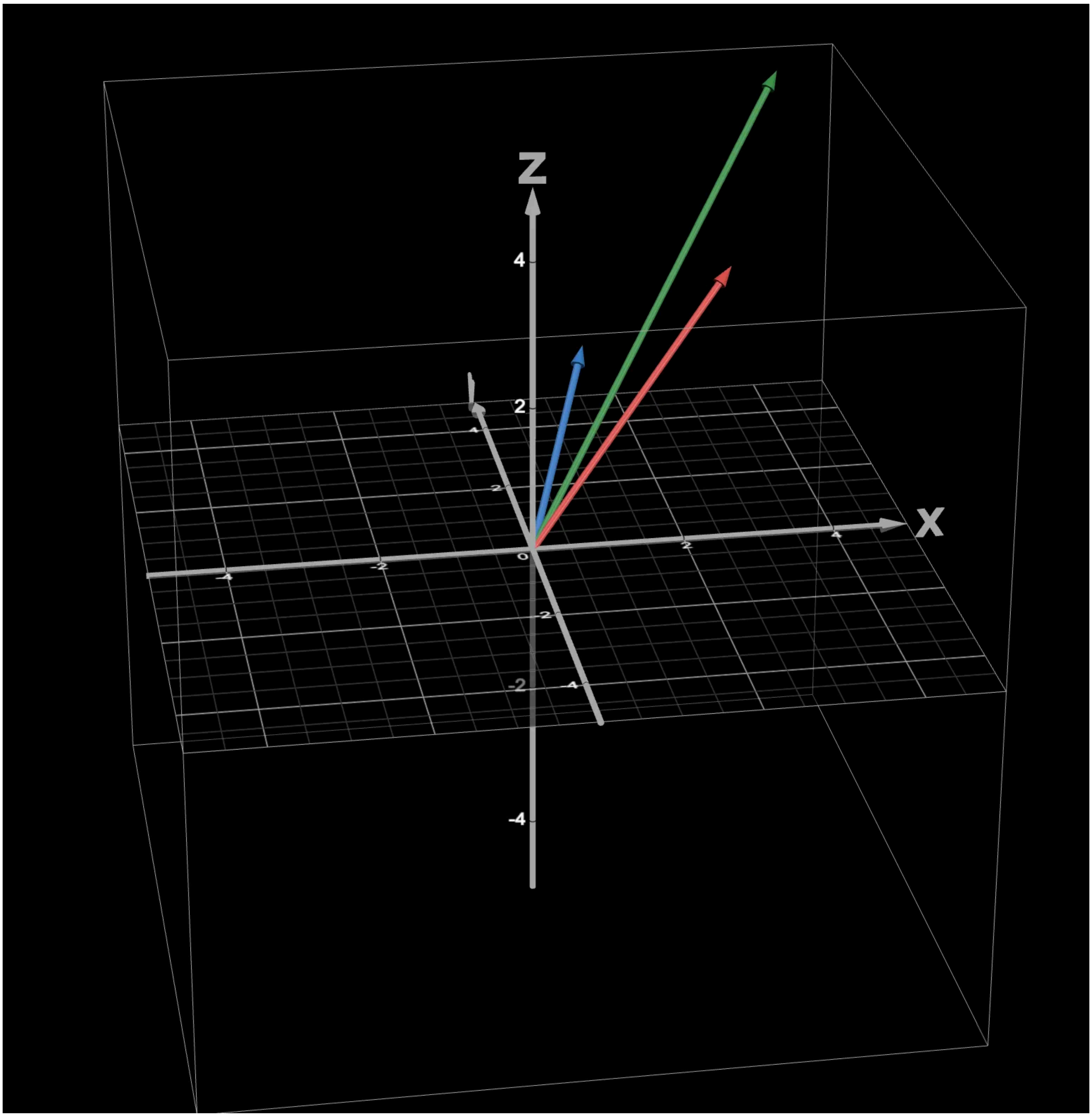
Member-only story

Understanding Vector Space Visually — The Foundation of AI

10 min read · 3 days ago

Asutosh Nayak Follow

Listen Share More



vectors in 3D space

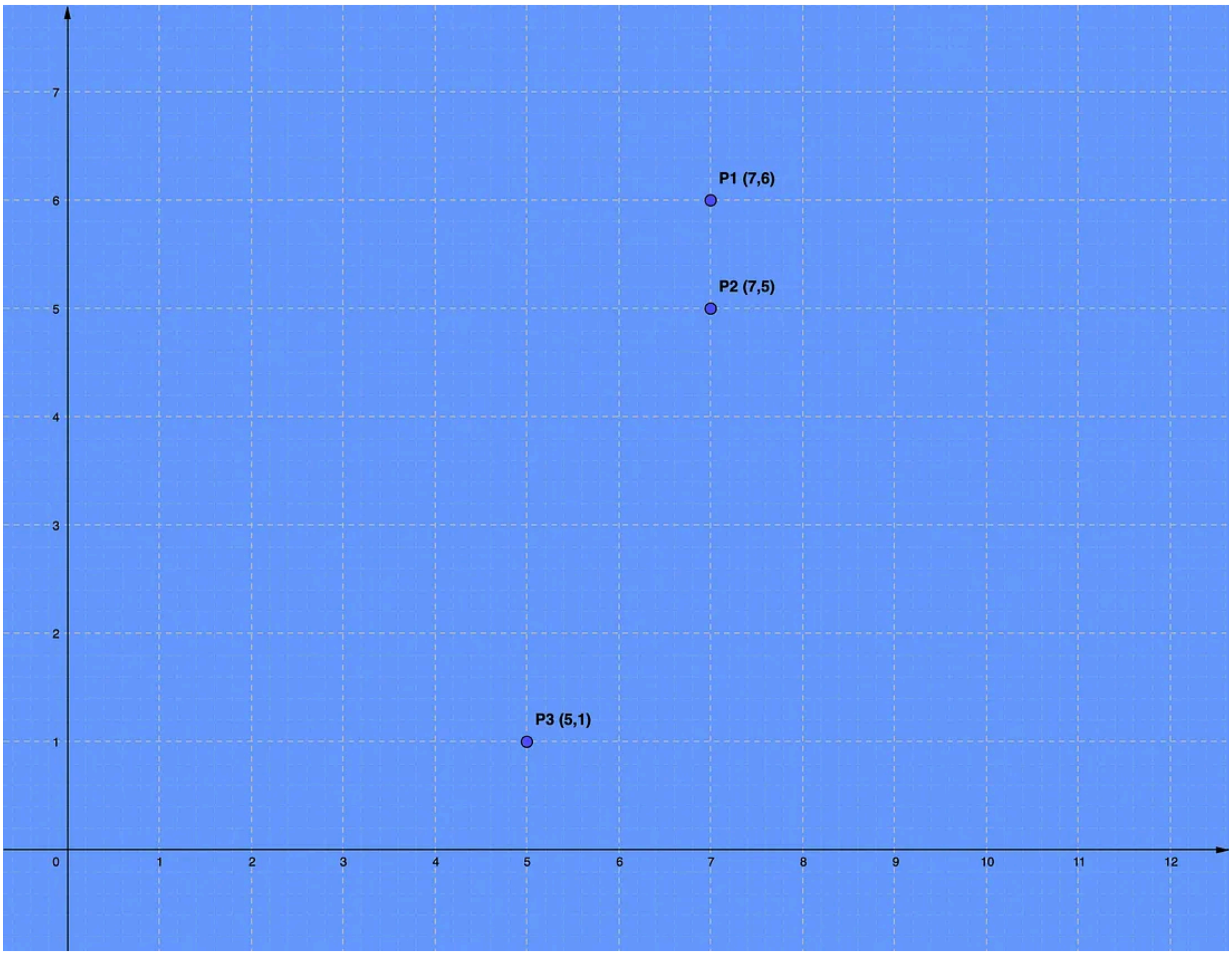
Imagine your friend asked you — "where is the TV remote?" What are the typical responses? "It's in the bedroom, near the table lamp".

Also, imagine someone asked you, which countries out of these are culturally most similar — Japan, India, Sri Lanka? Even if you haven't visited these countries or don't know much about them, your first guess would be to choose the 2 countries which are closest in the world map among the 3 (spoiler: India & Sri Lanka).

In the two examples above, we used two important concepts — **location** & **proximity**. In the first example we used bedroom and table lamp as landmarks to *locate* the remote. In the second example, we used proximity to guess *similarity*. Keep this in mind as we venture further.

Now, let's extend this concept to a 2 dimensional coordinate system. A coordinate system is a framework where we have numbers marked on horizontal (x axis) and vertical lines (y axis) and we can "**locate**" points or lines on it.

Assume we have been given 3 three points — P1 at (x=7, y=6), P2 at (x=7, y=5) and P3 (x=5, y=1). Let's locate them on coordinate system (just like we located TV remote in our home).



locate points in a 2D coordinate system

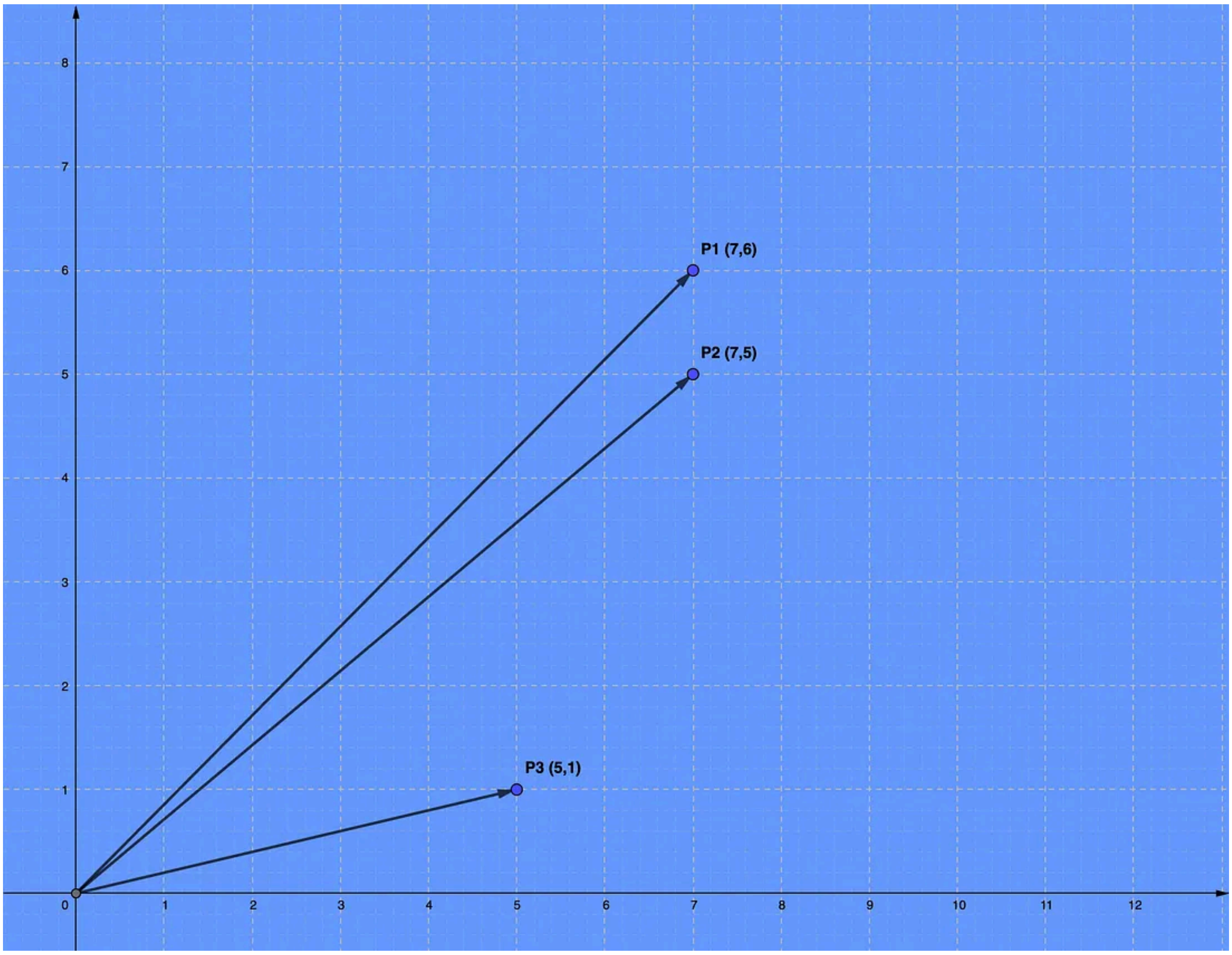
Now that we have located the points, we can easily visualize, which points are closer (see image above).

Vectors

Now, let's represent these points as vectors. But before that, what is a vector?

> *A vector is a mathematical or physical entity that has both magnitude (size or length) and direction. It is often represented graphically as an arrow, where the length of the arrow indicates the magnitude and the arrowhead indicates the direction.*

Our points don't have any magnitude or direction. But these can be represented as a vector if we trace and connect them with *origin* or center of the coordinate system. See below.



P1, P2, P3 as vectors from origin

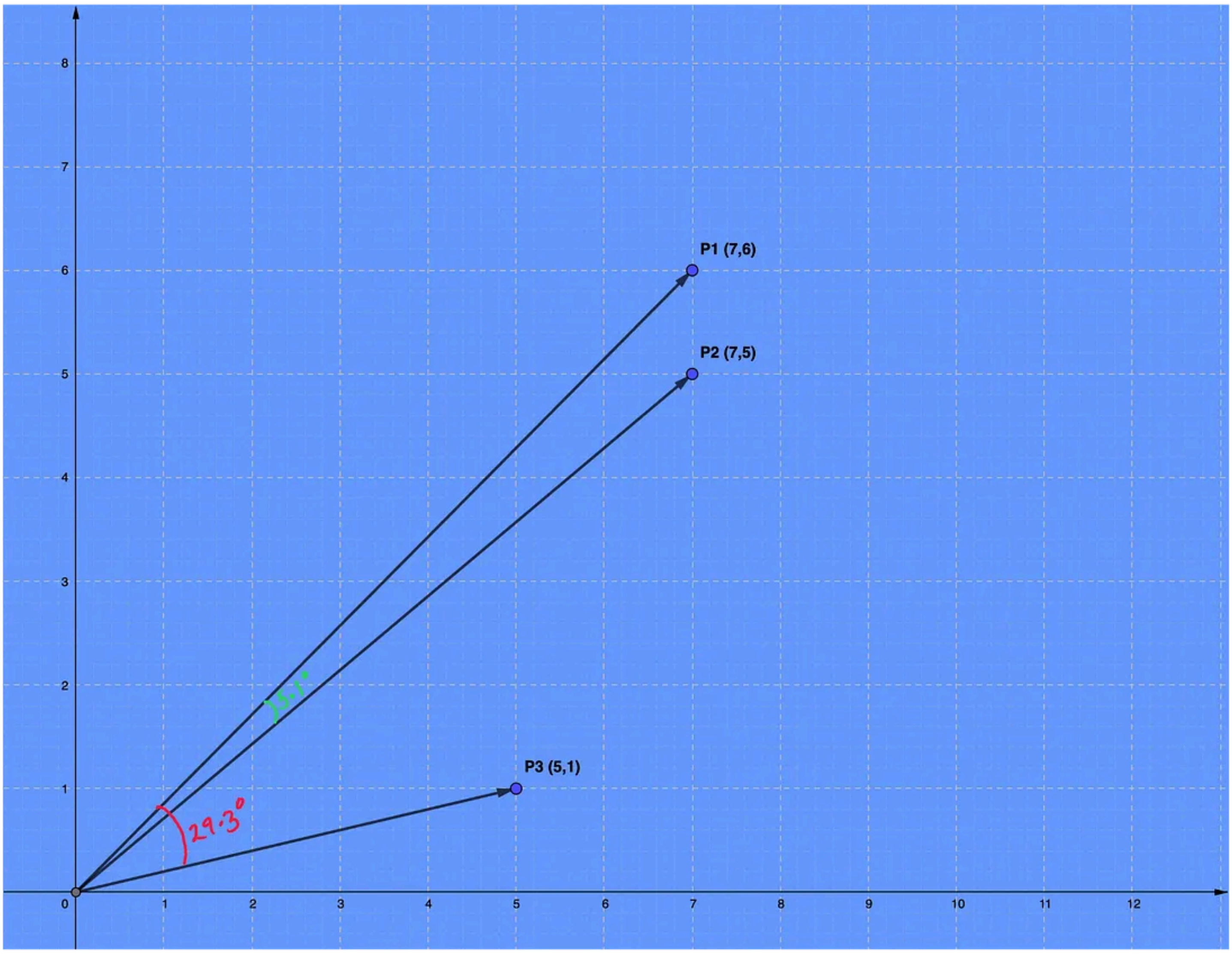
Now we have entities which have some magnitude (length of the line segment from [0,0] to [7,6]) and direction, so our points are mathematically vectors. Vector ending at P1 can be represented by notation OP1 (with arrow on top):

$$\vec{OP}_1$$

Angles Between Vectors

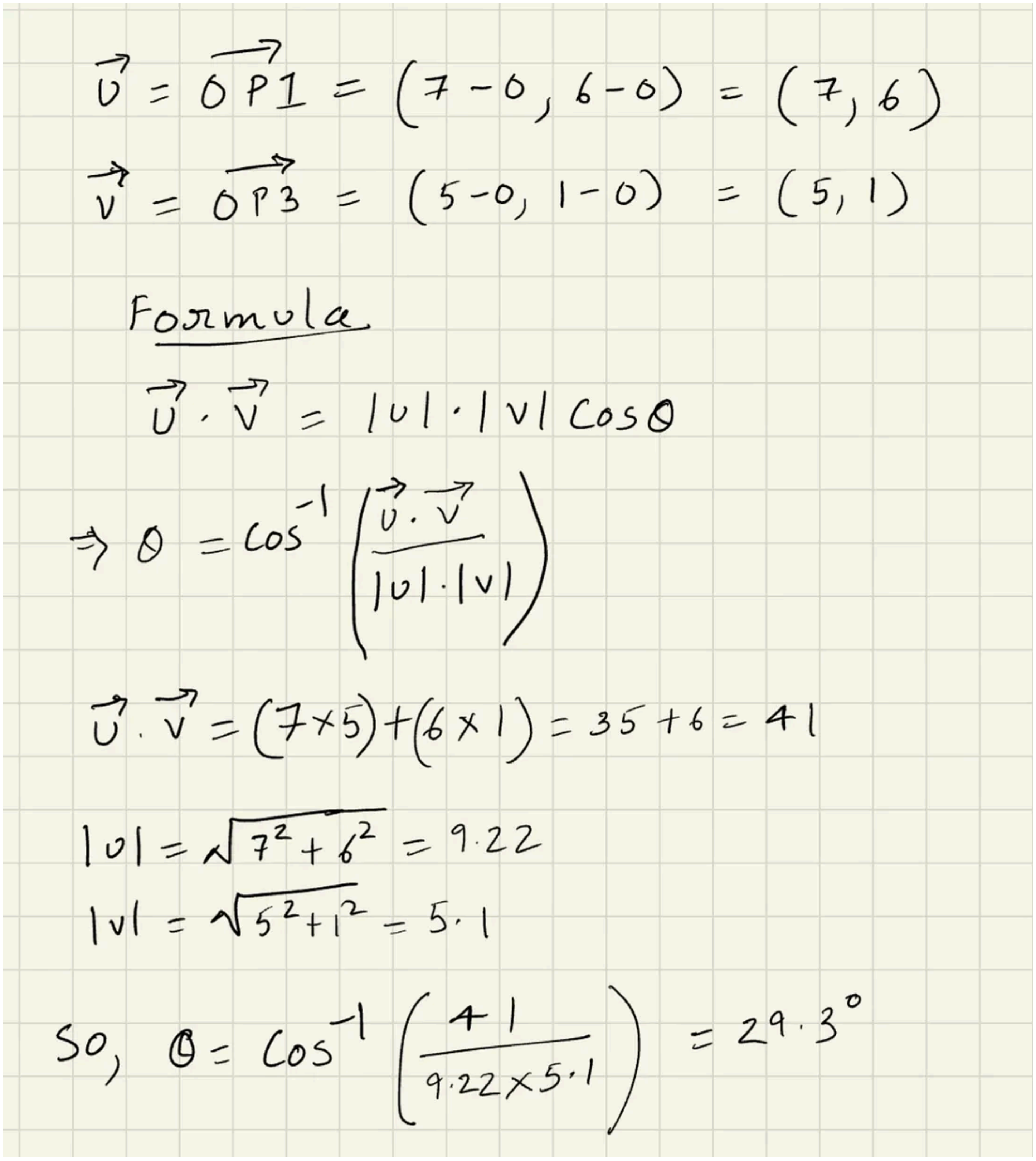
Recap — We have seen how we can locate points on coordinate system and how to represent points as vectors in coordinate system.

Now let's try to visualize which of the 3 vectors are closer to each other. We can easily see that vectors pointing to P1 and P2 are closer than P1 is to P3. Formally, it is determined by the angle formed between the two vectors.



As shown above, the angle between vector of OP1 & OP3 is much greater than OP1 & OP2. We can conclude with certainty that vectors OP1 and OP2 are much more similar.

The angle between two vectors (arrows) is how far apart they point. If they point exactly the same way, the angle is 0 degree. If they point in exactly opposite directions, it's 180 degrees. Finding angle between two vectors is high school level math and internet is rife with explanation. But in case you need a refresher —



$$\vec{u} = \overrightarrow{OP1} = (7-0, 6-0) = (7, 6)$$

$$\vec{v} = \overrightarrow{OP3} = (5-0, 1-0) = (5, 1)$$

<u>Formula</u>

$$\vec{u} \cdot \vec{v} = |u| \cdot |v| \cos\theta$$

$$\Rightarrow \theta = \cos^{-1}\left(\frac{\vec{u} \cdot \vec{v}}{|u| \cdot |v|}\right)$$

$$\vec{u} \cdot \vec{v} = (7 \times 5) + (6 \times 1) = 35 + 6 = 41$$

$$|u| = \sqrt{7^2 + 6^2} = 9.22$$

$$|v| = \sqrt{5^2 + 1^2} = 5.1$$

$$\text{So, } \theta = \cos^{-1}\left(\frac{41}{9.22 \times 5.1}\right) = 29.3^\circ$$

vector angle math workout

Notice how we used origin (0, 0) as source to represent points as vectors in first line. This is fairly common.

> *What's more, instead of calculating "theta" via inverse cosine (AKA arccosine), if we calculate directly cos(theta), it is called* **cosine similarity.**

This is one of the most important metric to find similar vectors along with Eucledian distance (more on this later, when we discuss applications of vectors). I will not traumatise you further with my horrible handwriting.

Different algebraic operations are possible with vectors like addition and scalar multiplication, each having some significance. For instance, addition of vectors (x1, y1) and (x2, y2) is simply (x1+x2, y1+y2)

So, adding vector OP1 and OP3 looks like OR vector:

vector addition

Think of these operations in intuitive way. The next example will give us better intuition:

What should be the result of adding red and green vectors:

adding opposite vectors of equal magnitude

Of course it will result in a “zero” vector — a vector with 0 magnitude and no direction. Work on a few more example and different operations to get a feel.

Similarly, multiplying a scalar (a number with only magnitude and no direction) to a vector amplifies the vector. Below you will see OP vector is multiplied by scalar 2, which gives us OR (4,2) (each dimension is multiplied by the scalar).

Congratulations! You have now worked in a *vector space.* Formally,

> *a vector space is a set of vectors which can be added, or multiplied by a scalar.*

P.S. — All 2D geometry is drawn using GeoGebra. Shout out to the creator for this awesome tool.

Venturing into 3D

We are now comfortable with finding our way in 2D vector space. Let’s step up and explore 3D space. Simply put, 3D space just adds another dimension into our earlier vector space — z axis.

Let’s plot a vector V1 from origin (x=0, y=0, z=0) to (x=3, y=2, z=3) [**Note**: going further I would omit origin and axis labels and represent vectors as array of numbers like— (3,2,3) for brevity]. This would look like this in 3D (shout out to the author of desmos):

3D vector visualization

All the other operations we saw earlier with 2D vectors remain the same conceptually (other than the fact that there is an additional z-axis value). For instance, if we add vectors V1 (3,2,3) & V2 (1,2,2) in the video below, then we get vector R (4,4,5), which would look like this in 3D space (Result is green colored vector):

Beyond 3D

Unfortunately, human mind is limited to 3 dimensions, so I can't efficiently animate 4D space (nor I'm expert in graphics). But human mind is excellent at extrapolating concepts. So you can imagine 4D as several 3D spaces stacked. In real-world applications, vectors are often several hundred dimensions large (and that's where the magic happens), so there is no point trying to visualise them.

Most of the times large vectors are converted into smaller size vectors (often 2D) to visualize. This is called *Dimensionality Reduction.* (example code will be seen later here)

Let's get to our main agenda now and see how these vectors are applied to Machine Learning/AI problems and understand.

AI and Vectors

Machines are good with numbers but they cannot process unstructured data like images or texts in their raw form. So, such data is converted into large vectors, called "***embeddings***", and fed into models (such as Neural networks, Logistic Regression or K-Means) as inputs to train the model. Also, vector operations in batches are highly optimal in GPU, which further helps us train large models.

For checking out vectors in applications, we will use image data. This is what we will do:

1. Train a Small CNN on MNIST (PyTorch) — We'll use PyTorch and torchvision to load MNIST and train a simple CNN.

2. Extract Embeddings from an Intermediate Layer — We'll grab outputs from the last-but-one layer as embeddings.

3. Compare Embeddings for Different Digits — We'll write a function to compare embeddings (using Euclidean distance).

4. Dimensionality Reduction & Plotting — We'll use PCA or t-SNE for visualization.

The full code, along with outputs can be found in my Kaggle notebook. To keep this blog short (although at this point it's a lost cause), I won't paste the full code here. I will just share the important parts of code and show the output/findings. I would encourage interested readers to view the notebook and play around.

Here is how we can get the output of intermediate layer of a neural network in Pytorch to use it as embeddings for downstream tasks:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# CNN Model Definition
def get_cnn_model():
    class SmallCNN(nn.Module):
        def __init__(self):
            super().__init__()
            self.conv1 = nn.Conv2d(1, 16, 3, padding=1)
            self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
            self.pool = nn.MaxPool2d(2, 2)
            self.fc1 = nn.Linear(32*7*7, 64)        # Embedding layer (last but
            self.fc2 = nn.Linear(64, 10)            # Output layer

        def forward(self, x):
            x = self.pool(F.relu(self.conv1(x)))
            x = self.pool(F.relu(self.conv2(x)))
            x = x.view(x.size(0), -1)
            embed = F.relu(self.fc1(x))                    # Save embedding
            out = self.fc2(embed)
            return out, embed
    return SmallCNN()

transform = transforms.ToTensor()
train_set = datasets.MNIST(root='./data', train=True, download=True, transform=
train_loader = DataLoader(train_set, batch_size=64, shuffle=True)

cnn = get_cnn_model()
optimizer = optim.Adam(cnn.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()


# Train the model and then get embeddings like shown below (full code at given

# Get test data
test_set = datasets.MNIST(root='./data', train=False, download=True, transform=
test_loader = DataLoader(test_set, batch_size=256, shuffle=False)

# Get embeddings and labels
embeddings = []
labels = []
cnn.eval()
with torch.no_grad():
    for images, lbls in test_loader:
        _, embed = cnn(images)
        embeddings.append(embed)
        labels.append(lbls)
embeddings = torch.cat(embeddings).numpy()
labels = torch.cat(labels).numpy()
```

Once we have the embeddings we can compare the embeddings and visualize like this:

```
def compare_digit_embeddings(embeddings, labels, digit1, digit2, num_per_digit=
    idx1 = np.where(labels == digit1)[0][:num_per_digit]
    idx2 = np.where(labels == digit2)[0][:num_per_digit]
    emb1 = embeddings[idx1]
    emb2 = embeddings[idx2]
    # Compute pairwise distances within each digit and between digits
    intra1 = cdist(emb1, emb1).mean()
    intra2 = cdist(emb2, emb2).mean()
    inter = cdist(emb1, emb2).mean()
    print(f"Avg distance within {digit1}: {intra1:.2f}")
    print(f"Avg distance within {digit2}: {intra2:.2f}")
    print(f"Avg distance between {digit1} and {digit2}: {inter:.2f}")
    return emb1, emb2

import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

def plot_embeddings(emb1, emb2, digit1, digit2):
    # Concatenate and reduce to 2D
    all_emb = np.vstack([emb1, emb2])
    pca = PCA(n_components=2)
    emb_2d = pca.fit_transform(all_emb)
    # Plot
    plt.figure(figsize=(6, 6))
    plt.scatter(emb_2d[:len(emb1), 0], emb_2d[:len(emb1), 1], label=f"{digit1}"
    plt.scatter(emb_2d[len(emb1):, 0], emb_2d[len(emb1):, 1], label=f"{digit2}"
    plt.legend()
    plt.title(f"Embeddings: {digit1} vs {digit2}")
    plt.xlabel("Component 1")
    plt.ylabel("Component 2")
    plt.show()

# images we want to compare embeddings
n1 = 3
n2 = 6
emb1, emb2 = compare_digit_embeddings(embeddings, labels, n1, n2)
plot_embeddings(emb1, emb2, n1, n2)
```

The output looks like this:

output: mean distance between embeddings and visualization

As you can see, all the embeddings (after dimensionality reduction) of images of 3 lie closer to each other than embeddings of images for 6. You can also see the average Euclidean distance (you could use cosine similarity which we discussed above too, since these can be represented as vectors starting from origin!) between images of 3 and 6 is significantly larger than distances of images of same numbers.

Understand that, these large 64 dimensional embeddings are in real a point in 64 dimension vector space. Each image input is mapped to a point in this space and that’s how the model learns to classify it depending on how close/far they are to each other. So, we have applied the concept of location and proximity from the start of this article. How far we have come from our 2D example!

You may ask why is this exciting? This signifies that the model has successfully "*understood"* the images/data and found a good way to represent them in vector space. You can use these embeddings to feed into another model to train for another task (**like generating text from image embeddings!**).

The better the quality of the embeddings, the better is the performance of the model. In case of NLP and textual data, each word (or subword, depending of tokenization method used) is represented as embeddings. Models use these to understand meaning of the text input and perform tasks like machine translation, text to image etc.

But, there are a few things you should take care of too. One of them is choosing the right size of embeddings for you model/task (in above example we set it as 64, the output of last but one layer). If it's too short, the model may not be able to capture all the intricacies of the data, and thus final prediction/generation may be way off from ground truth. On the other hand, if the embeddings are too large, the model may overfit on the dataset and again, the output would suffer. Also, the memory and time needed to train the model would increase. This is fondly called the "***curse of dimensionality***" in the scientific community.

As an exercise, try tweaking the value of output dimension in the above code and notice the difference.

Afterword

I hope I have covered all the major concepts one may need to understand and appreciate the role of vectors in what's going on in AI world currently.

Right from K-Means to the largest LLMs, everything is powered by these unsung heroes.

If this has helped you and you liked it, don't forget to clap, or say hi in comments.

Machine Learning · Artificial Intelligence · Data Science · Technology · AI

Follow

Published in The Muse Junction

76 followers · Last published 3 days ago

A junction for ideas, from technology to the art of storytelling

Follow

Written by Asutosh Nayak

956 followers · 54 following

Machine Learning | Android | Bibliophile | Aspiring Writer. Feel free to connect on LinkedIn
https://www.linkedin.com/in/nayakasu92/

Responses (2)

Bgerby

What are your thoughts?

See all responses