Quantum Machine Learning for cybersecurity, Part 1 — data preprocessing:

Marek Kowalik · Follow

7 min read · Sep 7, 2022

DDoS-type attacks recognition with Hybrid Quantum Neural Network (H-QNN).

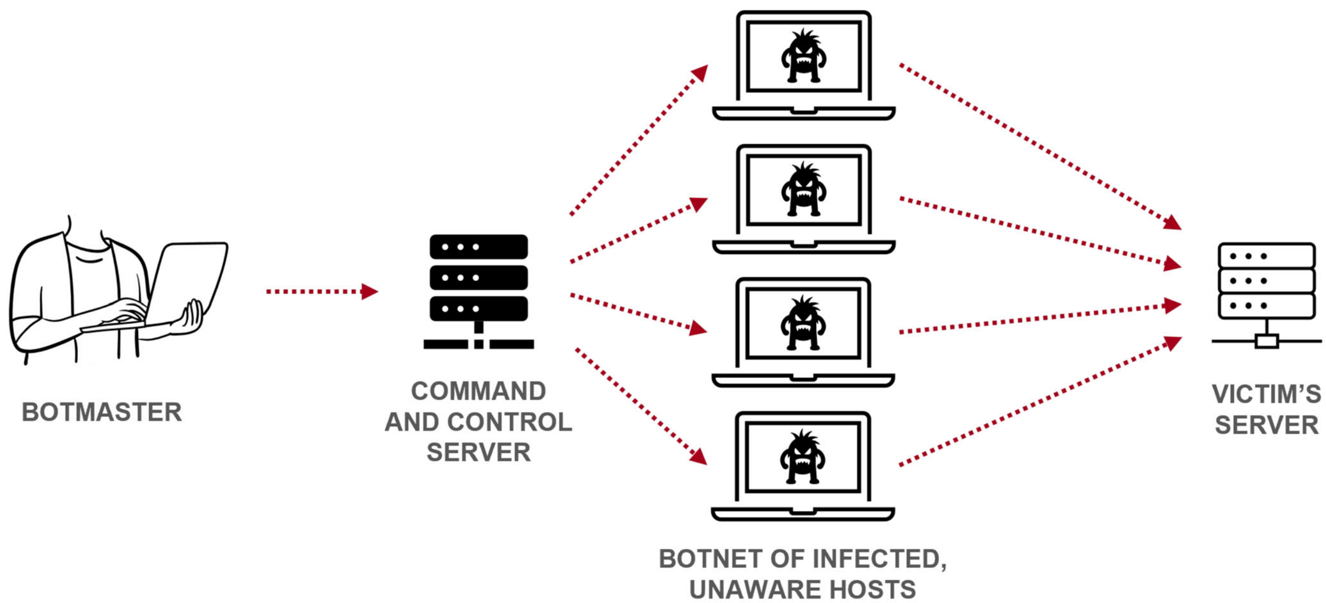
Introduction:

Imagine you're trying to access your government's website. You just wanted to print some form or read about law changes, but the site doesn't load. It's possible, that the servers are under the cyberattack. How can we prevent that scenario? With Machine Learning methods, of course. They are irreplaceable in the cybersecurity field by capturing data nuances, imperceptible by humans. In this short series, we'll talk about detecting a DDoS-type attack with a new technology: quantum computing.

Quantum computing offers alternative computing ways by using quantum phenomena. Recently, there is a lot of hype around this technology and quite a few Quantum Machine Learning (QML) algorithms appeared. However, finding and training Quantum Machine Learning models that present high level of performance is not an easy task. Finding a QML model that obtains this level of performance on a real dataset is truly outstanding. Let's take a look at one of these models and its skip all data cleaning, go to the next part where we implement and train the actual model on this data.

DDoS attacks:

Distributed Denial of Service (**DDoS**) is a type of cyberattack, where many hosts are trying to connect with a victim's server, until it crashes and is unable to process a legitimate request. It's coordinated action from one central point, usually performed with malicious software, which infects devices of unaware owners:



Recognizing this type of attack is important in the early stages of connection to the server. This prevents the attackers from taking resources, swamping, and finally shutting down the website or application. We need small, robust models to quickly classify, for example, a user request as benign or potential DDoS, without slowing down the whole process.

Further reading: What is a Distributed Denial-of-Service Attack?

- go to the dataset website and click on the download
- submit your information to get the access
- go to `CSVs` directory,
- download `CSV-01-122.zip` file

After that, unpack and find in `01-12` directory `DrDoS_SSDP.csv` file. That's our data.

Before jumping to coding, make sure you've got set up environment. Here's our stack:

Data preprocessing	Quantum Part of the H-QNN	Classical Part of the H-QNN	Evaluation	Visualization
pandas NumPy	PENNYLANE	TensorFlow	scikit learn	matplotlib seaborn

Let's begin with importing all needed packages:

```
# Data processing:
import numpy as np
import pandas as pd

# Utils:
import sklearn.decomposition
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Visuals:
import seaborn as sns
import matplotlib.pyplot as plt
```

```
comp_dtype = 'float32'

# Increase pandas printing limits
pd.set_option('display.max_columns', 90)
pd.set_option('display.width', 1000)
pd.set_option('display.max_rows', 90)
```

Let's see the size of our data:

```
data_read_path = os.path.join(".", "data", "raw", "CSV-01-12", "01-12", "DrDoS_SSDP.csv")
data_df = pd.read_csv(data_read_path)
data_df.drop(labels=['Unnamed: 0'], axis=1, inplace=True)
data_df.shape
```

```
(2611374, 87)
```


- **Columns with only one value**
- **'Irrelevant' columns** — timestamp, destination port, source ID. We can use more sophisticated methods to extract useful information (e.g. labeling data points by the hour from timestamps, to see time of day correlation with attacks), but for simplicity reasons, we'll leave that. However, the reader is free to experiment. Possibly, it can help to train the model.
- **Columns with 'NaN' values**
- **Correlated columns** — We'll get rid of columns with a correlation factor above

```
    if column.isna().sum()>0:
        unique_values = np.append(np.nan, unique_values)
    sorted_unique_values = unique_values[:min(5, 
unique_values.shape[0])]

    return sorted_unique_values

def gen_basic_info_dataframe(df, typical_values = True):
    '''
        Making DataFrame with:
         -number of unique values
         -data type
         -percentage of NaN values
         -list with max. 5 most common values in the column.
            If NaN occur in the column it's first element in this 
list,
            even if it's not the most common value.
    '''
    basic_info=pd.DataFrame(df.nunique(), columns=
['num_unique_values'])

    basic_info['data_type']= df.dtypes
    basic_info['NaN_percentage'] = (df.isna().sum())*100/df.shape[0]
    if typical_values:
        basic_info['typical_values'] = df.apply(first_unique_values, 
axis=0)

    return basic_info

basic_info = gen_basic_info_dataframe(data_df)
basic_info[20:25]
```

	num_unique_values	data_type	NaN_percentage	typical_values
Flow Bytes/s	142424	float64	0.000077	[nan, 802000000.0, 766000000.0, 750000000.0, 4...
Flow Packets/s	87307	float64	0.000000	[2000000.0, 1000000.0, 41666.66666666666, inf,...
Flow IAT Mean	94628	float64	0.000000	[1.0, 2.0, 48.0, 0.0, 49.0]
Flow IAT Std	565076	float64	0.000000	[0.0, 0.5773502691896257, 0.4472135954999579, ...
Flow IAT Max	46098	float64	0.000000	[1.0, 2.0, 48.0, 0.0, 49.0]

Now, we'll remove columns with 'irrelevant' information:

```
irrelevant_columns = [
    'Flow ID',
    ' Source IP',
    ' Source Port',
    ' Destination IP',
    ' Destination Port',
    ' Timestamp',
    'SimillarHTTP',
    ]

data_df.drop(labels=irrelevant_columns, axis=1, inplace=True)
```

After that, let's remove columns with one unique value:

```
# get these columns names as a list:
one_value_columns =
basic_info[basic_info['num_unique_values']==1].index.tolist()
print("Columns to drop:\n", one_value_columns)

data_df.drop(labels=one_value_columns, axis=1, inplace=True)
```

```
Columns to drop:
 [' Bwd PSH Flags', ' Fwd URG Flags', ' Bwd URG Flags', 'FIN Flag
Count', ' PSH Flag Count', ' ECE Flag Count', 'Fwd Avg Bytes/Bulk', '
Fwd Avg Packets/Bulk', ' Fwd Avg Bulk Rate', ' Bwd Avg Bytes/Bulk', '
Bwd Avg Packets/Bulk', 'Bwd Avg Bulk Rate']
```

Finally, we eliminate correlated columns. Let's generate correlation matrix:

```
corr_matrix = data_df.corr(method='pearson')
```

```
            square=True,
            annot=True,
            ax=ax)
plt.show()
```

Let's generate pairs to eliminate from matrix. We'll set the correlation factor threshold on `0.94`:

```
threshold = 0.94

filtered_corr_matrix = corr_matrix[np.abs(corr_matrix)>threshold]
unstack_f_cm = filtered_corr_matrix.unstack().dropna().reset_index()

# remove self-correlations:
corr_pairs =
unstack_f_cm[unstack_f_cm['level_0']!=unstack_f_cm['level_1']].reset_i
ndex().drop(labels=['index'], axis=1)

print(f'{len(corr_pairs)} correlations between columns with coeff.
over {threshold}.')
# set columns to drop:
corr_columns_to_drop = []
for id, row in corr_pairs.iterrows():
    if not (row['level_0'] in corr_columns_to_drop) and not
(row['level_1'] in corr_columns_to_drop):
        corr_columns_to_drop.append(row['level_0'])

print(f'{len(corr_columns_to_drop)} columns to drop.')
data_df.drop(labels=corr_columns_to_drop, axis=1, inplace=True)
```

```
124 correlations between columns with coeff. over 0.94.
31 columns to drop.
```

We can proceed our data preprocessing further e.g. one-hot encoding of columns with a few unique values, but let's keep it concise. Now, we can move on to training data preparation.

samples marked as no threat and randomly choose the rest of the samples to make `10,000` overall.

```
n_samples = 10000
```

We'll choose randomly `n` samples from over 2,600,000 rows:

- all (`763`) samples tagged as 'BENIGN'
- `n-763` samples tagged as 'DrDoS_SSDP'

```
# choose indices from both classes:
benign_ids = data_df[data_df[' Label']=='BENIGN'].index
ddos_ids = data_df[data_df[' Label']=='DrDoS_SSDP'].index

ddos_samples_ids = np.random.choice(ddos_ids, (n_samples-763))
benign_samples_ids = np.array(benign_ids)
# merge chosen indices:
samples_ids = np.concatenate((ddos_samples_ids, benign_samples_ids))

selected_data_df = data_df.iloc[samples_ids]
selected_data_df[' Label'].value_counts()
```

```
DrDoS_SSDP    9237
BENIGN         763
Name:  Label, dtype: int64
```

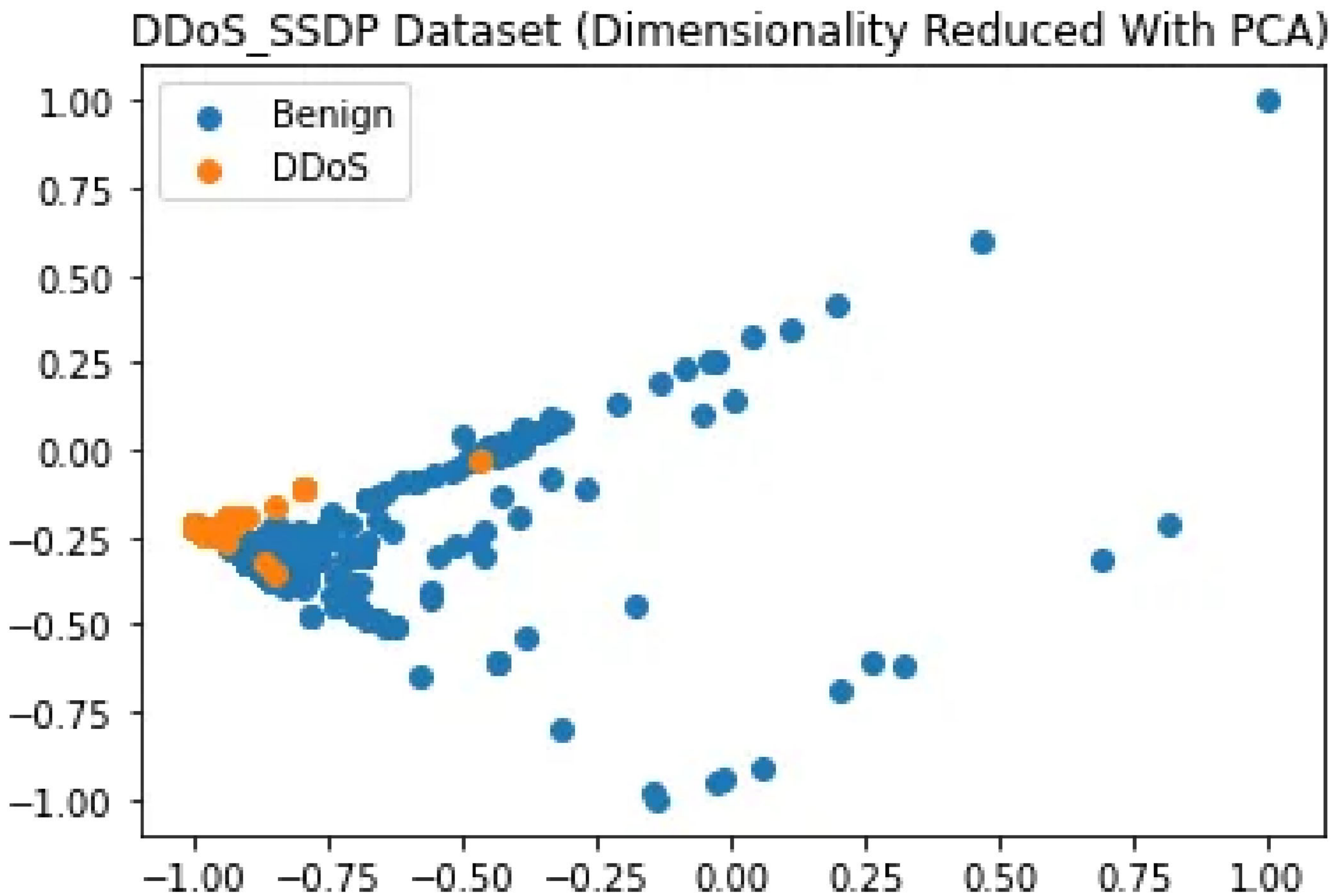
Finally, we need to change labels from string to integers, where `0` s mean benign data point and `1` s mean DDoS attack:

```
y = np.array(selected_data_df['
```

need to perform feature extraction. With PCA fitted on `10,000` previously chosen samples, we can go down to `2` features in ranges `-1` to `1`.

```
n_features = 2

# Standardize all the features:
x =
StandardScaler().fit_transform(np.array(selected_data_df.drop(columns=
[' Label'], inplace=False)))

# PCA fitting and transforming the data:
pca = sklearn.decomposition.PCA(n_components=n_features)
pca.fit(x)
x = pca.transform(x)

# Normalize the output to the range (-1, +1):
minmax_scale = MinMaxScaler((-1, 1)).fit(x)
x = minmax_scale.transform(x)
x = x.astype(comp_dtype)

# Plot results:
for k in range(0, 2):
    x_axis_data = x[np.array(y) == k, 0]
    y_axis_data = x[np.array(y) == k, 1]
    label = 'Benign' if k == 0 else 'DDoS'
    print(f'{label}: {x_axis_data.shape}')
    plt.scatter(x_axis_data, y_axis_data, label=label)

plt.title("DDoS_SSDP Dataset (Dimensionality Reduced With PCA)")
plt.legend()
plt.show()
```

```
Benign: (763,)
DDoS: (9237,)
```



And that's it! In the next part we'll train on this data a hybrid Quantum Neural Network model.

Summary:

In this article we've preprocessed a tabular cybersecurity dataset. We've prepared the data for training small, quantum models with dropping some features and performing the Principal Component Analysis (PCA).

References:

[1] Quantum machine learning for intrusion detection of distributed denial of service attacks: a comparative overview, *E. D. Payares and J. C. Martinez-Santos, 2021*

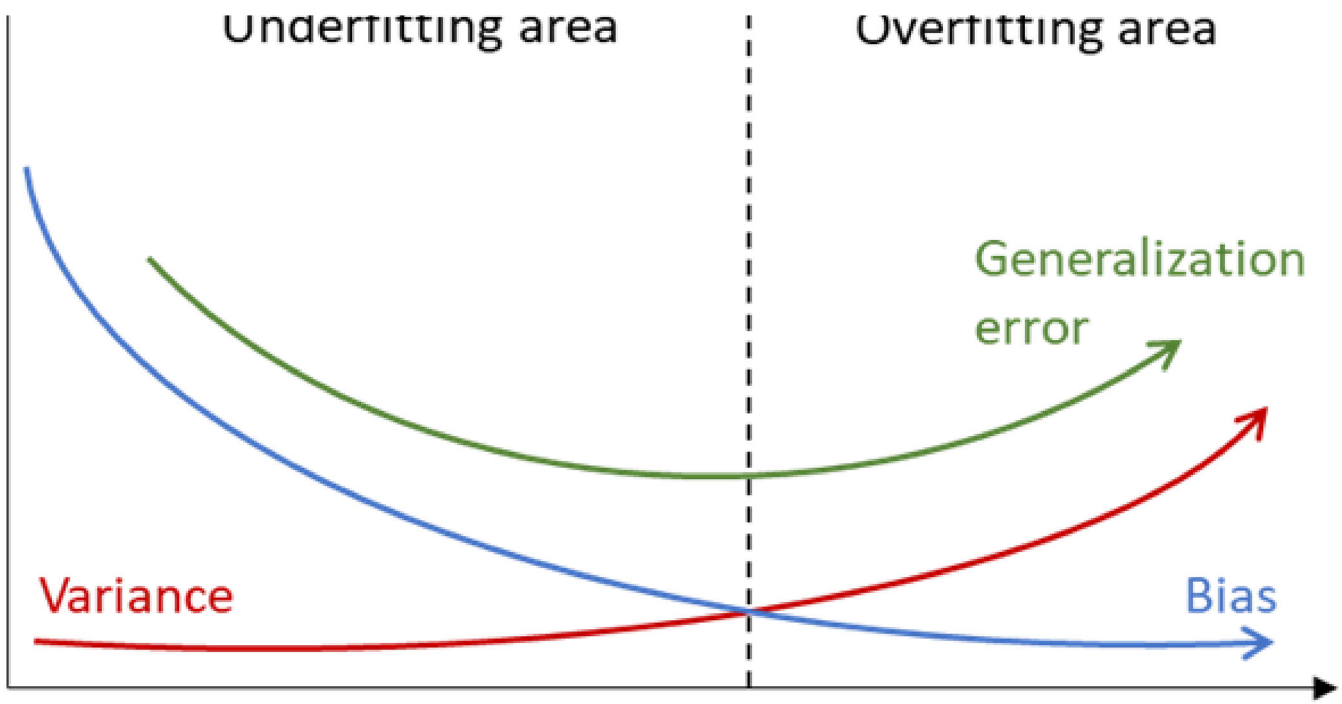
Follow

Written by Marek Kowalik

17 Followers

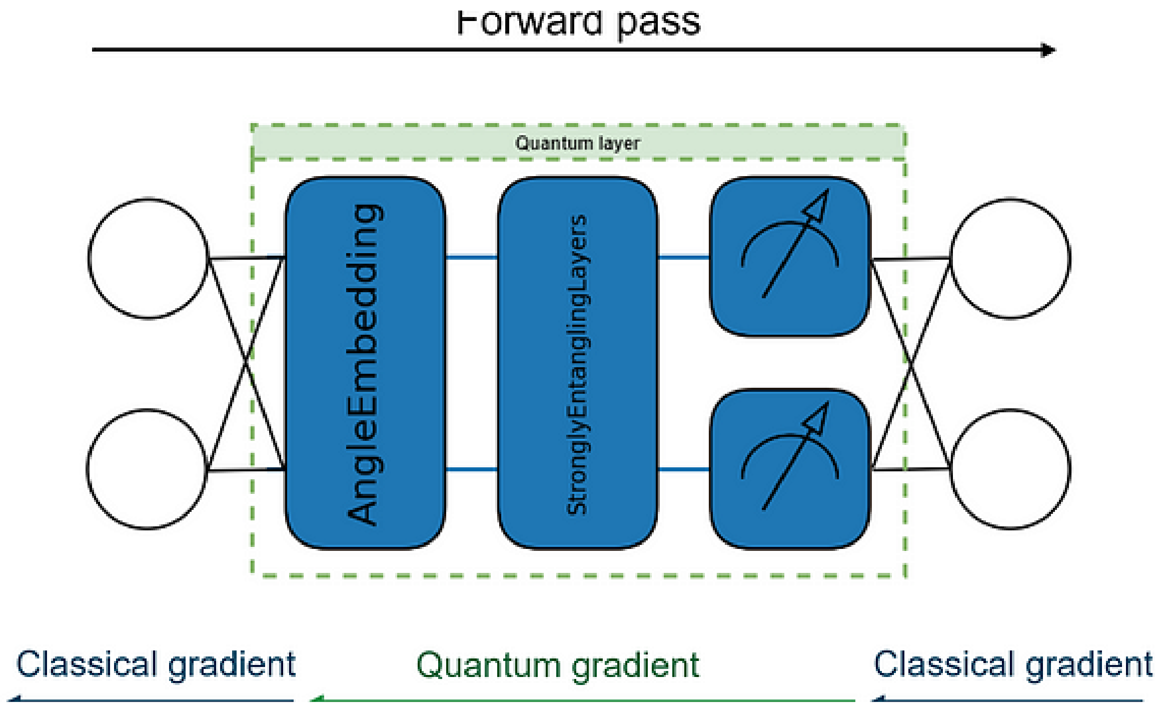
Junior Data Scientist and Quantum Developer in Capgemini Quantum Lab.

More from Marek Kowalik



Marek Kowalik

Capacities of Quantum Neural Networks, Part 1

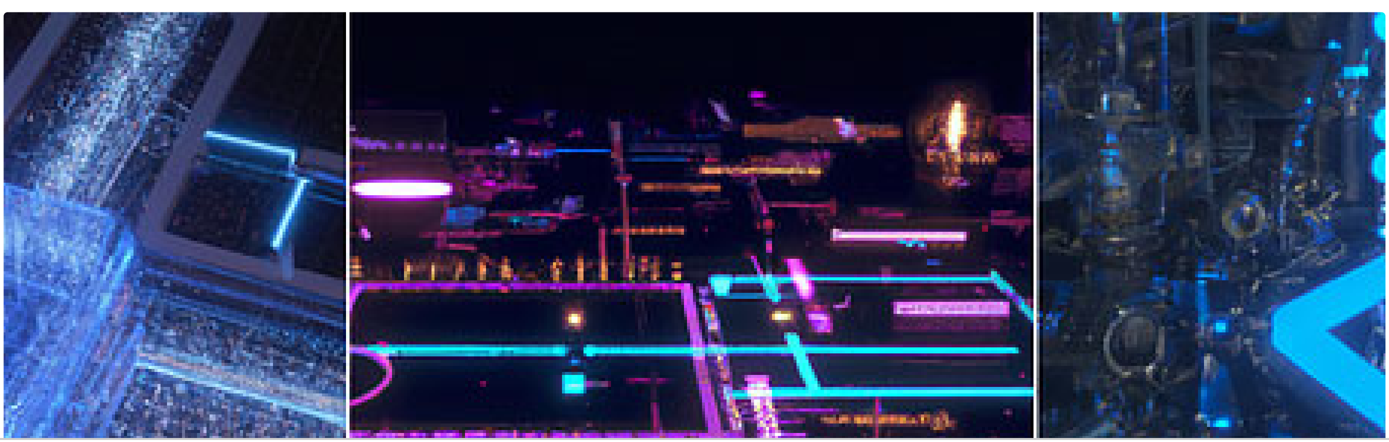


Marek Kowalik

Quantum Machine Learning for Cybersecurity, Part 2

9 min read · Sep 7, 2022

5

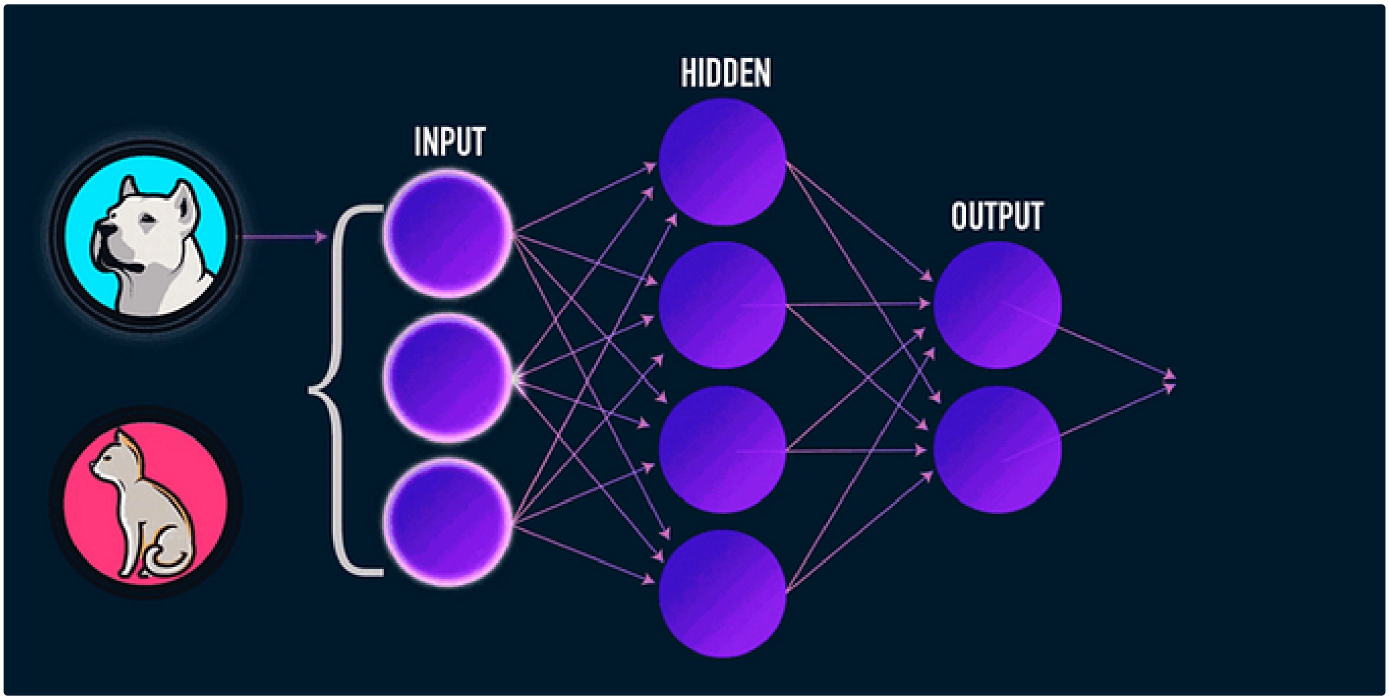


Parallelization for Quantum Computers — multi-programming

Parallelization is a game changer when it comes to the optimization of programs; particularly for training deep learning models, it has...

7 min read · Oct 12, 2022

55 1



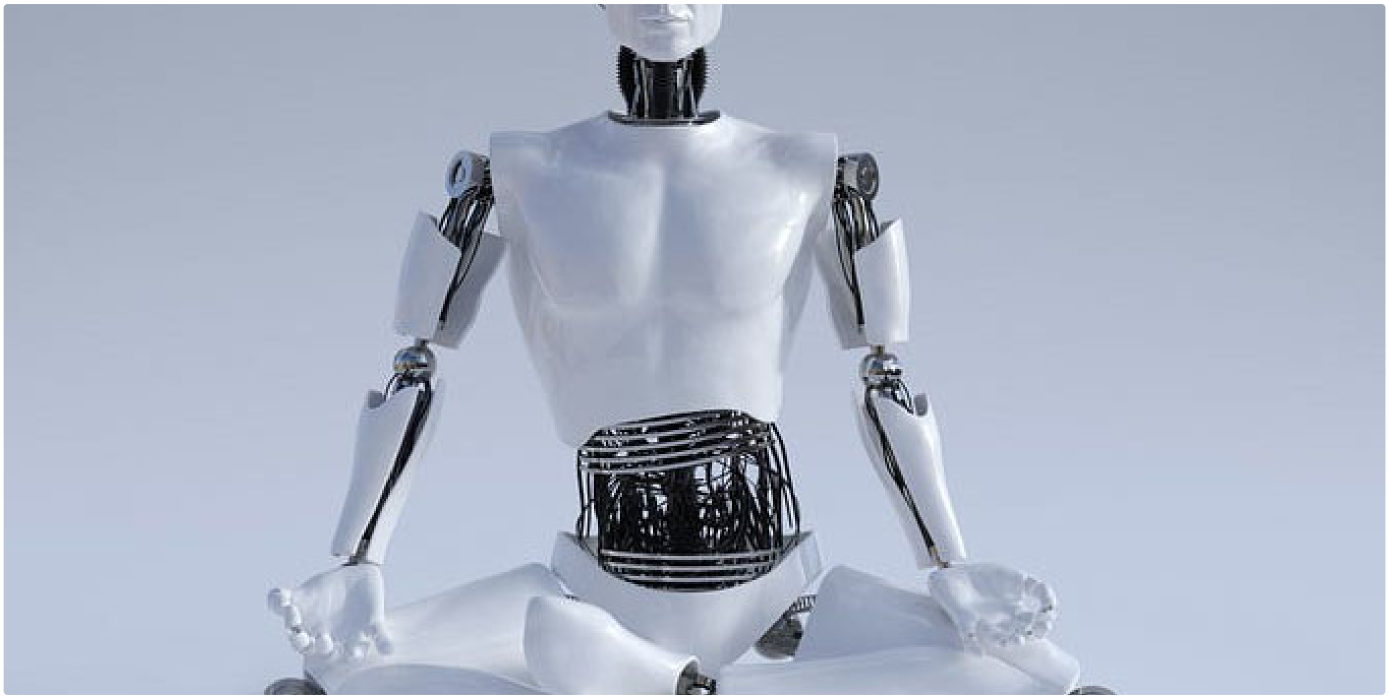
Marek Kowalik

Capacities of Quantum Neural Networks, Part 2

On a way to compare quantum and classical machine learning models

8 min read · 3 days ago

Recommended from Medium



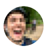
The PyCoach in Artificial Corner

You're Using ChatGPT Wrong! Here's How to Be Ahead of 99% of ChatGPT Users

Master ChatGPT by learning prompt engineering.

· 7 min read · Mar 17

17.9K 326

 Matt Chapman in Towards Data Science

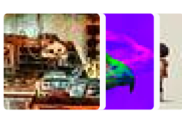
The Portfolio that Got Me a Data Scientist Job

Spoiler alert: It was surprisingly easy (and free) to make

· 10 min read · Mar 24

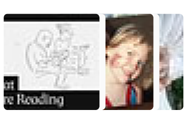
2.8K 43

Lists



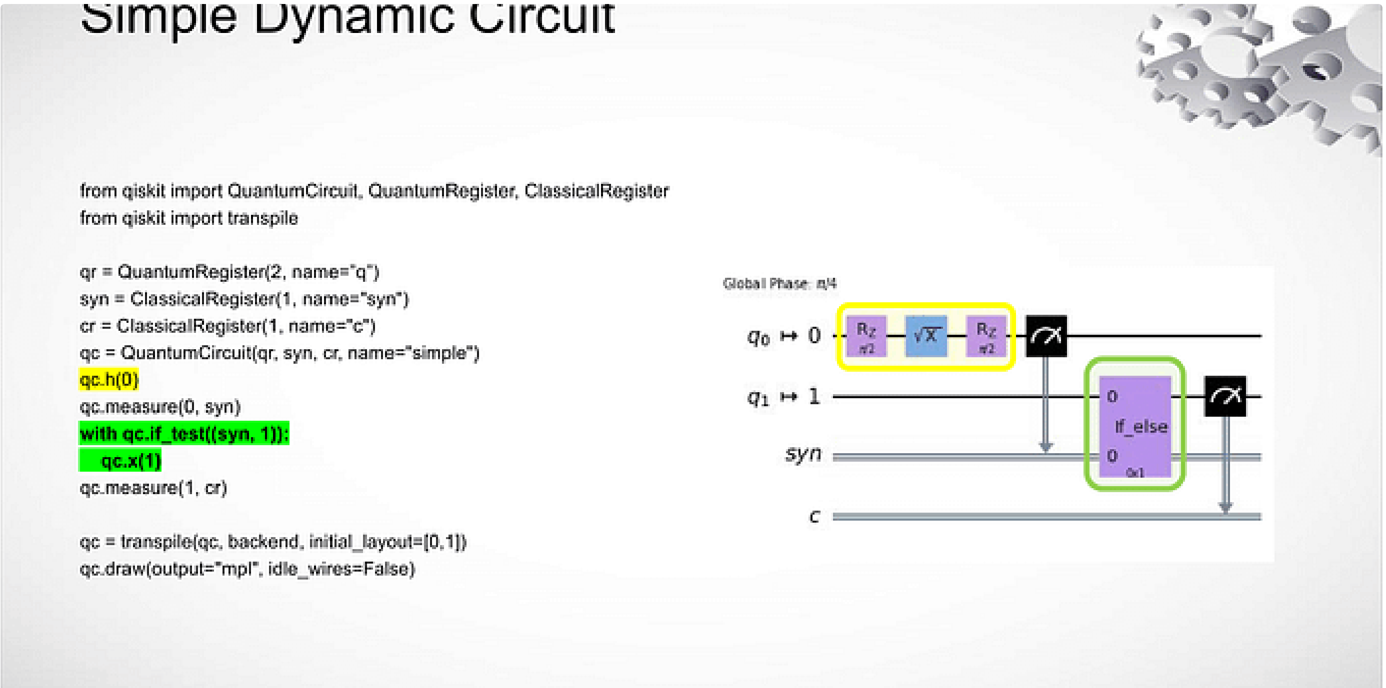
What is ChatGPT?

9 stories · 21 saves



Staff Picks

300 stories · 59 saves



Brian N. Siegelwax in Level Up Coding

Dynamic Quantum Circuits, Lesson 1

FYI: code snippets won't work.

· 2 min read · Dec 5, 2022

54

Continue to Medium

Sign up Already have an account? Sign in

Click "Sign Up" to agree to Medium's Terms of Service and acknowledge that Medium's Privacy Policy applies to you.

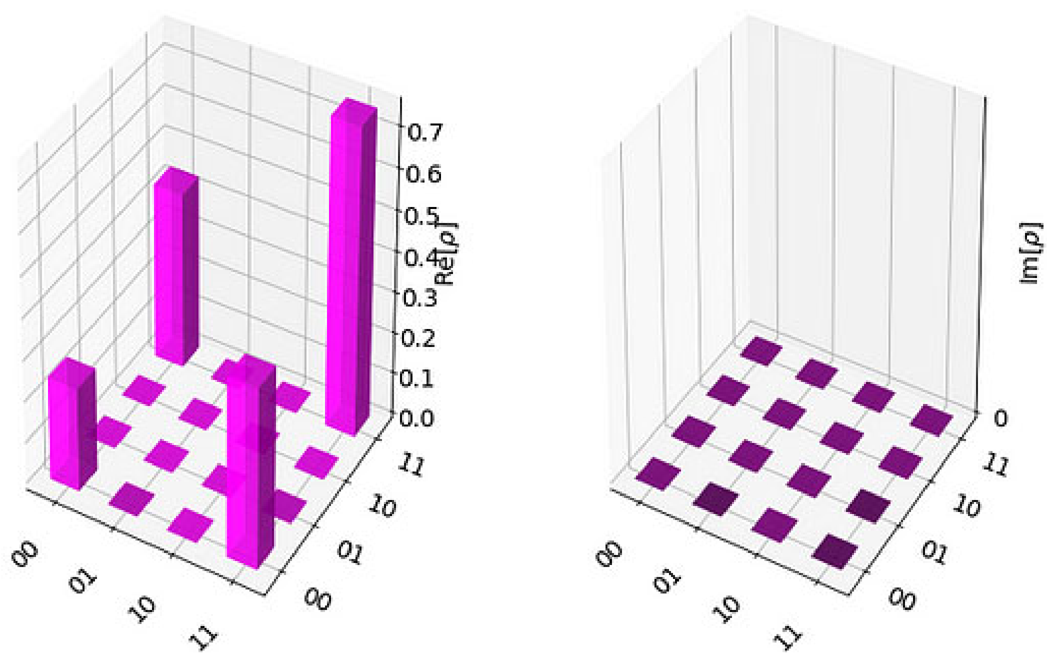
Shawhin Talebi in Towards Data Science

The Wavelet Transform

An Introduction and Example

· 6 min read · Dec 21, 2020

422 4



Saptashwa Bhattacharyya in A Bit of Qubit

Understanding Bloch Sphere: From Density Matrix Perspective

Pure & Mixed States, Density Matrix & Bloch Sphere

· 11 min read · Jan 26

54 1

Dennis Ganzaroli in MLearning.ai

The Best kept Secret in Data Science is KNIME

Discover KNIME, the best kept secret in data science. This powerful and versatile open source platform offers a visual interface and wide...

· 14 min read · Feb 2

367 3

See more recommendations