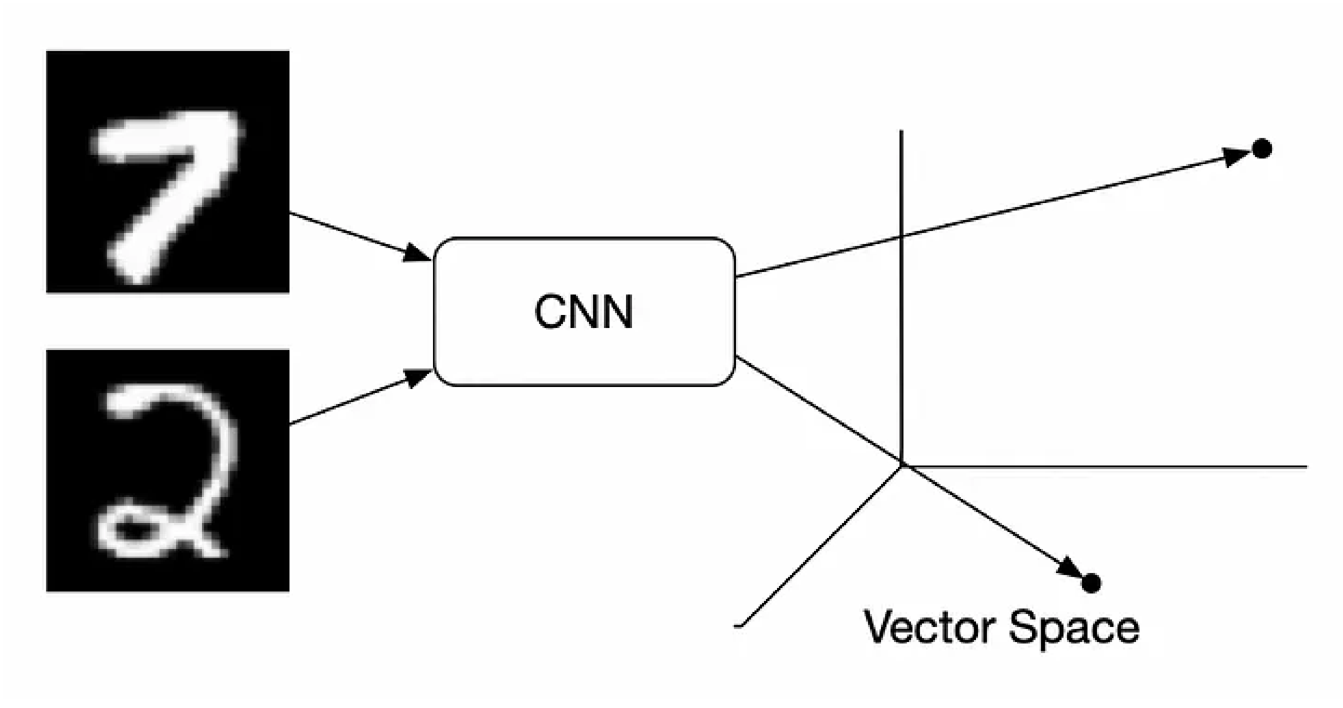
Member-only story

Contrastive Loss Explained

Brian Williams · Follow
Published in Towards Data Science
6 min read · Mar 3, 2020

Listen Share

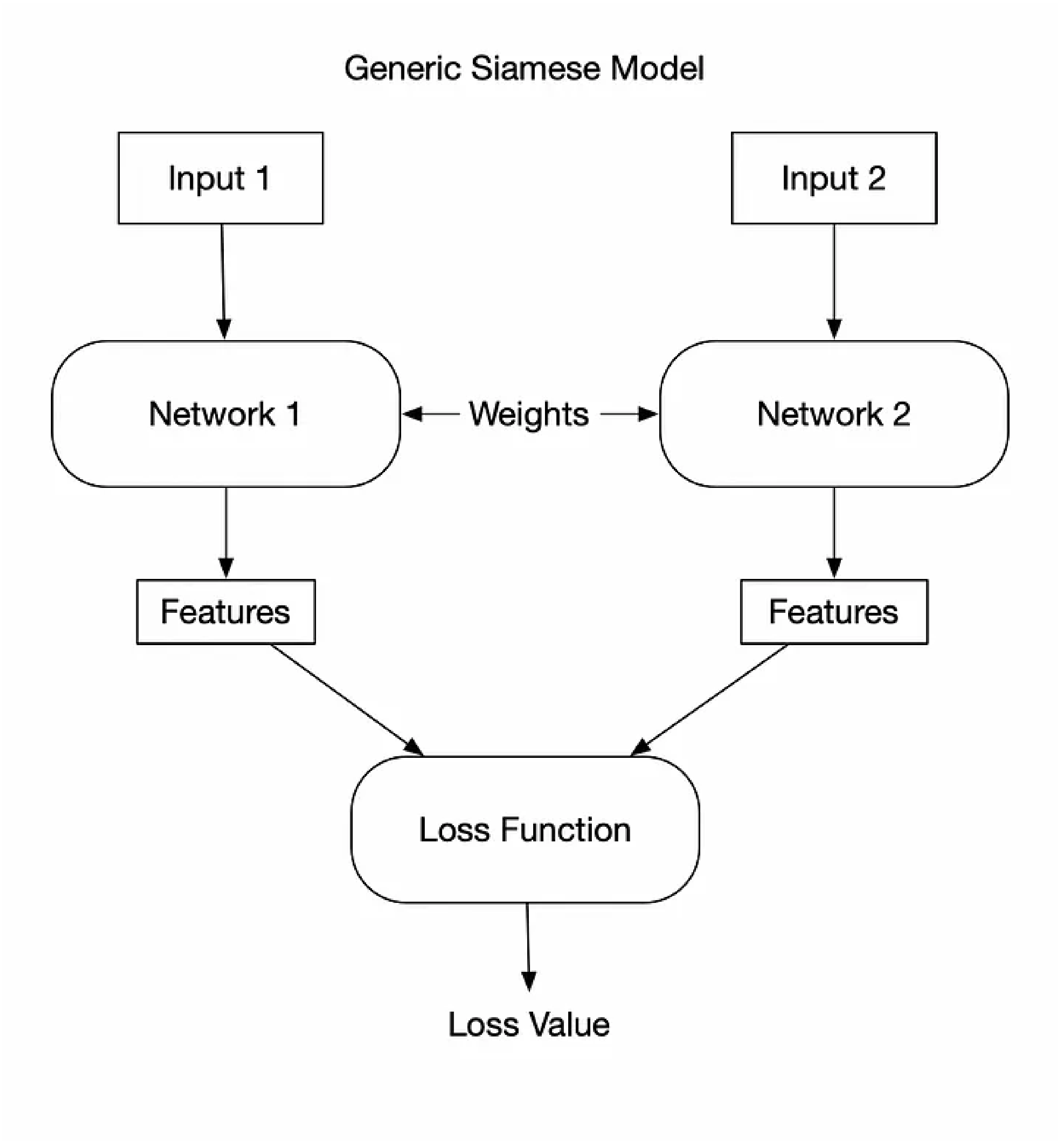


Contrastive loss has been used recently in a number of papers showing state of the art results with unsupervised learning. MoCo, PIRL, and SimCLR all follow very similar patterns of using a siamese network with contrastive loss. When reading these papers I found that the general idea was very straight 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 idea. I will attempt to clearly explain how contrastive loss works and provide a complete implementation using Python and Numpy.

Siamese Networks

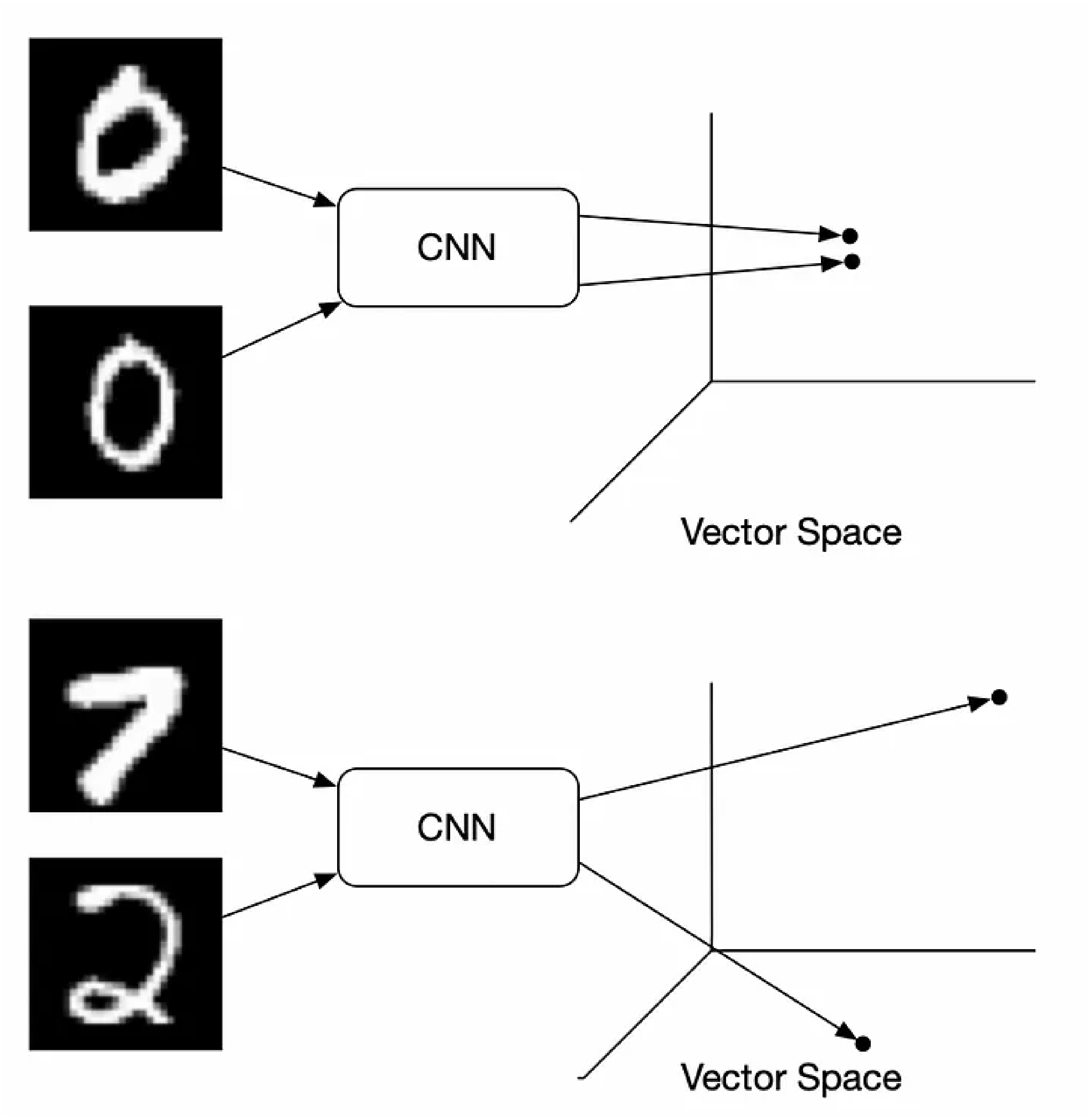
Before digging into the details, it will be helpful to talk about siamese networks (also called **twin networks** but it's not a widely used term). When training a siamese network, 2 or more inputs are encoded and the output features are compared. This comparison can be done in a number of ways. Some 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.



Example

Let's look at an example where we want to make features out of MNIST numbers. 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 data, as is usually the case. There is much more unlabeled data than labeled data available in the world. This is where contrastive loss comes in.

Contrastive loss takes the output of the network for a positive example and calculates its distance to an example of the same class and contrasts that with the 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]])
```

Please note that I have used pre-normalized vectors (aka unit vectors) here.

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 effective measure. The cosine distance measures the cosine of the angle between the vectors. The cosine of identical vectors is 1 while orthogonal and opposite vectors are 0 and -1 respectively. More similar vectors will result in a larger number. Calculating the cosine distance is done by taking the dot product of the vectors. *When not using unit vectors, you would have either normalize the vectors or divide but the product to the normed vectors.*

```
# 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)
softmax_out -> [0.4296791, 0.0633071, 0.2145353, 0.1083572,
0.1013523, 0.0827687]
```

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.

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

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))
loss -> 4.9068650660314756e-05
```

That's all there is to it. Contrastive loss can be implemented as a modified version of cross-entropy loss. Contrastive loss, like triplet and magnet loss, is used to map 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.

Deep Learning Siamese Networks Loss Function Contrastive Loss Softmax

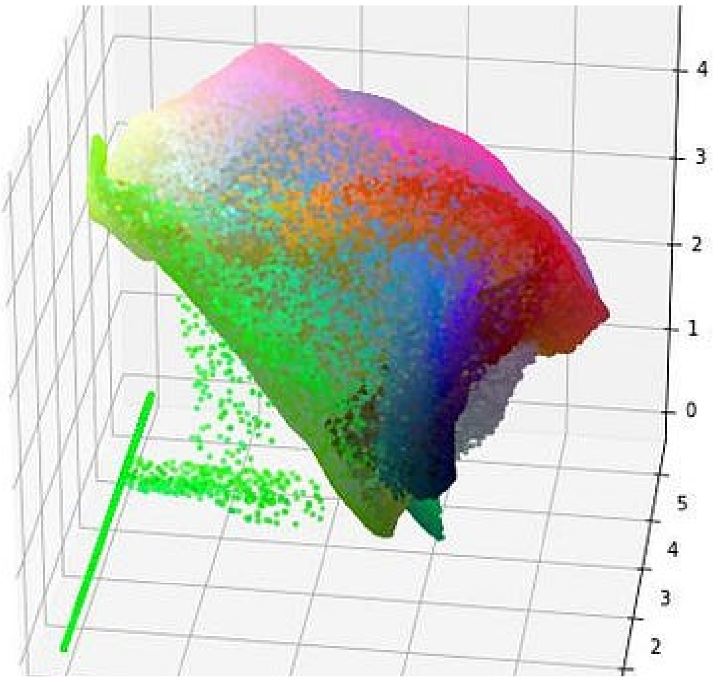
Follow

Written by Brian Williams

115 Followers · Writer for Towards Data Science

Machine learning practitioner, software engineer, lover of well designed things. I solve technical problems in the service of better user experience.

More from Brian Williams and Towards Data Science



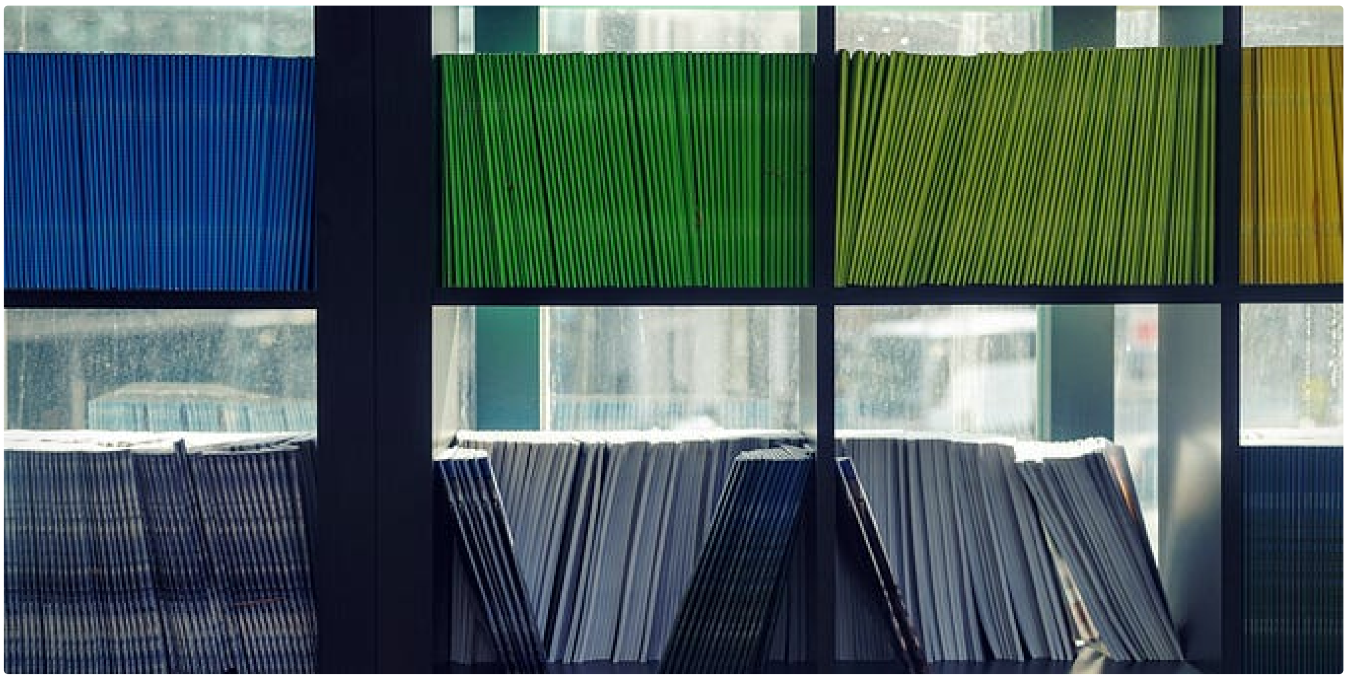
Brian Williams in Towards Data Science

Do androids see electric blue? — visualizing the latent space of color perception

TL;DR

7 min read · Sep 6, 2018

108 1



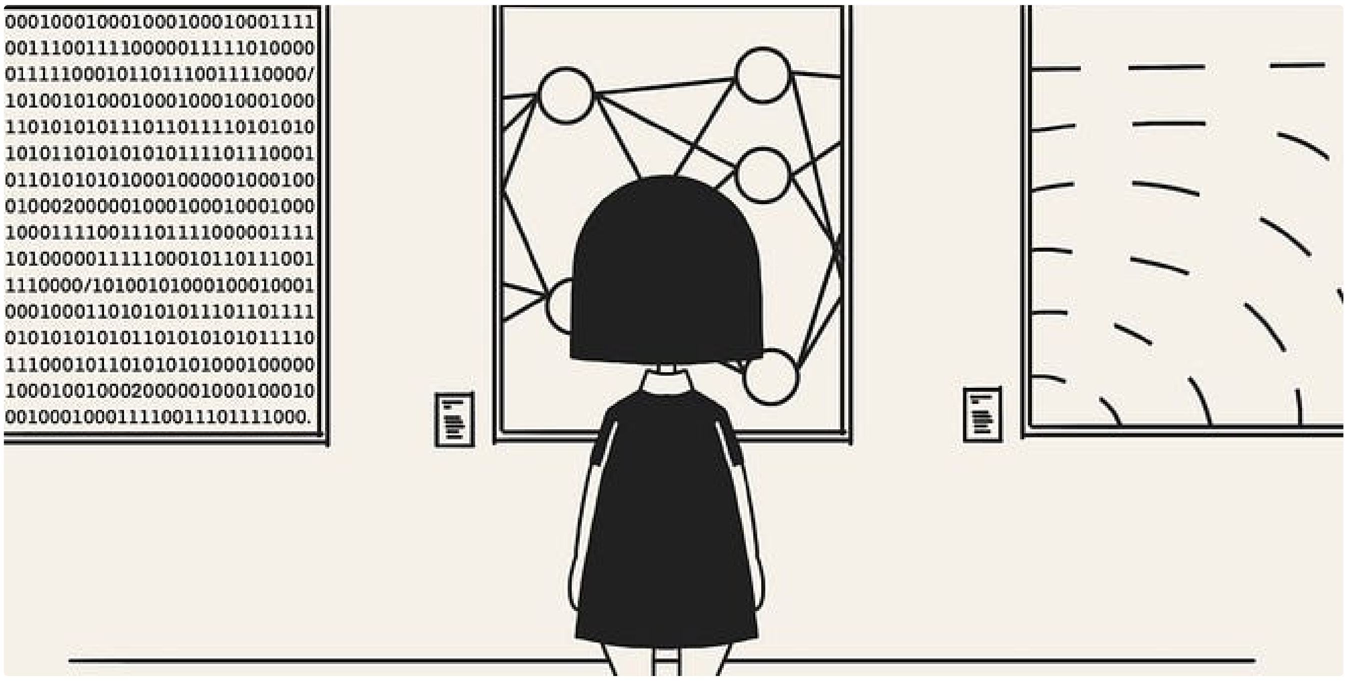
Jacob Marks, Ph.D. in Towards Data Science

How I Turned My Company’s Docs into a Searchable Database with OpenAI

And how you can do the same with your docs

15 min read · Apr 25

3.7K 47



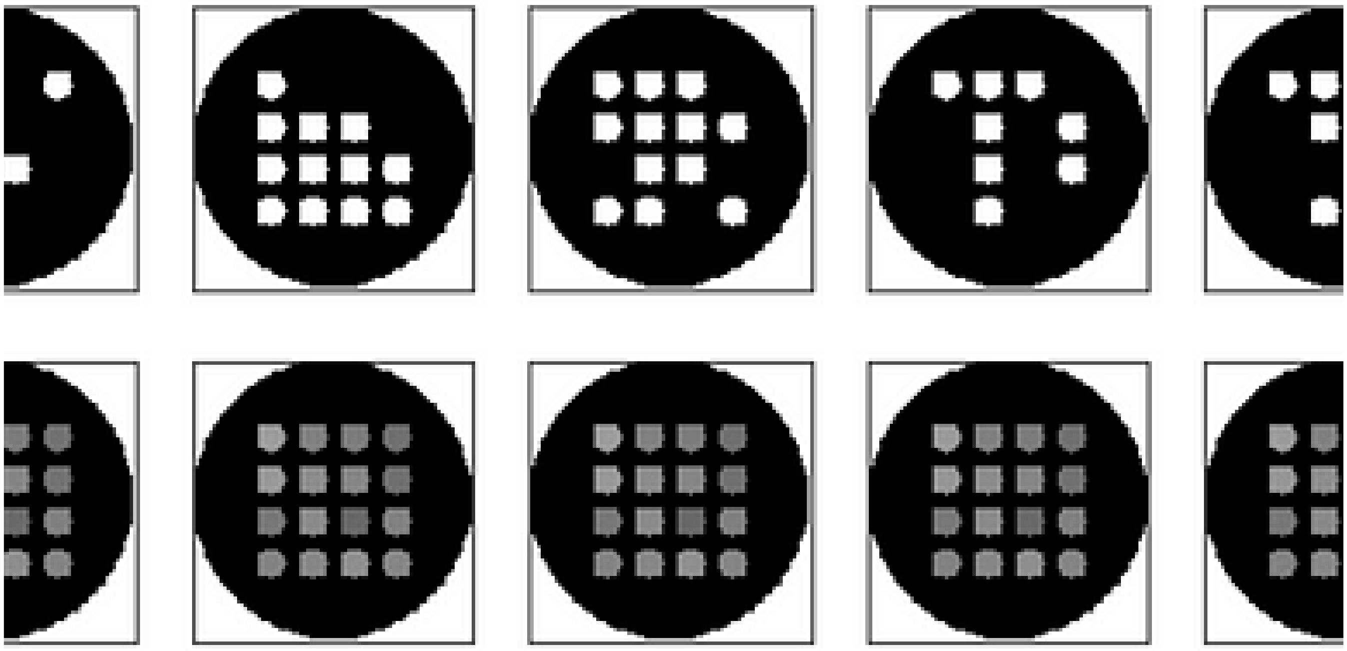
Leonie Monigatti in Towards Data Science

10 Exciting Project Ideas Using Large Language Models (LLMs) for Your Portfolio

Learn how to build apps and showcase your skills with large language models (LLMs). Get started today!

· 11 min read · May 15

1.2K 7



Brian Williams in Towards Data Science

25 Lights

Part I: Problem Introduction and Simple Solution

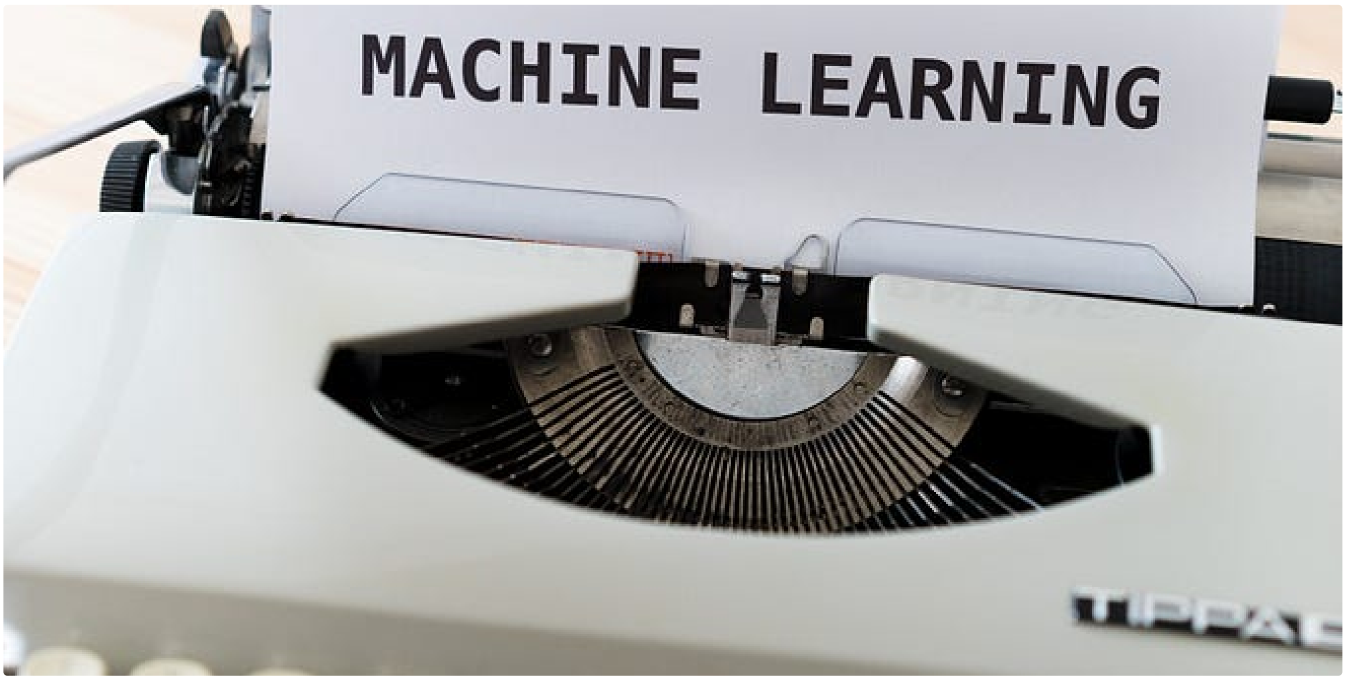
4 min read · Oct 2, 2017

219

See all from Brian Williams

See all from Towards Data Science

Recommended from Medium



Marco Sanguineti in Towards Data Science

Implementing Custom Loss Functions in PyTorch

Understanding the theory and implementation of custom loss functions in PyTorch using the MNIST dataset

· 12 min read · Jan 17

10 2

Konstantin Rink in Towards Data Science

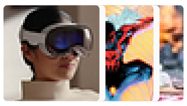
Mean Average Precision at K (MAP@K) clearly explained

One of the most popular evaluation metrics for recommender or ranking problems step by step explained

· 7 min read · Jan 18

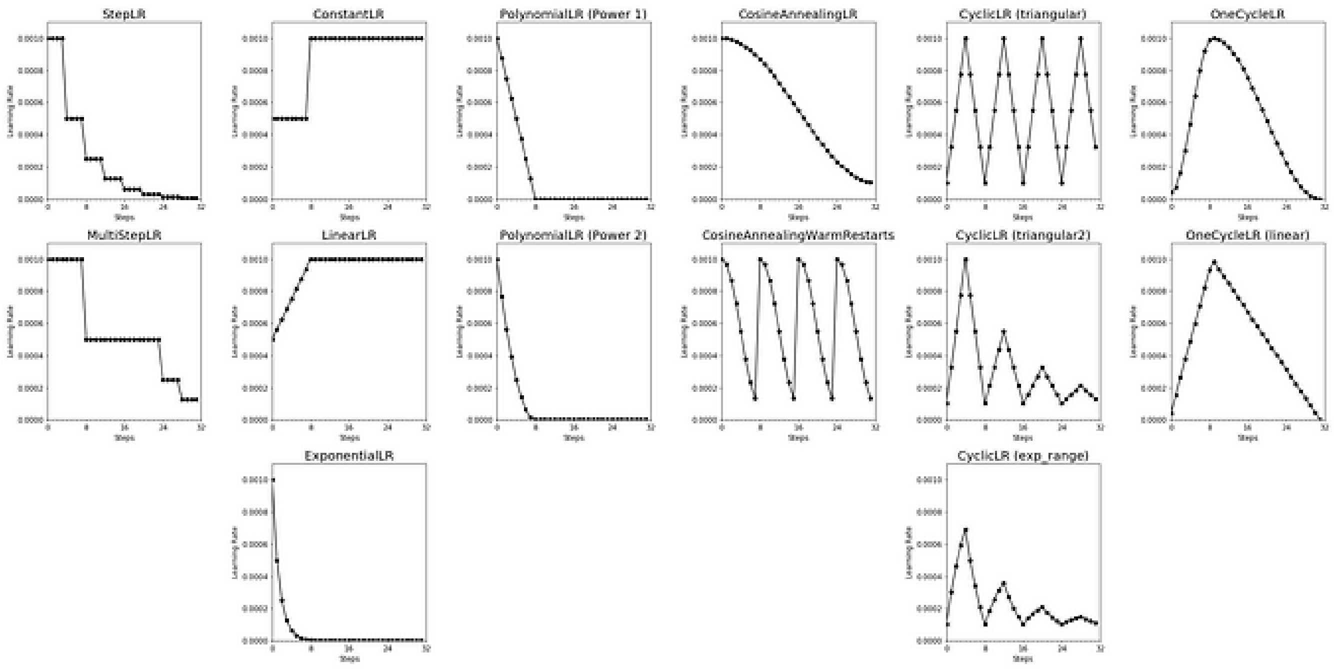
43 5

Lists



Staff Picks

342 stories · 98 saves



Leonie Monigatti in Towards Data Science

A Visual Guide to Learning Rate Schedulers in PyTorch

LR decay and annealing strategies for Deep Learning in Python

· 9 min read · Dec 7, 2022

1K 6

Steins

Diffusion Model Clearly Explained!

How does AI artwork work? Understanding the tech behind the rise of AI-generated art.

· 7 min read · Dec 26, 2022

1.2K 4

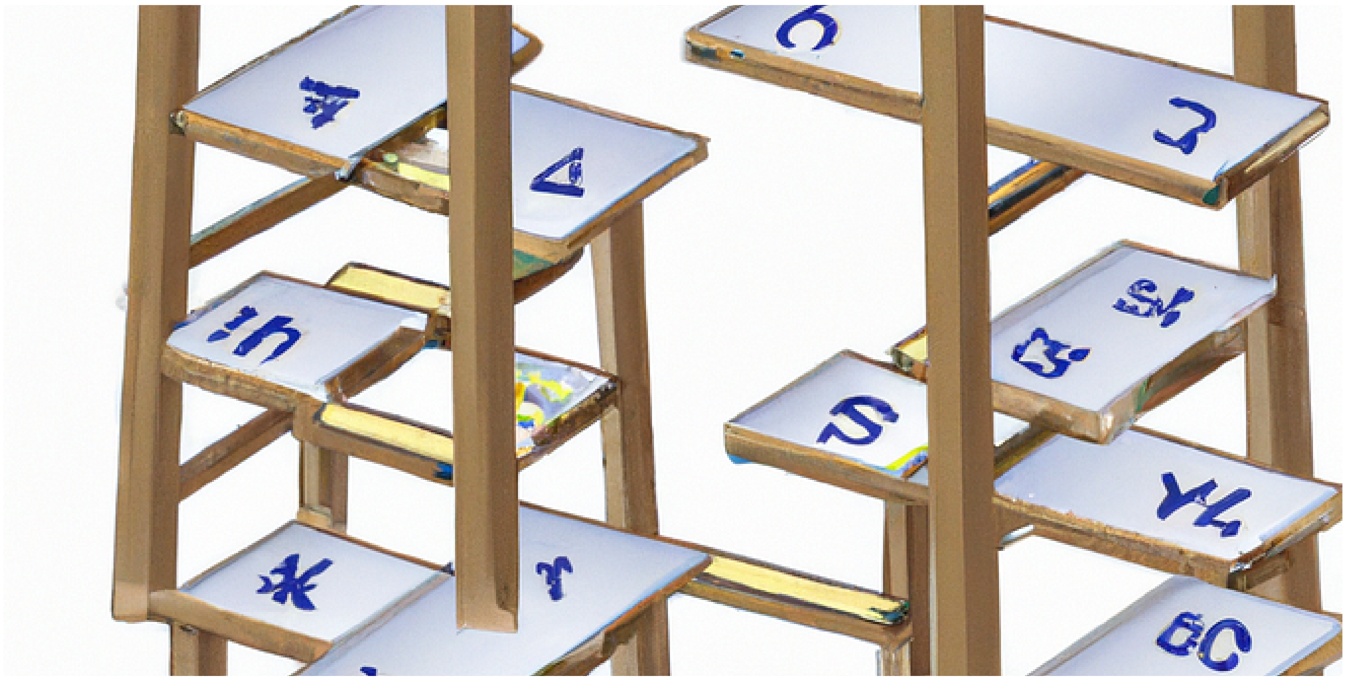
Dr. Robert Kübler in Towards Data Science

Introduction to Embedding-Based Recommender Systems

Learn to build a simple matrix factorization recommender in TensorFlow

· 13 min read · Jan 26

385 4



Will Badr in Towards Data Science

The Secret to Improved NLP: An In-Depth Look at the nn.Embedding Layer in PyTorch

Dissecting the `nn.Embedding` layer in PyTorch and a complete guide on how it works

· 8 min read · Jan 25

124 2

See more recommendations