Deep Learning for NLP Summary

Lecture 1 Chapter 1: Introduction to Machine Learning – Overview

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.

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 Rm (sometimes also by matrices, tensors).

Typical rectangular data sets are defined as follows. Dataset: collection of examples Data is stored in a design matrix: m: number of examples n: number of features Examples: Xi,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 X , or find interesting properties of X .
- Example: Clustering (find groups of similar images/documents)
- Training data: only X . Supervised learning:
- Predict specific additional properties from 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 y in Rm together with the input X can be used for training the model.
- The task is then to predict the labels of new data, where only the features 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(x), i.e. estimate the distribution Supervised learning: learn mapping from x to y, typically by estimating the conditional distribution p(y|x). The supervised learning task is often formulated as the problem of estimating (or approximating) a function f , such that where epsilon is a random component (error term) describing the probabilistic nature of the task. Supervised learning in an unsupervised way (Bayes theorem): Note: if y is continuous, the sum (in the denominator) must 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 :Rn $\rightarrow\{1 \ldots k\}$ Typical example: Categorize image patches

- Feature vector: color intensities for each pixel; derived features.
- Output categories: Predefined set of labels

Task: Regression

- Predict a numerical value given some input.
- stock price

Task: Structured Prediction

Predict a multi-valued output with special inter-dependencies and constraints. Typical examples:

- Part-of-speech tagging
- Syntactic parsing
- Machine Translation

Often involves search and problem-specific algorithms.

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.
- EMNLP 2017: The computer negotiates with humans in natural language in order to maximize its points in a game

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

ACL 2017: Schizophrenia patients can be detected by their non-standard use of metaphors, and more extreme sentiment expressions.

Algorithms in supervised ML

Supervised linear regression logistic regression decision trees (e.g. C 4.5) CART random forest Boosting, e.g. (artificial) neural networks

Some Algorithms for unsupervised ML

Unsupervised Clustering (k-means) Clustering (partition around medoids (PAM)) PCA (principal components analysis) k-nearest neighbor Isolation forest4 Autoencoders

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 - 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):
- 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).
- Accuracy may be biased in data very skewed

Examples for Continuous Loss Functions

- Squared error (regression): (y - f (x))2
- Hinge loss (classification):
  - max(0,1-f(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. Input layer, hidden layer and output layer.

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
- Algorithmes themselves creates features on every layer
- Feature learning as apposed to feature engineering
- Still builds on general ML components like the loss function

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. L is called loss function or cost function (we will discuss it further in the next chapter). Loss function measures approximation/prediction and the label. Fitting data vs. generalizing from data. If too much adaptation, we get overfitting. If too simple, under-fitting.

Lecture 2 Introduction to Machine Learning – ML Basics

Overview

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

Scalars, vectors, matrices and tensors I

Scalar is a real time X. A Vector is an array of real numbers. We define vector as column vectors, each element stack on top of each other. A matrix is a rectangular scheme of real numbers or a 2-dimensional array. Tensor is a multidimensional array.

Transpose

Exchange rows and columns.

Addition

Two matrices A and B can usually only be added if they have the same dimensions. The addition C = A + B is done element by element, i.e. cij = aij + bij. If b is a scalar = C = A + b added to each element of A.

Multiplication

Only multiplied if number of rows A equal number of rows B. The result C = AB. Dot product defined when same length. if b is scalar, C = bA, means that each element of A is multiplied by b. One frequent used is element by element multiplication aka: **Hadamard Product** if have same dimensions. The Kronecker product, each element of b is multiplied by a matrix.

Multiplication rules

C(A+B) = CA+ CB C(BA) = (CB)A (AB)ˆT= BˆT AˆT

- Trace is the sum of diagonal elements.

Special matrices:

Aˆ-1 A = I Inverse only exist if all rows/columns are linear independent. Eigenvalue decomposition.

$A = \Gamma\Lambda\Gamma^T$

Gamma = Orthogonal eigenvalues of A. And Lambda is the diagnomal matrix with eigenvalues on the diagonal.

Norms

L1 - p is equal to 1 L2 - euclidean norm

- Frobenius norm (#TODO)

Derivations

Gradient notation is used for derivative of a scalar wrt a vector w. Chain-rule important for backpropagation.

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.

Probability distributions in NLP

Joint distribution of a sequence of words w1, w2, . . . , wn, n may be any number: The right hand side of the equation is a short form of the left hand side. Example: P(The,dog,is,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:

Factorization of the joint distribution

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

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(wi|wi-1,. . . ,w1) = P(wi|wi-1) (i.e., we only consider bigrams).

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. Uses 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 ~ N(μ,sigma2)) where μ andsigma 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

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:

If f is discrete, the integral is replaced by a sum. The variance of g is then:

#TODO

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: #TODO * For densities: let p(x,y) the joint density of two random variables, p(x) the marginal and p(y|x) the conditional density. 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: #TODO 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 L is a function into the positive real numbers (including 0).

- Let y the actual value and y^ a prediction then the loss function is given by L = L(y,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 $EL = E(x, y)(L(Y, \hat{Y}))$. Usually, the prediction Y is estimated as a prediction function f (x), depending on the features x,

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

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

Loss function II

If one chooses a quadratic loss function L(Y , f (X)) = [Y - f (X)] , the expected loss (or prediction error) is

Factorizing P(Y,X) = P(Y|X)P(X), this can be written as

$E(L(Y, f(x)) = E_x E_{y|x}([Y - f(X)]^2|X)$

A pointwise minimization leads to

$f(x) = argmin E_{y|x}[Y - c]^2|X = x)$

This is called the regression function (conditional expectation)

f(x)= E(Y|X=x)

Loss function III

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

#TODO

It holds

#TODO

With the 0-1-loss one gets

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):

where $Yj = f(x_j)$ is the prediction of the jth 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:

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:

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

Values between 0 and 1.

Softmax function

generalization of the sigmoid function.

Rectified linear unit (ReLu)

Derivative between 0 and 1.

Lecture 3 Linear and Logistic Regression

Introduction to Machine Learning –

Linear Regression

In linear regression, the prediction y^ of a label y is a linear function:

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

- Note, that in this notation, the parameter vector w in Rn stands already for the estimated parameter vector.
- Weight wj decides if value of feature xj increases or decreases prediction y.
- Only linear in the weights.
- It does not have to be linear in the feature but can be transformed in the features.
- Hyperplane points above and below

Matrix Notation I

- m samples

Prediction for one sample

$\hat{y_i} = w^T x_i$

residual error:

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

Squared error:

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

Matrix Notation II

Stack all into stacked vectors. Prediction vector: $\hat{y} = Xw$

Can also look into feature/design matrix X. Sometimes the intercept term is also called bias or shift term. To estimate the parameters W, usually has to be done in the way we have learned in ML. Firstly, we need to use some training data. Then we can make higher level decisions. Finally, to see how good the prediction quality is we need to evaluate on test data.

Simple Example: Housing Price

2 weights that have to be estimated. The first weight is for the intercept column. The second weight is the square feet of the property on the outcome Y. #TODO(matrix stuff)

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.

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

#TODO (matrix notation)

Learning: Improving on MSE

- Gradient of a function f : Vector whose components are the n partial derivatives of f wrt to the parameters, here w.
- View MSE as a function f (w) of w
- Minimum is where gradient is 0:

Minimum is where gradient delta MSE = 0. w

Why minimum and 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. As long as one feature value is greater 0 and we can guarantee that we have found a minimum.

Deriving the Normal Equation

Solving for w: We now know that minimum is where gradient is 0. If we solve this problem is solving the so called normal equations. The w itself is of dimension n * t.

TODO (OLS estimation)

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

Practical Remarks

What if 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
  - a Moore-Penrose pseudo-inverse picks solution with smallest Euclidean norm.
  - adding a regularization term makes the system of equations non-singular (uniquely solvable)
    - ridge regression
    - normal equation becomes $w = (X^T X + \lambda I)^{-1} X^T y$
- In general: Avoid computing the matrix inverse when implementing least squares (slow/unstable)!
- There are usually special routines for solving linear least squares and rigde regression.

Second Derivative Test for Multi-Dimensional Case

- Second derivative test for multi-dimensional w in Rn
- Is Hessian positive-semidefinite? (zT Hz $>$ 0 for z/= 0) equivalent
  - Function is convex.
- Hessian: Matrix of second-order partial derivatives.

Matrix algebra:

- derivative w respect to w leads to show that XˆTX is at least semi-definite and guarantees that minimum squared error.
- #TODO (math)

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 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.
- Probability density of the data set is the product of example probability densities.

Maximum Likelihood Estimation

- Likelihood: Probability density of data viewed as function of parameters theta
- (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 -> take negative of function.

Maximum likelihood estimator for theta:

#TODO

Conditional Log-Likelihood

Log-likelihood can be stated for supervised and unsupervised tasks

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

Linear Regression as Maximum Likelihood

- Taking the log makes it a quadratic function!

Conditional negative log-likelihood:

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

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 y^, treat y as a random variable. However, what about the uncertainty about our parameters theta? Shouldn't theta be treated as a random variable, too? (Instead of a point estimate theta?).

Uncertainty about theta influenced by:

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

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,

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 ∈ {theta, 1} based on features x. Logistic Regression is a Generalized Linear Model: Linear model is related to the response variable via a link function.

Classification: Outcome (per example) 0 or 1 Logistic Regression: Linear function + logistic sigmoid

Binary Logistic Regression Probability of different outcomes (for one example): Probability of positive outcome:

Probability of a Training Example

(i) (i) given features x Probability for actual label y Can be written for both labels (0 and 1) without case distinction Label exponentiation trick: use x0 = 1

Binary Logistic Regression Conditional Negative Log-Likelihood (NLL):

#TODO

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

(i) p(Y=1|x ;theta)>0.5

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

Lecture 4 Optimization, Deep Feed Forward Networks, Backproagation, Regularization

Introduction to Machine Learning

Optimization

If #TODO is convex, it is minimized where #TODO = 0 If #TODO 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

Local Minima

Gradient Descent for Logistic Regression

Gradient Descent: Summary Iterative method for function minimization. Gradient indicates rate of change in objective function, given a local change to feature weights. Substract the gradient: I decrease parameters that (locally) have positive correlation with objective I increase parameters that (locally) have negative correlation with objective Gradient updates only have the desired properties in a small region around previous parameters theta of t. Control locality by the learning rate eta. 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.

#TODO

Other Choices for Hidden Units

A good activation function aids learning, and provides large gradients. Sigmoidal functions (logistic sigmoid) I have only a small region before they flatten out in either direction. I Practice shows that this seems to be ok in conjunction with Log-loss objective. I But they don't work as well as hidden units. I ReLU are better alternative since gradient stays constant. Other hidden unit functions: I

maxout: take maximum of several values in previous layer. I purely linear: can serve as low-rank approximation.

Summary

Gradient descent: Minimize loss by iteratively substracting 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

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

L2 Regularization

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

Lecture 5 Word2Vec

Motivation word embeddings

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

word embeddings are dense l= sparse), trainable word vectors Allows us to calculate similarities between words, e.g., with cosine similarity:

Word embeddings can be trained from scratch on supervised data: Initialize 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 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 (awesome,great) ~1

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.

Word2Vec

Mikolov et al. (2013): Efficient estimation of word representations in vector space.

distributional hypothesis: “a word is characterized by the company it keeps”’

Skigram

The skipgram task is to maximize the likelihood of the context words, given their center word: Expressed as a negative log likelihood loss:

CBOW Task

The continuous bag of words (CBOW) task is to maximize the likelihood of the center words, given their context words: Expressed as a negative log likelihood loss:

- Naive softmax model
- Hierachical softmax model
- Negative sampling model

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.

FastText

Even if we train Word2Vec on a very large corpus, we will still encounter unknown words at test time

Extension of Word2Vec by **Bojanowski et al. (2017):** Enriching Word Vectors with Subword Information.

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

Applications of pre-trained word embeddings

Lecture 6 Convolutational Neural Networks

An architecture is an abstract design for a neural network

Examples of architectures: * Fully Connected Neural Networks * Convolutional Neural Networks (CNNs) * Recurrent Neural Networks (RNNs) * Transformers (self-attention)

locality:The choice of architecture is often informed by assumptions about the data, based on our domain knowledge

translation invariance: The features that make up a meaningful object do not depend on that object’s absolute position in the input.

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 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.

Convolutational 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?

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 smoothe it with our rolling average filter first: $\hat{y}_j = wh_j + b$
$h_j = (f \times x)_j$

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 (closest high, later smaller) Alternative: Let the model choose the optimal filter parameters, based on the training data. How? Gradient descent!

Backpropagation for 1 D Layer

#TODO

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):

#TODO

Edge cases: * Possible strategies for when the filter overlaps with the edges (beginning/end) of x: * Outputs where filter overlaps with edge are undefined, i.e., h has fewer dimensions than $x(K - 1$ fewer, to be exact) * Pad x with zeros before applying the filter * Pad 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 f to the j'th window of 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 $X \in RJ1\ddot{O}...\ddot{O}JL$ be a tensor that has L axes and let $F \in RK1\ddot{O}...\ddot{O}KL$ 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, #TODO are not available on day j.) Then the output $H = F \times X$ is a tensor with L axes, where

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:$X \in RJ1\ddot{O}...\ddot{O}JL\ddot{O}M$ 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 X:

#todo

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

#todo

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 X independently, and their outputs are stacked (they form a new axis in the output). So let this be our input and filter bank:

Then our output is a tensor of shape $H \in RJ1\ddot{O}...\ddot{O}JL, N$ * (assuming that we are padding the first L axes of X to preserve their dimensionality in H) where:

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

Pooling layers

Pooling layers are parameter-less. Divide axes of 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):

#TODO

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"

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

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: $XinRJ\ddot{O}M$ * Filter bank of $Nfilters : F \in R^{K\ddot{O}M\ddot{O}N}; K < J$ * Output matrix: $HinRJ\ddot{O}N$ * (assuming that we padded 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 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:

Lecture 7 Recurrent Neural Networks

- 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" h
  - RNN consumes one input $x(j)$ per time step j
  - Update function: $h(j) = f(x(j), h(j-1); \theta)$
  - where parameters θ are shared across all time steps

Vanilla RNN

- h is initial stage
- x is our input
- theta be our parameters

Which words does h depend on?

The cat sat:

- h(1) = the
- h(2) = the cat
- h(3) = the cat sat

Vanilla RNN cell

With a lose node, we have a parametrized linear layer with non-linearity.

#TODO

Training RNN's

RNN for binary sentence classification:

Iterative compute hidden state.

Backpropagation through time

Assume we calculated partial derivatives. Since we are pretending its a normal feed forward normal, we can apply backpropagation. 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 (j) dummy parameters θ , which are identical copies of theta. Gradients are simply the sum of the dummy parameters.

Vanishing Gradients

- Means that impact that input has on gradient is smaller when it is further away from the loss in the computational graph.
- Means that if words that your RNN should be paying attention toa re far from the loss the network will not adjust its weights to those words.

Exploding gradients

Can adjust non-linearity with a derivative larger than 1. Vanisher is the less evil that exploding.

Long Short Term Memory Network (LSTM)

Became popular in 2010.

Two states:

- h ("short-term memory")
- c is ("long-term memory")

Candidate state h in Rd corresponds to h in the Vanilla RNN d. Interactions are mediated by "gates" in (0, 1) , which apply element wise:

- Forget gate f decides what information from c should be forgotten
- Input gate i decides what information from h should be added to c
- Output gate o decides what information from c should be exposed to h
- Each gate and the candidate state have their own parameters.
- Gradient highway from c_j to cj_1 with no non-linearties or matrix multiplications.

At time step j we calculate the gate vector. We add everything together including the bias and apply an element based sigmoid nonlinearity. Next we calculate the candidate state. In the next step, we calculate the forget gate, the previous memory. Since the forget and input gate are close to 0 and 1. We are selectively adding new entries with i. Which entries we erase is up to the model. We apply our nonlinearity to see in the next step.

insert graph

- blue cell parametrizations
- input output and forget gate + candidate state
- Input to candidate state and calculate the new state.

- red arrow is the gradient highway.
- There are no nonlinearities on the highway.
- All gradients that are passed to close to 1 are moving without restricition and all close to 0 are stopped.
- Gradients can travel a lot further than in a Vanilla RNN.

Gated Recurrent Unit

Proposed by *Cho et al. (2014)* GRU only has one state and 3 sets of parameters. Lightweight alternative to LSTM, with only one state and three sets of parameters. State h is a dynamic "interpolation" of long and short term memory. It is up to the model, how many dimensions to assign to the memory. Reset gate r in (0, 1)ˆd controls what information passes from h to candidate state h hat. The update gate z interpolates between h and h_bar.

At step j we first calculate the reset and update gate. Then we select some. Then we calculate the candidate state. We use the z to interpolate. The new state h_j:

#TODO

When we plot the cell as a computation cell, we have a reset gate, update gate and candidate state. We have the interpolation between the previous state and the candidate state. Again, we see that we have a gradient highway. Any gradients that respond to 0 in the update gate z. Any gradients that are stuck to 1 are kept.

Vanilla RNN worse, and the GRU and LSTM is not clearly better. GRU is faster to train though. Always best to use both, and if no time use GRU.

Be able to name components and be able to stay what they do conceptually. Also, be able to tell which architecture they belong to.

#TODO

RNN Applications

Classifying sentences, or tagging problems. In part of speech tagging, we want to tag every word with its part of speech. First we would code into hidden words. Imagine h stems from one of the above models. Then we feed the H vector into a linear layer. The linear layer is shared between all time steps. The output of the hidden layer is a softmax function. #TODO During inference, when evaluating our model or making predictions, we take the argmax of our model. We take the tag that maximizes our abilities.

Next step, we do **autoregressive language modelling.** After reading the beginning of sentence tokening, we want to read the next word. The output space is the entire set of next words. The predicted probability is compared against the true next words. During training our input is usually the next word. During inference, when we use the model to generate the actual text, we would instead feed back the actual predictions. Gummybear, we would feed gummybear into timestep 2.

Sequence to sequence Model (Machine Translation)

The 2 RNN's may share parameters or be completely separate. First the encoder #TODO

page is then the initial state of the decoder. A decoder functions exactly like an ALM. During training the inputs of the decoder are the words of the true target sentence. During inference, we don't know what the true target sentence is.

Extensions: Multilayer RNNs, bidirectionality

Multilayer RNNs

Is a stack of numerous RNNs (also mixture). Each RNN has its own parameters. The input vectors to the first RNN are the word embedding as usual.

Bidirectional RNNs

Reads from both directions. Consists of two RNNS with two parameters. Theta forward and backward.

The bidirectional RNN yields two sequences of hidden states: #TODO # Important Question: If we are dealing with a sentence classification task, which states should we use to represent the sentence?

Which are the states that have seen the entire sentence, are the forward h_j and backward h_j. Usually we would concatenate these two and feed them towards our final layer.

#TODO

- Concatenate [h ; h], because they have "seen" the entire sentence

#TODO For tagging task, represent the j'th word as [h ; h] # Important 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).

Limitations of RNNs

At a given point in time j, the information about all past inputs is "crammed" into the state h (and c for an LSTM). So for long sequences, the state becomes a bottleneck. Long sentence but large dimensionality, 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

- One query vector
- J key vectors:
- J value vectors:
- Scoring function
- Maps a query-key pair to a scalar ("score")
- a may be parametrized by parameters θ_a

Step 1: Apply a to q and all keys kj to get scores (one per key): Step 2: Turn e into probability distribution with the softmax function Step 3: alpha-weighted sum over V yields one Rdv -dimensional output vector o

Attention in Bahdanau et al., 2015

- Encoder simple bidirectional RNN.
- Source sentence: (x1 . . . xTx), encoded as hidden states (h1 . . . hTx) by encoder RNN
- Decoder is defined as: hidden state of decoder given the annotations from the encoder is computed by.

#TODO

$s = tanh(WE_y i - 1 + U[r_i * si_1] + Cc_i)$

The i matrix is the embedding matrix and i-1 is the embedding of the previous words. Dropped the bias matrix.

Exercise: Recognize the architecture (hint: they don't use boldface) * This is a GRU * si are the states (which we called h(i)) * Eyi-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 W*() and V()*), and they drop the bias * C* seem to be additional parameter matrices . . . but where does the vector ci come from?

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 ci (o in our recipe) as an attention-weighted sum over encoder states. So the encoder states hj are our value vectors (vj in our recipe).
- The second equation is step 2.
- The third equation is step 1, which defines the raw scores.
- si-1 (previous decoder state) is the query vector (q in our recipe)
- The encoder states hj are also our key vectors (so kj = vj in this case)

Scoring function is a Feed-Forward Net:

Simple feed forward net. Can tell from the fact that its in the appendix, its not conceptually important.

What does attention learn?

Attention matrix, pairwise states of decoder and encoder. Conceptually the encoder decoder does not need to remember but needs to know where it has to look for it. Major improvements especially for long sentences.

Effect on translation quality for long sentences

- 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.

Lecutre 8 Self-Attention and the Transformers

Limitations of RNN's

- In an RNN, at a given point in time j, the information about all past is "crammed" into the state vector h_j and c_j for LSTM
- So for long sequences, the state becomes a bottleneck inputs x for an LSTM).
- 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.

Basic Recipe:

- One query vector q
- J key vector K
- J value vector V
- Scoring function a (maps query key pair to scalar)
- a may be parametrized b parameter θ_a
- job of a is to scale a score
- Step 1: Apply a to q and all keys kj to get scores (one per key): (put scores into e)
- Step 2: Turn e into a probability distribution with the softmaxfunction
- Step 3: alpha-weighted sum over V yields one dv -dimensional output vector
- Intuition: alpha_j is how much "attention" the model pays to vj when computing o.

Analogy

- maps with geolocations. K
- current weather conditions V

- q vector which is a new location
- for this location we want the weather location (q request for information)
- e_j is the relvance and is the inverse difference between weather condition and geolocation
- weighted sum of known weather conditions aka. stations that are close to the weather conditions

Attention in neural networks

Contrary to our geolocation example, the q, kj and vj 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 What model learns is based on the training data. Model learned french words via training data without command.

- 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 pre-train Transformers on lots of data

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")

encoder left, decoder on right. If you dont need a decoder, can only use encoder. Orginally proposed as sequences to sequence model. Input Y, (e.g. German Translation). Attention to transformer box. Every block is repeated n times. Each block consists of multiple layers.

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 (Assume we have 2 sequences. Y attends to X)
- Self-attention (Only one sequences and X attends to itself; X=Y)

Cross-attention

Here, we describe cross-attention. Self-attention can easily be derived by assuming X = Y . Three W matrices. We transform Y into a matrix of query vectors: We transform X into matrices of key and value vectors:

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:

#TODO

- Note: This requires that dq = dk

- Attention weights and outputs are defined like before (steps 2 and 3 of the basic recipe):

Parallelized attention

We want to apply our attention recipe to every query vector qj. We could simply loop over all time steps 1 smaller or equal j smaller or equal y and calculate each oj independently. Then stack all oj into an output matrix. But a loop does not use the GPU's capacity for parallelization. So it might be unnecessarily slow.

Do some inputs (e.g., qj) depend on previous outputs? If not, we can parallelize the loop into a single function: 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 si (a.k.a. the query of the i + 1'st step) depends on ci (a.k.a. theattention output of the i'th step), see last lecture:

Step 1: The parallel application of the scaled dot product to all query-key pairs can be written as: Step 2: Softmax with normalization over the second axis (key axis): Step 3: Weighted sum

#TODO

GPUs like matrix multiplications * usually a lot faster than RNN! But: The memory requirements of E and A are O(Jy Jx) 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.

Multi-layer attention

Sequential application of several attention layers, with separate parameters The transformer: Sequential application of transformer blocks There are some additional position-wise layers inside the Transformer undergoes some additional transformations before becoming the input to the next Transformer block n + 1

Multi-head Attention

Application of several attention layers ("heads") in parallel. Several attention layers are called heads. Each Theta contains three W matrixes. For every head, compute in parallel: Concatenate all O along their last axis; then down-project the concatenation with an additional parameter matrix. 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 Standard for transformer models.

Masked self-attention

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 h_j depends **only** on past inputs. Thus RNN is unable to cheat. Unable to look at future inputs before predicting them.

Self-attention for autoregressive LM & decoding

With attention, all oj depend on all vj' (and by extension, all xj'). This means that the model can easily cheat by looking at future words (red connections) Useless at inference time when not given.

So when we use self-attention for language modeling or in a sequence-to-sequence decoder, we have to prevent oj from attending to any vj' where j' > j. By hardcoding ej,j' = -infinity when j' > j (in practice, "infinity" is just a large constant) That way, exp(ej,j') = alpha_j,j' = 0, so vj' has no impact on oj

Parallelized masked self-attention

Step 1: Calculate E like we usually would Step 1B:

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 (as not paralelized), but at least we don't have to worry about masking anymore

Residual connections and layer normalization

Benefits: Information retention (we add to X but don't replace it)

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 oj is normalized independently

#TODO

The Transformer architecture

Position encodings

Can self-attention model word order? Does it get same vector representation. Hard to model relationship stuff and object subject differentation.

Since addition is commutative and the permutation is bijective, it is sufficent to show that: #TODO

Lets show equality for v vectors. Since we know word embeddings are equal and we apply same parameter matrix, it follows that the parameter vectors are also equal.

Step 2: Show equality also holds for alphas. Since the sum in the denominator is commuatitve and permutation is bijective it is suffcient to show that equality holds for the underlying e scores. Using the definition of e, we know all word embeddings are equal. Again we are applying the same parameter matrixes. The same equality holds for query and key vectors. Since that is true, we also know the dotproduct is equal.

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.) Example: "john loves mary" vs. "mary loves john"

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 are 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.

Question: 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 wird enbedding a position embedding
- Option 1: Trainable parameter P (dim Jmax times d) like Devlin 2018.
- Obvious disadvantage becuase cannot deal with sentences longer than Jmax
- Option 2: Sinusiodal position embeddings (deterministic)

Remember: Sinusiodal position embeddings make pretty big models. Increase of dimensionality seen via depth. The first dimensionalities have short wave length and allow to differentiate between immediate neighbors. The slower ones, deeper in the model, and give model more of an idea whether word at beginning or end.

Lecture 8b) Hyperparameter Optimization

Hyperparameters

Things that may affect the performance of a model on a task, without being a trainable parameter in theta.

- 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 non-linearity (ReLU vs. tanh vs. ...)
- 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)
- 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

Train / Validation / Test sets

Needed: separate train/validation*/test sets

- a.k.a. dev(elopment), 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

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 smaller or equal n smaller or equal 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”.

A hyperparameter configuration

A hyperparameter configuration is a specific choice of hyperparameters. Every hyperparameter has a range, or a set of permissible values.

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

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 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, . . .)

Lecture 9 Transfer Learning

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

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.(word more than sum of characters)

- Using (sets of) n-grams is kind of heuristic.

Smart alternatives:

- BytePair encoding
- WordPiece
- SentencePiece

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

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:(to get vocab size)

1. Initialize the 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" (triangle! 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*

WordPiece

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 the vocabulary with basic Unicode characters (22k for Japanese, 11k for Korean) #TODO Spaces are indicated by an underscore attached before (of 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

- Adoptions:
- Application to Western languages leads to a lower number of basic units (~500)
- Add space markers (underscores) only at the beginning of words
- Finally vocabulary sizes between 8k and 32k yield a good balance between accuracy and fast decoding speed

Independent vs. join endocdings for source and 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

SentencePiece Kudo et al. (2018b)

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

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

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

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 and a backward LM to arrive at the following loglikelihood:

Character-based (context-independent) token representations

Two-layer biLSTM as main architecture:

- Two context-dependent token representations per layer, i.e.
- Four context-dependent token representations in total:
- Five representations per token in total:

ELMO

Including ELMo in downstream tasks: Calculate task-specific weights of all five representations:

Trainable parameters during the adaption:

- s_j task are trainable (softmax-normalized) weights
- gamma 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

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

Alleviation/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.

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 optimiziation

Embedding layer + three LSTM layers + Softmax Layer

LM fine-tuning:

- Discriminative fine-tuning Classifier fine-tuning:

- Concat Pooling
- Gradual unfreezing

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:

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 BooksCorpus Wikitext-103 CommonCrawl Chelba et al. (2013) Zhu et al. (2015) Merity et al. (2016) 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 (~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:

Lecture 10 BERT

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

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.

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.

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.

OpenAI GPT Radford et al., 2018 abandon the use of LSTMs. The 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.

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.

Core of Bert

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

Major architectural differences:

Devlin et al. (2018) ELMo uses to 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-Attenion

- 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: 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

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 s2 follows s1 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: #todo [CLS] token as sequence representation for classification [SEP] token for separation of the two input sequences

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 BERTBase (BERTLarge)

Loss function: LossBERT = LossMLM + LossNSP

- 1.000.000 steps on batches of 256 sequences with a sequence length of 512 tokens

Pre-Training BERT – Maximum sequence length

Limitation: Source: Vaswani et al. (2017)

BERT can only consume sequences of up to 512 tokens Two sentences for NSP are sampled such that lengthsentenceA + lengthsentenceB smaller or equal 512 Reason: Computational complexity of Transformer scales quadratically with the sequence length * Longer sequences are disproportionally expensive

Fine-Tuning BERT

#TODO

Successors of 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.

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 changer, 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".

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.

RoBERTa Liu et al., 2019 concentrate on improving the original BERT architecture by (1) careful hyperaparameter 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.

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.

BERTology

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 .. * .. explanining 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

RoBERTa

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)

Dynamic vs. Static Masking

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) -> aprox. 125M (360M) for the BASE (LARGE) variant

Performance differences:

#TODO

ALBERT

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 » E enlargens model capacity without increasing the size of the embedding matrix, since $O(V \times H) > O(V \times E + E \times H)$ if H » 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

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 RoBERTa: * https://github.com/pytorch/fairseq/tree/master/examples/roberta 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

Lecture 11 Alternatives to BERT

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 smaller or equal 512

XLNet Yang et al., 2019 Autoregressive (AR) language modeling vs. Autoencoding (AE)

- AR: Factorizes likelihood to p(x) = #todo Tt=1 p(xt|x< t)
- Only uni-directional (backward factorization instead would also be possible)
- AE: Objective to reconstruct masked tokens x from corrupted sequence ^x: p(x|^x) =#todoTt=1 mt · p(xt|^x), mt 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 z of the index sequence [1, 2, . . . , T] * Used to permute the factorization order, not the sequence order. Original sequence order is retained by using positional encodings PLM objective (with ZT as set of all possible permutations):

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 Sets of queries (Q), keys (K) and values (V) from content stream* Yields a content embedding denoted as $h\theta(xzsmallerorequalt)$

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 $g\theta(xz < t, zt)$

Additional encodings: Relative segment encodings: * BERT adds absolute segment embeddings ($EA\&EB$) * XLNet uses relative encodings ($s + \&s-$) Relative Positional encodings: * BERT encodes information about the absolute position (E0, E1, E2, . . .) * XLNet uses relative encodings (Ri-j)

- Partial Prediction: Only predict the last tokens in a factoriztion order (reduces optimization difficulty)
- Segment recurrence mechanism: Allow for learning extended contexts in Transformer-based architectures, see
- No independence assumption:

T5

- A complete encoder-decoder Transformer architecture
- All tasks reformulated as text-to-text tasks
- From BERT-size up to 11 Billion parameters
- 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

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

(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)

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 xcorrupt to D Generation of samples: with approx. 15% of the tokens masked out (via choice of k) D predicts whether $xt, t \in 1, ..., T$ 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)

Using the masked & corrupted input sequences, the (joint) loss can be written down as follows: Loss functions: Combined:

with lambda set to 50, since the discriminator's loss is typically much lower than the geneator's.

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)

Weight sharing (experimental): Same size of G and D: All weights can be tied G smaller than D: Share token & positional embeddings

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.

Model distillation

Model compression scheme: Motivation comes from having computationally expensive, cumbersome ensemble models.Bucila et al. (2006) Compressing the knowlegde 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: #TODO Knowledge transfer via soft targets with high T from original model. When true labels are known: Weighted average of two different objective functions

DistilBERT 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 Lce = #TODO i ti · log(si) + MLM-Loss Lmlm + Cosine-Embedding-Loss Lcos 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

The O(n2) 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?

Efficient Transformers Tay et al. (2020)

Broad overview on so-called "X-formers":

Efficient & fast Transformer-based models * Reduce complexity from O(n2) to (up to) 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 #todo

Introducing Patterns

Reasoning:

Making every token attend to every other token might be unnecessary Introduce sparsity in the commonly dense attention matrix

Example:

#ToDO

Self Attention

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

DeBERTa

Disentangled Attention: Each token represented by two vectors for content (Hi) and relative position (Pi|j) Calculation of the Attention Score:

#todo

with content-to-content, content-to-position, position-to-content and position-to-position attention

Disentangled Attention Standard (Single-head) Self-Attention:

#todo

GPT Models Lecture 12

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

- 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?
- 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?
- 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.** 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. *There is always a shift in reality. A model trained for 2020 wont be able to adapt to changes in 2021. This is the finetuning. Great idea in priciple but not generalizable/external validity*

GPT: Intro

Like BERT, GPT is a language model. But not MLM, but a conventional language model: it predicts the next word (or subword). *(this rose smells good example)* Like BERT, GPT is trained on a huge corpus, actually an even huger corpus. (much larger) 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.

In-context learning which technically is not deep learning.

GPT: Two types of learning

Does in-context learning (similar problems) Models sees examples.

GPT: Effective in-context learning

zeroshot, oneshot, fewshot when zeroshot, the accruacy is very low. It can leverage a single training example a huge amount. By adding more, it gets better. The big jump is the first. With enough labeled examples, you dont need a task description. With one shot, there is a huge millage due to task descriptions.

X-shot comparison and effect of larger corpora

few shot is performing best but only marginal difference with huge corpora.

Fine-tuning (not used by GPT)

Zero-shot (no gradient update)

The model predicts the answer given only a natural language description of the task. No gradient update.

#TODO one shot:

Few-shot (no gradient update) Gets several examples. No gradient update performed.

Architecture: Effect of size on model. Getting better, the more parameters are given.

Compute PeaFLOP/s-days

Looks linear, the more compute the use, the better your complexity. The larger model, even better. The main concern is that there is not enough text.

GPT3 results on tasks

Impression of types of tasks used in NLP to test capability of models.

Lambada task

GPT3 we have context and the model is asked to predict the next word (or several). It does not change its parameters. It is just doing the language change objective. Designed to make it hard to predict the next word for language model. SOTA is state of the art. GPT3 is better than SOTA. Not specifically trained on this task. It gets better performance in the zero shot set. Closed book question answering QA.

Winograd task

Inference based on context. GPT3 does well but not state of the art here. Just a language model but still gets common sense well.

ARC task

Question about physics in high school. Context: which palms produce the most heat example.

RACE task

Long task and pick which continuation is the correct one. Huge gap between GPT 3 and SOTA. Need to use state of language model to store stuff (short term memory)

SuperGLUE task

SuperGLUE BoolQ (Yes/No, here about physics. Can have problem cannot go back; GPT3 doesnt so wekk) CB (true/false/neither) RTE (similar to natural language inference) WiC MultiRC (true/false)

Wic task

Yes/no: Is the word used in the same way? Problem again GPT cannot go back and compare context. GPT3 cannot do that. Terrible performance, also compared to BERT GPT3 cannot store and compare thing.

COPA Task

Compare two answers and pick correct one. Again physics, still comparably well.

WSC (Winograd Schema Challenge) task

Similar to lambada but to make decision difficult because dependent on understanding of the text. Performance again really bad.

ReCoRD task

Long context and decide whether continuation is correct or not. GPT is able to do well here.

Superglue ends here.

ANLI task

Again far below state of the art. More parameters probably able to do better.

SAT Analogies task

#TODO

Miscellaneous

GPT3 can correct grammar. Language model learns correct sequence of words and hence it is not suprising.

Context given to GPT3:

- Three "training" articles to condition gpt3
- Title and subtitle of a 4th article
- gpt3 then has to generated 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.

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.

GPT limitations

Next generation

- Repetitions (can easily tell that generated texts have rep. and are not human generated)
- Lack of coherence (normal human would not say on the follow up)
- Contradictions (the dodgers scored 3:2 and later say they must have lost.)

Common sense physics is a huge problem for gpt3. E.g., "If I put cheese in the fridge, will it melt?" See below GPT3 doesnt have experience in the real world.

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
- GPT3 CANNOT GO BACK

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. This is about the fact that we always predict the next word. We go through the text to predict the next and we cannot learn things from this task we need for natural language understanding. Our behaviour as humans is goal oriented while computers not.

Low sample efficiency (how well exploit text being trained on) 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.

Size/Interpretability/Calibration

- Difficult to use in practice due to its size.
- Behavior hard to interpret.
- Probability badly calibrated.

In human discourse, we try to mix in interesting words from time to time. If you have sequence of words and now you are thinking about the next word, then you will have a long tail of words with low probability. Humans select words that are unlikely. They select words that fit the context but GPT has a huge problem doing that. Does GPT3 learn from context?

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.

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

GPT3 hype Not really capable of writing an article. Wrote Reddit posts. Average quality on Reddit is pretty poor though. The verge talks about the text generating AI it said too dangerous to share. Very expensive to train this model. It is not widely accesible. Once you have trained the model, you can use it in many different ways. It is not clear whether it is a knockout argument against it.

Marcus & Davis (1) Anti-Article

(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.

(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.

(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.

(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.

(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.

(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.

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 ("Kennzeichungspflicht")

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.

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

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.

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

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).

Multilingual models Lecture 13

Bert Recap

mBERT

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?

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

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) #### (appreciation doesn't imply benefit)

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. #### contradiction because either vacation or not

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.

if you form a line which is same as beginning a line. All info first sentence follows in the second sentence.

XNLI

- 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."
- Translate train(training set translation into foreign) vs translate test vs zero shot (no translation)
- **advantage mBert** you only need one model. you dont 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 multlingual 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

- Single model in mutlilingual setting
- Supports easy transfer learning
- In Particular: transfer learning for low resource languages

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 (than Bert which is a huge improvement)
- Claim: performance competitive with monolingual model As in the case of mBERT: no parallel data is used.
- Much larger dataset

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.

- The transfer-interference trade-off: Low-resource languages benefit from scaling to more languages, until dilution (interference) kicks in and degrades overall performance.
- Adding more capacity to the model alleviates the curse of multilinguality, but remains an issue for models of moderate 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.

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.

Performance

As good as monolingual models

Challenges for mBERT and XLM-R

- Vocabulary coverage
  - (same preprocessing despite language difference)
- 250k not enough for 100 languages
- Low-resource transfer doesn't work well for very different languages
  - How to evaluate?
  - XNLI?

Paper: Identifying Elements Essential for BERT's Multilinguality

Fake-English: No vocabulary overlap with English Good performance without vocabulary overlap

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.

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.