Published in Towards Data Science

Angel Igareta Follow
Jul 28, 2021 · 4 min read · Listen

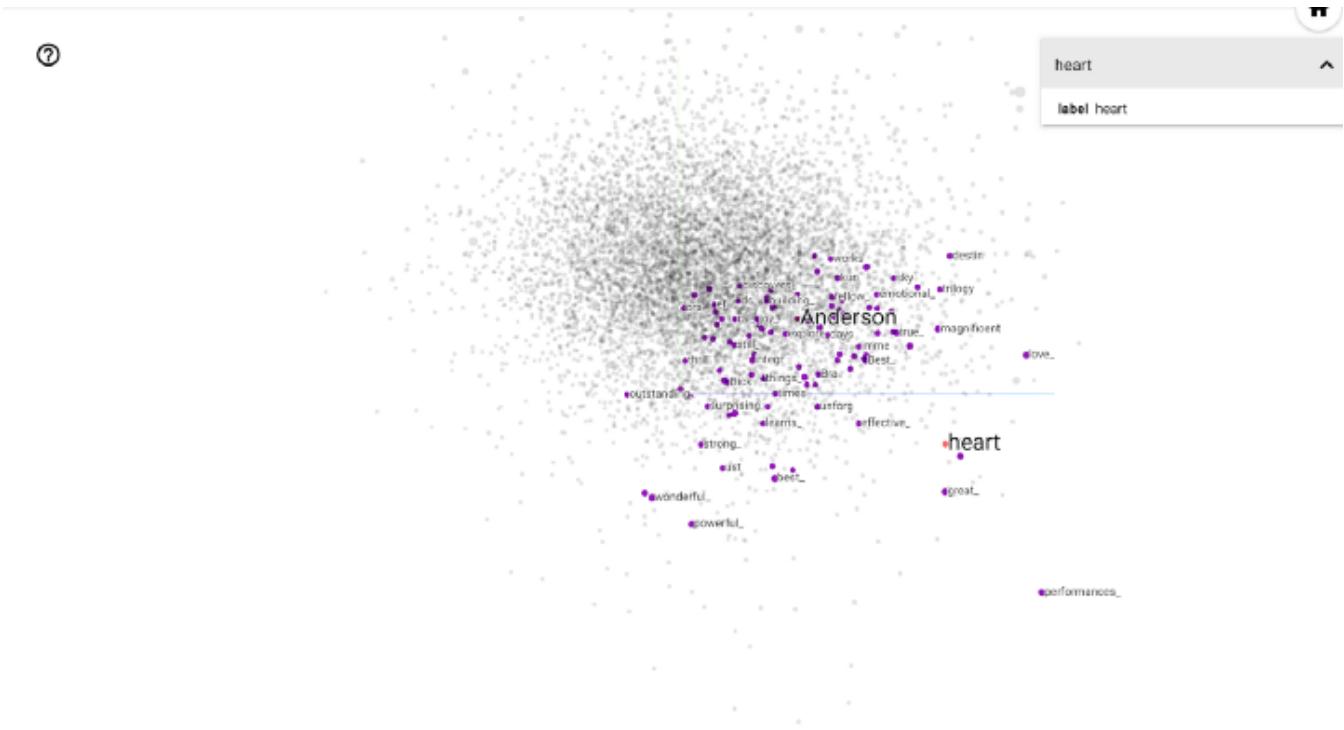
Taking the TensorBoard Embedding Projector to the Next Level

TensorBoard Projector allows to graphically represent low-dimensional embeddings. Here I show you how, instead of displaying a point, you can render the image to which the embedding refers.

to do so, first, it applies a dimensionality reduction algorithm to the input embeddings, between UMAP, T-SNE, PCA, or a custom one, to reduce their dimension to three and be able to render them in a three-dimensional space.

Once the map is generated, this tool can be used, for example, to search for specific keywords associated with the embeddings or highlight similar points in space. Ultimately, its goal is to provide a way to better interpret the embeddings that our machine learning model is generating, to check if the similar ones according to our definition are plotted nearby in the 3D space.

When the embeddings we want to display originate from words, plotting the points with the tag it refers is more than enough, which is exactly the use case presented in the TensorBoard documentation.



Word embeddings projected on TensorBoard. Source: TensorFlow.

However, if we want to project images embeddings, the label, in this case, would not be relevant. On the other hand, if instead of visualizing 3D points we display the images, it would certainly allow us to check if similar images are plotted nearby.

That is exactly the purpose of this tutorial.

Image embeddings projected on TensorBoard.

Implementation

> *Before starting, this tutorial assumes you have a model developed on TensorFlow and a dataset containing the paths of the images used to train/test the model. Projecting embeddings should be the last stage.*

The first step is, in a Jupyter notebook, to load TensorBoard and the extensions we are going to use.

```
%load_ext tensorboard

import csv
import numpy as np
import tensorflow as tf
from PIL import Image
```

Next, we will create a second model that has the same input as our original model but produces a low-dimensional representation. This depends on the architecture used for the original model. Typically, you can use the last layer before applying the activation function, or the one before the last layer if the size of the embedding is smaller.

Once our *embedder* is created, we need to take a sample of images that will be displayed on the TensorBoard projector. The reason is that this tool runs in the browser and is not capable of displaying a large number of images. However, with a sample of about 1000 to 1500 images, the results would already provide high interpretability. In this example, the sample is taken from the validation set.

```
def get_img(img_path):
    img = tf.io.read_file(img_path)
    # convert the compressed string to a 3D uint8 tensor
    img = tf.image.decode_jpeg(img, channels=3)
    # resize the image to the desired size for your model
    img = tf.image.resize_with_pad(img, 100, 100)
    return img

# Generate embeddings
images_pil = []
images_embeddings = []
labels = []
for x in raw_val_ds.take(1500):
    img_path = x[0]
    img_tf = get_img(img_path)

    # Save both tf image for prediction and PIL image for sprite
    img_pil = Image.open(img_path.numpy()).resize((100, 100))
    img_embedding = embeddings(tf.expand_dims(img_tf, axis=0))
    images_embeddings.append(img_embedding.numpy()[0])
    images_pil.append(img_pil)

    # Assuming your output data is directly the label
    label = x[1]
    labels.append(label)
```

To provide the embeddings to TensorBoard, we need to output a *feature_vecs.tsv* **inside the folder we will launch TensorBoard from.**

```
with open(f'{LOG_DIR}/embeddings/feature_vecs.tsv', 'w') as fw:
    csv_writer = csv.writer(fw, delimiter='\t')
    csv_writer.writerows(images_embeddings)
```




```
one_square_size = int(np.ceil(np.sqrt(len(images_embeddings))))
master_width = 100 * one_square_size
master_height = 100 * one_square_size

spriteimage = Image.new(
    mode='RGBA',
    size=(master_width, master_height),
    color=(0,0,0,0) # fully transparent
)

for count, image in enumerate(images_pil):
    div, mod = divmod(count, one_square_size)
    h_loc = 100 * div
    w_loc = 100 * mod
    spriteimage.paste(image, (w_loc, h_loc))

spriteimage.convert("RGB").save(f'{LOG_DIR}/embeddings/sprite.jpg',
transparency=0)
```

Optionally, we can also output *metadata.tsv* to link an image embedding with a certain label (and then be able to search by it). This can be for instance the category of the image.

> *Again, the labels to be in the same order than the embeddings.*

```
with open(f'{LOG_DIR}/embeddings/metadata.tsv', 'w') as file:
    for label in labels:
        file.write(f"{label}\n")
```

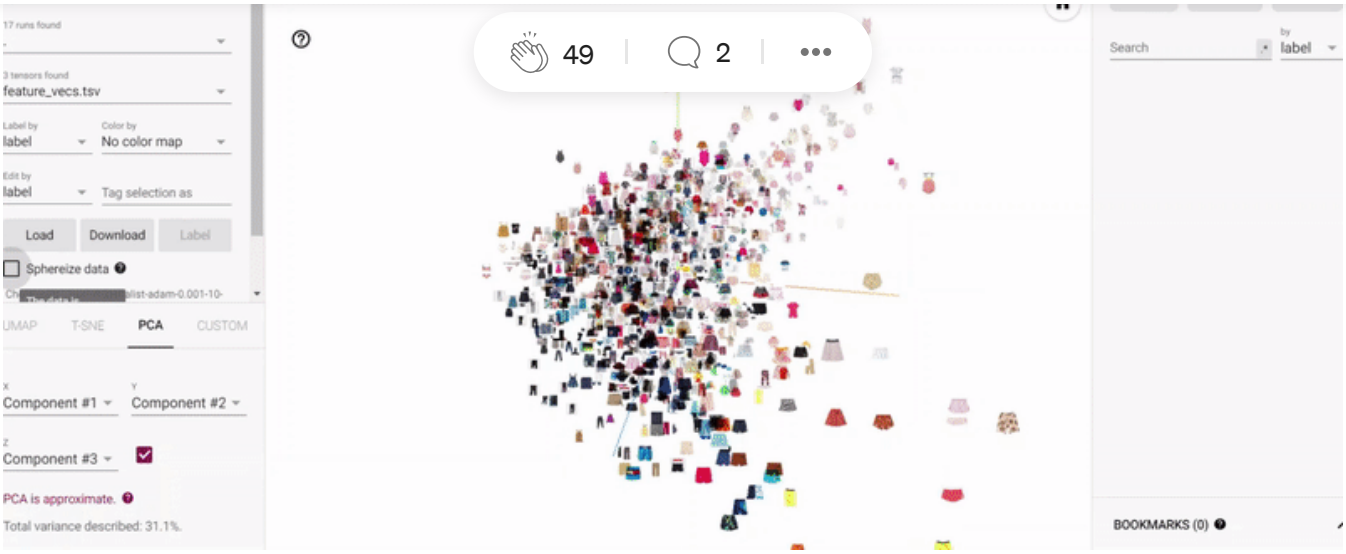
Result

Once the necessary files have been generated, we just need to launch TensorBoard against our log data. This can be done by executing in a cell:

```
%tensorboard — logdir LOG_DIR
```

Here you can see the expected result when selecting the Projector tab in TensorBoard.



> Test between the different dimensionality reduction algorithms until you are satisfied with the projection!

Conclusions

In this tutorial, we have seen how to leverage TensorBoard to not only represent word embeddings but also image embeddings together with the image they refer to. This can provide interpretability insights when analyzing Deep Learning models.

If you want to discover more posts like this one, you can find me at:

- GitHub
- LinkedIn
- Personal Website