1 What is self-supervised learning?

One way to view SSL is as a slight twist on unsupervised learning, i.e. learning from unlabelled data. The difference with SSL is there is usually a loss function involved. It is easiest to understand with an example. Suppose we have access to a large dataset $\mathcal{D}$ of *unlabelled* images (just pick your favourite dataset, e.g. ImageNet and throw away the labels). We want to learn a representation of this dataset that can be used for downstream tasks. To make this concrete, a representation is just a function f_θ (usually a neural network) which maps an image x to a latent variable r. i.e. we want to learn a function:

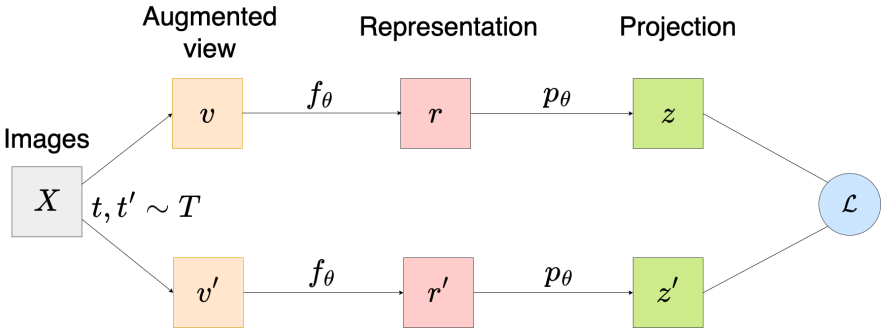
$$f_\theta(x) = r \tag{1}$$

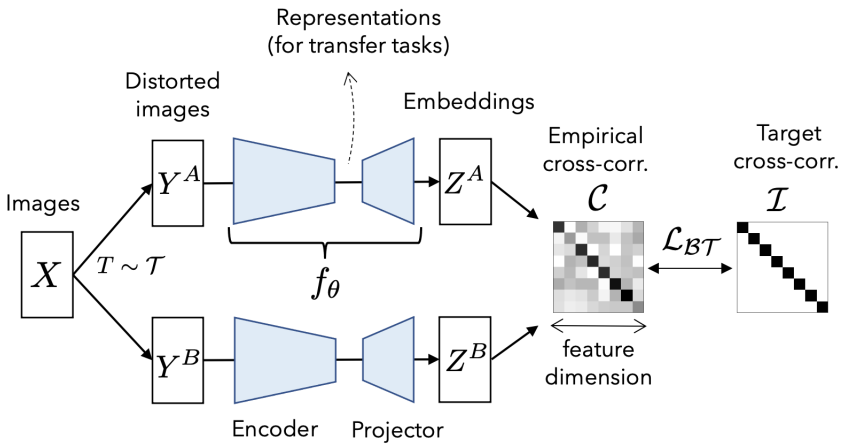
To give an example of a downstream task - suppose we have access also to a small labelled $\mathcal{D}'$ which consists of the same distribution of images as $\mathcal{D}$ but this time with the labels. Hence, $\mathcal{D}$ might be a large fraction of unlabelled ImageNet samples x and $\mathcal{D}'$ a small fraction of ImageNet samples (x, y) *with* the labels y. Then once we have learned f_θ on the unlabelled $\mathcal{D}$ the downstream task is to fine tune a linear classifier h_ϕ on top of the representation. That is, perform standard supervised learning on top of the learned representation:

$$h_\phi(z) = h_\phi(f_\theta(x) \approx y. \tag{2}$$

We can either freeze the θ weights and just learn ϕ (which is called linear evaluation), or fine-tune both θ and ϕ. Intuitively, freezing θ makes the downstream supervised learning task harder (as we are just training a linear head h_ϕ on top of the frozen representation). Hence, if this task is successful it is plausible that the representation is of higher quality and will be useful on other downstream tasks e.g. transfer learning to other datasets.

(1) is the SSL task. I will now describe one way of performing this task, which is a variant of non-contrastive SSL.




Representations
(for transfer tasks)
Distorted
images
Embeddings
Empirical
cross-corr.
Target
cross-corr.
Images
$\mathcal{C}$
$\mathcal{I}$
Y^A
Z^A
$\mathcal{L}_{\mathcal{BT}}$
X
$T \sim \mathcal{T}$
f_θ
Y^B
Z^B
feature
dimension
Encoder
Projector

1 Independence vs. Uncorrelated

For *any* random variables X, Y if they are independent then they are uncorrelated:

$$\text{(i)}\ \ X \perp\!\!\!\perp Y \implies \text{Corr}(X, Y) = 0.$$

The reverse equivalence is *false* in general:

$$\text{(ii)}\ \ \text{Corr}(X, Y) = 0 \not\implies X \perp\!\!\!\perp Y.$$

That is, random variables can have correlation of zero while still being dependent. However, for a pair (X, Y) that are multivariate normal, the reverse of (i) is true: in this case, $\text{Corr}(X, Y) = 0 \implies X \perp\!\!\!\perp Y$. It is sometimes wrongly believed that if a pair (X, Y) has normal marginals then this still holds. However an example shows this is false:

If $X \sim \mathcal{N}(0, 1)$, and $Y = WX$ where W takes value 1 or -1 with probability $\frac{1}{2}$ then $Y \sim \mathcal{N}(0, 1)$ as well. It is also clear X and Y are (*very*) dependent. (In fact $|X| = |Y|$). Despite this a simple calculation shows that $\text{Corr}(X, Y) = 0$. Hence we have an example of (ii) with both X and Y normal. However, if

$$\text{(iii)}\ \ \text{Corr}(h(X), g(Y)) = 0,$$

for all $h, g \in \mathcal{H}$, a suitable class of test functions (e.g. continuous functions), then we can say that $X \perp\!\!\!\perp Y$.

1 Review of BT and description of RBT

In this section, to simplify notation, we ignore the invariance term of BT since it is the same in RBT. Also when we write $\mathcal{L}_{ij}$ it is understood to mean for $i \neq j$ and also that the final loss term is a sum over all such i, j, i.e. $\mathcal{L} = \sum_{i \neq j} \mathcal{L}_{ij}$

Each time a batch of data is sampled, BT takes a gradient descent step optimizing the following loss term (note that of course z_i^A, z_j^B depend on the batch):

$$\mathcal{L}_{ij}^{bt} = \mathrm{Corr}(z_i^A, z_j^B)^2 \tag{1}$$

As we have seen, this objective does not disallow degenerate solutions. In fact, a possible solution to (1) involves random variables the same, up to sign.

However, we have the following:

Theorem 1.1.

If $\mathrm{Corr}(f(X), g(Y)) = 0$ *for all* $f, g \in \mathcal{H}$, *then* X *and* Y *are independent.*

This motivates a *regularized* form of BT. Roughly speaking, each time we sample a batch of data, we also sample random functions, apply them as in the theorem, and optimize the new objective. More specifically, RBT is a convex combination of two terms. The first term is:

$$\mathcal{L}_{ij}^{rand} = \frac{1}{2}\,\mathrm{Corr}(r_1(z_i^A), z_j^B)^2 + \frac{1}{2}\,\mathrm{Corr}(z_i^A, r_2(z_j^B))^2, \tag{2}$$

where $r_1, r_2 \sim \mathcal{R}$ are randomly sampled sinusoid functions (this means each time we sample a batch of data, a new random r_1 and r_2 are sampled). The next term is:

$$\mathcal{L}_{ij}^{max} = \sup_{f,g}\{\frac{1}{2}\,\mathrm{Corr}(f(z_i^A), z_j^B)^2 + \frac{1}{2}\,\mathrm{Corr}(z_i^A, g(z_j^B))^2\}, \tag{3}$$

where the supremum is over a suitable class of functions. In practise, we find that if (2) is too well optimised, learning is not possible. We take f, g to be feedforward neural networks with one hidden layer (newly initialized for each minibatch) that are trained for 5 steps of SGD.

Final loss term is convex combination of (2) and (3)

$$\mathcal{L}_{ij}^{rbt} := t\mathcal{L}_{ij}^{rand} + (1-t)\mathcal{L}_{ij}^{max} \tag{4}$$

We hypothesize that this new objective regularizes BT via constraining the search space to rule out degenerate solutions. For example, BT allows random variables (that are meant to be encoding redundancy reduction) to differ only by sign, whereas RBT does not.

It is clear that, mathematically speaking, constraint (2) and (3) are equivalent (provided the class of test functions is large enough in each case). It is therefore interesting that performance is highest when *both* terms are included. This can be viewed in terms of exploration/exploitation. (2) via sampling different functions for each batch tries to enforce that the correlation is zero for all

functions; on the other hand (3) tries to make the correlation zero in the worst possible case. This makes more sense when we consider that the optimization is performed on a noisy mini-batch.

1 Preliminary results

We randomly sample $16384 = 512 \times 32$ MNIST images and use these to train BT and RBT. Then we randomly sample 20 labelled MNIST samples (two samples for each class) and fine tune a linear classifier on the frozen representations, i.e. the linear evaluation protocol. The results are evaluated on the remaining 43596 images. This second step is repeated for 5 different random 20 samples.

Seed	BT	RBT
1	76.67	82.38
2	68.43	78.34
3	84.68	89.99
4	76.87	86.01
5	72.88	82.21
mean	**75.91**	**83.79**

The results in the table give the performance when training a linear classifier on 20 samples, (on top of the frozen representation), repeated for 5 different seeds. Although RBT beat BT each time it is important to remember that we optimized the RBT specific hyperparameters using this specific train, fine-tune split of the data. Therefore, using this setting of the hyperparameters we repeated this procedure for a different random training set, and different tuning sets. The results are similar, which is encouraging.

Seed	BT	RBT
6	75.32	80.16
7	75.01	78.49
8	78.08	84.16
9	78.04	83.64
10	74.61	79.86
mean	**76.21**	**81.26**

2 Concerns / Work to do

1. Biggest concern is that I have been overfitting to MNIST (although all work so far is only on the training set) Need to test out on another dataset to determine whether method really works.
2. If hypothesis is that RBT works as a regularizer, then makes sense to compare (and combine?) it with other forms of regularisation e.g. dropout.
3. If method does work, is of interest to do an analysis on the differences in the representations learned by RBT vs MBT. e.g. do independence tests on (z_i^A, z_j^B).