Lecture 7 — Transfer Learning & Fine-Tuning BERT

The Culmination of the Course

Tamara Grechanaya
Multimodal Computational Methods for Political Science

Summer Semester 2026

Contents

Learning objectives

By the end of this lecture, you should be able to:

1. Explain the pre-train / fine-tune paradigm and why it has become the dominant approach in modern NLP.
2. Describe Masked Language Modelling and how it provides a self-supervised training signal that requires no human labels.
3. Explain how BERT is adapted for classification: the role of the `[CLS]` token and the classification head.
4. Identify the typical hyperparameters for fine-tuning (learning rate, batch size, number of epochs) and explain why they differ from training a neural network from scratch.
5. Recognise the common mistakes in fine-tuning BERT and how to avoid them.
6. Make an honest comparison between fine-tuned BERT and classical baselines, identifying tasks where BERT helps and tasks where logistic regression is sufficient.
7. Distinguish BERT from its main successors (RoBERTa, DeBERTa, multilingual variants) and recognise when each is appropriate.
8. Understand NLI-based classifiers as a transfer-learning alternative that requires less labeled data.
9. Fine-tune a BERT model on a political classification task using the Hugging Face `transformers` library.
10. Articulate the methodological principles that apply across all the techniques covered in this course.

1 Part I — Transfer learning

1.1 The old way: training from scratch

Until 2018, the typical pipeline for a supervised NLP task was straightforward. You started with a randomly-initialised neural network. You collected labeled data for your specific task — perhaps 5,000 hand-coded political speeches. You trained the network on this labeled data until validation loss stopped improving. You then applied the trained model to new data.

This approach has a fundamental problem. The model has to learn *everything* from your labeled data: vocabulary, grammar, syntax, semantic relationships, world knowledge, *and* your specific classification task. A few thousand labeled examples is nowhere near enough for a neural network to learn all of that. The network either fails to learn meaningful patterns at all, or it memorises the training data and fails to generalise.

This is exactly why Lecture 5's simple neural network barely beat Lecture 2's logistic regression. With limited labeled data, classical methods with hand-engineered features (like TF-IDF) often match or exceed neural networks trained from scratch. The neural network's theoretical advantages are real — learning representations, capturing word order — but they require massive data to materialise.

1.2 The new way: pre-train, then fine-tune

Since 2018, the paradigm has shifted dramatically. The new approach has two stages.

Stage 1: pre-training. A large model (BERT, GPT, etc.) is trained on *enormous* quantities of unlabeled text — billions of words. The training task is not your specific classification task; instead, it is a generic "self-supervised" task like predicting masked words. This pre-training is done once, by organisations with substantial compute resources (Google, Meta, Microsoft, OpenAI). The result is a pre-trained model whose internal representations encode rich linguistic knowledge.

Stage 2: fine-tuning. You take the pre-trained model and adapt it to your specific task using your labeled data. Crucially, you do not start from scratch: you start from the pre-trained weights and gently adjust them. The model already knows language; you are just specialising it for your task.

The deep insight is that fine-tuning is teaching a polyglot a new dialect, not teaching a baby a language. The hard part — learning language at all — has been done by the pre-training. You only need to specialise the model for your task. This is why modern NLP works so well with so few labeled examples.

1.3 Why transfer learning works

Two complementary explanations help understand why pre-training transfers so well to downstream tasks.

Linguistic knowledge generalises across tasks. A model that knows what "increase" commonly relates to (cost, taxes, support, momentum, and so on) does not need to relearn that knowledge for every new task. The contextual representations BERT produces are useful simultaneously for sentiment analysis, topic classification, named entity recognition, and stance detection. The model captures *language*, not just task-specific patterns.

The model has already seen the words you care about. BERT was pre-trained on roughly 3.3 billion words from Wikipedia and books. Almost every word in your political corpus has been seen thousands of times in some context during pre-training. When you fine-tune, you are not asking the model to learn what "austerity" means — it already has a rich representation of austerity from pre-training. You are asking it to use that representation for your specific classification.

1.4 The economics of transfer learning

The economic implications are striking. Pre-training BERT cost an estimated several million dollars in compute alone. It required hundreds of GPUs running for days. Almost no academic political science group has the resources to do this — and almost none needs to. The pre-trained models are released for free under permissive licences.

Fine-tuning, by contrast, can be done on a laptop or, more practically, on Google Colab's free GPU. A typical political science fine-tuning experiment costs nothing in monetary terms and takes minutes of compute. You get the linguistic knowledge of billions of words for free, and you only pay for the marginal adjustment to your task.

This division of labour — big tech does the expensive pre-training, researchers do the cheap fine-tuning — has democratised state-of-the-art NLP. A political science PhD student with a laptop can now produce text-classification results that would have been impossible at any university five years ago.

2 Part II — How BERT was pre-trained

2.1 The architecture (recap)

BERT (Devlin, Chang, Lee, and Toutanova, 2018) is a Transformer encoder, the architecture we built in Lecture 6. Two sizes are common. **BERT-base** has 12 encoder layers, 12 attention heads per layer, 768-dimensional embeddings, and 110 million parameters total. **BERT-large** has 24 encoder layers, 16 attention heads, 1024-dimensional embeddings, and 340 million parameters.

The architectural innovation in BERT is not the Transformer itself — that came from the 2017 "Attention Is All You Need" paper. BERT's innovation is the *pre-training task*, which produced extraordinarily useful representations.

2.2 Pre-training task 1: Masked Language Modelling

Masked Language Modelling (MLM) is the core idea behind BERT. The procedure is simple to describe but extraordinarily powerful in effect.

For each training sentence, randomly hide some of the words and ask the model to guess them based on context. If the original sentence is "The Prime Minister announced new climate measures," the masked version might be "The Prime Minister `[MASK]` new climate `[MASK]`." The model must predict the original words at the masked positions.

Consider what the model must learn to predict the mask correctly. The predicted word must be a real English word (vocabulary and morphology). Position 4 follows a noun phrase, so it likely needs a verb (syntactic structure). A Prime Minister typically "announces," "proposes," or "rejects" things — not "cooks" or "swims" (semantic constraints). And the topic is climate policy, so position 7 should be a noun semantically related to government policy: "measures," "targets," "policies" (world knowledge). Doing this task billions of times across diverse text teaches the model an enormous amount about language, all without any human annotation.

The exact masking procedure is slightly more nuanced. BERT selects 15% of tokens randomly. Of those selected tokens, 80% are replaced with the special `[MASK]` token, 10% are replaced with a random word from the vocabulary, and 10% are kept unchanged but still "flagged" for prediction.

This 80/10/10 split is a regularisation trick. If the model only ever saw `[MASK]` during training, it would only learn good representations for the `[MASK]` token — and `[MASK]` never appears at fine-tuning or inference time. By occasionally training on random or unchanged words, the model is forced to maintain good representations for every token regardless of whether it might be masked.

2.3 The self-supervised insight

The deepest aspect of MLM is that it is *self-supervised*. The labels for training (the original words at the masked positions) come from the text itself. No human annotation is required. This is what allows BERT to be trained on billions of words: there is essentially unlimited unlabeled text on the internet, and MLM turns all of it into training data automatically.

This is the same principle behind Word2Vec from Lecture 3 (predict context words from a centre word, or vice versa). The general idea — train a model on a pretext task using the text itself as the source of labels, then extract the learned representations — is the foundation of every modern language model. It scales to any amount of available text.

2.4 Pre-training task 2: Next Sentence Prediction

The original BERT also used a second pre-training task: Next Sentence Prediction (NSP). The model is given two sentences and asked to predict whether the second sentence actually follows the first in the original document, or whether it is a random unrelated sentence.

The motivation for NSP was to teach the model about discourse-level relationships, which is useful for question answering and natural language inference. In retrospect, subsequent research (RoBERTa, ALBERT) showed that NSP contributes little to downstream performance. Modern BERT variants typically drop it entirely.

NSP nonetheless left an architectural legacy that affects how we use BERT today. The two-sentence input format requires two special tokens: `[CLS]` at the start of every input, and `[SEP]` between (and after) sentences. The `[CLS]` token's output is used as the sentence-pair representation for NSP — and as we will see in a moment, it is also used as the input to the classification head when we fine-tune for our tasks.

2.5 Tokenisation: WordPiece

BERT does not tokenise on word boundaries the way `quanteda` does. It uses a subword tokeniser called WordPiece. Common words are kept whole, but rare words are split into smaller subword units.

Consider the sentence "The Bundestag passed legislation on globalisation." After WordPiece tokenisation this becomes `[CLS], the, bun, ##des, ##tag, passed, legislation, on, global, ##isation, ., [SEP]`. The `##` marks indicate continuation of a previous word. The rare word "Bundestag" is split into three subwords; "globalisation" is split into two. Common words like "the" and "passed" remain whole.

This subword approach has two advantages. First, it handles unknown words gracefully: any word, even one never seen during training, can be represented as a sequence of subwords (in the worst case, individual characters). Second, it helps the model learn morphology: prefixes, suffixes, and word roots are explicit subword units.

The practical implication for users is that BERT's input has a maximum length of 512 tokens (not words!). Long documents must be truncated or split into chunks. A 400-word parliamentary speech might tokenise to 600+ tokens after WordPiece splitting, exceeding BERT's limit. We will return to this constraint when we discuss the practical session.

2.6 What BERT learns

Empirical analyses (Clark et al. 2019; Rogers et al. 2020; Tenney et al. 2019) have shown that BERT's representations encode a hierarchy of linguistic information across its layers.

Lower layers encode surface features: tokenisation patterns, capitalisation, basic word identity, simple positional information.

Middle layers encode syntactic features: part-of-speech, syntactic dependencies, phrase structure, agreement. Remarkably, BERT learns syntax without ever being shown a parse tree — syntactic information emerges from the pre-training task.

Upper layers encode semantic features: word sense disambiguation, semantic roles, coreference, textual entailment. These are the layers that capture meaning in context.

World knowledge is distributed across many layers. BERT "knows" that Paris is the capital of France, that Margaret Thatcher was Prime Minister of the UK, and many other facts —

though this knowledge is encoded probabilistically, not as explicit facts, and can be incorrect or outdated.

BERT also encodes *biases* from its training data. Studies using WEAT-style methods on BERT have found gender bias, racial bias, religious bias, and political bias. When you fine-tune BERT on your task, these biases come along for the ride. They can be amplified or attenuated by your fine-tuning data, but they do not disappear. For socially sensitive applications, probe your fine-tuned model for differential error rates across groups.

3 Part III — Fine-tuning BERT

3.1 The fine-tuning architecture

To use BERT for a classification task, we add a small "classification head" on top of the pre-trained model. The full architecture has three components.

The pre-trained BERT. All 12 layers (for BERT-base), all 110 million parameters, loaded from the pre-trained checkpoint. These weights have been learned from billions of words of pre-training.

The `[CLS]` token output. After BERT processes the input sequence, each token has a final-layer representation (a 768-dimensional vector for BERT-base). For classification, we use only the representation at the `[CLS]` position — a single vector that, after fine-tuning, summarises the entire input for the classification task.

The classification head. A small randomly-initialised linear layer that maps the 768-dimensional `[CLS]` vector to K output logits (one per class), followed by softmax. For binary classification with $K = 2$, this head has only $768 \times 2 + 2 = 1{,}538$ parameters — a trivial fraction of BERT's 110 million.

During fine-tuning, gradients flow from the classification loss back through the classification head and into BERT's layers. Both BERT's weights and the classification head's weights are updated — though BERT's weights move only slightly (because we use a very small learning rate), while the randomly-initialised head must learn from scratch.

3.2 The fine-tuning procedure

The training loop is essentially the same as in Lecture 5, but two practical details are critical and differ from training-from-scratch.

Start from pre-trained weights. This is the entire point of transfer learning. Use `AutoModelForSequenceClassification.from_pretrained(...)` rather than initialising weights randomly. The classification head is automatically added on top.

Use a very small learning rate. The pre-trained weights are already very good. Large learning-rate updates would destroy that knowledge. The standard fine-tuning learning rate is in the range 2×10^{-5} to 5×10^{-5} — roughly 50 to 100 times smaller than the 10^{-3} you would use for from-scratch training in Lecture 5.

The typical hyperparameter recipe, from the original BERT paper and confirmed by years of subsequent practice, is: learning rate 2×10^{-5}, 3×10^{-5}, or 5×10^{-5}; batch size 16 or 32; 2 to 4 epochs (not 10–50 as in Lecture 5); AdamW optimiser with weight decay around 0.01; and a linear learning-rate warmup over the first 10% of training, followed by linear decay.

Just two to four epochs is enough. BERT learns your task very quickly because most of the work has already been done in pre-training. Training for many more epochs almost always leads

to overfitting — the model memorises the training data instead of generalising.

3.3 To freeze or not to freeze?

A natural question: should we update all of BERT's parameters during fine-tuning, or only the classification head?

Full fine-tuning updates all 110 million parameters of BERT plus the classification head. This is the standard approach and gives the best accuracy in most cases. The downsides are higher compute requirements and a real risk of "catastrophic forgetting" if the training data is very small or noisy.

Frozen BERT plus head only treats BERT as a fixed feature extractor: BERT's weights do not update, only the classification head learns. This is much faster (only the small head's gradients need to be computed) and uses much less memory. It tends to be 2–5 percentage points lower in accuracy than full fine-tuning, but is robust with very small datasets.

Parameter-efficient methods like LoRA and adapters update only a small fraction of parameters — typically less than 1% of BERT's weights — while leaving the rest frozen. This is a popular compromise: most of the speed and memory benefits of frozen BERT, most of the accuracy of full fine-tuning. For most political science applications, you do not need this complexity, but it is good to know it exists.

The default recommendation for this course is full fine-tuning. Use frozen BERT only if you have severe compute constraints or fewer than a few hundred labeled examples.

3.4 Common fine-tuning mistakes

Several mistakes recur in student fine-tuning projects. Knowing them in advance will save you time.

Learning rate too high. A learning rate of 10^{-3}, sensible for from-scratch training, destroys BERT's pre-trained weights and produces a model that performs at chance level. Always use the 10^{-5} range for fine-tuning.

Training for too many epochs. Past 4–5 epochs, BERT typically overfits hard. Validation loss starts climbing while training loss continues to fall. Use early stopping based on validation loss.

Forgetting to hold out a test set. The fine-tuning code reports validation metrics during training. If you choose your model based on these metrics (e.g., picking the best epoch), those metrics are no longer unbiased. You still need a separate test set that you evaluate exactly once at the end.

Mismatched tokeniser and model. Each pre-trained model has its own tokeniser trained jointly with it. The `bert-base-uncased` tokeniser cannot be used with the `roberta-base` model. Mixing them silently corrupts inputs — the model receives token IDs that mean nothing to it. Always load the tokeniser from the same pre-trained checkpoint as the model.

Ignoring the 512-token limit. If your documents exceed 512 tokens after tokenisation, they must be truncated, summarised, or split into chunks. Pretending they fit produces silent errors — the tokeniser truncates them without warning, and you may be classifying only the first 400 words of a 2,000-word speech.

Comparing without baselines. "My BERT got 87% accuracy!" is meaningless without context. You must show that logistic regression on the same train/test split got 84%, 86%, or

88%. Without baselines, your number is uncalibrated. This is the most common mistake in published methods papers.

Not setting a random seed. BERT fine-tuning is stochastic — different random initialisations of the classification head and different orderings of the training data produce different results. Running three to five seeds and reporting mean ± standard deviation is essential. A 1% accuracy difference between two models becomes meaningless if the seed-to-seed variation within each model is also 1%.

4 Part IV — BERT in practice

4.1 When BERT beats logistic regression — honestly

After years of using BERT in political science applications, the empirical picture is clearer than it was in 2018. There are cases where BERT clearly outperforms classical methods, and cases where logistic regression is competitive or even better.

Tasks requiring contextual understanding where word order or local context matters: stance detection, sentiment with negation, irony, sarcasm, complex argumentative structure. Logistic regression with bag-of-words cannot distinguish "not acceptable" from "acceptable"; BERT can.

Short documents where TF-IDF has limited signal: tweets, headlines, brief comments. With only 10–20 words per document, classical methods do not have enough vocabulary overlap with training examples to generalise well. BERT's semantic representations help.

Semantically subtle classes where different categories use overlapping vocabulary. Distinguishing populist rhetoric from mainstream conservative rhetoric, for example, is hard for bag-of-words because both classes mention similar concepts. BERT can pick up subtler stylistic and argumentative differences.

Moderate-sized labeled datasets (1,000 to 10,000 examples). Below this range, BERT can overfit; above it, logistic regression is already strong and gains less from BERT. The sweet spot for BERT is the typical political science labeled dataset size.

By contrast, classical methods remain competitive on **frequency-based tasks** where category membership is largely about which words are present: topic classification, party prediction in long documents, prediction of policy areas. They are competitive on **long documents** (several thousand words) where the rich TF-IDF signal is hard to beat with BERT's 512-token-window approach. They are competitive when labeled data is **very abundant** (tens of thousands of examples) — classical methods saturate slowly with data, and BERT's advantage shrinks at scale. And they remain attractive whenever **interpretability** matters: logistic regression's coefficients are immediately readable, while BERT requires more elaborate interpretation methods.

The single most important practical rule is therefore the same one we have repeated throughout the course: **always run a logistic regression baseline**. If LR gets 85% F1 and BERT gets 86% F1, the gain often does not justify 100× the compute cost. If LR gets 75% F1 and BERT gets 89% F1, the gain is real and worth pursuing. Without the baseline, you cannot know which case you are in.

4.2 Computational requirements: a reality check

What you actually need to fine-tune BERT-base in practice depends on your hardware. On CPU only (a laptop), 5,000 speeches take 1–3 hours per epoch — workable for a final-project run, painful for experimentation. On free Google Colab GPU (T4), the same epoch takes 2–5

minutes — fast enough for development, and strongly recommended. On Colab Pro (V100 or A100), 30–60 seconds per epoch make smooth iteration possible. On a dedicated university GPU, training takes seconds per epoch and is suitable for research-scale work with many runs and large datasets.

The practical recommendation for this course is to develop on Colab's free GPU tier. It is sufficient for everything we do, and it is genuinely free. For larger experiments in your thesis or research projects, university clusters or Colab Pro become useful.

One memory-management tip: GPU memory is usually the binding constraint, not time. If you run out of memory, reduce your batch size (from 32 to 16, or from 16 to 8) and your problem usually goes away. The model trains slightly slower but does not require more memory than you have.

4.3 Validating a fine-tuned BERT model

The validation principles from Lecture 2 apply — with additional care because the model is more complex and the failure modes are subtler.

Hold out a real test set. Split your data at the start of the project. Do not touch the test set during model development. Evaluate exactly once, at the end. This is the most violated rule in applied machine learning.

Report all standard metrics. Accuracy alone is insufficient. Report precision, recall, F1, and the confusion matrix. For multi-class tasks, report per-class metrics.

Run multiple seeds. BERT fine-tuning is stochastic. Report mean and standard deviation over at least three seeds. If seed-to-seed variation is similar in magnitude to the difference between your models, you cannot claim one is better than the other.

Compare against baselines. Always include logistic regression with TF-IDF. Optionally include a from-scratch neural network. Without these, your BERT result has no context.

Inspect the errors. Look at the documents BERT misclassifies. Is there a pattern? Length, language, ambiguity, specific topics? Error analysis is often as informative as accuracy metrics.

Probe for bias. If your task is socially sensitive (sentiment toward specific groups, hate speech, content moderation), check whether the model's error rates differ systematically across groups. A model with 90% overall accuracy that has 80% accuracy for one demographic group and 95% for another is failing on an important dimension.

5 Part V — Beyond BERT

5.1 BERT variants

BERT was the first widely-used encoder-only Transformer for NLP. Many improvements followed, and several are worth knowing about for your future research.

RoBERTa (Facebook, 2019) uses the same architecture as BERT but is trained on more data, for longer, with more carefully tuned hyperparameters. NSP is removed. RoBERTa is typically 1–2 percentage points better than BERT on standard benchmarks. It is the practical default starting point for English fine-tuning today.

DistilBERT (Hugging Face, 2019) is a smaller, faster version of BERT distilled from a teacher model. It has roughly 66 million parameters (60% of BERT-base) and achieves about 97% of BERT's accuracy at about 40% of the compute cost. Useful when speed or memory matters.

DeBERTa (Microsoft, 2020) introduces a more sophisticated attention mechanism called "disentangled attention," separating content and position information in attention scores. DeBERTa-v3 is the strongest single-model performer on most NLP benchmarks today. Slightly more expensive than BERT, but worth it if you need maximum accuracy.

ELECTRA uses a different self-supervised pre-training objective: replaced-token detection instead of masked language modelling. Pre-training is more compute-efficient; fine-tuning performance is comparable to BERT.

ALBERT is a parameter-sharing variant of BERT that achieves similar performance with a much smaller memory footprint, useful for very long sequences.

The practical recommendation: start with `roberta-base` for English political-text tasks. Use `deberta-v3-base` or `deberta-v3-large` if you need state-of-the-art accuracy and have the compute.

5.2 Multilingual models

For non-English text or cross-country research designs, three options dominate.

Multilingual BERT (mBERT) is BERT trained on 104 Wikipedia languages simultaneously. Surprisingly, the model can transfer across languages: you can fine-tune on English-labeled data and apply the model directly to German or French text. Cross-lingual transfer is imperfect but often workable.

XLM-RoBERTa is a multilingual version of RoBERTa trained on 100 languages with much more data than mBERT. It is consistently better than mBERT on cross-lingual tasks and is the default multilingual choice today.

Language-specific BERTs are pre-trained on a single language using language-specific data: GermanBERT, FlauBERT (French), AraBERT (Arabic), BETO (Spanish), and many others. For monolingual analysis where you do not need cross-lingual transfer, a language-specific BERT usually outperforms a multilingual one.

A representative research design illustrates the power of this family: fine-tune XLM-R on English UK parliamentary speeches with labeled stance, then apply the fine-tuned model directly to French National Assembly debates without any French labels. The model transfers its understanding of stance across languages. This is genuinely transformative for European political science: it makes cross-national text analysis far cheaper than manual translation, with reasonable accuracy.

5.3 NLI-based classifiers: less labelling, more classifying

A recent line of work by Laurer, van Atteveldt, Casas, and Welbers (2024, *Political Analysis*) makes a compelling case for an alternative approach: **NLI-based classifiers**.

The idea is to reframe classification as Natural Language Inference (NLI), a task that pre-trained models are already trained for. NLI models judge whether a hypothesis is *entailed* by, *contradicted* by, or *neutral* with respect to a premise.

Consider a small example. The *premise* is your document: "The minimum wage increase will burden small businesses and harm employment." The *hypothesis* is your class definition: "This text expresses a conservative economic view." The NLI model's job is to decide whether the hypothesis is entailed by, contradicted by, or neutral with respect to the premise. If the NLI model says "entails" with high confidence, you classify the document as "conservative."

The advantage of this approach is that you can change the hypothesis without retraining the model. You define your classes by describing them in natural language. This allows zero-shot classification (no labeled examples) and few-shot classification (a handful of labeled examples), both of which are essentially impossible with standard fine-tuning.

Laurer et al. show that NLI-based classifiers often match the accuracy of fully fine-tuned BERT with far fewer labeled examples. This is a major practical win for political scientists, who typically have small labeled datasets and large unlabeled corpora.

5.4 Large language models for classification

The most recent development is using large language models (GPT-4, Claude, Gemini, LLaMA-3, etc.) directly for classification via prompting. The model is given an instruction in natural language and produces a classification without any fine-tuning.

A typical zero-shot prompt might read: "Classify the following UK parliamentary speech as expressing a Government or Opposition perspective. Speech: [text]. Answer in one word: Government or Opposition." The LLM produces an answer directly. No fine-tuning needed, no labeled training data required.

This is genuinely powerful for some applications: conceptual tasks where the LLM "understands" what you are asking, exploratory research, pilot studies where you do not yet have labeled data. The accuracy can be surprisingly high.

But there are real cautions. API costs add up at scale: classifying a million documents at $0.01 per call costs $10,000. Outputs are non-deterministic, making reproducibility harder. The model can hallucinate confidently, producing wrong labels with high apparent confidence. Privacy and GDPR concerns arise when sending political text to commercial APIs. And validation against hand-coded labels remains essential — LLMs do not eliminate the need for empirical validation, even if they reduce the need for labeled training data.

The best practice emerging from the literature is to use LLMs as a fast first pass for exploratory work, validate against hand-coded data, and fine-tune BERT (or use NLI-based classifiers) for production research that needs precision and reproducibility.

5.5 Recent political science applications

Several recent studies illustrate the state of the art in political science applications of BERT-family models.

Widmann and Wich (2022) in *Political Analysis* fine-tune BERT on German political text to detect discrete emotions. They show that BERT clearly outperforms dictionary methods and classical embeddings, especially for emotions like anger and fear that depend on context.

Bestvater and Monroe (2023) in *Political Analysis* argue that off-the-shelf sentiment classifiers conflate sentiment with stance — two distinct constructs. They call for target-aware opinion classification using fine-tuned transformers.

Laurer, van Atteveldt, Casas, and Welbers (2024) introduce NLI-based classifiers for political text and show strong results with very small labeled datasets.

Licht (2023) uses multilingual transformers for cross-lingual ideological scaling, demonstrating that supervised methods can transfer across European languages.

A common theme runs through all of this work: BERT-family models offer real gains over classical methods for nuanced tasks, but only when researchers validate carefully and report

honestly. The methodological principles from Lecture 1 (Grimmer and Stewart) apply with full force in the BERT era. Better methods do not relax the need for good methodology.

6 Part VI — The practical session

The practical session for this lecture is the culmination of the course. We fine-tune a BERT model on the same Government vs. Opposition classification task we used in Lecture 2, then compare three approaches: logistic regression with TF-IDF (Lecture 2), a simple neural network with BiLSTM (Lecture 5), and fine-tuned BERT (today).

The complete code is in `Lecture7_Practical.py`, designed to run on Google Colab's free GPU tier. The workflow has four key components. First, **data preparation**: load the same UK Commons speeches as in Lectures 1, 2, and 5, convert to a Hugging Face `Dataset` object, and tokenise with the BERT tokeniser using truncation to 512 tokens and padding. Second, **model setup**: load `bert-base-uncased` via `AutoModelForSequenceClassification.from_pretrained(...)`, which automatically adds a classification head with the right number of output classes. Third, **training**: use the Hugging Face `Trainer` API with the standard hyperparameters — learning rate 2×10^{-5}, three epochs, batch size 16, weight decay 0.01, evaluation at the end of each epoch, and "load best model at end" enabled. Fourth, **evaluation**: final F1, accuracy, precision, recall, and confusion matrix on the held-out test set, plus a comparison table with the Lecture 2 and Lecture 5 results.

Expected outcomes: BERT typically lands 2–5 percentage points above the baselines on this task. The gain may be modest because Government vs. Opposition has rich word-level signal that TF-IDF captures well. For a more challenging task — say, fine-grained ideology scaling or stance detection — BERT's advantage would typically be larger.

7 Course wrap-up

7.1 What you have learned

Over seven lectures, you have built the complete toolkit of modern computational text analysis for political science. You can now preprocess any political text corpus and justify every choice (Lecture 1). You can train and properly evaluate classical classifiers, interpret their weights, and avoid common mistakes (Lecture 2). You can use word embeddings to measure semantic similarity, detect biases, and study political vocabulary geometrically (Lecture 3). You can discover latent themes with topic models and estimate covariate effects with STM (Lecture 4). You can build neural networks for text, understand RNNs and LSTMs, and recognise their limits (Lecture 5). You understand the Transformer architecture from the ground up, including self-attention, multi-head attention, and the encoder block (Lecture 6). And you can fine-tune pre-trained models like BERT on political classification tasks and compare them rigorously against simpler baselines (Lecture 7).

More importantly than any particular technique, you have learned the methodological principles that apply across all these methods. You know that preprocessing is a modelling decision, not a neutral preparation step. You know that classical methods are strong baselines that should always be tried first. You know that validation is essential and that accuracy alone is insufficient. You know that transparency about choices and limitations matters more than impressive headline numbers.

You can now read and replicate state-of-the-art political science research that uses text data. You can apply these methods responsibly to your own research questions. You can recognise methodological problems in others' work and propose better designs.

7.2 The five principles to carry forward

The techniques in this course will evolve. BERT will be replaced; new architectures will emerge; new self-supervised objectives will improve on MLM. The half-life of specific methods in NLP is measured in years, sometimes months. But the methodological principles below are durable. They will still be relevant in ten years.

Always try the classical methods first. Logistic regression with TF-IDF is an honest, well-understood, computationally cheap baseline. If it solves your problem, you do not need BERT. The instinct to reach for the most complex tool is widespread and usually wrong.

Preprocessing is a modelling decision. Document every choice. Show robustness to alternative preprocessing pipelines. Two researchers running the same analysis with different stop word lists or different stemming choices may reach different substantive conclusions.

Validate everything. Hand-code a test set at the beginning of your project and do not touch it until the end. Read the documents your model misclassifies. Report confidence intervals and seed-to-seed variation. Run multiple preprocessing pipelines.

Computational methods augment humans, they do not replace them. You still need substantive expertise to ask good questions and interpret answers. A 95% accurate classifier is useless if it is solving the wrong problem. Methodological sophistication without substantive judgment produces accurate answers to badly-framed questions.

Be honest about limitations. A clear, honest analysis with modest claims is better than an inflated one with hidden caveats. Reviewers will respect you for it. Future readers — including future you — will trust your work. The most important sentence in any methods section is usually the limitations paragraph.

7.3 What comes next

This course gives you the foundations. The path forward depends on your research interests, but several directions are worth pursuing.

For your immediate research, the best next step is to apply these methods in your thesis or final project. Replicating a published computational text analysis paper is an excellent learning exercise — you will encounter exactly the practical issues this course taught you to handle.

For topics this course did not cover but you may want to learn, the most important are: generative LLMs in depth (prompt engineering, retrieval-augmented generation, evaluation of LLM outputs); causal inference with text (text as treatment, text as outcome, sensitivity analysis); advanced topic models (dynamic topic models, hierarchical models); and network methods on text data (citation networks, co-mention graphs, dynamic networks of political actors).

For staying current, the most relevant journals are *Political Analysis* (the methodological flagship), *Political Communication* (text-heavy applications), *Journal of Politics* and *American Journal of Political Science* for applications, and the proceedings of the NLP+CSS workshop series for the latest computational social science methods.

The field is moving fast. The specific methods you learn next year will partly replace what you learned this year. But the principles from this course — careful preprocessing, honest validation, classical baselines, substantive interpretation, transparency about limitations — will not. They are the methodology. The techniques are the implementation.

Key takeaways

1. **The pre-train / fine-tune paradigm** is the foundation of modern NLP. Large models learn language once from billions of unlabeled words; researchers specialise them cheaply on small labeled datasets.
2. **Masked Language Modelling** is a self-supervised pre-training task: hide some words, predict them from context. It scales to any amount of available text because no human labels are required.
3. **BERT's representations** encode a hierarchy of linguistic information: surface features in lower layers, syntax in the middle, semantics on top. They also encode biases from the training data.
4. **Fine-tuning BERT** requires very small learning rates ($\sim 10^{-5}$), few epochs (2–4), and careful comparison against classical baselines. The hyperparameters differ sharply from those used for from-scratch training.
5. **BERT wins** on context-sensitive tasks, short documents, semantically subtle classes, and moderate-sized labeled datasets. Classical methods remain competitive on frequency-based tasks, long documents, very large labeled datasets, and tasks that require interpretability.
6. **Always run a logistic regression baseline.** Without it, a BERT accuracy number is uncalibrated.
7. **Beyond plain BERT**: RoBERTa, DeBERTa, DistilBERT, multilingual variants, NLI-based classifiers, and large language models. Each has a niche; none replaces the discipline of validation and baselines.
8. **The methodological principles from Lecture 1 apply with full force.** Preprocessing as a modelling decision, classical baselines, careful validation, substantive interpretation, transparency about limitations. Better methods do not relax the need for good methodology.

Required reading

- Jurafsky, D. & Martin, J.H. (2025). *Speech and Language Processing,* Chapter 10: Masked Language Models. `https://web.stanford.edu/~jurafsky/slp3/10.pdf`

Optional reading

- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." *NAACL 2019*. `https://arxiv.org/abs/1810.04805` (The original BERT paper. Read sections 1–3 carefully; the rest is empirical benchmarks.)
- Hugging Face NLP Course, Chapter 3: Fine-tuning a pretrained model. `https://huggingface.co/learn/nlp-course/chapter3` (The best hands-on introduction to fine-tuning BERT with the `transformers` library.)

Other relevant reading (political science applications)

- Widmann, T. & Wich, M. (2022). "Creating and Comparing Dictionary, Word Embedding, and Transformer-Based Models to Measure Discrete Emotions in German Political Text."

Political Analysis.

- Laurer, M., van Atteveldt, W., Casas, A., & Welbers, K. (2024). "Less Annotating, More Classifying: Addressing the Data Scarcity Issue of Supervised Machine Learning with Deep Transfer Learning and BERT-NLI." *Political Analysis*. (The NLI-classifier paper. Essential reading for anyone planning to use BERT-family models in research.)
- Bestvater, S.E. & Monroe, B.L. (2023). "Sentiment Is Not Stance: Target-Aware Opinion Classification for Political Text Analysis." *Political Analysis*. (A careful analysis of when off-the-shelf classifiers fail and why task-specific fine-tuning matters.)
- Rogers, A., Kovaleva, O., & Rumshisky, A. (2020). "A Primer in BERTology: What We Know About How BERT Works." *Transactions of the ACL*. (Comprehensive review of BERT analysis research. Reference work, not for cover-to-cover reading.)