CS 182/282A Designing, Visualizing and Understanding Deep Neural Networks
Spring 2022 Marvin Zhang

Discussion 12

This discussion focuses on self-supervised learning and massive models.

1 Self-Supervised Learning

Deep learning has seen considerable success when we have large labeled datasets to learn from, e.g. ImageNet. However, it is usually not practical to collect large labeled datasets for each task and domain that we want to apply deep learning to. To some extent, we can leverage models trained on such datasets for transfer learning on different tasks, and hope that the representations learned can generalize. However, this approach is still limited because the learned representations may overfit to the original dataset and not generalize well beyond it, and we still need to collect labeled data for each domain and modality we are interested in.

While labeled data can be hard to come by, there often exists large amounts of unlabeled data that is widely available, e.g. text on the internet. Although we cannot apply supervised learning directly to this data, we can still hope to potentially learn something about the structure of the data, and thus learn representations that capture understanding of the data domain and can be reused for different tasks in the domain.

How can we learn from unlabeled data? One popular class of methods involve predicting one part of the input data from another part. For example, the masked language modeling objective used in BERT is an example of this, since we are predicting masked-out words from other words in their context.

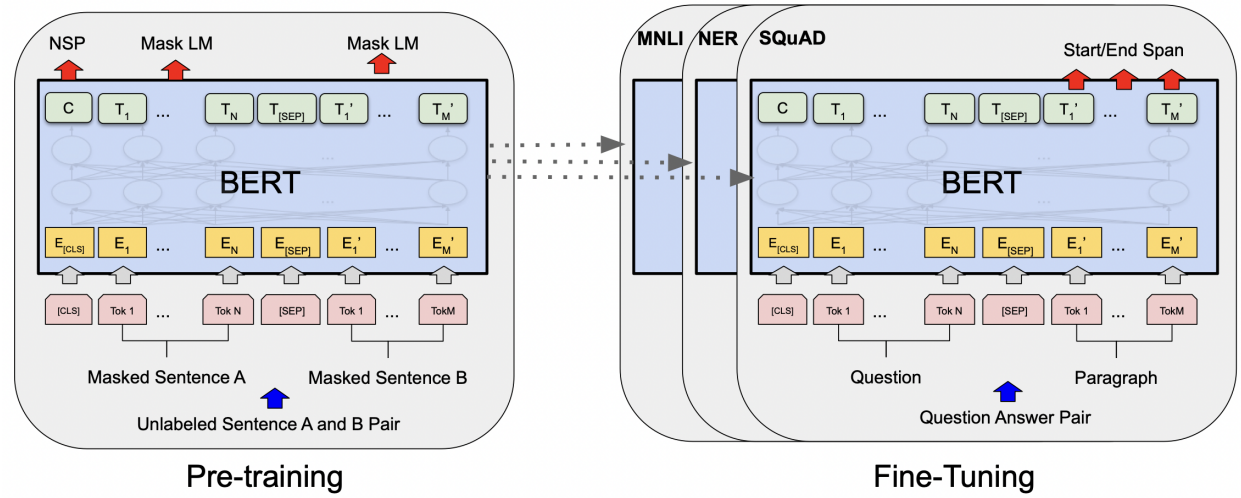

Figure 1: The masked language modeling objective we saw previously with BERT is an example of self-supervised learning.

Another example of this are variational autoencoders (VAEs), which we previously saw in the context of generative modeling. VAEs can also be seen as self-supervised representation learning, since we are asking our decoder to predict the original input from the latent distribution produced by the encoder, which cannot contain all the original information in the input. The output of the encoder can then be used as learned representations for a downstream task.

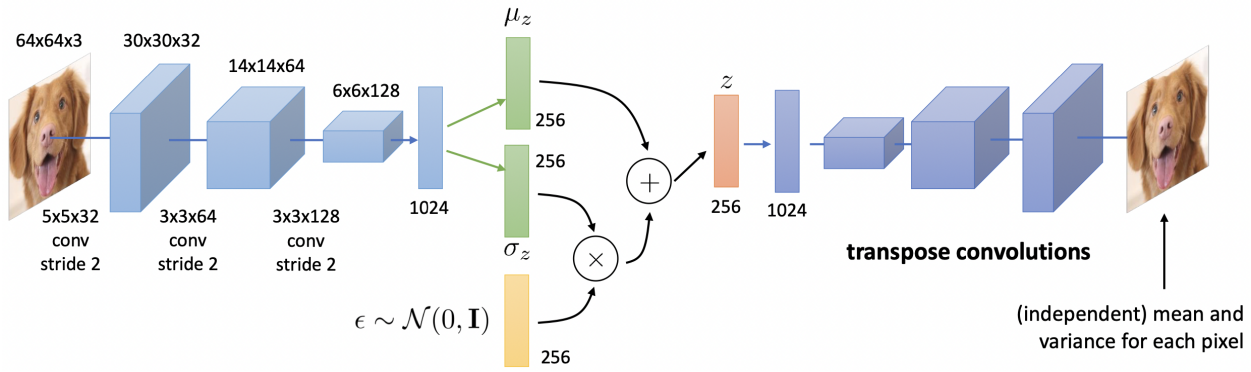

Figure 2: Example of a convolutional VAE with images. It is common to use μ_z as the representation for a downstream task.

Another popular class of methods involve data augmentation, usually in the context of image data. In these methods, we provided augmented versions of data to our models, and ask them to make predictions that are in some way consistent with another augmented version of the same data. Through this, we implicitly provide the model with domain knowledge to learn from, specifically the key idea that two augmented versions of the same data should represent the same thing. For example, two different random crops of a dog image should still represent a dog.

Usually, after we have learned representations using self-supervised learning, we can either use them as frozen feature extractors for supervised learning on a downstream task, or we can fine-tune the entire network end-to-end on our downstream task. In either case, we usually require a significantly smaller amount of labeled data than if we had trained from scratch.

We will now cover a few modern examples of data augmentation based self-supervised learning.

1.1 SimCLR

SimCLR is a simple framework for contrastive representation learning with images. The idea behind contrastive representation learning is that we want representations for similar inputs to also be similar. However, if we train a model to only maximize the similarity between representations for similar inputs, then the model can recover a degenerate solution such as always outputting a vector of all zeros, which trivially maximizes this objective. With contrastive learning, we additionally train our model to maximize the dissimilarity between representations for pairs of arbitrary inputs, so that we do not recover such a degenerate solution. Usually, a pair of similar inputs is referred to as a positive pair, and an arbitrary pair of inputs is referred to as a negative pair.

The loss function used in SimCLR for a positive pair of examples (i, j), which has also been used in other contrastive learning methods, is defined as:

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} 1_{k \neq i} \exp(\text{sim}(z_i, z_k)/\tau)}$$

z_i and z_j are (transformed) representations for our pair of inputs, $\text{sim}(z_i, z_j)$ refers to the cosine similarity between z_i and z_j, τ is a temperature hyperparameter, and N is the amount of pairs in our training batch. In SimCLR, each positive pair consists of two different randomly augmented versions of the same image. We use the other augmented images in our training batch to construct negative pairs with our input i, which can be seen in the denominator of the loss function. By minimizing this loss function, we maximize the similarity between positive pairs, and minimize the similarity between negative pairs.

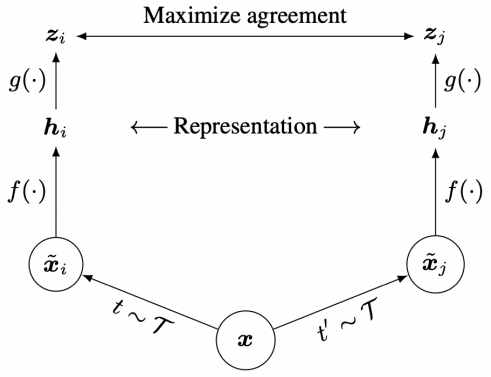

Figure 3: Diagram of SimCLR. x is the original image, t and t' are randomly sampled augmentations from our augmentation distribution T, and $\tilde{x}_i$ and $\tilde{x}_j$ are the respective transformed images. Notably, we apply our contrastive loss not directly to our representations h, but to their transformations z. The representation network f and the additional transformation network g are both learned jointly on the contrastive loss.

1.2 Bootstrap Your Own Latent (BYOL)

BYOL is another data augmentation based self-supervised learning method, but it does not use contrastive learning. Instead, we take our pair of augmented images and feed them individually into two different networks, the main online network we are training, and a target network. We take the (transformed) representation from our online network, feed it through an additional prediction network, and minimize the mean squared error between this and the (transformed) representation from the target network. Notably, we do not train the target network here. The target network has an identical architecture to the online network (except for the prediction head), but its weights are an exponential moving average of the online network's weights. This means that after each training step, we update its weights to be closer to the online network, but not fully.

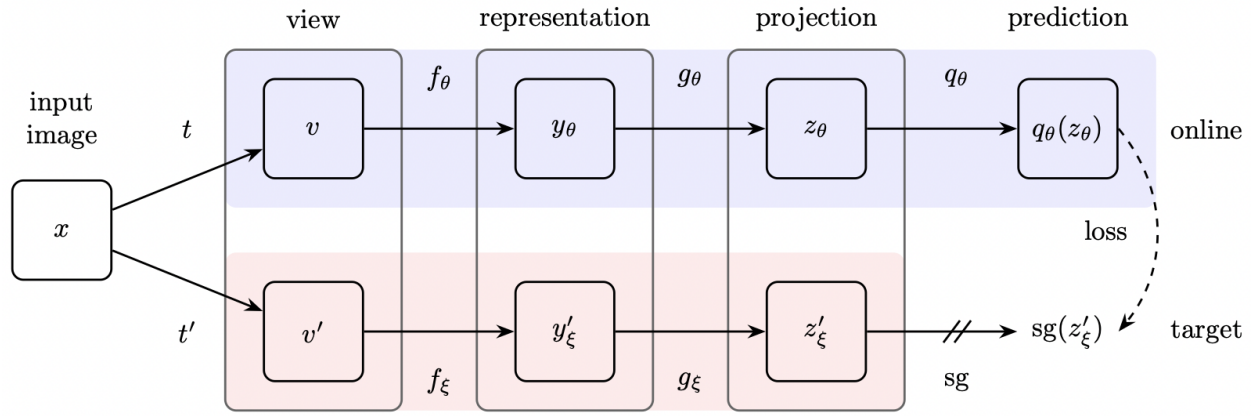

Figure 4: Diagram of BYOL. sg refers to stop gradient, meaning we do not update the target network to minimize the loss.

Problem 1: Self-supervised Learning

How does SimCLR avoid degenerate solutions? Is this issue something to be potentially concerned about with BYOL?

Solution 1: Self-supervised Learning

SimCLR avoids degenerate solutions through a contrastive loss that encourages arbitrary image pairs to have dissimilar representations. BYOL in principle could suffer from degenerate solutions, since outputting only zeros for everything would minimize its loss. However, in practice BYOL does not suffer from this, and its authors hypothesize that the additional predictor network as well as the exponential moving average of parameters in the target network help prevent this from happening.

2 Massive Models

With the recent rise of self-supervised learning, there has been another emerging trend that has emerged alongside it: massive models. With self-supervised learning, we now have the ability to leverage much larger and diverse datasets than before, meaning we can also now train much larger models that can take advantage of all this data. Most of these models have been developed in the domain of natural language processing (NLP), and the all of the very large ones (>100 billion parameters) developed so far are transformer decoder-only language models. The transformer architecture is widely used for large-scale models partly because the parallelizability of the attention mechanism allows for more efficient utilization of compute.

Recent examples of massive NLP models include GPT-3, Gopher, MT-NLG, and PaLM. While these models vary in terms of datasets, architectures, and hyperparameters, they are all trained on the standard language modeling objective, using large amounts of data obtained from the internet and other sources. While their training objective is simple, the scale and diversity of their training data enables them to do a wide range of impressive things. The general trend we've seen so far in these models is that performance continues to improve as we scale up both dataset size and model capacity.

2.1 In-context Learning

Large pre-trained models have been leveraged for downstream tasks by using them for representations, either by only training a small model on top of these representations, or fine-tuning the entire model. This approach can be impractical for massive models, due to the compute needed to run and potentially fine-tune them on moderately sized datasets. However, these massive models have shown to perform downstream tasks with only a few examples of the task, and without any fine-tuning or gradient updates. This is done through in-context learning, where we simply provide the task examples to the model in the prompt, and we ask the model to predict a next sequence of tokens in order to complete the task. For example, if we want to do machine translation from English to French, we can prompt our language model with some examples of translations, and then provide our model with the English sentence we want translated in the same format as the examples, and have our model do translation simply by predicting the next tokens that should follow.

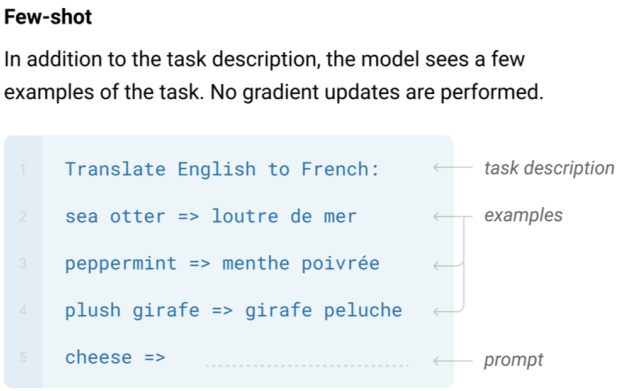

Figure 5: Example of few-shot in-context learning for machine translation.

In-context learning can be successful despite the fact that the model's parameters are never updated on text sequences that closely resemble those seen in the prompt. Through large-scale training, these models are able to generalize to these new scenarios and apply their understanding of language to them. However, this is only a recently observed phenomenon, and it is still unclear exactly how in-context learning works, what is important for it (e.g. how to best construct prompts for different tasks), and how we can best leverage it. Also, when fine-tuning is possible, it usually still results in better performance.