

Deep Learning

20230323



Reference

Platform



Book



DIVE INTO
DEEP LEARNING

PyTorch: <https://pytorch.org/>

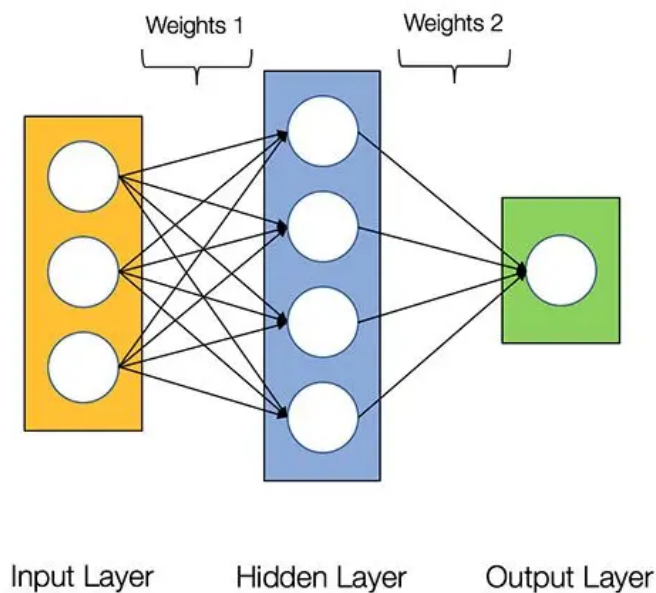
D2L: https://zh.d2l.ai/chapter_installation/index.html



Introduction

Data preprocessing → Input data

Normalization
Feature extraction



Output Label } Loss → **Evaluation and prediction**

Finetune
Downstream task

Task: Regression, Classification .etc



Introduction

[nature](#) > [nature communications](#) > [articles](#) > [article](#)

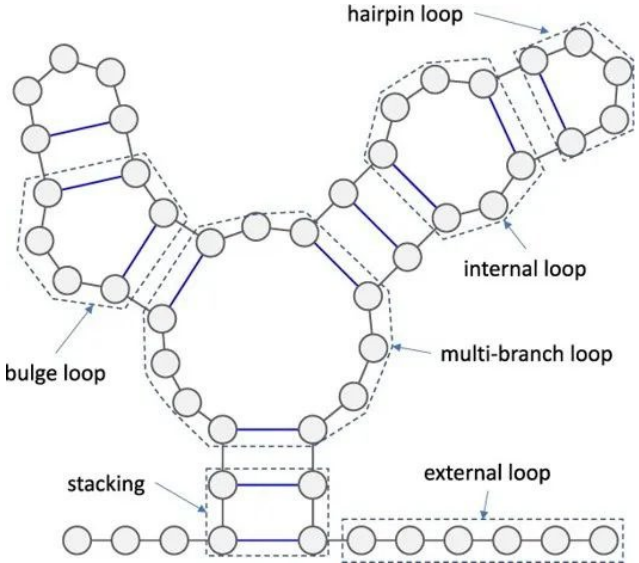
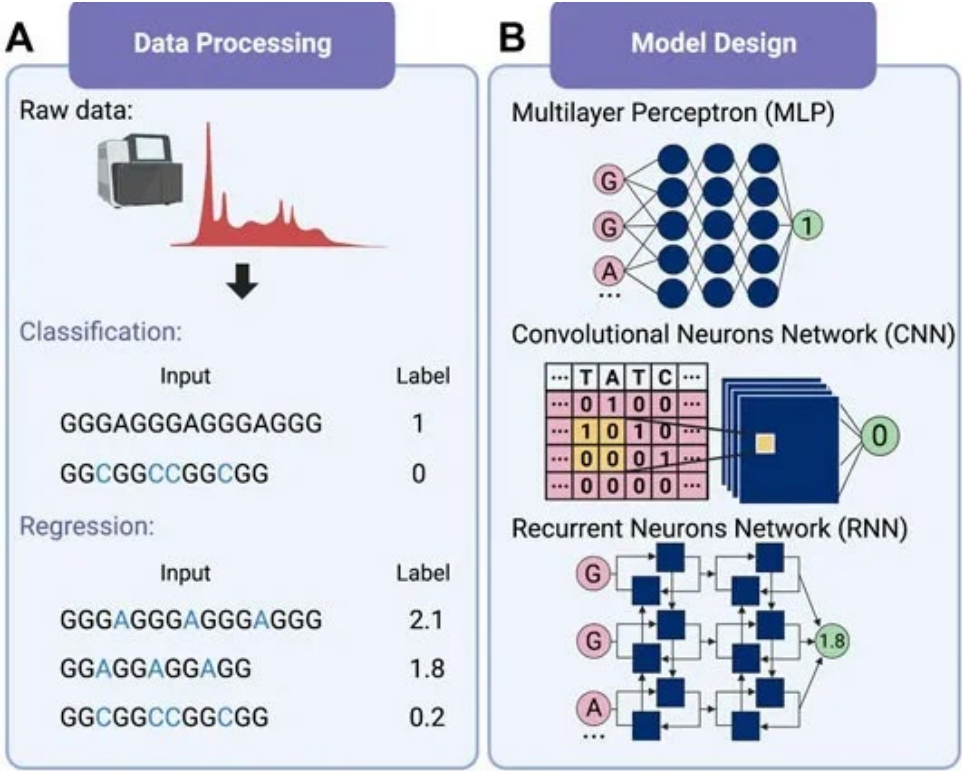
Article | [Open access](#) | [Published: 27 November 2019](#)

RNA secondary structure prediction using an ensemble of two-dimensional deep neural networks and transfer learning

[Jaswinder Singh](#), [Jack Hanson](#), [Kuldip Paliwal](#) & [Yaoqi Zhou](#)

[Nature Communications](#) **10**, Article number: 5407 (2019) | [Cite this article](#)

46k Accesses | 167 Citations | 71 Altmetric | [Metrics](#)

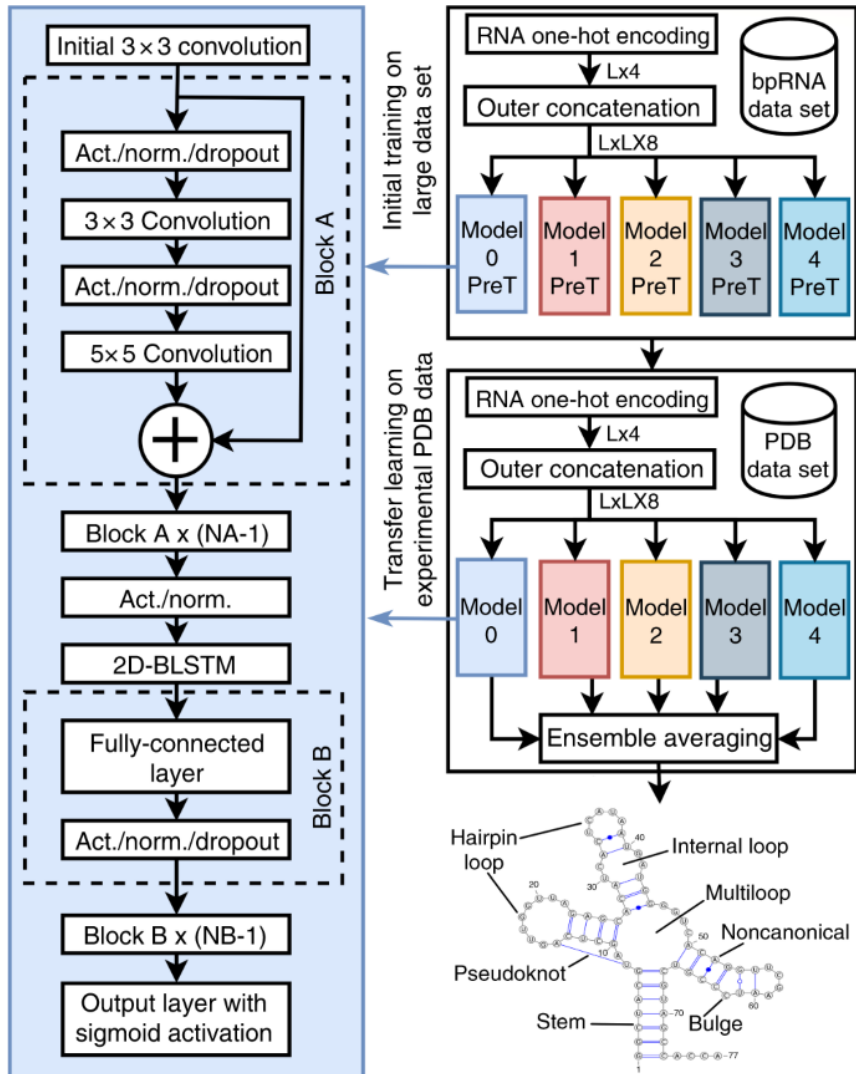


Pairing Features

G	0.00	4.82	0.00	0.00	2.62	0.80	0.00	0.80	0.00	0.00
C	4.82	0.00	4.82	3.49	0.00	0.00	0.00	0.00	0.00	3.00
G	0.00	4.82	0.00	0.00	0.80	1.40	0.00	0.80	0.00	0.00
G	0.00	3.49	0.00	0.00	1.88	0.80	0.00	2.01	0.00	0.00
U	2.62	0.00	0.80	1.88	0.00	0.00	2.49	0.00	2.00	2.31
U	0.80	0.00	1.40	0.80	0.00	0.00	3.21	0.00	3.99	0.80
A	0.00	0.00	0.00	0.00	2.49	3.21	0.00	4.81	0.00	0.00
U	0.80	0.00	0.80	2.01	0.00	0.00	4.81	0.00	3.21	0.80
A	0.00	0.00	0.00	0.00	2.00	3.99	0.00	3.21	0.00	0.00
G	0.00	3.00	0.00	0.00	2.31	0.80	0.00	0.80	0.00	0.00
G		C	G	G	U	U	A	U	A	G

From UFold

Model



- Data preprocessing: One hot encoding and concatenation
- Initial convolution layer:
- Resnet blocks(CNN): extract the feature of image like matrix
- BLSTM layer(RNN): learn the information in the context
- Full connected layer blocks: increase the robustness of model
- Output layer:

100

[illegible]

>1c0a-1-B
GGAGCGGUAGUUCAGUCGGUAGAAUACCUGCCUGUCACGCAGGGGGUCGCGGGTPCGAGUCCCGPCCGUUCCGCCA

```
# 1c0a-1-B L = 77
i j
1 73
2 72
3 71
4 70
5 69
6 68
7 67
8 14
8 22
9 24
10 26
10 46
11 25
12 24
13 23
15 49
18 56
19 20
```

```
def generate_pairing_matrix_bpRNA(filename,pseudo):
    with open(filename, 'r') as file:
        lines = file.readlines()
        if '(' and '.' and ')' in lines[4]:
            sequence = lines[4][:-1]
            aucg = lines[3][:-1]
            n = len(sequence)
            stack = []
            pairing_matrix = [[0] * n for _ in range(n)]

            # Iterate over each character in the sequence
            for i, char in enumerate(sequence):
                if char == '(':
                    stack.append(i) # Push the index of '(' to the stack
                elif char == ')':
                    if stack: # Ensure stack is not empty
                        left_index = stack.pop() # Pop the index of '(' from the stack
                        pairing_matrix[left_index][i] = 1 # Mark the pair in the matrix
                        pairing_matrix[i][left_index] = 1 # Mark the pair in the matrix (symmetric)
                else: # 假结怎么办呢？怎么标记？？
                    aucg = 0
                    sequence = lines[5][:-1]
                    n = len(sequence)
                    pseudo.append(sequence)
                    pairing_matrix = 0

    return pseudo, aucg, pairing_matrix
```

```
def create_files_bprnn(sequence):
    seq, feature, one_hot = get_data(sequence)
    return seq, feature, one_hot

def one_hot(seq):
    RNN_seq = seq
    BASES = 'AUCG'
    bases = np.array([base for base in BASES])
    feat = np.concatenate(
        [[(bases == base.upper()).astype(int)] if str(base).upper() in BASES else np.array([[-1] * len(BASES)]] for base
         in RNN_seq))

    return feat

def get_data(seq):
    seq_len = len(seq)
    one_hot_feat = one_hot(seq)
    #print(one_hot_feat[-1])
    #zero_mask = z_mask(seq_len)[None, :, :, None]
    #label_mask = l_mask(one_hot_feat, seq_len)
    temp = one_hot_feat[None, :, :]
    temp = np.tile(temp, (temp.shape[1], 1, 1))
    feature = np.concatenate([temp, np.transpose(temp, [1, 0, 2])], 2)

#out = true_output
    return seq_len, [i for i in (feature.astype(float)).flatten()], one_hot_feat.reshape(1,-1,4)
```

100

Data encoding

[illegible]

```
train_x = np.load('../datasets/bpRNA_dataset/train_x.npy', allow_pickle=True)
test_x = np.load('../datasets/bpRNA_dataset/test_x.npy', allow_pickle=True)
train_y = np.load('../datasets/bpRNA_dataset/train_y.npy', allow_pickle=True)
test_y = np.load('../datasets/bpRNA_dataset/test_y.npy', allow_pickle=True)
```

One hot

```
one_hot('ACGA')
```

```
array([[1, 0, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1],
       [1, 0, 0, 0]])
```

Pairing matrix

	A	C	G	A
A	0	0	0	0
C	0	0	1	0
G	0	1	0	0
A	0	0	0	0

L*L*8 matrix

```
array([[1., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 1., 0., 0., 0.],
       [1., 0., 0., 0., 1., 0., 0., 0.]])

array([[1., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 1., 0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0., 0., 1., 0.],
       [1., 0., 0., 0., 0., 0., 1., 0.]])

array([[1., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 1., 0., 0., 0., 0., 1.],
       [0., 0., 0., 1., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 1.]])

array([[1., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 1., 0., 0., 0.],
       [1., 0., 0., 0., 1., 0., 0., 0.]])
```

Train_x: Sequence with AUCG
Train_y: L*L*8 pairing matrix

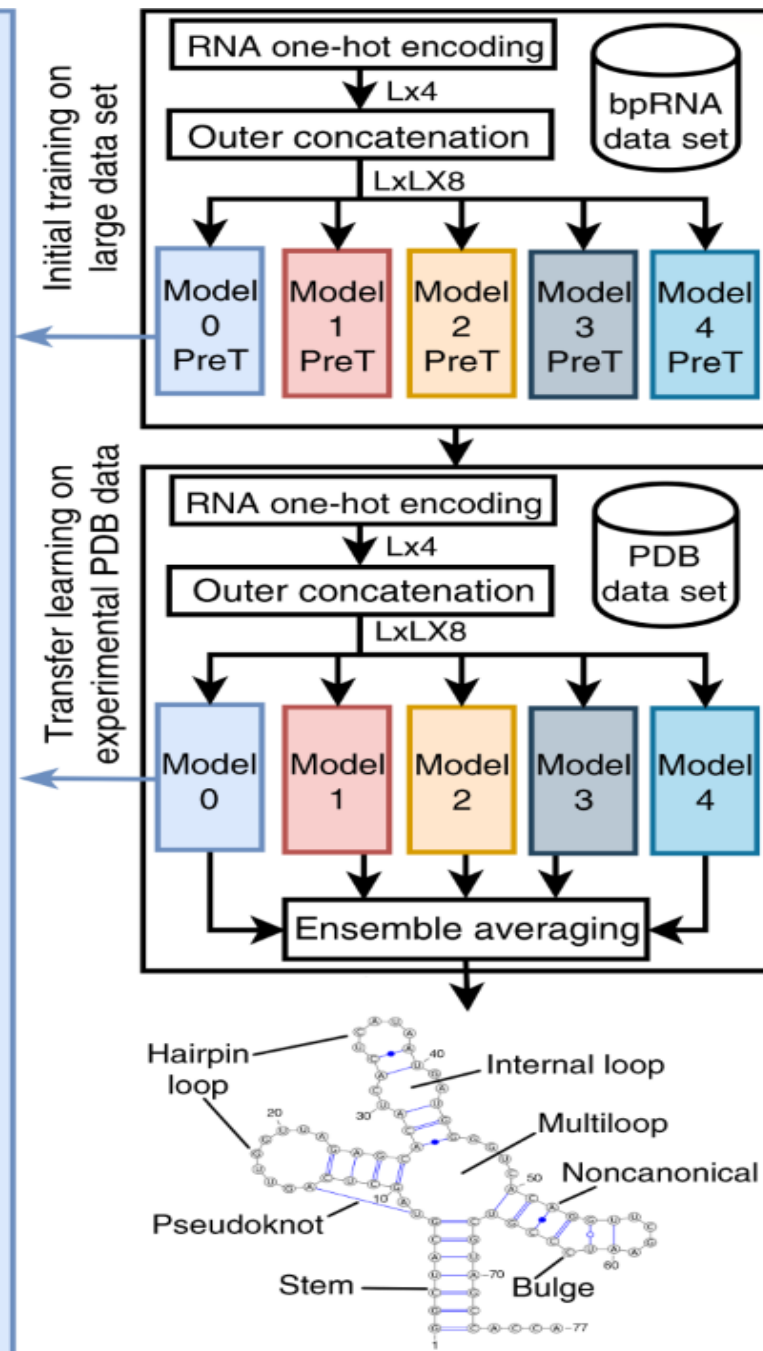
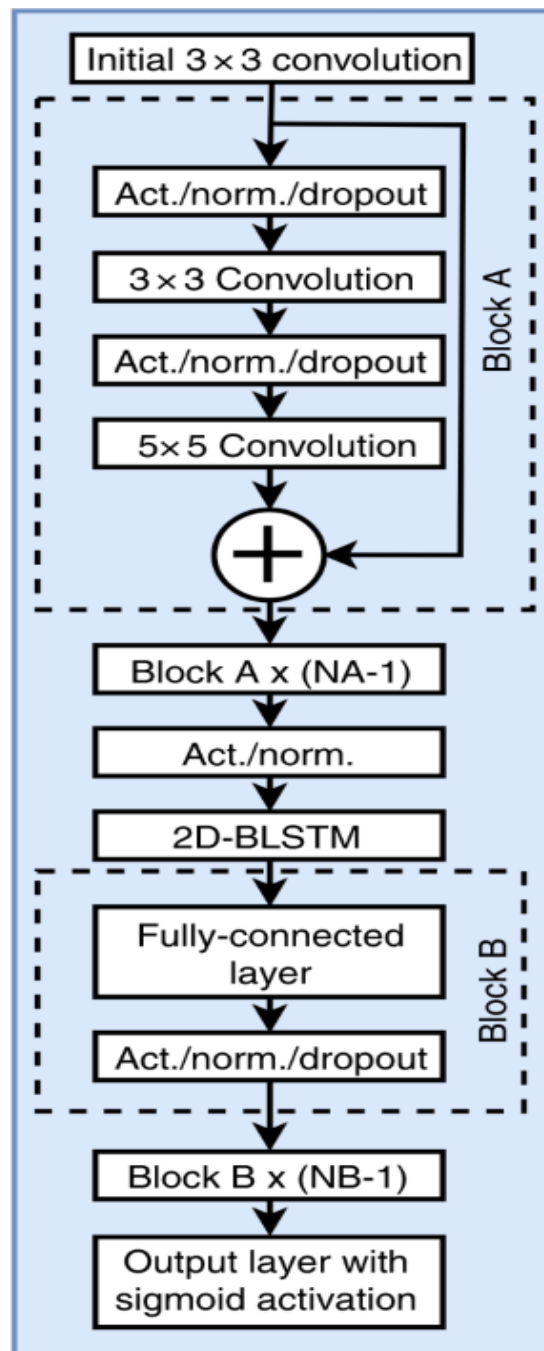
Model and datasets

```
class ResidualBlock(nn.Module):
    def __init__(self, dilation=1):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1, bias=True)
        self.conv2 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=5, padding=2, bias=True)
        self.norm = nn.LayerNorm([64]) #BN层, 不是norm层! 建议将bn层放在卷积层 (Conv) 和激活层 (例如Relu)

    def forward(self, x):
        residual = x
        #self.norm = nn.LayerNorm([64, x.shape[-1], x.shape[-1]])
        out = self.conv1(x).permute(0, 2, 3, 1)
        out = self.norm(out).permute(0, 3, 2, 1)
        out = F.dropout(F.elu(out), p=0.25, training=self.training)
        out = self.conv2(x).permute(0, 2, 3, 1)
        out = self.norm(out).permute(0, 3, 2, 1)
        out = F.dropout(F.elu(out), p=0.25, training=self.training)
        out += residual
        return out

class EnsembleModel(nn.Module):
    def __init__(self, num_res_blocks):
        super(EnsembleModel, self).__init__()
        self.conv3 = nn.Conv2d(in_channels=8, out_channels=64, kernel_size=3, padding=1, bias=True)
        self.res_blocks = nn.ModuleList([ResidualBlock() for _ in range(num_res_blocks)])
        self.blstm = nn.LSTM(input_size=Dbl, hidden_size=800, bidirectional=True, batch_first=True)
        self.fc = nn.Linear(64, 512)
        self.fc_blocks = nn.ModuleList([nn.Linear(512, 512) for _ in range(Nfc-1)])
        self.output_fc = nn.Linear(512, 1)
        self.norm1 = nn.LayerNorm([64])
        self.norm2 = nn.LayerNorm([512])
        #self.norm = nn.BatchNorm2d(64) #BN层, 不是norm层! 建议将bn层放在卷积层 (Conv) 和激活层 (例如Relu)

    def forward(self, x):
        x = x.permute(2, 0, 1)
        out = self.conv3(x).unsqueeze(0)
        for block in self.res_blocks:
            out = block(out)
        out = out.permute(0, 2, 3, 1)
        out = F.elu(self.norm1(out))
        out = out.permute(0, 3, 2, 1)
        if bilstm > 0:
            out, (h_n, c_n) = self.blstm(out)
        out = out.permute(0, 2, 3, 1)
        out = F.dropout(F.elu(self.norm2(self.fc(out))), p=0.5, training=self.training)
        for fc_block in self.fc_blocks:
            out = F.dropout(F.elu(self.norm2(fc_block(out))), p=0.5, training=self.training)
        out = self.output_fc(out)
        out = torch.sigmoid(out)
        return out
```



Train and evaluation

From UFold

```
def evaluate_exact_new(pred_a, true_a, eps=1e-11):

    tp_map = torch.sign(torch.Tensor(pred_a)*torch.Tensor(true_a))
    tp = tp_map.sum()
    pred_p = torch.sign(torch.Tensor(pred_a)).sum()
    true_p = true_a.sum()
    fp = pred_p - tp
    fn = true_p - tp
    tn = pred_a.shape[0]-(tp+fp+fn)

    recall = (tp + eps)/(tp+fn+eps)
    precision = (tp + eps)/(tp+fp+eps)
    f1_score = (2*tp + eps)/(2*tp + fp + fn + eps)
    #mcc = (tp * tn - fp * fn) / np.sqrt((tp + fp) * (tp + fn) * (tn + fp) * (tn + fn))
    mcc = 0
    return precision, recall, f1_score, mcc
```

TP = 1, FP = 2, FN = 1 TN = 12

Accuracy = $(TN+TP) / ALL = (12+1)/16 = 0.75$

Recall = $TP / (TP+FN) = 1/2 = 0.5$

Precision = $TP / (TP + FP) = 1/3 = 0.33$

Firstly prediction -> 0 tp -> 0

Precision >> Recall in the first epochs

Label

	A	C	G	A
A	0	0	0	0
C	0	0	1	0
G	0	1	0	0
A	0	0	0	0

Prediction

	A	C	G	A
A	0	0	1	0
C	0	0	0	0
G	0	1	0	0
A	0	0	0	1

```
a = constraint_matrix_batch(one_hot1)
out = torch.mul(out, a)
test_out = (out > 0.5).float()
```

```
Evaluation Results:fil: 0.30, Loss: 0.0089,Sensitivity:0.65, Precision:0.60, F1 Score:0.60,
Evaluation Results:fil: 0.40, Loss: 0.0089,Sensitivity:0.63, Precision:0.64, F1 Score:0.60,
Evaluation Results:fil: 0.50, Loss: 0.0089,Sensitivity:0.60, Precision:0.67, F1 Score:0.60,
Evaluation Results:fil: 0.60, Loss: 0.0089,Sensitivity:0.56, Precision:0.71, F1 Score:0.59,
Evaluation Results:fil: 0.70, Loss: 0.0089,Sensitivity:0.52, Precision:0.75, F1 Score:0.56,
Evaluation Results:fil: 0.80, Loss: 0.0089,Sensitivity:0.45, Precision:0.80, F1 Score:0.51,
Evaluation Results:fil: 0.90, Loss: 0.0089,Sensitivity:0.30, Precision:0.87, F1 Score:0.37,
```

Details

Important hyperparameters and settings

1. Blstm is expensive for 4080, use it instead of the Nfc layers
2. Nfc should be less than 2 for its expensive cost.
3. Lr = 0.0001 is the best lr for this model.
4. BCELoss is good enough, CrossEntropy also works well. You should not use BCELogitLoss because of the sigmoid function.
5. Channels of the resnet are also flexible though it take some effort to customize the code
6. Threshold for out may increase f1 score to some extent

```
Db1 = 64
num_res_blocks = 20
num_epochs = 100
Nfc = 1
bilstm = 0

model = EnsembleModel(num_res_blocks)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)
```

Supplementary Table 9. SPOT-RNA ensemble architectures.

	Num. blocks/layers			Depth of Layer			Dilation Factor
	N_A	N_{BL}	N_B	D_{RES}	D_{BL}	D_{FC}	
Model 0	16	-	2	48	-	512	-
Model 1	20	-	1	64	-	512	-
Model 2	30	-	1	64	-	512	-
Model 3	30	1	-	64	200	-	-
Model 4	30	-	1	64	-	512	$2^{i\%5}$

Details

Evaluation and transfer learning

transfer learning

```
def train_model(model, num_res_blocks, criterion, optimizer, train_loader, test_loader, num_epochs=50):
    device = torch.device("cuda:1" if torch.cuda.is_available() else "cpu")
    model.to(device)
    model.load_state_dict(torch.load(f'models/model{MODEL}.pt'))
    # lr decay, the learning rate will be decayed by 0.9 every 5 epochs
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, 5, gamma=0.9, last_epoch=-1)

    for epoch in range(num_epochs):
        print("-----第{}轮训练开始-----".format(epoch + 1))
        # switch to train mode
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0
        eval_final = []
        tik = time.time()

        for i, (sample, target) in enumerate(train_loader):
            # data preprocessing for the input data and target data
            seq, mtx, one_hot1 = create_files_bpRNA(sample[0])
            mtx = np.array(mtx).reshape((seq, seq, 8))
            mtx = torch.from_numpy(mtx).to(torch.float).to(device)
            one_hot1 = torch.from_numpy(one_hot1).float().to(device)
            target = torch.Tensor(target).to(device)
```

Evaluation

```
def evaluate_model(model, criterion, data_loader, epoch):
    # switch to evaluate mode
    device = torch.device("cuda:1" if torch.cuda.is_available() else "cpu")
    model.to(device)

    model.eval()
    running_loss = 0.0
    correct = 0
    all_predictions = []
    all_targets = []
    eval_final = []
    # evaluate the model
    with torch.no_grad(): # no need to calculate the gradients and no dropout will be used
        for i, (sample, target) in enumerate(data_loader):
            outputs = []
            seq, mtx, one_hot1 = create_files_bpRNA(sample[0])
            mtx = np.array(mtx).reshape((seq, seq, 8))
            mtx = torch.from_numpy(mtx).to(torch.float).to(device)
            one_hot1 = torch.from_numpy(one_hot1).float().to(device)
            target = torch.Tensor(target).to(device)

            for MODEL in range(NUM_MODELS):
                model.load_state_dict(torch.load(f'models/model{MODEL}.pt'))
                out = model(mtx).squeeze(3)
                outputs[MODEL] = out.view(-1)

            input_flat = torch.mean(torch.stack(outputs), dim=0)

            #input_flat = out.view(-1)
            target_flat = target.view(-1)
```

Result

Model1 (models/0318_32_4.pt)

Training and validation

```
Epoch [33/150], lr: [5.314410000000002e-05], Loss: 0.0075,time:6766.838826417923  
Evaluation Results: Loss:0.0075, Sensitivity:0.56, Precision:0.76, F1 Score:0.60, MCC:0.00  
Evaluation Results:Loss: 0.0090,Sensitivity:0.59, Precision:0.68, F1 Score:0.59, MCC:0.00
```

Test

```
Evaluation Results:Loss: 0.0089,Sensitivity:0.60, Precision:0.67, F1 Score:0.60, MCC:0.61
```

Transfer learning(model/0321_66_qhr2_PDB.pt)

Transfer learning test with canonical constraint

```
Evaluation Results:Loss: 0.0428,Sensitivity:0.54, Precision:0.84, F1 Score:0.64, MCC:0.65
```

Transfer learning test without canonical constraint

```
Evaluation Results:Loss: 0.0428,Sensitivity:0.59, Precision:0.81, F1 Score:0.65, MCC:0.67
```

Supplementary Table 1. Performance of individual models and the ensemble on TS0 after initial training on TR0.

predictor	MCC ^a	F1 ^b	Precision	Sensitivity	Accuracy
Model 0	0.617	0.618	0.642	0.596	0.998
Model 1	0.612	0.612	0.657	0.573	0.998
Model 2	0.593	0.591	0.654	0.537	0.998
Model 3	0.569	0.568	0.618	0.526	0.998
Model 4	0.611	0.605	0.705	0.531	0.995
Ensemble	0.629	0.626	0.709	0.550	0.999

^a Matthews correlation coefficient. ^b Harmonic mean of precision and sensitivity.

Supplementary Table 3. Performance on the independent test set TS1 after transfer learning by individual models and their ensemble during 5-fold cross-validation.

predictor	MCC ^a	F1 ^b	Precision	Sensitivity	Accuracy
Model 0	0.662 (0.01 ^c)	0.647 (0.01 ^c)	0.831 (0.02 ^c)	0.530 (0.02 ^c)	0.994 (0.00 ^c)
Model 1	0.668 (0.01 ^c)	0.655 (0.01 ^c)	0.832 (0.03 ^c)	0.541 (0.01 ^c)	0.994 (0.00 ^c)
Model 2	0.666 (0.01 ^c)	0.656 (0.01 ^c)	0.810 (0.04 ^c)	0.554 (0.02 ^c)	0.994 (0.00 ^c)
Model 3	0.655 (0.00 ^c)	0.641 (0.00 ^c)	0.833 (0.02 ^c)	0.520 (0.01 ^c)	0.994 (0.00 ^c)
Model 4	0.647 (0.00 ^c)	0.636 (0.00 ^c)	0.809 (0.04 ^c)	0.531 (0.02 ^c)	0.994 (0.00 ^c)
Ensemble	0.690 (0.02 ^c)	0.687 (0.01 ^c)	0.888 (0.02 ^c)	0.562 (0.02 ^c)	0.995 (0.00 ^c)

^a Matthews correlation coefficient. ^b Harmonic mean of precision and sensitivity. ^c Standard deviation based on five fold cross validation.



Pipeline

dl_example

> datasets

> models

bpRNA_test.py

bpRNA_training.py

data_preprocess.py

PDB_tl_training.py

↓ readme.markdown

test_res32.py

```
# This folder is used to reproduce the SPOTRNA in Pytorch instead of tensorflow
```

Datasets

All the datasets are in dl_example/datasets

The bpRNA_datasets is the large datasets for pre-training and PDB_datasets are for transfer learning

The raw data are collected in the sub folders, TR is for training, VL for validation and TS for test

There are also some processed data like train_x,train_y are for training, test_x,test_y are for validation, ts_x,ts_y are for test, respectively. They can be simply loaded into the model to simplify the pipeline.

Models

The outstanding model has been collected in dl_example/models as example.

0318_32_4.pt is the pre-trained model with f1 score 0.6 and mcc 0.61 in TS0

0321_66_qhr2_PDB.pt is the transfer learning model with f1 score 0.65 and mcc 0.67 in TS1

Codes

The bpRNA_training.py is for training model on bpRNA datasets.

The major hyperparameters are in the bottom of the code to be modified. Details are mentioned in slides.

Usually, the model take 6000s to train a epoch and the model may converge at 30 epoch. Then it may need 50h to converge.

The test_res32.py is an example for model testing.

The code structure is similar to the previous file. The major difference is loading the trained model and shutdown the gradient as code do. Also, the hyperparameters should match the model.

Then it may take 10-20mins to print the result for model test.

The bpRNA_test.py is the formal testing code.

When you trained more than one model, you can use this file to test the model as an ensemble.

The PDB_tl_training.py is for transfer learning on PDB datasets

The pretrained model can be loaded and re-trained on PDB datasets. Remember to set a different save path.

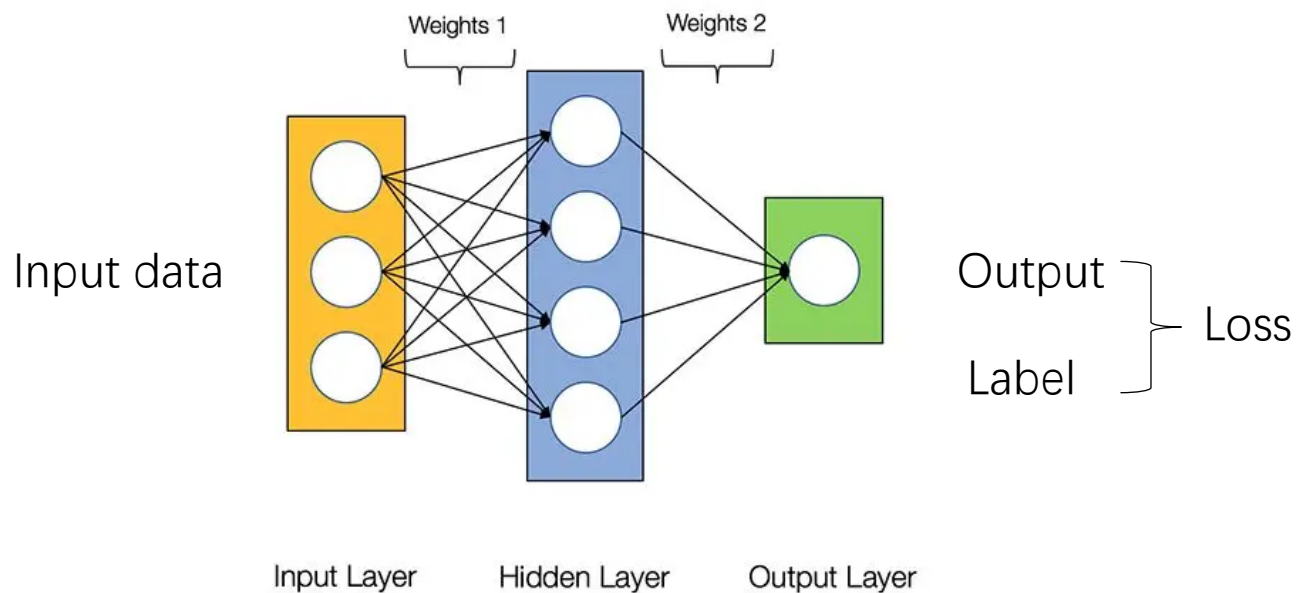
It may take 5-10mins to train a outstanding model.

<https://github.com/jaswindersingh2/SPOT-RNA/blob/master/utils/utils.py>

Thank you for listening



Introduction



Task: Regression, Classification . etc

Linear

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$$

$$L(\mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^n l^{(i)}(\mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^n \frac{1}{2} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)} \right)^2.$$

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}.$$

Complicated

gradient descent

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_{\mathbf{w}} l^{(i)}(\mathbf{w}, b) = \mathbf{w} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathbf{x}^{(i)} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)} \right), \\ b &\leftarrow b - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_b l^{(i)}(\mathbf{w}, b) = b - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)} \right). \end{aligned}$$

Introduction

Parameters and hyperparameters

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_{\mathbf{w}} l^{(i)}(\mathbf{w}, b) = \mathbf{w} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathbf{x}^{(i)} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)} \right), \\ b &\leftarrow b - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_b l^{(i)}(\mathbf{w}, b) = b - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)} \right).\end{aligned}$$

Parameters: w, b ,

Hyperparameters: η ,

updates when training
assign before training

Learning rate(lr)

