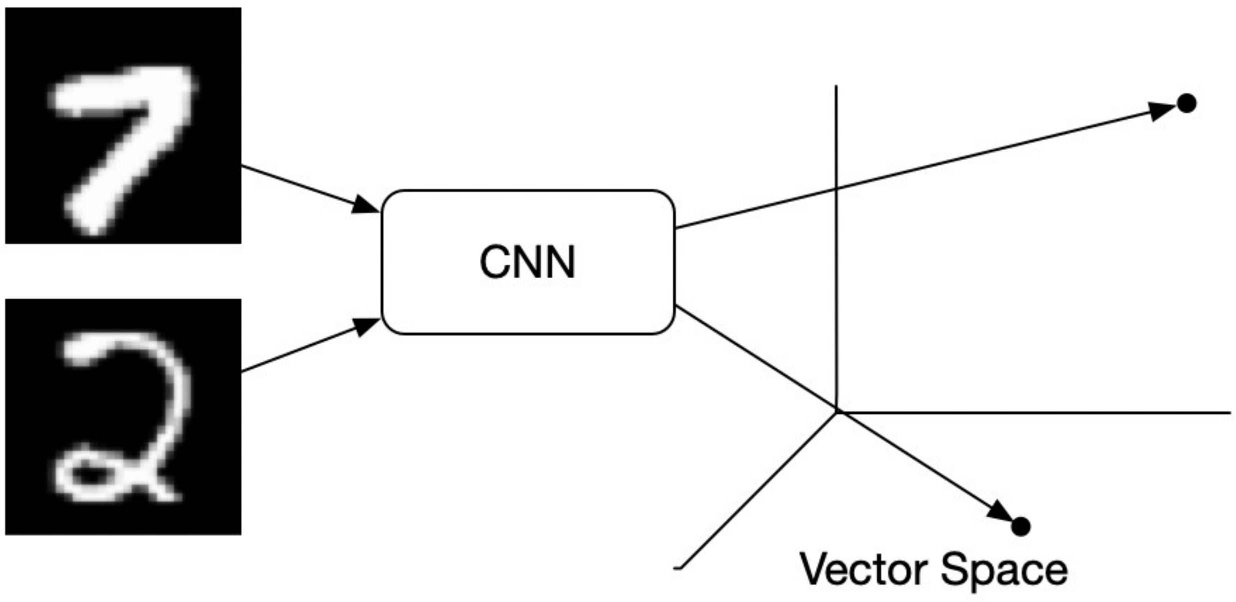
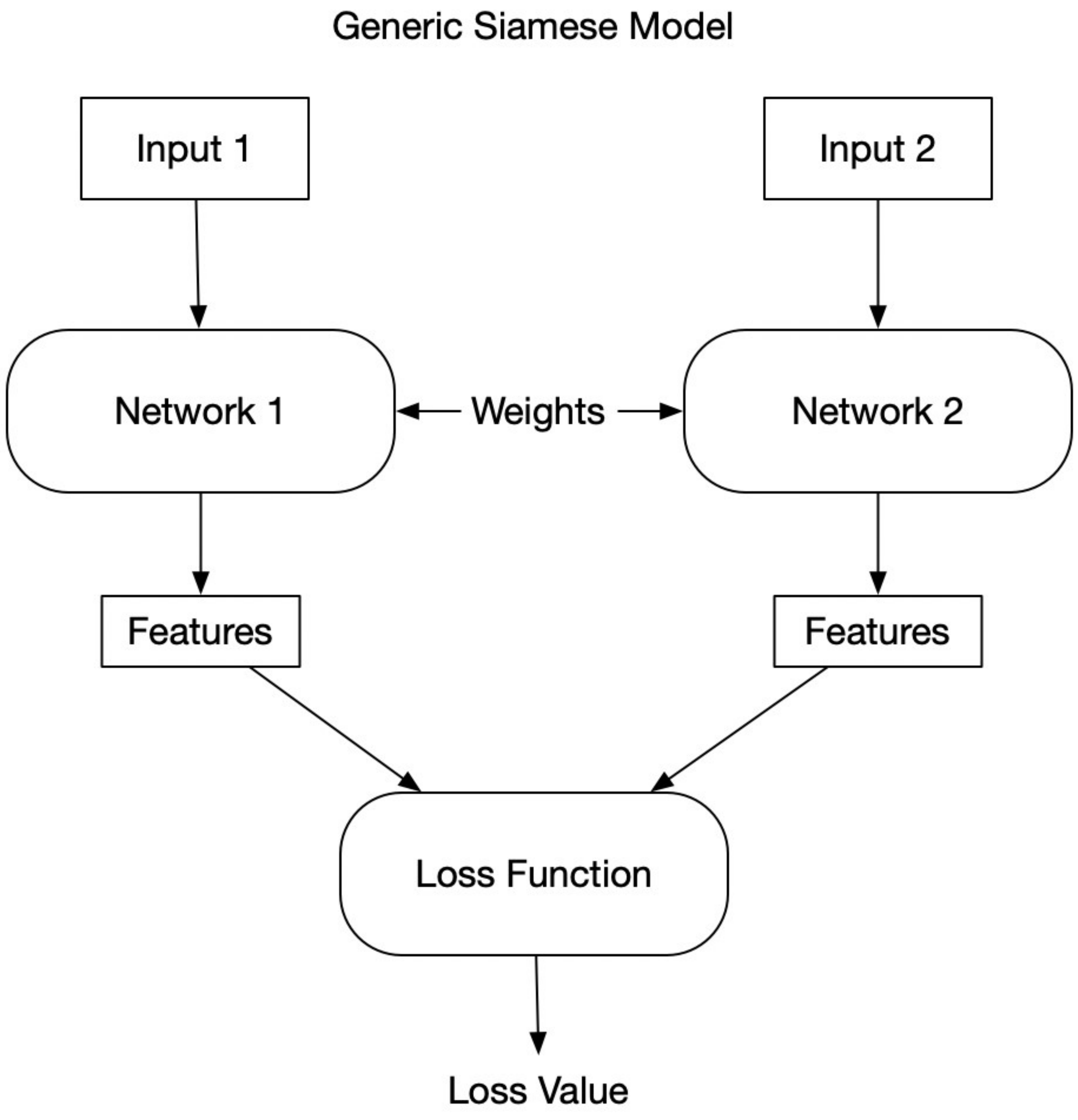
Brian Williams Follow
Mar 3, 2020 · 5 min read ★ · Listen

# Contrastive Loss Explained

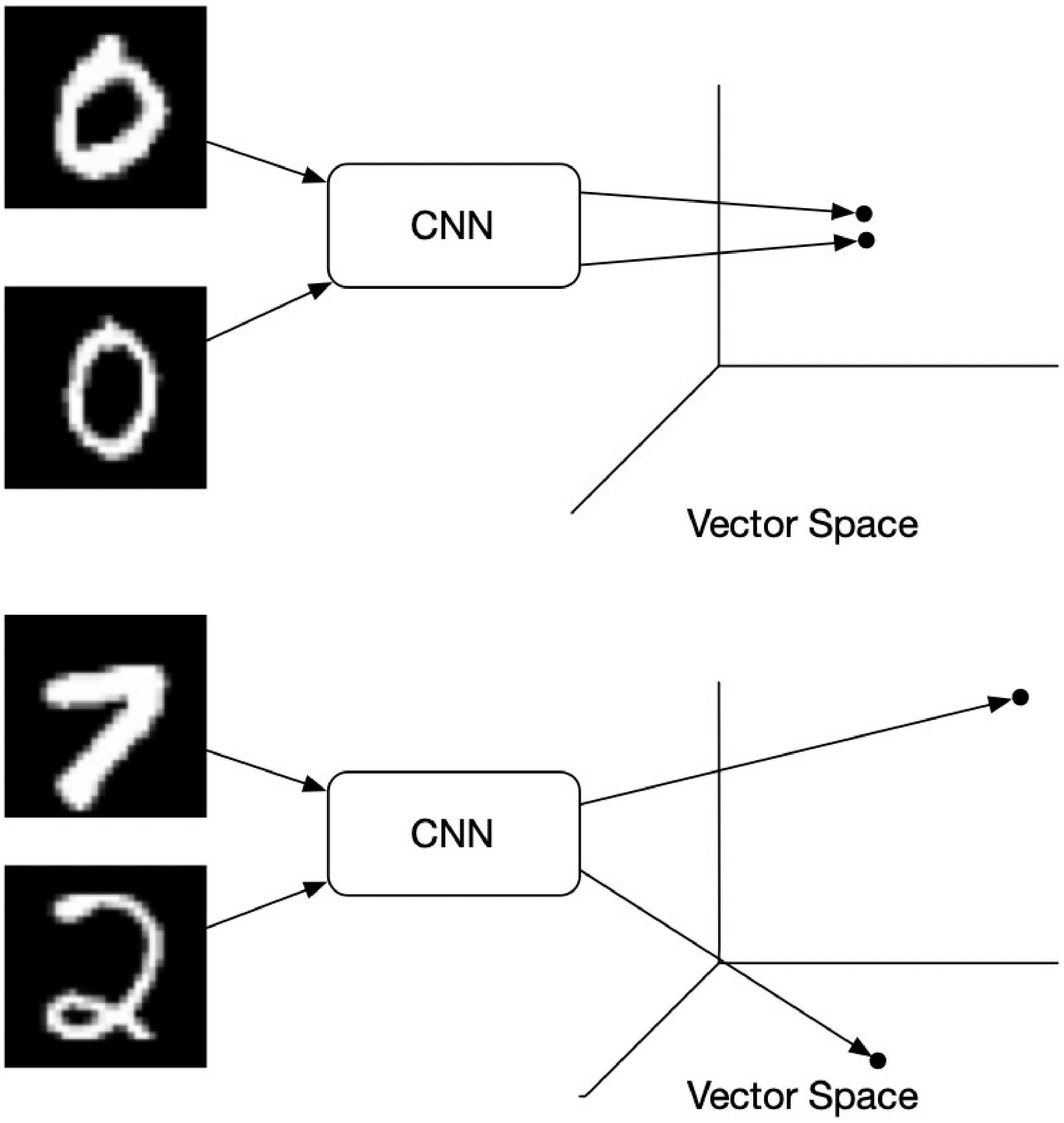


Contrastive loss has been used recently in a number of papers showing state of the art results with unsupervised learning. MoCo, PIRL, and SimCLR all follow very similar patterns of using a siamese network with contrastive loss. When reading these papers I found that the general idea was very straight forward but the translation from the math to the implementation wasn't well explained. As is often the case in machine learning papers, the math isn't difficult once you've got the main called **twin networks** but it’s not a widely used term). When training a siamese network, 2 or more inputs are encoded and the output features are compared. This comparison can be done in a number of ways. Some of the comparisons are triplet loss, pseudo labeling with cross-entropy loss, and contrastive loss.

A siamese network is often shown as two different encoding networks that share weights, but in reality the same network is just used twice before doing backpropagation.



Each image of an MNIST number should encode into a vector that is close to vectors from images of the same class. Conversely different numbers should encode into vectors that are far from each other.



Positive (top) and negative samples embedded into a vector space

Since we have the class labels for MNIST inputs we could use a regular network and Categorical Cross-Entropy loss. The problem is when we don't have nicely labeled distance to negative examples. Said another way, the loss is low if positive samples are encoded to similar (closer) representations and negative examples are encoded to different (farther) representations.

This is accomplished by taking the cosine distances of the vectors and treating the resulting distances as prediction probabilities from a typical categorization network. The big idea is that you can treat the distance of the positive example and the distances of the negative examples as output probabilities and use cross entropy loss.

When performing supervised categorization, the network outputs are typically run through a softmax function then the negative log-likelihood loss.

Let’s make this more concrete. This example will have two vectors that are similar and an array of dissimilar ones. P1 and p2 are the positive vectors, with p2 being a slightly modified version of p1. Neg is the array of dissimilar vectors.

**Example Setup**

```
import numpy as np

p1 = np.array([-0.83483301, -0.16904167, 0.52390721])
p2 = np.array([-0.83455951, -0.16862266, 0.52447767])
neg = np.array([
 [ 0.70374682, -0.18682394, -0.68544673],
 [ 0.15465702,  0.32303224,  0.93366556],
 [ 0.53043332, -0.83523217, -0.14500935],
 [ 0.68285685, -0.73054075,  0.00409143],
 [ 0.76652431,  0.61500886,  0.18494479]])
```

**Calculating Distance**

In order to measure how similar two vectors are to each other, we need a way of measuring distance. In 2 or 3 dimensions the euclidian distance (“ordinary” or straight-line distance) is a great choice for measuring the distance between two points. However, in a large dimensional space, all points tend to be far apart by the euclidian measure. In higher dimensions, the angle between vectors is a more

```
# P1 and p2 are nearly identically, thus close to 1.0
pos_dot = p1.dot(p2)
pos_dot -> 0.999999716600668

# Most of the negatives are pretty far away, so small or negative
num_neg = len(neg)
neg_dot = np.zeros(num_neg)
for i in range(num_neg):
    neg_dot[i] = p1.dot(neg[i])

neg_dot -> [-0.91504053,  0.30543542, -0.37760565, -0.44443608,
-0.64698801]
```

**Softmax**

The softmax function takes a vector of real numbers and forces them into a range of 0 to 1 with the sum of all the numbers equaling 1. One other nice property of softmax is that one of the values is usually much bigger than the others. When calculating the loss for categorical cross-entropy, the first step is to take the softmax of the values, then the negative log of the labeled category.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \dots, K \text{ and } \mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K$$

Softmax, the mathy version

Let’s take the softmax of pos_dot appended with neg_dot vector.

```
# make a vector from the positive and negative vectors comparisons
v = np.concatenate(([pos_dot], neg_dot))

# take e to the power of each value in the vector
exp = np.exp(v)

# divide each value by the sum of the exponentiated values
softmax_out = exp/np.sum(exp)
```

Our positive example (0.4296791) is now much bigger than the random ones, and all bigger than zero and less than one.

**Contrastive Loss**

Finally, we get to focus of this article, contrastive loss.

$$\ell_{i,j} = -\log \frac{\exp(\mathrm{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\mathrm{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)} ,$$

The contrastive loss function

Contrastive loss looks suspiciously like the softmax function. That's because it is, with the addition of the vector similarity and a temperature normalization factor. The similarity function is just the cosine distance that we talked about before. The other difference is that values in the denominator are the cosign distance from the positive example to the negative samples. Not very different from CrossEntropyLoss. The intuition here is that we want our similar vectors to be as close to 1 as possible, since -log(1) = 0, that's the optimal loss. We want the negative examples to be close to 0 , since any non-zero values will reduce the value of similar vectors.

```
# Contrastive loss of the example values
# temp parameter
t = 0.07

# concatenated vector divided by the temp parameter
logits = np.concatenate(([pos_dot], neg_dot))/t

#e^x of the values
exp = np.exp(logits)

# we only need to take the log of the positive value over the sum
of exp.
loss = - np.log(exp[0]/np.sum(exp))
```

vectors that model the similarity of input items. These mappings can support many tasks, like unsupervised learning, one-shot learning, and other distance metric learning tasks. I hope that this article has helped you better understand contrastive loss.