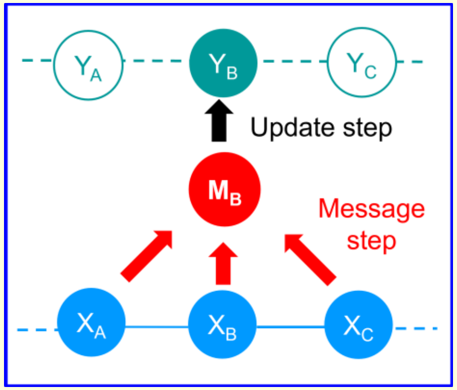
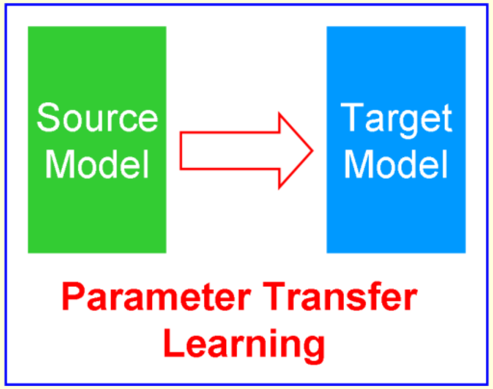
My other computer is BIOWULF

Deep Learning by Example on Biowulf

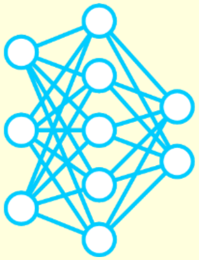
Class #7: Message Passing and Self Attention-based Networks, data augmentation, transfer learning and their application to drug molecule property prediction

Gennady Denisov, PhD

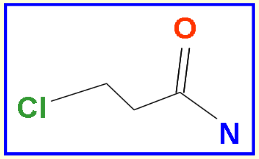




Intro: application of Deep Learning to analysis of molecules



Molecule as a graph



Nodes = atoms, **Edges** = bonds
The representation is **unique**

One-hot encoded matrices

Nodes features:

	symbol				valence				n_H			hybridization	
n_atoms	...C...	Cl...	N...	O...	1	2	3	4...	0	1	2...	sp2	sp3
	...0...	1 ...	0...	0...	1	0	0	0...	1	0	0...	0	1
	...1...	0 ...	0...	0...	0	0	0	1...	0	0	1...	0	1
	...1...	0 ...	0...	0...	0	0	0	1...	0	0	1...	0	1
	...0...	0 ...	0...	1...	0	1	0	0...	1	0	0...	1	0
	...1...	0 ...	0...	0...	0	0	0	1...	1	0	0...	1	0
	...0...	0 ...	1...	0...	0	0	1	0...	0	0	1...	1	0

Edges features

	bond_type				
n_bonds	single	double	triple	arom.	
	1	0	0	0	...
	0	1	0	0	...
	1	0	0	0	...
	1	0	0	0	...
	1	0	0	0	...

Adjacency matrix

	Cl	C	C	O	C	N
Cl	1	1				
C	1	1	1			
C		1	1		1	
O				1	1	
C			1	1	1	1
N					1	1

Graph Convolutional or similar layers

Molecule as a SMILES string

NC(=O)CCCl

The representation **is ambiguous**

SMILES enumeration:

O=C(CCCl)N
O=C(N)CCCl

alternative SMILES strings representing the same molecule

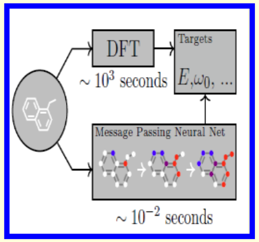
SMILES tokenization:

'N', 'C', '(', '=', 'O', ')', 'C', 'C', 'Cl‘

Recurrent or similar layers

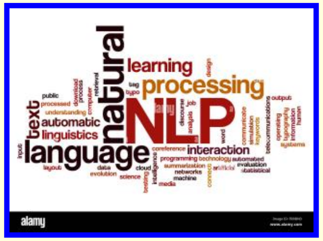
Relevant non-bio applications:

Predicting quantum mechanical properties of organic molecules



Message Passing (= generalization of Graph Convolution)

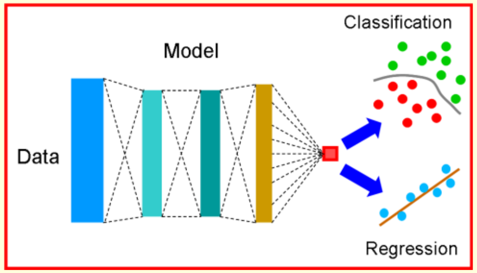
Natural language Processing (NLP)



text document ↔ SMILES dataset
sentence ↔ SMILES string,
word ↔ SMILES token,

Biological example:

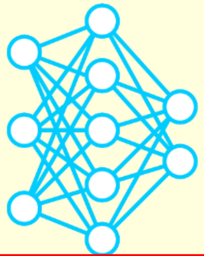
MPSAN-MP: Message Passing and Self-Attention based Networks for Molecular Property prediction



A composite application that supports **both the graph and SMILES** representations of molecules

Examples overview

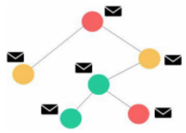
https://hpc.nih.gov/training/deep_learning_by_example.html



Class #	1	2	3	4	5	6	7
Bio app	Bioimage segmentation / fly brain connectome	Genomics / prediction of function of non-coding DNA	Genomics / reduction of dimensionality of cancer transcriptome	Bioimage synthesis / developmental biology	Drug molecule design	Genomics / classification of cancer types	Drug molecule property prediction
Neural network type	Convolutional	Recurrent or 1D-Convolutional	Autoencoder	Generative Adversarial	Reinforcement Learning	Graph Convolutional	Message Passing, Transformer
ML type	Supervised	Supervised	Unsupervised	Unsupervised	Reinforcement	Supervised	Supervised

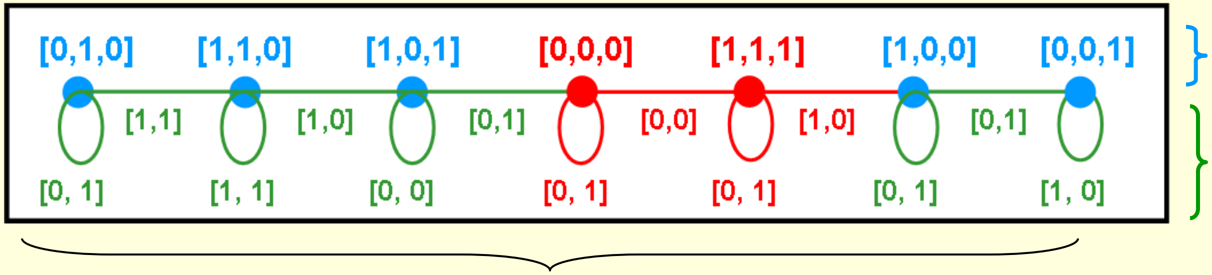
Supervised ML approach, in a way similar to those of classes #1, #2 and #6.
Data is an extended set of data from class #5, but the tasks/methods are different.
Two new types of layers to discuss: MessagePassing and MultiHeadAttention.
How data augmentation and transfer learning can help to fight overfitting?

MPN,SAN



Prototype example #1: Message Passing Network (MPN) model for classification of molecules represented by graphs

node features, edge features, pair indices



N (=3) features per node/atom

L (=2) features per link/bond

M (=7) atoms

Adjacency matrix
(class #6):

	0	1	2	3	4	5	6
0	1	1					
1	1	1	1				
2		1	1	1			
3			1	1	1		
4				1	1	1	
5					1	1	1
6						1	1

Pair_indices matrix
(= a non-sparse version of the adjacency matrix):

[[0,0],
[0,1], [1,1], [1,0],
[1,2], [2,2], [2,1],
[2,3], [3,3], [3,2],
[3,4], [4,4], [4,3],
[4,5], [5,5], [5,4],
[5,6], [6,6], [6,5]]

Input:

1) a set of randomly generated **cartoon drug molecules,**
   - each molecule represented by a linear/unbranched chain of **M atoms/nodes,**
   - linked to each other as specified by the adjacency / **pair indices matrix**
   - with each **node/atom** possessing a random vector of **N binary features**, and
   - each **link/bond** possessing a random vector of **L binary features**;

2) a target **motif** with **fixed values for the atom and bond features**.
   - if a drug molecule contains the **motif**, it will be considered "good" for treating a **hypothetical disease** and assigned the ground truth **label = 1**,
   - otherwise, it will be assigned the **label = 0** and not supposed to help in treating the disease**.**

Task:

Train a **Message Passing Network (MPN)** model on this data, so that it could **predict the class labels** for new, previously unseen cartoon molecules.

Prototype example #1 (cont.): MessagePassing layer vs vanilla Graph Convolution

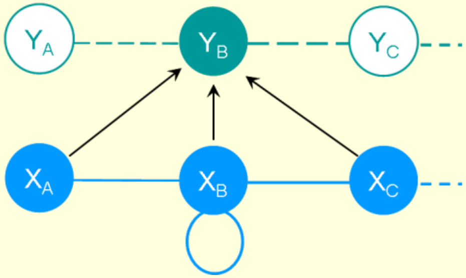
message passing

Gilmer et al., arXiv 2017

GCNConv (class #6):

X_i - “old” features
Y_i – “new” features
w_i, b_B – adjustable weights
$\tilde{d}_B$(=3) – degree of a node B.

$$\mathbf{Y_B = \frac{1}{\tilde{d}_B} \sum_{\substack{\text{one-hop}\\ \text{neighbors}\\ \text{and self}}} X_i \cdot w_i + b_B\,, \quad i = A, B, C}$$

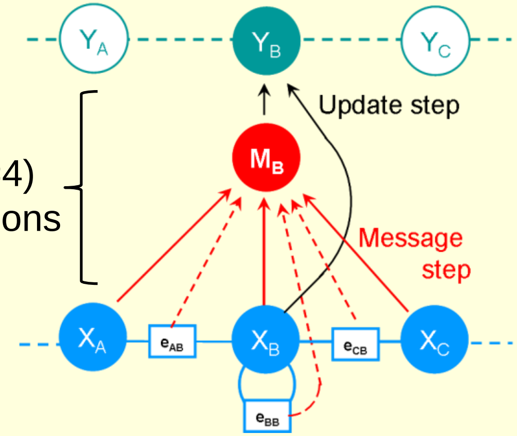


Message Passing :
- M_B - message variable
- e_{ij} - edge features
- U(. , .) - update function

$$\mathbf{1)\ \ M_B = \frac{1}{\tilde{d}_B} \sum_{\substack{\text{one-hop}\\ \text{neighbors}\\ \text{and self}}} X_i \cdot e_{iB} \cdot w_i + b_B, \quad i = A, B, C}$$

$$\mathbf{2)\ \ Y_B = U(M_B, X_B)} \qquad \text{\# in our code: U()} \equiv \text{GRUcell()}$$

T (=4) iterations

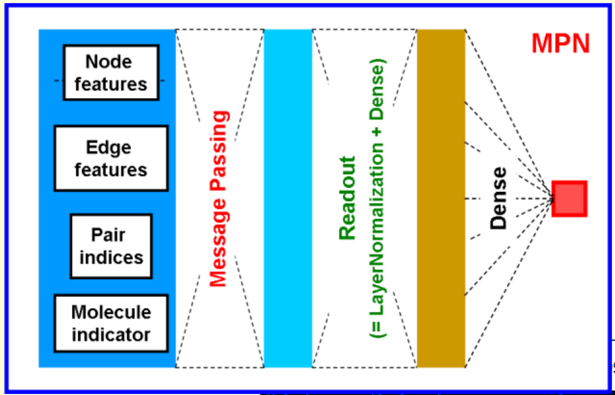


CONCLUSIONS:

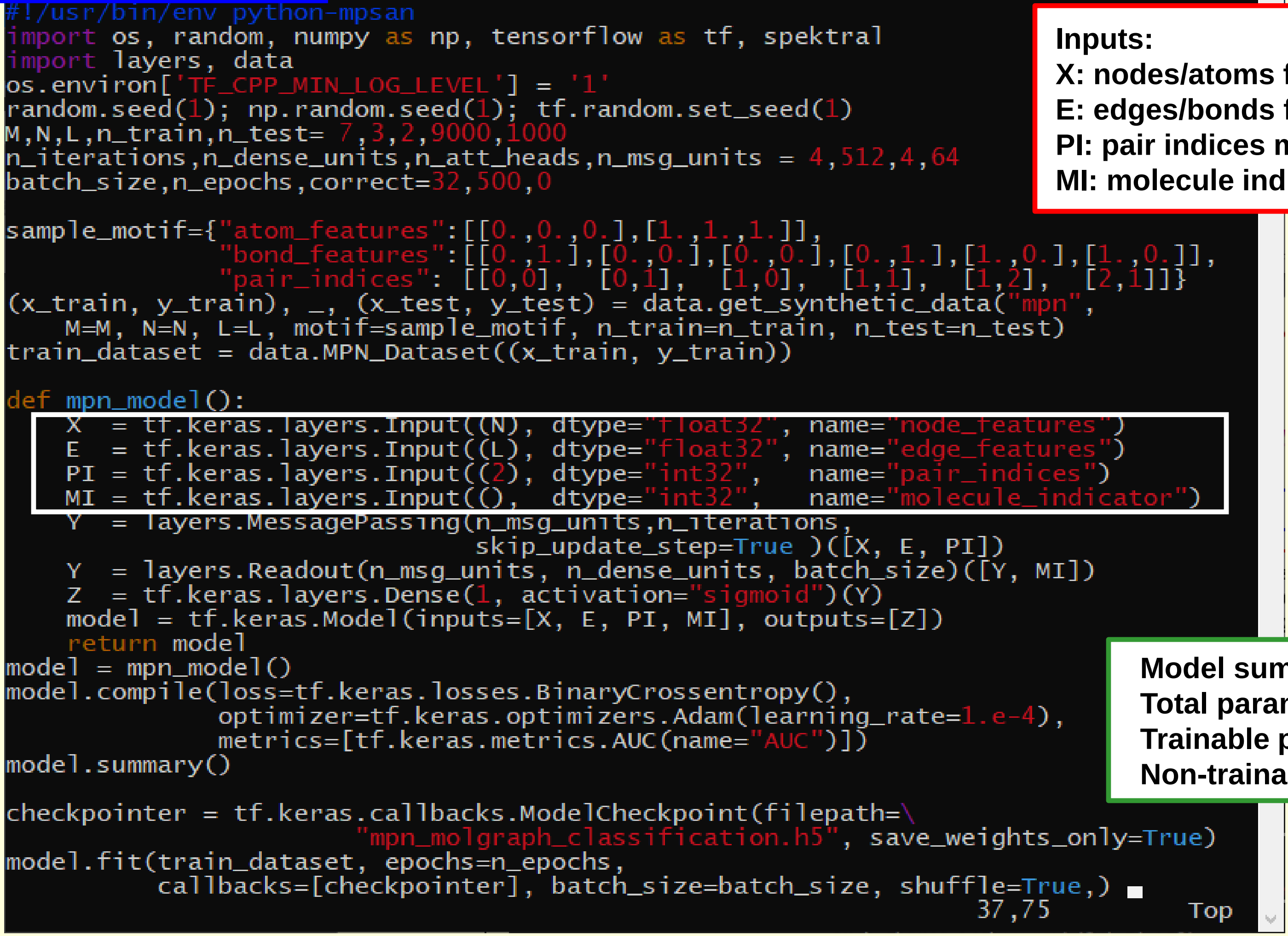
- **the MP filtering is performed in 2 steps: the Message step and the Update step**
- **these interspersed steps are iterated T times (T ≥ 1)**
- **the Message computations involve both the adjacent node features and edge features**
- **the edge features are not updated during the MP**
- **the Message Passing (MP) filtering is a generalization of the GCNConv transformation**
- **like GCNConv, the MP is a local transformation, involving current node and its one-hop neighbors**

Prototype example #1 (cont.): the training code

https://keras.io/examples/graph/mpnn-molecular-graphs/



MI = np.array([1,1,1,1,1,2,2,2,2,3,3,3,3,3,3,3,3,4,4,4,…])



```
#!/usr/bin/env python-mpsan
import os, random, numpy as np, tensorflow as tf, spektral
import layers, data
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '1'
random.seed(1); np.random.seed(1); tf.random.set_seed(1)
M,N,L,n_train,n_test= 7,3,2,9000,1000
n_iterations,n_dense_units,n_att_heads,n_msg_units = 4,512,4,64
batch_size,n_epochs,correct=32,500,0

sample_motif={"atom_features":[[0.,0.,0.],[1.,1.,1.]],
              "bond_features":[[0.,1.],[0.,0.],[0.,0.],[0.,1.],[1.,0.],[1.,0.]],
              "pair_indices": [[0,0],  [0,1],  [1,0],  [1,1],  [1,2],  [2,1]]}
(x_train, y_train), _, (x_test, y_test) = data.get_synthetic_data("mpn",
    M=M, N=N, L=L, motif=sample_motif, n_train=n_train, n_test=n_test)
train_dataset = data.MPN_Dataset((x_train, y_train))

def mpn_model():
    X  = tf.keras.layers.Input((N), dtype="float32", name="node_features")
    E  = tf.keras.layers.Input((L), dtype="float32", name="edge_features")
    PI = tf.keras.layers.Input((2), dtype="int32",   name="pair_indices")
    MI = tf.keras.layers.Input((),  dtype="int32",   name="molecule_indicator")
    Y  = layers.MessagePassing(n_msg_units,n_iterations,
                               skip_update_step=True )([X, E, PI])
    Y  = layers.Readout(n_msg_units, n_dense_units, batch_size)([Y, MI])
    Z  = tf.keras.layers.Dense(1, activation="sigmoid")(Y)
    model  = tf.keras.Model(inputs=[X, E, PI, MI], outputs=[Z])
    return model
model = mpn_model()
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(learning_rate=1.e-4),
              metrics=[tf.keras.metrics.AUC(name="AUC")])
model.summary()

checkpointer = tf.keras.callbacks.ModelCheckpoint(filepath=\
                       "mpn_molgraph_classification.h5", save_weights_only=True)
model.fit(train_dataset, epochs=n_epochs,
          callbacks=[checkpointer], batch_size=batch_size, shuffle=True,)
                                                                37,75         Top
```

Header
- imports
- HP defs

Get data

Define a model

Run the model
- fit

Inputs:
X: nodes/atoms features
E: edges/bonds features
PI: pair indices matrix
MI: molecule indicator

Model summary:
Total params: 71,041
Trainable params: 71,041
Non-trainable params: 0

Prototype example #2: Self-Attention Network (SAN) model for classification of molecules represented by sequences of tokens

Transformer: Vaswani et al. Attention is all you need, NIPS 2017

Residual connection: He et al, CVPR 2016.

X (data)	Y (labels)
ABCBDFFGCA	1
DGCEBFEACG	0
BAGAEDCBFA	0
DCFABCBEBE	1
DCFGAEADCE	0
DCCDBEACGF	0
CGCBFGAABC	1
GAACEADEEC	0
DCDDABCDBC	1
. . .	...

Input:

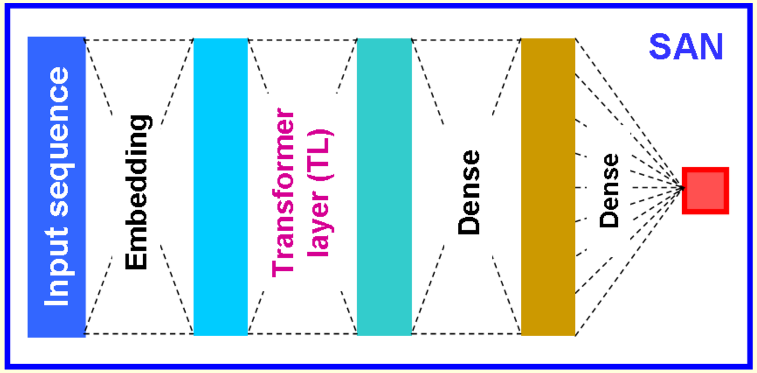
1) a set of cartoon **drug molecules** represented by strings of fixed length M(=10), generated randomly from a certain set of **tokens**, e.g. {**’A’, ’B’, ’C’, ’D’, ’E’, ’F’, ’G’**}.

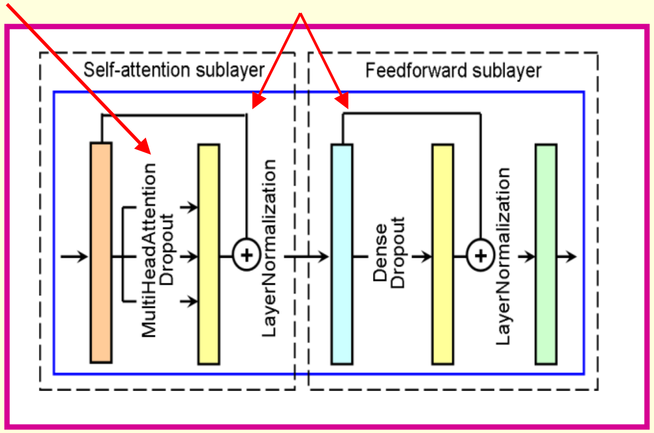
2) a **motif**, i.e. specific string of **”functional”** tokens, e.g. “**ABC**”.

If a **drug molecule** contains the **motif**, it will be considered “good” for treating a hypothetical disease and assigned the ground truth **label = 1, **otherwise it will be assigned the **label = 0**, and is not supposed to help in treating the disease**.**

Task:

train a **Self-Attention Network (SAN)** model on this data, so that it could **predict the class labels** for new, previously unseen cartoon drug molecules.





Transformer layer (TL) construct

CONCLUSION:

sequence analysis can be performed with models involving Attention mechanism implemented in the Transformer layer(s), i.e. there’s no need in the Recurrent or 1D Convolutional layers.

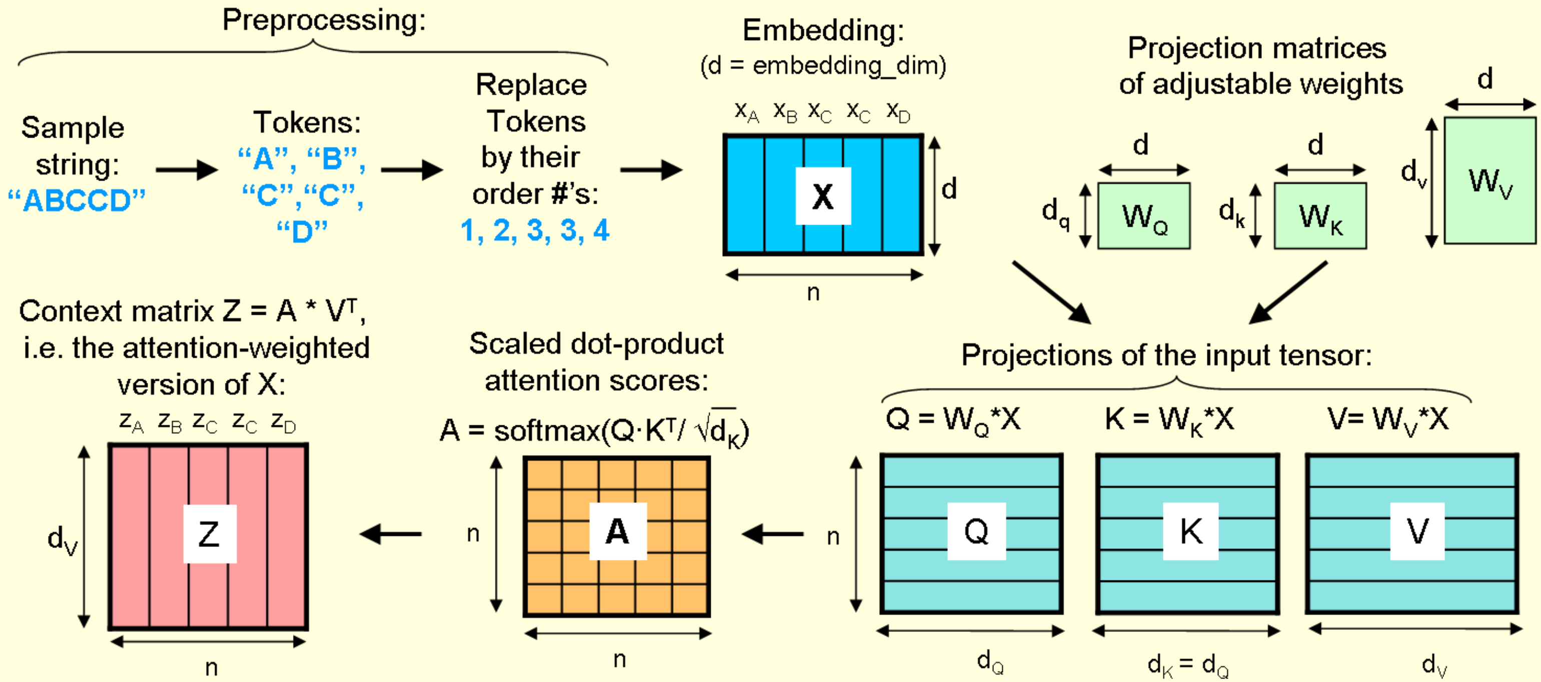
Prototype example #2: the Self-Attention algorithm

query, key, value, self attention, multi-head attention

https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html

Purpose: capture the **long-range dependencies and relationships** within input sequences; identify and **weigh the importance of different parts of the sequence,** with weight coeff. depending on input values.

Analogy with a Web search for a video on YouTube: the search engine will map your **query** (**Q**), i.e. the text in a search bar, against a set of **keys** (**K**), e.g. video title, description, etc., associated with candidate videos in a database, then present you **values** (**V**), e.g. the best-matched videos.



Preprocessing:

Sample string: "ABCCD" → Tokens: "A", "B", "C","C", "D" → Replace Tokens by their order #'s: 1, 2, 3, 3, 4 →

Embedding: (d = embedding_dim)

Projection matrices of adjustable weights

Projections of the input tensor:

$Q = W_Q{*}X$ $K = W_K{*}X$ $V= W_V{*}X$

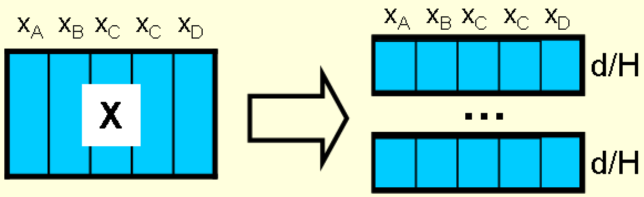
Scaled dot-product attention scores:

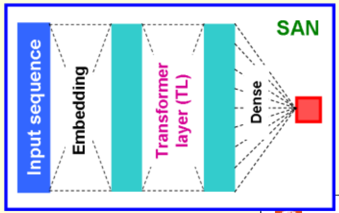
$A = \text{softmax}(Q{\cdot}K^T/\sqrt{d_K})$

Context matrix $Z = A * V^T$, i.e. the attention-weighted version of X:

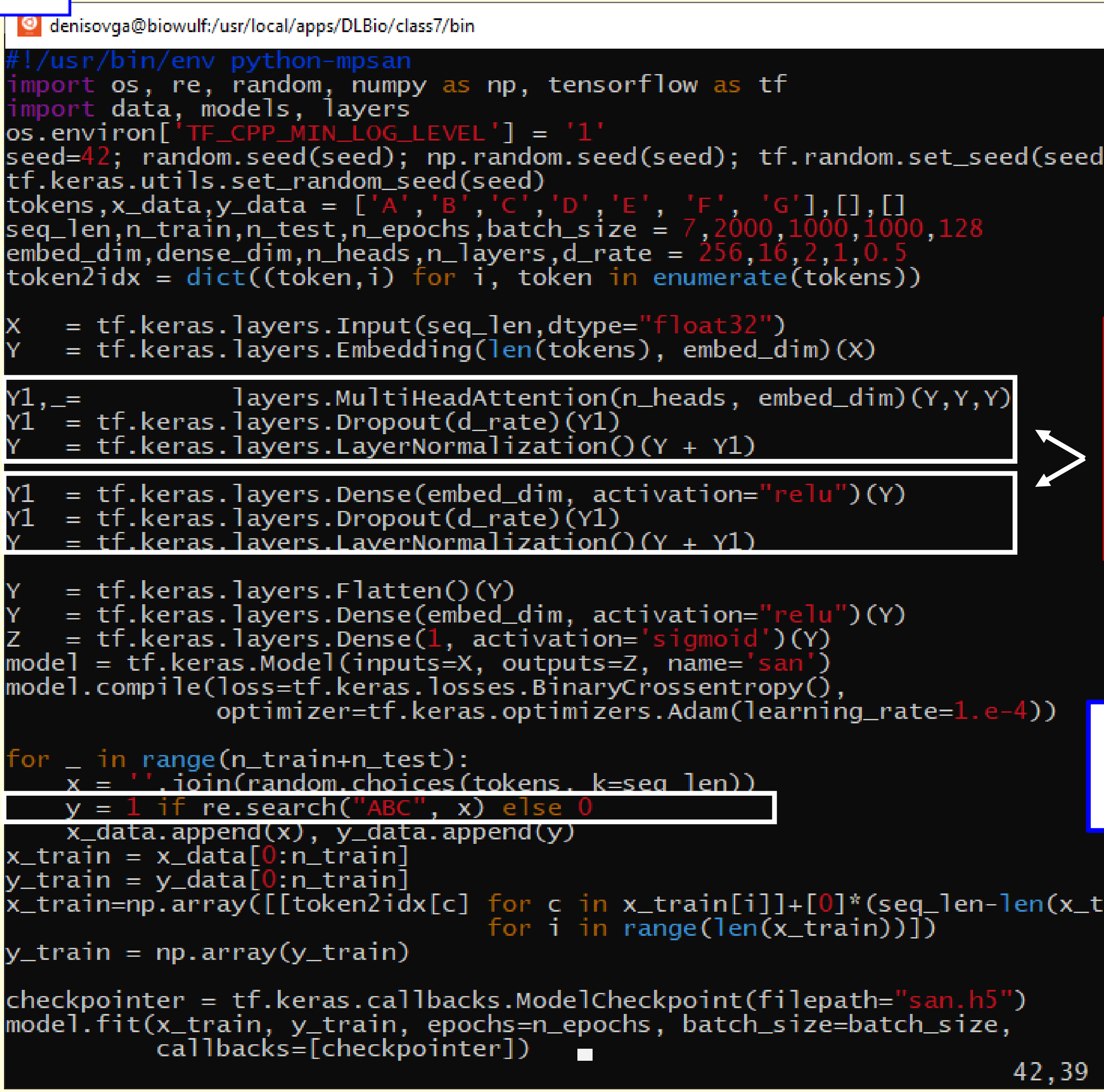
Multi-Head Attention:

- the algorithm outlined above represents a single "attention head"
- with H > 1 attention heads, tensor X will be split into H subtensors along the embedding dimension, then each the subtensor processed independently and the results concatenated.



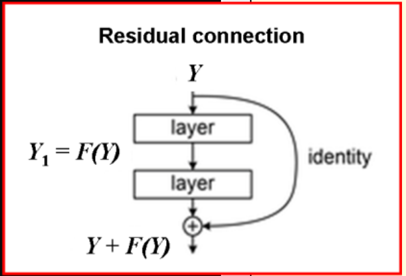


Prototype example #2 (cont.): the training code



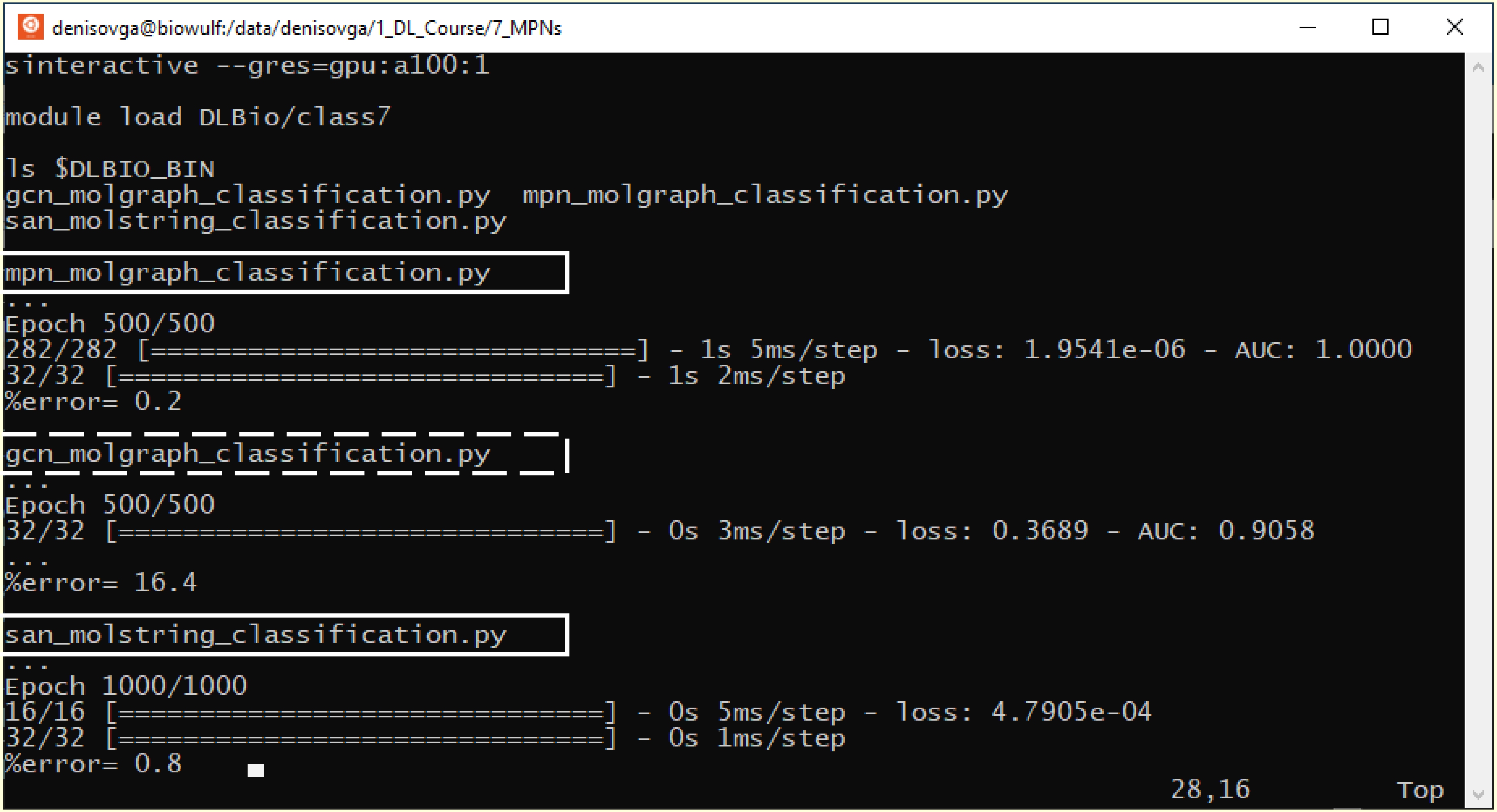
```
#!/usr/bin/env python-mpsan
import os, re, random, numpy as np, tensorflow as tf
import data, models, layers
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '1'
seed=42; random.seed(seed); np.random.seed(seed); tf.random.set_seed(seed)
tf.keras.utils.set_random_seed(seed)
tokens,x_data,y_data = ['A','B','C','D','E', 'F', 'G'],[],[]
seq_len,n_train,n_test,n_epochs,batch_size = 7,2000,1000,1000,128
embed_dim,dense_dim,n_heads,n_layers,d_rate = 256,16,2,1,0.5
token2idx = dict((token,i) for i, token in enumerate(tokens))

X   = tf.keras.layers.Input(seq_len,dtype="float32")
Y   = tf.keras.layers.Embedding(len(tokens), embed_dim)(X)

Y1,_=          layers.MultiHeadAttention(n_heads, embed_dim)(Y,Y,Y)
Y1  = tf.keras.layers.Dropout(d_rate)(Y1)
Y   = tf.keras.layers.LayerNormalization()(Y + Y1)

Y1  = tf.keras.layers.Dense(embed_dim, activation="relu")(Y)
Y1  = tf.keras.layers.Dropout(d_rate)(Y1)
Y   = tf.keras.layers.LayerNormalization()(Y + Y1)

Y   = tf.keras.layers.Flatten()(Y)
Y   = tf.keras.layers.Dense(embed_dim, activation="relu")(Y)
Z   = tf.keras.layers.Dense(1, activation='sigmoid')(Y)
model = tf.keras.Model(inputs=X, outputs=Z, name='san')
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(learning_rate=1.e-4))

for _ in range(n_train+n_test):
    x = ''.join(random.choices(tokens, k=seq_len))
    y = 1 if re.search("ABC", x) else 0
    x_data.append(x), y_data.append(y)
x_train = x_data[0:n_train]
y_train = y_data[0:n_train]
x_train=np.array([[token2idx[c] for c in x_train[i]]+[0]*(seq_len-len(x_train[i])) \
                                for i in range(len(x_train))])
y_train = np.array(y_train)

checkpointer = tf.keras.callbacks.ModelCheckpoint(filepath="san.h5")
model.fit(x_train, y_train, epochs=n_epochs, batch_size=batch_size,
          callbacks=[checkpointer])
                                                                      42,39        Top
```

Multihead Attention layer



Label assignment based on the presence of the motif “ABC”

How to run the prototype examples on Biowulf?



denisovga@biowulf:/data/denisovga/1_DL_Course/7_MPNs

```
sinteractive --gres=gpu:a100:1

module load DLBio/class7

ls $DLBIO_BIN
gcn_molgraph_classification.py  mpn_molgraph_classification.py
san_molstring_classification.py

mpn_molgraph_classification.py
...
Epoch 500/500
282/282 [==============================] - 1s 5ms/step - loss: 1.9541e-06 - AUC: 1.0000
32/32 [==============================] - 1s 2ms/step
%error= 0.2

gcn_molgraph_classification.py
...
Epoch 500/500
32/32 [==============================] - 0s 3ms/step - loss: 0.3689 - AUC: 0.9058
...
%error= 16.4

san_molstring_classification.py
...
Epoch 1000/1000
16/16 [==============================] - 0s 5ms/step - loss: 4.7905e-04
32/32 [==============================] - 0s 1ms/step
%error= 0.8
                                                                          28,16         Top
```

Biological example #7. MPSAN-MP: Message Passing and Self-Attention based Networks for Molecular Property prediction

https://hpc.nih.gov/apps/mpsan_mp.html

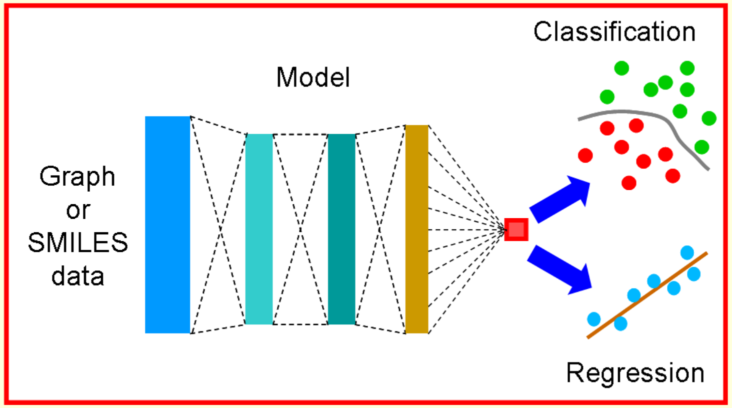
MPN model (Keras): https://keras.io/examples/graph/mpnn-molecular-graphs/

SMILES-BERT (PyTorch): https://github.com/uta-smile/SMILES-BERT

Molecular-graph-BERT (Keras): https://github.com/zhang-xuan1314/Molecular-graph-BERT

Task:

given a molecule represented by
either a **graph** or a sequence of **SMILES** tokens,
predict, depending on the type of data used,
- either a **discrete property value / binary label** (e.g. drug is “good” or “bad”),
- or a **continuous property value /label** (e.g. “how good” the drug is)



SMILES enumeration:

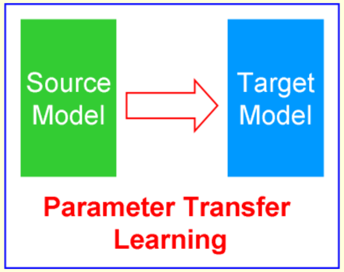
NC(=O)CCCl
O=C(CCCl) N
…

Problem:

limited amount of labeled ground truth data,
which is insufficient for training a full-scale target model,
leads to **over-fitting.**

Two solutions to explore

1) increase the size of a training dataset ⇒ **data augmentation**
2) decrease the # of adjustable parameters ⇒ **transfer learning**



Parameter Transfer Learning

The code overview

Header

- parse command line options

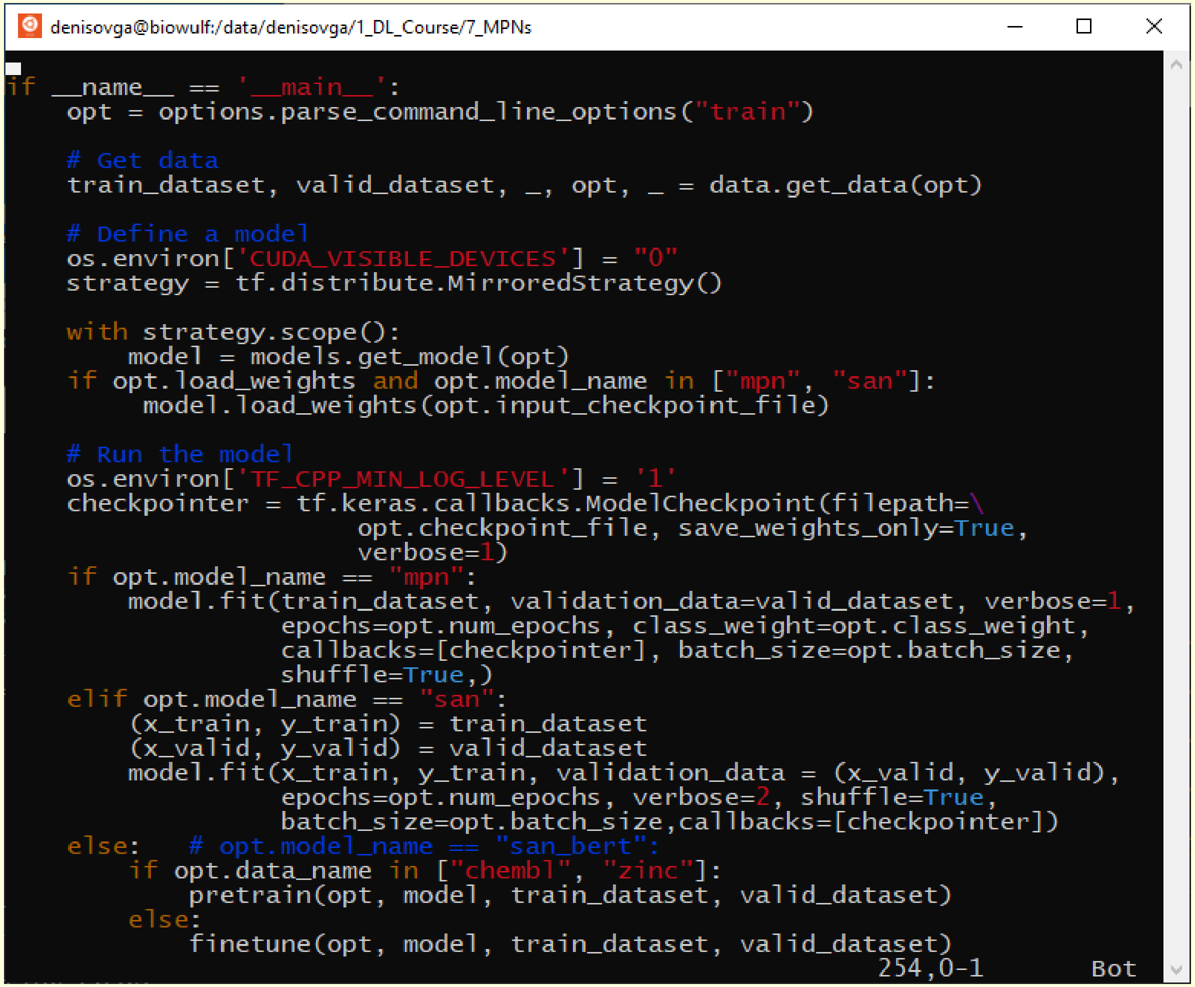
Get data: ChEMBL, ZINC, BBBP, JAK2, LogP, bLogP

Define model(s)

- MPN, SAN,
- BERT
- SAN-BERT

Run the models

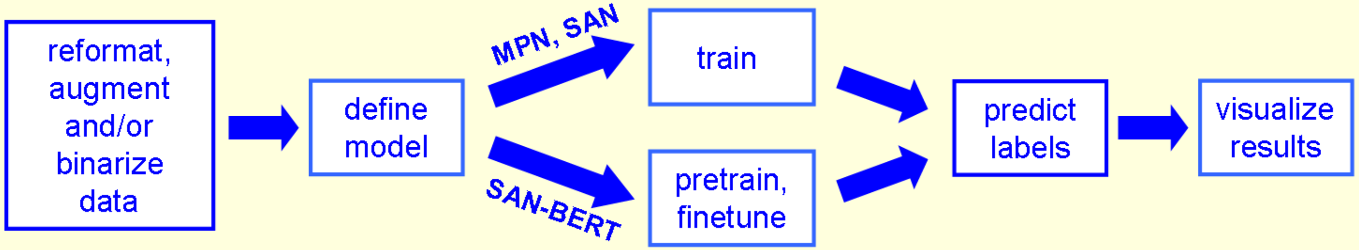
- train (=fit)
- pretrain
- finetune



denisovga@biowulf:/data/denisovga/1_DL_Course/7_MPNs

```
if __name__ == '__main__':
    opt = options.parse_command_line_options("train")

    # Get data
    train_dataset, valid_dataset, _, opt, _ = data.get_data(opt)

    # Define a model
    os.environ['CUDA_VISIBLE_DEVICES'] = "0"
    strategy = tf.distribute.MirroredStrategy()

    with strategy.scope():
        model = models.get_model(opt)
    if opt.load_weights and opt.model_name in ["mpn", "san"]:
         model.load_weights(opt.input_checkpoint_file)

    # Run the model
    os.environ['TF_CPP_MIN_LOG_LEVEL'] = '1'
    checkpointer = tf.keras.callbacks.ModelCheckpoint(filepath=\
                       opt.checkpoint_file, save_weights_only=True,
                       verbose=1)
    if opt.model_name == "mpn":
        model.fit(train_dataset, validation_data=valid_dataset, verbose=1,
                  epochs=opt.num_epochs, class_weight=opt.class_weight,
                  callbacks=[checkpointer], batch_size=opt.batch_size,
                  shuffle=True,)
    elif opt.model_name == "san":
        (x_train, y_train) = train_dataset
        (x_valid, y_valid) = valid_dataset
        model.fit(x_train, y_train, validation_data = (x_valid, y_valid),
                  epochs=opt.num_epochs, verbose=2, shuffle=True,
                  batch_size=opt.batch_size,callbacks=[checkpointer])
    else:   # opt.model_name == "san_bert":
        if opt.data_name in ["chembl", "zinc"]:
            pretrain(opt, model, train_dataset, valid_dataset)
        else:
            finetune(opt, model, train_dataset, valid_dataset)
                                                       254,0-1       Bot
```

Flowchart:



reformat, augment and/or binarize data

define model

MPN, SAN

SAN-BERT

train

pretrain, finetune

predict labels

visualize results

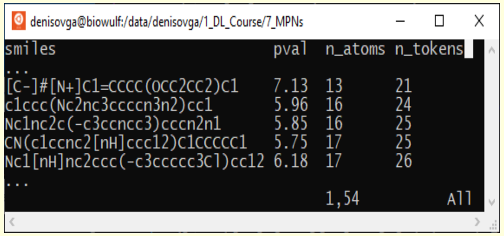
preprocess.py (optional) **train.py** **predict.py** **visualize.R**

MPSAN-MP data (preprocessed)

JAK2: https://github.com/isayev/ReLeaSE/blob/master/data/jak2_data.csv
LogP: https://github.com/isayev/ReLeaSE/blob/master/data/logP_labels.csv
BBBP: https://deepchemdata.s3-us-west-1.amazonaws.com/datasets/BBBP.csv
ChEMBL: https://ftp.ebi.ac.uk/pub/databases/chembl/ChEMBLdb/releases/chembl_33/
ZINC: https://bioinformaticsreview.com/20200720/how-to-download-small-molecules-for-virtual-screening-from-zinc-database/

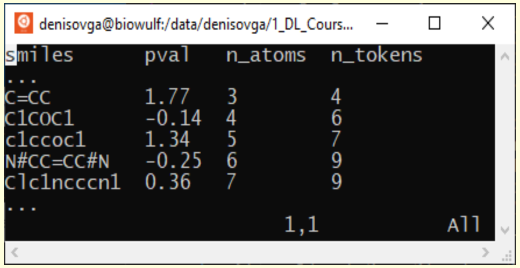
Continuous labels/PVs

jak2(a) dataset: ~2K
SMILES and cont. PV:
Janus protein kinase 2
inhibition coefficient.



```
smiles                        pval  n_atoms n_tokens
...
[C-]#[N+]C1=CCCC(OCC2CC2)C1    7.13  13      21
c1ccc(Nc2nc3ccccn3n2)cc1       5.96  16      24
Nc1nc2c(-c3ccncc3)cccn2n1      5.85  16      25
CN(c1ccnc2[nH]ccc12)C1CCCCC1   5.75  17      25
Nc1[nH]nc2ccc(-c3ccccc3Cl)cc12 6.18  17      26
...
                                     1,54          All
```

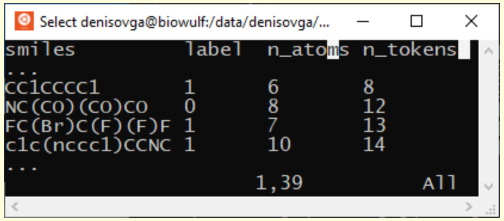
logp(a) dataset: ~14K
SMILE and cont. PV:
logarithm of n-octanol/
water partition coeff.



```
smiles      pval   n_atoms  n_tokens
...
C=CC        1.77   3        4
C1COC1      -0.14  4        6
c1ccoc1     1.34   5        7
N#CC=CC#N   -0.25  6        9
Clc1ncccn1  0.36   7        9
...
                        1,1           All
```

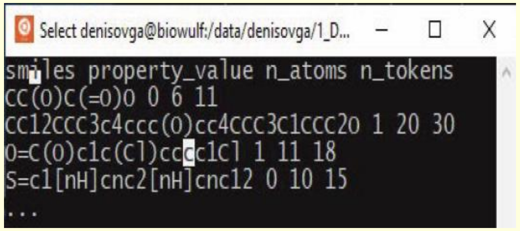
Discrete/binary labels/PVs

bbbp(a) dataset: ~2K
SMILES and binary label:
whether or not molecule
can penetrate blood/brain
barrier membrane



```
smiles          label  n_atoms n_tokens
...
Cc1cccc1        1      6       8
NC(CO)(CO)CO    0      8       12
FC(Br)C(F)(F)F  1      7       13
c1c(nccc1)CCNC  1      10      14
...
                     1,39          All
```

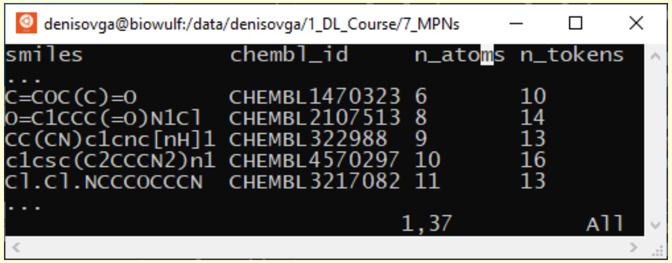
blogp(a) dataset: ~14K
SMILE and binary label:
"binarized" logp(a) dataset
label=1 if PV. > 1.88,
=0 otherwise.



```
smiles property_value n_atoms n_tokens
CC(O)C(=O)O 0 6 11
CC12CCC3c4ccc(O)cc4CCC3C1CCC2O 1 20 30
O=C(O)c1c(Cl)cccc1Cl 1 11 18
S=c1[nH]cnc2[nH]cnc12 0 10 15
...
```

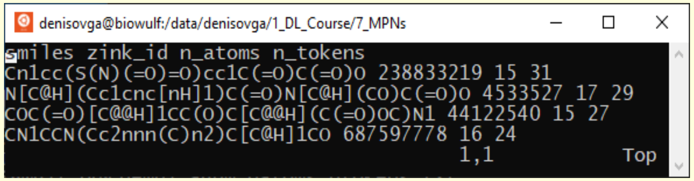
No labels/PVs

chembl: ~2.4M
SMILES, ChEMBL-33 database id



```
smiles              chembl_id       n_atoms n_tokens
...
C=COC(C)=O          CHEMBL1470323 6       10
O=C1CCC(=O)N1Cl     CHEMBL2107513 8       14
CC(CN)c1cnc[nH]1    CHEMBL322988  9       13
c1csc(C2CCCN2)n1    CHEMBL4570297 10      16
Cl.Cl.NCCCOCCCN     CHEMBL3217082 11      13
...
                              1,37          All
```

zinc: a random subset = 25M
SMILES, ZINC database id



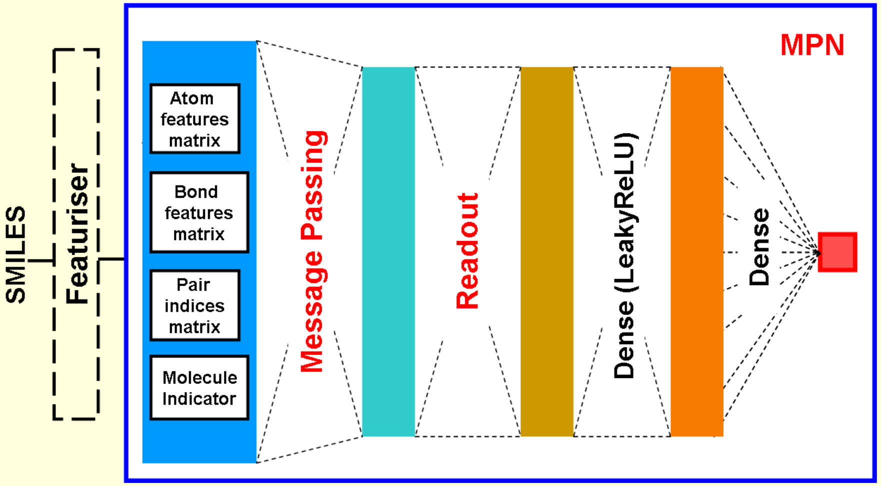
```
smiles zink_id n_atoms n_tokens
Cn1cc(S(N)(=O)=O)cc1C(=O)C(=O)O 238833219 15 31
N[C@H](Cc1cnc[nH]1)C(=O)N[C@H](CO)C(=O)O 4533527 17 29
COC(=O)[C@@H]1CC(O)C[C@@H](C(=O)OC)N1 44122540 15 27
CN1CCN(Cc2nnn(C)n2)C[C@H]1CO 687597778 16 24
                                           1,1          Top
```

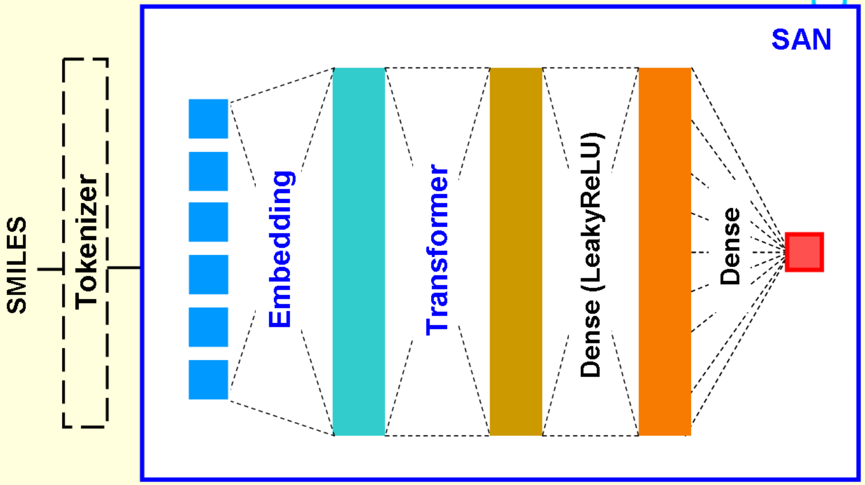
NOTEs:

- **suffix "a" stands for ~10x augmentation**
- **the *blogp* dataset was generated following the SMILES-BERT paper**

The effect of the data augmentation: predictions from the MPN and SAN models

SMILES enumeration / data augmentation : https://github.com/EBjerrum/SMILES-enumeration





(1) mean squared error in predicted PVs vs observed PVs, or
(2) %error in prediction of binary labels

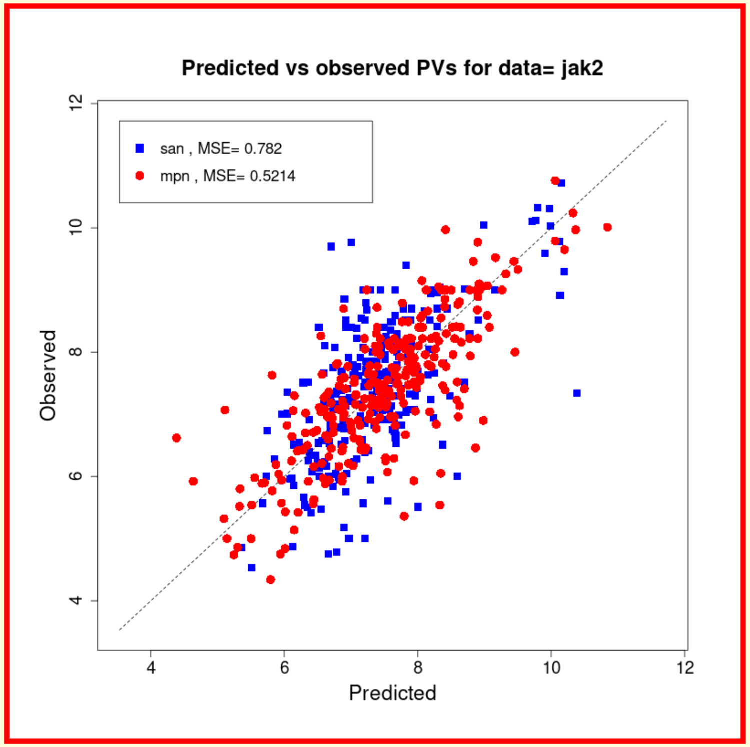
Target dataset (size) / Model (source dataset)	(1)				(2)			
	jak2 (1.9K)	*jak2a* (21K)	*logp* (14.1K)	*logpa* (142K)	*bbbp* (2.04K)	*bbbpa* (21K)	*blogp* (14.1K)	*blogpa* (142K)
MPN	0.52	0.039	0.16	0.019	11.51%	0.98%	7.24%	0.05%
SAN	0.78	0.43	0.61	0.28	13.1%	4.85%	16.2%	21%

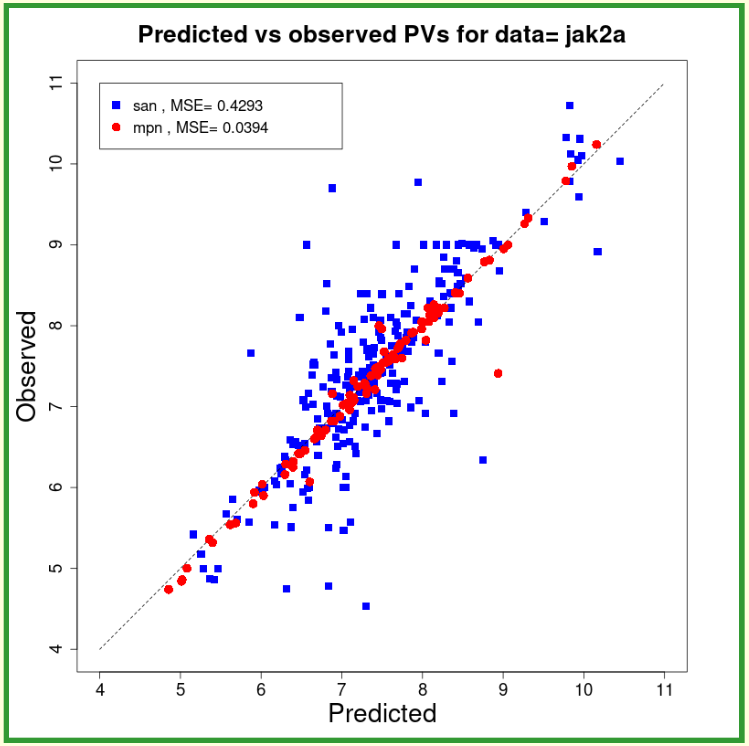
Conclusion

- data augmentation dramatically and consistently improves PV predictions from the MPN
- according to the published literature, this phenomenon was unknown prior to the class

Why the data augmentation works well for the MPN model, and not for SAN model?

https://hpc.nih.gov/apps/mpsan_mp.html





Original *jak2* data: both the models overfit the data

The ~10x augmented *jak2* data: MPN dramatically outperforms SAN

CONCLUSION:

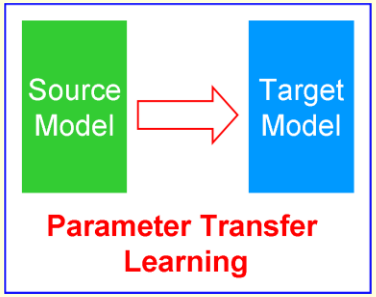
- **SAN: computing the attention scores between the tokens in the input sequence located at different positions is confused by their reshuffling as the result of SMILES enumeration.**
- **MPN: SMILES enumeration does not affect the molecular graph, which is unique, so the model will only benefit from the increased size of the augmented training dataset.**

Transfer learning: the SMILES-BERT approach

BERT: Bidirectional Encoder Representation from Transformers, J.Devlin et al., arXiv 2019

SMILES-BERT paper: S.Wang et al, ACM-BCB ’19, September 7–10, 2019

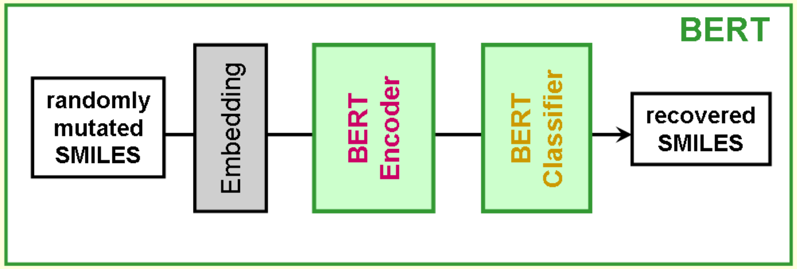
Q: Is there a way we could do better with the SAN-based model as well?



Transfer learning:
knowledge learned from a task is **re-used** in order to boost performance on a **related task**

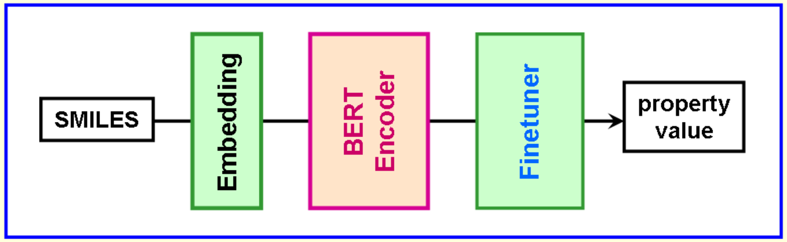
The SMILES-BERT setup:
the **source** model (**BERT**) and the **target** model **share** a block of layers known as **BERT_Encoder**

Pretraining on unlabeled data



🟩 trainable 🟥 weights frozen

Finetuning on labeled data



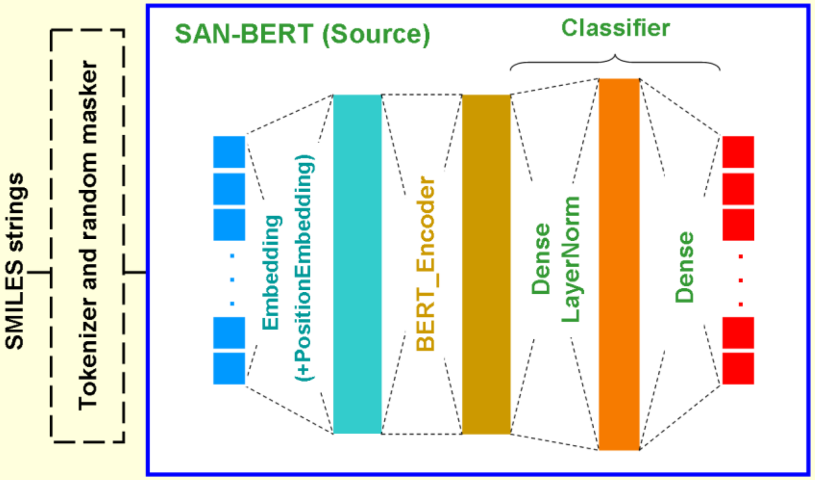
Two stages of training the SMILES-BERT model:

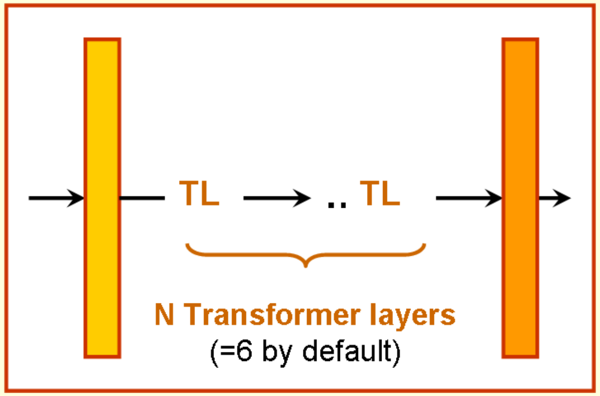
1) pretraining the (auxiliary) **source model**
on the vast amount of unlabeled data
employs the **masked language modeling (MLM):**
- input and output sequences of the same length
- a fraction of input tokens is randomly **mutated** by **masking** or **substitution** with other tokens
- model is trained to **recover the mutated tokens**

2) **finetuning** the **target model** on labeled data:
- **re-use** BERT_Encoder with **frozen parameters**
- train the target model to **output property values**

Transfer learning: pre-training the SAN-BERT model

https://hpc.nih.gov/apps/MPSAN_MP.htm





BERT_Encoder

The SAN-BERT source model

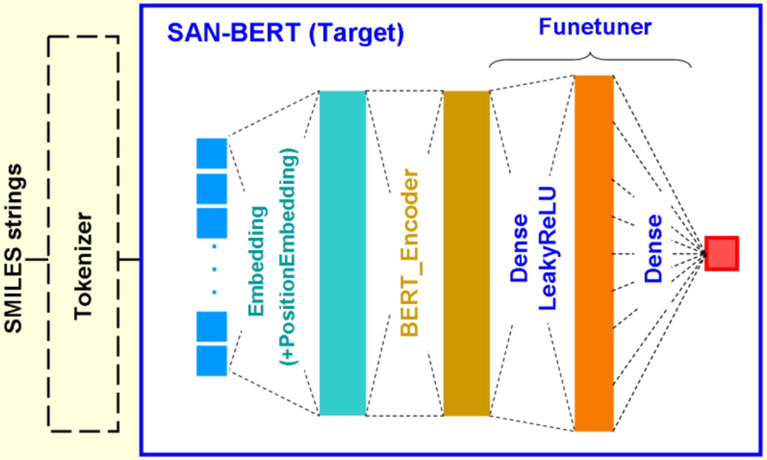
- similar to **SMILES-BERT**, but **implemented in Keras**
- **BERT_Encoder:** N(=6) consecutive Transformer layers
- **Classifier**: 2 Dense layers + layer(s) without adjustable params
- inputs and outputs sequences of the **length specified by user** (default = 64); discards longer and pads shorter sequences
- employs **SmilesPE tokenizer**,
  vocabulary = [‘<pad>’, ‘#', '%10’, …, '-', '/’, '1', '2’, …,'=', 'B’,
  'Br', 'C’, 'Cl’, …, '[NH-]', '[NH2+]', …, '[n+]', '[n-]’,
  '[nH+]', …, '\\', 'o', 'p’, …, **‘<unk>’, <mask>’]**
  # total = 110

Pretraining loss (accuracy)

Data (size) / Model	*chembl* (2.4M)	*zinc* (25M)
SAN-BERT	0.00014 (99%)	0.006 (96%)

chembl data: 30 epochs
zinc data: 10 epochs, as advised by the SMILES-BERT paper

Transfer learning: accuracy of predictions from the fine-tuned SAN-BERT model



(1) MSE in predicted PVs vs observed PVs,
(2) %error in prediction of binary labels

Target data Model (source data)	(1)		(2)	
	jak2	*logp*	*bbbp*	*blogp*
SAN	0.78	0.61	13.1%	16.2%
SAN-BERT (chembl)	0.9	0.7	6.3%	18.6%
SAN-BERT (zinc)	0.8	0.56	9.3%	12.4%

error comparable to the ~8.5% error reported in the SMILES-BERT paper for the same data

The SAN-BERT target model

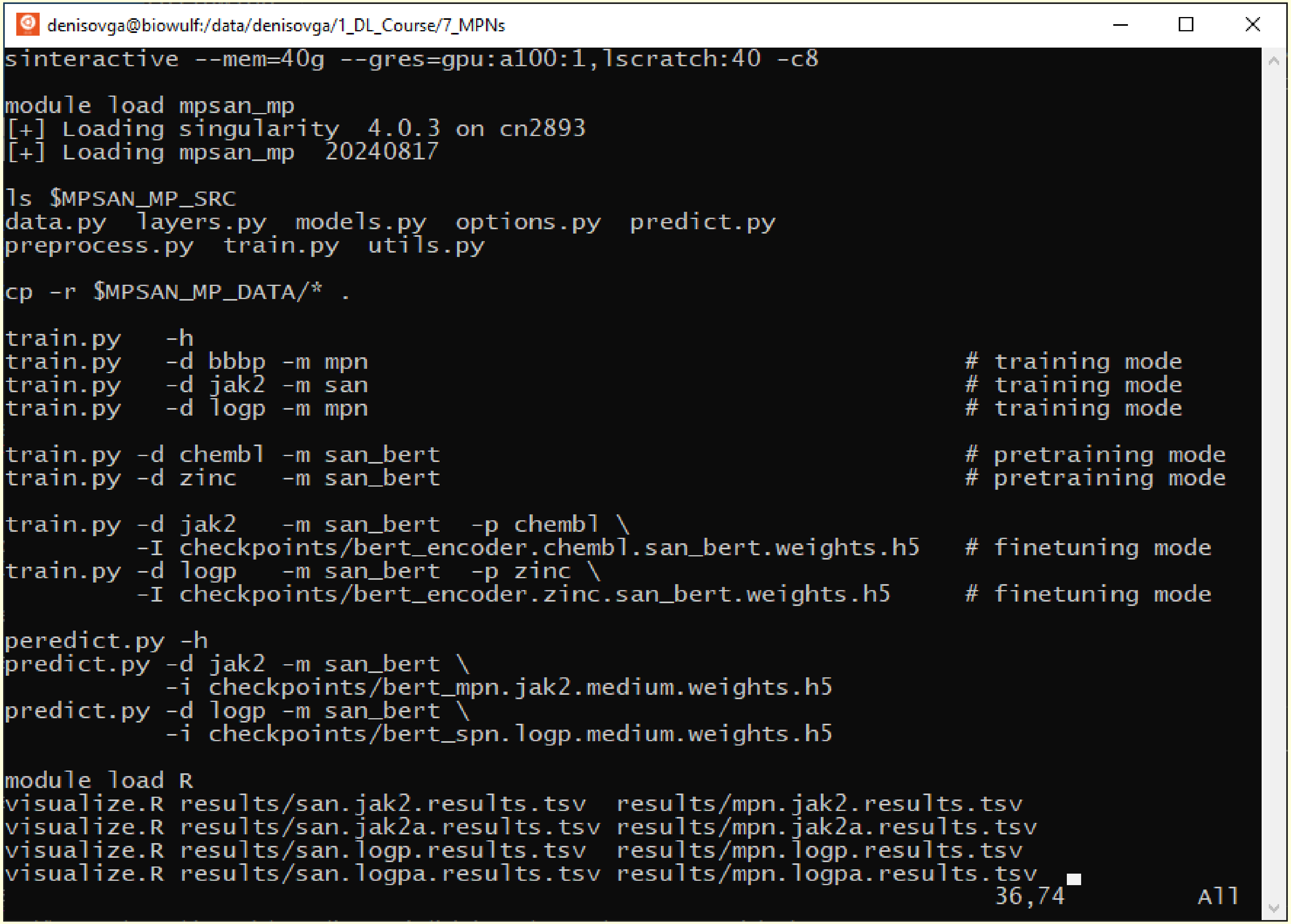
- inputs **unmutated** sequences
- re-uses the BERT_Encoder block with **frozen parameters**
- employs the **Finetuner** block instead of Classifier
- is trained to output the scalar **property values**

CONCLUSIONS:

- the MPN model using data enhanced by augmentation remains the winner
- the pretrained BERT_Encoder block may not provide optimal inference performance for the prediction task, since the pretraining stage was focused on a quite different (MLM) task.
- more generally, the transfer learning which aims to re-use knowledge learned from one task in order boost performance on a related task, may or may not succeed, since the notion of a related task is not strictly defined.

How to run the MPSAN-MP application on Biowulf?

https://hpc.nih.gov/apps/mpsan_mp.html



```
sinteractive --mem=40g --gres=gpu:a100:1,lscratch:40 -c8

module load mpsan_mp
[+] Loading singularity  4.0.3 on cn2893
[+] Loading mpsan_mp  20240817

ls $MPSAN_MP_SRC
data.py  layers.py  models.py  options.py  predict.py
preprocess.py  train.py  utils.py

cp -r $MPSAN_MP_DATA/* .

train.py   -h
train.py   -d bbbp -m mpn                                   # training mode
train.py   -d jak2 -m san                                   # training mode
train.py   -d logp -m mpn                                   # training mode

train.py -d chembl -m san_bert                              # pretraining mode
train.py -d zinc   -m san_bert                              # pretraining mode

train.py -d jak2   -m san_bert  -p chembl \
         -I checkpoints/bert_encoder.chembl.san_bert.weights.h5   # finetuning mode
train.py -d logp   -m san_bert  -p zinc \
         -I checkpoints/bert_encoder.zinc.san_bert.weights.h5     # finetuning mode

peredict.py -h
predict.py -d jak2 -m san_bert \
           -i checkpoints/bert_mpn.jak2.medium.weights.h5
predict.py -d logp -m san_bert \
           -i checkpoints/bert_spn.logp.medium.weights.h5

module load R
visualize.R results/san.jak2.results.tsv  results/mpn.jak2.results.tsv
visualize.R results/san.jak2a.results.tsv results/mpn.jak2a.results.tsv
visualize.R results/san.logp.results.tsv  results/mpn.logp.results.tsv
visualize.R results/san.logpa.results.tsv results/mpn.logpa.results.tsv
                                                                  36,74         All
```

Summary and conclusions

1) The introductory part:

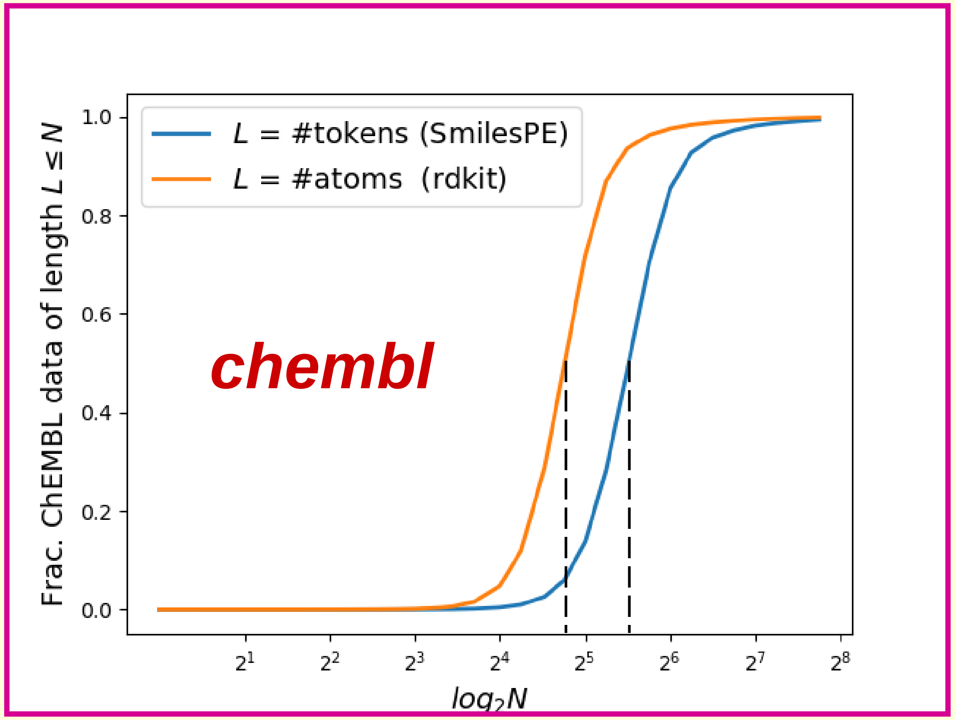
- cartoon examples have been used to introduce (a) the Message Passing mechanism/ network model (MPN) and (b) the Self-Attention Network mechanism / model (SAN)
- the Message Passing layer / data transformation (a) employs both node and edge features of a graph, and (b) generalizes the vanilla Graph Convolution layer
- like the Graph Convolution, the Message Passing is a local transformation of data
- the Self-Attention mechanism captures the long-range relationships and dependencies within input sequences

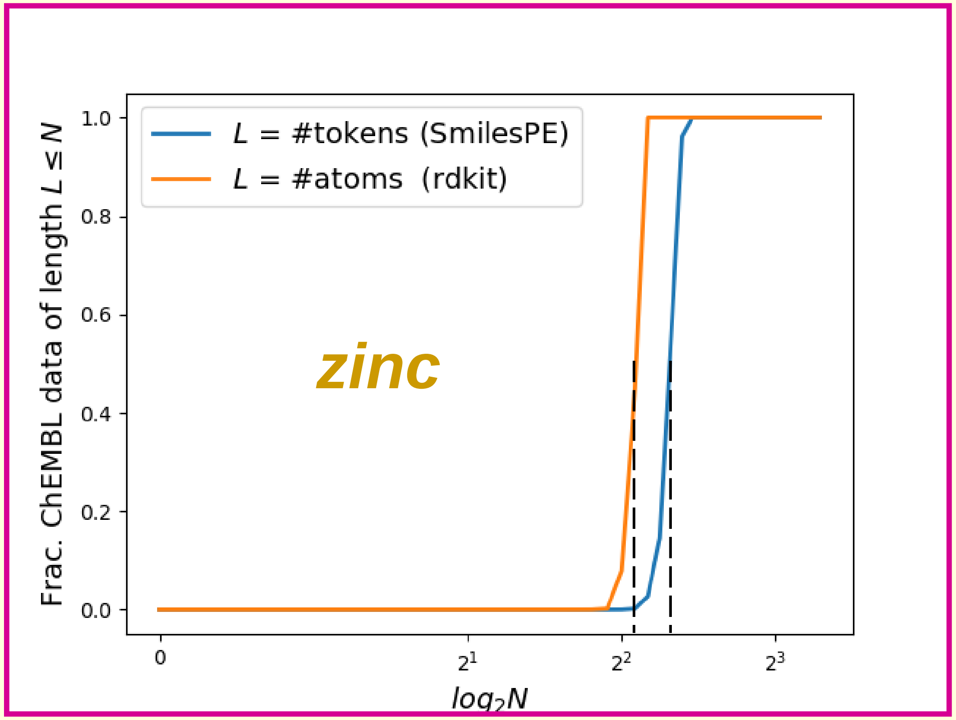
2) The biological example:

- the MPSAN-MP (a) employs neural networks implementing Message Passing or Self-Attention-based models, (b) depending on the model used, takes as input a set of molecules represented by either graphs or SMILES strings, (c) for each molecule, depending on the type of data used, predicts either a discrete/binary property value (PV) (classification task) or a continuous label/PV (regression task)
- the overfitting issue in PV prediction can be addressed via two approaches:
  (a) using MPN model on data augmented by SMILES enumeration, and
  (b) to some extent, using the SAN-BERT model and transfer learning
- the first of these approaches is preferable / a clear winner; it allows for a dramatic and consistent improvement in the accuracy of PV predictions
- the transfer learning approach may or may not succeed, depending on how much the source/auxiliary task/data is related to the target task/data

BACKUP SLIDES

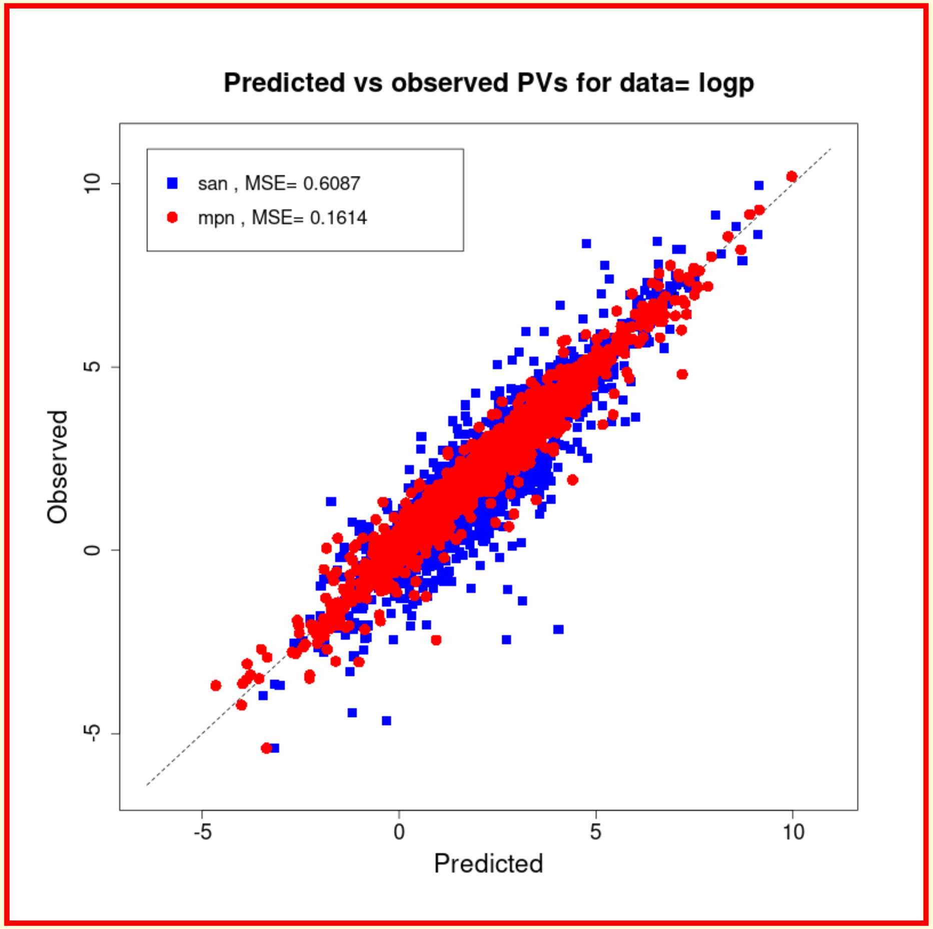
Cumulative distribution of the SMILES sizes (#tokens, #atoms) in ChEMBL and ZINC data

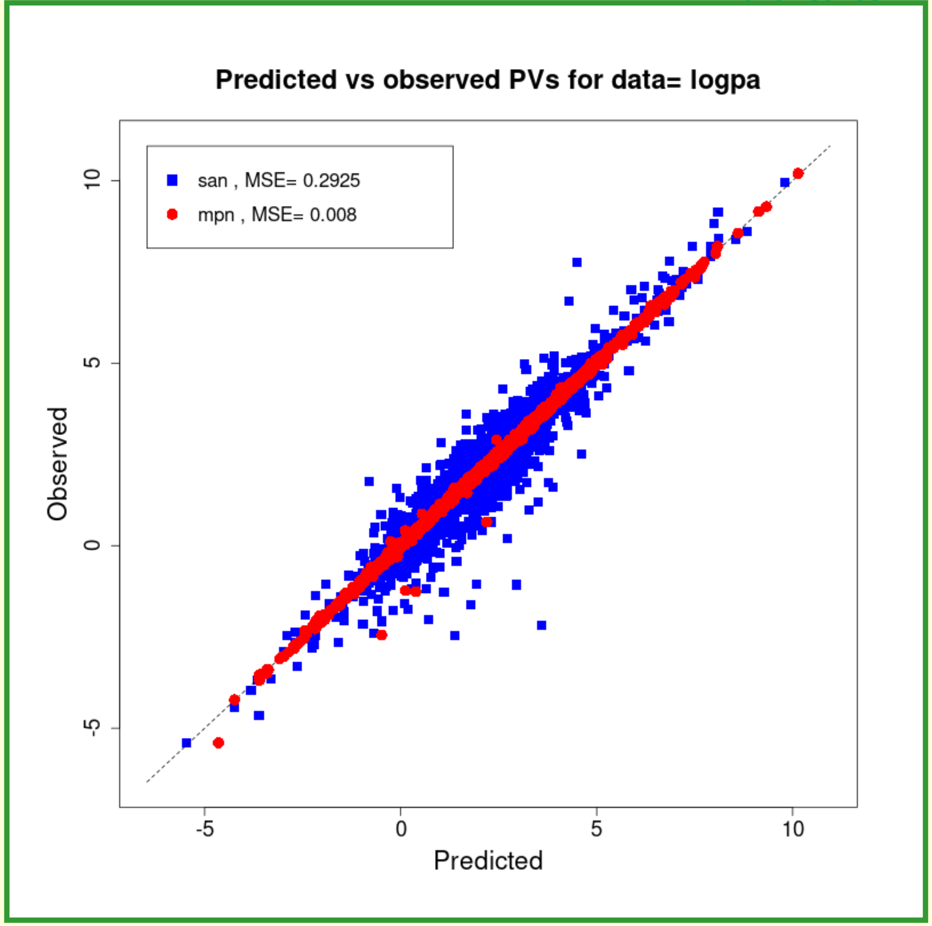




CONCLUSION:
in the *chembl* data, the SMILES sizes are loosely distributed around the values: # tokens ~45 and # atoms ~27, respectively, whereas in *zinc* data, they are more sharply focused near the values: # tokens ~32 and # atoms ~ 19.

Data augmentation works well for the MPN model on *logp* data





Original *logp* dataset:
both the models perform better
than on the smaller *jak2* dataset

The ~10x augmented *logp* dataset:
MPN dramatically outperforms SAN