transformer fine-tuning for text classification

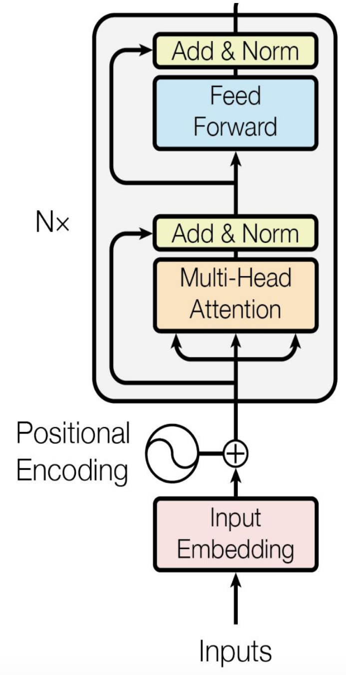
motivation, intuition, and methods

Recap

pre-trained transformer encoder models

Contextualization

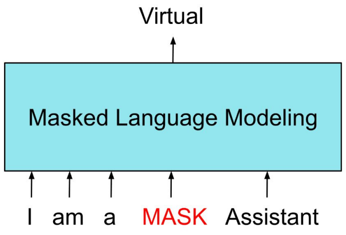
- pre-trained transformer encoder models allow us to generate **contextualized embeddings** of input sequences
- the contextualization enables, among others, word sense disambiguation
- overall, transformer embeddings are rich representations of words contextual meaning



pre-trained transformer encoder models

LM pre-training

- **training** in machine learning: use labeled data (input–output pairs) to optimize a prediction model
- encoders like BERT use **masked language modeling**: predict masked-out words in sequence of words
- **motivation**: pre-training a model to perform language modeling on large text datasets allows it to mimic humans' natural language understanding abilities



(a)

BERT & Co. use this

pre-trained transformer encoder models

Transfer learning

- machine learning approach to reuse general-purpose model for a specific task
- ***intuition*** knowledge gained while learning to perform a general task (e.g., language modeling) can be applied to quickly solve related task (e.g., sentiment classification)
- ***premise*** general features learned during pre-training can be relevant for more specific tasks

Supervised text classification

What’s your plans?

Based on your work on the posters on Monday,

- who wants to classify texts?
- who wants to classify texts into predefined categories?
- who already has or will have (human-)labeled data?

Supervised text classification

Task: assign each text to a predefined list of categories

Approach:

- take **labeled examples** $(y_i, \mathbf{x}_i)$
- y_i in predefined list of **label classes**
- **train** machine learning model

Example of labeled text dataset

text	***label***
I found transformers and LLMs confusing and intimidating.	negative
Then I took a course with Lisa and Hauke.	neutral
Now I feel very confident that I can master these methods.	positive

Supervised text classification

Task: assign each text to a predefined list of categories

Approach:

- take **labeled examples** (y_i, $\mathbf{x}_i$)
- y_i in predefined list of **label classes**
- **train** machine learning model

Classic machine learning

1. construct features from set of labeled texts (e.g., bag-of-words document-term matrix)
2. select a machine learning algorithm (e.g., Naive Bayes, Random Forest, XGBoost)
3. use the labeled data and the ML algorithm to train a model
   - ➢ optimization problem: find the model parameters that lead to best predictions of observed labels given the input features
4. apply trained model to
   - ➢ labeled held-out data ⇒ evaluation
   - ➢ unlabeled data ⇒ “inference” (prediction)

Supervised text classification

Task: assign each text to a predefined list of categories

Approach:

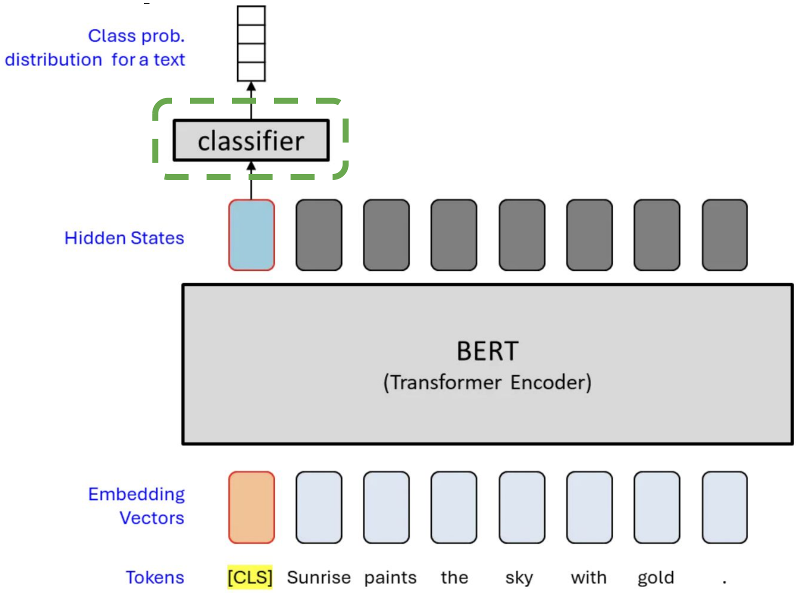
- take **labeled examples** (y_i, $\mathbf{x}_i$)
- y_i in predefined list of **label classes**
- **train** machine learning model

Transfer learning approach: fine-tuning

1. take a pre-trained (encoder) model (e.g., BERT) to generate embeddings ⇒ features
2. add a classification layer on top of the pre-trained (encoder) model
3. use the labeled data to update ("fine-tune") the model parameters
   - ➢ optimization problem: update the model parameters that lead to best predictions of observed labels given the input features
4. apply the fine-tuned model to
   - ➢ labeled held-out data ⇒ evaluation
   - ➢ unlabeled data ⇒ "inference" (prediction)

Supervised text classification

Figure shows BERT model with classification head

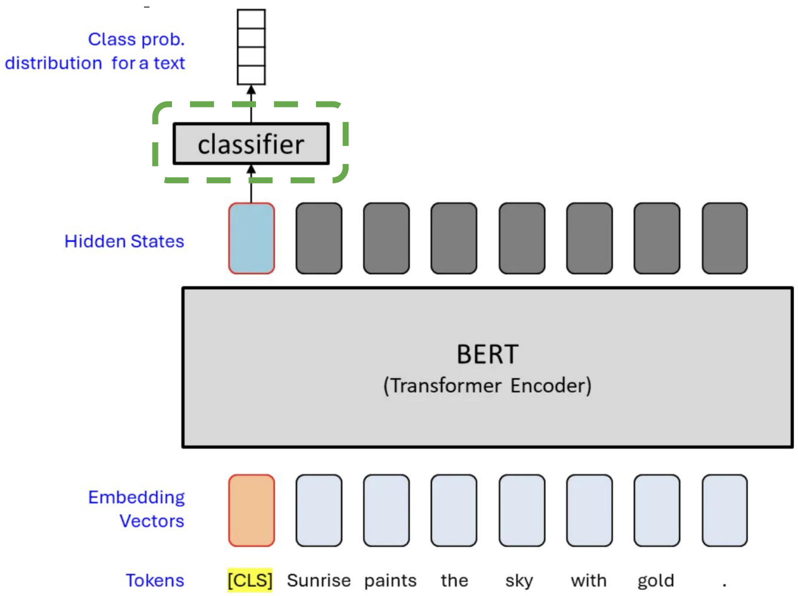


Transfer learning approach: fine-tuning

1. take a pre-trained (encoder) model (e.g., BERT) to generate embeddings ⇒ features
2. add a classification layer on top of the pre-trained (encoder) model
3. use the labeled data to update ("fine-tune") the model parameters
   - ➢ optimization problem: update the model parameters that lead to best predictions of observed labels given the input features
4. apply the fine-tuned model to
   - ➢ labeled held-out data ⇒ evaluation
   - ➢ unlabeled data ⇒ "inference" (prediction)

Supervised text classification

Figure shows BERT model with classification head



The classification layer

- **input**: d_{model}-dimensional contextualized embedding of [CLS] token
- **output**: $n_{classes}$-dimensional vector ⇒ "logits"
- applying **softmax** function to logits creates pseudo-probabilities (in [0, 1], sum to 1)
- *note*: if $n_{classes}$=2, this is a logistic regression

Example of observed labels vs. predictions

	class 1	class 2	class 3	class 4
labels	0	0	1	0
prediction	0.1	0.2	0.6	0.1

Optimization

four ingredients

1. **labeled data**: texts assigned to label classes
2. **prediction model**: takes texts as inputs and predicts which label class they belong to
3. **loss function**: measures how far off the model's predictions are from the actual observed labels (the higher, the worse)

The cross-entropy loss

define as

$$L = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

where

- N is the number of label classes.
- y_i is 1 if the observed class label is i and 0 otherwise
- $\hat{y}_i$ is the predicted probability for class i.

Optimization

four ingredients

1. **labeled data**: texts assigned to label classes
2. **prediction model**: takes texts as inputs and predicts which label class they belong to
3. **loss function**: measures how far off the model's predictions are from the actual observed labels (the higher, the worse)

The cross-entropy loss

simplifies to

$$L = -\log(\hat{y}_{\text{true}})$$

when only on label is correct per example

Example of low loss

	class 1	**class 2**	**class 3**	**class 4**
labels	0	0	1	0
prediction	0.1	0.2	0.6	0.1
loss			–log(0.6)	

Optimization

four ingredients

1. **labeled data**: texts assigned to label classes
2. **prediction model**: takes texts as inputs and predicts which label class they belong to
3. **loss function**: measures how far off the model's predictions are from the actual observed labels (the higher, the worse)

The cross-entropy loss

simplifies to

$$L = -\log(\hat{y}_{\text{true}})$$

when only on label is correct per example

Example of high loss

	class 1	**class 2**	**class 3**	**class 4**
labels	0	0	1	0
prediction	0.6	0.2	0.1	0.1
loss			–log(0.1)	

Optimization

four ingredients

1. **labeled data**: texts assigned to label classes
2. **prediction model**: takes texts as inputs and predicts which label class they belong to
3. **loss function**: measures how far off the model's predictions are from the actual observed labels (the higher, the worse)

The cross-entropy loss

pred. prob	loss
0.01	-log(0.01) = 4.61
0.10	-log(0.1) = 2.3
0.25	-log(0.25) = 1.39
0.50	-log(0.5) = 0.69
0.75	-log(0.75) = 0.29
0.99	-log(0.99) = 0.01

Optimization

four ingredients

1. **labeled data**: texts assigned to label classes
2. **prediction model**: takes texts as inputs and predicts which label class they belong to
3. **loss function**: measures how far off the model's predictions are from the actual observed labels (the higher, the worse)
4. **optimization algorithm**: takes the loss and updates the prediction model's parameters

Gradient descent optimization

Gradient: The gradient $\nabla_\theta L$ tells us how the loss changes with respect to each parameter (weight) in the model.
Our goal*: Update* the parameters such that we *reduce* the loss

$$\nabla_\theta L = \left(\frac{\partial L}{\partial \theta_1}, \frac{\partial L}{\partial \theta_2}, \ldots, \frac{\partial L}{\partial \theta_n} \right)$$

Analogy: It's like how adjust coefficients in an OLS regression to minimize the deviations from the regression slope.

Optimization

four ingredients

1. **labeled data**: texts assigned to label classes
2. **prediction model**: takes texts as inputs and predicts which label class they belong to
3. **loss function**: measures how far off the model's predictions are from the actual observed labels (the higher, the worse)
4. **optimization algorithm**: takes the loss and updates the prediction model's parameters

Mini-batch stochastic gradient descent

- Unlike an OLS regression,
  - neural nets have complex interactions
  - relations between in- and outputs are nonlinear
- Unlike OLS, there exists no closed-form solution to find optimal parameters

Solution

- take a small *batch* of examples; generate predictions; and then update the model parameters given the loss for this batch
- iterate over all examples in batches (= 1 epoch); repeat for n_{epochs}

Let have a look at some code illustration these ideas

Go to notebook finetuning_sequence_classifier_illustration.ipynb and follow along

Let’s code

Go to notebook finetune_sequence_classifier.ipynb and follow along