Deep Learning for NLP Summary

Daniel Saggau

2/4/2021

Lecture 4 Introduction to Machine Learning - Optimization, Deep Feed Forward Networks, Backproagation, Regularization

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 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 CNN

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)

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: $y = wh + b; h = (f \times x)jjjj$ But maybe we should weight the datapoints in our 7-day window differently? Maybe we should give a higher weight to datapoints close to the current day j? We can manually engineer a filter that we think is better, for example: #todo Alternative: Let the model choose the optimal filter parameters, based on the training data. How? Gradient descent!

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): K #TODO Edge cases: * Possibel 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 $X \in RJ1Ö...ÖJL$ be a tensor that has L axes and let $F \in RK1Ö...ÖKL$ 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:$X \in RJ1$Ö...$ÖJLÖM$ 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 $H \in RJ1$Ö...ÖJL, 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): 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 $Nfilters: FinRK\ddot{O}M\ddot{O}N; K < J$ * 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 RNN

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

#TODO

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

LSTM

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 elementwise: * 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

Gated Recurrent Unit

Proposed by *Cho et al. (2014)* 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 Reset gate r in (0, 1)^d controls what information passes from h to candidate state h hat

Bidirectional RNNs

The bidirectional RNN yields two sequences of hidden states: #TODO Question: If we are dealing with a sentence classification task, which states should we use to represent the sentence? #TODO * Concatenate [h ; h], because they have "seen" the entire sentence #TODO For tagging task, represent the j'th word as [h ; h] 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 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 theta 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

#todo

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)
- Step 1: Apply a to q and all keys kj to get scores (one per key):
- Step 2: Turn e into a probability distribution with the softmax function
- 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? 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.
- 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.

Lecture 8b) Hyperparameter Optimization

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 nonlinearity (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

CV

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: * Foreveryfold 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.
- 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:

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

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 × H) > O(V × E + E × 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

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)

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.

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.

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.”

XLM-R

Investingating the mystery