EECS 182 Deep Neural Networks
Spring 2023 Anant Sahai

# Discussion 7

## 1. Beam Search

When making predictions with an autoregressive sequence model, it can be intractable to decode the true most likely sequence of the model, as doing so would require exhaustively searching the tree of all $O(M^T)$ possible sequences, where $M$ is the size of our vocabulary, and $T$ is the max length of a sequence. We could decode our sequence by greedily decoding the most likely token each timestep, and this can work to some extent, but there are no guarantees that this sequence is the actual most likely sequence of our model.

Instead, we can use beam search to limit our search to only candidate sequences that are the most likely so far. In beam search, we keep track of the $k$ most likely predictions of our model so far. At each timestep, we expand our predictions to all of the possible expansions of these sequences after one token, and then we keep only the top $k$ of the most likely sequences out of these. In the end, we return the most likely sequence out of our final candidate sequences. This is also not guaranteed to be the true most likely sequence, but it is usually better than the result of just greedy decoding.

The beam search procedure can be written as the following pseudocode:

**Algorithm 1** Beam Search

**for** each time step $t$ **do**
  **for** each hypothesis $y_{1:t-1,i}$ that we are tracking **do**
    find the top $k$ tokens $y_{t,i,1}$,...,$y_{t,i,k}$
  **end for**
  sort the resulting $k^2$ length $t$ sequences by their total log-probability
  store the top $k$
  advance each hypothesis to time $t+1$
**end for**

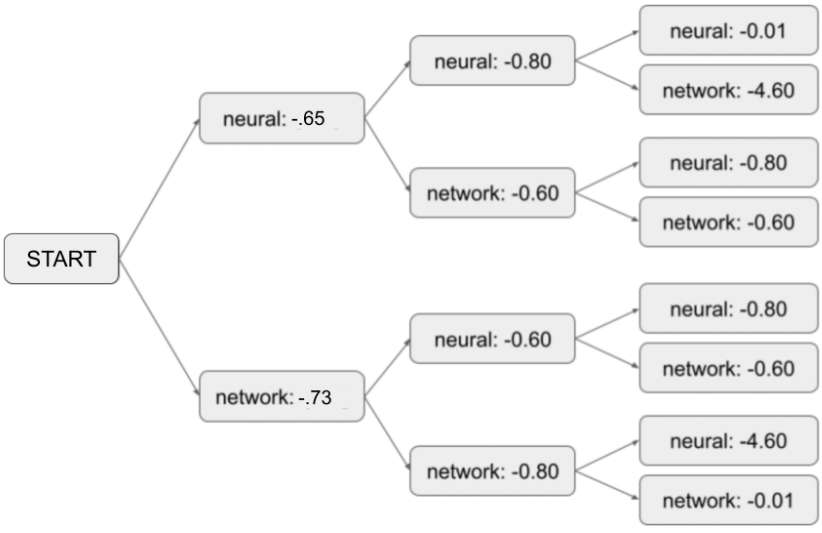


**Figure 1:** The numbers shown are the decoder's log probability prediction of the current token given previous tokens.

  

We are running the beam search to decode a sequence of length 3 using a beam search with $k = 2$. Consider predictions of a decoder in Figure 1, where each node in the tree represents the next token **log probability** prediction of one step of the decoder conditioned on previous tokens. The vocabulary consists of two words: "neural" and "network".

(a) **At timestep 1, which sequences is beam search storing?**

**Solution:** There are only two options, so our beam search keeps them both: "neural" (log prob = -.65) and "network" (log prob = -.73).

(b) **At timestep 2, which sequences is beam search storing?**

**Solution:** We consider all possible two word sequences, but we then keep only the top two, "neural network" (with log prob = -.65 - .6 = -1.25) and "network neural" (with log prob = -.73 - .6 = -1.33).

(c) **At timestep 3, which sequences is beam search storing?**

**Solution:** We consider three word sequences that start with "neural network" and "network neural", and the top two are "neural network network" (with log prob = -.65 - .6 - .6 = -1.85) and "network neural network" (with log prob = -.73 - .6 - .6= -1.93).

(d) **Does beam search return the overall most-likely sequence in this example? Explain why or why not.**

**Solution:** No, the overall most-likely sequence is "neural neural neural" with log prob = -.65 - .8 - .01= -1.46). These sequences don't get returned since they get eliminated from consideration in step 2, since "neural neural" is not in the $k = 2$ most likely length-2 sequences.

(e) **What is the runtime complexity of generating a length-$T$ sequence with beam size $k$ with an RNN?** Answer in terms of $T$ and $k$ and $M$. (Note: an earlier version of this question said to write it in terms of just $T$ and $k$. This answer is also acceptable.)

**Solution:**

- Step RNN forward one step for one hypothesis = $O(M)$ (since we compute one logit for each vocab item, and none of the other RNN operations rely on $M$, $T$, or $K$).
- Do the above, and select the top $k$ tokens for one hypothesis. We do this by sorting the logits: $O(M \log M)$. (Note: there are more efficient ways to select the top $K$, for instance using a min heap. We just use this way since the code implementation is simple.) Combined with the previous step, this is $O(M \log M + M) = O(M \log M)$.
- Do the above for all $k$ current hypotheses $O(KM \log M)$.
- Do all above + choose the top $K$ of the $K^2$ hypotheses currently stored: we do this by sorting the array of $K^2$ items: $O(K^2 \log(K^2)) = O(K^2 \log(K))$ (since $\log(K^2) = 2\log(K)$). (Note: there are also more efficient ways to do this). Combining this with the previous steps, we get $O(KM \log M + K^2 \log(K))$. When one term is strictly larger than another we can take the max of the two. Since $M \geq K$, we could also write this as $O(KM \log M)$.
- Repeat this for $T$ timesteps: $O(TKM \log M)$.

  

# 2. Graph Neural Network Conceptual Questions

Diagrams in this question were taken from https://distill.pub/2021/gnn-intro. This blog post is an excellent resource for understanding GNNs and contains interactive diagrams:

A. You are studying how organic molecules break down when heated. For each molecule, you know the element and weight of each atom, which other atoms its connected to, the length of the bond the atoms, and the type of molecule it is (carbohydrate, protein, etc.) You are trying to predict which bond, if any, will break first if the molecule is heated.

(a) How would you represent this as a graph? (What are the nodes, edges, and global state representations? Is it directed or undirected?) **Solution:** Each atom is a node with the element and weight as its value. There is an undirected edge between each pair of bonded atoms, with the bond length as the value. The global representation includes the molecule type.

(b) How would you use the outputs of the last GNN layer to make the prediction? **Solution:** This is a classification problem across edges. One reasonable approach would be to softmax across logits produced by each edge as well as one logit produced by the global state which represents the option that no edges break.

(c) How would you encode the node representation? **Solution:** Use a learned embedding for the element ID. Concatenate this with the atom weight. If atom weights are large, you should normalize the weights first.

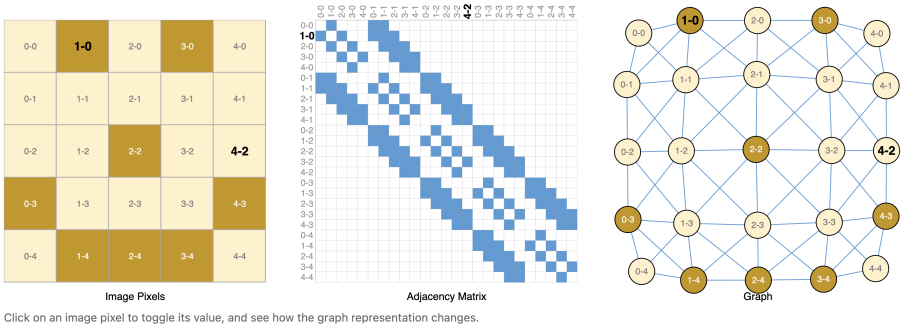


**Figure 2:** Images as Graphs

B. If you're doing a graph-level classification problem, but node values are missing for some of your graph nodes, how would you use this graph for prediction? **Solution:** There are multiple strategies (including all of the strategies used in other ML problems with missing values), including creating a special 'missing' token or filling the node with the mean value, and training with dropout on the input. In graphs where individual nodes don't matter much (e.g. point clouds), it may be appropriate to remove the node and associated edges entirely.

  

C. Consider the graph neural net architecture shown in Figure 3. It includes representations of nodes ($V_n$), edges ($E_n$), and global state ($U_n$). At each timestep, each node and edge is updated by aggregating neighboring nodes/edges, as well as global state. The global state is the updated by pooling all nodes and edges. For more details on the architecture setup, see https://distill.pub/2021/gnn-intro/#passing-messages-between-parts-of-the-graph.

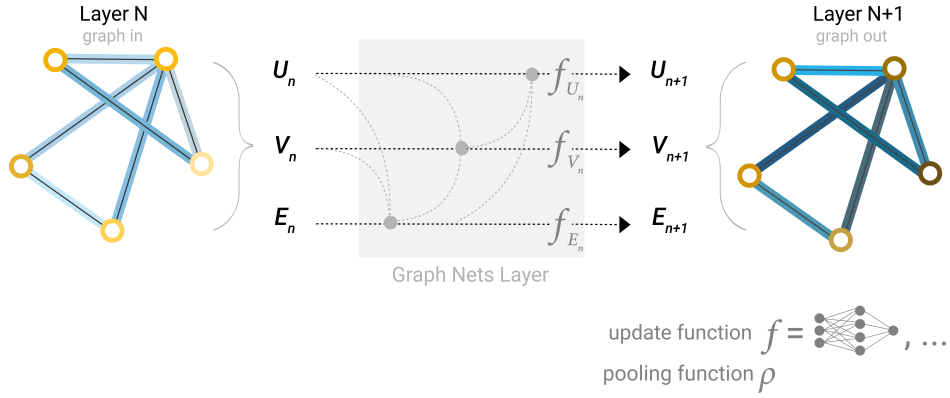


**Figure 3:** GNN architecture

(a) If we double the number of nodes in a graph which only has node representations, how does this change the number of learned weights in the graph? How does it change the amount of computation used for this graph if the average node degree remains the same? What if the graph if fully connected? (Assume you are not using a global state representation). **Solution:** The number of learned weights remains the same. If the node degree remains the same, computation will also double. If the graph is fully connected, computation scales by a factor of 4 (2x nodes * 2x neighbors to aggregate per timestep). (Here, we're thinking about how computational complexity scales with the number of nodes and edges. There are some fixed costs which don't scale, so it won't be quite 2x or 4x in practice.)

(b) Where in this network are learned weights incorporated? **Solution:** The update functions include learned weights.

(c) The diagram provided shows undirected edges. How would you incorporate directed edges? **Solution:** There are multiple strategies here, but one reasonable option is that within a node's update function, use different weights to incorporate incoming and outgoing edges and nodes. (Incoming edges are edges pointing toward the node, and outgoing edges have the arrows away from the node.) In some cases, it might make sense to use only incoming or outgoing nodes/edges for the update.

(d) The differences between an MLP and a RNN include (a) weights are shared between timesteps and (b) data is fed in one timestep at a time. How would you make an MLP-like GNN? What about an RNN-like GNN? **Solution:** An MLP-like GNN uses separate weights at each layer. The RNN-like GNN uses the same weights at each timestep. If data is sequential, then at each timestep

  

concatenate the node/edge values from your data at time $t$ with the hidden node/edge values for that timestep.

D. Let's say you run a large social network and are trying to predict whether individuals in the network like a particular ad. Each individual is represented as a node in the graph, and we are doing a node-level prediction problem. You have labels for around half of the people in the network and are trying to predict the other half. Unlike a traditional ML problem, where there there are many independent training points, here we have a single social network. How would you do prediction on this graph?
**Solution:** You can think about each node's classification as a separate training point. Split the nodes for which you have labels into train and validation sets, then train on the training nodes only. During training, we keep validation and unlabeled nodes in the graph (this is fine, since we aren't using their labels). Use the validation set to do model selection, then predict on the nodes without labels.

E. (Optional) Play around with different GNN design choices in the GNN playground at `https://distill.pub/2021/gnn-intro/#gnn-playground`. Which design choices lead to the best AUC?

**Contributors:**

- CS 182 Staff from past semesters.
- Olivia Watkins.
- Kumar Krishna Agrawal.
- Dhruv Shah.

  