Firstly, what we introduce:

1. unidirectional PLM, AKA auto-regression model
2. bidirectional PLM, AKA auto-encoding model
3. few-shot
4. prompting method in NLP
5. zero-shot

Secondly, we could talk about what they did:

1. auto-regression model  generates class-conditioned training dataset.
2. auto-encoding model fine-tune the model based on generated prompting.

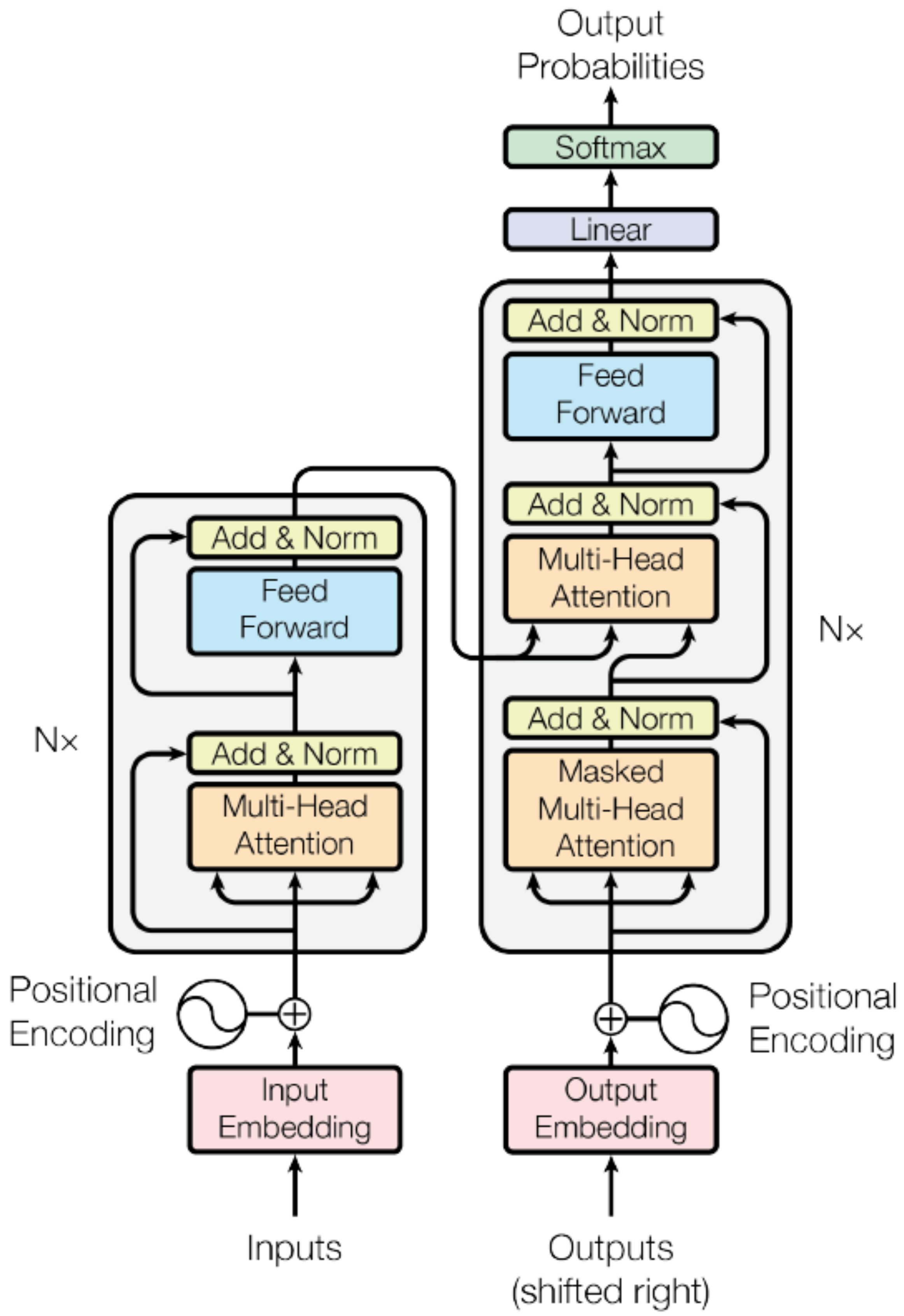
Thirdly, we could compare SuperGen to other model which uses zero-shot:

1. moderate the model size
2. no need annotation. Existing zeros-shot methods do not use any form of training data from the test domain (but may train on other domains) and directly perform inference on the target task.

Lastly, discuss the keys for working well:

1. selecting quality training dataset. (details in paper)
2. using label smoothing and temporal ensemabling to regularize fine-tuning on generated data.

## First part -- prerequisites


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

## 1.1 autoencoding vs autoregressive

Transformers consists of Encoder and Decoder

Encoder: AKA bidirectional or auto-encoding

Decoder: AKA unidirectional or auto-regressive

Why:

1. in the transformers encoder, the process of handling the previous token and following token are simultaneous. Every token could read the information from start to end of the input and the information from end to the start of the input. Its function looks like bi-LSTM. It is good at NLI task. Since it could tackle the information from both side.
2. in the transformers decoder, the subsequent token must wait until the previous ones have been processed. It's good at Text Generation. Because people write anything from left to right.

auto-encoding model: BERT, RoBerta, AlBert, XLNET.

auto-regressive model: GPT, Transformer-XL.

hybrid model: BART, T5, UniLM

## 1.2 few-shot learning

https://www.borealisai.com/research-blogs/tutorial-2-few-shot-learning-and-meta-learning-i/

Few-shot learning refers to the ability of learning new concepts by training machine learning models with only a few examples. Few-shot learning is usually studied using *N-way-K-shot classification*.

in the below figure, we use one set of classification problems to help solve other unrelated sets.

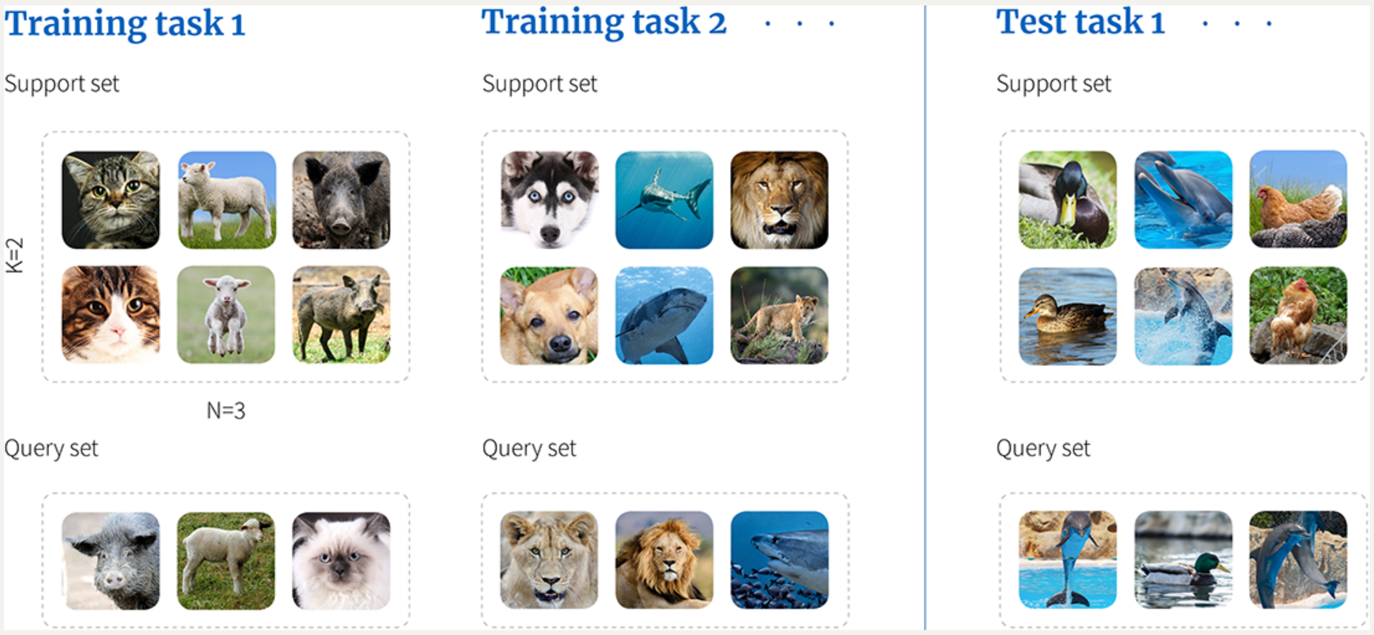


Figure 1. An algorithm is trained using a series of training tasks. Here, each task is a 3-way-2-shot classification problem because each training task contains a support set with three different classes and two examples of each. During training the cost function assesses performance on the query set for each task in turn given the respective support set. At test time, we use a completely different set of tasks, and evaluate performance on the query set, given the support set. Note that there is no overlap between the classes in the two training tasks {cat, lamb, pig}, {dog, shark, lion} and between those in the test task {duck, dolphin, hen}, so the algorithm must learn to classify image classes in general rather than any particular set.

## IN NLP

The most strict few-shot learning setting does not assume access to any unlabeled data or large validation sets for hyperparameter tuning, where prompt-based methods are prominently deployed to inject task descriptions into PLMs and make effective use of their language modeling capability for improved training data efficiency in low-data regimes.

## 1.3 prompting

A prompt refers to a text input that is given to a model to elicit a specific response or output. The prompt acts as a guide, instructing the model on what type of information or response is required.

Imagine such a scenario that you train a model for the NER task today, you will be asked to do a text generation task tomorrow, what's more, you will be asked to do a QA bot, and so on. Every day all you need to do is training different models again and again.

Prompt could be considered as an auxiliary sentence which help generalize model to have different ability.
for example:

1. Please rate this movie. + input
2. Please answer the following question. + input

after training, the model will know which task it will tackle based on the prompt.

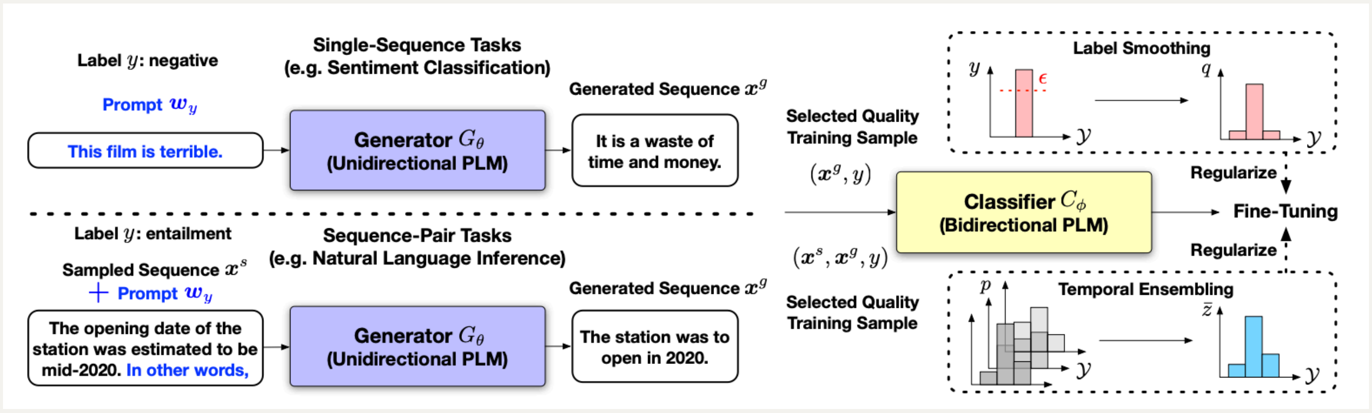
## 1.4 zero-shot

Zero-shot learning is a much more challenging setting with absolutely no access to any task-specific data. When prompt-based methods are directly used to obtain predictions from PLMs without any training, their zero-shot performance can be much worse — difficult NLU tasks can be barely formulated as prompts that resemble the format of pretraining data, posing great challenges for PLMs to accurately interpret and leverage the prompts without given any training samples.

The current mainstream of zero-shot learning is based on transfer learning: By converting a set of tasks with abundant annotations into instruction templates, entailment pairs or question-answer formats and fine-tuning PLMs on them, the PLMs acquire the cross-task transfer ability to execute unseen tasks when they are formulated in a similar format.

## Second part -- modeling pipeline

### 2.1 overview


Label $y$: negative
Single-Sequence Tasks (e.g. Sentiment Classification)
Prompt $w_y$
This film is terrible.
Generator $G_\theta$ (Unidirectional PLM)
Generated Sequence $x^g$
It is a waste of time and money.
Label $y$: entailment
Sequence-Pair Tasks (e.g. Natural Language Inference)
Sampled Sequence $x^s$ + Prompt $w_y$
The opening date of the station was estimated to be mid-2020. In other words,
Generator $G_\theta$ (Unidirectional PLM)
Generated Sequence $x^g$
The station was to open in 2020.
Selected Quality Training Sample
$(x^g, y)$
$(x^s, x^g, y)$
Selected Quality Training Sample
Classifier $C_\phi$ (Bidirectional PLM)
Fine-Tuning
Regularize
Regularize
Label Smoothing
Temporal Ensembling

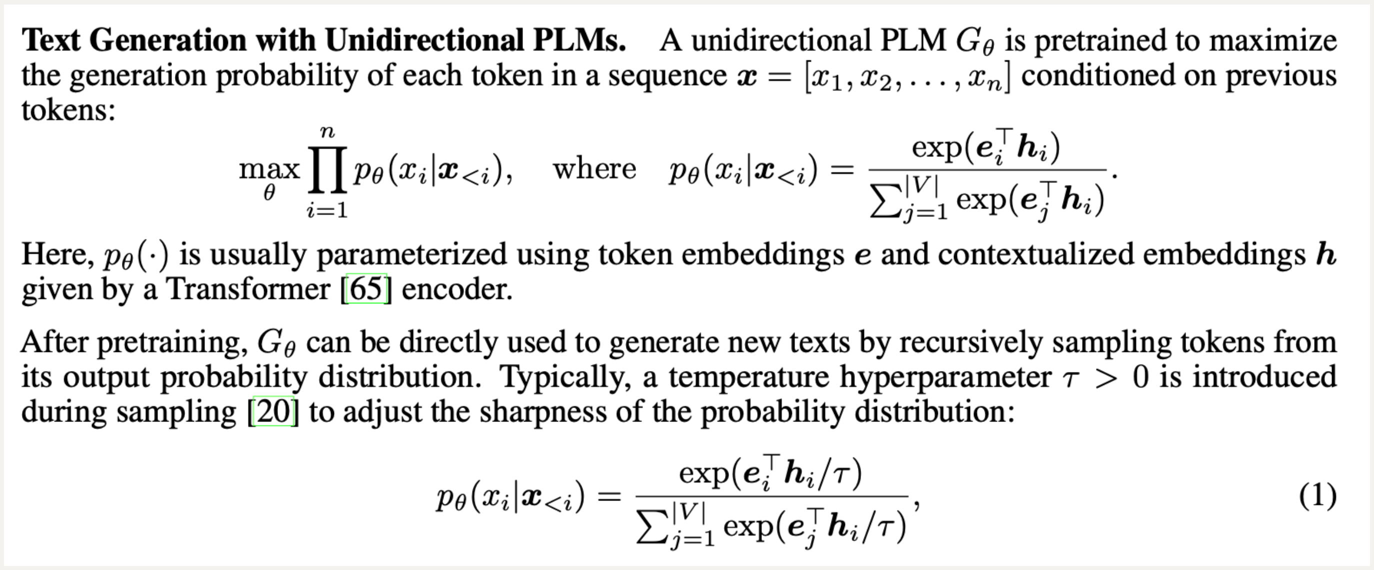
label -> prompt

sample text + prompt + unidirectional PLM -> Generated Sequence

Generated Sequence + label + label smoothing + temporal Ensembling -> the result/prediction

Use a unidirectional PLM to generate training data for fine-tuning another PLM on the target task.

### 2.2 Generateion Data with Unidirectional PLM



**Text Generation with Unidirectional PLMs.** A unidirectional PLM $G_\theta$ is pretrained to maximize the generation probability of each token in a sequence $\boldsymbol{x} = [x_1, x_2, \dots, x_n]$ conditioned on previous tokens:

$$\max_\theta \prod_{i=1}^{n} p_\theta(x_i|\boldsymbol{x}_{<i}), \quad \text{where} \quad p_\theta(x_i|\boldsymbol{x}_{<i}) = \frac{\exp(\boldsymbol{e}_i^\top \boldsymbol{h}_i)}{\sum_{j=1}^{|V|} \exp(\boldsymbol{e}_j^\top \boldsymbol{h}_i)}.$$

Here, $p_\theta(\cdot)$ is usually parameterized using token embeddings $\boldsymbol{e}$ and contextualized embeddings $\boldsymbol{h}$ given by a Transformer [65] encoder.

After pretraining, $G_\theta$ can be directly used to generate new texts by recursively sampling tokens from its output probability distribution. Typically, a temperature hyperparameter $\tau > 0$ is introduced during sampling [20] to adjust the sharpness of the probability distribution:

$$p_\theta(x_i|\boldsymbol{x}_{<i}) = \frac{\exp(\boldsymbol{e}_i^\top \boldsymbol{h}_i/\tau)}{\sum_{j=1}^{|V|} \exp(\boldsymbol{e}_j^\top \boldsymbol{h}_i/\tau)}, \tag{1}$$

## Prompt

The pre-train model is good enough so that there is no need to fine-tune or update the parameters of it.

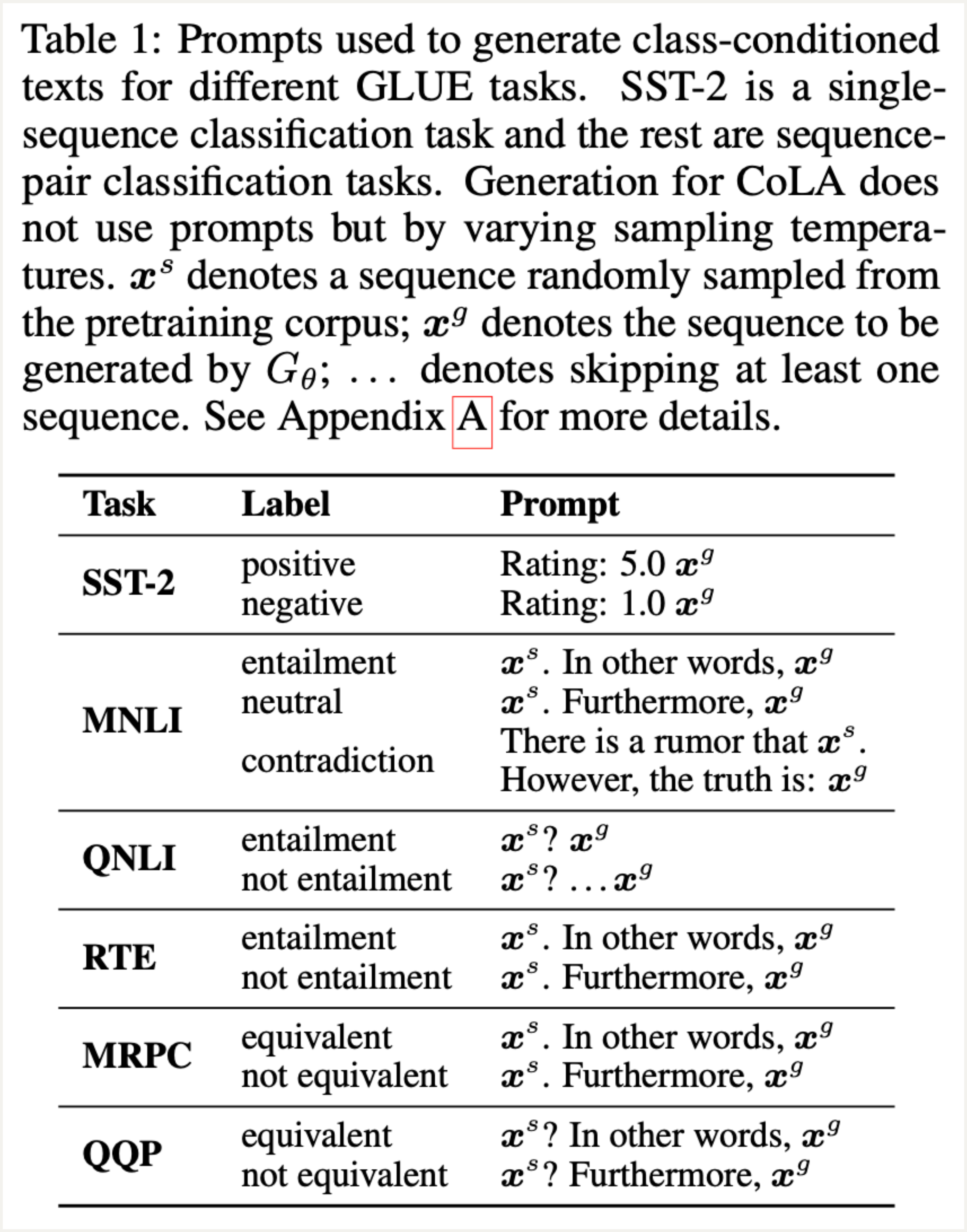


Table 1: Prompts used to generate class-conditioned texts for different GLUE tasks. SST-2 is a single-sequence classification task and the rest are sequence-pair classification tasks. Generation for CoLA does not use prompts but by varying sampling temperatures. $\boldsymbol{x}^s$ denotes a sequence randomly sampled from the pretraining corpus; $\boldsymbol{x}^g$ denotes the sequence to be generated by $G_\theta$; $\ldots$ denotes skipping at least one sequence. See Appendix A for more details.

| **Task** | **Label** | **Prompt** |
|---|---|---|
| **SST-2** | positive | Rating: 5.0 $\boldsymbol{x}^g$ |
| | negative | Rating: 1.0 $\boldsymbol{x}^g$ |
| **MNLI** | entailment | $\boldsymbol{x}^s$. In other words, $\boldsymbol{x}^g$ |
| | neutral | $\boldsymbol{x}^s$. Furthermore, $\boldsymbol{x}^g$ |
| | contradiction | There is a rumor that $\boldsymbol{x}^s$. However, the truth is: $\boldsymbol{x}^g$ |
| **QNLI** | entailment | $\boldsymbol{x}^s$? $\boldsymbol{x}^g$ |
| | not entailment | $\boldsymbol{x}^s$? $\ldots \boldsymbol{x}^g$ |
| **RTE** | entailment | $\boldsymbol{x}^s$. In other words, $\boldsymbol{x}^g$ |
| | not entailment | $\boldsymbol{x}^s$. Furthermore, $\boldsymbol{x}^g$ |
| **MRPC** | equivalent | $\boldsymbol{x}^s$. In other words, $\boldsymbol{x}^g$ |
| | not equivalent | $\boldsymbol{x}^s$. Furthermore, $\boldsymbol{x}^g$ |
| **QQP** | equivalent | $\boldsymbol{x}^s$? In other words, $\boldsymbol{x}^g$ |
| | not equivalent | $\boldsymbol{x}^s$? Furthermore, $\boldsymbol{x}^g$ |

There two kinds of prompts (unnecessary to dive in):

1. generate single sequence (SST-2 dataset)
2. generate sequence pairs (the rest of dataset)

## Issues:

A common issue in text generation is degenerate repetition where generated texts get stuck in repetition loops.

one approach is to discourage repetition by reducing the logits of tokens that are already in the sequence before performing sampling. In sequence pair generation, however, it is sometimes desirable to encourage the second sequence to repeat some words in the first sentence.

$$p_\theta(x_i|\boldsymbol{x}_{<i}) = \frac{\exp(\boldsymbol{e}_i^\top \boldsymbol{h}_i/\omega)}{\sum_{j=1}^{|V|} \exp(\boldsymbol{e}_j^\top \boldsymbol{h}_i/\omega)}, \quad \text{where} \quad \omega = \begin{cases} \tau\alpha & x_i \in \boldsymbol{x}^s \wedge x_i \notin \boldsymbol{x}^g \\ \tau\beta & x_i \in \boldsymbol{x}^g \\ \tau & \text{else} \end{cases}, \tag{2}$$

and $\alpha > 0, \beta > 0$ are hyperparameters. By setting $\alpha < 1$ and $\beta > 1$, we can promote tokens in $\boldsymbol{x}^s$ that have not appeared in $\boldsymbol{x}^g$ to have a higher chance of being generated, and discourage the generation of repetitive tokens in $\boldsymbol{x}^g$ to mitigate degenerate repetition. The parameters used for different tasks are listed in Appendix B Table 9.

## Selecting Quality Training Data

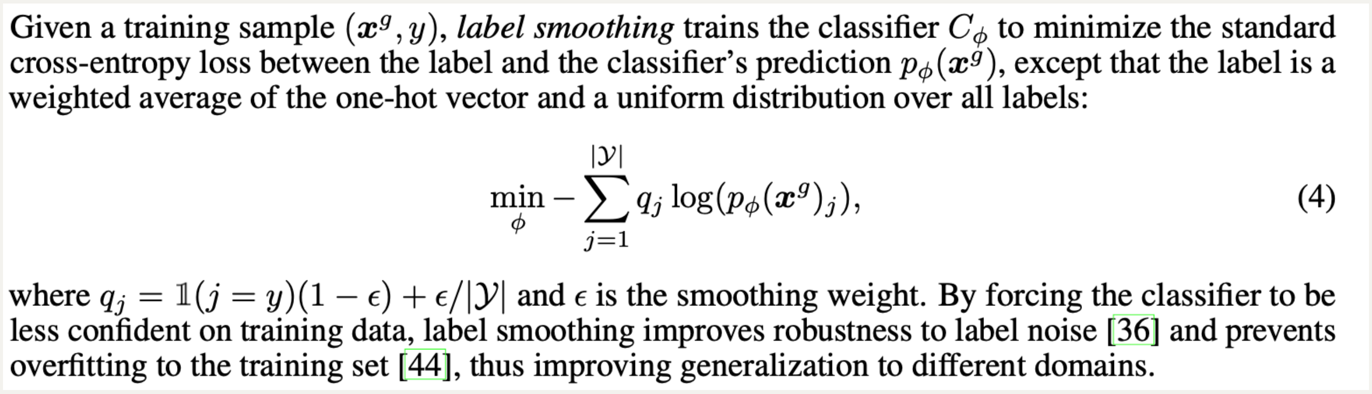
$$r = \frac{1}{n} \sum_{i=1}^{n} \log p_\theta \left( x_i \middle| [\boldsymbol{w}_y; \boldsymbol{x}^g_{<i}] \right).$$

Calculate the score based on the above equation. To construct a training set consisting of N samples per class, we will generate more samples (*e.g.*, 10N). Get rid of the remaining 9N samples, only keep the most highest score samples.

## 2.3 fine-tune

Even with the above training data selection procedure, the resulting training set may still contain noise and there exists domain difference from the downstream tasks.

### label smoothing

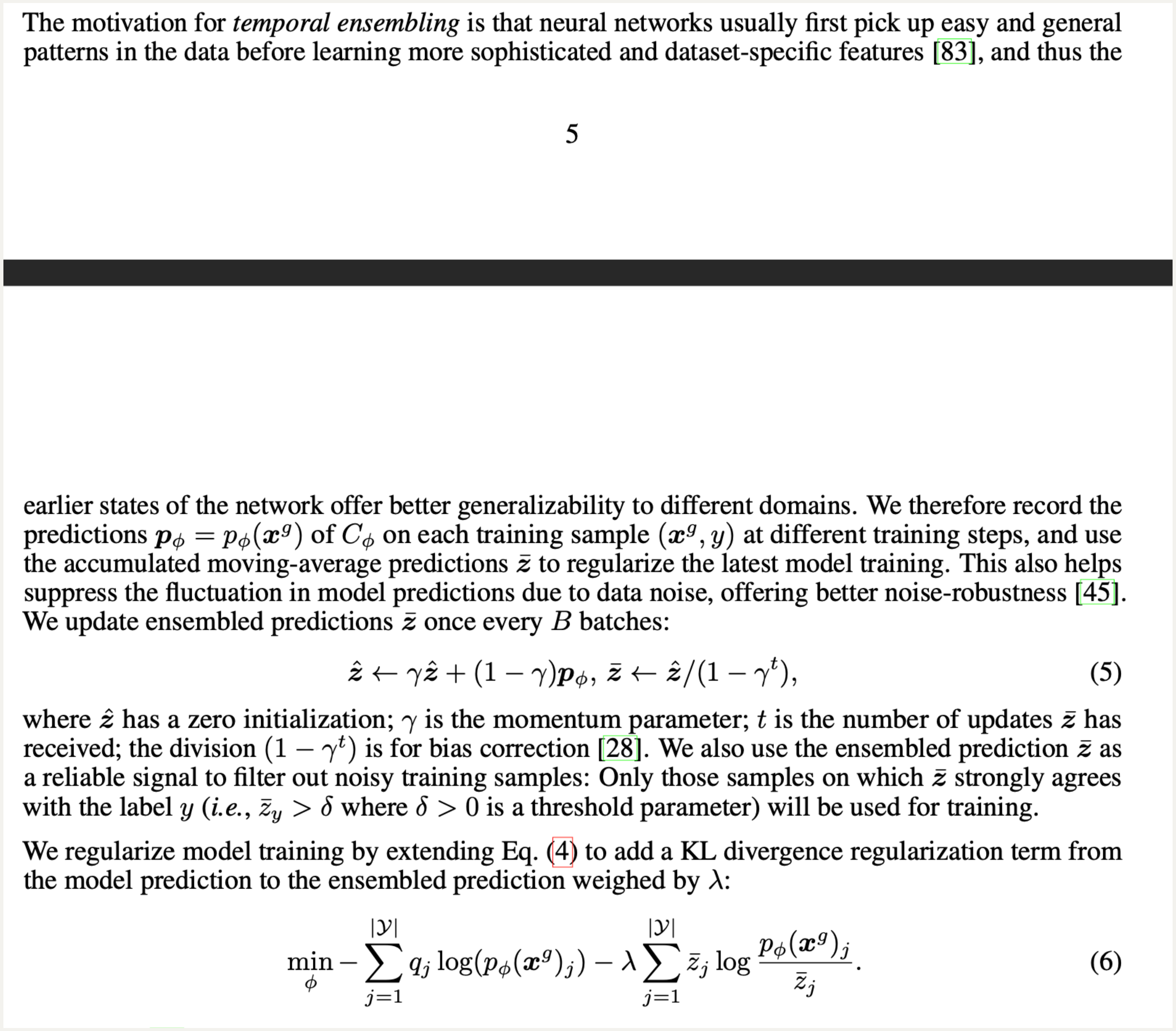


Given a training sample $(\boldsymbol{x}^g, y)$, *label smoothing* trains the classifier $C_\phi$ to minimize the standard cross-entropy loss between the label and the classifier's prediction $p_\phi(\boldsymbol{x}^g)$, except that the label is a weighted average of the one-hot vector and a uniform distribution over all labels:

$$\min_\phi - \sum_{j=1}^{|\mathcal{Y}|} q_j \log(p_\phi(\boldsymbol{x}^g)_j), \tag{4}$$

where $q_j = \mathbb{1}(j = y)(1 - \epsilon) + \epsilon/|\mathcal{Y}|$ and $\epsilon$ is the smoothing weight. By forcing the classifier to be less confident on training data, label smoothing improves robustness to label noise [36] and prevents overfitting to the training set [44], thus improving generalization to different domains.

### temporal ensembling



The motivation for *temporal ensembling* is that neural networks usually first pick up easy and general patterns in the data before learning more sophisticated and dataset-specific features [83], and thus the

earlier states of the network offer better generalizability to different domains. We therefore record the predictions $\boldsymbol{p}_\phi = p_\phi(\boldsymbol{x}^g)$ of $C_\phi$ on each training sample $(\boldsymbol{x}^g, y)$ at different training steps, and use the accumulated moving-average predictions $\bar{\boldsymbol{z}}$ to regularize the latest model training. This also helps suppress the fluctuation in model predictions due to data noise, offering better noise-robustness [45]. We update ensembled predictions $\bar{\boldsymbol{z}}$ once every $B$ batches:

$$\hat{\boldsymbol{z}} \leftarrow \gamma \hat{\boldsymbol{z}} + (1-\gamma)\boldsymbol{p}_\phi, \; \bar{\boldsymbol{z}} \leftarrow \hat{\boldsymbol{z}}/(1-\gamma^t), \tag{5}$$

where $\hat{\boldsymbol{z}}$ has a zero initialization; $\gamma$ is the momentum parameter; $t$ is the number of updates $\bar{\boldsymbol{z}}$ has received; the division $(1-\gamma^t)$ is for bias correction [28]. We also use the ensembled prediction $\bar{\boldsymbol{z}}$ as a reliable signal to filter out noisy training samples: Only those samples on which $\bar{\boldsymbol{z}}$ strongly agrees with the label $y$ (*i.e.*, $\bar{z}_y > \delta$ where $\delta > 0$ is a threshold parameter) will be used for training.

We regularize model training by extending Eq. (4) to add a KL divergence regularization term from the model prediction to the ensembled prediction weighed by $\lambda$:

$$\min_\phi - \sum_{j=1}^{|\mathcal{Y}|} q_j \log(p_\phi(\boldsymbol{x}^g)_j) - \lambda \sum_{j=1}^{|\mathcal{Y}|} \bar{z}_j \log \frac{p_\phi(\boldsymbol{x}^g)_j}{\bar{z}_j}. \tag{6}$$

## hird Part -- Pros and Cons:

Pros:

1. Moderate size of Model
2. Removing the need for a large amount of cross-task annotations
3. Eliminating the task difference in training and inference

4. Different from previous studies that rely on labeled data to fine-tune the generative PLM, we directly use prompts to guide data generation without fine-tuning

## Cons:

1. the lack of access to task-specific samples for hyperparameter tuning
2. the generated training data quality may not be high enough to achieve good performance on challenging tasks, especially when the task distribution is significantly different from the pretraining data distribution.