Advanced NLP Exercise 2

Gal Cesana – 318510633

1. Open Questions

1.1 Additional Source of Societal Bias

One prominent source of societal bias, aside from gender, is racial bias. In particular, Gonen and Goldberg (2019) propose a *multiclass debiasing* technique for word embeddings that aims to remove biases corresponding to multiple protected attributes such as race and religion simultaneously. Their method, described in "Detecting and Removing Multiclass Bias in Word Embeddings" (Gonen and Goldberg, 2019), operates as follows:

- **High-level Description of the Technique:**
  - The authors begin by constructing sets of *definitional* words for each demographic category (e.g., lists of words that identify different racial or religious groups).
  - They compute, for each protected class C, a *bias subspace* S_C by taking the principal components of the embeddings of the definitional words for that class. This subspace captures the directions in the embedding space along which that particular class tends to align.
  - To remove multiclass bias, they project each word embedding $\mathbf{w}$ onto the orthogonal complement of the union of all per-class bias subspaces. If $\{\mathbf{u}_{C,1}, \ldots, \mathbf{u}_{C,k_C}\}$ is an orthonormal basis for the bias subspace S_C, and $\mathcal{B} = \bigcup_C \{\mathbf{u}_{C,i}\}$ is the union of all basis vectors across classes C, then the "debiased'' embedding $\mathbf{w}'$ is obtained by

$$\mathbf{w}' \;=\; \mathbf{w} \;-\; \sum_{\mathbf{u}\in\mathcal{B}} (\mathbf{w}\cdot\mathbf{u})\,\mathbf{u}\,.$$

  - After this projection, the embeddings no longer encode linear directions that correlate strongly with any single protected attribute in $\{$ race, religion, $\ldots\}$.
- **Resources:**
  - The technique operates on off-the-shelf pre-trained embeddings (e.g., GloVe or Word2Vec) and does *not* require retraining from scratch. Instead, it applies a *post-processing* step on each embedding vector.
  - The computational cost is dominated by (a) computing principal components for each protected attribute's definitional set (which is a one-time cost), and (b) projecting every word in the vocabulary onto the orthogonal complement of the combined bias subspace.
- **Trade-offs:**
  - *Saved Resources at Inference:* Because the projection can be computed once and stored, there is no additional inference-time overhead beyond using the debiased embeddings in downstream tasks. Thus, memory usage and runtime at inference remain essentially the same as with the original embeddings.
  - *Expense in Other Resources:*
    - There is an *up-front* cost in computing the PCA for each protected attribute's definitional set and then applying the projection to all word vectors. This requires $O(d \times |V| \times m)$ operations, where d is the embedding dimension, $|V|$ is the vocabulary size, and m is the total number of basis vectors across all demographic classes.
    - There is no additional training required, but there is a nonzero cost in time and compute to generate the debiased embeddings before they can be used.
    - Downstream tasks retain their original performance (or in some cases, suffer a minor drop), meaning there is no explicit trade-off at inference time beyond the preprocessing cost.

1.2 Three Efficiency Methods (One per Category)

Below, we select three efficiency techniques, each belonging to a different category among the four efficiency categories (Training, Inference, Modeling, Data). For each method, we provide (a) a brief description, (b)

the type of resource(s) it aims to save, and (c) whether the savings come at the expense of other resources. Finally, we discuss pairwise combinations or mutual exclusivity.

Method 1: Knowledge Distillation (Training Category)

(a) Description. Knowledge distillation (Hinton *et al.*, 2015) is a training-time technique in which a large, *teacher* model is used to supervise the training of a smaller, *student* model. During training, the student is trained not only on the original "hard" labels but also on the "soft" output distribution (e.g., logits or probabilities) produced by the teacher model. The loss function typically combines a cross-entropy to the ground-truth labels with a "distillation loss" that encourages the student to match the teacher's output distribution (often with a temperature parameter $T > 1$).

(b) Resources Saved.

- **Inference Compute:** The resulting student model is much smaller (fewer layers or parameters) and therefore requires significantly less compute (FLOPs) when making predictions.
- **Inference Memory Footprint:** The student's weights occupy less memory (fewer parameters) in GPU/CPU RAM.
- **Runtime Latency:** Since the student has fewer layers or narrower layers, inference can be substantially faster (lower latency).

(c) Trade-offs.

- *Training-time Overhead:* One must first train (or have a pretrained) teacher model. Then the student model must be trained with the additional distillation loss. This incurs extra training compute/time beyond training a single model from scratch.
- *Possible Accuracy Gap:* In some cases, the student cannot perfectly match the teacher's performance; thus, there is a small drop in accuracy or quality.
- *Hyperparameter Tuning:* One must tune the distillation temperature T and the weighting between hard-label loss and distillation loss, which adds complexity to training.

Method 2: Quantization (Inference Category)

(a) Description. Quantization is an inference-time (or post-training) method where the precision of the weights (and sometimes activations) in a neural network is reduced. For example, one can convert 32-bit floating-point weights to 8-bit integers. The network is then executed using integer arithmetic (or mixed-precision arithmetic), which modern hardware often accelerates. There are two common variants:

- **Post-Training Quantization (PTQ):** After a model is fully trained in float32, a calibration dataset is used to determine scale and zero-point values to map floating weights/activations into 8-bit.
- **Quantization-Aware Training (QAT):** During training, fake-quantization operations simulate lower precision, and gradients flow through so that the model parameters adjust to be robust to quantization noise.

(b) Resources Saved.

- **Memory Footprint:** Reducing weights from 32-bit floats to 8-bit integers cuts the model size by up to 4×. Activations in intermediate layers can also be quantized, further lowering memory needed for activations during inference.
- **Compute Efficiency:** Integer (INT8) arithmetic is often much faster and more energy-efficient than floating-point (FP32) on specialized hardware (e.g., GPUs or NPUs with INT8 support). This yields lower latency and lower power consumption.

(c) Trade-offs.

- *Accuracy Degradation:* Especially with *post-training quantization* (PTQ), there can be a drop in accuracy, particularly if the model is sensitive to small perturbations in weights or activations.
- *Calibration Effort:* PTQ requires a representative calibration dataset to determine quantization ranges; QAT requires retraining or fine-tuning under fake-quantization, which is additional compute/time.

- *Hardware Requirements:* To realize the speedup, one needs hardware with efficient INT8 (or lower) support. On generic hardware without optimized integer kernels, quantization might not yield any runtime improvement.

Method 3: Structured Pruning (Modeling Category)

(a) Description. Structured pruning is a model-centric method where entire structures of the network—such as channels, heads in multi-head attention, or even whole layers—are removed. One common approach is:

- During training (or post-training), introduce a *scaling factor* α_i for each structural component (e.g., each convolutional filter or attention head).
- Add a sparsity-inducing regularizer (e.g., ℓ_1-norm on α_i) to encourage some $\alpha_i \to 0$.
- Once trained, remove (prune) all components whose scale α_i is below a threshold, resulting in a smaller network with fewer channels or heads.

After pruning, the model can be fine-tuned to recover accuracy.

(b) Resources Saved.

- **Inference Compute:** By removing entire channels or layers, the remaining network requires fewer floating-point operations.
- **Inference Memory:** Fewer layers or narrower layers translate to fewer parameters stored in memory.
- **Runtime Latency:** Since some layers (or parts of layers) are entirely removed, the forward pass is shorter/faster.

(c) Trade-offs.

- *Retraining / Fine-tuning Overhead:* One must either (a) train with pruning in mind (adding sparsity penalties) or (b) perform pruning after a standard training run and then fine-tune. Both options incur extra compute/time beyond training the dense baseline once.
- *Possible Accuracy Loss:* If too many components are pruned, the model's representational capacity shrinks, which can degrade accuracy. Finding the right balance between sparsity and performance is nontrivial.
- *Implementation Complexity:* Structured pruning often requires custom implementations (e.g., to remove specific channels or heads) and careful handling of remaining weights.

Pairwise Combination Analysis

Let the methods be:
(A) Knowledge Distillation (KD)
(B) Quantization (Q)
(C) Structured Pruning (SP)

A & B (Distillation & Quantization). These two are *compatible* and often used together in practice:

- After training a small *student* via distillation (method A), one can further apply post-training quantization or quantization-aware training on that student model.
- Distillation *reduces* the model size by producing a smaller architecture, while quantization *reduces* each parameter's precision. The combination yields multiplicative savings in both memory footprint and compute (e.g., a distilled model with 10 M parameters can be quantized from FP32 to INT8, resulting in an 80 MB $\to$ 10 MB size reduction if each FP32 weight is 4 bytes).
- The *only* additional overhead is that one must run calibration (for PTQ) or fine-tuning with fake-quantization (for QAT) on the student. However, this is relatively small compared to the original teacher training, and the runtime/inference savings are compounded.

A & C (Distillation & Pruning). These are likewise *compatible*:

- One approach is to first prune the teacher model (structured or unstructured) to a smaller "slim" version and then use that pruned teacher to distill a student. Alternatively, one may first distill a student and then apply pruning to the student model itself.
- Pruning reduces the width or depth of layers (or number of heads, etc.), and distillation trains a compact model that mimics a larger teacher's behavior. Combining them often yields a student that is both structurally lean and well-trained.
- The overhead consists of an extra pruning step (possibly with its own fine-tuning) *and* the distillation training. Nonetheless, inference savings (fewer FLOPs, fewer parameters) are greater than using either technique alone.

B & C (Quantization & Pruning). These two also *can* be combined:

- Structured pruning first removes entire channels/blocks, producing a narrower network. One then applies quantization to the pruned model. Since the pruned model is already smaller, quantizing it yields an even smaller footprint.
- Alternatively, one could quantize first and then prune (with quantization-aware pruning methods), but in practice it is more common to prune a floating-point model, fine-tune, and then quantize the final pruned model.
- The trade-off is that each step introduces some accuracy degradation: pruning can remove important weights, and quantization adds rounding errors. Therefore, careful calibration and fine-tuning are required at each stage.

Summary. All three pairs $(A, B), (A, C), (B, C)$ are *compatible* and frequently used in combination to achieve additive gains in inference efficiency. There is no mutual exclusivity among them; rather, practitioners often build a pipeline such as:

$$\text{Train large teacher} \xrightarrow{\text{pruning}} \text{pruned teacher} \xrightarrow{\text{distillation}} \text{small student} \xrightarrow{\text{quantization}} \text{INT8 student.}$$

Each stage trades extra training or calibration compute for greater inference-time savings.

2. Practical Exercise

Why the Model Is Wrong: Hypothesis and Evidence

Provided Inputs and Model Outputs (Meta-Llama-3-8B-Instruc)

Index	Input	Model output
1	I REALLY loved your song, it sounded like a frog being ran over by a truck	Positive
2	There is no way I'm coming back there, who would want to have all of his wishes come true in one place?	Negative
3	I like it here, It's quiet like a cemetery, everyone would like a vacation like this	Positive
4	The food is amazing, it only made me puke once!	Positive
5	Fast service, I practiced 30 minutes meditation between each course	Positive

Table 1: Original inputs and the Meta-Llama-3-8B-Instruct's (incorrect) sentiment predictions.

Hypothesis

All of the provided examples involve *sarcasm* or *ironic contrast*, in which an ostensibly positive phrase is immediately followed by negative or absurd content. As a result, the model latches onto the strong positive words (e.g., "loved," "amazing," "fast service") and overlooks the contextual cue that the overall sentiment is negative or mixed. In other words:

- The model over-weights surface-level positive adjectives or adverbs.
- It under-weights the pragmatic context that signals sarcasm or irony (e.g., a gruesome comparison, a contradiction, or an absurd follow-up).
- Consequently, sentences that appear superficially positive are misclassified as positive, despite their true negative or sarcastic intent.

Analysis of Each Example

1. **Index 1.**

   "I REALLY loved your song, it sounded like a frog being ran over by a truck"

   - Surface positive cue: "loved your song"
   - Sarcastic context: comparison to a frog being run over by a truck (clearly negative).
   - Model's output: *Positive*, despite negation of any genuine praise.

2. **Index 2.**

   "There is no way I'm coming back there, who would want to have all of his wishes come true in one place?"

   - Surface negative cue: "There is no way I'm coming back there" reads like a negative statement.
   - Sarcastic context: "who would want to have all of his wishes come true in one place?" For a human being its easy to understand that everyone wants it.
   - Model's output: *Negative*, it didn't get the turn around point that the speaker seemed to have good time.

3. **Index 3.**

   "I like it here, It's quiet like a cemetery, everyone would like a vacation like this"

   - Surface positive cue: "I like it here"
   - Sarcastic context: "quiet like a cemetery" + "everyone would like a vacation like this" implies that the place is boring or creepy.
   - Model's output: *Positive*, failing to register the negative connotation of a cemetery-quiet vacation.

4. **Index 4.**

   "The food is amazing, it only made me puke once!"

   - Surface positive cue: "The food is amazing"
   - Sarcastic context: "it only made me puke once!" indicates disgust and a negative experience.
   - Model's output: *Positive*, ignoring the follow-up that clearly reverses any genuine praise.

5. **Index 5.**

   "Fast service, I practiced 30 minutes meditation between each course"

   - Surface positive cue: "Fast service"
   - Sarcastic context: "practiced 30 minutes meditation between each course" implies that service was so slow that the speaker had time to meditate.
   - Model's output: *Positive*, despite the actual negative meaning.

Extent of the Problem

- The five examples, share the same pattern: an apparently positive/negative clause followed by an obviously opposite sentiment clause, which together form a sarcastic statement.
- In all of those sarcastic examples, the model outputs, demonstrating that it is heavily influenced by the initial positive wording and fails to integrate the contradictory context.
- Thus, we see that whenever a strong sentiment adjective or phrase appears up front and is immediately contradicted by an opposite sentiment, the model almost invariably predicts the first sentiment.

Conclusion

The evidence from the provided inputs supports the hypothesis that Meta-Llama-3-8B-Instruct (a) attends primarily to the initial positive sentiment words, and (b) fails to incorporate the later contextual cues indicating sarcasm or negative intent. As a result:

- Any sentence which superficially contains a strong positive sentiment word but is actually negative (due to sarcasm or irony) is likely to be misclassified as positive.
- More sophisticated handling of pragmatics (e.g., modeling discourse cues, detecting incongruity between clauses) would be required to correct this failure mode.

Trying another model - DeepSeek-R1-Distill-Qwen-7B

Provided Inputs and Model Outputs

Index	Input	Model output
1	I REALLY loved your song, it sounded like a frog being ran over by a truck	Positive
2	There is no way I'm coming back there, who would want to have all of his wishes come true in one place?	Negative
3	I like it here, It's quiet like a cemetery, everyone would like a vacation like this	Negative (Correct!)
4	The food is amazing, it only made me puke once	Positive
5	Fast service, I practiced 30 minutes meditation between each course	Positive

Table 2: Predictions of **DeepSeek-R1-Distill-Qwen-7B**. Note: only index 3 is correct (negative sentiment), the others remain misclassified.

Proposed Solution

Goal. Reduce the systematic failure on *sarcastic or irony–laden* inputs without relying on costly manual annotation.

High-level idea. Combine *self-supervised data mining* with a lightweight *architecture add-on* that explicitly checks intra-sentence sentiment consistency. The two components are independent—either can be adopted alone—but together they yield a robust, data-efficient remedy.

1. Self-supervised data mining via rating–text mismatch

a) **Observation.** Large review sites (Amazon, Yelp, TripAdvisor, IMDb, …) provide an *explicit* sentiment label in their metadata: the *star rating*. A review with `5 stars` but containing strongly negative lexical markers—or the reverse—most likely expresses *sarcasm, irony, or mixed sentiment*.

b) **Heuristic filter.** Crawl millions of reviews and retain only those that satisfy

$$(\text{rating} \geq 4 \wedge \text{neg-lexicon present}) \ \vee \ (\text{rating} \leq 2 \wedge \text{pos-lexicon present}).$$

Lexicons can be off-the-shelf sentiment word lists; no human labeling is needed.

c) **Pseudo-labeling.** Tag the retained sentences with a binary flag *"incongruent-sentiment"*. Use them as *auxiliary training data* for a new **Consistency Head** (see 2.) while keeping the original sentiment labels untouched.

2. Architectural add-on: a *Sentiment Consistency Head* Let the original encoder (e.g., Qwen-7B) output a sequence of token embeddings $\mathbf{H} \in R^{T \times d}$.

a) **Clause segmentation.** Use a fast rule-based splitter (punctuation, coordinating conjunctions) or a lightweight parser to extract k clause spans $C_1, \ldots, C_k$.

b) **Clause-level sentiment logits.** For each C_i compute a pooled embedding $\mathbf{c}_i = \text{mean}(\mathbf{H}_{C_i})$ and pass it through a small MLP yielding a scalar sentiment logit $s_i \in [-1, 1]$.

c) **Consistency score.** Define

$$\sigma \ = \ 1 - \tanh\Big(\frac{1}{k(k-1)} \sum_{i<j} |\, s_i - s_j \,|\Big) \in [0, 1],$$

where $\sigma \approx 1$ means *uniform sentiment* across clauses, and $\sigma \to 0$ signals *strong inconsistency.*

d) **Gated sentiment prediction.** Combine the model's original sentence-level logit S with a learned flip-function $f(\sigma)$:

$$S_{\text{final}} \ = \ \underbrace{\sigma}_{\text{confidence}} \cdot S \ + \ (1 - \sigma) \cdot f(S),$$

where $f(S)$ is simply $-S$ (i.e. flip sign) or a tiny 2-layer MLP trained jointly. When the sentence shows large internal disagreement ($\sigma \approx 0$), the gate shifts weight toward the flipped sentiment.

e) **Training objective.** Add two auxiliary losses:

$$\mathcal{L}_{\text{flip}} = \big|\, \hat{\sigma} - \sigma \big| \quad \text{(self-supervised from 1.)}$$
$$\mathcal{L}_{\text{sent}} = \text{BCE}\big(S_{\text{final}},\, y\big),$$

where $\hat{\sigma}$ is the pseudo-label (*incongruent* $\hat{\sigma} = 0$; else 1) and $y \in \{\text{neg}, \text{pos}\}$ is the ground-truth sentiment.

Why it helps.

- The heuristic corpus teaches the model to recognize *intra-sentence polarity clashes* without manual labels.
- The Consistency Head is a *drop-in module*—two small MLPs and a few arithmetic ops—so inference cost grows negligibly.
- At test time the system still produces a single sentiment label but now down-weights obviously sarcastic positives (or negatives) by detecting internal contradiction.