Chapter 11
Memory Networks

Memory network is a powerful extension of attention models. The memory network models have shown initial successes in natural language processing such as question answering [60, 92, 112, 141, 166]. In particular, memory networks use external memory components to assist the deep neural networks in remembering and storing information. Various memory network based models have been proposed, such as [92, 112, 141]. In healthcare, memory networks can be valuable due to their capacities in utilizing medical knowledge and patient history, as shown in [134].

11.1 Original Memory Networks

Memory Networks [166] and Differentiable Neural Computers (DNC) [60] proposed to use external memory components to assist the deep neural networks in remembering and storing information. After that, various memory network models [92, 112, 141] have been proposed. The original memory networks are proposed for solving question answering in the natural language processing domain [92, 112, 141, 166].

At a high level, a memory network consists of memory $\boldsymbol{m}$ and the following four learning components.

1. **Input feature map** I converts the incoming input to the internal feature representation.
2. **Generalization** component G updates old memories given new input. We call this generalization as there is an opportunity for the network to compress and generalize its memories at this stage for some intended future use.
3. **Output feature map** O produces a new output in the feature representation space, given the new input and the current memory state.
4. **Response** component R converts the output embeddings into the response format desired. For example, a text description or a particular class label.

C. Xiao, J. Sun, *Introduction to Deep Learning for Healthcare*,
https://doi.org/10.1007/978-3-030-82184-5_11

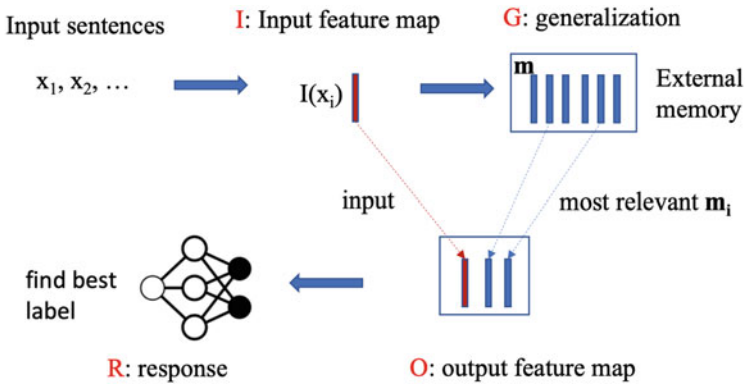

Fig. 11.1 Components of memory networks

Next, we illustrate the interaction between those components. Given an input $\boldsymbol{x}$ such as a sequence of words or sequence of clinical events, the flow of the memory network is given as follows.

1. First, we will convert $\boldsymbol{x}$ to an internal feature representation $I(\boldsymbol{x})$. Here the representation of inputs and memories can use all kinds of encodings such as a bag of words or multi-hot embeddings, RNN embedding.
2. Second, we store $I(\boldsymbol{x})$ into the memory, or update existing memory $\boldsymbol{m}_i$ with generalization and compression. Mathematically, it denotes as $\boldsymbol{m}_i = G(\boldsymbol{m}_i, I(\boldsymbol{x}), \boldsymbol{m}) \forall i$. In the simplest form, G function just stores $I(\boldsymbol{x})$ into a memory slot $\boldsymbol{m}_i$ in memory bank $\boldsymbol{m}$.
3. Third, we compute the output feature o given the new input $I(\boldsymbol{x})$ and the current memory bank $\boldsymbol{m}$. This component tries to find the best matching memory slots based on some matching function $\boldsymbol{m}$: $o = O(I(\boldsymbol{x}), \boldsymbol{m})$. For example, Fig. 11.1 retrieves two most relevant memory slots to combine with input $I(\boldsymbol{x})$ as the output feature map.
4. Finally we map output o to give final response $r = R(o)$. We can choose any output decoders—for example, softmax for classification or RNN for sequence generation.

We need to specify the inference mechanism in O and R modules to implement the memory network with neural networks. One simple way is to use the input as a query to score all memory slots and retrieve the ones with highest scores: $o_1 = \arg\max_i S_O(x, \boldsymbol{m}_i)$ where S_O is the output scoring function that computes the similarity between input x and memory slot $\boldsymbol{m}_i$. This can be generalized to more than one memory slot. The response module will score the input and most relevant memory slots to generate the response. For example, if the response is a single word from a set W, the response will be $r = \arg\max_{w \in W} S_R([x, o_1], w)$ where S_R is the response scoring function between a word w and the combination of input x and most relevant memory o_1. In [166], the similarity function is

$S(x, y) = \Phi_x(x)^\top \boldsymbol{U}^\top \boldsymbol{U} \Phi_y(y)$ where $\Phi_x(x)$,$\Phi_y(y)$ are the input embeddings and $\boldsymbol{U}$ is the linear embedding matrix. Finally, the model is trained in a fully supervised fashion using the ranking loss of the output module and response module:

$$\underbrace{\sum_{\bar{f} \neq o_1} \max(0, \gamma - S_O(x, o_1) + s_O(x, \bar{f}))}_{\text{output module}} + \underbrace{\sum_{\bar{r} \neq r} \max(0, \gamma - S_R([x, o_1], r) + s_R([x, o_1], \bar{r}))}_{\text{response module}}$$

where $\bar{f}$ and $\bar{r}$ are the other choices than the correct label, and γ is the margin. In the paper [166], the top two output o_1 and o_2 are generated, which leads to a slightly more complicated expression. One major limitation of the original memory network is the lack of end-to-end training due to the arg max operations which break the gradient flows which is needed for backpropagation.

11.2 End-to-End Memory Networks

The original memory network [166] uses the arg max function to find the optimal memory slots and the best output. Later on, the authors [141] extended this "hard" attention with "soft" attention via softmax function, which leads to memory networks that can be trained in an end-to-end fashion.

This end-to-end memory network involves input $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ stored in the memory bank and a query $\boldsymbol{q}$. And this network outputs an answer $\boldsymbol{a}$. For example, input $\boldsymbol{x}_i$ can be all the historical patient records in the EHR database. In contrast, query $\boldsymbol{q}$ is the record of the current patient, and the answer $\boldsymbol{a}$ can be the medication recommended for the current patient. Two different memory representations are constructed for the input: one input memory representation called the *keys*, and one output memory representation called the *values*. Figure 11.2 illustrates the architecture of the end-to-end memory network.

Input Memory (Keys) All the input $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ are embedded via matrix $\boldsymbol{A}$ into a set of vectors $\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots, \boldsymbol{m}_n$. For example, a simple linear embedding can be applied $\boldsymbol{m}_i = \boldsymbol{A}\boldsymbol{x}_i$. The query $\boldsymbol{q}$ is also embedded using a different matrix $\boldsymbol{B}$ to produce the embedded vector $\boldsymbol{u}$. For example, $\boldsymbol{u} = \boldsymbol{B}\boldsymbol{q}$. Then the similarity matching score between query embedding $\boldsymbol{u}$ and input embedding $\boldsymbol{m}_i$ can be computed as inner product followed by softmax:

$$p_i = \text{Softmax}(\boldsymbol{u}^\top \boldsymbol{m}_i).$$

Here p_i is the matching score between the query $\boldsymbol{u}$ and input $\boldsymbol{m}_i$. Once we calculated the matching scores from the query to all input embeddings, denoted by $p_1, p_2, \ldots, p_n$, we can retrieve the output embeddings with highest matching scores.

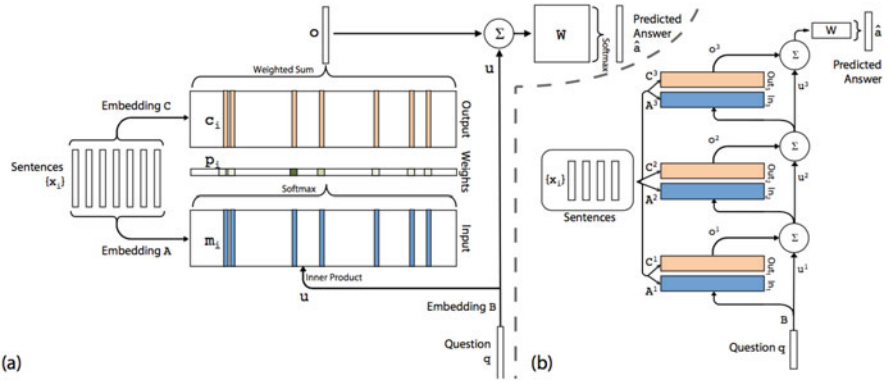

Fig. 11.2 End-to-end memory networks: (**a**) a single layer version, (**b**) multi-layer version

Output Memory (Value) Similar to input memory, the output memory are another set embedding vectors transformed from input $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ via embedding matrix $\boldsymbol{C}$ via $\boldsymbol{c}_i = \boldsymbol{C}\boldsymbol{x}_i$. The response vector is the weighted sum of all output embedding $\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_n$, namely

$$\boldsymbol{o} = \sum_i p_i \boldsymbol{c}_i.$$

Generating Final Prediction Both the response vector $\boldsymbol{o}$ and the query vector $\boldsymbol{u}$ are used to generate the final prediction:

$$\hat{\boldsymbol{a}} = \text{Softmax}(\boldsymbol{W}(\boldsymbol{o} + \boldsymbol{u})).$$

With this architecture, all the embedding matrix $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ and $\boldsymbol{W}$ are jointly trained using the cross-entropy loss between ground truth label $\boldsymbol{a}$ (a one-hot vector) and the prediction $\hat{\boldsymbol{a}}$.

Multiple Layers Multiple memory layers can be stacked:

- The input to layer $k+1$ can be the sum between output $\boldsymbol{o}^k$ and the input $\boldsymbol{u}^k$ from layer k, namely, $\boldsymbol{u}^{k+1} = \boldsymbol{o}^k + \boldsymbol{u}^k$;
- Each layer will have its own embedding matrices $\boldsymbol{A}^k$, $\boldsymbol{B}^k$, $\boldsymbol{C}^k$;
- Finally the prediction per layer can be computed as $\hat{\boldsymbol{a}} = \text{Softmax}(\boldsymbol{W}\boldsymbol{u}^{k+1})$.

The authors proposed two ways to connect the parameters across layers.

- **Adjacent:** Output embedding matrix of layer k is the input embedding matrix for layer $k+1$: $\boldsymbol{A}^{k+1} = \boldsymbol{C}^k$.
- **Layer-wise:** Tying input and output embeddings across all layers, i.e., $\boldsymbol{A}^1 = \boldsymbol{A}^2 = \ldots = \boldsymbol{A}^k$ and $\boldsymbol{C}^1 = \boldsymbol{C}^2 = \ldots = \boldsymbol{C}^k$. Then a linear mapping $\boldsymbol{H}$ is added between input layers: $\boldsymbol{u}^{k+1} = \boldsymbol{H}\boldsymbol{u}^k + \boldsymbol{o}^k$.

11.3 Self-Attention and Transformer

Self-attention is a more efficient strategy for computing attention scores among the original input without any dependency on RNN models. Standard attention models heavily rely on RNN as the annotation vectors $\boldsymbol{h}_i$ are the results of the encoder RNN and the hidden states $\boldsymbol{s}_i$ are also outputs of another RNN. Because of the sequential nature of RNN, the model training process leads to significant computational and optimization challenges. More specifically, attention models need all the hidden states $\boldsymbol{h}_1$ to $\boldsymbol{h}_T$ of the entire input sequence to compute the context vector $\boldsymbol{c}_i$. Such a training process is intrinsically sequential within an input example, limiting its ability to optimize training in parallel, which is computationally more efficient.

To address this parallel training challenge, *Transformer* model is proposed, which introduces a new type of attention without RNN [160]. The insight comes from a simple question "why don't we compute context vectors over $\boldsymbol{x}_i$ directly". We actually can, and the way to achieve this is through the *self-attention* mechanism. Conceptually, the self-attention model replaces the RNNs from the encoder and decoder completely with an attention mechanism with the input to itself.

The goal of transformer model is to "transform" an input embedding $\boldsymbol{x}_i$ into a better embedding $\boldsymbol{x}'_i$ that captures the contexts (or correlation) of $\boldsymbol{x}_i$ with all the other input. To achieve that, a query, keys, and values are introduced similar to an end-to-end memory network [142]. Here a query is a specific input $\boldsymbol{x}_i$, while keys and values are all input $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$. There are two steps involved:

- **Similarity search:** The idea is to compute similarity between query $\boldsymbol{x}_i$ and all input represented by $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d_x \times n}$. Instead of using original input $\boldsymbol{X}$, a linear embedding layer is introduced: $\boldsymbol{K} = \boldsymbol{W}^K \boldsymbol{X} \in \mathbb{R}^{d_K \times n}$ where $\boldsymbol{W}^K \in \mathbb{R}^{d_K \times d_x}$. Similarly, we linearly transform the query $\boldsymbol{q}_i = \boldsymbol{W}^Q \boldsymbol{x}_i \in \mathbb{R}^{d_K}$. A scaled dot product is computed as the similarity score vector $\boldsymbol{e}_i = \frac{\boldsymbol{K}^\top \boldsymbol{q}_i}{\sqrt{d_K}}$. The scaled dot product is very similar to a dot product $\boldsymbol{K}^\top \boldsymbol{q}_i$ but with additional $1/\sqrt{d_K}$ factor. The idea of $1/\sqrt{d_K}$ factor is to alleviate the concern that the softmax function of a high dimensional input may have an extremely small gradient. Like the standard attention mechanism, the attention weight is normalized by $\boldsymbol{a}_i = \text{softmax}(\boldsymbol{e}_i) \in \mathbb{R}^n$.
- **Value retrieval:** Based on the similarity score $\boldsymbol{a}_i$, we can retrieve or recombine the input to generate the output embedding $\boldsymbol{x}'_i = \boldsymbol{W}^V \boldsymbol{X} \boldsymbol{a}_i$.

Note that all weight matrices $\boldsymbol{W}^K, \boldsymbol{W}^V, \boldsymbol{W}^Q \in \mathbb{R}^{d_K \times d_x}$ are of the same size. The reason to have separate weight matrices is to avoid dependency and maximize potential parallelism in training. The self-attention mechanism eliminates any sequential dependency; hence the training can be done in parallel more easily.

Besides the self-attention mechanism, the Transformer model introduces a few other innovations, namely, **multi-head attention** and **positional embedding**.

- The idea of multi-head attention is to compute multiple $\boldsymbol{a}_i$ for the same input $\boldsymbol{x}_i$ with different sets of weight matrices, which is analogous to multiple filters in

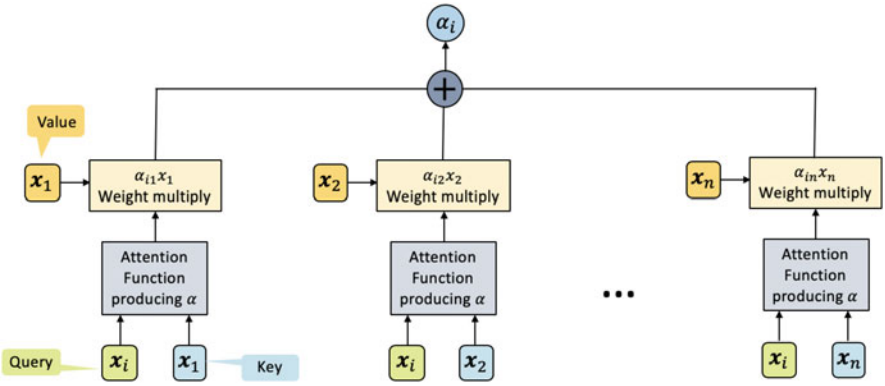

Fig. 11.3 Concept illustration of self-attention mechanism

CNN models. The benefit having multiple attention scores instead of one is to provide more robust estimate of the embedding.

- The positional embedding adds some features related to the position of $\boldsymbol{x}_i$ in the input sequence. This is important for natural language process application as a sequence corresponds to a sentence, and $\boldsymbol{x}_i$ is a word in the sequence. Therefore, where the word $\boldsymbol{x}_i$ appears in the sentence (at the beginning or the end) can be important (Fig. 11.3).

Now we understand the intuition and key components of the Transformer model. Next, we describe the overall algorithm, which follows an encoder and decoder architecture like a sequence-to-sequence model. Figure 11.4 illustrates the Transformer model.

Transformer Encoder The encoder can have multiple multi-head self-attention modules. The input embeddings (with optional positional embedding) are the input for the first layer. For subsequent layers, the output of the previous layer will be its input. The same input set will be used within each layer as keys, queries, and values to compute self-attention. The self-attention output will send through a feed-forward network layer with a skipped connection.

Transformer Decoder The decoder can also have multiple layers. Each layer is similar to the encoder but with different inputs. The input will be the output embedding of the decoder so far. Specifically, the input of the first layer will be the word embeddings generated so far. And subsequent layers will take the output of the previous layer as its input. Within each layer, a **masked multi-head self-attention** is computed with an appropriate mask so the attention will not be computed over the unseen words (i.e., those words to the right of the current word). For example, if the decoder output generates 5 tokens so far while the entire sequence has 20 tokens, the masked attention will not compute attention with the token 6–20 as they are not yet seen. The output is sent to another multi-head attention layer with the input embedding produced by the encoder. Finally, a feed-forward network layer

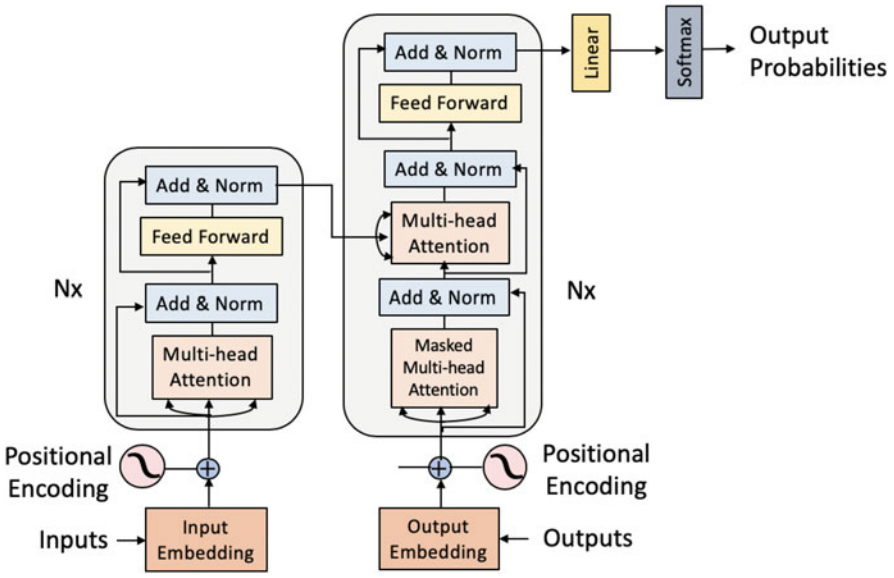

Fig. 11.4 The Transformer model architecture: The left side is the encoder and the right side is the decoder

like in encoder is applied at the end. The prediction task at position i is to predict next token at position $i + 1$ for all positions.

11.4 BERT: Pre-training of Deep Bidirectional Transformers

One important application of the Transformer is the BERT model for supporting natural language processing (NLP) tasks such as language understanding and question answering [37]. BERT stands for Bidirectional Encoder Representations from Transformers. The key ideas behind BERT are (1) context-specific pre-training and (2) masked language model.

Context Specific Pre-training Embedding vectors are the basis for deep learning models. In many tasks, people find it is useful to pre-train embedding from large datasets. For example, Word2Vec [110] and GloVe [117] are often used to generate pre-trained embeddings on large text corpus based on co-occurrence information between words. However, the same word will have the same embedding no matter what the contexts are. For example, although "bank account" and "river bank" have completely different meanings, the word "bank" will have the same embedding. BERT changed this assumption by introducing the context-specific embeddings, where the embedding of word x is dynamically computed based on the context of the sentence where x is in. BERT can be trained in an unsupervised fashion on a

large corpus without any specific target label. If the labeled data exist, BERT also allows fine-tuning using the labeled data.

Masked Language Model Context-specific language models such as RNN are trained either from left to right or right to left. Bi-directional models are achieved by training 2 models from both directions (left to right and right to left) and concatenating the hidden states shown in bi-directional RNN. BERT introduces a new mechanism to achieve a bi-directional language model by randomly masking words. More specifically, by masking $k\%$ of input words (e.g., $k = 15$), the BERT model tries to predict the mask words with a modified Transformer decoder. The details of the masking procedure are more involved:

- 80% of the time we replace the masked word with a special token named [MASK];
- 10% of the time we replace the word with a random word from the entire vocabulary;
- 10% of the time we keep the original word unchanged. The purpose is to guide the BERT model towards this actual word.

The BERT model provides a power embedding method based on pre-training and masked language model, which have shown great performance on many real-world applications.

11.5 Case Study: Doctor2Vec—Doctor Recommendation for Clinical Trial Recruitment

Problem Massive electronic health records (EHRs) enable the success of learning accurate patient representations to support various predictive health applications. In contrast, doctor representation was not well studied, although doctors play pivotal roles in healthcare. How to construct the right doctor representations? How to use doctor representation to solve important health analytic problems? The authors of [8] study the doctor representation problem for the *clinical trial recruitment* application, which is about identifying the right doctors to help conduct the trials based on the trial description and patient EHR data of those doctors.

Method They propose Doctor2Vec, which simultaneously learns (1) doctor representations from EHR data and (2) trial representations from the description and categorical information about the trials. In particular, Doctor2Vec utilizes a *dynamic memory network* where the doctor's experience with patients is stored in the memory bank. The network will dynamically assign weights based on the trial representation via an attention mechanism. Figure 11.5 illustrates the overall framework of Doctor2Vec. Since each doctor sees a diverse set of patients, doctor representation should be dynamically constructed for a given trial instead of a static embedding vector staying the same for all trials. The authors achieved that by

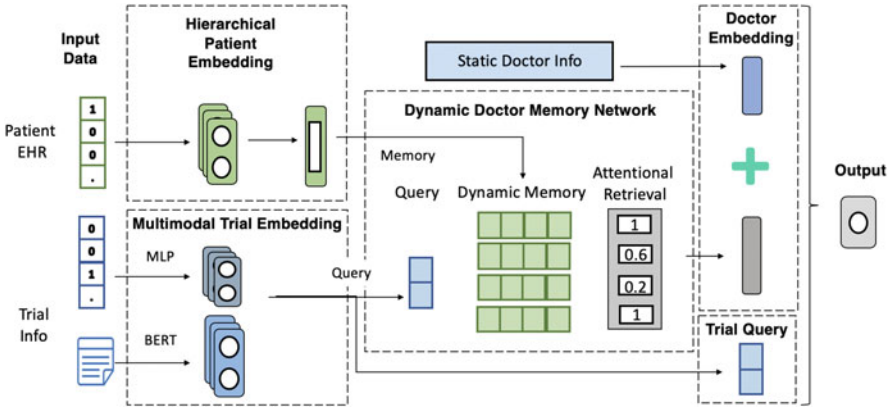

Fig. 11.5 Doctor2Vec Framework. (1) Hierarchical patient embedding: We obtain patient embeddings from patient visits using a Bi-LSTM with attention module. Unstructured text and categorical data are modeled using MLP and BERT, respectively. (2) Multimodal clinical trial information embedding: The obtained clinical trial embeddings from text and categorical data are concatenated together to form the query vector for the memory network. (3) Memory network module: This query vector is used to attend over the memory bank consisting of patient embeddings. The attention vector is used to obtain the final doctor embedding. The doctor embedding $\mathbf{Doc_{emb}}$ is combined with clinical trial embedding $\mathbf{Q_{emb}(l)}$ and static information about the doctors $\mathbf{Doc_{static}}$ to predict the enrollment rate of the clinical trial (output)

a dynamic memory network where patients are stored as memory vectors of the doctor. Using a trial embedding as a query, they fetch the relevant patient vectors from the memory bank and dynamically assemble a doctor representation for this trial.

Inspired by Weston et al. [166], four memory components **I**, **G**, **O**, **R** are proposed which mimics the architecture of modern computer in storing and processing information.

1. **Input Memory Representation**. This layer converts the patient representations to the input representation. They pass all the patient representations through a dense layer to obtain the input representations.

$$\mathbf{I_f(k)} = \mathbf{W_i I(k)} + \mathbf{b_i}$$

   where $\mathbf{I(k)}$ and $\mathbf{I_f(k)}$ are input and final embedding of patient k.
2. **Generalization**. Typically generalization can be referred to as the process of updating memory representation for the memory bank. In our case, they use the patient representations to initialize the memory representation $\mathbf{M_d}$, which is the combination of all the patient representations. They then apply an LSTM layer to update the memory via multiple iterations.

$$\mathbf{M_d} = \text{LSTM}(\mathbf{I_f(1)}, \cdots, \mathbf{I_f(k)}) \tag{11.1}$$

3. **Output**. In this step, the final output memory representation is generated. They calculate the relevance between trial embedding $\mathbf{Q_{emb}(l)}$ and doctor embedding $\mathbf{M_d}$ to obtain $\mathbf{A(k)}$ as the attention vector over patient representations.

$$\mathbf{A(k)} = \text{softmax}[\mathbf{Q_{emb}(l)}^{\mathbf{T}}\mathbf{M_d}] \tag{11.2}$$

4. **Response**. In this step, they obtain the final $\mathbf{Doc_{emb}}$ using the patient embeddings and attention weights over the patients.

$$\mathbf{Doc_{emb}} = \sum \mathbf{A(k)I_f(k)} \tag{11.3}$$

They use the doctor representation, composed of patients, and the clinical trial representation to obtain a final context vector. Besides dynamic doctor embedding $\mathbf{Doc_{emb}}$, they also include static information about doctors in the final embedding, such as their educational history, length of practice, length of practice into the feature vector. The resulting final embedding vectors are then fed into a fully connected layer and passed through a softmax to obtain class labels.

$$\mathbf{Y} = \text{Softmax}([\mathbf{Doc_{emb}}; \mathbf{Q_{emb}(l)}; \mathbf{Doc_{static}}])$$

where the input to Softmax are concatenation of dynamic doctor embedding $\mathbf{Doc_{emb}}$, trial query embedding $\mathbf{Q_{emb}(l)}$ and static doctor embedding $\mathbf{Doc_{static}}$.

The final prediction target is the trial enrollment rate for each doctor. The enrollment rate of a doctor is the number of patients enrolled by a doctor to the trial. And the enrollment rate is normalized by min and max values within each trial. Then the enrollment rate category is obtained by binning the continuous enrollment rate. They divide the continuous enrollment scores into five discrete classes ranging at 0–0.2, 0.2–0.4, 0.4–0.6, 0.6–0.8, 0.8–1.0. The five enrollment categories are used labels for classification.

Result Doctor2Vec is validated on large real-world trials and EHR data, including 2609 trials, 25K doctors and 430K patients.

They consider the following baselines.

1. Median Enrollment (Median). The current industry standard considers each therapeutic area's median enrollment rate as the estimated rate for all trials in that area.
2. Logistic Regression (LR). They combine the medication, diagnosis, procedure codes, and clinical trial information to create feature vectors and then apply LR to predict the enrollment rate category.
3. Random Forest (RF). They combine the medication, diagnosis, and procedure codes and clinical trial information to create feature vectors and then pass it to RF to predict the enrollment rate category.

Table 11.1 Doctor2Vec achieves the best performance on both metrics in predicting actual enrollment rate (regression task) and rate categories (classification task) compared to state-of-the-art baselines. Results of ten independent runs

	PR-AUC	R^2 Score
Median	0.571 ± 0.014	0.54 ± 0.072
LR	0.672 ± 0.041	0.314 ± 0.082
RF	0.731 ± 0.034	0.618 ± 0.034
AdaBoost	0.747 ± 0.002	0.684 ± 0.146
MLP	0.761 ± 0.019	0.762 ± 0.049
LSTM	0.792 ± 0.034	0.780 ± 0.621
DeepMatch	0.735 ± 0.068	0.821 ± 0.073
Doctor2Vec	$\mathbf{0.861 \pm 0.021}$	$\mathbf{0.841 \pm 0.072}$

The bold values indicate the best performance numbers across all methods

4. AdaBoost. They combine the medication, diagnosis, procedure codes, and clinical trial information to create feature vectors and then apply the AdaBoost classifier to predict the enrollment rate categories.
5. Multi-layer Perceptron (MLP). They use MLP to process doctor features. In this case, they obtain the doctor features by converting all the visit vectors associated with a doctor to a count vector of different diagnoses, medication, procedure codes. They convert categorical information of clinical trials to multi-hot vectors and obtain TF-IDF features from text information of clinical trials.
6. Long Short-Term Memory Networks (LSTM). They process all the temporally ordered visit vectors associated with a doctor using an LSTM. The embedding obtained from LSTM is concatenated with embedding obtained from categorical and text information of clinical trials to predict enrollment rate.
7. DeepMatch [56] In this model, the doctors' features are obtained by collecting the top 50 most frequent medical codes and passed through an MLP layer to obtain an embedding vector. This embedding is concatenated with embedding obtained from categorical and text information of clinical trials via MLP and TF-IDF to predict enrollment rate finally.

They conducted experiments for both classification (e.g., predict enrollment rate category) and regression (e.g., predict actual rate) tasks. Results are provided in Table 11.1. From the results, they observe that Doctor2Vec achieved the best performance in both settings.

In particular, Doctor2Vec demonstrated improved performance over the best baseline LSTM by up to 8.7% in PR-AUC. In the actual rate prediction task, Doctor2Vec gains 2.4% relative improvement in R^2 over the best baseline DeepMatch.

11.6 Case Study: Medication Recommendation

Problem How to select effective and safe medications? Can we perform medication recommendations based on medical history from EHR data and relevant

Table 11.2 Statistics of the data

# Patients	6350
# Clinical events	15,016
# Diagnosis	1958
# Procedure	1426
# Medication	145
Avg # of visits	2.36
Avg # of diagnosis	10.51
Avg # of procedure	3.84
Avg # of medication	8.80
Max # of diagnosis	128
Max # of procedure	50
Max # of medication	55
# Medication in DDI knowledge base	123
# DDI types in knowledge base	40

medical knowledge base such as drug-drug interactions (DDI) database? A good medication recommendation needs to perform recommendation based on longitudinal patient history and considers drug safety in their modeling, especially adverse drug-drug interactions (DDI). Authors [134] of proposed a memory network based approach called GAMENet for recommending medication.

Data The experiment is conducted using EHR data from MIMIC-III [81]. Here patients with more than one visit are included in the dataset. They used DDI knowledge from TWOSIDES dataset [152]. In this work, the Top-40 severity DDI types are transformed to ATC Third Level for integrating with MIMIC-III data. The dataset statistics are summarized in Table 11.2.

Method GAMENet model is a graph augmented memory network that embeds multiple knowledge graphs into the memory bank. GAMENet also enables attention-based memory search using query generated from longitudinal patient records. In particular, GAMENet not only stores the EHR graph and the drug-drug interaction graph as facts in Memory Bank (MB), but also inserts patient history to the Dynamic Memory (DM). Next, we present a high-level description of different modules in GAMENet, which is illustrated in Fig. 11.6.

- *Input medical embedding:* A visit $\boldsymbol{x}_t$ consists of $[\boldsymbol{c}_d^t, \boldsymbol{c}_p^t, \boldsymbol{c}_m^t]$ where each $\boldsymbol{c}_*^t$ is a multi-hot vector at the tth visit. The multi-hot vector $\boldsymbol{c}_*^t$ is binary encoded showing the existence of each medical codes recorded at the tth visit. A linear embedding is applied on the input vectors. We derive medical embeddings for $\boldsymbol{c}_d^t, \boldsymbol{c}_p^t$ for diagnosis codes and procedure codes separately at the tth visit as follows:

$$\boldsymbol{e}_*^t = \boldsymbol{W}_{*,e}\boldsymbol{c}_*^t$$

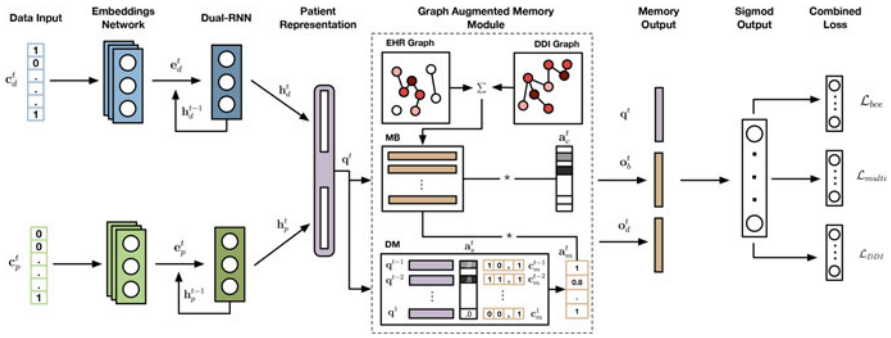

Fig. 11.6 GAMENet: At the tth visit, there are the multi-hot input vectors $\boldsymbol{c}_d^t, \boldsymbol{c}_p^t$ for diagnosis and procedure, respectively, which are mapped to embedding $\boldsymbol{e}_d^t, \boldsymbol{e}_p^t$ correspondingly. Then two separate RNN generates current hidden states on diagnosis $\boldsymbol{h}_d^t$ and procedure $\boldsymbol{h}_p^t$ based on embedding $\boldsymbol{e}_d^t, \boldsymbol{e}_p^t$. We use concatenated $\boldsymbol{h}_d^t, \boldsymbol{h}_p^t$ as query $\boldsymbol{q}^t$ (a.k.a. patient representation) to output $\boldsymbol{o}_b^t$ by reading from Memory Bank (MB) $\boldsymbol{M}_b$. Meantime, the Dynamic Memory (DM) stores key-value form history information along time and can be used to generate output $\boldsymbol{o}_d^t$. Finally, query and memory outputs are concatenated to make final medication recommendation. A combined loss is used to balance prediction performance and DDI

where $\boldsymbol{W}_{*,e} \in \mathbb{R}^{|\mathcal{C}_*| \times d}$ is the embedding matrix to learn and $*$ can be either d for diagnosis or p for procedure. Thus a visit $\boldsymbol{x}_t$ is transformed to $\hat{\boldsymbol{x}}_t = [\boldsymbol{e}_d^t, \boldsymbol{e}_p^t, \boldsymbol{c}_m^t]$ where $\boldsymbol{e}_d^t, \boldsymbol{e}_p^t$ are diagnosis and procedure embeddings at time t, and $\boldsymbol{c}_m^t$ is the multi-hot vectors for medication at time t.

- *Patient representation:* The two RNNs are used for modeling diagnosis sequences and procedure sequences of patients, separately. Thus, the RNNs accept all patient history to produce hidden states to generate patient representation as a query $\boldsymbol{q}^t$. Formally, for each input vector in transformed clinical history $[\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2, \cdots, \hat{\boldsymbol{x}}_t]$, we retrieve $\boldsymbol{e}_d, \boldsymbol{e}_p$ and utilize RNN to encode visit-level diagnosis and procedure embeddings respectively as follows:

$$\boldsymbol{h}_d^t = RNN_d(\boldsymbol{e}_d^1, \boldsymbol{e}_d^2, \cdots, \boldsymbol{e}_d^t)$$
$$\boldsymbol{h}_p^t = RNN_p(\boldsymbol{e}_p^1, \boldsymbol{e}_p^2, \cdots, \boldsymbol{e}_p^t)$$

  Finally, we can converts hidden states into a query vector as $\boldsymbol{q}^t = f([\boldsymbol{h}_d^t, \boldsymbol{h}_p^t])$ where we concatenate hidden diagnosis state $\boldsymbol{h}_d^t$ and procedure state $\boldsymbol{h}_p{}^t$ as the input patient health state. $f(\cdot)$ is a fully connected neural network with one hidden layer.
- *Static memory bank:* To leverage drug knowledge, we construct a graph augmented memory module that not only embeds and stores the EHR graph and the DDI graph as facts in Memory Bank (MB). Here graph convolution neural

networks (GCN)[1] are used to construct the knowledge embedding vectors for medications to be stored in the memory bank. More formally, EHR graph and DDI graph are represented by two adjacency matrices $\boldsymbol{A}_e$, $\boldsymbol{A}_d$, respectively. $\boldsymbol{A}_e$ incorporates medication co-prescriptions (i.e., edges between drugs that are often co-prescribed together) while $\boldsymbol{A}_d$ encodes drug-drug interactions (i.e., two drugs with DDI are connected). Following the GCN procedure [88], each $\boldsymbol{A}_*$ is pre-processed as follows:

$$\tilde{\boldsymbol{A}}_* = \tilde{\boldsymbol{D}}^{-\frac{1}{2}}(\boldsymbol{A}_* + \boldsymbol{I})\tilde{\boldsymbol{D}}^{-\frac{1}{2}}$$

where $\tilde{\boldsymbol{D}}$ is a diagonal matrix such that $\tilde{\boldsymbol{D}}_{ii} = \sum_j \boldsymbol{A}_{ij}$ and $\boldsymbol{I}$ is an identity matrix. Then we applied a two-layer GCN on $\boldsymbol{A}_e$ and $\boldsymbol{A}_d$, respectively. The output $\boldsymbol{M}_b$ is generated as a weighted sum of the two graph embeddings.

$$\begin{aligned} \boldsymbol{Z}_1 &= \tilde{\boldsymbol{A}}_e \tanh(\tilde{\boldsymbol{A}}_e \boldsymbol{W}_{e1})\boldsymbol{W}_1 \\ \boldsymbol{Z}_2 &= \tilde{\boldsymbol{A}}_d \tanh(\tilde{\boldsymbol{A}}_d \boldsymbol{W}_{e2})\boldsymbol{W}_2 \\ \boldsymbol{M}_b &= \boldsymbol{Z}_1 - \beta \boldsymbol{Z}_2 \end{aligned} \tag{11.4}$$

where $\boldsymbol{W}_{e1}$, $\boldsymbol{W}_{e2} \in \mathbb{R}^{|\mathcal{C}_m| \times d}$ are embedding matrices for EHR graph and DDI graph (each contains $|\mathcal{C}_m|$ number of d-dimensional vectors), $\boldsymbol{W}_1$, $\boldsymbol{W}_2 \in \mathbb{R}^{d \times d}$ are parameter matrices. All $\boldsymbol{W}_*$ are updated during the training phase. Then, medication/node embeddings $\boldsymbol{Z}_1$, $\boldsymbol{Z}_2 \in \mathbb{R}^{|\mathcal{C}_m| \times d}$ are computed using GCN. Finally we combine node embeddings $\boldsymbol{Z}_1$ and $\boldsymbol{Z}_2$ into $\boldsymbol{M}_b \in \mathbb{R}^{|\mathcal{C}_m| \times d}$ where β is a weighting variable to fuse both graphs. The combined embedding $\boldsymbol{M}_b$ for each medication will be stored into the memory bank.

- *Dynamic memory bank* stores patient history to Dynamic Memory (DM) as key-value pairs. The keys are the previous patient representation query $\boldsymbol{q}^{t'}, t' < t$, while the values are the corresponding multi-hot medication vectors $\boldsymbol{c}_m^{t'}, t' < t$ that are used in the past. Specifically, we can incrementally insert key-value pair after each visit step and treat $\boldsymbol{M}_d^t$ as a vectorized indexable dictionary as follows $\boldsymbol{M}_d^t = \{\boldsymbol{q}^{t'} : \boldsymbol{c}_m^{t'}\}_1^{t-1}$ where $\boldsymbol{M}_d^t$ is empty when $t = 1$. For clarity, we use $\boldsymbol{M}_{d,k}^t = [\boldsymbol{q}^1, \boldsymbol{q}^2, \cdots, \boldsymbol{q}^{t-1}] \in \mathbb{R}^{d \times |t-1|}$ to denote the key vectors and $\boldsymbol{M}_{d,v}^t = [\boldsymbol{c}_m^1; \boldsymbol{c}_m^2; \cdots; \boldsymbol{c}_m^{t-1}] \in \mathbb{R}^{|t-1| \times |\mathcal{C}_m|}$ to denote the value vectors at tth visit.
- *Medication recommendation:* We first query two memory banks MB and DM with current patient representation q^t to retrieve output $\boldsymbol{o}_b^t$, $\boldsymbol{o}_d^t$ as follows:

$$\boldsymbol{o}_b^t = \boldsymbol{M}_b^{\intercal} \overbrace{\text{Softmax}(\boldsymbol{M}_b \boldsymbol{q}^t)}^{\boldsymbol{a}_c^t}$$

[1] GCN will be covered in Chap. 10.

$$\boldsymbol{o}_d^t = \boldsymbol{M}_b^\intercal \overbrace{(\boldsymbol{M}_{d,v}^t)^\intercal \underbrace{\text{Softmax}((\boldsymbol{M}_{d,k}^t)^\intercal \boldsymbol{q}^t)}_{\boldsymbol{a}_s^t}}^{\boldsymbol{a}_m^t}$$

Then we concatenate query $\boldsymbol{q}^t$ and their output from memory banks and perform the final classification

$$\hat{\boldsymbol{y}}_t = \sigma([\boldsymbol{q}^t, \boldsymbol{o}_b^t, \boldsymbol{o}_d^t])$$

Results

The performance of GAMENet is compared with the following baselines.

- **Nearest** will simply recommend the same combination medications at previous visit for current visit (i.e., $\hat{Y}_t = Y_{t-1}$)
- **Logistic Regression (LR)** is a logistic regression with L2 regularization. Here we represent the input data by sum of one-hot vector.
- **LEAP** [174] is an instance-based medication combination recommendation method.
- **RETAIN** [25] can provide a sequential prediction of medication combination based on a two-level neural attention model that detects influential past visits and significant clinical variables within those visits.
- **DMNC** [95] is a recent work of medication combination prediction via memory augmented neural network.

Table 11.3 compares the performance on accuracy and safety issues. Results show GAMENet has the highest score among all baselines for Jaccard, PR-AUC, and F1.

Table 11.3 Performance comparison of GAMENet and baseline methods

Methods	DDI rate	△ DDI rate %	Jaccard	PR-AUC	F1	Avg # of med.
Nearest	0.0791	+1.80%	0.3911	0.3805	0.5465	14.77
LR	0.0786	+1.16%	0.4075	0.6716	0.5658	11.42
Leap	**0.0532**	**−31.53%**	0.3844	0.5501	0.5410	14.42
RETAIN	0.0797	+2.57%	0.4168	0.6620	0.5781	16.68
DMNC	0.0949	+22.14%	0.4343	0.6856	0.5934	20.00
GAMENet	0.0749	−3.60%	**0.4509**	**0.6904**	**0.6081**	14.02

The bold values indicate the best performance numbers across all methods

11.7 Case Study: Pre-training of Graph Augmented Transformers for Medication Recommendation

Problem G-BERT is another medication recommendation algorithm that extends the GAMENet model. Recall GAMENet utilizes memory networks for medication recommendation [134]. However, there are two limitations with GAMENet, which are addressed by G-BERT, a graph augmented Transformer model [133].

1. **Selection bias:** Data from patients who only have one hospital visit were discarded from training by GAMENet since it requires more than one visit to build a temporal prediction model like GAMENet. As a result, the training population is biased towards more severe patients with multiple inpatient visits.
2. **Lack of hierarchical knowledge:** Medical ontologies such as a diagnosis ontology follow hierarchical structures that were not utilized in GAMENet.

Method G-BERT is a model that first derives the initial embedding of medical codes from medical ontology using graph neural networks (**Ontology embedding**). Then, G-BERT constructs a variation of the BERT model on data from all patients of single or multiple visits (**Visit embedding**). Finally, we add a prediction layer to fine-tune the medication recommendation model (**Fine-tuning**). Figure 11.7 illustrates the model architecture of G-BERT.

Ontology Embedding

We constructed ontology embedding from diagnosis ontology $\mathcal{O}_d$ and medication ontology $\mathcal{O}_m$. Since the medical codes in raw EHR data can be considered leaf nodes in these ontology trees, we can enhance the medical code embedding using graph neural networks (GNNs) to integrate these codes' ancestors' information.

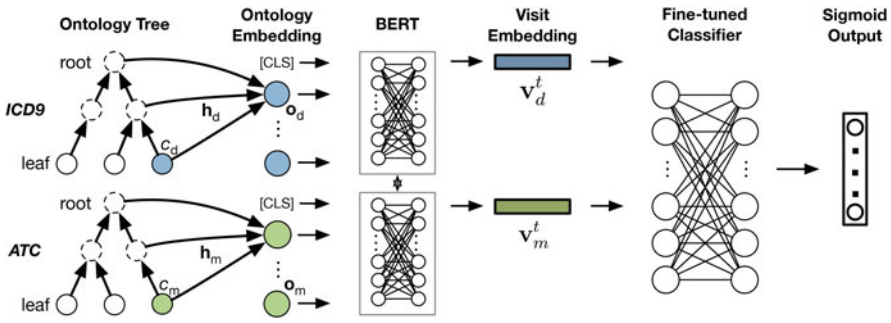

Fig. 11.7 G-BERT architecture: It consists of three parts: ontology embedding, visit embedding and classification fine-tuning. First, we derive ontology embedding for medical codes from medical ontology using graph attention networks. Second, we input diagnosis and medication ontology embeddings separately to the BERT models. Third, we concatenate the mean of all previous visit embeddings and the current visit embedding as the final feature vector to train a classifier for medication recommendation

Here we perform a two-stage procedure with a specially designed GNN for ontology embedding.

To start, we assign an initial embedding vector to every medical code $c_* \in \mathcal{O}_*$ with a learnable embedding matrix $\boldsymbol{W}_e \in \mathbb{R}^{|\mathcal{O}_*| \times d}$ where d is the embedding dimension. Here $\overline{\mathcal{C}_*}$ and $\mathcal{C}_*$ correspond to the set of non-leaf nodes and that of the leaf nodes, respectively.

Bottom-Up Stage For each non-leaf node $c_* \in \overline{\mathcal{C}_*}$, we obtain its enhanced medical embedding $\boldsymbol{h}_{c_*} \in \mathbb{R}^d$ as follows:

$$\boldsymbol{h}_{c_*} = g(c_*, ch(c_*), \boldsymbol{W}_e)$$

where $g(\cdot, \cdot, \cdot)$ is an aggregation function which accepts the target medical code c_*, its direct children $ch(c_*)$ and initial embedding matrix. Intuitively, the aggregation function can fuse information into the target node c_* from its direct children.

Top-Down Stage For the leaf nodes, we use the embedding matrix of non-leaf nodes $\boldsymbol{H}_e \in \mathbb{R}^{|\overline{\mathcal{O}_*}| \times d}$ in a top-down fashion to obtain their embeddings $c_* \in \mathcal{C}_*$.

$$\boldsymbol{o}_{c_*} = g(c_*, pa(c_*), \boldsymbol{H}_e)$$

where $g(\cdot, \cdot, \cdot)$ accepts the target medical code c_* and its parents $pa(c_*)$.

The option for the aggregation function $g(\cdot, \cdot, \cdot)$ is flexible, including *sum*, *mean*. Here we choose the one from graph attention networks (GAT) [161]. For example, for the top-down stage, the aggregation function is

$$g(c_*, p(c_*), \boldsymbol{H}_e) = \Big\|_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_{c_*}} \alpha_{c_*,j}^k \boldsymbol{W}^k \boldsymbol{h}_j\right)$$

where $\|$ represents concatenation of the K-dimensional output from the multi-head attention mechanism, σ is a nonlinear activation function, $\boldsymbol{W}^k \in \mathbb{R}^{m \times d}$ is the k-th weight matrix for input transformation, and $\alpha_{c_*,j}^k$ are the corresponding k-th normalized attention coefficients computed as follows:

$$\alpha_{c_*,j}^k = \frac{\exp\left(\text{LeakyReLU}(\boldsymbol{a}^\intercal [\boldsymbol{W}^k \boldsymbol{h}_{c_*} || \boldsymbol{W}^k \boldsymbol{h}_j])\right)}{\sum_{k \in \mathcal{N}_{c_*}} \exp\left(\text{LeakyReLU}(\boldsymbol{a}^\intercal [\boldsymbol{W}^k \boldsymbol{h}_{c_*} || \boldsymbol{W}^k \boldsymbol{h}_k])\right)} \tag{11.5}$$

where $\mathcal{N}_{c_*}$ is the neighbors of c_* which is $ch(c_*) \cup c_*$ in bottom-up stage and $pa(c_*) \cup c_*$ in top-down stage, respectively. $\boldsymbol{a} \in \mathbb{R}^{2d/K}$ is a learnable weight vector and LeakyReLU is a nonlinear activation function introduced in Sect. 4.1.1.

As shown in Fig. 11.7, ontology hierarchies for diagnosis and medications are constructed based on ICD and Anatomical Therapeutic Chemical (ATC) Classification, respectively.

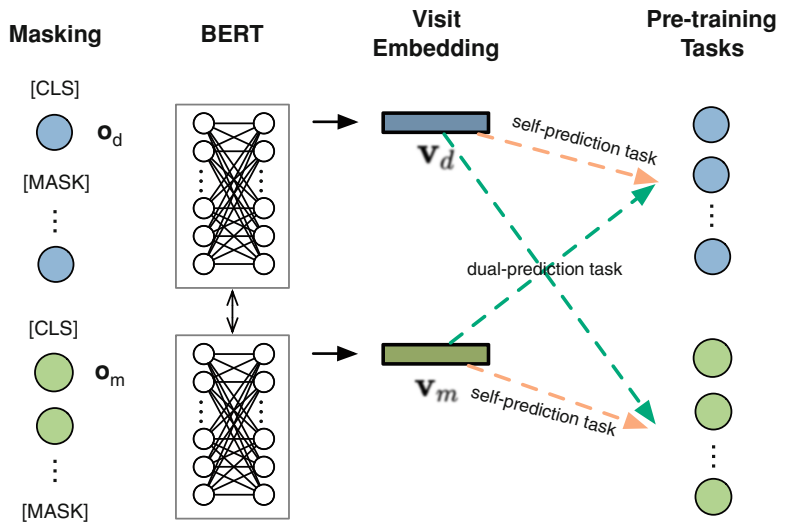

Fig. 11.8 Graphical illustration of pre-training procedure. We firstly randomly mask the input medical codes using a *[MASK]* symbol. **Orange arrow**: self-prediction task takes $\boldsymbol{v}_m$ or $\boldsymbol{v}_d$ as input to restore the original medical codes with the same type. **Green arrow**: dual-prediction task takes one type of visit embedding such as $\boldsymbol{v}_m$ or $\boldsymbol{v}_d$ and tries to predict the other type of medical codes

Visit Embedding

Similar to BERT, a multi-layer Transformer architecture is used to compute the visit embeddings. More specifically, the Transformer model maps a sequence of tokens (ontology embeddings of medical codes) to another sequence of embeddings (Fig. 11.8).

To generate a visit embedding summarizing all medical codes within the visit, a special token *[CLS]* is introduced to separate all the medical codes between visits. And the output embedding for *[CLS]* corresponds to the overall embedding for the entire visit. Formally, the visit embedding $\boldsymbol{v}_*^t \in \mathbb{R}^d$ for a patient at the t-th visit is the first element of the output from the Transformer model:

$$\boldsymbol{v}_*^t = \text{Transformer}(\{[CLS]\} \cup \{\boldsymbol{o}_{c_*}^t | c_* \in C_*^t\})[0] \tag{11.6}$$

where [0] means the first element of the output sequence, *[CLS]* is a special token as in BERT indicating the first position of each visit of a specific type, visit type $*$ can be diagnosis and medication. In fact, it is more reasonable to use Transformers as encoders (multi-head attention based architecture) than RNN for visit embedding since medical codes within one visit are unordered. Also, the positional embedding is not used in G-BERT as the unordered nature of medical codes within a visit.

Pre-training Before predicting medications, we need to learn better embeddings for medical codes using data from all visits. Many existing temporal models, such

as RNN, require two or more visits per patient to train a model. However, there are many patients with only a single visit, which before G-BERT are mostly ignored in model training.

The pre-training strategy of G-BERT utilizes the data of those single visits. In particular, G-BERT modified the original pre-training tasks, i.e., Masked language model task and Next Sentence prediction task to **self-prediction task** and **dual-prediction task**. The idea to conduct these tasks is to visit embedding, absorb enough information about what it is made of, and predict.

Thus, for the **self-prediction task**, we want the visit embedding v_* to recover *what it is made of*, i.e., the input medical codes $\mathcal{C}_*$ limited by the same type for each visit as follows:

$$\begin{aligned}\mathcal{L}_{se}(\boldsymbol{v}_*, \mathcal{C}_*^{(n)}) &= -\log p(\mathcal{C}_*^{(n)}|\boldsymbol{v}_*) \\ &= -\sum_{c_* \in \mathcal{C}_*^{(n)}} \log p(c_*|\boldsymbol{v}_*) + \sum_{c_* \in \{\mathcal{C}_* \setminus \mathcal{C}_*^{(n)}\}} \log p(c_*|\boldsymbol{v}_*)\end{aligned}$$

where $\mathcal{C}_*^{(n)}$ is the medical codes set of the n-th patient, $* \in \{d, m\}$ indicating diagnosis and medication respectively. And we minimize the binary cross entropy loss $\mathcal{L}_{se}$. For instance, assume that the n-th patient takes 10 different medications out of total 100 medications which means $|\mathcal{C}_m^{(n)}| = 10$ and $|\mathcal{C}_m| = 100$. In such case, we instantiate and minimize $L_{se}(\boldsymbol{v}_m, \mathcal{C}_m^{(n)})$ to produce high probabilities among 10 taken medications captured by $-\sum_{c_* \in \mathcal{C}_m^{(n)}} \log p(c_*|\boldsymbol{v}_m)$ and lower the probabilities among 90 other medications captured by $\sum_{c_* \in \{\mathcal{C}_m \setminus \mathcal{C}_m^{(n)}\}} \log p(c_*|\boldsymbol{v}_m)$. In practice, Sigmoid$(f(\boldsymbol{v}_*))$ is applied after a fully connected neural network $f(\cdot)$ with one hidden layer. G-BERT also uses specific symbol [MASK] to randomly replace the 15% original medical codes $c_* \in \mathcal{C}_*$.

The **dual-prediction task** predicts different types of medical codes based on the other types of codes. For example, we want to predict multiple medications given only the diagnosis codes. Inversely, we can also predict unknown diagnosis given the medication codes. Formally, predicting diagnoses based on medication embedding $p(\mathcal{C}_d|\boldsymbol{v}_m)$ and predicting medications based on diagnosis embedding $p(\mathcal{C}_m|\boldsymbol{v}_d)$ lead the following loss:

$$\mathcal{L}_{du} = -\log p(\mathcal{C}_d|\boldsymbol{v}_m) - \log p(\mathcal{C}_m|\boldsymbol{v}_d)$$

The probability scores $p(\mathcal{C}_d|\boldsymbol{v}_m)$ and $p(\mathcal{C}_m|\boldsymbol{v}_d)$ are learned from two DNN models with a hidden layer Sigmoid$(f_1(\boldsymbol{v}_m))$, Sigmoid$(f_2(\boldsymbol{v}_d))$. This is a direct adaptation of the next sentence prediction task.

Thus, our final pre-training optimization objective can simply be combining the aforementioned losses, as shown in Eq. (11.7).

$$\mathcal{L}_{pr} = \mathcal{L}_{se}(\boldsymbol{v}_d, \mathcal{C}_d^{(n)}) + \mathcal{L}_{se}(\boldsymbol{v}_m, \mathcal{C}_m^{(n)}) + \mathcal{L}_{du} \tag{11.7}$$

Fine Tuning
After obtaining pre-trained visit representation for each visit, we aggregate all the visit embedding and add a prediction layer for the medication recommendation task. To be specific, from pre-training on all visits, we have a pre-trained Transformer encoder, which can then be used to get the visit embedding $\boldsymbol{v}_*^\tau$ at time τ. The known diagnosis codes $\mathcal{C}_d^t$ at the prediction time t is also represented using the same model as $\boldsymbol{v}_*^t$. Concatenating the mean of previous diagnoses visit embeddings and medication visit embeddings, also the last diagnoses visit embedding, we built a DNN based prediction layer to predict the recommended medication codes as:

$$\boldsymbol{y}_t = \text{Sigmoid}(\boldsymbol{W}_1[(\frac{1}{t}\sum_{\tau<t}\boldsymbol{v}_d^\tau)||(\frac{1}{t}\sum_{\tau<t}\boldsymbol{v}_m^\tau)||\boldsymbol{v}_d^t] + b)$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{|\mathcal{C}_m|\times 3d}$ is a learnable transformation matrix.

Given the true labels $\hat{\boldsymbol{y}}_t$ at each time stamp t, the loss function for the whole EHR sequence (i.e. a patient) is

$$\mathcal{L} = -\frac{1}{T-1}\sum_{t=2}^{T}(\boldsymbol{y}_t^\intercal \log(\hat{\boldsymbol{y}}_t) + (1-\boldsymbol{y}_t^\intercal)\log(1-\hat{\boldsymbol{y}}_t)) \tag{11.8}$$

Results We used EHR data from MIMIC-III [81]. We utilize data from patients with both single visits and multiple visits in the training dataset as the pre-training data source (multi-visit data are split into visit slices). In this work, we transform the drug coding from NDC to ATC Third Level. The statistics of the datasets are summarized in Table 11.4.

Different baseline methods include

1. **Logistic Regression (LR)** is logistic regression with L1/L2 regularization. Here we represent sequential multiple medical codes by the sum of the multi-hot vector of each visit.
2. **LEAP** [174] is an instance-based medication combination recommendation method that formalizes the task in a multi-instance and multi-label learning framework. It utilizes an encoder-decoder based model with an attention mechanism to build complex dependency among diseases and medications.

Table 11.4 Statistics of the data (dx for diagnosis, rx for medication)

Stats	Single-visit	Multi-visit
# of patients	30,745	6350
Avg # of visits	1.00	2.36
Avg # of dx	39	10.51
Avg # of rx	52	8.80
# of unique dx	1997	1958
# of unique rx	323	145

3. **RETAIN** [25] makes sequential prediction of medication combination and diseases prediction based on a two-level neural attention model that detects influential past visits and clinical variables within those visits.
4. **GRAM** [28] injects domain knowledge (ICD9 Dx code tree) to **tanh** via attention mechanism.
5. **GAMENet** [134] is the method to recommend accuracy and safe medication based on memory neural networks and graph convolutional networks by leveraging EHR data and Drug-Drug Interaction (DDI) data source. For a fair comparison, we use a variant of GAMENet without DDI knowledge and procedure codes as input renamed as GAMENet^{-}.
6. **G-BERT** is our proposed model which integrated the GNN representation into Transformer-based visit encoder with pre-training on single-visit EHR data.

To measure the prediction accuracy, we used the Jaccard Similarity Score (Jaccard), Average F1 (F1), and Precision-Recall AUC (PR-AUC). Jaccard is defined as the size of the intersection divided by the size of the union of ground truth set $Y_t^{(k)}$ and predicted set $\hat{Y}_t^{(k)}$.

$$\text{Jaccard} = \frac{1}{\sum_k^N \sum_t^{T_k} 1} \sum_k^N \sum_t^{T_k} \frac{|Y_t^{(k)} \cap \hat{Y}_t^{(k)}|}{|Y_t^{(k)} \cup \hat{Y}_t^{(k)}|}$$

where N is the number of patients in the test set and T_k is the number of visits of the kth patient.

Table 11.5 compares the performance on the medication recommendation task. Incorporating both hierarchical ontology information and pre-training procedure, the end-to-end model G-BERT has more capacity and achieve comparable results with others.

G-BERT performs better than all the baseline methods including the GAMENet model.

Table 11.5 Performance on medication recommendation task

Methods	Jaccard	PR-AUC	F1	# of parameters
LR	0.4075	0.6716	0.5658	–
GRAM	0.4176	0.6638	0.5788	3,763,668
LEAP	0.3921	0.5855	0.5508	1,488,148
RETAIN	0.4456	0.6838	0.6064	2,054,869
GAMENet^{-}	0.4401	0.6672	0.5996	5,518,646
GAMENet	0.4555	0.6854	0.6126	5,518,646
G-BERT	**0.4565**	**0.6960**	**0.6152**	3,034,045

The bold values indicate the best performance numbers across all methods

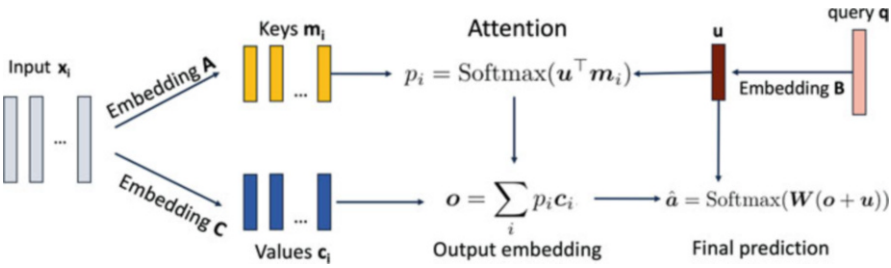

Fig. 11.9 End-to-end memory network exercise

11.8 Exercises

1. What is the main difference between the end-to-end memory network and the self-attention/transformer?
2. What is the most innovative step in the original memory network model and why?
3. What is NOT true about end-to-end memory networks as illustrated in Fig. 11.9?

   (a) It removes the arg max operation in the original memory networks so that gradient can be computed at all the steps.
   (b) The size of the embeddings in keys, values, and query are all the same.
   (c) Multi-layer end-to-end memory network can be created via multiple embedding matrices for keys and values
   (d) The model parameters are $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{W}$ matrices.

4. What is NOT true about self attention?

   (a) Three versions of embeddings of the input $\boldsymbol{X}$ are produced namely $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$.
   (b) The main reason to have multiple embedding matrices of $\mathbf{X}$ is to be able to apply and learn those embeddings independently in parallel.
   (c) The temporal dependency in the input sequences is essential in self-attention models.
   (d) Multiple versions of self-attention are concatenated together to produce more robust embedding, which is called multi-head self attention.

5. What are the method ideas used in Transformer encoder? [multiple correct choices]

   (a) Self attention
   (b) Positioning encoding
   (c) Multi-head attention
   (d) Residual connection

6. What are method ideas used in Transformer decoder? [multiple correct choices]

   (a) Masked attention
   (b) Self attention
   (c) Residual connection
   (d) Recurrent neural networks

7. What is NOT true about BERT model?

   (a) BERT provides a masked language model.
   (b) BERT masks x% of input words and try to predict them with other words.
   (c) BERT uses a transformer based model.
   (d) BERT provides static embeddings for all words.

8. What is the healthcare application in Doctor2Vec?
9. What is the healthcare application in GameNet?
10. What are the method ideas behind G-BERT? [multiple correct choices]

    (a) Medical ontology is used to embed medical concepts.
    (b) BERT model can be used to pre-train on large number of clinical visits.
    (c) Temporal dependencies across visits are important in prediction.
    (d) Multi-task learning is useful in clinical applications.