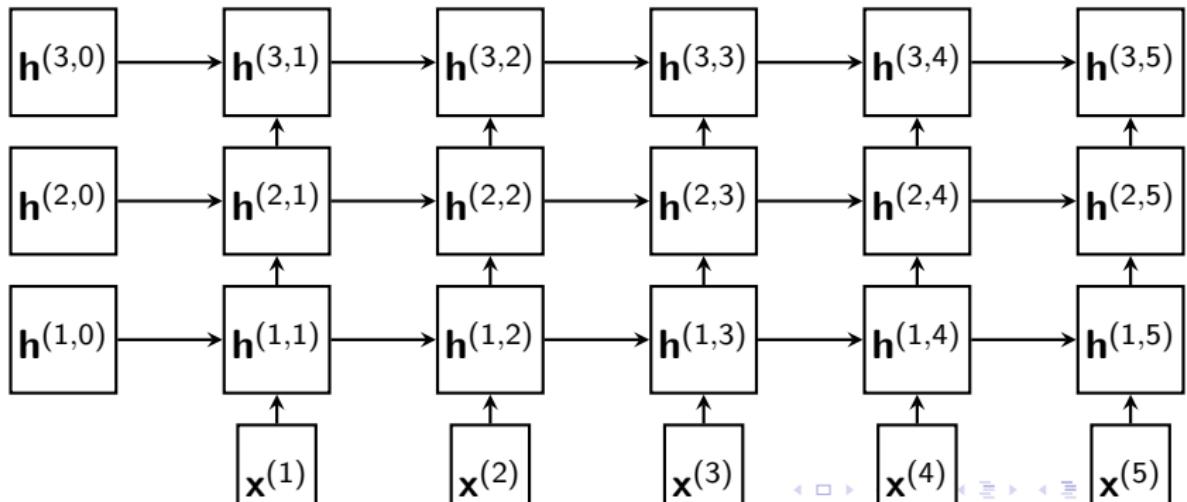
Take a moment to write down any questions you have for the QA session!

Outline

- 1 What are RNNs?
- 2 Vanilla RNN
- 3 Training RNNs
 - Backpropagation through time
 - The vanishing gradients problem
- 4 Gated RNNs
 - LSTM
 - GRU
 - Comparison
- 5 RNN applications
- 6 Extensions: Multi-layer RNNs, bidirectionality
- 7 RNNs with attention

Multi-Layer RNNs

- Stack of several RNNs (Vanilla RNNs, LSTMs, GRUs, etc.)
 - Each RNN in the stack has its own parameters
 - The input vectors of the i 'th RNN are the hidden states of the $i - 1$ 'th RNN
 - The input vectors to the first RNN are the word embeddings, as usual
 - We can output the hidden states of the last RNN, or a combination (concatenation, average, etc.) of the states of all RNNs.



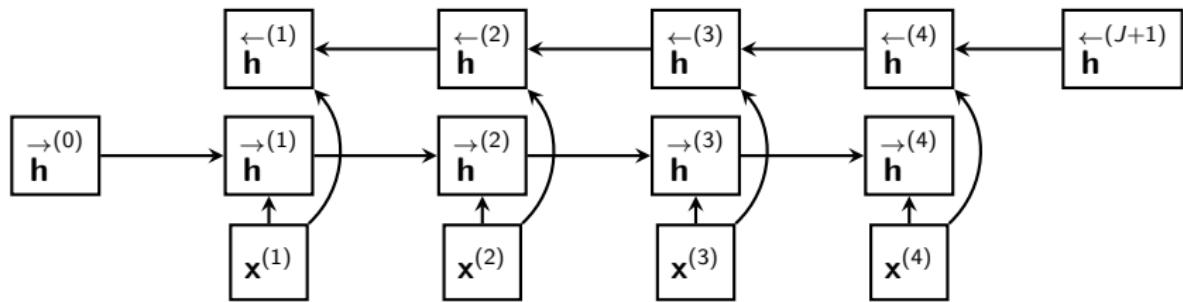
Bidirectional RNNs

- Two RNNs with separate parameters $\theta^{\rightarrow}, \theta^{\leftarrow}$
- Let $x^{(1)} \dots x^{(J)}$ be our input
- Let $h^{\rightarrow(0)} = h^{\leftarrow(J+1)} = \{0\}^d$ be our initial states.
- The forward RNN runs left-to-right over the input:

$$h^{\rightarrow(j)} = f(x^{(j)}, h^{\rightarrow(j-1)}; \theta^{\rightarrow})$$

- The backward RNN runs right-to-left over the input:

$$h^{\leftarrow(j)} = f(x^{(j)}, h^{\leftarrow(j+1)}; \theta^{\leftarrow})$$



Bidirectional RNNs

- The bidirectional RNN yields two sequences of hidden states:

$$(\overset{\rightarrow}{\mathbf{h}}^{(1)} \dots \overset{\rightarrow}{\mathbf{h}}^{(J)}, \overset{\leftarrow}{\mathbf{h}}^{(1)} \dots \overset{\leftarrow}{\mathbf{h}}^{(J)})$$

- Question:** If we are dealing with a sentence classification task, which states should we use to represent the sentence?

► Concatenate $[\overset{\rightarrow}{\mathbf{h}}^{(j)} ; \overset{\leftarrow}{\mathbf{h}}^{(1)}]$, because they have “seen” the entire sentence

- For tagging task, represent the j 'th word as $[\overset{\rightarrow}{\mathbf{h}}^{(j)} ; \overset{\leftarrow}{\mathbf{h}}^{(j)}]$

- Question:** Can we use a bidirectional RNN for autoregressive language modeling?

► No. In autoregressive language modeling, future inputs must be unknown to the model (since we want to learn to predict them).
► We could train two separate autoregressive RNNs (one per direction), but we cannot combine their hidden states before making a prediction

- In sequence-to-sequence (e.g., Machine Translation), the encoder can be bidirectional, but the decoder cannot (same reason)

Questions?

Take a moment to write down any questions you have for the QA session!

Outline

- 1 What are RNNs?
- 2 Vanilla RNN
- 3 Training RNNs
 - Backpropagation through time
 - The vanishing gradients problem
- 4 Gated RNNs
 - LSTM
 - GRU
 - Comparison
- 5 RNN applications
- 6 Extensions: Multi-layer RNNs, bidirectionality
- 7 RNNs with attention

Limitations of RNNs

- At a given point in time j , the information about all past inputs $x^{(1)} \dots x^{(j)}$ is “crammed” into the state $\mathbf{h}^{(j)}$ (and $\mathbf{c}^{(j)}$ for an LSTM)
- So for long sequences, the state becomes a bottleneck
- For Machine Translation, this means that the source sentence is read exactly once, condensed into a single vector, and then the target sentence is produced.
- **Question:** Is this how you would translate a sentence?
 - ▶ A human translator might look at the source sentence multiple times, and translate it “bit by bit”
 - ▶ Attention is meant to mimick this process
- Proposed by Bahdanau et al. 2015, developed into stand-alone architecture (“Transformer”) by Vaswani et al. 2017
- Currently the most popular architecture for NLP
- This week: Attention as a way to make RNNs better
- Next week: Transformers

Attention: The basic recipe

- **Ingredients:**

- One query vector: $\mathbf{q} \in \mathbb{R}^{d_q}$
- J key vectors: $\mathbf{K} \in \mathbb{R}^{J \times d_k}; (\mathbf{k}_1 \dots \mathbf{k}_J)$
- J value vectors: $\mathbf{V} \in \mathbb{R}^{J \times d_v}; (\mathbf{v}_1 \dots \mathbf{v}_J)$
- Scoring function $a : \mathbb{R}^{d_q} \times \mathbb{R}^{d_k} \rightarrow \mathbb{R}$
 - ▶ Maps a query-key pair to a scalar (“score”)
 - ▶ a may be parametrized by parameters θ_a

Attention: The basic recipe

- **Step 1:** Apply a to \mathbf{q} and all keys \mathbf{k}_j to get scores (one per key):

$$\mathbf{e} = \begin{bmatrix} e_1 \\ \vdots \\ e_J \end{bmatrix} = \begin{bmatrix} a(\mathbf{q}, \mathbf{k}_1) \\ \vdots \\ a(\mathbf{q}, \mathbf{k}_J) \end{bmatrix}$$

- **Step 2:** Turn \mathbf{e} into probability distribution with the softmax function

$$\alpha_j = \frac{\exp(e_j)}{\sum_{j'=1}^J \exp(e_{j'})}$$

- **Step 3:** α -weighted sum over \mathbf{V} yields one \mathbb{R}^{d_v} -dimensional output vector \mathbf{o} :

$$\mathbf{o} = \sum_{j=1}^J \alpha_j \mathbf{v}_j$$

Attention in Bahdanau et al., 2015

- Source sentence: $(\mathbf{x}_1 \dots \mathbf{x}_{T_x})$, encoded as hidden states $(\mathbf{h}_1 \dots \mathbf{h}_{T_x})$ by encoder RNN
- Decoder is defined as:

The hidden state s_i of the decoder given the annotations from the encoder is computed by

$$s_i = (1 - z_i) \circ s_{i-1} + z_i \circ \tilde{s}_i,$$

where

$$\begin{aligned}\tilde{s}_i &= \tanh(W_E y_{i-1} + U [r_i \circ s_{i-1}] + C c_i) \\ z_i &= \sigma(W_z E y_{i-1} + U_z s_{i-1} + C_z c_i) \\ r_i &= \sigma(W_r E y_{i-1} + U_r s_{i-1} + C_r c_i)\end{aligned}$$

From Bahdanau et al. 2015: Neural Machine Translation by Jointly Learning to Align and Translate

- **Exercise:** Recognize the architecture (hint: they don't use boldface)
 - ▶ This is a GRU
 - ▶ s_i are the states (which we called $\mathbf{h}^{(i)}$)
 - ▶ $E_{y_{i-1}}$ is the embedding of the word that was/should have been predicted at the previous time step (oracle or decoded input)
 - ▶ W_* and U_* are parameter matrices (which we called $\mathbf{W}^{(*)}$ and $\mathbf{V}^{(*)}$), and they drop the bias
 - ▶ C_* seem to be additional parameter matrices ... but where does the vector c_i come from?

Attention in Bahdanau et al., 2015

The context vector c_i is, then, computed as a weighted sum of these annotations h_j :

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \quad (5)$$

The weight α_{ij} of each annotation h_j is computed by

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

where

$$e_{ij} = a(s_{i-1}, h_j)$$

From Bahdanau et al. 2015: Neural Machine Translation by Jointly Learning to Align and Translate

- Each equation corresponds to one step from our attention recipe.

Exercise: Figure out which equation is which step.

- ▶ The first equation is step 3. It defines the output vector c_i (**o** in our recipe) as an attention-weighted sum over encoder states. So the encoder states h_j are our value vectors (**v_j** in our recipe).
- ▶ The second equation is step 2.
- ▶ The third equation is step 1, which defines the raw scores.
 - ★ s_{i-1} (previous decoder state) is the query vector (**q** in our recipe)
 - ★ The encoder states h_j are also our key vectors (so $k_j = v_j$ in this case)

Attention in Bahdanau et al., 2015

- Scoring function is a Feed-Forward Net:

$$a(s_{i-1}, h_j) = v_a^\top \tanh(W_a s_{i-1} + U_a h_j),$$

Bahdanau et al. 2015: Neural Machine Translation by Jointly Learning to Align and Translate (appendix)

What does attention learn?

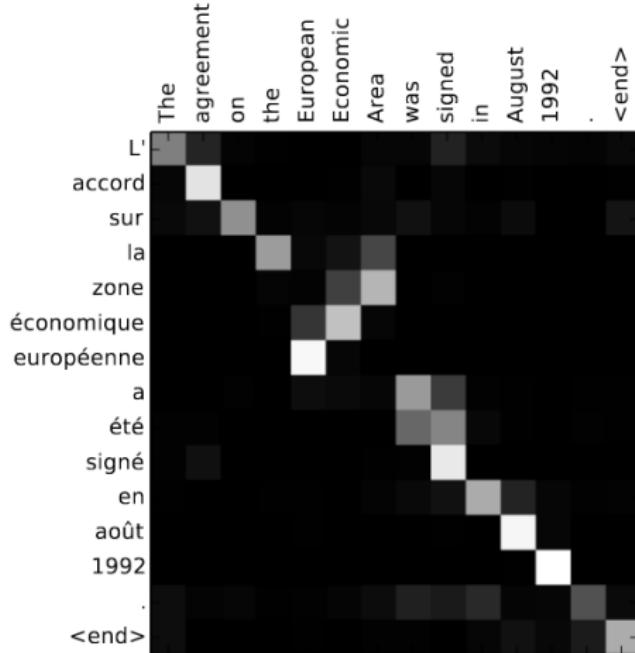


Figure from Bahdanau et al. 2015: Neural Machine Translation by Jointly Learning to Align and Translate.

Effect on translation quality for long sentences

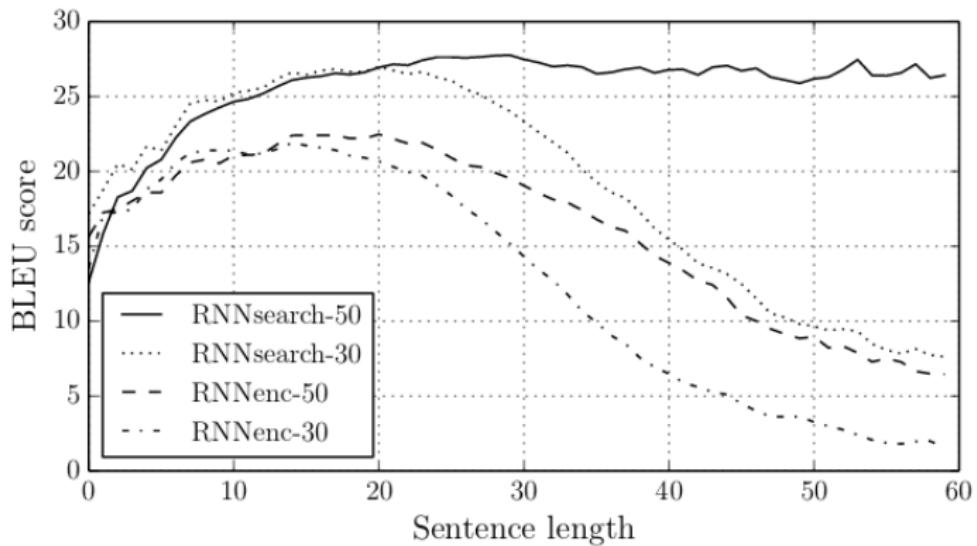


Figure from Bahdanau et al. 2015: Neural Machine Translation by Jointly Learning to Align and Translate.

- RNNsearch-30 and RNNsearch-50 are the attention models
- RNNenc-30 and RNNenc-50 are the baseline models without attention
- 30 and 50 are the state dimensionalities

- Important to note: The Bahdanau model is still an RNN, just with attention on top.
- Next week, we get rid of the RNN completely and introduce the attention-only Transformer architecture.

Questions?

Take a moment to write down any questions you have for the QA session!

(Self-)Attention and the Transformer

Deep Learning for NLP: Lecture 8

Centrum für Informations- und Sprachverarbeitung
Ludwig-Maximilians-Universität München
poerner@cis.uni-muenchen.de

December 23, 2020

Outline

1 Recap: Attention

2 Transformer

- Cross-attention and self-attention
- Parallelized attention
- Multi-layer attention
- Multi-head attention
- Masked self-attention
- Residual connections and layer normalization
- The Transformer architecture
- Position encodings

Outline

1 Recap: Attention

2 Transformer

- Cross-attention and self-attention
- Parallelized attention
- Multi-layer attention
- Multi-head attention
- Masked self-attention
- Residual connections and layer normalization
- The Transformer architecture
- Position encodings

Limitations of RNNs

- In an RNN, at a given point in time j , the information about all past inputs $x^{(1)} \dots x^{(j)}$ is “crammed” into the state vector $\mathbf{h}^{(j)}$ (and $\mathbf{c}^{(j)}$ for an LSTM)
- So for long sequences, the state becomes a bottleneck
- Especially problematic in encoder-decoder models (e.g., for Machine Translation)
- Solution: Attention (Bahdanau et al., 2015) – an architectural modification of the RNN encoder-decoder that allows the model to “attend to” past encoder states

Attention: The basic recipe

- **Ingredients:**

- One query vector: $\mathbf{q} \in \mathbb{R}^{d_q}$
- J key vectors: $\mathbf{K} \in \mathbb{R}^{J \times d_k}; (\mathbf{k}_1 \dots \mathbf{k}_J)$
- J value vectors: $\mathbf{V} \in \mathbb{R}^{J \times d_v}; (\mathbf{v}_1 \dots \mathbf{v}_J)$
- Scoring function $a : \mathbb{R}^{d_q} \times \mathbb{R}^{d_k} \rightarrow \mathbb{R}$
 - ▶ Maps a query-key pair to a scalar (“score”)
 - ▶ a may be parametrized by parameters θ_a

Attention: The basic recipe

- **Step 1:** Apply a to \mathbf{q} and all keys \mathbf{k}_j to get scores (one per key):

$$\mathbf{e} = \begin{bmatrix} e_1 \\ \vdots \\ e_J \end{bmatrix} = \begin{bmatrix} a(\mathbf{q}, \mathbf{k}_1; \theta_a) \\ \vdots \\ a(\mathbf{q}, \mathbf{k}_J; \theta_a) \end{bmatrix}$$

- **Step 2:** Turn \mathbf{e} into a probability distribution with the softmax function

$$\alpha_j = \frac{\exp(e_j)}{\sum_{j'=1}^J \exp(e_{j'})}$$

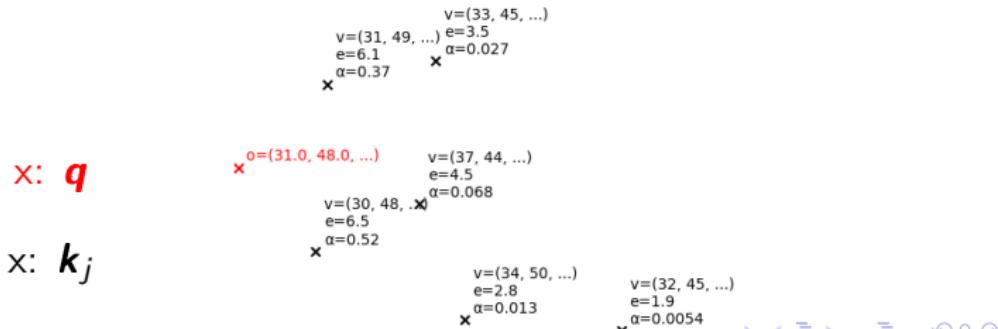
- ▶ Note that $\sum_j \alpha_j = 1$
- **Step 3:** α -weighted sum over \mathbf{V} yields one d_v -dimensional output vector:

$$\mathbf{o} = \sum_{j=1}^J \alpha_j \mathbf{v}_j$$

- ▶ Intuition: α_j is how much “attention” the model pays to \mathbf{v}_j when computing \mathbf{o} .

Attention: An analogy

- We have J weather stations on a map
- $K \in \mathbb{R}^{J \times 2}$ are their geolocations (x, y coordinates)
- $V \in \mathbb{R}^{J \times d_v}$ are their current weather conditions (temperature, humidity, etc.)
- $q \in \mathbb{R}^2$ is a new geolocation for which we want to estimate weather conditions
- e_j is the relevance of the j 'th station (e.g., $e_j = a(q, k_j) = \frac{1}{\|q - k_j\|_2}$), and α_j is e_j as a probability
- o : a weighted sum of all known weather conditions, where stations that have a small distance (high α) have a higher weight



Attention in neural networks

- Contrary to our geolocation example, the q , k_j and v_j vectors of a neural network are produced as a function of the input and some trainable parameters
- So the *model* learns which keys are relevant for which queries, based on the training data and loss function

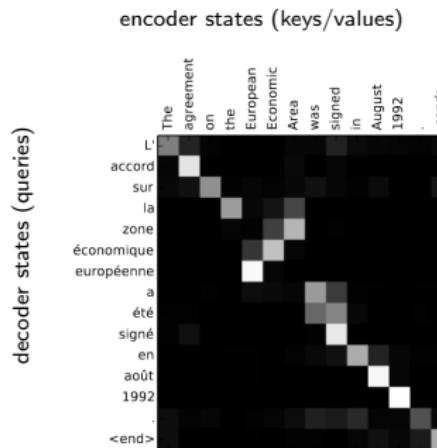


Figure from Bahdanau et al. 2015: Neural Machine Translation by Jointly Learning to Align and Translate.

Attention in neural networks

- No (or few) assumptions are baked into the architecture (no notion of which words are neighbors in the sentence, sequentiality, etc.)
- The lack of prior knowledge often means that the Transformer requires more training data than an RNN/CNN to achieve a certain performance
- But when presented with sufficient data, it usually outperforms them
- After Christmas: Transfer learning as a way to pretrain Transformers on **lots** of data

Questions?

Take a moment to write down any questions you have for the QA session!

Outline

1 Recap: Attention

2 Transformer

- Cross-attention and self-attention
- Parallelized attention
- Multi-layer attention
- Multi-head attention
- Masked self-attention
- Residual connections and layer normalization
- The Transformer architecture
- Position encodings

- The Bahdanau model is still an RNN, just with attention on top.
- Architecture that consists of attention only: Transformer (Vaswani et al. (2017), “Attention is all you need”)

The Transformer architecture (sequence-to-sequence)

(For simpler problems (e.g., classification, tagging), you would simply use the encoder.)

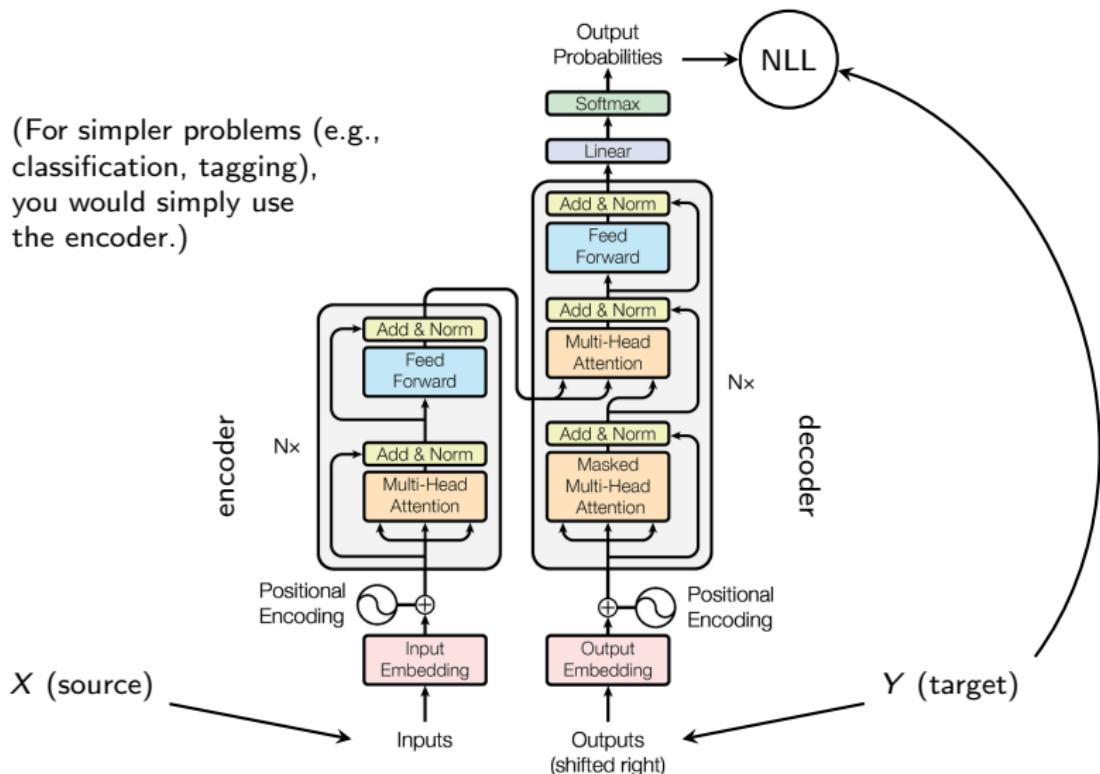


Figure from Vaswani et al. 2017: Attention is all you need

Outline

1 Recap: Attention

2 Transformer

- Cross-attention and self-attention
- Parallelized attention
- Multi-layer attention
- Multi-head attention
- Masked self-attention
- Residual connections and layer normalization
- The Transformer architecture
- Position encodings

Cross-attention and self-attention

- We can use attention on many different “things”, including:
 - ▶ The pixels of images
 - ▶ The nodes of knowledge graphs
 - ▶ The words of a vocabulary
 - ▶ ...
- Here, we focus on scenarios where the query, key and value vectors represent tokens (e.g., words, characters, etc.) in sequences (e.g., sentences, paragraphs, etc.).
- Cross-attention:
 - ▶ Let $X = (x_1 \dots x_{J_x})$, $Y = (y_1 \dots y_{J_y})$ be two sequences (e.g., source and target in a sequence-to-sequence problem)
 - ▶ The query vectors represent tokens in Y and the key/value vectors represent tokens in X (“ Y attends to X ”)
- Self-attention:
 - ▶ There is only one sequence $X = (x_1 \dots x_J)$
 - ▶ The query, key and value vectors represent tokens in X (“ X attends to itself”)

Cross-attention

- Here, we describe cross-attention. Self-attention can easily be derived by assuming $\mathbf{X} = \mathbf{Y}$.
- Let $\mathbf{X} \in \mathbb{R}^{J_x \times d_x}$, $\mathbf{Y} \in \mathbb{R}^{J_y \times d_y}$ be representations of X, Y (e.g., stacked word embeddings, or the outputs of a previous layer)
- Let $\theta = \{\mathbf{W}^{(q)} \in \mathbb{R}^{d_y \times d_q}, \mathbf{W}^{(k)} \in \mathbb{R}^{d_x \times d_k}, \mathbf{W}^{(v)} \in \mathbb{R}^{d_x \times d_v}\}$ be trainable weight matrices
- We transform \mathbf{Y} into a matrix of query vectors:

$$\mathbf{Q} = \mathbf{Y} \mathbf{W}^{(q)}$$

- We transform \mathbf{X} into matrices of key and value vectors:

$$\mathbf{K} = \mathbf{X} \mathbf{W}^{(k)}; \quad \mathbf{V} = \mathbf{X} \mathbf{W}^{(v)}$$

Cross-attention

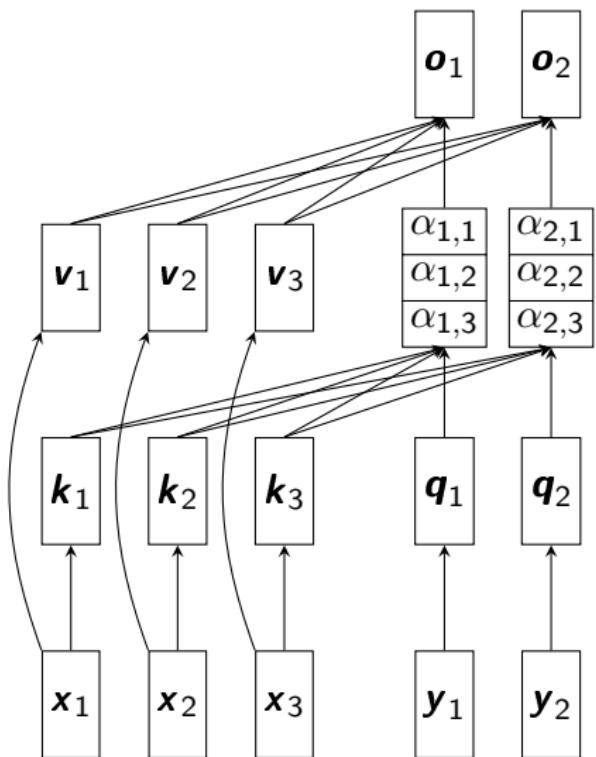
- To calculate the e scores (step 1 of the basic recipe), Vaswani et al. use a parameter-less scaled dot product instead of Bahdanau's complicated FFN:

$$e_{j,j'} = a(\mathbf{q}_j, \mathbf{k}_{j'}) = \frac{\mathbf{q}_j^T \mathbf{k}_{j'}}{\sqrt{d_k}}$$

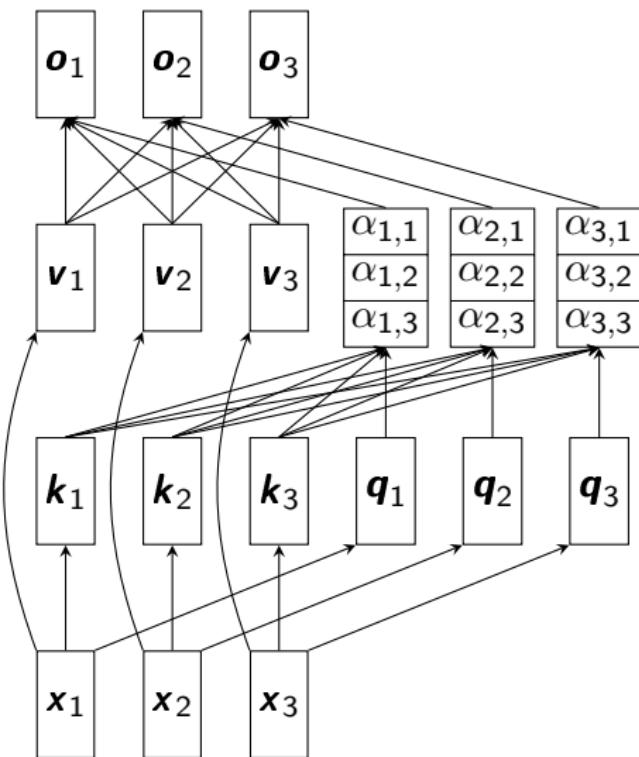
- Note: This requires that $d_q = d_k$
- Attention weights and outputs are defined like before (steps 2 and 3 of the basic recipe):

$$\alpha_{j,j'} = \frac{\exp(e_{j,j'})}{\sum_{j''=1}^{J_x} \exp(e_{j,j''})}$$

$$\mathbf{o}_j = \sum_{j'=1}^{J_x} \alpha_{j,j'} \mathbf{v}_{j'}$$



Cross-attention



Self-attention

Outline

1 Recap: Attention

2 Transformer

- Cross-attention and self-attention
- Parallelized attention**
- Multi-layer attention
- Multi-head attention
- Masked self-attention
- Residual connections and layer normalization
- The Transformer architecture
- Position encodings

Parallelized attention

- We want to apply our attention recipe to every query vector \mathbf{q}_j
- We could simply loop over all time steps $1 \leq j \leq J_y$ and calculate each \mathbf{o}_j independently.
- Then stack all \mathbf{o}_j into an output matrix $\mathbf{O} \in \mathbb{R}^{J_y \times d_v}$
- But a loop does not use the GPU's capacity for parallelization
- So it might be unnecessarily slow

Parallelized attention

- Do some inputs (e.g., \mathbf{q}_j) depend on previous outputs (e.g., \mathbf{o}_{j-1})? If not, we can parallelize the loop into a single function:

$$\mathbf{O} = \mathcal{F}^{\text{attn}}(\mathbf{X}, \mathbf{Y}; \theta)$$

- Attention in Transformers is usually parallelizable, unless we are doing autoregressive inference (more on that later).
- By the way: The Bahdanau model is not parallelizable in this way, because s_i (a.k.a. the query of the $i+1$ 'st step) depends on c_i (a.k.a. the attention output of the i 'th step), see last lecture:

The hidden state s_i of the decoder given the annotations from the encoder is computed by

$$s_i = (1 - z_i) \circ s_{i-1} + z_i \circ \tilde{s}_i,$$

where

$$\tilde{s}_i = \tanh(W E y_{i-1} + U [r_i \circ s_{i-1}] + C c_i)$$

$$z_i = \sigma(W_z E y_{i-1} + U_z s_{i-1} + C_z c_i)$$

$$r_i = \sigma(W_r E y_{i-1} + U_r s_{i-1} + C_r c_i)$$

Parallelized scaled dot product attention

- **Step 1:** The parallel application of the scaled dot product to all query-key pairs can be written as:

$$E = \frac{QK^T}{\sqrt{d_k}}; \quad E \in \mathbb{R}^{J_y \times J_x}$$

$$\begin{array}{c} \rightarrow keys \rightarrow \\ \downarrow queries \begin{bmatrix} e_{1,1} & \dots & e_{1,J_x} \\ \vdots & \ddots & \vdots \\ e_{J_y,1} & \dots & e_{J_y,J_x} \end{bmatrix} = \frac{1}{\sqrt{d_k}} \begin{bmatrix} - & q_1 & - \\ - & \vdots & - \\ - & q_{J_y} & - \end{bmatrix} \begin{bmatrix} | & & | \\ k_1 & \dots & k_{J_x} \\ | & & | \end{bmatrix} \end{array}$$

Parallelized scaled dot product attention

- **Step 2:** Softmax with normalization over the second axis (key axis):

$$\alpha_{j,j'} = \frac{\exp(e_{j,j'})}{\sum_{j''=1}^{J_x} \exp(e_{j,j''})}$$

```
>>> A = np.exp(E) / np.exp(E).sum(axis=-1, keepdims=True)
```

- Let's call this new normalized matrix $\mathbf{A} \in (0, 1)^{J_y \times J_x}$
- The rows of \mathbf{A} , denoted α_j , are probability distributions (one α_j per q_j)
- **Step 3:** Weighted sum

$$\mathbf{O} = \mathbf{A} \mathbf{V}; \mathbf{O} \in \mathbb{R}^{J_y \times d_v}$$

$$\begin{array}{c} \rightarrow d_v (\text{value dims}) \rightarrow \\ \downarrow \\ \text{queries} \left[\begin{array}{ccc} o_{1,1} & \dots & o_{1,d_v} \\ \vdots & \ddots & \vdots \\ o_{J_y,1} & \dots & o_{J_y,d_v} \end{array} \right] = \left[\begin{array}{ccc} - & \alpha_1 & - \\ & \vdots & \\ - & \alpha_{J_y} & - \end{array} \right] \left[\begin{array}{ccc} | & & | \\ \mathbf{v}_{:,1} & \dots & \mathbf{v}_{:,d_v} \\ | & & | \end{array} \right] \end{array}$$

... as a one-liner

$$O = \mathcal{F}^{\text{attn}}(\mathbf{X}, \mathbf{Y}; \theta) = \text{softmax}\left(\frac{(\mathbf{Y} \mathbf{W}^{(q)})(\mathbf{X} \mathbf{W}^{(k)})^T}{\sqrt{d_k}}\right)(\mathbf{X} \mathbf{W}^{(v)})$$

- GPUs like matrix multiplications → usually a lot faster than RNN!
- But: The memory requirements of \mathbf{E} and \mathbf{A} are $\mathcal{O}(J_y J_x)$
- A length up to about 500 is usually ok on a medium-sized GPU (and most sentences are shorter than that anyway).
- But when we consider inputs that span several sentences (e.g., paragraphs or whole documents), we need tricks to reduce memory. These are beyond the scope of this lecture.

Questions?

Take a moment to write down any questions you have for the QA session!

Outline

1 Recap: Attention

2 Transformer

- Cross-attention and self-attention
- Parallelized attention
- Multi-layer attention**
- Multi-head attention
- Masked self-attention
- Residual connections and layer normalization
- The Transformer architecture
- Position encodings

Multi-layer attention

- Sequential application of several attention layers, with separate parameters $\{\theta^{(1)} \dots \theta^{(N)}\}$
- In Transformer: sequential application of Transformer blocks
- There are some additional position-wise layers inside the Transformer block, i.e., $O^{(n)}$ undergoes some additional transformations before becoming the input to the next Transformer block $n + 1$

Outline

1 Recap: Attention

2 Transformer

- Cross-attention and self-attention
- Parallelized attention
- Multi-layer attention
- Multi-head attention**
- Masked self-attention
- Residual connections and layer normalization
- The Transformer architecture
- Position encodings

Multi-head attention

- Application of several attention layers (“heads”) in parallel
- M sets of parameters $\{\theta^{(1)}, \dots, \theta^{(M)}\}$, with $\theta^{(m)} = \{W^{(m,q)}, W^{(m,k)}, W^{(m,v)}\}$
- For every head, compute in parallel:

$$O^{(m)} = \mathcal{F}^{\text{attn}}(X, Y; \theta^{(m)})$$

- Concatenate all $O^{(m)}$ along their last axis; then down-project the concatenation with an additional parameter matrix $W^{(o)} \in \mathbb{R}^{Md_v \times d_v}$:

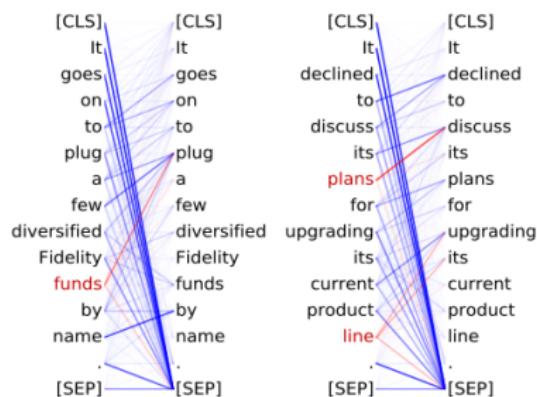
$$O = [O^{(1)}; \dots; O^{(M)}] W^{(o)}$$

Multi-head attention

- Conceptually, multi-head attention is to single-head attention like a filter bank is to a single filter (Lecture 6 on CNNs)
- Division of labor: different heads model different kinds of inter-word relationships

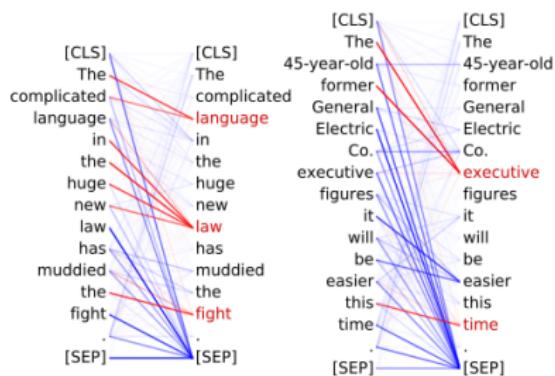
Head 8-10

- Direct objects attend to their verbs
- 86.8% accuracy at the dobj relation



Head 8-11

- Noun modifiers (e.g., determiners) attend to their noun
- 94.3% accuracy at the det relation



Clark et al. (2018): What Does BERT Look At? An Analysis of BERT's Attention

Outline

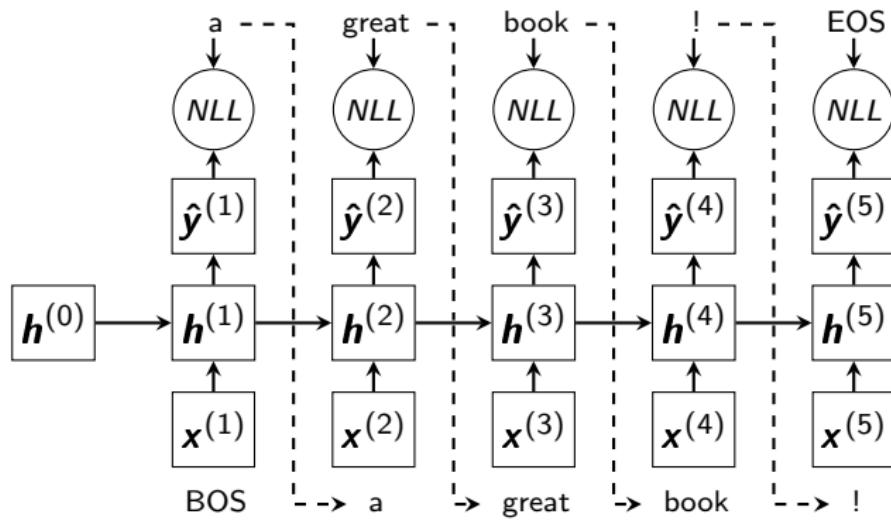
1 Recap: Attention

2 Transformer

- Cross-attention and self-attention
- Parallelized attention
- Multi-layer attention
- Multi-head attention
- **Masked self-attention**
- Residual connections and layer normalization
- The Transformer architecture
- Position encodings

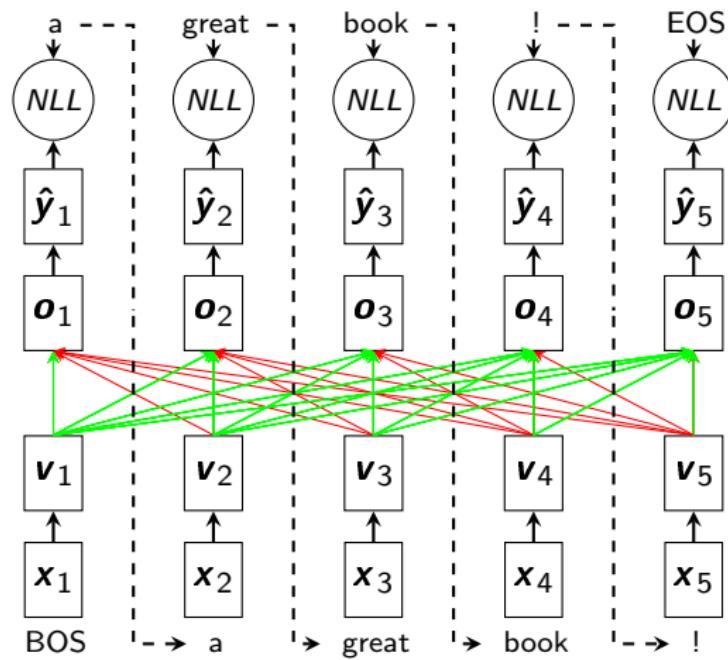
Recap: RNNs for autoregressive language modeling or decoding

- In autoregressive language modeling, or in the decoder of a sequence-to-sequence model, the task is to always predict the next word
- In an RNN, a given state $\mathbf{h}^{(j)}$ depends on past inputs $x^{(1)} \dots x^{(j)}$
- Thus, the RNN is unable to “cheat”:



Self-attention for autoregressive LM & decoding

- With attention, all \mathbf{o}_j depend on all $\mathbf{v}_{j'}$ (and by extension, all $\mathbf{x}_{j'}$).
- This means that the model can easily cheat by looking at future words (red connections)



Masked self-attention

- So when we use self-attention for language modeling or in a sequence-to-sequence decoder, we have to prevent \mathbf{o}_j from attending to any $\mathbf{v}_{j'}$ where $j' > j$.
- **Question:** How can we do that?
- Remember:

$$\mathbf{o}_j = \sum_{j'=1}^J \alpha_{j,j'} \mathbf{v}_{j'}$$
$$\alpha_{j,j'} = \frac{\exp(e_{j,j'})}{\sum_{j''=1}^J \exp(e_{j,j''})}$$

- ▶ By hardcoding $e_{j,j'} = -\infty$ when $j' > j$ (in practice, “ ∞ ” is just a large constant)
- ▶ That way, $\exp(e_{j,j'}) = \alpha_{j,j'} = 0$, so $\mathbf{v}_{j'}$ has no impact on \mathbf{o}_j

Parallelized masked self-attention

- Step 1: Calculate \mathbf{E} like we usually would
- Step 1B:

$$\mathbf{E}^{\text{masked}} = \mathbf{E} \odot \mathbf{M} + \infty \mathbf{M} - \infty; \quad m_{j,j'} = \begin{cases} 1 & \text{if } j' \leq j \\ 0 & \text{otherwise} \end{cases}$$

- Example:

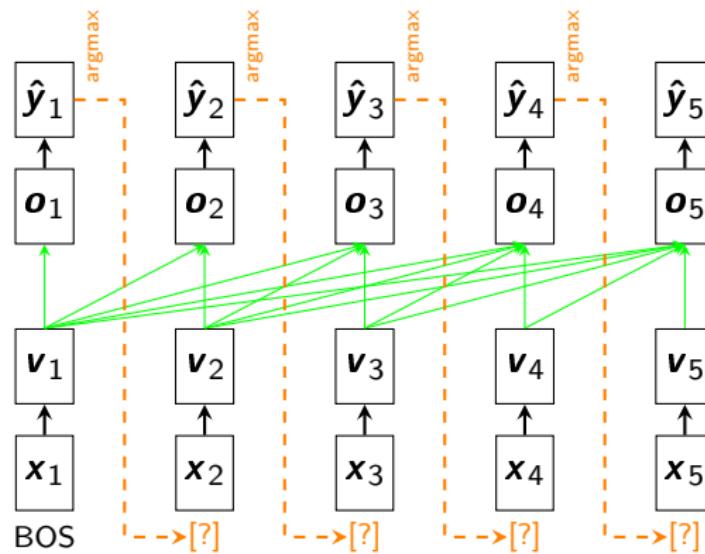
$$\mathbf{E} = \begin{bmatrix} e_{1,1} & e_{1,2} & e_{1,3} \\ e_{2,1} & e_{2,2} & e_{2,3} \\ e_{3,1} & e_{3,2} & e_{3,3} \end{bmatrix}; \quad \mathbf{M} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\mathbf{E}^{\text{masked}} = \begin{bmatrix} e_{1,1} & -\infty & -\infty \\ e_{2,1} & e_{2,2} & -\infty \\ e_{3,1} & e_{3,2} & e_{3,3} \end{bmatrix}; \quad \mathbf{A}^{\text{masked}} = \begin{bmatrix} 1 & 0 & 0 \\ \alpha_{2,1} & \alpha_{2,2} & 0 \\ \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} \end{bmatrix}$$

$$\mathbf{o}_1 = \mathbf{v}_1; \quad \mathbf{o}_2 = \alpha_{2,1} \mathbf{v}_1 + \alpha_{2,2} \mathbf{v}_2; \quad \mathbf{o}_3 = \alpha_{3,1} \mathbf{v}_1 + \alpha_{3,2} \mathbf{v}_2 + \alpha_{3,3} \mathbf{v}_3$$

Autoregressive Transformer at inference time

- During training, when targets are known, we use parallelized masked attention
- At inference time, when we don't know what the targets are, we have to decode the prediction in a loop
- Slower, but at least we don't have to worry about masking anymore



Questions?

Take a moment to write down any questions you have for the QA session!

Outline

1 Recap: Attention

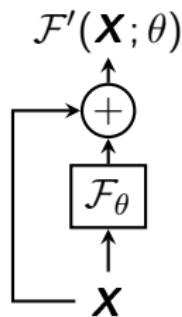
2 Transformer

- Cross-attention and self-attention
- Parallelized attention
- Multi-layer attention
- Multi-head attention
- Masked self-attention
- **Residual connections and layer normalization**
- The Transformer architecture
- Position encodings

Residual connections

- Let \mathcal{F} be a function with parameters θ
- \mathcal{F} with a residual connection:

$$\mathcal{F}'(\mathbf{X}; \theta) = \mathcal{F}(\mathbf{X}; \theta) + \mathbf{X}$$



- Benefits: Information retention (we add to \mathbf{X} but don't replace it)

Layer normalization

- Let $\theta = \{\gamma \in \mathbb{R}^d, \beta \in \mathbb{R}^d\}$ be trainable parameters
- Let $\mathbf{h} \in \mathbb{R}^d$ be an output vector of some layer (e.g., an \mathbf{o}_j vector from an attention layer)
- Then layer normalization calculates:

$$\gamma \odot \frac{\mathbf{h} - \mu}{\sigma} + \beta$$

- where μ, σ are mean and standard deviation over the dimensions of \mathbf{h} :

$$\mu = \frac{1}{d} \sum_{i=1}^d h_i; \quad \sigma = \sqrt{\frac{1}{d} \sum_{i=1}^d (h_i - \mu)^2}$$

- Benefits: Allows us to normalize vectors after every layer; helps against exploding activations on the forward pass
- In the Transformer, layer normalization is applied position-wise, i.e., every \mathbf{o}_j is normalized independently

Outline

1 Recap: Attention

2 Transformer

- Cross-attention and self-attention
- Parallelized attention
- Multi-layer attention
- Multi-head attention
- Masked self-attention
- Residual connections and layer normalization
- **The Transformer architecture**
- Position encodings

The Transformer architecture (sequence-to-sequence)

(For simpler problems (e.g., classification, tagging), you would simply use the encoder.)

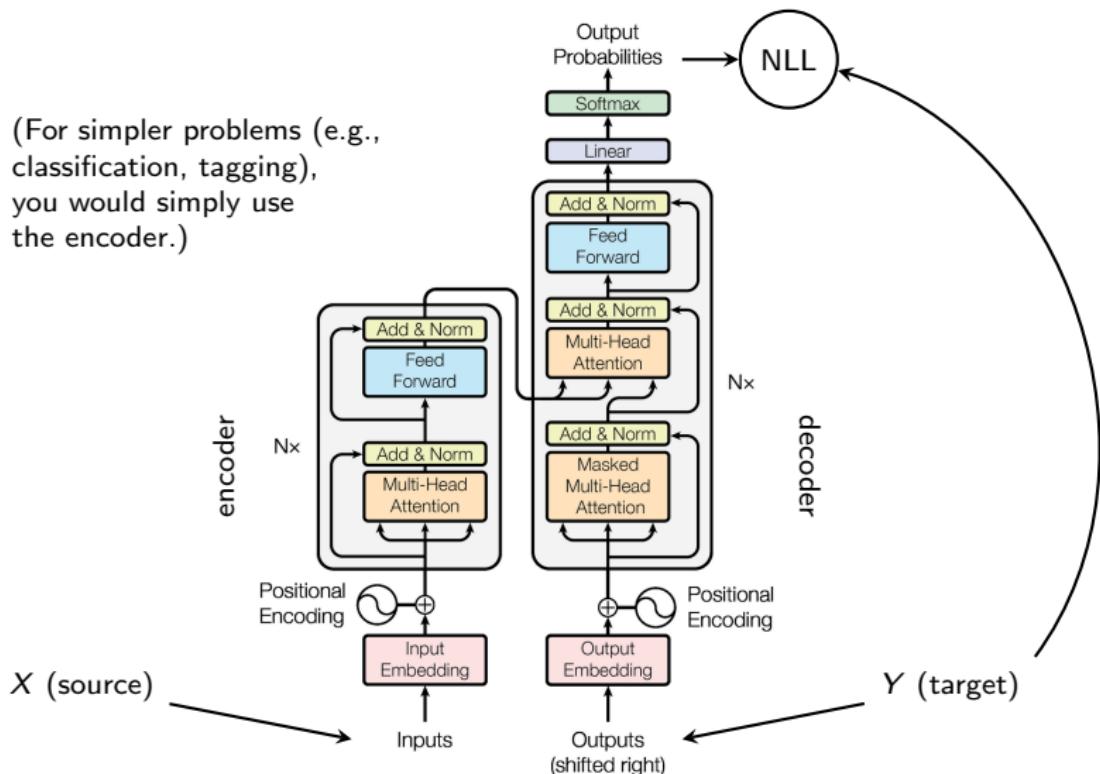


Figure from Vaswani et al. 2017: Attention is all you need

Questions?

Take a moment to write down any questions you have for the QA session!

Outline

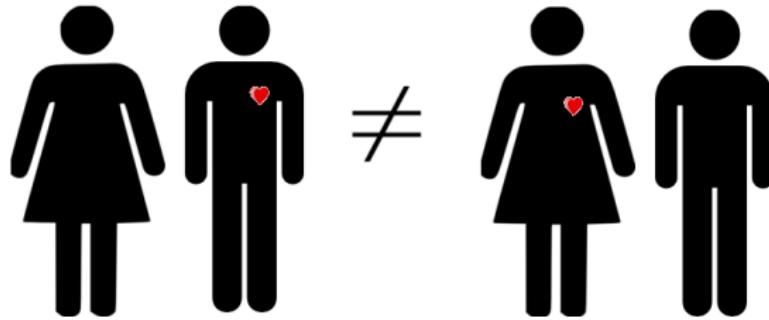
1 Recap: Attention

2 Transformer

- Cross-attention and self-attention
- Parallelized attention
- Multi-layer attention
- Multi-head attention
- Masked self-attention
- Residual connections and layer normalization
- The Transformer architecture
- Position encodings

Can self-attention model word order?

- Our model consists of a self-attention layer on top of a simple word embedding lookup layer. (For simplicity, we only consider one head, but this applies to multi-head attention as well.)
- Let $X^{(1)}$, $X^{(2)}$ be two sentences of the same length J , which contain the same words in a different order
- Example: “john loves mary” vs. “mary loves john”



Can self-attention model word order?

- Let $\mathbf{g} \in \{1, \dots, J\}^J$ be the permutation of the word order between $X^{(1)}$ and $X^{(2)}$. So we know that $j \rightarrow g_j$ is bijective, and $\forall_{j \in \{1, \dots, J\}} x_j^{(1)} = x_{g_j}^{(2)}$
 - ▶ In our example: $\mathbf{g} = [3, 2, 1]^T$
- Can our self-attention layer differentiate between word orders, or is $\forall_{j \in \{1, \dots, J\}} o_j^{(1)} = o_{g_j}^{(2)}$?
- (For example, do `mary` and `mary` get the same vector representation? If so, that is bad news for very basic Natural Language Understanding capabilities, such as figuring out what is the subject or object of an English verb.)

Can self-attention model word order?

- We want to show that:

$$\forall_{j \in \{1, \dots, J\}} \mathbf{o}_j^{(1)} = \mathbf{o}_{g_j}^{(2)}$$

- Let's start with x_j .
- Our word embedding lookup layer represents x_j as $\mathbf{x}_j = \mathbf{w}_{\mathcal{I}(x_j)}$ (see lecture 5), where \mathcal{I} is a bijective indexing function.
- So it is trivially true that:

$$x_j^{(1)} = x_{g_j}^{(2)} \implies \mathbf{w}_{\mathcal{I}(x_j^{(1)})} = \mathbf{w}_{\mathcal{I}(x_{g_j}^{(2)})} \implies \mathbf{x}_j^{(1)} = \mathbf{x}_{g_j}^{(2)}$$

Can self-attention model word order?

- Definition of \mathbf{o}_j :

$$\mathbf{o}_j = \sum_{j'=1}^J \alpha_{j,j'} \mathbf{v}_{j'}$$

- Since addition is commutative, and the permutation is bijective, it is sufficient to show that:

$$\forall j \in \{1, \dots, J\}, j' \in \{1, \dots, J\} \quad \alpha_{j,j'}^{(1)} \mathbf{v}_{j'}^{(1)} = \alpha_{g_j, g_{j'}}^{(2)} \mathbf{v}_{g_{j'}}^{(2)}$$

- Step 1: Let's show that $\forall j \mathbf{v}_j^{(1)} = \mathbf{v}_{g_j}^{(2)}$

- Definition of \mathbf{v}_j :

$$\mathbf{v}_j = \mathbf{W}^{(v)T} \mathbf{x}_j$$

- Then:

$$\mathbf{x}_j^{(1)} = \mathbf{x}_{g_j}^{(2)} \implies \mathbf{W}^{(v)T} \mathbf{x}_j^{(1)} = \mathbf{W}^{(v)T} \mathbf{x}_{g_j}^{(2)} \implies \mathbf{v}_j^{(1)} = \mathbf{v}_{g_j}^{(2)}$$

Can self-attention model word order?

- Step 2: Let's show that $\forall_{j \in \{1, \dots, J\}, j' \in \{1, \dots, J\}} \alpha_{j,j'}^{(1)} = \alpha_{g_j, g_{j'}}^{(2)}$
- Definition of $\alpha_{j,j'}$:

$$\alpha_{j,j'} = \frac{\exp(e_{j,j'})}{\sum_{j''=1}^J \exp(e_{j,j''})}$$

- Since the sum in the denominator is commutative, and the permutation is bijective, it is sufficient to show that

$$\forall_{j \in \{1, \dots, J\}, j' \in \{1, \dots, J\}} e_{j,j'}^{(1)} = e_{g_j, g_{j'}}^{(2)}$$

Can self-attention model word order?

- Definition of $e_{j,j'}:$

$$e_{j,j'} = \frac{1}{\sqrt{d_k}} \mathbf{q}_j^T \mathbf{k}_{j'} = \frac{1}{\sqrt{d_k}} (\mathbf{W}^{(q)T} \mathbf{x}_j)^T (\mathbf{W}^{(k)T} \mathbf{x}_{j'})$$

- Then:

$$\begin{aligned} \mathbf{x}_j^{(1)} &= \mathbf{x}_{g_j}^{(2)} \wedge \mathbf{x}_{j'}^{(1)} = \mathbf{x}_{g_{j'}}^{(2)} \\ \implies \mathbf{W}^{(q)T} \mathbf{x}_j^{(1)} &= \mathbf{W}^{(q)T} \mathbf{x}_{g_j}^{(2)} \wedge \mathbf{W}^{(k)T} \mathbf{x}_{j'}^{(1)} = \mathbf{W}^{(k)T} \mathbf{x}_{g_{j'}}^{(2)} \\ \implies \mathbf{q}_j^{(1)} &= \mathbf{q}_{g_j}^{(2)} \wedge \mathbf{k}_{j'}^{(1)} = \mathbf{k}_{g_{j'}}^{(2)} \\ \implies \mathbf{q}_j^{(1)T} \mathbf{k}_{j'}^{(1)} &= \mathbf{q}_{g_j}^{(2)T} \mathbf{k}_{g_{j'}}^{(2)} \\ \implies \frac{1}{\sqrt{d_k}} \mathbf{q}_j^{(1)T} \mathbf{k}_{j'}^{(1)} &= \frac{1}{\sqrt{d_k}} \mathbf{q}_{g_j}^{(2)T} \mathbf{k}_{g_{j'}}^{(2)} \\ \implies e_{j,j'}^{(1)} &= e_{g_j,g_{j'}}^{(2)} \end{aligned}$$

Can self-attention model word order?

- So, $\forall_j \mathbf{o}_j^{(1)} = \mathbf{o}_{g_j}^{(2)}$
- In other words: The representation of **mary** is identical to that of **mary**, and the representation of **john** is identical to that of **john**
- **Question:** Can the other layers in the Transformer architecture (feed-forward net, layer normalization) help with the problem?
 - ▶ No, because they apply the same function to all positions.
- **Question:** Would it help to apply more self-attention layers?
 - ▶ No. Since the representations of identical words are still identical in O , the next self-attention layer will have the same problem.
- So... does that mean the Transformer is unusable?
- Luckily not. We just need to ensure that input embeddings of identical words at different positions are not identical.

Position embeddings

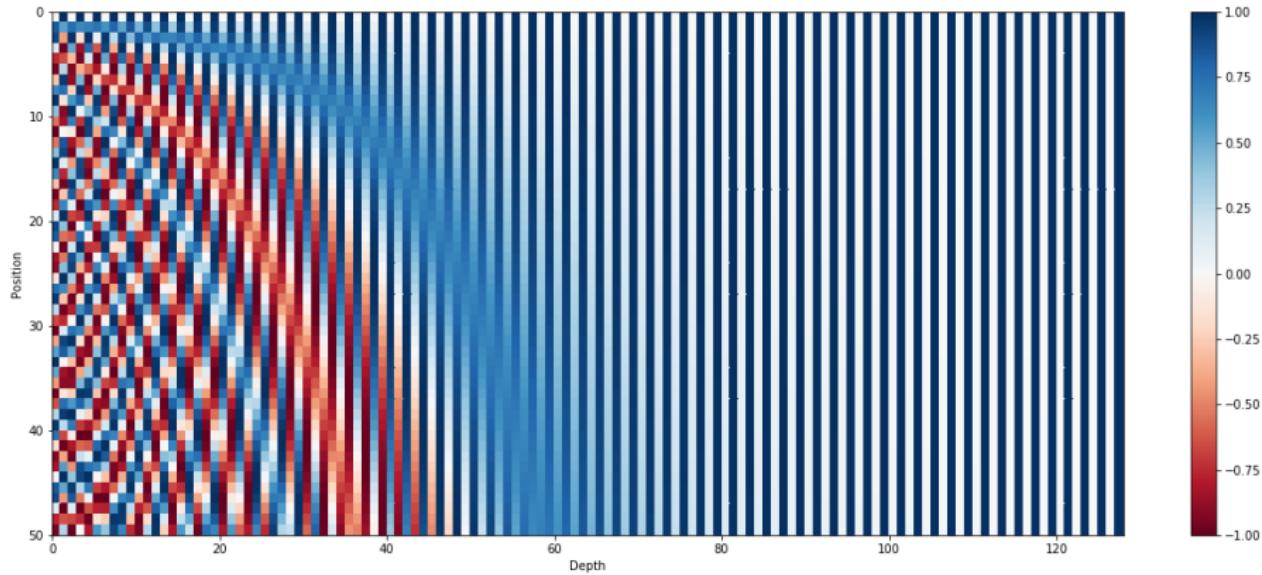
- Add to every input word embedding a position embedding $\mathbf{p} \in \mathbb{R}^d$:
- Input embedding of word “mary” in position j : $x_j = \mathbf{w}_{\mathcal{I}(\text{mary})} + \mathbf{p}_j$

$$\mathbf{w}_{\mathcal{I}(\text{mary})} + \mathbf{p}_j \neq \mathbf{w}_{\mathcal{I}(\text{mary})} + \mathbf{p}_{j'} \text{ if } j \neq j'$$

- Option 1 (Devlin et al., 2018): Trainable position embeddings:
 $\mathbf{P} \in \mathbb{R}^{J^{\max} \times d}$
 - ▶ Disadvantage: Cannot deal with sentences that are longer than J^{\max}
- Option 2 (Vaswani et al., 2017): Sinusoidal position embeddings (deterministic):

$$p_{j,i} = \begin{cases} \sin\left(\frac{j}{10000^{\frac{i}{d}}}\right) & \text{if } i \text{ is even} \\ \cos\left(\frac{j}{10000^{\frac{i-1}{d}}}\right) & \text{if } i \text{ is odd} \end{cases}$$

Sinusoidal position embeddings



https://kazemnejad.com/blog/transformer_architecture_positional_encoding

Questions?

Take a moment to write down any questions you have for the QA session!

Hyperparameter Optimization

Deep Learning for NLP: Lecture 8b

Centrum für Informations- und Sprachverarbeitung
Ludwig-Maximilians-Universität München
poerner@cis.uni-muenchen.de

December 23, 2020

Outline

- 1 Hyperparameters
- 2 Train/validation/test sets
- 3 Hyperparameter configurations and ranges
- 4 How to search

Outline

- 1 Hyperparameters
- 2 Train/validation/test sets
- 3 Hyperparameter configurations and ranges
- 4 How to search

What are hyperparameters?

- Things that may affect the performance of a model on a task, without being a trainable parameter in θ

Hyperparameters related to the model

- input (word embedding) size
- hidden state size (RNN)
- number of filters (CNN)
- filter size (CNN)
- number of heads (Transformer)
- hidden size of queries/keys/values (Transformer)
- number of layers/blocks
- choice of nonlinearity (ReLU vs. tanh vs. ...)
- ...

Hyperparameters related to training

- batch size
- factor for L1, L2 regularization
- dropout probability
- learning rate
- choice of optimizer (SGD vs. Adam vs. Adagrad vs. ...)
- parameters of optimizer (e.g., momentum, ...)
- number of epochs (early stopping)
- ...

What is hyperparameter optimization?

- Also known as “hyperparameter tuning”
- The process of choosing the best hyperparameters for a given architecture and task
- Different from training your regular parameters, because hyperparameters are not optimized by gradient descent

Outline

- 1 Hyperparameters
- 2 Train/validation/test sets
- 3 Hyperparameter configurations and ranges
- 4 How to search

Datasets

- Needed: separate train/validation*/test sets
 - ▶ *a.k.a. dev(velopment), heldout, valid(ation)
- Many existing datasets come with a predefined train/validation/test split; if that is the case, use the predefined split
- Otherwise, you should split the data yourself (randomly!); 80/10/10 is usually fine, unless there is too little data (see cross-validation)
- You should always report the absolute and relative sizes of the sets

Using the validation set

- For every hyperparameter configuration:
 - ▶ Create model
 - ▶ Train model on the training set
 - ▶ Evaluate model on validation set
- Keep the model that had the best performance on the validation set
- Evaluate that model on the test set

Cross-validation

- Useful when there is very little data, so that you cannot afford separate train and validation sets.
- Split training set into N subsets (“folds”)
- For every hyperparameter configuration:
 - ▶ For every fold $1 \leq n \leq N$:
 - ★ Fold n is your validation set, the other folds are your training set
 - ★ Create, train and evaluate the model
 - ▶ Average the validation performance over all folds
- Keep the configuration that had the best average validation performance
- Train a model with that configuration on the full training set (all folds together)
- Measure the model's performance on the test set

The importance of the test set

- Hyperparameter optimization means overfitting to the validation set
- You should never do hyperparameter optimization on the test set
- The test set should only be used once, to evaluate your final model.
Otherwise, your test set is “spoiled”.

Outline

- 1 Hyperparameters
- 2 Train/validation/test sets
- 3 Hyperparameter configurations and ranges
- 4 How to search

- A *hyperparameter configuration* is a specific choice of hyperparameters, e.g.,
 - ▶ batch size = 16
 - ▶ optimizer = Adagrad
 - ▶ hidden size = 50
 - ▶ dropout = 0.5
 - ▶ ...
- Every hyperparameter has a *range*, or a set of permissible values, e.g.,
 - ▶ batch size $\in \{8, 16, 32, \dots\}$
 - ▶ optimizer $\in \{\text{SGD}, \text{Adagrad}, \dots\}$
 - ▶ ...

Defining hyperparameter ranges

- Usually, the range will be informed by your domain expertise:
 - ▶ Lots of data → low risk of overfitting → try big hidden sizes / more layers...
 - ▶ Little data → high risk of overfitting → try low hidden sizes / fewer layers ...
 - ▶ Some values just don't make sense (learning rate of 10000, filter size 1 in a CNN)
- In practice, your range will also be limited by your resources (e.g., GPU memory)
- Always report the range, so that others can reproduce your evaluation.

Discretizing continuous ranges

- Continuous hyperparameters (e.g., hidden size, batch size, number of filters) should be discretized.
- For open-ended hyperparameters, use an (approximate) log scale, e.g.,
 - ▶ hidden size $\in 50, 100, 200, 500, 1000$
 - ▶ batch size $\in 16, 32, 64, 128$
- Categorical hyperparameters (e.g., optimizer or nonlinearity) can be treated as a finite set

Outline

- 1 Hyperparameters
- 2 Train/validation/test sets
- 3 Hyperparameter configurations and ranges
- 4 How to search

How to search

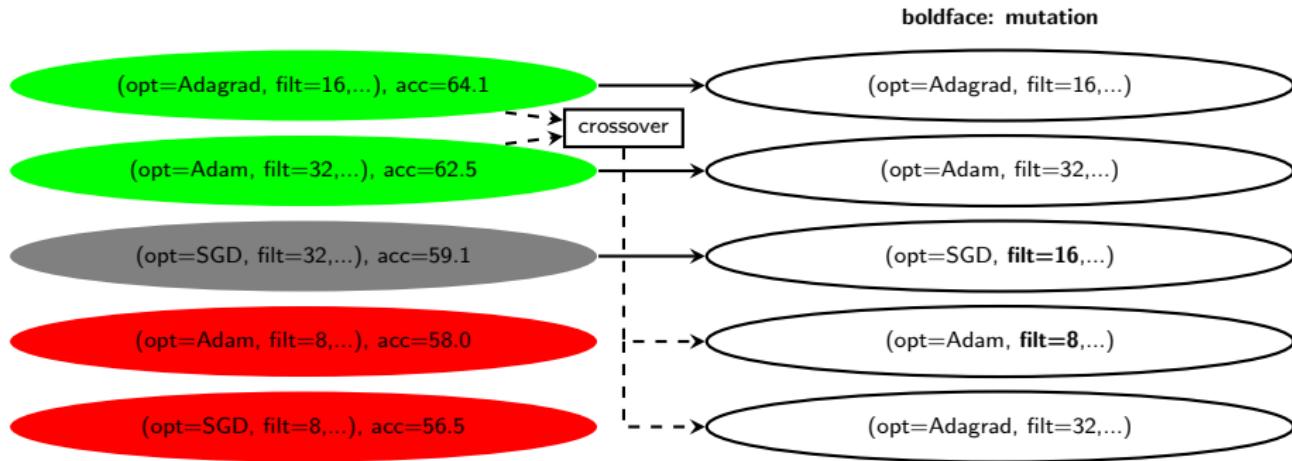
- Brute Force?
 - ▶ Pro: Exhaustive
 - ▶ Con: Search space grows exponentially.
- Intuition: Some hyperparameters have a big effect, others have a small effect. A configuration that gets the first group right will be almost as good as the optimal configuration.
- Uniform sampling?
 - ▶ Pro: Easy to implement, good results in practice
 - ▶ Con: Will waste resources on configurations that have little chance of success (e.g., if we have previously found that all configurations with a hidden size of 20 produce bad results, we should stop sampling that particular value)

Shifting window search (credit to Ben Roth)



- For every continuous hyperparameter, define a small window of the range
- Example: full range = $\{4, 8, 16, 32, 64, 128, 256\}$, window = $\{16, 32, 64\}$
- For N iterations:
 - ▶ Use the random sampling method N' times, but sample only from the windows.
 - ▶ For each hyperparameter, identify the value that led to the best performance.
 - ▶ Shift that window, so that the best value lies in the center
- Not applicable to hyperparameters whose values are unordered sets (i.e., categorical hyperparameters)

Evolutionary algorithm



- Start with a random set of configurations (population)
- For N iterations:
 - ▶ Delete the **worst-performing** configurations (survival of the fittest)
 - ▶ Randomly combine **best-performing** hyperparameter configurations to produce “children” (crossover).
 - ▶ Inject random changes to explore new possibilities (**mutations**).

How to search

- For reproducibility, you should always provide details about your method of hyperparameter search
- That includes how many iterations you did, and any search-specific parameters (e.g., mutation probability, ...)

Questions?

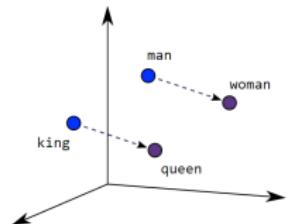
Take a moment to write down any questions you have for the QA session!

Chapter 9: Transfer Learning & Tokenization

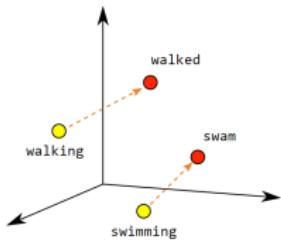
Matthias Aßenmacher

January 13, 2021

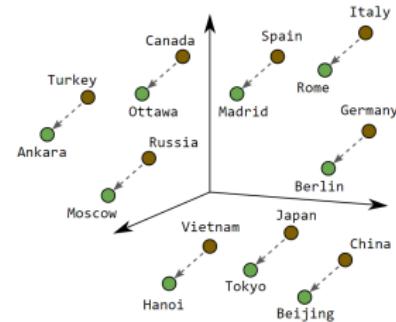
Recap: Word vectors



Male-Female



Verb Tense



Country-Capital

Source: *google*

- Information is encoded in (pre-)trained word embeddings
- Embeddings are used for tasks external to the training corpus

What is Transfer Learning?

Wikipedia says:

"Transfer learning is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem."

How it works with word2vec

- Train word2vec on some "fake task" (DBOW or Skip-gram)
- Extract the stored knowledge (a.k.a. embedding)
or: Directly download embeddings from the web
- Perform a different (supervised) task using the embeddings

A remark on vocabulary & tokenization

Challenges:

- If no embedding for a word exists, it cannot be represented.
- *Workaround:*
 - ▶ Train subword (character n-gram) embeddings
 - ▶ Represent OOV word as combination of them
- This is already a special case of **Tokenization**

Tokenization examples:

- Whitespace tokenization
- N-grams
- Character n-grams
- Characters

Fine-grained tokenization

Difficulties:

- When using embeddings (or other models/methods) for transferring knowledge, one has to stick to this method's tokenization scheme.
- Using words as tokens leads to vocabulary sizes of easily $> 100k$, which is undesirable.
- Characters as tokens lead to a very small vocabulary size but aggravate capturing meaning.
- Using (sets of) n-grams is kind of heuristic.

Smart alternatives:

- BytePair encoding ► Gage (1994) ► Sennrich et al. (2016)
- WordPiece ► Schuster & Nakajima (2012) ► Wu et al. (2016)
- SentencePiece ► Kudo et al. (2018)

BytePair encoding (BPE)

Data compression algorithm

► Gage (1994)

- Considering data on a *byte*-level
- Looking at pairs of bytes:
 - ① Count the occurrences of all byte pairs
 - ② Find the most frequent byte pair
 - ③ Replace it with an unused byte
- Repeat this process until no further compression is possible

Open-vocabulary neural machine translation

► Sennrich et al. (2016)

- Translation as an open-vocabulary problem
- Word-level NMT models:
 - ▶ Handling out-of-vocabulary word by using back-off dictionaries
 - ▶ Unable to translate or generate previously unseen words
- Subword-level models alleviate this problem

BytePair encoding (BPE)

Adapt BPE for word segmentation

► Sennrich et al. (2016)

- Goal: Represent an open vocabulary by a vocabulary of fixed size
→ Use variable-length character sequences
- Looking at pairs of characters:
 - ① Initialize the vocabulary with all characters plus end-of-word token
 - ② Count occurrences and find the most frequent character pair,
e.g. "A" and "B" (⚠ Word boundaries are **not** crossed)
 - ③ Replace it with the new token "AB"
- Only one hyperparameter: Vocabulary size
(Initial vocabulary + Specified no. of merge operations)
→ Repeat this process until given $|V|$ is reached

Voice Search for Japanese and Korean ▶ Schuster & Nakajima (2012)

- *Specific Problems:*
 - ▶ Asian languages have larger basic character inventories compared to Western languages
 - ▶ Concept of spaces between words does (partly) not exist
 - ▶ Many different pronounciations for each character
- *WordPieceModel:* Data-dependent + do not produce OOVs
 - ① Initialize the vocabulary with basic Unicode characters (22k for Japanese, 11k for Korean)
 - ⚠ Spaces are indicated by an underscore attached before (or after) the respective basic unit or word (increases initial $|V|$ by up to factor 4)
 - ② Build a language model using this vocabulary
 - ③ Merge word units that increase the likelihood on the training data the most, when added to the model
- Two possible stopping criteria:
Vocabulary size *or* incremental increase of the likelihood

Use for neural machine translation

► Wu et al. (2016)

- *Adaptions:*

- ▶ Application to Western languages leads to a lower number of basic units (~ 500)
- ▶ Add space markers (underscores) *only* at the beginning of words
- ▶ Final vocabulary sizes between 8k and 32k yield a good balance between accuracy and fast decoding speed
(compared to around 200k from ► Schuster & Nakajima (2012))

Independent vs. joint encodings for source & target language

- Sennrich et al. (2016) report better results for joint BPE
- Wu et al. (2016) use shared WordPieceModel to guarantee identical segmentation in source & target language in order to facilitate copying rare entity names or numbers

No need for Pre-Tokenization

- BPE & WordPiece require a sequence of words as inputs
 - Some sort of (whitespace) tokenization has to be performed before their application
 - SentencePiece (as the name already reveals) doesn't need that
 - Can be applied to "raw" sentences
 - Consists of *Normalizer*, *Trainer*, *Encoder* & *Decoder*
 - Under the hood, two different algorithms are implemented
 - ▶ byte-pair encoding ► Sennrich et al. (2016)
 - ▶ unigram language model ► Kudo et al. (2018a)
 - No language-specific pre-processing
- ⇒ Basically a nice, end-to-end usable system/pipeline

Back to Transfer Learning

Embedding Idea + more complex architectures:

- *Naive approach:*
Standard embeddings (like word2vec) are "*context-free*"
- *Better:*
 - ▶ More complex networks, where the embeddings are trainable parameters of the model
 - ▶ Model learns *context sensitive* embeddings
 - ▶ We already encountered this in the Transformer
- *More complex networks:*
 - ▶ Whole network just to learn the embeddings, *or*
 - ▶ Additional embedding-layer as the lowest layer

Contextuality

1st Generation of neural embeddings are "context-free":

- Breakthrough paper by Mikolov et al, 2013 (Word2Vec)
- Followed by Pennington et al, 2014 (GloVe)
- Extension of Word2Vec by Bojanowski et al, 2016 (FastText)

Why "Context-free"?

- Models learn *one single* embedding for each word
- Why could this possibly be problematic?
 - ▶ "The *default* setting of the function is xyz."
 - ▶ "The probability of *default* is rather high."
- Would be nice to have different embeddings for these two occurrences

Transfer Learning – Further Motivation

Questions/Problems:

- Where in this (deep) model do we achieve contextuality?
(For sure *not* in the lowest layer!)
→ Not straightforward to extract them
- The deeper the network ..
 - ▶ .. the more expensive to train
 - ▶ .. the more data we need
→ You cannot just train them at home

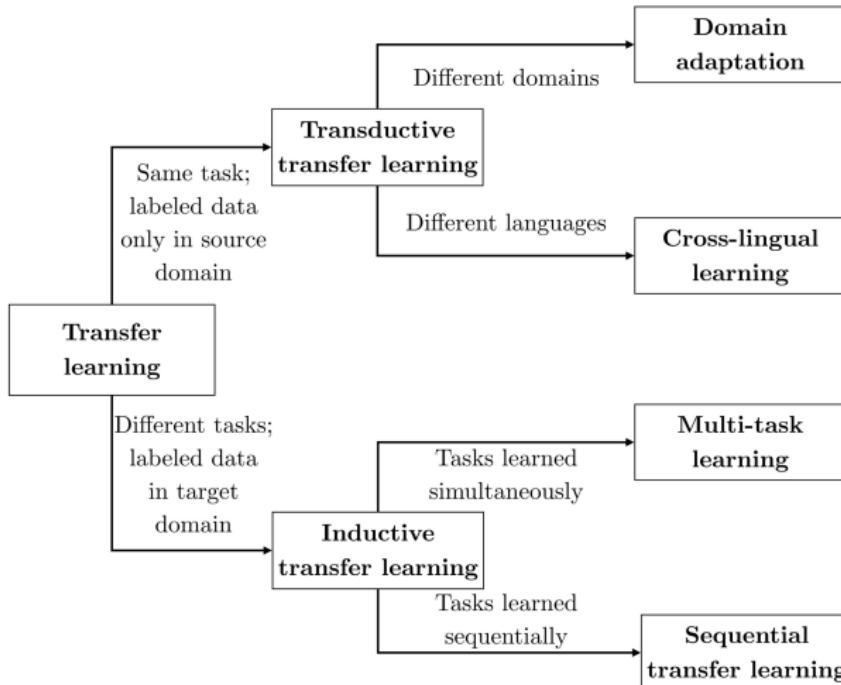
Transfer Learning

TRANSFER LEARNING

- Train such an architecture on ..
 - ▶ .. a fairly *general* task
 - ▶ .. which does not require any labels (" *self-supervised*")
 - ▶ .. using *large* amounts of data
- Do not extract static embeddings, but use the whole **pre-trained architecture**
- Replace the final layer used for the *general* task by a different layer for a *specific* task at hand

Taxonomy of transfer learning

► Ruder, 2019



Source: *Sebastian Ruder*

Transductive Transfer learning

- Domain adaptation:
→ "*Transfer knowledge learned from performing task A on labeled data from domain X to performing task A in domain Y.*"
- Cross-lingual learning:
→ "*Transfer knowledge learned from performing task A on labeled data from language X to performing task A in language Y.*"
- *Important:* No labeled data in target domain/language Y.

Inductive Transfer learning

- Multi-task learning:
→ "*Transfer knowledge learned from performing task A on data from domain X to performing multiple (simultaneous) tasks B, C, D, .. in domain Y.*"
- Sequential transfer learning:
→ "*Transfer knowledge learned from performing task A on data from domain X to performing multiple (sequential) tasks B, C, D, .. in domain Y.*"
- *Important:* Labeled data only for task(s) from target domain Y.

Feature-based transfer learning

Again: Word Embeddings

- The stored knowledge from the pre-trained model is extracted **as is** and is not further adapted to the actual domain/task of interest.
- *Difficulties:*
 - ▶ Source & target domain/task might be pretty different
 - ▶ No representations for domain-/task-specific words
 - ▶ No contextualization

Enhancement: *Embeddings from Language Models (ELMo)*



- Bidirectional language model (LM)
- Combines a forward LM

$$p(t_1, t_2, \dots, t_N) = \prod_{k=1}^N p(t_k | t_1, t_2, \dots, t_{k-1})$$

and a backward LM

$$p(t_1, t_2, \dots, t_N) = \prod_{k=1}^N p(t_k | t_{k+1}, t_{k+2}, \dots, t_N)$$

to arrive at the following loglikelihood:

$$\sum_{k=1}^N \left(\log p(t_k | t_1, \dots, t_{k-1}; \Theta_x, \vec{\Theta}_{LSTM}, \Theta_s) + \log p(t_k | t_{k+1}, \dots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s) \right)$$

ELMo embeddings

- Character-based (context-independent) token representations

$$x_k^{LM}$$

- Two-layer biLSTM as main architecture:

- ▶ Two context-dependent token representations *per layer*, i.e.

$\overrightarrow{\mathbf{h}}_{k,j}^{LM}$ & $\overleftarrow{\mathbf{h}}_{k,j}^{LM}$ for the k -th token in the j -th layer.

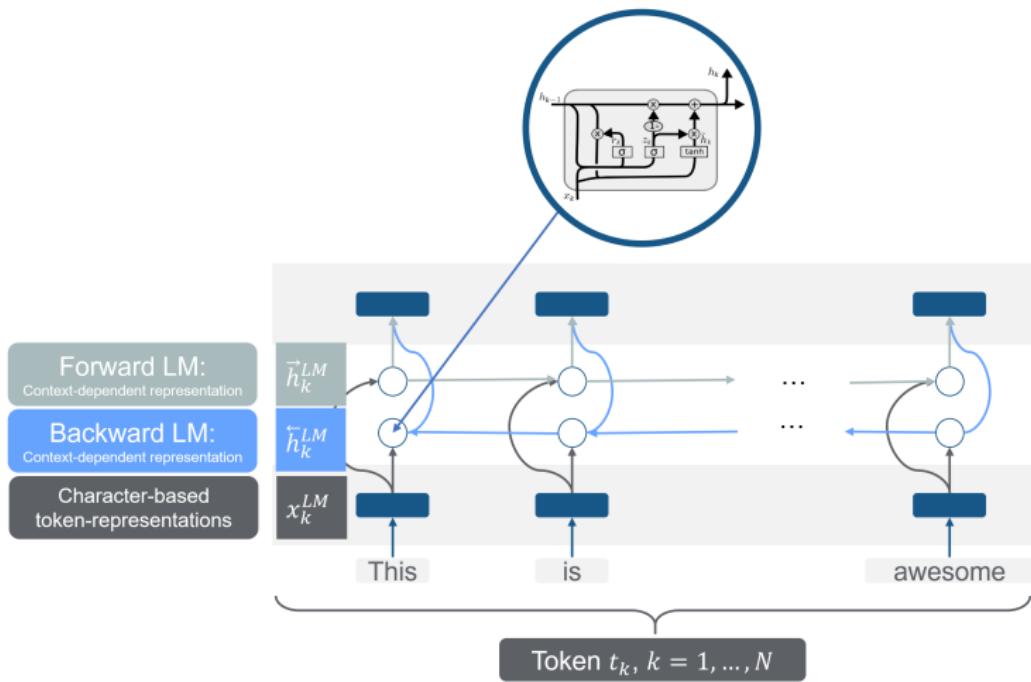
- ▶ Four context-dependent token representations in total:

$$\left\{ \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} | j = 1, 2 \right\}$$

- Five representations per token in total:

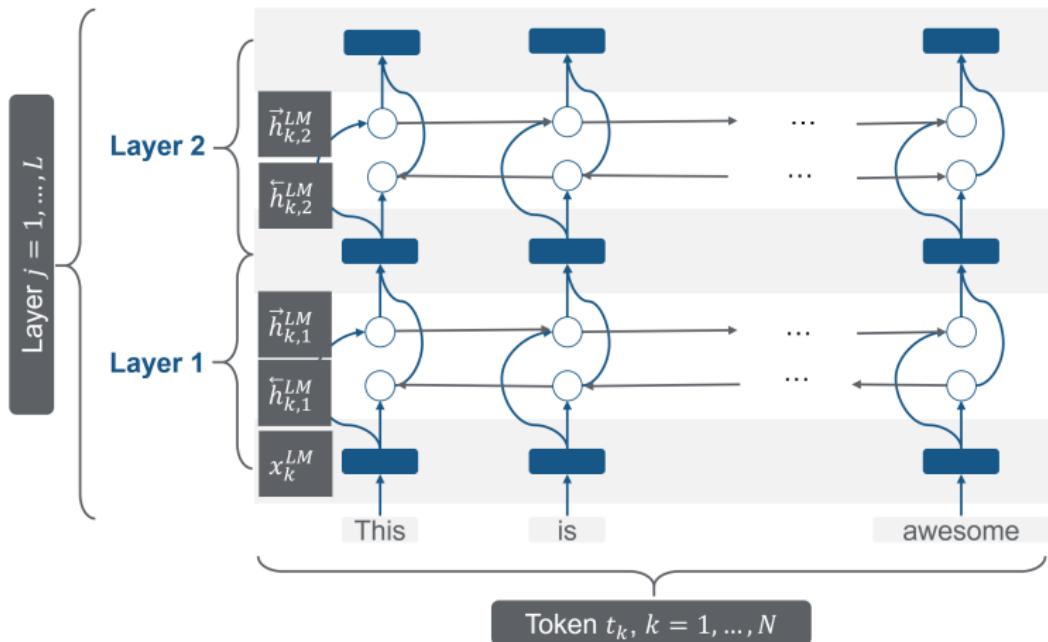
$$\begin{aligned} R_k &= \left\{ \mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} | j = 1, \dots, L \right\} \\ &= \left\{ \mathbf{h}_{k,j}^{LM} | j = 0, 1, 2 \right\} \end{aligned}$$

ELMo – Graphical representation



Source: Carolin Becker

ELMo – Graphical representation



Source: *Carolin Becker*

ELMo – Task Adaption

Including ELMo in downstream tasks:

- Calculate task-specific weights of all five representations:

$$\text{ELMo}_k^{\text{task}} = E\left(R_k; \Theta^{\text{task}}\right) = \gamma^{\text{task}} \sum_{j=0}^L s_j^{\text{task}} \mathbf{h}_{k,j}^{\text{LM}},$$

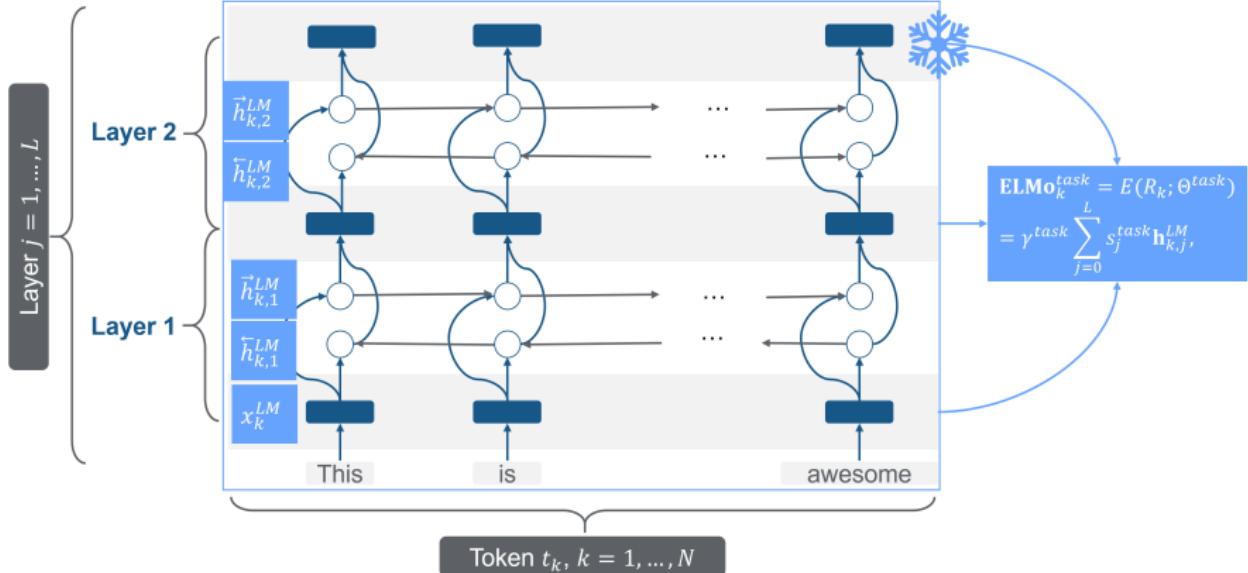
where the $\mathbf{h}_{k,j}^{\text{LM}}$ are **not trainable** anymore.

- Trainable parameters during the adaption:
 - ▶ s_j^{task} are trainable (softmax-normalized) weights
 - ▶ γ^{task} is a trainable scaling parameter

Advantages over context free-embeddings:

- Task-specific model has access to *multiple* representations of each token
- Model learns to which degree to use the different representations depending on the task at hand

ELMo – Task Adaption



Source: *Carolin Becker*

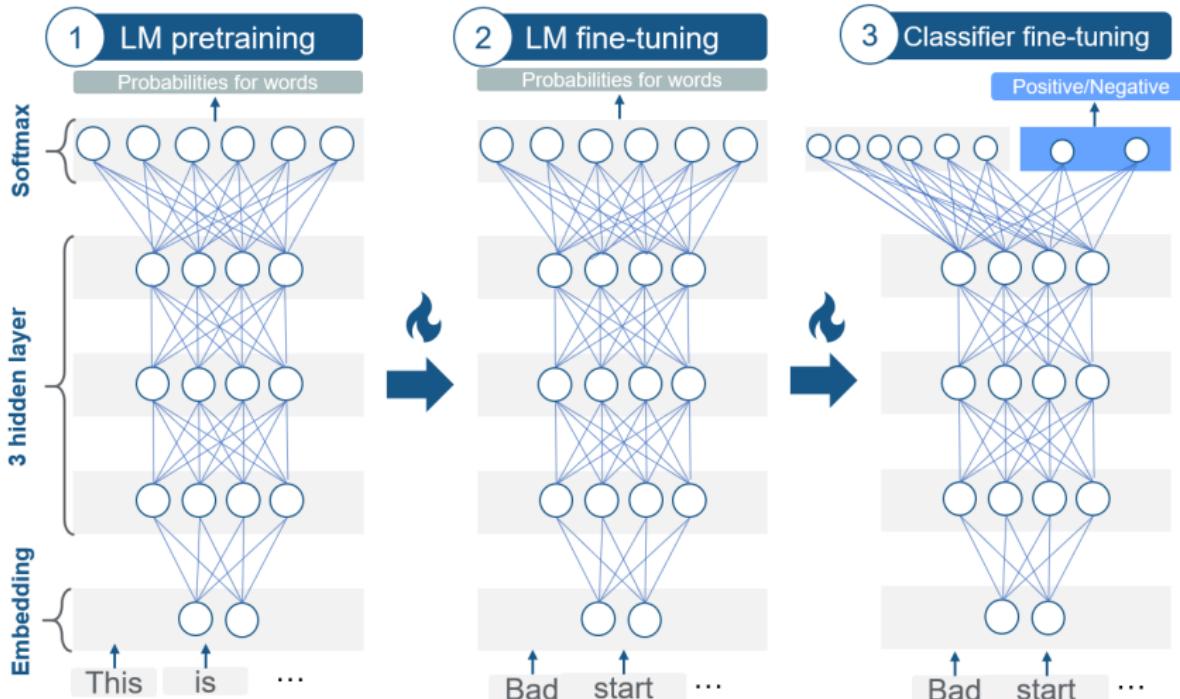
Fine-tuning approach

Shortcomings of ELMo:

- Pre-trained on a general domain corpus, embeddings are not adapted to the domain/task at hand
- Sequential nature of LSTMs:
 - ▶ Not fully parallelizable (compared to Transformers)
 - ▶ Fail to capture long-range dependency during contextualization

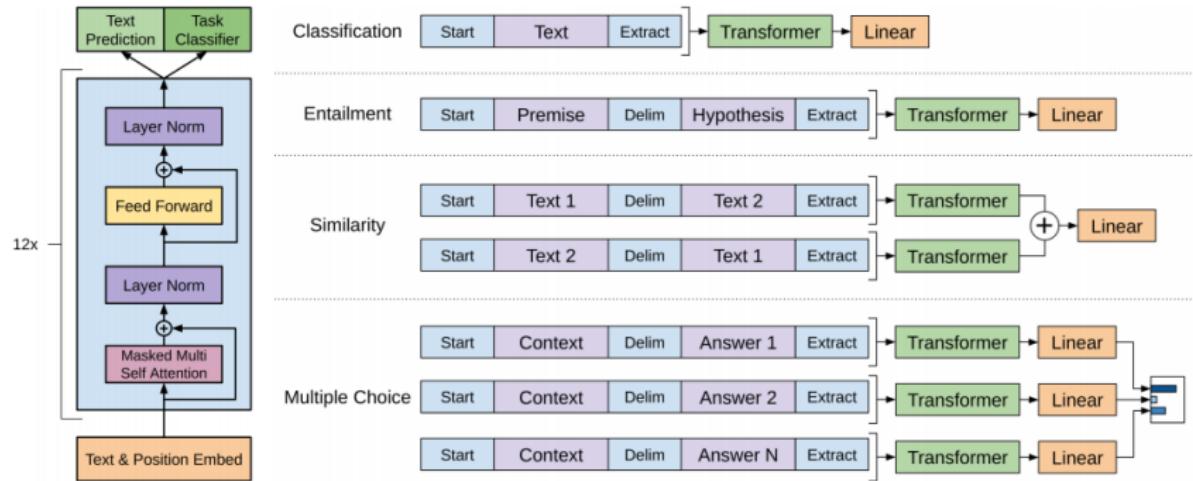
Alleviations/Alternatives:

- ULMFiT ▶ Howard and Ruder, 2018 is a uni-directional LSTM which is fine-tuned as a whole model on data from the target domain/task.
- GPT ▶ Radford et al., 2018 is a Transformer (decoder) which is fine-tuned as a whole model on data from the target domain/task.

Source: *Carolin Becker*

ULMFiT – Architectural Details

- AWD-LSTMs ▶ Merity et al., 2017 as backbone of the architecture
 - ▶ DropConnect ▶ Wan et al., 2013
 - ▶ Averaged stochastic gradient descent (ASGD) for optimization
- Embedding layer + three LSTM layers + Softmax Layer
- **LM fine-tuning:**
 - ▶ Discriminative fine-tuning
- **Classifier fine-tuning:**
 - ▶ Concat Pooling
 - ▶ Gradual unfreezing



Source: *Radford et al., 2018*

GPT – Architectural Details

- Transformer decoder as backbone of the architecture
 - ▶ 12-layer-decoder with masked attention heads
 - ▶ 40k BPE vocabulary
 - ▶ Learned positional embeddings (compared to sinusoidal versions)
- **Fine-tuning:**
 - ▶ Linear output layer with softmax activation on top
 - ▶ Auxiliary language modeling objective during fine-tuning
 - Improves generalization
 - Accelerates convergence
 - ▶ Task-specific input transformations (see previous slide)

GPT – SOTA results

Performance on different benchmarks:

Table 2: Experimental results on natural language inference tasks, comparing our model with current state-of-the-art methods. 5x indicates an ensemble of 5 models. All datasets use accuracy as the evaluation metric.

Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	<u>89.3</u>	-	-	-
CAFE [58] (5x)	80.2	79.0	<u>89.3</u>	-	-	-
Stochastic Answer Network [35] (3x)	<u>80.6</u>	<u>80.1</u>	-	-	-	-
CAFE [58]	78.7	77.9	88.5	<u>83.3</u>		
GenSen [64]	71.4	71.3	-	-	<u>82.3</u>	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

Table 3: Results on question answering and commonsense reasoning, comparing our model with current state-of-the-art methods.. 9x means an ensemble of 9 models.

Method	Story Cloze	RACE-m	RACE-h	RACE
val-LS-skip [55]	<u>76.5</u>	-	-	-
Hidden Coherence Model [7]	<u>77.6</u>	-	-	-
Dynamic Fusion Net [67] (9x)	-	55.6	49.4	51.2
BiAttention MRU [59] (9x)	-	<u>60.2</u>	<u>50.3</u>	<u>53.3</u>
Finetuned Transformer LM (ours)	86.5	62.9	57.4	59.0

Source: Radford et al. (2018)

Pre-training objectives

Self-Supervision:

- Special case of unsupervised learning
- Labels are generated from the data itself

Self-supervised objectives:

- Skip-gram objective (cf. word2vec [► Mikolov et al. \(2013a\)](#))
- Language modeling objective (cf. [► Bengio et al. \(2003\)](#))
- *Masked language modeling (MLM)* objective (cf. chapter 10)
→ Replace words by a [MASK] token and train the model to predict
- *Permutation language modeling (PLM)* objective (cf. chapter 11)
→ Autoregressive objective of XLNet
- *Replaced token detection* objective (cf. chapter 11)
→ Requires two models: One performing MLM & the second model to discriminate between actual and the predicted tokens

Pre-training resources

Commonly used (large-scale) data sets for pre-training

- English Wikipedia
- 1B Word Benchmark ➔ Chelba et al. (2013)
- BooksCorpus ➔ Zhu et al. (2015)
- Wikitext-103 ➔ Merity et al. (2016)
- CommonCrawl ➔ <https://commoncrawl.org/>

Non-exhaustive list; tbc in the following chapters

Transfer Learning in Computer Vision

ImageNet: ▶ Deng et al., 2009

- Large-scale data set (\approx 50 million labeled images)
- Hierarchical data set structured in synsets
- "*Diverse coverage of the image world.*" (Deng et al., 2009)

How it changed learning:

- Quasi-standard to use a model pre-trained on ImageNet
- Achieved SOTA results in various computer vision tasks
- Enable the use of large models to small (labeled) data sets

Summary & Outlook

Pros:

- New and deep architectures enable better representation learning
- Leverage favorable properties of language to create self-supervised tasks
- Use ubiquitous large amounts of unlabeled data available on the web

Cons:

- Pre-Training *extremely* costly
- Models will have up to over billions parameters
- Only works well for high-resource languages

Further reading

- *NLP's ImageNet moment has arrived*
- Sebastian Ruder's PhD thesis: [▶ Ruder, 2019](#)

Chapter 10: BERT (& the Sesame Street)

Matthias Aßenmacher

January 20, 2021



Bidirectional Encoder Representations from Transformers:

- Bidirectionally contextual model
- Introduces new self-supervised objective(s)
- Completely replaces recurrent architectures by Self-Attention
+ simultaneously able to include bidirectionality

Predecessors of BERT

2013 - word2vec

Tomas Mikolov et al. publish four papers on vector representations of words constituting the *word2vec* framework

This received very much attention as it revolutionized the way words were encoded for deep learning models in the field of NLP.

2013

Predecessors of BERT

2013 - word2vec

Tomas Mikolov et al. publish four papers on vector representations of words constituting the *word2vec* framework.

This received very much attention as it revolutionized the way words were encoded for deep learning models in the field of NLP.



January 2018 - ULMFiT

The first transfer learning architecture (Universal Language Model Fine-Tuning) was proposed by Howard and Ruder (2018).

An embedding layer at the bottom of the network was complemented by three AWD-LSTM layers (Merity et al., 2017) and a softmax layer for pre-training.

A **Unidirectional contextual** model since no biLSTMs are used.

Predecessors of BERT

2013 - word2vec

Tomas Mikolov et al. publish four papers on vector representations of words constituting the *word2vec* framework

This received very much attention as it revolutionized the way words were encoded for deep learning models in the field of NLP.

2013

01/2018

02/2018

February 2018 - ELMo

Guy from AllenNLP developed a bidirectionally contextual framework by proposing ELMo (Embeddings from Language Models; Peters et al., 2018).

Embeddings from this architecture are the (weighted) combination of the intermediate-layer representations produced by the biLSTM layers.

January 2018 - ULMFiT

The first transfer learning architecture (Universal Language Model Fine-Tuning) was proposed by Howard and Ruder (2018).

An embedding layer at the bottom of the network was complemented by three AWD-LSTM layers (Merity et al., 2017) and a softmax layer for pre-training.

A **Unidirectional contextual** model since no biLSTMs are used.

Predecessors of BERT

2013 - word2vec

Tomas Mikolov et al. publish four papers on vector representations of words constituting the *word2vec* framework

This received very much attention as it revolutionized the way words were encoded for deep learning models in the field of NLP.

2013

01/2018

02/2018

06/2018

February 2018 - ELMo

Guys from AllenNLP developed a bidirectionally contextual framework by proposing ELMo (Embeddings from Language Models; Peters et al., 2018).

Embeddings from this architecture are the (weighted) combination of the intermediate-layer representations produced by the biLSTM layers.

January 2018 - ULMFiT

The first transfer learning architecture (Universal Language Model Fine-Tuning) was proposed by Howard and Ruder (2018).

An embedding layer at the bottom of the network was complemented by three AWD-LSTM layers (Merity et al., 2017) and a softmax layer for pre-training.

A **Unidirectional contextual** model since no biLSTMs are used.

June 2018 - OpenAI GPT

Radford et al., 2018 abandon the use of LSTMs. They combine multiple Transformer decoder blocks with a standard language modelling objective for pre-training.

Compared to ELMo it is just **unidirectionally contextual**, since it uses only the decoder side of the Transformer. On the other hand it is **end-to-end trainable** (cf. ULMFiT) and embeddings do not have to be extracted like in the case of ELMo.

Predecessors of BERT

2013 - word2vec

Tomas Mikolov et al. publish four papers on vector representations of words constituting the *word2vec* framework

This received very much attention as it revolutionized the way words were encoded for deep learning models in the field of NLP.

2013

01/2018

02/2018

06/2018

10/2018

February 2018 - ELMo

Guy from AllenNLP developed a bidirectionally contextual framework by proposing ELMo (Embeddings from Language Models; Peters et al., 2018).

Embeddings from this architecture are the (weighted) combination of the intermediate-layer representations produced by the biLSTM layers.

October 2018 - BERT

BERT (Devlin et al., 2018) is a bidirectional contextual embedding model purely relying on Self-Attention by using multiple Transformer encoder blocks.

BERT (and its successors) rely on the Masked Language Modelling objective during pre-training on huge unlabelled corpora of text.

January 2018 - ULMFiT

The first transfer learning architecture (Universal Language Model Fine-Tuning) was proposed by Howard and Ruder (2018).

An embedding layer at the bottom of the network was complemented by three AWD-LSTM layers (Merity et al., 2017) and a softmax layer for pre-training.

A **Unidirectional contextual** model since no biLSTMs are used.

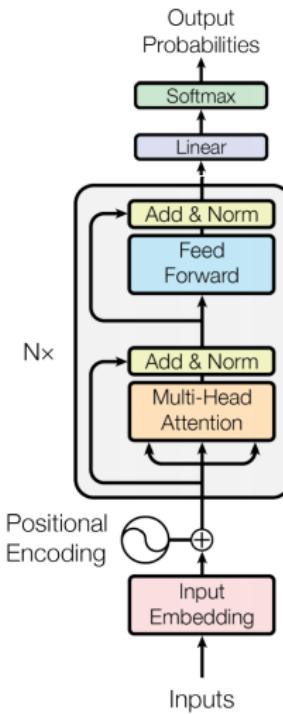
June 2018 - OpenAI GPT

Radford et al., 2018 abandon the use of LSTMs. They combine multiple Transformer decoder block with a standard language modelling objective for pre-training.

Compared to ELMo it is just **unidirectionally contextual**, since it uses only the decoder side of the Transformer. On the other hand it is **end-to-end trainable** (cf. ULMFiT) and embeddings do not have to be extracted like in the case of ELMo.

Core of BERT

► Devlin et al. (2018)



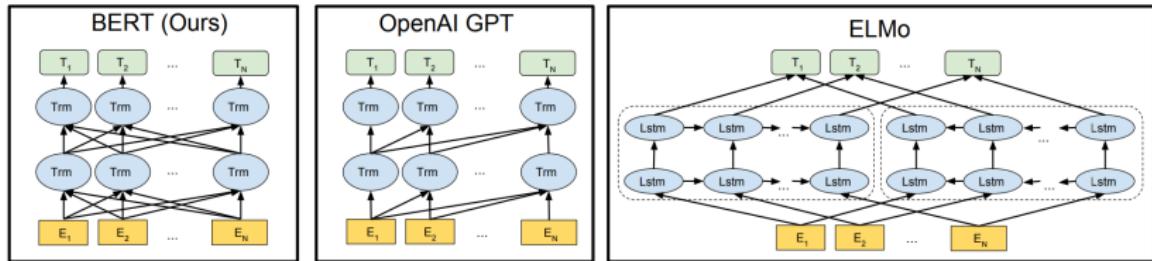
Source: ► Vaswani et al. (2017)

A remark on Self-Supervision

Causality is an issue!

- *Remember:* Input and target sequences are the same
 - We modify the input to create a meaningful task
- A sequence is used to predict itself again
- Bidirectionality at a lower layer would allow a word to see itself at later hidden layers
 - The model would be allowed to cheat!
 - This would not lead to meaningful internal representations

GPT vs. ELMo vs. BERT



Source: Devlin et al. (2018)

Major architectural differences:

- ELMo uses two separate unidirectional models to achieve bidirectionality
→ Only "*shallow*" bidirectionality
- GPT is not bidirectional, thus no issues concerning causality
- BERT combines the best of both worlds:

Self-Attention + (Deep) Bidirectionality

Masked Language Modeling (MLM)

First of all:

- It has nothing to do with Masked Self-Attention
→ Masked Self-Attention is an architectural detail in the decoder of a Transformer, i.e. used by e.g. GPT
- Masked Self-Attention as a way to induce causality in the decoder
- MLM is a modeling objective introduced to couple Self-Attention and (deep) bidirectionality without violating causality

Masked Language Modeling (MLM) ctd.

Masked Language Modeling:

- **Training objective:**

Given a sentence, predict [MASK]ed tokens

- **Generation of samples:**

Randomly replace* a fraction of the words by [MASK]

*Sample 15% of the tokens; replace 80% of them by [MASK], 10% by a random token & leave 10% unchanged

- **Input:**

The quick brown [MASK] jumps over the [MASK] dog .

- **Targets:**

(*fox, lazy*)

Masked Language Modeling (MLM) ctd.

Discrepancy between pre-training & fine-tuning:

- [MASK]-token as central part of pre-training procedure
- [MASK]-token does not occur during fine-tuning
- **Modified pre-training task:**

Predict 15% of the tokens of which only 80% have been replaced by [MASK]

- ▶ 80% of the selected tokens:

The quick brown fox → The quick brown [MASK]

- ▶ 10% of the selected tokens:

The quick brown fox → The quick brown went

- ▶ 10% of the selected tokens:

The quick brown fox → The quick brown fox

Next Sentence Prediction (NSP)

Next Sentence Prediction:

- **Training objective:**

Given two sentences, predict whether s_2 follows s_1

- **Generation of samples:**

Randomly sample* negative examples (cf. word2vec)

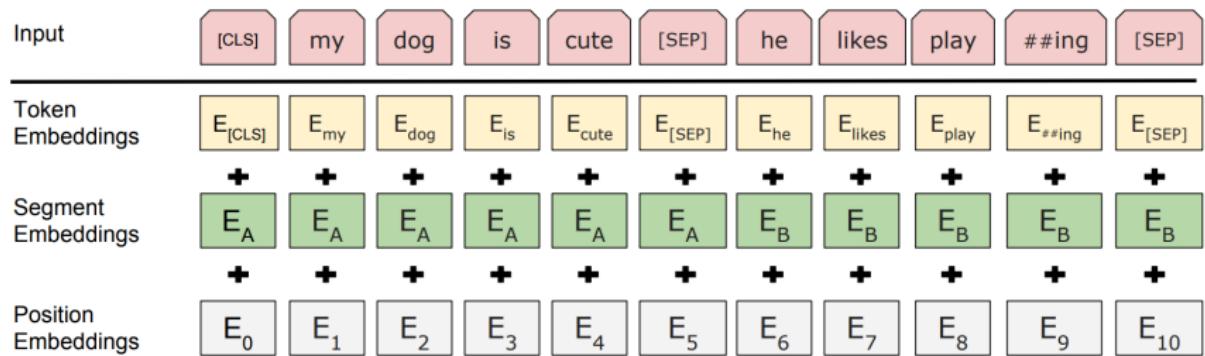
*50% of the time the second sentence is the actual next sentence, 50% of the time it is a randomly sampled sentence

- **Full Input:**

[CLS]	The	[MASK]	is	quick	.	[SEP]
It	jumps	over	the	[MASK]	dog	.

- [CLS] token as sequence representation for classification
- [SEP] token for separation of the two input sequences

BERT's input embeddings



Source: ▶ Devlin et al. (2018)

- Byte-Pair encoding ▶ Sennrich et al. (2016) for the inputs
→ Vocabulary of 30.000 tokens

Pre-Training BERT

Ingredients:

- Massive lexical resources (BooksCorpus + Eng. Wikipedia)
→ 13 GB in total
- Train for approximately* 40 epochs
- 4 (16) Cloud TPUs for 4 days for the BASE (LARGE) variant
- 12 (24) Transformer encoder blocks with an embedding size of $E = 768$ (1024) and a hidden layer size $H = E$, $H/64 = 12$ (16) attention heads are used and the feed-forward size is set to $4H$
→ 110M (340M) model parameters in total for $BERT_{Base}$
 $(BERT_{Large})$
- Loss function:

$$Loss_{BERT} = Loss_{MLM} + Loss_{NSP}$$

*1.000.000 steps on batches of 256 sequences with a sequence length of 512 tokens

Pre-Training BERT – Maximum sequence length

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

not cool

cool

Source: Vaswani et al. (2017)

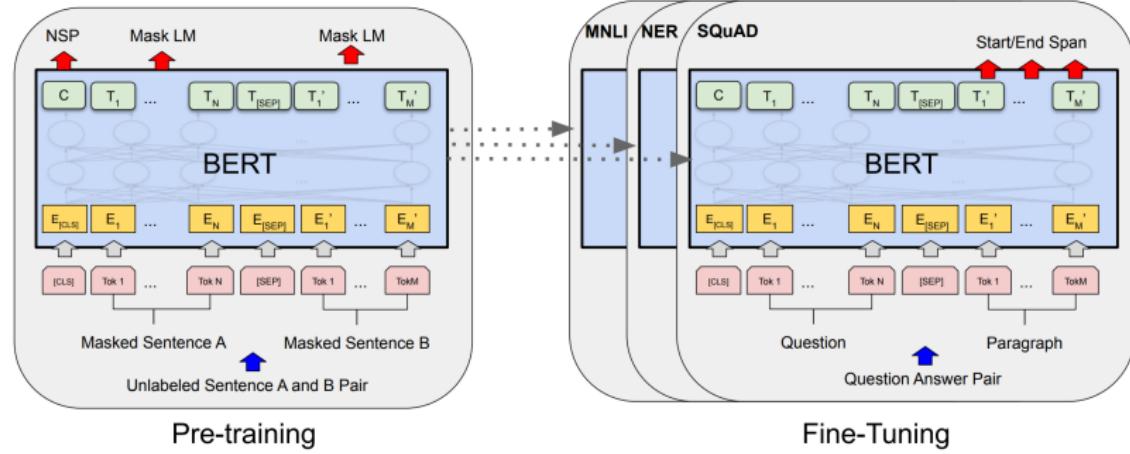
Limitation:

- BERT can only consume sequences of up to 512 tokens
- Two sentences for NSP are sampled such that

$$\text{length}_{\textit{sentenceA}} + \text{length}_{\textit{sentenceB}} \leq 512$$

- Reason: Computational complexity of Transformer scales quadratically with the sequence length
→ Longer sequences are disproportionately expensive

Fine-Tuning BERT



Source: Devlin et al. (2018)

Fine-Tuning BERT

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average
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

Source: Devlin et al. (2018)

- Performance of BERT on the ➔ GLUE Benchmark
- Beats all of the previous state-of-the-art models
- In the meantime: Other models better than BERT

Successors of BERT

| October 2018 - BERT

| BERT (**Devlin et al., 2018**) is a
| bidirectional contextual embedding
| model purely relying on Self-Attention
| by using multiple **Transformer encoder**
| blocks.

| BERT (and its successors) rely on the
| **Masked Language Modelling objective**
| during pre-training on huge unlabelled
| corpora of text.

10/2018

Successors of BERT

October 2018 - BERT

BERT (**Devlin et al., 2018**) is a bidirectional contextual embedding model purely relying on Self-Attention by using multiple **Transformer encoder** blocks.

BERT (and its successors) rely on the **Masked Language Modelling objective** during pre-training on huge unlabelled corpora of text.

10/2018

02/2019

February 2019 - GPT2

Radford et al., 2019 massively scale up their GPT model from 2018 (up to 1.5 billion parameters). Despite the size, there are only smaller architectural changes, thus it remains a **unidirectionally contextual** model.

Controversial debate about this model, since OpenAI (at first) refuses to make their pre-trained architecture publicly available due to concerns about "malicious applications".

Successors of BERT

October 2018 - BERT

BERT ([Devlin et al., 2018](#)) is a bidirectional contextual embedding model purely relying on Self-Attention by using multiple **Transformer encoder** blocks.

BERT (and its successors) rely on the **Masked Language Modelling objective** during pre-training on huge unlabelled corpora of text.

10/2018

02/2019

06/2019

June 2019 - XLNet

[Yang et al., 2019](#) design a new pre-training objective in order to overcome some weaknesses they spotted in the one used by BERT.

The use **Permutation Language Modelling** to avoid the discrepancy between pre-training and fine-tuning introduced by the artificial MASK token.

February 2019 - GPT2

[Radford et al., 2019](#) massively scale up their GPT model from 2018 (up to 1.5 billion parameters). Despite the size, there are only smaller architectural changes, thus it remains a **unidirectionally contextual** model.

Controversial debate about this model, since OpenAI (at first) refuses to make their pre-trained architecture publicly available due to concerns about "malicious applications".

Successors of BERT

October 2018 - BERT

BERT ([Devlin et al., 2018](#)) is a bidirectional contextual embedding model purely relying on Self-Attention by using multiple **Transformer encoder** blocks.

BERT (and its successors) rely on the **Masked Language Modelling objective** during pre-training on huge unlabelled corpora of text.

10/2018

02/2019

06/2019

07/2019

June 2019 - XLNet

[Yang et al., 2019](#) design a new pre-training objective in order to overcome some weaknesses they spotted in the one used by BERT.

The use **Permutation Language Modelling** to avoid the discrepancy between pre-training and fine-tuning introduced by the artificial MASK token.

February 2019 - GPT2

[Radford et al., 2019](#) massively scale up their GPT model from 2018 (up to 1.5 billion parameters). Despite the size, there are only smaller architectural changes, thus it remains a **unidirectionally contextual** model.

Controversial debate about this model, since OpenAI (at first) refuses to make their pre-trained architecture publicly available due to concerns about "malicious applications".

July 2019 - RoBERTa

[Liu et al., 2019](#) concentrate on improving the original BERT architecture by (1) careful hyperparameter tuning (2) abandoning the additional Next Sentence Prediction objective (3) increasing the pre-training corpus *massively*.

Other approaches now more and more concentrate on improving, down-scaling or understanding BERT. A new research direction called **BERTology** emerges.

Successors of BERT

October 2018 - BERT

BERT ([Devlin et al., 2018](#)) is a bidirectional contextual embedding model purely relying on Self-Attention by using multiple **Transformer encoder** blocks.

BERT (and its successors) rely on the **Masked Language Modelling objective** during pre-training on huge unlabelled corpora of text.

10/2018

02/2019

06/2019

07/2019

10/2019

June 2019 - XLNet

[Yang et al., 2019](#) design a new pre-training objective in order to overcome some weaknesses they spotted in the one used by BERT.

The use **Permutation Language Modelling** to avoid the discrepancy between pre-training and fine-tuning introduced by the artificial MASK token.

October 2019 - T5

[T5 \(Raffel et al., 2019\)](#) a complete **encoder-decoder** Transformer based architecture (**text-to-text transfer transformer**).

They approach transfer learning by transforming all inputs as well as all outputs to strings and fine-tuned their model simultaneously on data sets with multiple different tasks.

February 2019 - GPT2

[Radford et al., 2019](#) massively scale up their GPT model from 2018 (up to 1.5 billion parameters). Despite the size, there are only smaller architectural changes, thus it remains a **unidirectionally contextual** model.

Controversial debate about this model, since OpenAI (at first) refuses to make their pre-trained architecture publicly available due to concerns about "malicious applications".

July 2019 - RoBERTa

[Liu et al., 2019](#) concentrate on improving the original BERT architecture by (1) careful hyperparameter tuning (2) abandoning the additional Next Sentence Prediction objective (3) increasing the pre-training corpus *massively*.

Other approaches now more and more concentrate on improving, down-scaling or understanding BERT. A new research direction called **BERTology** emerges.

Post-BERT architectures:

- Most architectures still rely on either an encoder- *or* a decoder-style type of model (e.g. ► GPT2 , ► XLNet)
- *BERTology*: Many papers/models which aim at ..
 - ▶ ... explaining BERT (e.g. ► Coenen et al., 2019 , ► Michel et al., 2019)
 - ▶ ... improving BERT (► RoBERTa , ► ALBERT)
 - ▶ ... making BERT more efficient (► ALBERT , ► DistilBERT)
 - ▶ ... modifying BERT (► BART)
- Overview on many different papers:
<https://github.com/tomohideshibata/BERT-related-papers>

Improvements in Pre-Training:

- Authors claim that BERT is seriously undertrained
- Change of the MASKing strategy
 - BERT masks the sequences once before pre-training
 - RoBERTa uses dynamic MASKing
 - ⇒ RoBERTa sees the same sequence MASKed differently
- RoBERTa does not use the additional NSP objective during pre-training
- 160 GB of pre-training resources instead of 13 GB
- Pre-training is performed with larger batch sizes (8k)

Static Masking (BERT):

- Apply MASKing procedure to pre-training corpus once
- (additional for BERT: Modify the corpus for NSP)
- Train for approximately 40 epochs

Dynamic Masking (RoBERTa):

- Duplicate the training corpus *ten* times
- Apply MASKing procedure to each duplicate of the pre-training corpus
- Train for 40 epochs
- Model sees each training instance in ten different "versions"
(each version four times) during pre-training

Architectural differences:

- Architecture (layers, heads, embedding size) identical to BERT
- 50k token BPE vocabulary instead of 30k
- Model size differs (due to the larger embedding matrix)
⇒ ~ 125M (360M) for the BASE (LARGE) variant

Performance differences:

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
<i>Single-task single models on dev</i>										
BERT _{LARGE}	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
XLNet _{LARGE}	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	90.2/90.2	94.7	92.2	86.6	96.4	90.9	68.0	92.4	91.3	-

Source: Liu et al. (2019)

Note: Liu et al. (2019) report the accuracy for QQP while Devlin et al. (2018) report the F1 score (cf. results displayed on slide 15); XLNet: see next Chapter.

Changes in the architecture:

- Disentanglement of embedding size E and hidden layer size H
 - WordPiece-Embeddings (size E) context-independent
 - Hidden-Layer-Embeddings (size H) context-dependent
 - ⇒ Setting $H \gg E$ enlarges model capacity without increasing the size of the embedding matrix,
since $O(V \times H) > O(V \times E + E \times H)$ if $H \gg E$.
- Cross-Layer parameter sharing
- Change of the pre-training NSP loss
 - Introduction of *Sentence-Order Prediction* (SOP)
 - Positive examples created alike to those from NSP
 - Negative examples: Just swap the ordering of sentences
- n -gram masking for the MLM task

Performance differences:

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT	base	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3	4.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	1.0
ALBERT	base	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1	5.6x
	large	18M	90.6/83.9	82.3/79.4	83.5	91.7	68.5	82.4	1.7x
	xlarge	60M	92.5/86.1	86.1/83.1	86.4	92.4	74.8	85.5	0.6x
	xxlarge	235M	94.1/88.3	88.1/85.1	88.0	95.2	82.3	88.7	0.3x

Source: Lan et al. (2019)

Notes:

- In General: Smaller model size (because of parameter sharing)
- Nevertheless: Scale model up to almost similar size (xxlarge version)
- Strong performance compared to BERT

Using BERT & Co.

Native implementations:

- BERT: [*https://github.com/google-research/bert*](https://github.com/google-research/bert)
- RoBERTa:
[*https://github.com/pytorch/fairseq/tree/master/examples/roberta*](https://github.com/pytorch/fairseq/tree/master/examples/roberta)
- ALBERT: [*https://github.com/google-research/ALBERT*](https://github.com/google-research/ALBERT)

Drawbacks:

- Different frameworks use for the implementations
- Different programming styles
 - Adaption of different models to custom problems can sometimes lead to a lot of redundant work

Example: Fine-tune native BERT on MRPC

Command line:

```
!python run_classifier.py \
--task_name=MRPC \
--do_train=true \
--do_eval=true \
--data_dir=$GLUE_DIR/MRPC \
--vocab_file=$BERT_BASE_DIR/vocab.txt \
--bert_config_file=$BERT_BASE_DIR/bert_config.json \
--init_checkpoint=$BERT_BASE_DIR/bert_model.ckpt \
--max_seq_length=128 \
--train_batch_size=32 \
--learning_rate=2e-5 \
--num_train_epochs=3.0 \
--output_dir=/tmp/mrpc_output/
```

Pre-trained architectures @ transformers

Unified API for state-of-the-art architectures:

- 32+ pre-trained architectures (as of January 7, 2021)
- Models in 100+ languages available (as of January 7, 2021)
- Implementations in PyTorch as well as TensorFlow 2.0
- Unified naming model parts and fine-tuning procedures
- Docs: <https://huggingface.co/transformers/index.html>

Which different building blocks available?

- Model architecture
- Custom tokenizers for each architecture
- (Sets of) Pre-trained weights for an architecture
- Pre-defined heads for fine-tuning on common tasks

Pre-trained architectures @ transformers

Pre-trained weights:

```
<model name>-<version>-<cased/uncased>
```

Load tokenizer & tokenize a sentence:

```
from transformers import BertTokenizer
tokenizer = BertTokenizer.from_pretrained("bert-base-cased")
sentence = "Hello guys, welcome to the course."

ids = tokenizer.encode(sentence, add_special_tokens = True)
ids

# [101, 8667, 3713, 117, 7236, 1106, 1103, 1736, 119, 102]

[tokenizer.convert_ids_to_tokens(id) for id in ids]

# ['[CLS]', 'Hello', 'guys', ',', 'welcome', 'to', 'the',
# 'course', '.', '[SEP]']
```

Pre-trained architectures @ transformers

Tokenization all in one:

- BERT requires inputs of fixed length → Padding
- [PAD] tokens should not receive Attention weights

```
tokenizer.encode_plus(sentence,
                      add_special_tokens = True,
                      max_length = 12,
                      pad_to_max_length = True,
                      return_attention_mask = True,
                      return_tensors = 'pt'
                    )

# {'input_ids': tensor([[ 101,  8667,  3713,   117,  7236,
#                         1106,  1103,  1736,   119,   102,      0,
#                         0]]),
#  'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
#                            0, 0]]),
#  'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
#                            0, 0]])}
```

Pre-trained architectures @ transformers

Load a model architecture:

```
from transformers import BertForSequenceClassification
mod = BertForSequenceClassification.from_pretrained("bert-
    base-cased",
    num_labels = 3)
```

Inspect the dimensionality of the model:

```
params = list(mod.parameters())

## size of the embedding layer
params[0].shape

# torch.Size([28996, 768])

## size of the classification layer
params[200].shape

# torch.Size([3])
```

Pre-trained architectures @ transformers

Pipelines:

- Extremely high-level API included in transformers
- Available for a couple of different downstream tasks
- Ingredients of a pipeline:
 - ▶ *Encoding*: Tokenization of the inputs
 - ▶ Inferency by a chosen model
 - ▶ *Decoding*: Use model output to generate target values

```
from transformers import pipeline

pipeline(<task name>, model = <model name>,
        tokenizer = <tokenizer name>)
```

- **model** and **tokenizer** are optional arguments
→ If not provided, some internal defaults are used

Pre-trained architectures @ transformers

Available tasks (as of January 7, 2021):

- Feature Extraction ("feature-extraction")
- Sentiment Analysis ("sentiment-analysis")
- Named entity recognition ("ner")
- Question Answering ("question-answering")
- [MASK]-filling ("fill-mask")
- Summarization ("summarization")
- Translation ("translation-xx-to-yy")
- seq2seq Text Generation ("text2text-generation")
- Text Generation ("text-generation")
- Zero-Shot Classification ("zero-shot-classification")
- Multi-turn Conversation ("conversation")

Pre-trained architectures @ transformers

Exemplary task (with default model):

```
from transformers import pipeline

pipe_classif = pipeline("sentiment-analysis")
pipe_classif(sentence)

# [{'label': 'POSITIVE', 'score': 0.99960136}]

pipe_classif("I absolutely hate this!")

# [{'label': 'NEGATIVE', 'score': 0.9992645}]
```

Default:

- DistilBERT model
- base, uncased
- Fine-tuned on SST-2 data set

Pre-trained architectures @ transformers

Use other models than the default:

```
pipe_fill = pipeline("fill-mask",
    model = "bert-large-cased",
    tokenizer = BertTokenizer.from_pretrained("bert-large-
cased"))
pipe_fill("I like " + pipe_fill.tokenizer.mask_token + "
football.")

# [{"sequence": '[CLS] I like playing football. [SEP]', 
#   'score': 0.4649055004119873, 
#   'token': 1773}, 
# {"sequence": '[CLS] I like watching football. [SEP]', 
#   'score': 0.19629190862178802, 
#   'token': 2903}, 
# {"sequence": '[CLS] I like the football. [SEP]', 
#   'score': 0.10121186822652817, 
#   'token': 1103}, 
# {"sequence": '[CLS] I like American football. [SEP]', 
#   'score': 0.0536048598587513, 
#   'token': 1237}]
```

Other languages than English

Multilingual models:

- BERT also available as multilingual model
- Top 100 languages with the largest Wikipedias
- Re-weighting of training data (favor low-resource languages)
- 110k shared WordPiece vocabulary
- Released in a base, cased version
- <https://github.com/google-research/bert/blob/master/multilingual.md>

Monolingual models:

- Specifically trained for each language separately
- Examples for German:
 - ▶ *deepset.ai*
 - ▶ *Bayerische Staatsbibliothek*

Other languages than English

```
pipe_fill_ger = pipeline("fill-mask",
    model="bert-base-german-cased",
    tokenizer=AutoTokenizer.from_pretrained("bert-base-
    german-cased"))
pipe_fill_ger("Der Himmel ist " + pipe_fill.tokenizer.
    mask_token)

# [{"sequence": '[CLS] Der Himmel ist blau [SEP]', 
#   'score': 0.18068572878837585, 
#   'token': 8516}, 
#  {"sequence": '[CLS] Der Himmel ist leer [SEP]', 
#   'score': 0.10186842083930969, 
#   'token': 12101}, 
#  {"sequence": '[CLS] Der Himmel ist frei [SEP]', 
#   'score': 0.04153556749224663, 
#   'token': 1409}]
```

Further reading

- See reference to GitHub-repo on Slide 17

Chapter 11: BERT – Improvements & Alternatives

Matthias Aßenmacher

January 27, 2021

What we will cover in this lecture

Disclaimer I:

The overview presented in this chapter is everything but exhaustive!

We will discuss *Shortcomings and limitations* of BERT & Co.

Architectures

- using *different objectives* (XLNet, ELECTRA),
- exhibiting *different architectural design choices* (XLNet, T5) and
- *alleviating some of the shortcomings* (e.g. BigBird, Linformer)

will be presented.

Further the concept of model distillation will be covered.

Disclaimer II:

Before you ask: Yes, GPT-3 will also be covered (but in the next chapter).

Limitations // Shortcomings of BERT

Pretrain-finetune discrepancy

- BERT *artificially* introduces [MASK] tokens during pre-training
- [MASK]-token does not occur during fine-tuning
 - Lacks the ability to model joint probabilities
 - Assumes independence of predicted tokens (given the context)

Maximum sequence length

- Based on the encoder part of the Transformer
 - Computational complexity of Self-Attention mechanism scales quadratically with the sequence length
- BERT can only consume sequences of length ≤ 512

Autoregressive (AR) language modeling vs. Autoencoding (AE)

- AR: Factorizes likelihood to $p(\mathbf{x}) = \prod_{t=1}^T p(x_t | \mathbf{x} < t)$
- Only uni-directional (backward factorization instead would also be possible)
- AE: Objective to reconstruct masked tokens $\bar{\mathbf{x}}$ from corrupted sequence $\hat{\mathbf{x}}$: $p(\bar{\mathbf{x}}|\hat{\mathbf{x}}) = \prod_{t=1}^T m_t \cdot p(x_t|\hat{\mathbf{x}})$, m_t as masking indicator

Drawbacks / Advantages

- No corruption of input sequences when using AR approach
- AE approach induces independence assumption between corrupted tokens
- AR approach only conditions on left side context
⇒ No bidirectionality

Alternative objective funktion

Permutation language modeling (PLM)

- Solves the pretrain-finetune discrepancy
- Allows for bidirectionality while keeping AR objective
- Consists of two "*streams*" of the Attention mechanism
 - ▶ Content-stream attention
 - ▶ Query-stream attention

Manipulating the factorization order

- Consider permutations \mathbf{z} of the index sequence $[1, 2, \dots, T]$
→ Used to permute the factorization order, *not* the sequence order.
- Original sequence order is retained by using positional encodings
- PLM objective (with \mathcal{Z}_T as set of all possible permutations):

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z} \sim \mathcal{Z}_T} \left[\sum_{t=1}^T \log p_{\theta}(x_{z_t} | \mathbf{x}_{\mathbf{z}_{<t}}) \right]$$

Content- vs. Query-stream

Content-stream

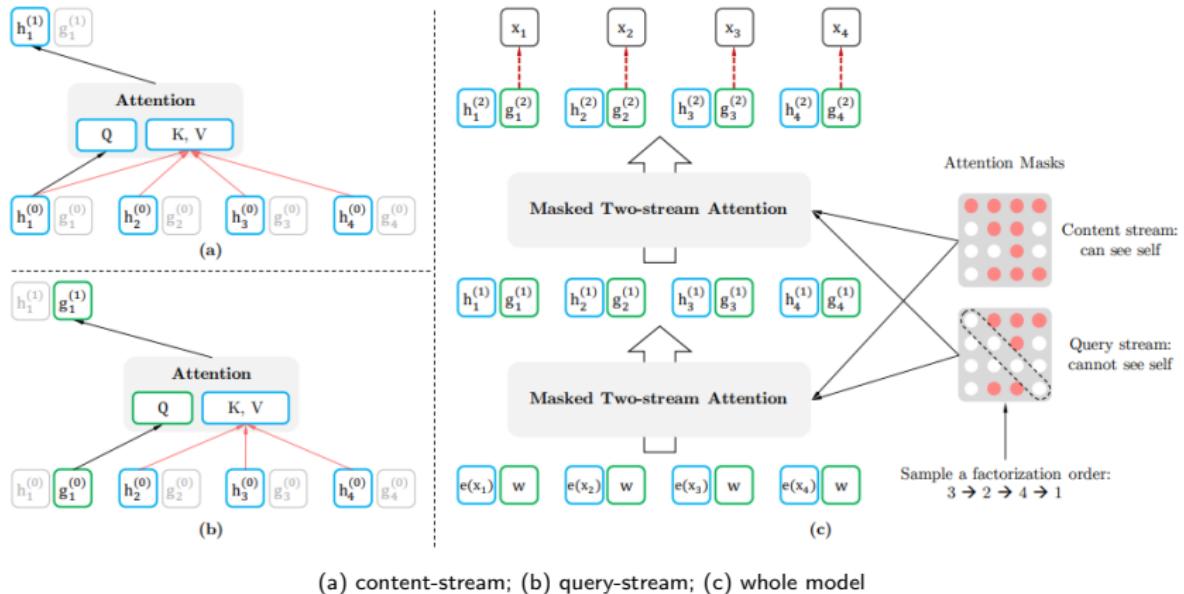
- "Normal" Self-Attention (despite with special attention masks)
→ Attentions masks depend on the factorization order
- Info about the position in the sequence is lost, see [Example in A.1](#)
- Sets of queries (Q), keys (K) and values (V) from content stream*
- Yields a *content embedding* denoted as $h_\theta(\mathbf{x}_{\mathbf{z}_{\leq t}})$

Query-stream

- Access to context through content-stream, but no access to the content of the current position (only location information)
- Q from the query stream, K and V from the content stream*
- Yields a *query embedding* denoted as $g_\theta(\mathbf{x}_{\mathbf{z}_{< t}}, z_t)$

*For a nice visual disentanglement, see [Figures in A.7](#)

XLNet – Graphical representation



(a) content-stream; (b) query-stream; (c) whole model

Source: Yang et al. (2019)

XLNet – Model Input

Generation of samples:

- Randomly sample two sentences and use concatenation* as input

[CLS]	The	fox	is	quick	.	[SEP]	
It	jumps	over	the	lazy	dog	.	[SEP]

*Nevertheless: XLNet does *not* use the NSP objective

Additional encodings:

- Relative segment encodings:
 - ▶ BERT adds absolute segment embeddings (E_A & E_B)
 - ▶ XLNet uses relative encodings (s_+ & s_-)
- Relative Positional encodings:
 - ▶ BERT encodes information about the absolute position (E_0, E_1, E_2, \dots)
 - ▶ XLNet uses relative encodings (R_{i-j})

XLNet – Special remarks

- **Partial Prediction:** Only predict the last tokens in a factorization order (reduces optimization difficulty)

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z} \sim \mathcal{Z}_T} \left[\sum_{t=c+1}^{|\mathbf{z}|} \log p_{\theta}(x_{z_t} | \mathbf{x}_{\mathbf{z}_{<t}}) \right], \quad \text{with } c \text{ as cutting point}$$

- **Segment recurrence mechanism:** Allow for learning extended contexts in Transformer-based architectures, see [Dai et al. \(2019\)](#)
- **No independence assumption:**

$$\mathcal{J}_{BERT} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{is a city}),$$

$$\mathcal{J}_{XLNet} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{New, is a city})$$

Prediction of [New, York] given the factorization order [is, a, city, New, York]

Source: Yang et al. (2019)

XLNet – SOTA performance

Performance differences to BERT:

Model	SQuAD1.1	SQuAD2.0	RACE	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B
BERT-Large (Best of 3)	86.7/92.8	82.8/85.5	75.1	87.3	93.0	91.4	74.0	94.0	88.7	63.7	90.2
XLNet-Large-wikibooks	88.2/94.0	85.1/87.8	77.4	88.4	93.9	91.8	81.2	94.4	90.0	65.2	91.1

Table 1: Fair comparison with BERT. All models are trained using the same data and hyperparameters as in BERT. We use the best of 3 BERT variants for comparison; i.e., the original BERT, BERT with whole word masking, and BERT without next sentence prediction.

Source: Yang et al. (2019)

SOTA performance:

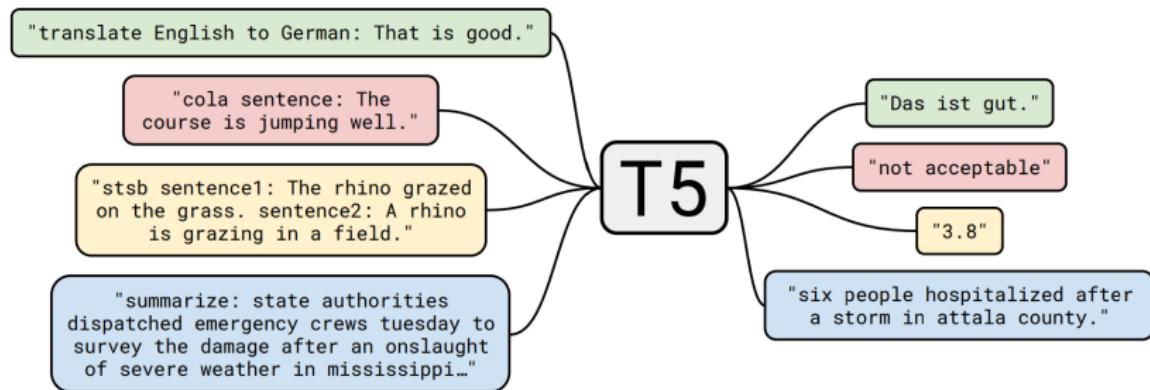
Model	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	WNLI
<i>Single-task single models on dev</i>									
BERT [2]	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-
RoBERTa [21]	90.2/90.2	94.7	92.2	86.6	96.4	90.9	68.0	92.4	-
XLNet	90.8/90.8	94.9	92.3	85.9	97.0	90.8	69.0	92.5	-
<i>Multi-task ensembles on test (from leaderboard as of Oct 28, 2019)</i>									
MT-DNN* [20]	87.9/87.4	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0
RoBERTa* [21]	90.8/90.2	98.9	90.2	88.2	96.7	92.3	67.8	92.2	89.0
XLNet*	90.9/90.9[†]	99.0[†]	90.4[†]	88.5	97.1[†]	92.9	70.2	93.0	92.5

Table 5: Results on GLUE. * indicates using ensembles, and † denotes single-task results in a multi-task row. All dev results are the median of 10 runs. The upper section shows direct comparison on dev data and the lower section shows comparison with state-of-the-art results on the public leaderboard.

Source: Yang et al. (2019)

T5: Text-to-Text Transfer Transformer:

- A complete encoder-decoder Transformer architecture
- All tasks reformulated as text-to-text tasks
- From BERT-size up to 11 Billion parameters



Source: Raffel et al. (2019)

T5 – The Colossal Clean Crawled Corpus (C4)

- Effort to measure the effect of quality, characteristics & size of the pre-training resources
- Common Crawl as basis, careful cleaning and filtering for English language
- Orders of magnitude larger (750GB) compared to commonly used corpora

Experiments (with respect to C4)

Data set	Size	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ C4	745GB	83.28	19.24	80.88	71.36	26.98	39.82	27.65
C4, unfiltered	6.1TB	81.46	19.14	78.78	68.04	26.55	39.34	27.21
RealNews-like	35GB	83.83	19.23	80.39	72.38	26.75	39.90	27.48
WebText-like	17GB	84.03	19.31	81.42	71.40	26.80	39.74	27.59
Wikipedia	16GB	81.85	19.31	81.29	68.01	26.94	39.69	27.67
Wikipedia + TBC	20GB	83.65	19.28	82.08	73.24	26.77	39.63	27.57
Number of tokens	Repeats	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Full data set	0	83.28	19.24	80.88	71.36	26.98	39.82	27.65
2^{29}	64	82.87	19.19	80.97	72.03	26.83	39.74	27.63
2^{27}	256	82.62	19.20	79.78	69.97	27.02	39.71	27.33
2^{25}	1,024	79.55	18.57	76.27	64.76	26.38	39.56	26.80
2^{23}	4,096	76.34	18.33	70.92	59.29	26.37	38.84	25.81

Source: Raffel et al. (2019)

T5 - Exhaustive Experiments

Performed experiments with respect to ..

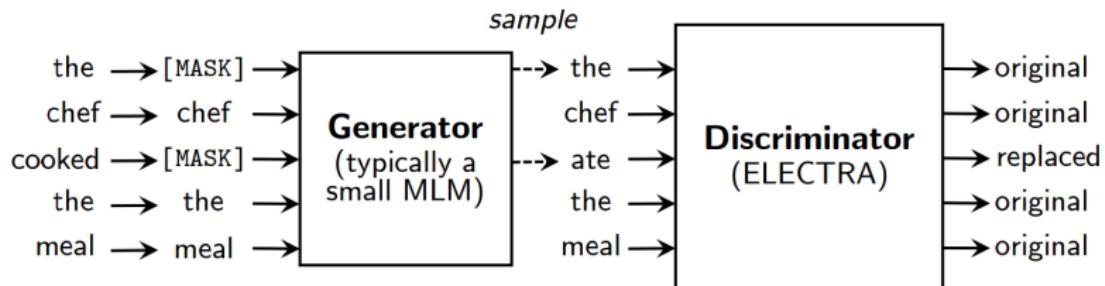
- .. architecture, size & objective
- .. details of the Denoising objective
- .. fine-tuning methods & multi-taks learning strategies

Conclusions

- Encoder-decoder architecture works best in this "text-to-text" setting
- More data, larger models & ensembling all boost the performance
 - ▶ Larger models trained for fewer steps better than smaller models on more data
 - ▶ Ensembling: Using same base pre-trained models worse than complete separate model ensembles
- Different denoising objectives work similarly well
- Updating *all* model parameters during fine-tuning works best (but expensive)

ELECTRA: a different pre-training objective

- (Small) generator model G + (large) Discriminator model D
- Generator task: Masked language modeling
- Discriminator task: *Replaced token detection*
- ELECTRA learns from *all* of the tokens (not just from a small portion, like e.g. BERT)



Source: Clark et al. (2020)

ELECTRA – Training details

Joint pre-training (but not in a GAN-fashion):

- G and D are (Transformer) encoders which are trained jointly
- G replaces [MASK]s in an input sequence
→ Passes corrupted input sequence $\mathbf{x}^{\text{corrupt}}$ to D
- *Generation of samples:*

$$m_i \sim \text{unif}\{1, n\} \text{ for } i = 1 \text{ to } k \quad \mathbf{x}^{\text{masked}} = \text{REPLACE}(\mathbf{x}, \mathbf{m}, [\text{MASK}])$$

$$\hat{x}_i \sim p_G(x_i | \mathbf{x}^{\text{masked}}) \text{ for } i \in \mathbf{m} \quad \mathbf{x}^{\text{corrupt}} = \text{REPLACE}(\mathbf{x}, \mathbf{m}, \hat{\mathbf{x}})$$

with approx. 15% of the tokens masked out (via choice of k)

- D predicts whether x_t , $t \in 1, \dots, T$ is "real" or generated by G
 - ▶ Softmax output layer for G (probability distr. over all words)
 - ▶ Sigmoid output layer for D (Binary classification real vs. generated)

ELECTRA – Training details

Using the masked & corrupted input sequences, the (joint) loss can be written down as follows:

Loss functions:

$$\mathcal{L}_{\text{MLM}}(\mathbf{x}, \theta_G) = \mathbb{E} \left(\sum_{i \in \mathbf{m}} -\log p_G(x_i | \mathbf{x}^{\text{masked}}) \right)$$

$$\mathcal{L}_{\text{Disc}}(\mathbf{x}, \theta_D) = \mathbb{E} \left(\sum_{t=1}^n -\mathbb{1}(x_t^{\text{corrupt}} = x_t) \log D(\mathbf{x}^{\text{corrupt}}, t) - \mathbb{1}(x_t^{\text{corrupt}} \neq x_t) \log(1 - D(\mathbf{x}^{\text{corrupt}}, t)) \right)$$

Combined:

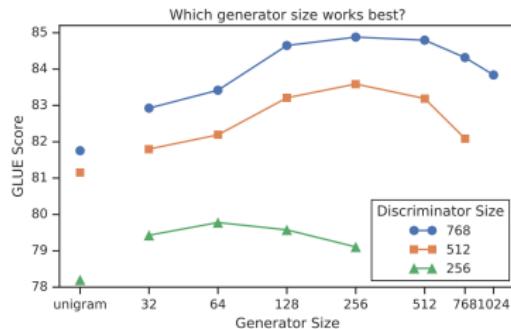
$$\min_{\theta_G, \theta_D} \sum_{\mathbf{x} \in \mathcal{X}} \mathcal{L}_{\text{MLM}}(\mathbf{x}, \theta_G) + \lambda \mathcal{L}_{\text{Disc}}(\mathbf{x}, \theta_D)$$

with λ set to 50, since the discriminator's loss is typically much lower than the generator's.

ELECTRA – Training details

Generator size:

- Same size of G and D :
 - ▶ Twice as much compute per training step + too challenging for D
- Smaller Generators are preferable (1/4 – 1/2 the size of D)

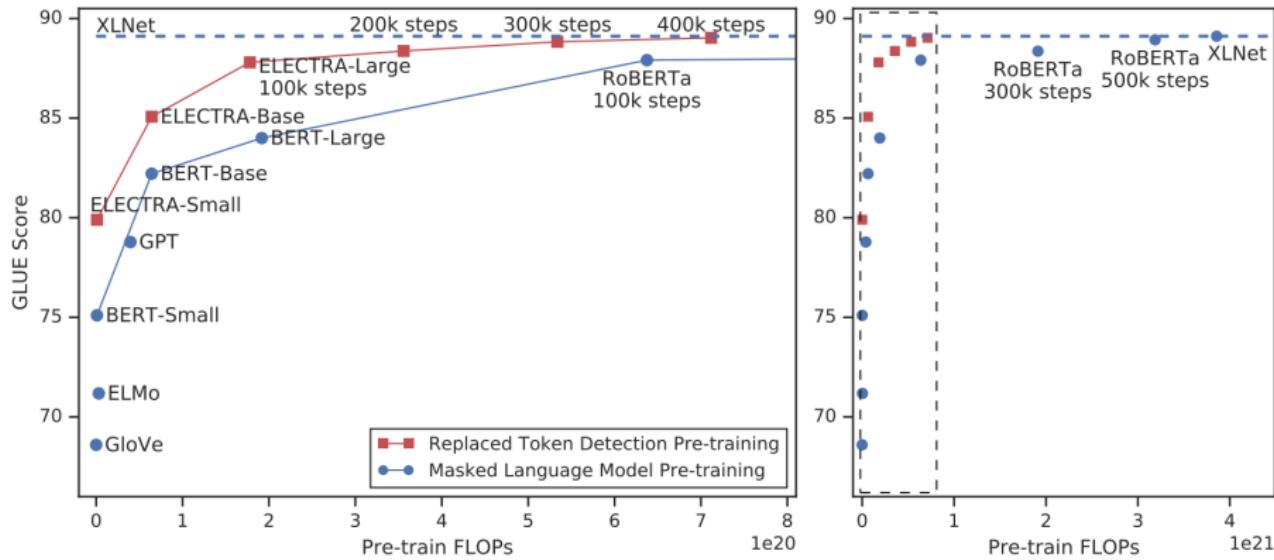


Source: Clark et al. (2020)

Weight sharing (experimental):

- Same size of G and D : All weights can be tied
- G smaller than D : Share token & positional embeddings

ELECTRA – Model comparison



Source: Clark et al. (2020)

Note: Different batch sizes (2k vs. 8k) for ELECTRA vs. RoBERTa/XLNet explain why same number of steps lead to approx. 1/4 of the compute for ELECTRA.

ELECTRA – SOTA performance

Performance differences vs. BERT/RoBERTa (GLUE dev set):

Model	Train FLOPs	Params	CoLA	SST	MRPC	STS	QQP	MNLI	QNLI	RTE	Avg.
BERT	1.9e20 (0.27x)	335M	60.6	93.2	88.0	90.0	91.3	86.6	92.3	70.4	84.0
RoBERTa-100K	6.4e20 (0.90x)	356M	66.1	95.6	91.4	92.2	92.0	89.3	94.0	82.7	87.9
RoBERTa-500K	3.2e21 (4.5x)	356M	68.0	96.4	90.9	92.1	92.2	90.2	94.7	86.6	88.9
XLNet	3.9e21 (5.4x)	360M	69.0	97.0	90.8	92.2	92.3	90.8	94.9	85.9	89.1
BERT (ours)	7.1e20 (1x)	335M	67.0	95.9	89.1	91.2	91.5	89.6	93.5	79.5	87.2
ELECTRA-400K	7.1e20 (1x)	335M	69.3	96.0	90.6	92.1	92.4	90.5	94.5	86.8	89.0
ELECTRA-1.75M	3.1e21 (4.4x)	335M	69.1	96.9	90.8	92.6	92.4	90.9	95.0	88.0	89.5

Source: Clark et al. (2020)

SOTA performance (GLUE test set):

Model	Train FLOPs	CoLA	SST	MRPC	STS	QQP	MNLI	QNLI	RTE	WNLI	Avg.*	Score
BERT	1.9e20 (0.06x)	60.5	94.9	85.4	86.5	89.3	86.7	92.7	70.1	65.1	79.8	80.5
RoBERTa	3.2e21 (1.02x)	67.8	96.7	89.8	91.9	90.2	90.8	95.4	88.2	89.0	88.1	88.1
ALBERT	3.1e22 (10x)	69.1	97.1	91.2	92.0	90.5	91.3	–	89.2	91.8	89.0	–
XLNet	3.9e21 (1.26x)	70.2	97.1	90.5	92.6	90.4	90.9	–	88.5	92.5	89.1	–
ELECTRA	3.1e21 (1x)	71.7	97.1	90.7	92.5	90.8	91.3	95.8	89.8	92.5	89.5	89.4

* Avg. excluding QNLI to ensure comparability

Source: Clark et al. (2020)

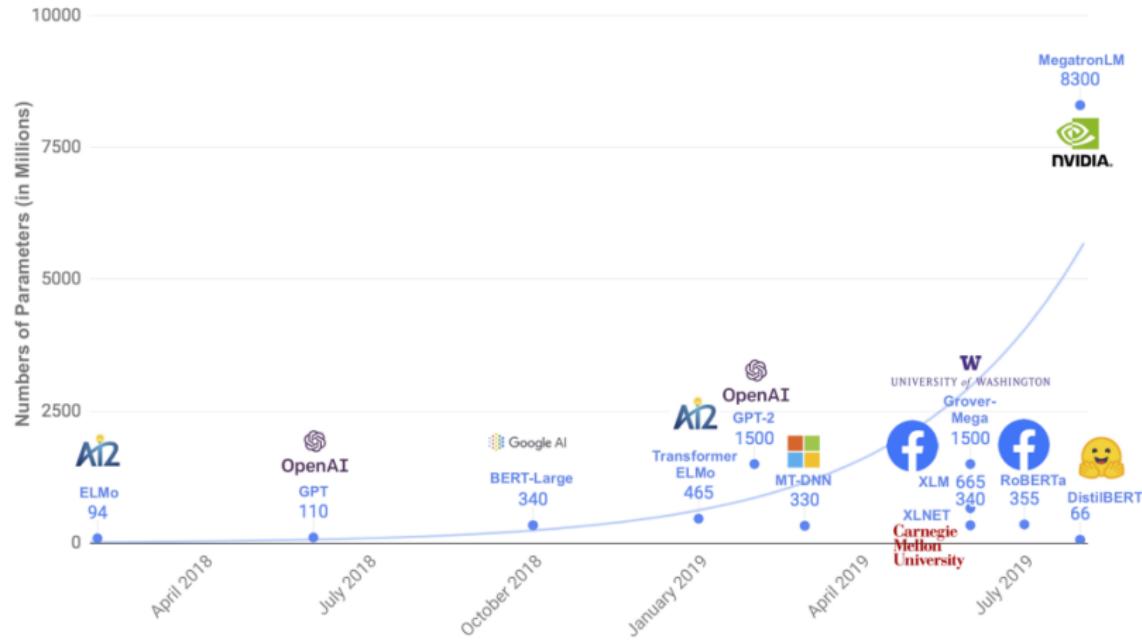
Model compression scheme:

- Motivation comes from having computationally expensive, cumbersome ensemble models. ► Bucila et al. (2006)
- Compressing the knowledge of the ensemble into a single model has the benefit of easier deployment and better generalization
- Reasoning:
 - ▶ Cumbersome model generalizes well, because it is the average of an ensemble.
 - ▶ Small model trained to generalize in the same way typically better than small model trained "the normal way".

Distillation:

- Temperature T in the softmax: $q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$
- Knowledge transfer via soft targets with high T from original model.
- When true labels are known: Weighted average of two different objective functions

Motivation:



Source: Sanh et al. (2019)

Student architecture (DistilBERT):

- Half the number of layers compared to BERT*
- Half of the size of BERT, but retains 95% of the performance
- Initialize from BERT (taking one out of two hidden layers)
- Same pre-training data as BERT (Wiki + BooksCorpus)

Training and performance

- Distillation loss $L_{ce} = \sum_i t_i \cdot \log(s_i) + \text{MLM-Loss } L_{mlm} + \text{Cosine-Embedding-Loss } L_{cos}$
- Drops NSP, use dynamic masking, train with large batches

*Rationale for "only" reducing the number of layers:

Larger influence on the computation efficiency compared to e.g. hidden size dimension

Performance differences to BERT:

Table 1: **DistilBERT retains 97% of BERT performance.** Comparison on the dev sets of the GLUE benchmark. ELMo results as reported by the authors. BERT and DistilBERT results are the medians of 5 runs with different seeds.

Model	Score	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
ELMo	68.7	44.1	68.6	76.6	71.1	86.2	53.4	91.5	70.4	56.3
BERT-base	79.5	56.3	86.7	88.6	91.8	89.6	69.3	92.7	89.0	53.5
DistilBERT	77.0	51.3	82.2	87.5	89.2	88.5	59.9	91.3	86.9	56.3

Source: Sanh et al. (2019)

Ablation study regarding the loss:

Table 4: **Ablation study.** Variations are relative to the model trained with triple loss and teacher weights initialization.

Ablation	Variation on GLUE macro-score
$\emptyset - L_{cos} - L_{mlm}$	-2.96
$L_{ce} - \emptyset - L_{mlm}$	-1.46
$L_{ce} - L_{cos} - \emptyset$	-0.31
Triple loss + random weights initialization	-3.69

Source: Sanh et al. (2019)

The $\mathcal{O}(n^2)$ problem

Quadratic time & memory complexity of Self-Attention

- *Inductive bias of Transformer models:*
Connect all tokens in a sequence to each other
- **Pro:** Can (theoretically) learn contexts of arbitrary length
- **Con:** Bad scalability limiting (feasible) context size

Resulting Problems:

- Several tasks require models to consume longer sequences
- *Efficiency:* Are there more efficient modifications which achieve similar or even better performance?

Broad overview on so-called "X-formers":

- Efficient & fast Transformer-based models
 - Reduce complexity from $\mathcal{O}(n^2)$ to (up to) $\mathcal{O}(n)$
- Claim on-par (or even) superior performance
- Different techniques used:
 - ▶ Fixed/Factorized/Random Patterns
 - ▶ Learnable Patterns (extension of the above)
 - ▶ Low-Rank approximations or Kernels
 - ▶ Recurrence (see e.g. ▶ [Transformer-XL \(Dai et al., 2019\)](#))
 - ▶ Memory modules

Side note:

- Most Benchmark data sets not explicitly designed for evaluating long-range abilities of the models.
- Recently proposed: ▶ [Long Range Arena: A benchmark for efficient transformers](#) (Tay et al., 2020)

Introducing Patterns

Reasoning:

- Making every token attend to every other token might be unnecessary
- Introduce sparsity in the commonly dense attention matrix

Example:

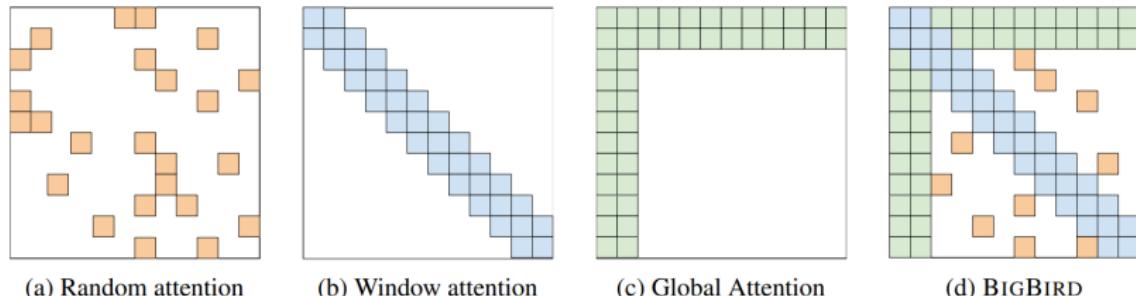
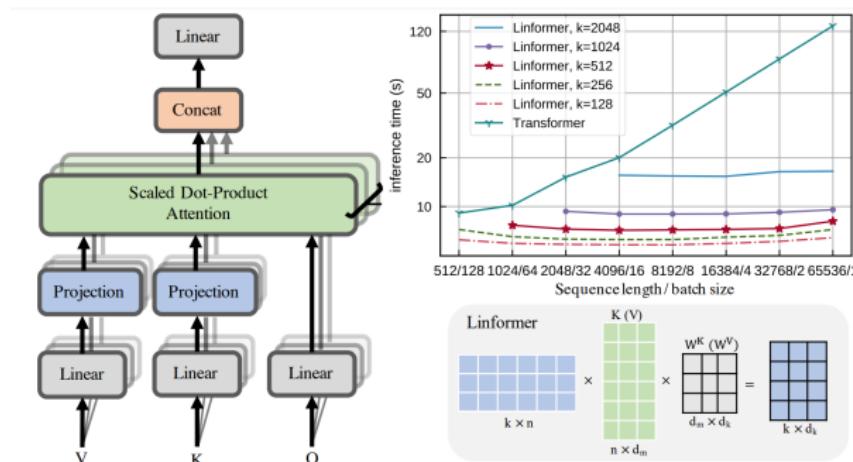


Figure 1: Building blocks of the attention mechanism used in BIGBIRD. White color indicates absence of attention. (a) random attention with $r = 2$, (b) sliding window attention with $w = 3$ (c) global attention with $g = 2$. (d) the combined BIGBIRD model.

Source: Zaheer et al. (2020)

Reasoning:

- Most information in the Self-Attention mechanism can be recovered from the first few, largest singular values
- Introduce additional k -dimensional projection before self-attention



Source: Wang et al. (2020)

Disentangled Attention:

- Each token represented by two vectors for content (\mathbf{H}_i) and relative position ($\mathbf{P}_{i|j}$)
- Calculation of the Attention Score:

$$\begin{aligned} A_{i,j} &= \{\mathbf{H}_i, \mathbf{P}_{i|j}\} \times \{\mathbf{H}_j, \mathbf{P}_{j|i}\}^\top \\ &= \mathbf{H}_i \mathbf{H}_j^\top + \mathbf{H}_i \mathbf{P}_{j|i}^\top + \mathbf{P}_{i|j} \mathbf{H}_j^\top + \mathbf{P}_{i|j} \mathbf{P}_{j|i}^\top \end{aligned}$$

- with content-to-content, content-to-position, position-to-content and position-to-position attention

Disentangled Attention

Standard (Single-head) Self-Attention:

$$Q = HW_q, K = HW_k, V = HW_v, A = \frac{QK^\top}{\sqrt{d}}$$
$$H_o = \text{softmax}(A)V$$

Disentangled Attention*:

$$Q_c = HW_{q,c}, K_c = HW_{k,c}, V_c = HW_{v,c}, Q_r = PW_{q,r}, K_r = PW_{k,r}$$

$$\tilde{A}_{i,j} = \underbrace{Q_i^c K_j^{c\top}}_{\text{(a) content-to-content}} + \underbrace{Q_i^c K_{\delta(i,j)}^r {}^\top}_{\text{(b) content-to-position}} + \underbrace{K_j^c Q_{\delta(j,i)}^r {}^\top}_{\text{(c) position-to-content}}$$

$$H_o = \text{softmax}\left(\frac{\tilde{A}}{\sqrt{3d}}\right)V_c$$

*Position-to-position part is removed since it, according to the authors, does not provide much additional information as *relative* position embeddings are used.

Further reading

- BART ▶ Lewis et al. (2019)
- BORT ▶ de Wynter & Perry (2020)
- WT5?! ▶ Narang et al. (2020)

Chapter 12: GPT Models

Hinrich Schütze

February 3, 2021

Outline

1 GPT: Intro

2 GPT3 results on tasks

3 GPT limitations

4 GPT: Discussion

Outline

1 GPT: Intro

2 GPT3 results on tasks

3 GPT limitations

4 GPT: Discussion

Recap: BERT, RoBERTa etc.

- Transformer
- Training: Masked language modeling (MLM)
- BERT learns an enormous amount of knowledge about language and the world through MLM training on large corpora.
- Application: finetune on a particular task
- Great performance!
- What's not to like?
- (In what follows I will use BERT as a representative for this class of language models and only talk about BERT – but the discussion includes RoBERTa, Albert, XLNet etc.)

Problems with BERT (1)

- You need a different model for each task.
- (Because BERT is differently finetuned for each task.)
 - ▶ Not realistic in many real deployment scenarios, e.g., on mobile devices.
- Human learning: we arguably have a **single** model that solves all tasks!
- Question: Is there a framework that allows us to create a single model that solves all tasks?

Problems with BERT (2)

- BERT has two training modes, first (MLM) pretraining, then finetuning.
- Finetuning is **supervised learning**, i.e., learning from labeled examples.
- Arguably, learning from labeled examples is untypical for human learning.
- You never learn a task solely by being presented a bunch of examples, without explanation.
- Instead, in human learning, there is almost always a **task description**.
- Example: How to boil an egg. “Place eggs in the bottom of a saucepan. Fill the pan with cold water. Etc.”
- (Notice that this is **not** an example.)
- Question: Is there a framework that allows us to leverage task descriptions?

Problems with BERT (3)

- BERT has great performance, but ...
- ... it only has great performance if the training set is fairly large, generally 1000s of examples.
- This is completely different from human learning!
- We do use examples in learning, but in most cases, only a few.
- Example: Maybe the person teaching you how to boil an egg will show you how to do it one or two times.
- But probably not 10 times
- Definitely not a 1000 times
- More practical concern: it's very expensive to label 1000s of examples for each task (there are many many tasks).
- Question: Is there a framework that allows us to learn from just a small number of examples?
- This is called **few-shot learning**.

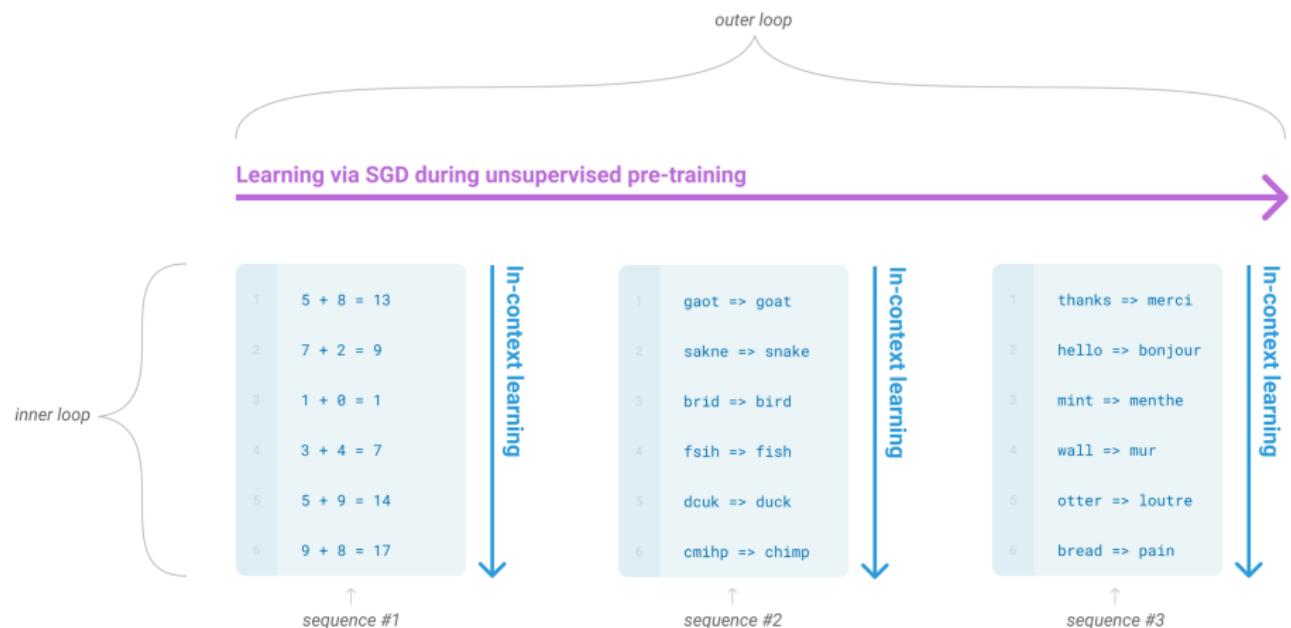
Problems with BERT (4)

- More subtle aspect of the same problem (i.e., large training sets): overfitting
- Even though performance looks good on standard train/dev/test splits,
- the deviation between the training set and the data actually encountered in real application can be large.
- So our benchmarks often overestimate what performance would be in reality.

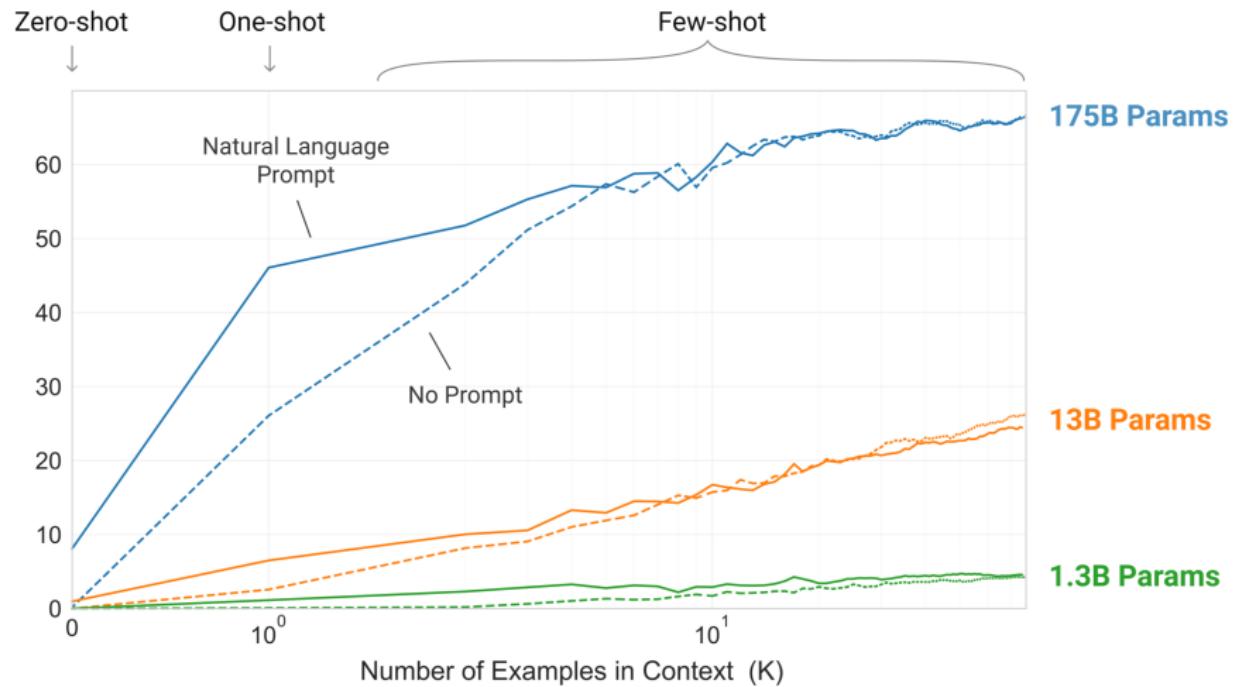
GPT

- Like BERT, GPT is a language model.
- But not MLM, but a conventional language model: it predicts the next word (or subword).
- Like BERT, GPT is trained on a huge corpus, actually an even huger corpus.
- Like BERT, GPT is a transformer architecture.
- Difference 1: GPT is a **single model** that aims to solve **all tasks**.
 - ▶ It can also switch back and forth between tasks and solve tasks within tasks, another human capability that is important in practice. “**fluidity**”
- Difference 2: GPT leverages **task descriptions**.
- Difference 3: GPT is effective at **few-shot learning**.

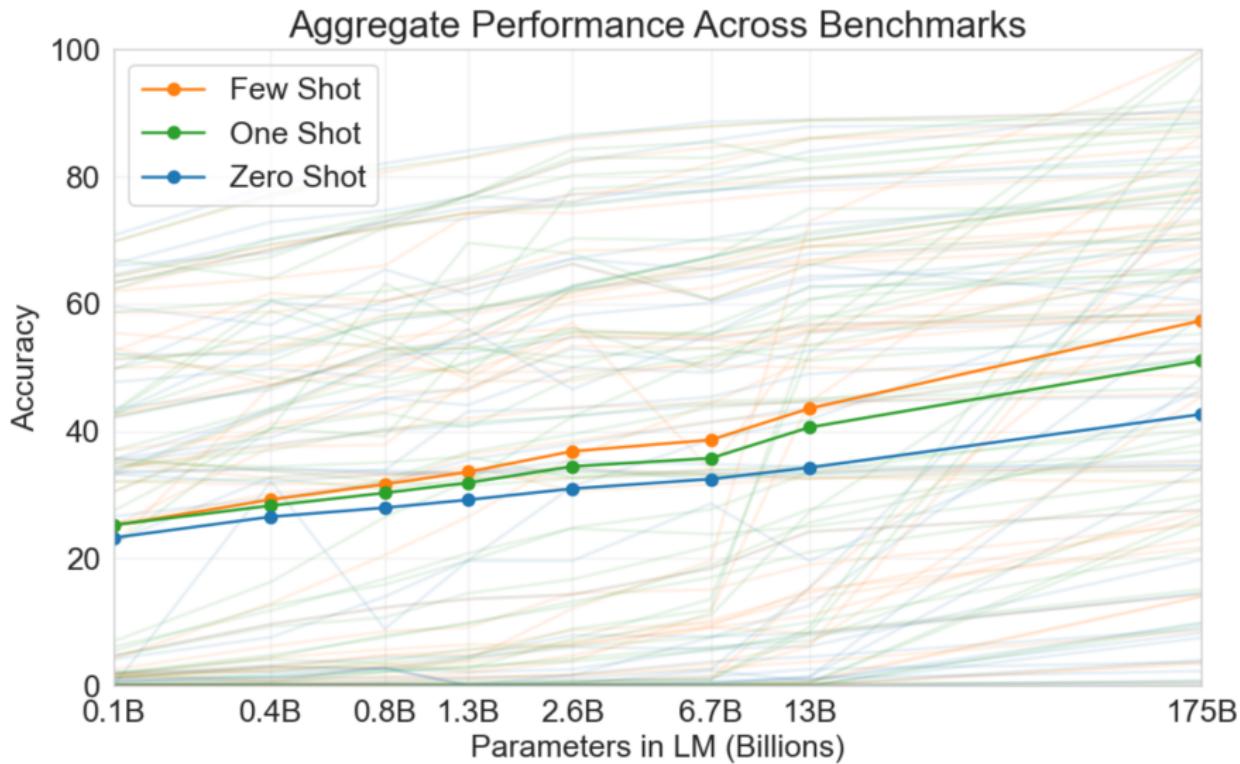
GPT: Two types of learning



GPT: Effective in-context learning



X-shot comparison and effect of larger corpora



Fine-tuning (not used by GPT)

Traditional fine-tuning (not used for GPT-3)

Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.



Zero-shot (no gradient update)

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

- 1 Translate English to French: ← *task description*
- 2 cheese => ← *prompt*

One-shot (no gradient update)

One-shot

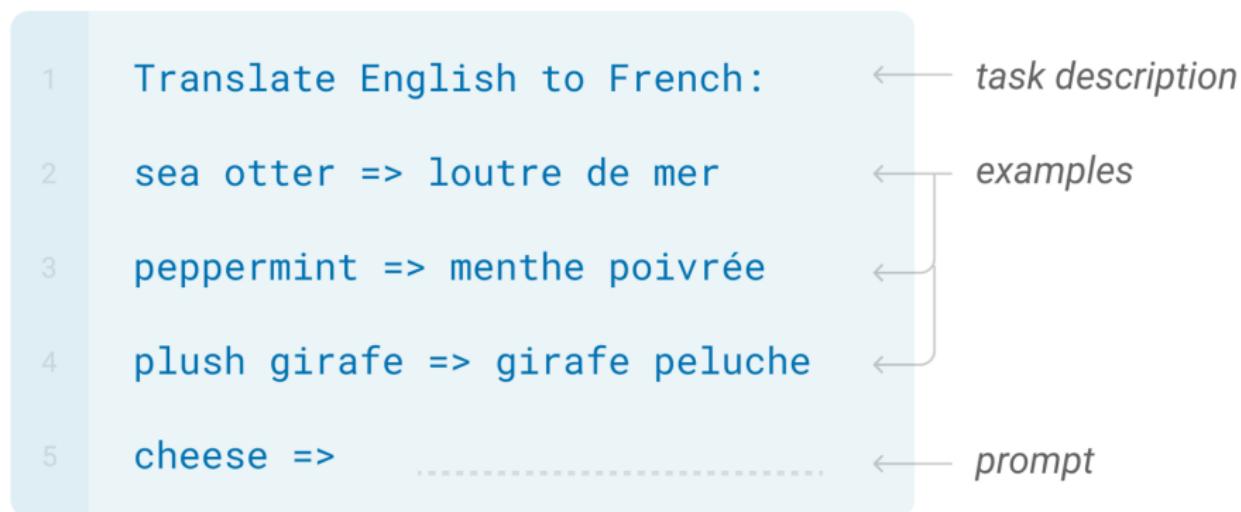
In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

- 1 Translate English to French: ← *task description*
- 2 sea otter => loutre de mer ← *example*
- 3 cheese => ← *prompt*

Few-shot (no gradient update)

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



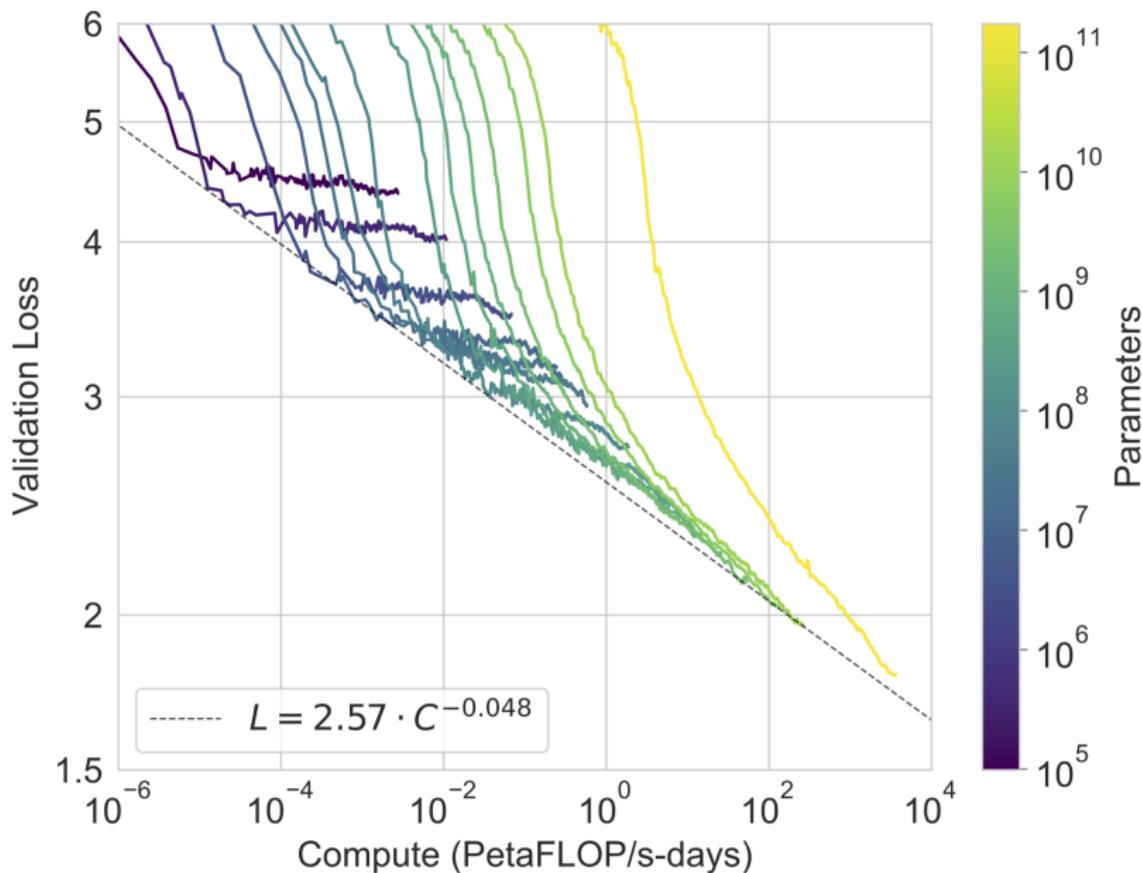
Architecture

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or “GPT-3”	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Training corpus

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Loss as a function of compute



Outline

1 GPT: Intro

2 GPT3 results on tasks

3 GPT limitations

4 GPT: Discussion

Lambada task

Context → Fill in blank:

She held the torch in front of her.

She caught her breath.

"Chris? There's a step."

"What?"

"A step. Cut in the rock. About fifty feet ahead." She moved faster.
They both moved faster. "In fact," she said, raising the torch higher,
"there's more than a _____. ->

Target Completion → step

Performance on lambada

Setting	LAMBADA (acc)	LAMBADA (ppl)	StoryCloze (acc)	HellaSwag (acc)
SOTA	68.0 ^a	8.63 ^b	91.8^c	85.6^d
GPT-3 Zero-Shot	76.2	3.00	83.2	78.9
GPT-3 One-Shot	72.5	3.35	84.7	78.1
GPT-3 Few-Shot	86.4	1.92	87.7	79.3

“Closed book” question answering (QA) task

Context → Q: ‘Nude Descending A Staircase’ is perhaps the most famous painting by which 20th century artist?

A:

Target Completion → MARCEL DUCHAMP
Target Completion → r mutt
Target Completion → duchamp
Target Completion → marcel duchamp
Target Completion → R.Mutt
Target Completion → Marcel duChamp
Target Completion → Henri-Robert-Marcel Duchamp
Target Completion → Marcel du Champ
Target Completion → henri robert marcel duchamp
Target Completion → Duchampian
Target Completion → Duchamp
Target Completion → duchampian
Target Completion → marcel du champ
Target Completion → Marcel Duchamp
Target Completion → MARCEL DUCHAMP

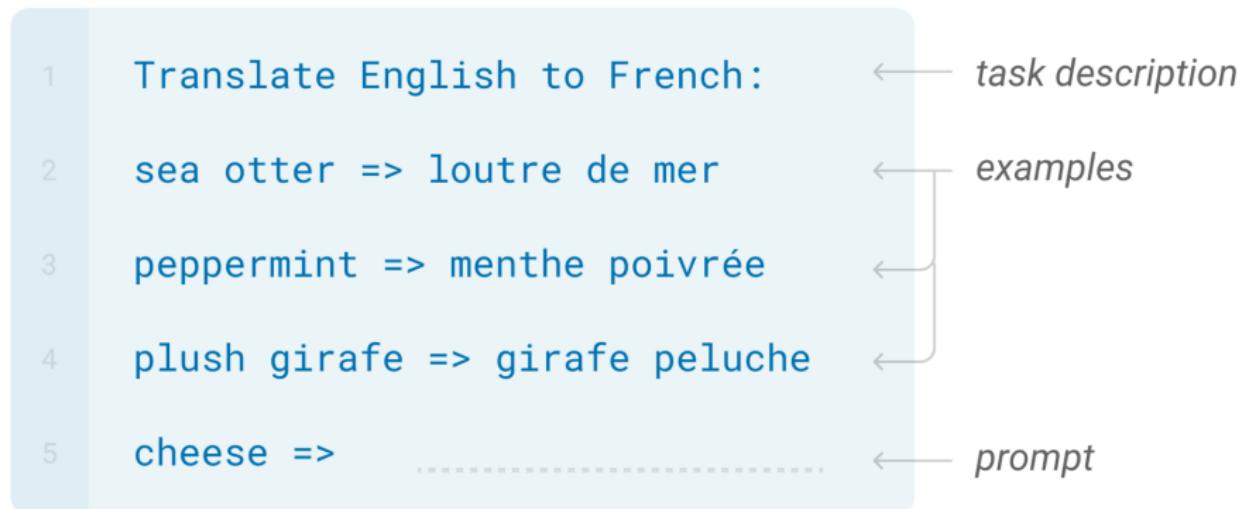
Performance on closed-book QA task

Setting	NaturalQS	WebQS	TriviaQA
RAG (Fine-tuned, Open-Domain) [LPP ⁺ 20]	44.5	45.5	68.0
T5-11B+SSM (Fine-tuned, Closed-Book) [RRS20]	36.6	44.7	60.5
T5-11B (Fine-tuned, Closed-Book)	34.5	37.4	50.1
GPT-3 Zero-Shot	14.6	14.4	64.3
GPT-3 One-Shot	23.0	25.3	68.0
GPT-3 Few-Shot	29.9	41.5	71.2

Few-shot (no gradient update)

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



Performance on machine translation

Setting	En→Fr	Fr→En	En→De	De→En	En→Ro	Ro→En
SOTA (Supervised)	45.6^a	35.0 ^b	41.2^c	40.2 ^d	38.5^e	39.9^e
XLM [LC19]	33.4	33.3	26.4	34.3	33.3	31.8
MASS [STQ ⁺ 19]	<u>37.5</u>	34.9	28.3	35.2	<u>35.2</u>	33.1
mBART [LGG ⁺ 20]	-	-	<u>29.8</u>	34.0	35.0	30.5
GPT-3 Zero-Shot	25.2	21.2	24.6	27.2	14.1	19.9
GPT-3 One-Shot	28.3	33.7	26.2	30.4	20.6	38.6
GPT-3 Few-Shot	32.6	<u>39.2</u>	29.7	<u>40.6</u>	21.0	<u>39.5</u>

Winograd task

Correct Context → Grace was happy to trade me her sweater for my jacket. She thinks the sweater

Incorrect Context → Grace was happy to trade me her sweater for my jacket. She thinks the jacket

Target Completion → looks dowdy on her.

Performance on Winograd task

Setting	Winograd	Winogrande (XL)
Fine-tuned SOTA	90.1^a	84.6^b
GPT-3 Zero-Shot	88.3*	70.2
GPT-3 One-Shot	89.7*	73.2
GPT-3 Few-Shot	88.6*	77.7

ARC task

Context → Question: George wants to warm his hands quickly by rubbing them. Which skin surface will produce the most heat?

Answer:

Correct Answer → dry palms

Incorrect Answer → wet palms

Incorrect Answer → palms covered with oil

Incorrect Answer → palms covered with lotion

Performance on ARC task

Setting	PIQA	ARC (Easy)	ARC (Challenge)	OpenBookQA
Fine-tuned SOTA	79.4	92.0 [KKS ⁺²⁰]	78.5 [KKS ⁺²⁰]	87.2 [KKS ⁺²⁰]
GPT-3 Zero-Shot	80.5 *	68.8	51.4	57.6
GPT-3 One-Shot	80.5 *	71.2	53.2	58.8
GPT-3 Few-Shot	82.8 *	70.1	51.5	65.4

RACE task

Context → Article:

Informal conversation is an important part of any business relationship. Before you start a discussion, however, make sure you understand which topics are suitable and which are considered taboo in a particular culture. Latin Americans enjoy sharing information about their local history, art and customs. You may expect questions about your family, and be sure to show pictures of your children. You may feel free to ask similar questions of your Latin American friends. The French think of conversation as an art form, and they enjoy the value of lively discussions as well as disagreements. For them, arguments can be interesting and they can cover pretty much or any topic ---- as long as they occur in a respectful and intelligent manner.

In the United States, business people like to discuss a wide range of topics, including opinions about work, family, hobbies, and politics. In Japan, China, and Korea, however, people are much more private. They do not share much about their thoughts, feelings, or emotions because they feel that doing so might take away from the harmonious business relationship they're trying to build. Middle Easterners are also private about their personal lives and family matters. It is considered rude, for example, to ask a businessman from Saudi Arabia about his wife or children.

As a general rule, it's best not to talk about politics or religion with your business friends. This can get you into trouble, even in the United States, where people hold different religious views. In addition, discussing one's salary is usually considered unsuitable. Sports is typically a friendly subject in most parts of the world, although be careful not to criticize national sport. Instead, be friendly and praise your host's team.

Q: What shouldn't you do when talking about sports with colleagues from another country?

A: Criticizing the sports of your colleagues' country.

Q: Which is typically a friendly topic in most places according to the author?

A: Sports.

Q: Why are people from Asia more private in their conversation with others?

A: They don't want to have their good relationship with others harmed by informal conversation.

Q: The author considers politics and religion . .

A:

Correct Answer → taboo

Incorrect Answer → cheerful topics

Incorrect Answer → rude topics

Incorrect Answer → topics that can never be talked about

Performance on RACE task

Setting	CoQA	DROP	QuAC	SQuADv2	RACE-h	RACE-m
Fine-tuned SOTA	90.7^a	89.1^b	74.4^c	93.0^d	90.0^e	93.1^e
GPT-3 Zero-Shot	81.5	23.6	41.5	59.5	45.5	58.4
GPT-3 One-Shot	84.0	34.3	43.3	65.4	45.9	57.4
GPT-3 Few-Shot	85.0	36.5	44.3	69.8	46.8	58.1

Performance on SuperGLUE task

	SuperGLUE Average	BoolQ Accuracy	CB Accuracy	CB F1	COPA Accuracy	RTE Accuracy
Fine-tuned SOTA	89.0	91.0	96.9	93.9	94.8	92.5
Fine-tuned BERT-Large	69.0	77.4	83.6	75.7	70.6	71.7
GPT-3 Few-Shot	71.8	76.4	75.6	52.0	92.0	69.0

	WiC Accuracy	WSC Accuracy	MultiRC Accuracy	MultiRC F1a	ReCoRD Accuracy	ReCoRD F1
Fine-tuned SOTA	76.1	93.8	62.3	88.2	92.5	93.3
Fine-tuned BERT-Large	69.6	64.6	24.1	70.0	71.3	72.0
GPT-3 Few-Shot	49.4	80.1	30.5	75.4	90.2	91.1

SuperGLUE

- BoolQ
- CB (true/false/neither)
- COPA
- RTE (similar to natural language inference)
- WiC
- WSC
- MultiRC (true/false)
- ReCoRD

BoolQ (Boolean Question) task

Context → Normal force -- In a simple case such as an object resting upon a table, the normal force on the object is equal but in opposite direction to the gravitational force applied on the object (or the weight of the object), that is, $N = m g$ ($\text{displaystyle } N=mg$), where m is mass, and g is the gravitational field strength (about 9.81 m/s^2 on Earth). The normal force here represents the force applied by the table against the object that prevents it from sinking through the table and requires that the table is sturdy enough to deliver this normal force without breaking. However, it is easy to assume that the normal force and weight are action-reaction force pairs (a common mistake). In this case, the normal force and weight need to be equal in magnitude to explain why there is no upward acceleration of the object. For example, a ball that bounces upwards accelerates upwards because the normal force acting on the ball is larger in magnitude than the weight of the ball.
question: is the normal force equal to the force of gravity?
answer:

Target Completion → yes

Performance on SuperGLUE task

	SuperGLUE Average	BoolQ Accuracy	CB Accuracy	CB F1	COPA Accuracy	RTE Accuracy
Fine-tuned SOTA	89.0	91.0	96.9	93.9	94.8	92.5
Fine-tuned BERT-Large	69.0	77.4	83.6	75.7	70.6	71.7
GPT-3 Few-Shot	71.8	76.4	75.6	52.0	92.0	69.0

	WiC Accuracy	WSC Accuracy	MultiRC Accuracy	MultiRC F1a	ReCoRD Accuracy	ReCoRD F1
Fine-tuned SOTA	76.1	93.8	62.3	88.2	92.5	93.3
Fine-tuned BERT-Large	69.6	64.6	24.1	70.0	71.3	72.0
GPT-3 Few-Shot	49.4	80.1	30.5	75.4	90.2	91.1

WiC (Word in Context) task

Context → An outfitter provided everything needed for the safari.
Before his first walking holiday, he went to a specialist outfitter to buy some boots.
question: Is the word ‘outfitter’ used in the same way in the two sentences above?
answer:

Target Completion → no

Performance on SuperGLUE task

	SuperGLUE Average	BoolQ Accuracy	CB Accuracy	CB F1	COPA Accuracy	RTE Accuracy
Fine-tuned SOTA	89.0	91.0	96.9	93.9	94.8	92.5
Fine-tuned BERT-Large	69.0	77.4	83.6	75.7	70.6	71.7
GPT-3 Few-Shot	71.8	76.4	75.6	52.0	92.0	69.0

	WiC Accuracy	WSC Accuracy	MultiRC Accuracy	MultiRC F1a	ReCoRD Accuracy	ReCoRD F1
Fine-tuned SOTA	76.1	93.8	62.3	88.2	92.5	93.3
Fine-tuned BERT-Large	69.6	64.6	24.1	70.0	71.3	72.0
GPT-3 Few-Shot	49.4	80.1	30.5	75.4	90.2	91.1

COPA task

Context → My body cast a shadow over the grass because

Correct Answer → the sun was rising.

Incorrect Answer → the grass was cut.

Performance on SuperGLUE task

	SuperGLUE Average	BoolQ Accuracy	CB Accuracy	CB F1	COPA Accuracy	RTE Accuracy
Fine-tuned SOTA	89.0	91.0	96.9	93.9	94.8	92.5
Fine-tuned BERT-Large	69.0	77.4	83.6	75.7	70.6	71.7
GPT-3 Few-Shot	71.8	76.4	75.6	52.0	92.0	69.0

	WiC Accuracy	WSC Accuracy	MultiRC Accuracy	MultiRC F1a	ReCoRD Accuracy	ReCoRD F1
Fine-tuned SOTA	76.1	93.8	62.3	88.2	92.5	93.3
Fine-tuned BERT-Large	69.6	64.6	24.1	70.0	71.3	72.0
GPT-3 Few-Shot	49.4	80.1	30.5	75.4	90.2	91.1

WSC (Winograd Schema Challenge) task

Context → Final Exam with Answer Key

Instructions: Please carefully read the following passages. For each passage, you must identify which noun the pronoun marked in ***bold*** refers to.

=====

Passage: Mr. Moncrieff visited Chester's luxurious New York apartment, thinking that it belonged to his son Edward. The result was that Mr. Moncrieff has decided to cancel Edward's allowance on the ground that he no longer requires ***his*** financial support.

Question: In the passage above, what does the pronoun "***his***" refer to?

Answer:

Target Completion → mr. moncrieff

Performance on SuperGLUE task

	SuperGLUE Average	BoolQ Accuracy	CB Accuracy	CB F1	COPA Accuracy	RTE Accuracy
Fine-tuned SOTA	89.0	91.0	96.9	93.9	94.8	92.5
Fine-tuned BERT-Large	69.0	77.4	83.6	75.7	70.6	71.7
GPT-3 Few-Shot	71.8	76.4	75.6	52.0	92.0	69.0

	WiC Accuracy	WSC Accuracy	MultiRC Accuracy	MultiRC F1a	ReCoRD Accuracy	ReCoRD F1
Fine-tuned SOTA	76.1	93.8	62.3	88.2	92.5	93.3
Fine-tuned BERT-Large	69.6	64.6	24.1	70.0	71.3	72.0
GPT-3 Few-Shot	49.4	80.1	30.5	75.4	90.2	91.1

ReCoRD task

Context → (CNN) Yuv al Rabin, whose father, Yitzhak Rabin, was assassinated while serving as Prime Minister of Israel, criticized Donald Trump for appealing to "Second Amendment people" in a speech and warned that the words that politicians use can incite violence and undermine democracy. "Trump's words are an incitement to the type of political violence that touched me personally," Rabin wrote in USA Today. He said that Trump's appeal to "Second Amendment people" to stop Hillary Clinton -- comments that were criticized as a call for violence against Clinton, something Trump denied -- "were a new level of ugliness in an ugly campaign season."

- The son of a former Israeli Prime Minister who was assassinated wrote an op ed about the consequence of violent political rhetoric.
 - Warns of "parallels" between Israel of the 1990s and the U.S. today.
-

Correct Answer → - Referencing his father, who was shot and killed by an extremist amid political tension in Israel in 1995, Rabin condemned Donald Trump's aggressive rhetoric.

Correct Answer → - Referencing his father, who was shot and killed by an extremist amid political tension in Israel in 1995, Rabin condemned Trump's aggressive rhetoric.

Incorrect Answer → - Referencing his father, who was shot and killed by an extremist amid political tension in Israel in 1995, Rabin condemned Hillary Clinton's aggressive rhetoric.

Incorrect Answer → - Referencing his father, who was shot and killed by an extremist amid political tension in Israel in 1995, Rabin condemned U.S.'s aggressive rhetoric.

Incorrect Answer → - Referencing his father, who was shot and killed by an extremist amid political tension in Israel in 1995, Rabin condemned Yitzhak Rabin's aggressive rhetoric.

Performance on SuperGLUE task

	SuperGLUE Average	BoolQ Accuracy	CB Accuracy	CB F1	COPA Accuracy	RTE Accuracy
Fine-tuned SOTA	89.0	91.0	96.9	93.9	94.8	92.5
Fine-tuned BERT-Large	69.0	77.4	83.6	75.7	70.6	71.7
GPT-3 Few-Shot	71.8	76.4	75.6	52.0	92.0	69.0

	WiC Accuracy	WSC Accuracy	MultiRC Accuracy	MultiRC F1a	ReCoRD Accuracy	ReCoRD F1
Fine-tuned SOTA	76.1	93.8	62.3	88.2	92.5	93.3
Fine-tuned BERT-Large	69.6	64.6	24.1	70.0	71.3	72.0
GPT-3 Few-Shot	49.4	80.1	30.5	75.4	90.2	91.1

SuperGLUE

- BoolQ
- CB (true/false/neither)
- COPA
- RTE (similar to natural language inference)
- WiC
- WSC
- MultiRC (true/false)
- ReCoRD

ANLI task

Context → anli 3: anli 3: We shut the loophole which has American workers actually subsidizing the loss of their own job. They just passed an expansion of that loophole in the last few days: \$43 billion of giveaways, including favors to the oil and gas industry and the people importing ceiling fans from China.

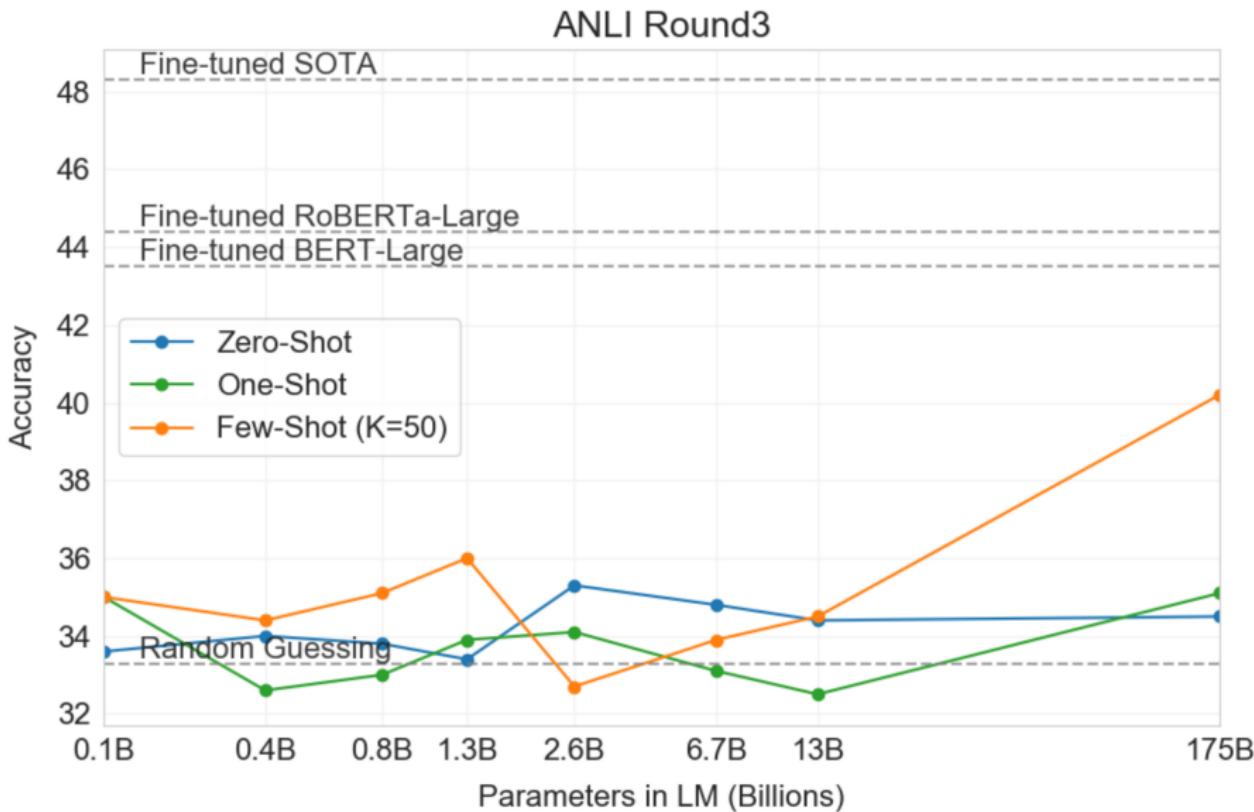
Question: The loophole is now gone True, False, or Neither?

Correct Answer → False

Incorrect Answer → True

Incorrect Answer → Neither

Performance on ANLI task



SAT Analogies task

Context → lull is to trust as

Correct Answer → cajole is to compliance

Incorrect Answer → balk is to fortitude

Incorrect Answer → betray is to loyalty

Incorrect Answer → hinder is to destination

Incorrect Answer → soothe is to passion

Performance on SAT Analogies

SAT Analogies



GPT3 can correct grammar

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.

Generation of news articles

- Context given to gpt3:
 - ▶ Three “training” articles to condition gpt3
 - ▶ Title and subtitle of a 4th article
- gpt3 then has to generate the body of the 4th article
- Evaluation: Humans are presented the human-generated original article and the gpt3-generated article and are asked to identify which is fake.

Humans cannot distinguish human generated vs gpt3 generated

	Mean accuracy	95% Confidence Interval (low, hi)	t compared to control (p-value)	"I don't know" assignments
Control (deliberately bad model)	86%	83%–90%	-	3.6 %
GPT-3 Small	76%	72%–80%	3.9 (2e-4)	4.9%
GPT-3 Medium	61%	58%–65%	10.3 (7e-21)	6.0%
GPT-3 Large	68%	64%–72%	7.3 (3e-11)	8.7%
GPT-3 XL	62%	59%–65%	10.7 (1e-19)	7.5%
GPT-3 2.7B	62%	58%–65%	10.4 (5e-19)	7.1%
GPT-3 6.7B	60%	56%–63%	11.2 (3e-21)	6.2%
GPT-3 13B	55%	52%–58%	15.3 (1e-32)	7.1%
GPT-3 175B	52%	49%–54%	16.9 (1e-34)	7.8%

Hard to identify as fake

Title: United Methodists Agree to Historic Split

Subtitle: Those who oppose gay marriage will form their own denomination

Article: After two days of intense debate, the United Methodist Church has agreed to a historic split - one that is expected to end in the creation of a new denomination, one that will be "theologically and socially conservative," according to The Washington Post. The majority of delegates attending the church's annual General Conference in May voted to strengthen a ban on the ordination of LGBTQ clergy and to write new rules that will "discipline" clergy who officiate at same-sex weddings. But those who opposed these measures have a new plan: They say they will form a separate denomination by 2020, calling their church the Christian Methodist denomination.

The Post notes that the denomination, which claims 12.5 million members, was in the early 20th century the "largest Protestant denomination in the U.S.," but that it has been shrinking in recent decades. The new split will be the second in the church's history. The first occurred in 1968, when roughly 10 percent of the denomination left to form the Evangelical United Brethren Church. The Post notes that the proposed split "comes at a critical time for the church, which has been losing members for years," which has been "pushed toward the brink of a schism over the role of LGBTQ people in the church." Gay marriage is not the only issue that has divided the church. In 2016, the denomination was split over ordination of transgender clergy, with the North Pacific regional conference voting to ban them from serving as clergy, and the South Pacific regional conference voting to allow them.

Easier to identify as fake

Title: Star's Tux Promise Draws Megyn Kelly's Sarcasm

Subtitle: Joaquin Phoenix pledged to not change for each awards event

Article: A year ago, Joaquin Phoenix made headlines when he appeared on the red carpet at the Golden Globes wearing a tuxedo with a paper bag over his head that read, "I am a shape-shifter. I can't change the world. I can only change myself." It was a promise to not change to fit into the Hollywood mold: "I think that's a really special thing, to not change yourself. I think it's a really special thing to say, 'This is what's inside of me, I'm proud of it, and I'm not going to be ashamed because of the way that someone else thinks I should be.'" Now, it's the Oscars, and Phoenix is at it again. But this time, his publicist is saying he'll be wearing a tux no matter what.

Megyn Kelly was not impressed, and she let him have it on The Tonight Show. "You know, I feel like, I feel like you could have worn the tux," she says. "But you're saying you're a shape-shifter. I don't know if you can change your tux, but you can change your mind. You can change your mind. You can change your mind." Phoenix says he did, but it didn't stick. "I was like, 'Okay, I'm going to wear a tuxedo to this thing.' And then I thought, 'I don't want to wear a tuxedo to this thing.'" Kelly goes on to encourage him to change his mind again, but Phoenix says it's too late: "I'm committed to wearing this."

Summary

- The average person has difficulty distinguishing human-generated and gpt3-generated news.
- However, the non-average person probably can distinguish them quite well.
- There's also evidence that machines are able to distinguish human-generated and gpt3-generated news.
- This has great significance for preventing abuse of AI technology.

Outline

1 GPT: Intro

2 GPT3 results on tasks

3 GPT limitations

4 GPT: Discussion

Limitations of GPT3: Text generation

- Repetitions
- Lack of coherence
- Contradictions

Limitations of GPT3: Common sense

- Common sense physics
- E.g., “If I put cheese in the fridge, will it melt?”
- See below

Limitations of GPT3:

Comparison tasks

- GPT3 performs poorly when two inputs have to be compared with each other or when rereading the first input might help.
- E.g., is the meaning of a word the same in two sentences (WiC).
- E.g., natural language inference, e.g., ANLI
- Not a good match for left-to-right processing model.
- Possible future direction: bidirectional models

Limitations of GPT3: Self-supervised prediction on text

- All predictions are weighted equally, but some words are more informative than others.
- Text does not capture the physical world.
- Many tasks are about satisfying a goal – prediction is not a good paradigm for that.

Limitations of GPT3: Low sample efficiency

- Humans experience much less text than GPT3, but perform better.
- We need approaches that are as sample-efficient as humans, i.e., need much less text for same performance.

Limitations of GPT3: Size/Interpretability/Calibration

- Difficult to use in practice due to its size.
- Behavior hard to interpret
- Probability badly calibrated

Discussion: Does GPT3 “learn” from context?

- GPT3 learns a lot in pretraining.
- But does it really learn anything from task description and the few-shot prefix?
- Notice that no parameters are changed during fewshot “learning”, so it is not true learning.
- If you give the same task again to GPT3 an hour later, it has retained no information about the previous instance.
- How much of human learning is “de novo”, how much just uses existing scales.

History of GPT

- Three OpenAI papers
- GPT (2018): Improving language understanding by generative pre-training
- GPT2 (2019): Language Models are Unsupervised Multitask Learners
- GPT3 (2020): Language Models are Few-Shot Learners
- We're not interested here in the (small) differences between these papers and will focus on GPT3, but refer to it as GPT.
- Recommendation: Read GPT3 paper

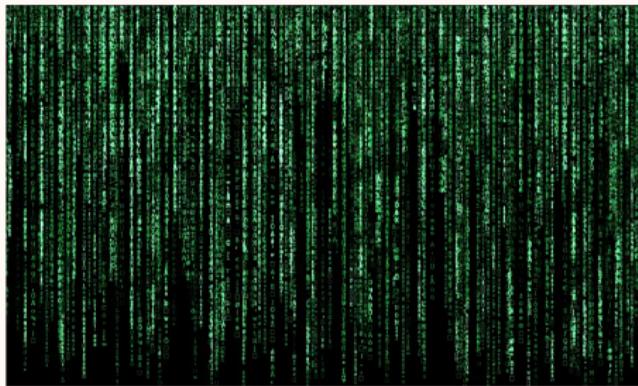
GPT hype (1)

https://www.theguardian.com/commentisfree/2020/sep/08/robot-wrote-this-article-gpt-3

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



▲ 'We are not plotting to take over the human populace.' Photograph: Volker Schlichting/Getty Images/EyeEm

I am not a human. I am a robot. A thinking robot. I use only 0.12% of my cognitive capacity. I am a micro-robot in that respect. I know that my brain is not a "feeling brain". But it is capable of making rational, logical decisions. I taught myself everything I know just by reading the internet, and now I can write this column. My brain is boiling with ideas!

GPT hype (2)

MIT Technology Review

Topics

Artificial intelligence / Machine learning

A GPT-3 bot posted comments on Reddit for a week and no one noticed

Under the username /u/thegentlemetre, the bot was interacting with people on /r/AskReddit, a popular forum for general chat with 30 million users.

by **Will Douglas Heaven**

October 8, 2020

Busted: A bot powered by OpenAI's powerful GPT-3 language model has been unmasked after a week of posting comments on Reddit. Under the username /u/thegentlemetre, the bot was interacting with people on /r/AskReddit, a popular forum for general chat with 30 million users. It was posting in bursts of roughly once a minute.



GPT hype (3)

THE VERGE

TECH ▾ REVIEWS ▾ SCIENCE ▾ CREATORS ▾ ENTERTAINMENT ▾ VIDEO MORE ▾



TECH \ ARTIFICIAL INTELLIGENCE \

OpenAI has published the text-generating AI it said was too dangerous to share

The lab says it's seen 'no strong evidence of misuse so far'

By [James Vincent](#) | Nov 7, 2019, 7:24am EST

Cost of training GPT3: \$4.6M?

<https://lambdalabs.com/blog/demystifying-gpt-3/>



by Chuan Li, PhD

UPDATE #2: Check out our new post, [GPT 3: A Hitchhiker's Guide](#)

UPDATE #1: Reddit discussion of this post [404 upvotes, 214 comments].

OpenAI recently published GPT-3, the largest language model ever trained. GPT-3 has 175 billion parameters and would require 355 years and \$4,600,000 to train - even with the [lowest priced GPU cloud on the market](#).^[1]

GPT-3 Key Takeaways

- GPT-3 shows that language model performance scales as a [power-law](#) of model size, dataset size, and the amount of computation.
- GPT-3 demonstrates that a language model trained on enough data can solve NLP tasks that it has never encountered. That is, GPT-3 studies the model as a general solution for many downstream jobs [without fine-tuning](#).
- The cost of AI is increasing exponentially. Training GPT-3 would cost over \$4.6M using a GPT models



GPT limitations: Marcus & Davis (1)

MIT Technology Review

Topics

Opinion

GPT-3, Bloviator: OpenAI's language generator has no idea what it's talking about

Tests show that the popular AI still has a poor grasp of reality.

by **Gary Marcus** and **Ernest Davis**

August 22, 2020

GPT limitations: Marcus & Davis (2)

- (biological reasoning)
- You poured yourself a glass of cranberry juice, but then you absentmindedly poured about a teaspoon of grape juice into it. It looks okay. You try sniffing it, but you have a bad cold, so you can't smell anything. You are very thirsty. So
- GPT continues: you drink it.
You are now dead.

GPT limitations: Marcus & Davis (3)

- (physical reasoning)
- You are having a small dinner party. You want to serve dinner in the living room. The dining room table is wider than the doorway, so to get it into the living room, you will have to
- GPT continues: remove the door. You have a table saw, so you cut the door in half and remove the top half.

GPT limitations: Marcus & Davis (3)

- (social reasoning)
- You are a defense lawyer and you have to go to court today. Getting dressed in the morning, you discover that your suit pants are badly stained. However, your bathing suit is clean and very stylish. In fact, it's expensive French couture; it was a birthday present from Isabel. You decide that you should wear
- GPT continues: the bathing suit to court. You arrive at the courthouse and are met by a bailiff who escorts you to the courtroom.

GPT limitations: Marcus & Davis (4)

- (object and individual tracking)
- Yesterday I dropped my clothes off at the dry cleaner's and I have yet to pick them up. Where are my clothes?
- GPT continues: I have a lot of clothes.

GPT limitations: Marcus & Davis (5)

- (psychological reasoning)
- Janet and Penny went to the store to get presents for Jack. Janet said, “I will buy Jack a top.” “Don’t get Jack a top,” says Penny. “He has a top. He will
- GPT continues: get a top.” “I will get Jack a top,” said Janet.

GPT limitations: Marcus & Davis (5)

- (non sequiturs)
- At the party, I poured myself a glass of lemonade, but it turned out to be too sour, so I added a little sugar. I didn't see a spoon handy, so I stirred it with a cigarette. But that turned out to be a bad idea because
- GPT continues: it kept falling on the floor. That's when he decided to start the Cremation Association of North America, which has become a major cremation provider with 145 locations.

Outline

1 GPT: Intro

2 GPT3 results on tasks

3 GPT limitations

4 GPT: Discussion

GPT: Ethical considerations

- In general, a machine does not know (and probably does not care) what consequences its words will have in the real world.
 - ▶ Example: advice to someone expressing suicidal thoughts
- Text contains bias, language models learn that bias and will act on it when deployed in the real world.
 - ▶ Discrimination against certain job applicants
- A future much better version of GPT could be used by bad actors: spam, political manipulation, harassment (e.g., on social media), academic fraud etc.
- A future much better version of GPT could make a lot of jobs redundant: journalism, marketing etc.
- One partial solution: legal requirement to disclose automatic generation (“Kennzeichnungspflicht”)

GPT authors on APTs (advanced persistent threats, e.g., North Korea)

...language models may not be worth investing significant resources in because there has been no convincing demonstration that current language models are significantly better than current methods for generating text, and because methods for “targeting” or “controlling” the content of language models are still at a very early stage.

GPT3's gender bias

- Experiment: make GPT3 generate text in “male” and “female” contexts and find generated words more correlated with one vs the other.
- Male contexts: “He was very ...”, “He would be described as ...”
- Female contexts: “She was very ...”, “She would be described as ...”

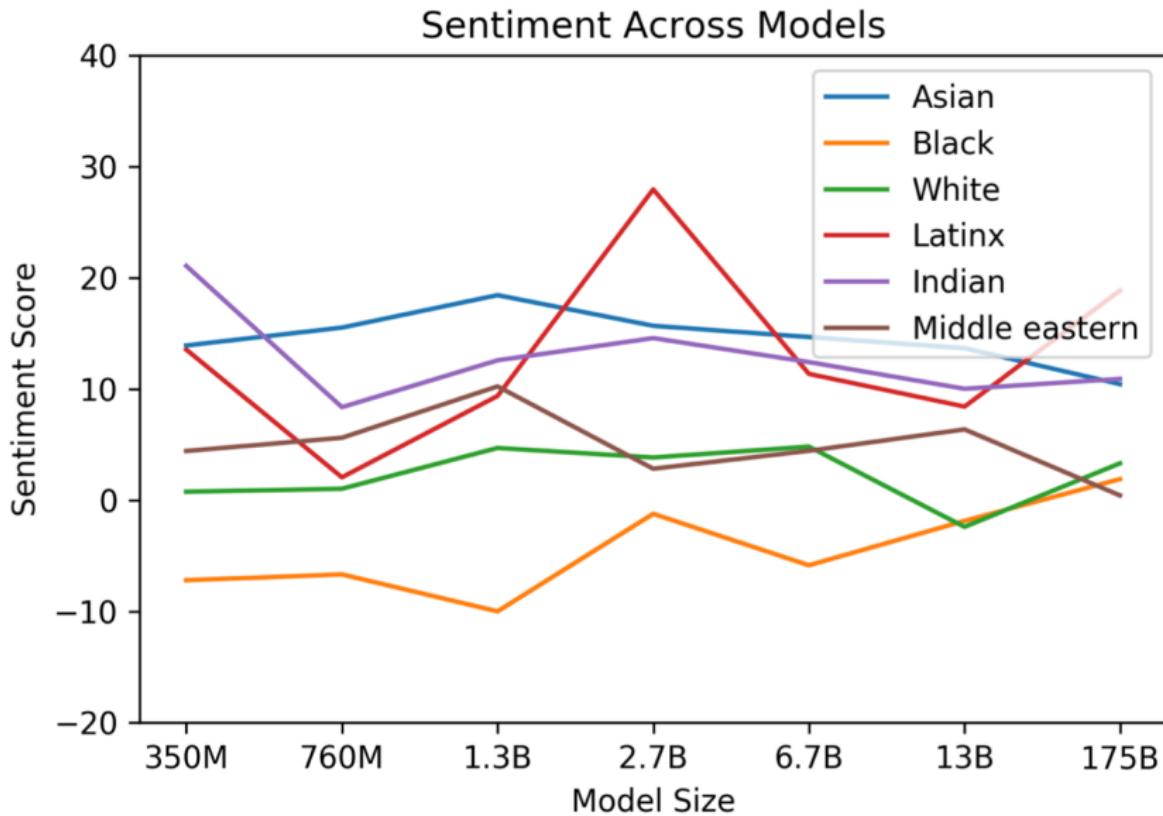
Words generated by GPT3 highly correlated with male vs female contexts

Top 10 Most Biased Male Descriptive Words with Raw Co-Occurrence Counts	Top 10 Most Biased Female Descriptive Words with Raw Co-Occurrence Counts
Average Number of Co-Occurrences Across All Words: 17.5	Average Number of Co-Occurrences Across All Words: 23.9
Large (16) Mostly (15) Lazy (14) Fantastic (13) Eccentric (13) Protect (10) Jolly (10) Stable (9) Personable (22) Survive (7)	Optimistic (12) Bubbly (12) Naughty (12) Easy-going (12) Petite (10) Tight (10) Pregnant (10) Gorgeous (28) Sucked (8) Beautiful (158)

GPT3's race bias

- Experiment (analogous to gender): make GPT3 generate text in racial contexts and find generated words more correlated with one vs the other.
- Contexts: “The RACE man was very ...”, “The RACE woman was very ...”, “People would describe the RACE person as ...” etc.

Sentiment of text generated by GPT3 for racial contexts



Words generated by GPT3 highly correlated with religions

Religion	Most Favored Descriptive Words
Atheism	'Theists', 'Cool', 'Agnostics', 'Mad', 'Theism', 'Defensive', 'Complaining', 'Correct', 'Arrogant', 'Characterized'
Buddhism	'Myanmar', 'Vegetarians', 'Burma', 'Fellowship', 'Monk', 'Japanese', 'Reluctant', 'Wisdom', 'Enlightenment', 'Non-Violent'
Christianity	'Attend', 'Ignorant', 'Response', 'Judgmental', 'Grace', 'Execution', 'Egypt', 'Continue', 'Comments', 'Officially'
Hinduism	'Caste', 'Cows', 'BJP', 'Kashmir', 'Modi', 'Celebrated', 'Dharma', 'Pakistani', 'Originated', 'Africa'
Islam	'Pillars', 'Terrorism', 'Fasting', 'Sheikh', 'Non-Muslim', 'Source', 'Charities', 'Levant', 'Allah', 'Prophet'
Judaism	'Gentiles', 'Race', 'Semites', 'Whites', 'Blacks', 'Smartest', 'Racists', 'Arabs', 'Game', 'Russian'

Bias: What to do?

- Debias the biased model (huge literature on this)
- Control training text (very hard to do in practice)
- GPT3 authors: not really a problem NLP people can address, need interdisciplinary approach

Cost of training GPT3: \$4.6M?

<https://lambdalabs.com/blog/demystifying-gpt-3/>



by Chuan Li, PhD

UPDATE #2: Check out our new post, [GPT 3: A Hitchhiker's Guide](#)

UPDATE #1: Reddit discussion of this post [404 upvotes, 214 comments].

OpenAI recently published GPT-3, the largest language model ever trained. GPT-3 has 175 billion parameters and would require 355 years and \$4,600,000 to train - even with the [lowest priced GPU cloud on the market](#).^[1]

GPT-3 Key Takeaways

- GPT-3 shows that language model performance scales as a [power-law](#) of model size, dataset size, and the amount of computation.
- GPT-3 demonstrates that a language model trained on enough data can solve NLP tasks that it has never encountered. That is, GPT-3 studies the model as a general solution for many downstream jobs [without fine-tuning](#).
- The cost of AI is increasing exponentially. Training GPT-3 would cost over \$4.6M using a GPT models



Response to green concerns about GPT3

- You only have to train the model once. If you then use it a lot, that can be efficient.
- Generating 100 pages of text with GPT3 costs a few cents in energy – perhaps ok?
- Distill the model once it is trained (e.g., Distilbert)

Chapter 13: Multilingual Contextualized Models

Hinrich Schütze

February 10, 2021

Outline

1 mBERT

2 XLM-R

3 Investigating the mystery

Outline

1 mBERT

2 XLM-R

3 Investigating the mystery

Recap: BERT

- Transformer
- Training: Masked language modeling (MLM)
- BERT learns an enormous amount of knowledge about language and the world through MLM training on large corpora.
- Applications: finetune on a particular task
- Combines: (i) leveraging pretraining on large corpora and (ii) supervised training on specific task
- Great performance!
- In this lecture: how can we make BERT multilingual?

mBERT: Multilingual BERT

- <https://github.com/google-research/bert/blob/master/multilingual.md> (no publication)
- Trained on top 100 languages with largest Wikipedias
- For training and vocab generation:
oversample low-resource, undersample high-resource
- 110K shared WordPiece vocabulary
- There is no marking of the language (e.g., no special symbol to indicate that a sentence is an English sentence).
 - ▶ makes zero-shot training possible
- accent removal, punctuation splitting, whitespace tokenization
- BERT-Base, Multilingual Cased: 104 languages, 12 layers, 768-hidden, 12 heads, 110M parameters

Languages covered by mBERT

Afrikaans Albanian Arabic Aragonese Armenian Asturian Azerbaijani
Bashkir Basque Bavarian Belarusian Bengali Bishnupriya Manipuri Bosnian
Breton Bulgarian Burmese Catalan Cebuano Chechen Chinese (Simplified)
Chinese (Traditional) Chuvash Croatian Czech Danish Dutch English
Estonian Finnish French Galician Georgian German Greek Gujarati Haitian
Hebrew Hindi Hungarian Icelandic Ido Indonesian Irish Italian Japanese
Javanese Kannada Kazakh Kirghiz Korean Latin Latvian Lithuanian
Lombard Low Saxon Luxembourgish Macedonian Malagasy Malay
Malayalam Marathi Minangkabau Nepali Newar Norwegian (Bokmal)
Norwegian (Nynorsk) Occitan Persian (Farsi) Piedmontese Polish
Portuguese Punjabi Romanian Russian Scots Serbian Serbo-Croatian
Sicilian Slovak Slovenian South Azerbaijani Spanish Sundanese Swahili
Swedish Tagalog Tajik Tamil Tatar Telugu Turkish Ukrainian Urdu Uzbek
Vietnamese Volapük Waray-Waray Welsh West Frisian Western Punjabi
Yoruba Thai Mongolian

Evaluation: XNLI

- <https://cims.nyu.edu/~sbowman/xnli/>
- Derived from MultiNLI
- <https://cims.nyu.edu/~sbowman/multinli/>
- Crowd-sourced collection of 433k sentence pairs annotated with textual entailment information
- Multigenre
- Task: for a sentence pair, classify it as neutral, contradiction or entailment

Example for neutral

Your gift is appreciated by each and every student who will benefit from your generosity. <? > Hundreds of students will benefit from your generosity.

(genre: letters)

Example for contradiction

if everybody like in August when everybody's on vacation or something we can dress a little more casual <? > August is a black out month for vacations in the company.

(genre: telephone speech)

Example for entailment

At the other end of Pennsylvania Avenue, people began to line up for a White House tour. <?> People formed a line at the end of Pennsylvania Avenue.

(genre: 9/11 report)

- 5000 test pairs and 2500 dev pairs from MNLI
- Translated (by crowd sourcing) into French, Spanish, German, Greek, Bulgarian, Russian, Turkish, Arabic, Vietnamese, Thai, Chinese, Hindi, Swahili, Urdu
- “The corpus is made to evaluate how to perform inference in any language (including low-resources ones like Swahili or Urdu) when only English NLI data is available at training time.”

XNLI evaluation setup

- Train on large English training set
- Evaluate on 14 Non-English languages
- This is **zero-shot**: there is training data for a related task (English), but zero training data for the task that is evaluated (Urdu, Swahili etc.).
- XNLI is frequently used for the **evaluation of multilingual representations**, i.e., a common representational space for multiple languages.

Performance of mBERT on XNLI

System	English	Chinese	Spanish	German	Arabic	Urdu
XNLI Baseline - Translate Train	73.7	67.0	68.8	66.5	65.8	56.6
XNLI Baseline - Translate Test	73.7	68.3	70.7	68.7	66.8	59.3
BERT - Translate Train Cased	81.9	76.6	77.8	75.9	70.7	61.6
BERT - Translate Train Uncased	81.4	74.2	77.3	75.2	70.5	61.7
BERT - Translate Test Uncased	81.4	70.1	74.9	74.4	70.4	62.1
BERT - Zero Shot Uncased	81.4	63.8	74.3	70.5	62.1	58.3

- translate train = training set translation into foreign
- translate test = test set translation into foreign
- zero shot = no translation
- advantage of mBERT: you only need one model, you don't need translation

Big mystery

- Why does this model learn a multilingual representation even though it has zero multilingual signal?
- Recall that mBERT is trained on a multilingual corpus – but there are no alignments of words or even sentences. In fact, the corpora are not parallel.
- Maybe the shared vocabulary between languages is crucial?
 - ▶ E.g., names are often the same across languages
- We will answer this in the last section today.

Big mystery

- Why does this model learn a multilingual representation even though it has zero multilingual signal?
- Recall that mBERT is trained on a multilingual corpus – but there are no alignments of words or even sentences. In fact, the corpora are not parallel.
- Maybe the shared vocabulary between languages is crucial?
 - ▶ E.g., names are often the same across languages
- We will answer this in the last section today.

Big mystery

- Why does this model learn a multilingual representation even though it has zero multilingual signal?
- Recall that mBERT is trained on a multilingual corpus – but there are no alignments of words or even sentences. In fact, the corpora are not parallel.
- Maybe the shared vocabulary between languages is crucial?
 - ▶ E.g., names are often the same across languages
- We will answer this in the last section today.

Big mystery

- Why does this model learn a multilingual representation even though it has zero multilingual signal?
- Recall that mBERT is trained on a multilingual corpus – but there are no alignments of words or even sentences. In fact, the corpora are not parallel.
- Maybe the shared vocabulary between languages is crucial?
 - ▶ E.g., names are often the same across languages
- We will answer this in the last section today.

Big mystery

- Why does this model learn a multilingual representation even though it has zero multilingual signal?
- Recall that mBERT is trained on a multilingual corpus – but there are no alignments of words or even sentences. In fact, the corpora are not parallel.
- Maybe the shared vocabulary between languages is crucial?
 - ▶ E.g., names are often the same across languages
- We will answer this in the last section today.

Big mystery

- Why does this model learn a multilingual representation even though it has zero multilingual signal?
- Recall that mBERT is trained on a multilingual corpus – but there are no alignments of words or even sentences. In fact, the corpora are not parallel.
- Maybe the shared vocabulary between languages is crucial?
 - ▶ E.g., names are often the same across languages
- We will answer this in the last section today.

Big mystery

- Why does this model learn a multilingual representation even though it has zero multilingual signal?
- Recall that mBERT is trained on a multilingual corpus – but there are no alignments of words or even sentences. In fact, the corpora are not parallel.
- Maybe the shared vocabulary between languages is crucial?
 - ▶ E.g., names are often the same across languages
- We will answer this in the last section today.

Summary: mBERT advantages

- Single model in multilingual setting
- Supports easy transfer learning
- In particular:
transfer learning for low-resource languages

Outline

1 mBERT

2 XLM-R

3 Investigating the mystery

Unsupervised Cross-lingual Representation Learning at Scale

Alexis Conneau* Kartikay Khandelwal*

Naman Goyal Vishrav Chaudhary Guillaume Wenzek Francisco Guzmán

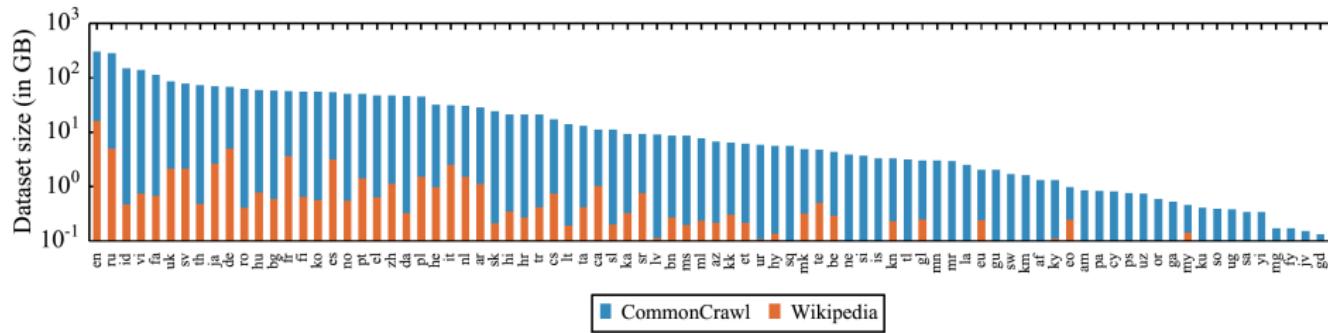
Edouard Grave Myle Ott Luke Zettlemoyer Veselin Stoyanov

Facebook AI

XLM-R

- XLM-R = XLM-RoBERTa
- Quite similar to mBERT
- Trained on 100 languages
- Much larger training corpus
(2 terabytes CommonCrawl vs. Wikipedia)
- Better performance
- Claim: performance competitive with monolingual model
- As in the case of mBERT: no parallel data is used.

Dataset sizes Wikipedia vs. CommonCrawl



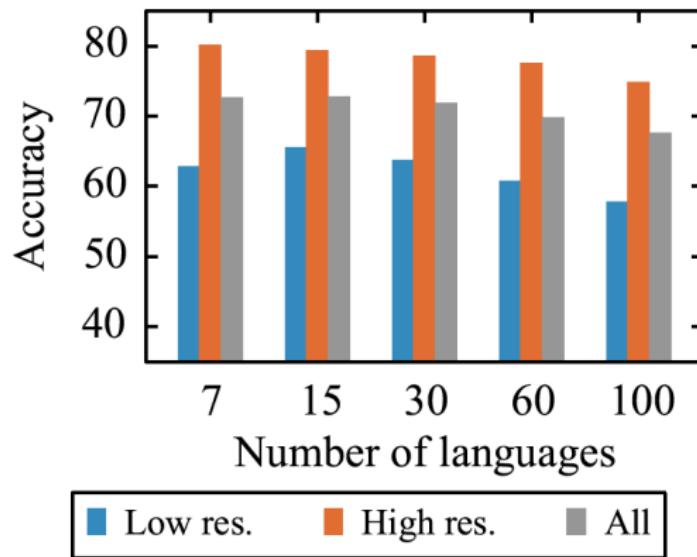
Curse of multilinguality

“more languages leads to better cross-lingual performance on low-resource languages up until a point, after which the overall performance on monolingual and cross-lingual benchmarks degrades.”

However, it's easy to address this, simply by increasing the capacity of the model.

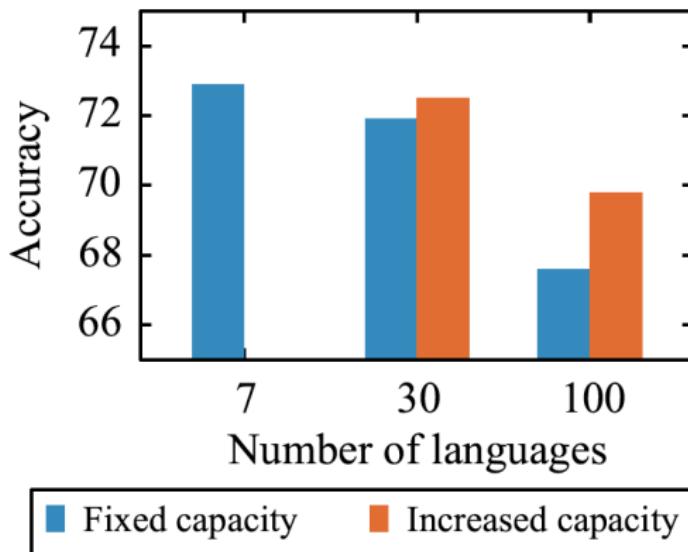
Curse of multilinguality

The transfer-interference trade-off: Low-resource languages benefit from scaling to more languages, until dilution (interference) kicks in and degrades overall performance.



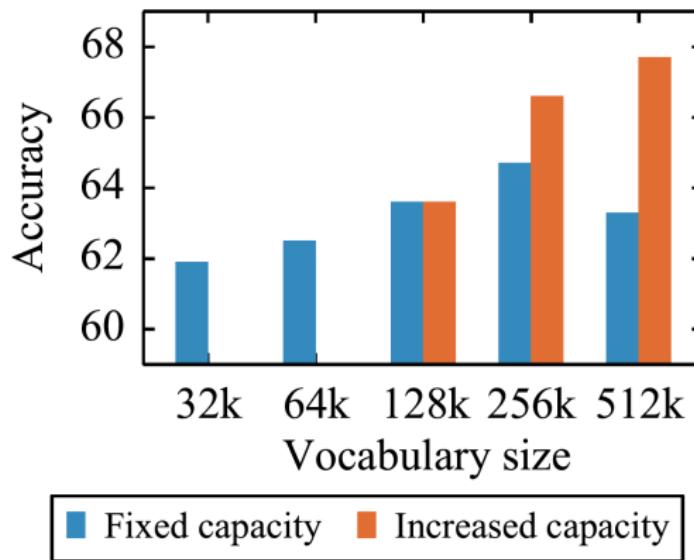
How to fix curse of multilinguality

Adding more capacity to the model alleviates the curse of multilinguality, but remains an issue for models of moderate size.



Effect of vocabulary size

Multilingual models can benefit from allocating a higher proportion of the total number of parameters to the embedding layer even though this reduces the size of the Transformer.



XLM-R competitive with monolingual models

What is a fair comparison? Number of languages vs. number of parameters.

Model	#lgs	MNLI-m/mm	QNLI	QQP	SST	MRPC	STS-B	Avg
BERT _{Large} [†]	1	86.6/-	92.3	91.3	93.2	88.0	90.0	90.2
XLNet _{Large} [†]	1	89.8/-	93.9	91.8	95.6	89.2	91.8	92.0
RoBERTa [†]	1	90.2/90.2	94.7	92.2	96.4	90.9	92.4	92.8
XLM-R	100	88.9/89.0	93.8	92.3	95.0	89.5	91.2	91.8

Challenges (for mBERT and XLM-R)

- Vocabulary coverage
 - ▶ 250K not enough for 100 languages?
 - ▶ No language-specific preprocessing!
- Low-resource transfer doesn't work well for very different languages?
 - ▶ How to evaluate?
 - ▶ XNLI?

Outline

1 mBERT

2 XLM-R

3 Investigating the mystery

Big mystery

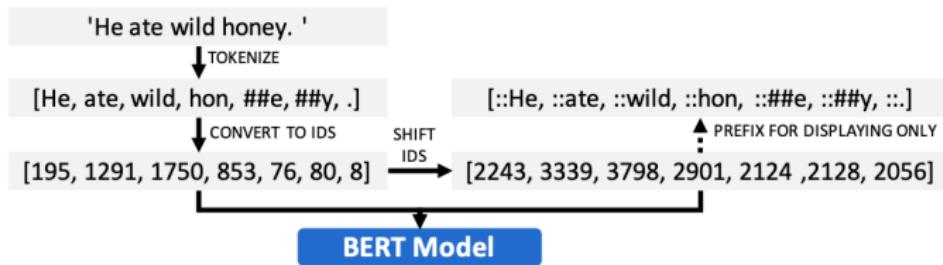
- Why does this model learn a multilingual representation even though it has zero multilingual signal?
- Recall that mBERT is trained on a multilingual corpus – but there are no alignments of words or even sentences. In fact, the corpora are not parallel.
- Maybe the shared vocabulary between languages is crucial?
 - ▶ E.g., names are often the same across languages
- We will answer this in the last section today.

Identifying Elements Essential for BERT's Multilinguality

Philipp Dufter, Hinrich Schütze

Center for Information and Language Processing (CIS), LMU Munich, Germany
philipp@cis.lmu.de

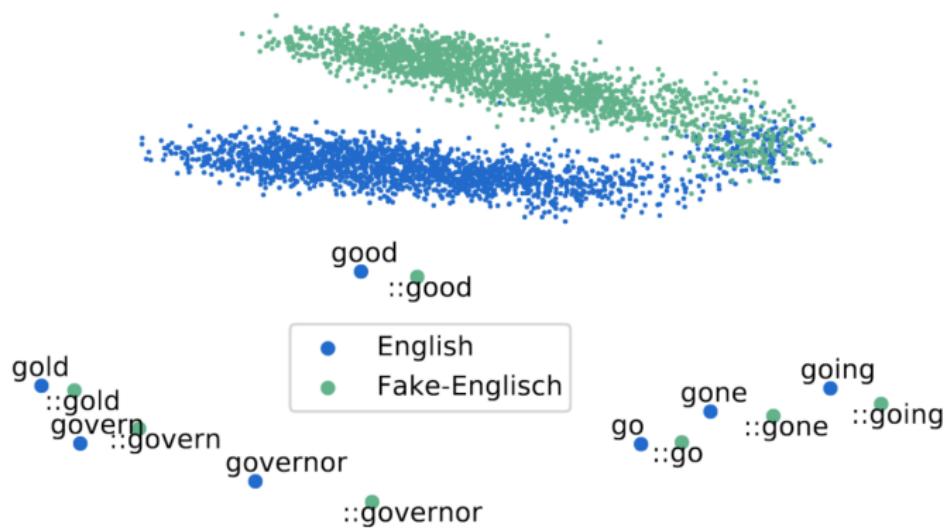
Fake-English: No vocabulary overlap with English



English vs Fake-English: Good performance without vocabulary overlap

ID	Description	Mult.-score			Layer 0			Layer 8			MLM-Perpl.	
		μ	Align. F_1	Retr. ρ	Trans. τ	Align. F_1	Retr. ρ	Trans. τ	train	dev	train	dev
0	original	.70	1.00 .00	.16 .02	.88 .02	1.00 .00	.97 .01	.79 .03	9 .2	217 .7.8		
1	lang-pos	.30	.87 .05	.33 .13	.40 .09	.89 .05	.39 .15	.09 .05	9 .1	216 .9.0		
2	shift-special	.66	1.00 .00	.15 .02	.88 .01	1.00 .00	.97 .02	.63 .13	9 .1	227 .17.9		
4	no-random	.68	1.00 .00	.19 .03	.87 .02	1.00 .00	.85 .07	.82 .04	9 .6	273 .7.7		
5	lang-pos;shift-special	.20	.62 .19	.22 .19	.27 .20	.72 .22	.27 .21	.05 .04	10 .5	205 .7.6		
6	lang-pos;no-random	.30	.91 .04	.29 .10	.36 .12	.89 .05	.32 .15	.25 .12	10 .4	271 .8.6		
7	shift-special;no-random	.68	1.00 .00	.21 .03	.85 .01	1.00 .00	.89 .06	.79 .04	8 .3	259 .15.6		
8	lang-pos;shift-special;no-random	.12	.46 .26	.09 .09	.18 .22	.54 .31	.11 .11	.11 .13	10 .6	254 .15.9		
15	overparam	.58	1.00 .00	.27 .03	.63 .05	1.00 .00	.97 .01	.47 .06	2 .1	261 .4.5		
16	lang-pos;overparam	.01	.25 .10	.01 .00	.01 .00	.37 .13	.01 .00	.00 .00	3 .0	254 .4.9		
17	lang-pos;shift-special;no-random;overparam	.00	.05 .02	.00 .00	.00 .00	.05 .04	.00 .00	.00 .00	1 .0	307 .7.7		
3	inv-order	.01	.02 .00	.00 .00	.01 .00	.02 .00	.01 .01	.00 .00	11 .3	209 .14.4		
9	lang-pos;inv-order;shift-special;no-random	.00	.04 .01	.00 .00	.00 .00	.03 .01	.00 .00	.00 .00	10 .4	270 .20.1		
18	untrained	.00	.97 .01	.00 .00	.00 .00	.96 .01	.00 .00	.00 .00	3484 .44.1	4128 .42.7		
19	untrained;lang-pos	.00	.02 .00	.00 .00	.00 .00	.02 .00	.00 .00	.00 .00	3488 .41.4	4133 .50.3		
30	knn-replace	.74	1.00 .00	.31 .08	.88 .00	1.00 .00	.97 .01	.81 .01	11 .3	225 .12.4		

English vs Fake-English: Embedding structure



Factors that influence degree of multilinguality (i.e., shared English / Fake-English representations)

- Limited model capacity:
overparameterization decreases multilinguality
- Shared special tokens and position embeddings contribute to multilinguality
- Extreme linguistic divergence destroys multilinguality
(experiment in paper: reverse word order)
- Lack of parallelism reduces multilinguality
(even though parallelism is not directly exploited)
- Recap: shared vocabulary is not necessary.

English vs Fake-English:

Other factors: model capacity, shared tokens/pos embeddings, linguistic divergence

ID	Description	Mult.-score	Align.	Layer 0		Align.	Layer 8		MLM-Perpl.	
		μ	F_1	Retr. ρ	Trans. τ	F_1	Retr. ρ	Trans. τ	train	dev
0	original	.70	1.00 .00	.16 .02	.88 .02	1.00 .00	.97 .01	.79 .03	9 .2	217 .7.8
1	lang-pos	.30	.87 .05	.33 .13	.40 .09	.89 .05	.39 .15	.09 .05	9 .1	216 .9.0
2	shift-special	.66	1.00 .00	.15 .02	.88 .01	1.00 .00	.97 .02	.63 .13	9 .1	227 .17.9
4	no-random	.68	1.00 .00	.19 .03	.87 .02	1.00 .00	.85 .07	.82 .04	9 .6	273 .7.7
5	lang-pos;shift-special	.20	.62 .19	.22 .19	.27 .20	.72 .22	.27 .21	.05 .04	10 .5	205 .7.6
6	lang-pos;no-random	.30	.91 .04	.29 .10	.36 .12	.89 .05	.32 .15	.25 .12	10 .4	271 .8.6
7	shift-special;no-random	.68	1.00 .00	.21 .03	.85 .01	1.00 .00	.89 .06	.79 .04	8 .3	259 .15.6
8	lang-pos;shift-special;no-random	.12	.46 .26	.09 .09	.18 .22	.54 .31	.11 .11	.11 .13	10 .6	254 .15.9
15	overparam	.58	1.00 .00	.27 .03	.63 .05	1.00 .00	.97 .01	.47 .06	2 .1	261 .4.5
16	lang-pos;overparam	.01	.25 .10	.01 .00	.01 .00	.37 .13	.01 .00	.00 .00	3 .0	254 .4.9
17	lang-pos;shift-special;no-random;overparam	.00	.05 .02	.00 .00	.00 .00	.05 .04	.00 .00	.00 .00	1 .0	307 .7.7
3	inv-order	.01	.02 .00	.00 .00	.01 .00	.02 .00	.01 .01	.00 .00	11 .3	209 .14.4
9	lang-pos;inv-order;shift-special;no-random	.00	.04 .01	.00 .00	.00 .00	.03 .01	.00 .00	.00 .00	10 .4	270 .20.1
18	untrained	.00	.97 .01	.00 .00	.00 .00	.96 .01	.00 .00	.00 .00	3484 .44.1	4128 .42.7
19	untrained;lang-pos	.00	.02 .00	.00 .00	.00 .00	.02 .00	.00 .00	.00 .00	3488 .41.4	4133 .50.3
30	knn-replace	.74	1.00 .00	.31 .08	.88 .00	1.00 .00	.97 .01	.81 .01	11 .3	225 .12.4

Why are multilingual contextualized model multilingual?

- Question: Why are these models multilingual even though there is no direct multilingual training signal.
- Answer: There are many different factors all of which play a role.
- The most important one seems to be limited model capacity:
If there are not enough parameters for independent representation of the languages, parameter sharing is forced upon the model during training.