Introduction

Deep learning models are an essential part of big data anlaysis within healthcare. Neutral networks are utilized to optimally predict future diagnosis, hospital readmission, drug interaction, treatment recommendation, etc. Specifically, graph neural networks (GNNs) are vital to drug discovery, molecular property prediction, and other drug interaction prediciton tasks; one example is drug pair scoring, the prediciton task of determining the relationship between a pair of drugs, based on the known effects of various other drug pairs. Some specific predictions between drugs include drug-drug interaction, synergy, and polypharmacy side effects.

While there are open-scource drug interaction libraries available, they often predict based on a single molecule and do not allow for drug synergy predictions, predicting if the combined effect of a pair of drugs is more effective than the additive effect of each drug separately. There is no multi-use, streamlined framwork that is easily accessible by machine learning researchers.

The paper introduces ChemicalX [1], a library of PyTorch implementations of 11 prominent deep-learning models that predict various drug-drug interactions. Aside from the drug pair scoring deep-learning model architectures, this library also provides data loaders, integrated benchmark datasets, training, and evaluation routines.

All ChemicalX models were trained and tested across multiple datasets to predict drug-drug interaction, synergy, and polypharmacy side effects; and all models were evaulated using AUPRC, AUROC, and F1 metrics. Finally, the authors address use cases of the library by oncologists, computational chemists, drug safety researchers, and various oher machine learning researchers.

Links:

- GitHub Repository: https://github.com/nilakrishnan/dlh_598_final_project
- Final Presentation Video: https://youtu.be/NO2DBRdzHII

Scope of Reproducibility

DeepSynergy [2] and DeepDDS [3] are two of the models implemented by the ChemicalX library.

The specific hypothesis that will be tested in this report is that DeepSynergy will outperform DeepDDS when trained and tested using the DrugComb [4] dataset to predict drug synergy. Specifically, the DeepSynergy model implemented in this report will have a higher AUPRC, higher AUROC, and higher F1 score than the DeepDDS model from the ChemicalX library.

Methodology

Environment

Please ensure that the notebook is connected to a runtime with Python3 (version 3.10) and a T4 GPU before proceeding.

First, all necessary dependencies are installed:

```
import os
import torch

!pip install torch-scatter -f https://data.pyg.org/whl/torch-{torch.__version__}.htm
!pip install torch-cluster -f https://data.pyg.org/whl/torch-{torch.__version__}.htm

!pip install torchdrug
```

```
Looking in links: https://data.pyg.org/whl/torch-2.2.1+cu121.html
Requirement already satisfied: torch-scatter in /usr/local/lib/python3.10/dist-p
Looking in links: https://data.pyg.org/whl/torch-2.2.1+cu121.html
Requirement already satisfied: torch-cluster in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy<1.28.0,>=1.21.6 in /usr/local/lib/python3.1
Requirement already satisfied: torchdrug in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: torch>=1.8.0 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: torch-scatter>=2.0.8 in /usr/local/lib/python3.10
Requirement already satisfied: torch-cluster>=1.5.9 in /usr/local/lib/python3.10
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: rdkit-pypi>=2020.9 in /usr/local/lib/python3.10/d
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: ninja in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: lmdb in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: fair-esm in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in /usr/local/li
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in /usr/local/
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in /usr/local/li
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in /usr/local/lib/pyt
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in /usr/local/lib/py
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in /usr/local/lib/py
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in /usr/local/lib/
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in /usr/local/li
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in /usr/local/li
Requirement already satisfied: nvidia-nccl-cu12==2.19.3 in /usr/local/lib/python
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in /usr/local/lib/pyth
Requirement already satisfied: triton==2.2.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: nvidia-nvjitlink-cu12 in /usr/local/lib/python3.1
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dis
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/di
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dis
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-pa
```

The **chemicalx** dependecy is installed so the model implemented in this report can be evaulated against the implementation from the ChemicalX library as well as the DeepDDS model implementation. The **torchdrug** dependency is installed to utilize the PackedGraph and Molecule data structures.

```
import csv
import io
import json
import time
import torchdrug
import urllib.request
import warnings

import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from torch.utils.data import Dataset, DataLoader
from torchdrug.data import PackedGraph, Molecule
from sklearn.metrics import average_precision_score, f1_score, roc_auc_score
from sklearn.model_selection import train_test_split
```

Data

The raw data is from ChemicalX's GitHub repository, which was originally sourced from DrugCombDB [2], which contains data on known synergistic drug pairs that are effective at destroying certain cancer cell types. The raw data includes 3 datasets: *drug_set.json* , *context_set.json*, and *labeled_triples.csv*.

drug_set.json provides drug IDs, their respective simplified molecular-input line-entry system (SMILE), which is a line notation describing the structure of the molecular compound, and their features.

context_set.json provides contexts, which are specific IDs of cancer cell lines and their features. Cell lines are populations of isolated cells that are commonly used in cancer treatment research.

labeled_triples.csv includes two drug IDs, a context, and a boolean indicating if the pair of drugs are synergetic.

```
# ignore warnings from torchdrug.data.Molecule
warnings.filterwarnings('ignore')
```

Download the raw data files:

```
!gdown 1B4rFgBQrH-UgY3MYx48awHcWHMIODumI
!gdown 15fjupmoovZgCUad5I6lNcYURkdTqECvn
!gdown 1SCNCDmwgCI9ATtc_WbiDtrl6wZs3qcg1&usp
```

```
Downloading...
From: https://drive.google.com/uc?id=1B4rFgBQrH-UgY3MYx48awHcWHMIODumI
To: /content/drug_set.json
100% 11.8M/11.8M [00:00<00:00, 63.2MB/s]
Downloading...
From: https://drive.google.com/uc?id=15fjupmoovZgCUad5I6lNcYURkdTqECvn
To: /content/context_set.json
100% 140k/140k [00:00<00:00, 11.0MB/s]
/bin/bash: line 1: usp: command not found
Downloading...
From: https://drive.google.com/uc?id=1SCNCDmwgCI9ATtc_WbiDtrl6wZs3qcg1
To: /content/labeled_triples.csv
100% 4.27M/4.27M [00:00<00:00, 38.2MB/s]
```

```
drug_set_path = '/content/drug_set.json'
context_set_path = '/content/context_set.json'
labeled_triples_path = '/content/labeled_triples.csv'
```

Load the raw data:

```
def load_raw_data(drug_set_path, context_set_path, labeled_triples_path):
  drug_set = None
  context_set = None

  with open(drug_set_path) as f:
    drug_set = json.load(f)

  with open(context_set_path) as f:
    context_set = json.load(f)

  drug_features = {k: torch.FloatTensor(v["features"]).view(1, -1) for k, v in drug_
  context_features = {k: torch.FloatTensor(np.array(v).reshape(1, -1)) for k, v in c
  labeled_triples = pd.read_csv(labeled_triples_path, encoding="utf8", sep=",", dtyp

  return drug_features, context_features, labeled_triples

drug_features, context_features, labeled_triples = load_raw_data(drug_set_path, cont
```

```
def calculate_stats(data):
  num_features = len(data)
  return num_features

print("Number of drug features: {}".format(calculate_stats(drug_features)))
print("Number of context features: {}".format(calculate_stats(context_features)))
print("Number of labeled triples: {}".format(calculate_stats(labeled_triples)))
```

```
    Number of drug features: 2956
    Number of context features: 112
    Number of labeled triples: 191391
```

```
class DeepSynergyDataset(Dataset):
    def __init__(self, drug_features, context_features, labeled_triples):
        self.drug_features = drug_features
        self.context_features = context_features
        self.labeled_triples = labeled_triples

    def __len__(self):
        return len(self.labeled_triples)

    def __getitem__(self, index):
        row = self.labeled_triples.iloc[index]
        return self.drug_features[row["drug_1"]], self.drug_features[row["drug_2"]],

dataset = DeepSynergyDataset(drug_features, context_features, labeled_triples)
```

```
def process_data(drug_features, context_features, labeled_triples):
  drug_feature_matrix = torch.cat([drug_features[i] for i in drug_features])
  context_feature_matrix = torch.cat([context_features[i] for i in context_features]
  return drug_feature_matrix, context_feature_matrix

drug_feature_matrix, context_feature_matrix = process_data(drug_features, context_fe
```

Model

The model below is based on the implementation from the ChemicalX's GitHub repository of the DeepSynergy [2] model.

The model with 624 input channels and includes four fully connected linear layers, alternated with activation layers using ReLU. Dropout is applied before the hidden states are passed through the last linear layer and the final activation layer using the Sigmoid function.

```
class DeepSynergy(torch.nn.Module):
  def __init__(self, context_channels, drug_channels):
    super().__init__()
    self.l1 = torch.nn.Linear(2 * drug_channels + context_channels, 128)
    self.l2 = torch.nn.Linear(128, 32)
    self.l3 = torch.nn.Linear(32, 16)
    self.l4 = torch.nn.Linear(16, 1)

    self.dropout = torch.nn.Dropout(0.5)
    self.relu = torch.nn.ReLU()
    self.sigmoid = torch.nn.Sigmoid()


  def forward(self, context_features, drug_features_left, drug_features_right):
    context_features = context_features.squeeze()
    drug_features_left = drug_features_left.squeeze()
    drug_features_right = drug_features_right.squeeze()
    hidden = torch.cat([context_features, drug_features_left, drug_features_right],
    hidden = self.l3(self.relu(self.l2(self.relu(self.l1(hidden)))))
    return self.sigmoid(self.l4(self.dropout(self.relu(hidden))))

deep_synergy_model = DeepSynergy(context_feature_matrix.shape[1], drug_feature_matri
```

Training

The batch size used to split the traning data is 5120, which is the default number used to test all the models withing the ChemicalX library. Smaller batch sizes between 2000 - 4000 were also tested, but required much more execution time. Other hyperparameters that were tested was the dropout rate and the input & output channels for the second, third, and fourth linear layers.

The model is trained using 100 epochs, which is the default number epochs used to test all the models within the ChemicalX library. The average length of each epoch is 2.211 seconds. While using Google Colab without a subscription, only a certain amount of compute units are provided, so the runtime is limited when connecting to a GPU.

The loss function is specified as Binary Cross Entropy and the optimizer is set to Adam algorithm with the default learning rate of 0.001.

```
training_data, test_data = train_test_split(dataset, train_size=0.8, random_state=42

train_loader = DataLoader(training_data, batch_size=5120, shuffle=True)
test_loader = DataLoader(test_data, batch_size=5120, shuffle=False)
```

```
loss_func = torch.nn.BCELoss()
optimizer = torch.optim.Adam(deep_synergy_model.parameters())
```

```
for epoch in range(100):
  deep_synergy_model.train()
  train_loss = 0

  for batch in train_loader:
    optimizer.zero_grad()

    drug_features_left, drug_features_right, context, label = batch
    prediction = deep_synergy_model(context, drug_features_left, drug_features_right

    label = label.to(torch.float32)
    loss = loss_func(prediction, label)
    loss.backward()
    train_loss += loss.item()

    optimizer.step()

  train_loss = train_loss / len(train_loader)
  print('Epoch: {} \t Training Loss: {:.6f}'.format(epoch + 1, train_loss))
```

```
Epoch: 55        Training Loss: 0.305460
Epoch: 56        Training Loss: 0.307383
Epoch: 57        Training Loss: 0.306064
Epoch: 58        Training Loss: 0.300142
Epoch: 59        Training Loss: 0.298454
Epoch: 60        Training Loss: 0.296502
Epoch: 61        Training Loss: 0.295857
Epoch: 62        Training Loss: 0.297548
Epoch: 63        Training Loss: 0.291891
Epoch: 64        Training Loss: 0.292716
Epoch: 65        Training Loss: 0.289087
Epoch: 66        Training Loss: 0.289860
Epoch: 67        Training Loss: 0.292739
Epoch: 68        Training Loss: 0.284289
Epoch: 69        Training Loss: 0.282288
Epoch: 70        Training Loss: 0.279451
Epoch: 71        Training Loss: 0.281792
Epoch: 72        Training Loss: 0.276957
Epoch: 73        Training Loss: 0.279416
Epoch: 74        Training Loss: 0.273141
Epoch: 75        Training Loss: 0.272454
Epoch: 76        Training Loss: 0.271097
Epoch: 77        Training Loss: 0.273192
Epoch: 78        Training Loss: 0.269232
Epoch: 79        Training Loss: 0.270749
Epoch: 80        Training Loss: 0.268607
Epoch: 81        Training Loss: 0.270070
Epoch: 82        Training Loss: 0.263636
Epoch: 83        Training Loss: 0.260536
Epoch: 84        Training Loss: 0.260225
Epoch: 85        Training Loss: 0.258119
Epoch: 86        Training Loss: 0.255894
Epoch: 87        Training Loss: 0.254976
Epoch: 88        Training Loss: 0.253798
Epoch: 89        Training Loss: 0.252284
Epoch: 90        Training Loss: 0.257546
Epoch: 91        Training Loss: 0.252883
Epoch: 92        Training Loss: 0.248700
Epoch: 93        Training Loss: 0.245903
Epoch: 94        Training Loss: 0.248226
Epoch: 95        Training Loss: 0.246734
Epoch: 96        Training Loss: 0.243288
Epoch: 97        Training Loss: 0.243623
Epoch: 98        Training Loss: 0.241976
Epoch: 99        Training Loss: 0.239625
Epoch: 100       Training Loss: 0.238925
```

Results

The model implemented in this report is evaluated on the same metrics used to evaulate all the models in the ChemicalX paper: AUPRC, AUROC, and F1.

1. AUPRC: This metric indicates how well your model ranks predictions. A high AUPRC ...

2. AUROC: This metric... A high AUROC indicates a high TPR (true positive rate) and a low FPR (false positive rate).
3. F1: The F1 score indicates the reliability of a model. A high F1 score indicates that the model can achieve both high recall and high precision.

```
deep_synergy_model.eval()

y_pred = torch.LongTensor()
y_score = torch.Tensor()
y_true = torch.LongTensor()

for batch in test_loader:
    drug_features_left, drug_features_right, context, label = batch
    y_hat = deep_synergy_model(context, drug_features_left, drug_features_right).squ
    label = label.to(torch.float32)

    y_score = torch.cat((y_score,  y_hat.detach()), dim = 0)
    y_pred = torch.cat((y_pred,  (y_hat > 0.5).int().detach()), dim = 0)
    y_true = torch.cat((y_true, label.detach()), dim = 0)

auprc = average_precision_score(y_true, y_score)
auroc = roc_auc_score(y_true, y_score)
f1 = f1_score(y_true, y_pred)

print('AUPRC: {:.3f} \nAUROC: {:.3f} \nF1: {:.3f}'.format(auprc, auroc, f1))
```

```
AUPRC: 0.688
AUROC: 0.827
F1: 0.610
```

Model comparison

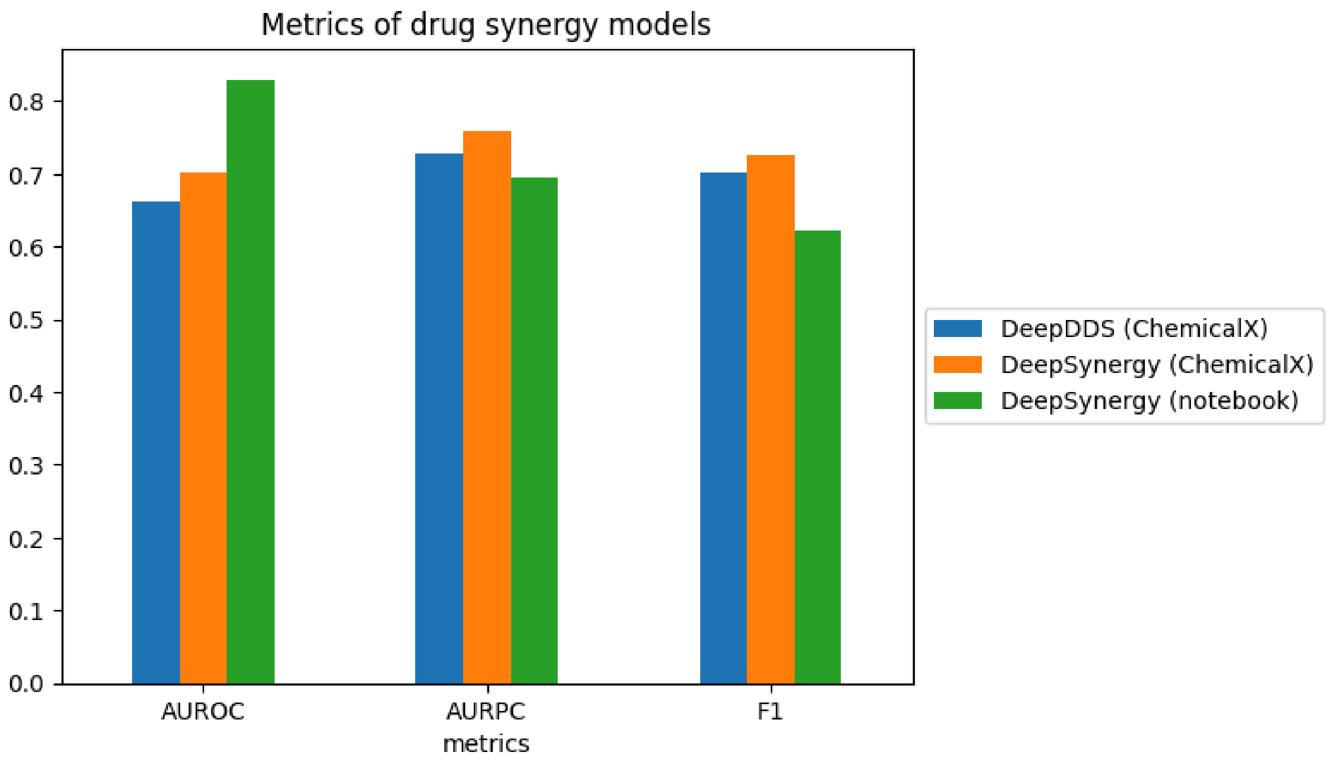
Here are the metrics of the DeepSynergy and DeepDDS models from the ChemicalX library for comparision:

```
pd.DataFrame({
    'metrics' : ['AURPC', 'AUROC', 'F1'],
    'DeepSynergy (notebook)' : [0.694, 0.830, 0.622],
    'DeepSynergy (ChemicalX)' : [0.758, 0.702, 0.725],
    'DeepDDS (ChemicalX)' : [0.729, 0.663, 0.702]
}).head()
```

	metrics	DeepSynergy (notebook)	DeepSynergy (ChemicalX)	DeepDDS (ChemicalX)
0	AURPC	0.694	0.758	0.729
1	AUROC	0.830	0.702	0.663
2	F1	0.622	0.725	0.702

```
%matplotlib inline
df = pd.DataFrame({
    'models' : ['DeepSynergy (notebook)', 'DeepSynergy (notebook)', 'DeepSynergy (no
    'metrics' : ['AURPC', 'AUROC', 'F1', 'AURPC', 'AUROC', 'F1', 'AURPC', 'AUROC', '
    'values' : [0.694, 0.830, 0.622, 0.758, 0.702, 0.725, 0.729, 0.663, 0.702]
    }).pivot(index='metrics', columns='models', values='values')

df.plot.bar(rot=0)
plt.title('Metrics of drug synergy models', color='black')
plt.legend(loc='center left', bbox_to_anchor=(1.0, 0.5))
plt.show()
```



As seen above, the implementation of DeepSynergy in this report achieved a higher AUROC than the DeepSynergy and DeepDDS model implmentations by ChemicalX; however, it did not achieve a higher AURPC or F1 score than those two models.

One reason the model may not have performed as well as the original implementation is that not all of the optimizations in the ChemicalX library were implemented in this report, such as the training & evalution pipline, batch generators, and other various abstractions.

Discussion

In this report, the DeepSynergy model implementation from ChemicalX was reproduced. Referencing existing code from the library's GitHub repository helped shape the flow of this report; however, the library included many more abstractions to accomdate the 10 other models implemented by ChemicalX. Some abstractions include a data structure for drug pairs, a parent class for all models, training data batch generators, etc. These abstractions were not included in this report since it was unecessary to implement one model.

References

1. Rozemberczki, B., Hoyt, C. T., Gogleva, A., Grabowski, P., Karis, K., Lamov, A., Nikolov, A., Nilsson, S., Ughetto, M., Wang, Y., Derr, T., & Gyori, B. M., ChemicalX: A Deep Learning Library for Drug Pair Scoring, Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2022, KDD ’22:The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 3819–3828, doi:https://doi.org/10.1145/3534678.3539023

2. Preuer, K., Lewis, R. P. I., Hochreiter, S., Bender, A., Bulusu, K. C., & Klambauer, G., DeepSynergy: predicting anti-cancer drug synergy with Deep Learning, Bioinformatics, 2018, 34:9, 1538–1546, doi:https://doi.org/10.1093/bioinformatics/btx806

3. J. Wang and X. Liu and S. Shen and L. Deng and H. Liu., DeepDDS: deep graph neural network with attention mechanism to predict synergistic drug combinations, Briefings in Bioinformatics, 2021, 23:1 doi:https://doi.org/10.1093/bib/bbab390

4. Liu, H., Zhang, W., Zou, B., Wang, J., Deng, Y., & Deng, L., DrugCombDB: a comprehensive database of drug combinations toward the discovery of combinatorial therapy, Nucleic Acids Research, 2020, 48:D1, D871-D881, doi:https://doi.org/10.1093/nar/gkz1007