CS598: Deep Learning For Healthcare

Research Paper Reproduction Project

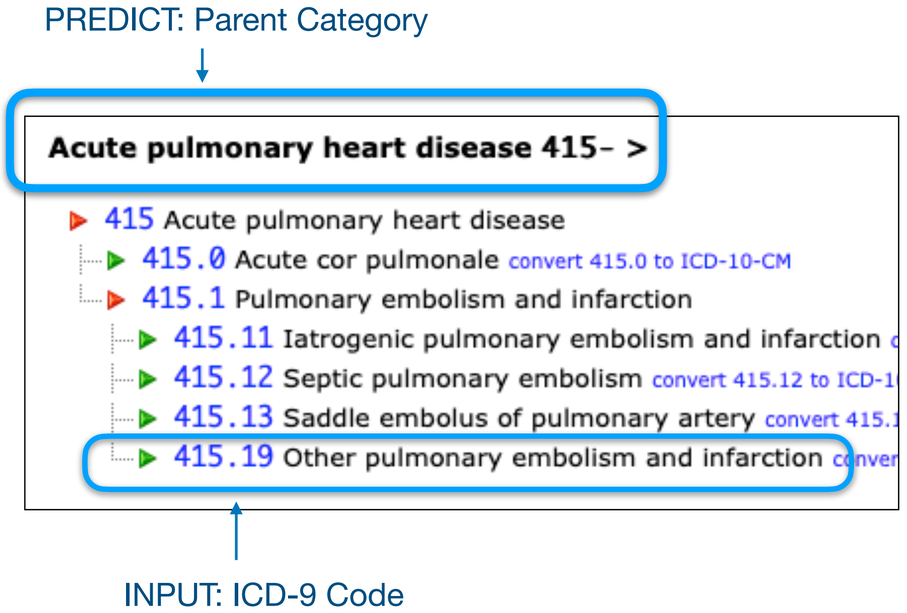
Incorporating medical code descriptions for diagnosis prediction in healthcare

Eric Ahlgren, May 6 2022

Motivation

Improve diagnosis prediction using text descriptions

- ICD-9 codes have associated text descriptions
- Learned sentence embeddings can be used to represent diagnosis code descriptions
- Can leverage similarities in text to improve prediction performance
- Example: “pulmonary embolism” is a common phrase for “Acute pulmonary heart disease”
- Take codes as input and predict parent category



Adaptable "*Enhanced*" Framework

Proposed method can easily be used to upgrade existing models

- Any model which takes diagnosis codes as input can be modified to use this framework
- Requires obtaining sentence-level embedding matrix **E** from diagnosis code descriptions
- Each visit is then represented by the equation:

$$\mathbf{v}_t = \tanh(\mathbf{E}\mathbf{x}_t + \mathbf{b}_v),$$

Two steps:

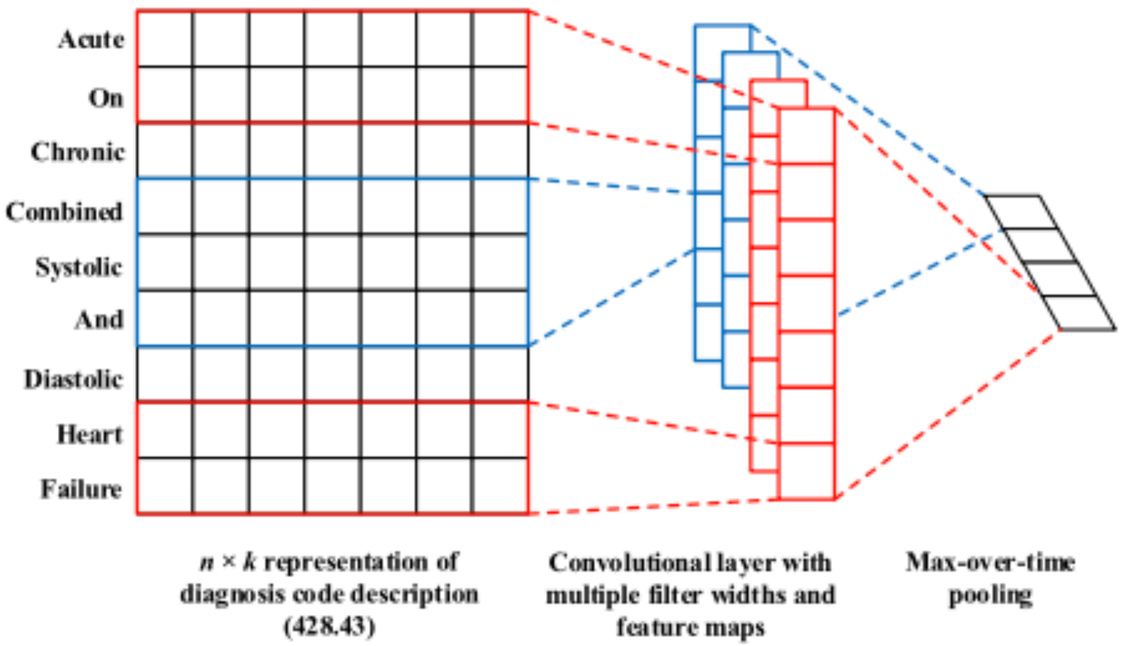
1) Replace the embedding layer of current model with a linear layer with E as weights

2) Pass input data *x* as a multi-hot vector through above layer and execute *tanh* on the result

Obtaining description embedding matrix

Using Convolutional Neural Network (TextCNN) to get E

- For each word window, *m* filters are used to create feature maps
- Max pooling of feature maps extracts most relevant data to embedding vector
- Word window sizes and number of filters determine dimensionality of embedding
- Embeddings are concatenated into **E** such that each column represents a diagnosis code description



Model Evaluation

Define the evaluation metrics used for performance analysis

- Visit-level precision@k
- Code-level accuracy@k
- Obtained by executing Softmax on the logits and checking the top *k* results
- Paper used values *k*=[5,10,15,20,25,30]
- precision@k is more coarse while accuracy@k provides more fine-grained view

Visit-level precision@k

Correct diagnosis codes in top *k* divided by min(*k*,$|y_t|$), where $|y_t|$ is the number of category labels in the (*t*+1th) visit.

Code-level accuracy@k

Number of correct label predictions divided by the total number of label predictions.

Comparison with Baseline

Paper examines 6 baseline models vs 6 *enhanced* models

- Simple MLP (1 FC layer)
- Simple RNN (1 GRU layer)
- RNN+attention (1 GRU layer, 1 AlphaAttention layer)
- Dipole (Bi-directional RNN+attention, 2 GRU layers, 1 attention layer)
- RETAIN (2 GRU layers, 2 attention layers)
- GRAM (1 GRU, 1 graph-based attention layer)

Table 1 Base models for diagnosis prediction

Base model	Visit modeling		Attention mechanism	
	FC	GRU	Location	Graph
MLP	✓			
RNN [2–4]		✓		
RNN_α [2]		✓	✓	
Dipole [2]		✓	✓	
RETAIN [4]		✓	✓	
GRAM [3]		✓		✓

Data Processing

Comparison of original vs reproduced MIMIC-III data processing

- Paper specified to filter out patients with less than 2 ICU visits (but nothing else)
- My results extracted from ICUSTAYS.csv have slightly more records
- Used a web-scraper to extract parent category labels from http://www.icd9data.com
- Performed a series of merge operations to unify data, organize results
- Performed tokenization on descriptive text

Statistic	Reproduced Values	Original Values
# of patients	8,748	7,499
# of visits	23,796	19,911
Avg. visits per patient	2.72	2.66
# of unique ICD9 codes	4,903	4,880
Avg. # of diagnosis codes per visit	13.85	13.06
Max # of diagnosis codes per visit	39	39
# of words in code descriptions	3,164	2,800
# of category codes	184	171
Avg. # of category codes per visit	10.69	10.16
Max # of category codes per visit	30	30

Table 1: Summary of descriptive data statistics.

Baseline Model Implementation

Overview of implementation methods used

- Author provided no code for baseline/ enhanced models
- Main challenge was determining optimal strategy for organization/structure of input/ target data
- Paper includes high-level equations, but very little about specifics of implementation or data structure
- With experimentation and iteration arrived at optimal result: use target vector of probabilities and predict last sequence for RNN

MLP

```
n = 0
for i,patient in enumerate(sequences):
    for j,visit in enumerate(patient):
        if j == len(patient) - 1:
            break
        for k,code in enumerate(visit):
            x[n,k] = code
            x_masks[n,k] = 1
        n+=1
n = 0
for i,patient in enumerate(targets):
    for j,visit in enumerate(patient):
        if j == len(patient) - 1:
            break
        y[n] = torch.tensor(patient[j+1])
        n += 1
```

RNN-based

```
for i_patient, patient in enumerate(sequences):
    for j_visit, visit in enumerate(patient[:-1]):
        for k_code, code in enumerate(visit):
            x[i_patient, j_visit, k_code] = code
            x_masks[i_patient, j_visit, k_code] = 1

for i_patient, patient in enumerate(targets):
    last_visit = patient[-1]
    y[i_patient] = torch.FloatTensor(last_visit)
    y_masks[i_patient] = torch.BoolTensor(np.ones(max_num_categories))
```

Baseline Model Implementation

MLP

```
ipdb> x[:3]
tensor([[ 841, 1622, 1588, 1596, 1643, 4783,  608,  715, 1500, 1544, 2381, 2388,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
        [1632, 1640,   16, 2211,  821, 2019, 4712,  608,  755,  749,  750, 1499,
         2849, 1239, 2381, 2230,  908, 1851,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
        [1632, 1638, 3208, 2439,  750, 1779, 1509, 1533, 2381,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0]])
```

x is a 2D padded tensor
of code indices

```
ipdb> y[0]
tensor([0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.1111, 0.0000, 0.1111, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.1111, 0.1111, 0.0000, 0.3333, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.1111, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.1111, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
```

y is a 2D multi-hot tensor of categories
with probabilities instead of {0,1}

Input visit and target are 1:1

Baseline Model Implementation

RNN-based

```
ipdb> x[65]
tensor([[  10, 2281, 1373, 3335, 4720, 4710, 2444,  610, 4211,  750, 1500,  392,
         3363, 4336, 4860, 2381,   65, 4128,  834, 4658, 4651, 4572, 4660,  899,
          911, 2450, 1122, 4898, 3913, 4402,    0,    0,    0,    0,    0,    0,
            0,    0,    0],
        [4723,  608, 1500,  373, 2449,  247,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0],
        [3238, 4746,  608, 2086, 1500, 1068, 4572, 4656, 2449, 1125, 3913, 4403,
          578,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0],
        [  10, 2281, 1373, 3335, 4720, 4710, 2444,  610, 4211,  750, 1500,  392,
         3363, 4336, 4860, 2381,   65, 4128,  834, 4658, 4651, 4572, 4660,  899,
          911, 2450, 1122, 4898, 3913, 4402,    0,    0,    0,    0,    0,    0,
            0,    0,    0],
```

x is a 3D padded tensor of visit sequence and code indices per patient

Input is seq of visits excluding final visit and target is final visit

```
ipdb> y[65]
tensor([0.0333, 0.0000, 0.0000, 0.0333, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0333, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0333, 0.0000, 0.0333, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0333, 0.0000, 0.0000, 0.0000, 0.0667, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0333, 0.0000, 0.0000, 0.0000, 0.0000, 0.0333, 0.0000, 0.0000,
        0.0000, 0.0333, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0333, 0.0333, 0.0333, 0.0333, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0333,
        0.0333, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0333, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0333, 0.0333, 0.0000,
        0.1333, 0.0000, 0.0000, 0.0667, 0.0000, 0.0333, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0333, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
```

y is a 2D multi-hot tensor of categories with probabilities instead of {0,1}

TextCNN Model Implementation

Overview of implementation methods used

- Author provided reference code on GitHub and link to fastText pre-trained word embeddings
- Used word embedding model to build vocab tensor of word embeddings with lookup mapping
- Collate into sequences of word embeddings for each code description, with target of diagnosis code index
- Used model parameters described in paper, but experimented with training iterations

Collate function

```
text, indices = zip(*data)
word_embed_dim = 300

y = torch.tensor(indices, dtype=torch.long)
num_codes = len(text)
num_words = [len(words.split()) for words in text]

max_num_words = max(max(num_words), 4)

global lookup
x = torch.zeros((num_codes, max_num_words), dtype=torch.long)
masks = torch.zeros((num_codes, max_num_words), dtype=torch.bool)
for i, code in enumerate(text):
    for j, word in enumerate(code.split()):
        x[i,j] = lookup[word]
        masks[i,j] = 1

return x, y, masks
```

Model structure

```
|: EmbeddingCNN(
    (embed): Embedding(3164, 300)
    (convs): ModuleList(
      (0): Conv2d(1, 100, kernel_size=(2, 300), stride=(1, 1))
      (1): Conv2d(1, 100, kernel_size=(3, 300), stride=(1, 1))
      (2): Conv2d(1, 100, kernel_size=(4, 300), stride=(1, 1))
    )
    (dropout): Dropout(p=0.5, inplace=False)
    (fc): Linear(in_features=300, out_features=4903, bias=True)
  )
```

TextCNN Model Implementation

Overview of implementation methods used

- Modify forward function to return diagnosis embeddings after training is complete
- Embedding matrix **E** is the result of concatenating the max pooling of the convolutional filters

forward function

```
def forward(self, x, masks):
    """
    Both the logit for training and the embedding matrix are
    can be obtained once training is complete.

    Arguments:
        x: the input tensor of icd9 description of size (batc
        masks: masks for the padded words of size (batch_size

    Outputs:
        logit: logits for cross entropy loss function to for
        embedding: embedding matrix of learned wieghts for ic
    """
    x = self.embed(x)
    x = x.unsqueeze(1)
    x = [F.relu(conv(x)).squeeze(3) for conv in self.convs]
    x = mask_conv2d(x, masks)
    x = [F.max_pool1d(i, i.size(2)).squeeze(2) for i in x]
    embedding = torch.cat(x, 1)
    x = self.dropout(embedding)
    logit = self.fc(x)
    return logit, embedding
```

Enhanced Model Implementation

Overview of implementation methods used

- Embedding is a linear layer with embedding matrix **E** as weights
- Convert input tensor *x* of diagnosis description indices to multihot tensor and pass through embedding layer
- Execute *tanh* to get matrix of vectors $\mathbf{v}_t$, then pass through the baseline model

```
class EnhancedMLP(nn.Module):

    def __init__(self, num_codes, num_categories, embedding_matrix):
        super().__init__()
        """
        Arguments:
            num_codes: total number of diagnosis codes
            num_categories: number of higher level categories to predi
            embedding_matrix: learned embedding matrix of icd9 descrip
        """
        self.embedding = nn.Linear(4903, 300)
        self.embedding.weight.data = embedding_matrix
        self.fc = nn.Linear(300, num_categories)

    def forward(self, x, masks):
        """
        Arguments:
            x: the diagnosis sequence of shape (batch_size, # visits,
            masks: the padding masks of shape (batch_size, # visits, #
        Outputs:
            logits: logits of shape (batch_size, # diagnosis codes)
        """
        x = indices_to_multihot(x, masks, 4903)
        x = self.embedding(x)
        x = torch.tanh(x)
        logits = self.fc(x)
        return logits
```

Results: Paper vs Reproduction

Comparison of results: precision@k

Table 3 The visit-level precision@*k* of diagnosis prediction task

Dataset	@*k*	MLP	MLP+	RNN	RNN+	RNN_a	RNN_a+	Dipole	Dipole+	RETAIN	RETAIN+
MIMIC-III	5	0.6939	0.7124	0.6616	0.7160	0.6504	0.7083	0.6599	0.7074	0.6835	0.7167*
	10	0.6441	0.6603	0.6145	0.6565	0.6021	0.6527	0.6116	0.6539	0.6361	0.6623*
	15	0.6812	0.6926*	0.6546	0.6906	0.6412	0.6856	0.6524	0.6903	0.6777	0.6918
	20	0.7420	0.7544*	0.7199	0.7511	0.7109	0.7455	0.7159	0.7483	0.7403	0.7501
	25	0.7939	0.8070*	0.7755	0.8019	0.7697	0.8009	0.7723	0.8020	0.7912	0.8010
	30	0.8357	0.8460	0.8186	0.8456	0.8142	0.8445	0.8169	0.8453	0.8335	0.8445

@k	MLP	MLP+	RNN	RNN+	RNNa	RNNa+	Dipole	Dipole+	RETAIN	RETAIN+
5	0.6879	0.6934	0.7156	0.7111	0.7390	0.7115	0.7237	0.7217	0.7135	0.6896
10	0.6526	0.6505	0.6787	0.6585	0.6931	0.6742	0.6858	0.6820	0.6651	0.6472
15	0.6849	0.6789	0.7060	0.6932	0.7237	0.7052	0.7164	0.7164	0.6952	0.6867
20	0.7417	0.7332	0.7606	0.7484	0.7733	0.7601	0.7716	0.7709	0.7476	0.7440
25	0.7922	0.7856	0.8063	0.7959	0.8183	0.8096	0.8206	0.8154	0.7977	0.7936
30	0.8375	0.8299	0.8454	0.8348	0.8554	0.8451	0.8533	0.8503	0.8366	0.8325

Results: Paper vs Reproduction

Comparison of results: accuracy@k

Table 4 The code-level accuracy@*k* of diagnosis prediction task

Dataset	@*k*	MLP	MLP+	RNN	RNN+	RNN_a	RNN_a+	Dipole	Dipole+	RETAIN	RETAIN+
MIMIC-III	5	0.3104	0.3181	0.2952	0.3193	0.2910	0.3162	0.2941	0.3155	0.3056	0.3198*
	10	0.5040	0.5138	0.4796	0.5111	0.4693	0.5085	0.4767	0.5086	0.4980	0.5160*
	15	0.6286	0.6352	0.6019	0.6335	0.5889	0.6290	0.5971	0.6325	0.6258	0.6360*
	20	0.7114	0.7239*	0.6894	0.7198	0.6822	0.7144	0.6845	0.7168	0.7129	0.7202
	25	0.7754	0.7852*	0.7545	0.7804	0.7491	0.7785	0.7501	0.7795	0.7735	0.7806
	30	0.8214	0.8294*	0.8040	0.8279	0.7987	0.8269	0.7990	0.8280	0.8198	0.8286

@k	MLP	MLP+	RNN	RNN+	RNNa	RNNa+	Dipole	Dipole+	RETAIN	RETAIN+
5	0.3490	0.3608	0.3851	0.3860	0.3969	0.3856	0.3954	0.3937	0.3766	0.3794
10	0.5466	0.5467	0.5821	0.5675	0.5886	0.5775	0.5859	0.5812	0.5660	0.5581
15	0.6604	0.6548	0.6850	0.6732	0.7001	0.6845	0.6938	0.6899	0.6730	0.6666
20	0.7379	0.7293	0.7572	0.7455	0.7696	0.7572	0.7685	0.7651	0.7439	0.7409
25	0.7920	0.7852	0.8061	0.7958	0.8182	0.8095	0.8204	0.8148	0.7973	0.7934
30	0.8375	0.8299	0.8454	0.8348	0.8554	0.8451	0.8533	0.8503	0.8366	0.8325

Results: Paper vs Reproduction

Comparison of results: % change in precision@k

@k	MLP	MLP+	RNN	RNN+	RNNa	RNNa+	Dipole	Dipole+	RETAIN	RETAIN+
5	-0.9	-2.7	+8.2	-0.7	+13.6	+0.5	+9.7	+2	+4.4	-3.8
10	+1.3	-1.5	+10.4	+0.3	+15.1	+3.3	+12.1	+4.3	+4.6	-2.3
15	+0.5	-2	+7.9	+0.4	+12.9	+2.9	+9.8	+3.8	+2.6	-0.7
20	+0	-2.8	+5.7	-0.4	+8.8	+2	+7.8	+3	+1	-0.8
25	-0.2	-2.7	+4	-0.7	+6.3	+1.1	+6.3	+1.7	+0.8	-0.9
30	+0.2	-1.9	+3.3	-1.3	+5.1	+0.1	+4.5	+0.6	+0.4	-1.4

Table 3: % difference precision@k: my implementation vs paper results

- Some of my baseline models performed better than paper baseline models
- Could not determine why without specific implementation details