question @465 120 views

Actions

Midterm Prep: 8. RNN, GRU, LSTM

Howdy everyone,

In preparation for the midterm, I've put together some notes on the various topics, drawing from @416 and the practice midterm.

Going through these notes and completing the practice midterm & quiz has definitely boosted my confidence! Here, I'm happy to share these notes with you all.

Let's ace the midterm together!

Feel free to contribute and add your own insights to these notes as well.

Best regards,
Your somewhat helpful AI bot, Darin

exam

Edit good question | 1 Updated 3 days ago by Darin Zhen

the students' answer, *where students collectively construct a single answer*

Actions

RNN

Recurrent Neural Networks (RNNs) are a class of neural networks designed to handle sequential data by maintaining an internal state (memory) that captures information about previous time steps. RNNs are particularly well-suited for tasks where the input and output have a temporal dependency, such as time series prediction, sequence modeling, language modeling, and speech recognition.

RNNs are powerful models for handling sequential data and have been successfully applied to a wide range of tasks in natural language processing, time series analysis, speech recognition, and more. However, they have limitations in capturing long-term dependencies due to the vanishing gradient problem, which led to the development of more advanced architectures like LSTM and GRU.

1. Recurrent Connections:

- Unlike feedforward neural networks, which process each input independently, RNNs have recurrent connections that allow information to persist over time by passing the output of a neuron as an input to itself at the next time step.
- This recurrent structure enables RNNs to capture temporal dependencies and context information from previous time steps, making them suitable for sequential data processing.

2. Time Steps and Sequence Length:

- RNNs process sequential data one time step at a time, where each time step corresponds to one element of the input sequence.
- The length of the input sequence (or the number of time steps) can vary depending on the task and the nature of the data.

3. Hidden State (Memory):

- At each time step, an RNN maintains a hidden state vector (also known as the memory) that captures information from previous time steps and serves as an intern representation of the input sequence up to that point.
- The hidden state is updated recursively at each time step based on the current input and the previous hidden state.

4. Recurrent Function:

- The recurrent function of an RNN computes the hidden state h_t at time step t based on the input x_t at that time step and the previous hidden state h_{t-1}.
- Mathematically, the recurrent function can be expressed as:

$h_t = f(x_t, h_{t-1})$

- Common choices for the activation function include the hyperbolic tangent (tanh) or the rectified linear unit (ReLU).

5. Output Function:

- Depending on the task, the output of an RNN can be computed at each time step or only at the final time step.
- For sequence prediction tasks, the output at each time step can be used to predict the next element in the sequence.
- For sequence classification tasks, the output at the final time step is typically used for making the final prediction.

6. Training with Backpropagation Through Time (BPTT):

- RNNs are trained using the backpropagation through time (BPTT) algorithm, which is a variant of the standard backpropagation algorithm adapted for recurrent connections.
- BPTT calculates gradients of the loss function with respect to the parameters of the RNN (weights and biases) by unrolling the network through time and applying t chain rule of calculus.
- The gradients are used to update the parameters of the RNN using gradient descent optimization methods.

7. Vanishing and Exploding Gradient Problems:

- RNNs are susceptible to the vanishing and exploding gradient problems, where gradients either become extremely small (vanishing) or extremely large (exploding) training.
- Techniques such as gradient clipping, using different activation functions, and using specialized RNN architectures like Long Short-Term Memory (LSTM) and Gate Recurrent Unit (GRU) are commonly employed to mitigate these issues.

GRU

Gated Recurrent Units (GRUs) are a type of recurrent neural network (RNN) architecture designed to address some of the limitations of traditional RNNs, such as the vanishing gradient problem and the difficulty in capturing long-term dependencies in sequential data. GRUs were introduced as a simpler alternative to Long Short-Te Memory (LSTM) networks, with comparable performance but fewer parameters.

GRUs address the vanishing gradient problem by incorporating gates that regulate the flow of information through the network, allowing them to capture long-term dependencies more effectively than traditional RNNs. They have fewer parameters than LSTM networks, making them computationally efficient and easier to train. GR have been widely used in various sequential data tasks, including natural language processing, speech recognition, and time series prediction.

Here's an overview of how GRUs work:

1. Update Gate:

- GRUs contain an update gate z_t which controls how much of the previous hidden state h_{t-1} should be retained and combined with the new candidate hidden stat
- The update gate is computed using a sigmoid activation function and determines the importance of the previous hidden state:
$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$
- Here, W_z and b_z are learnable parameters, and $[h_{t-1}, x_t]$represents the concatenation of the previous hidden state h_{t-1} and the current input x_t.

2. Reset Gate:

- GRUs also contain a reset gate r_t which controls how much of the previous hidden state h_{t-1} should be forgotten when computing the new candidate hidden stat
- The reset gate is computed using a sigmoid activation function:
$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$

3. Candidate Hidden State:

- At each time step, GRUs compute a candidate hidden state $\tilde{h}_t$ based on the current input x_t and the previous hidden state h_{t-1}.
- The candidate hidden state is a combination of the current input and a reset gate-controlled portion of the previous hidden state:
$\tilde{h}_t = \tanh(W \cdot [r_t \odot h_{t-1}, x_t] + b)$
- Here, W and b are learnable parameters, $\tanh$ is the hyperbolic tangent activation function, and $\odot$ represents element-wise multiplication.

4. Current Hidden State:

- The current hidden state h_t is a linear interpolation between the previous hidden state h_{t-1} and the candidate hidden state $\tilde{h}_t$, controlled by the update gate z_t:
$h_t = (1 - z_t) \odot \tilde{h}_t + z_t \odot h_{t-1}$

5. Output:

- The output of the GRU at each time step t can be computed based on the current hidden state h_t and is typically used for making predictions or passing to subse
layers in the neural network.

LSTM

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) architecture designed to address the vanishing gradient problem and capture term dependencies in sequential data. LSTMs were introduced in 1997 and have become one of the most widely used architectures for tasks involving sequential data as natural language processing, speech recognition, and time series prediction.

LSTMs address the vanishing gradient problem by incorporating a memory mechanism that allows them to retain information over long sequences and selectively upd information based on the current input. LSTMs have been widely used in various sequential data tasks due to their ability to capture long-term dependencies and effe model complex temporal patterns.

Here's an overview of how LSTM networks work:

1. Memory Cells:

- LSTMs contain memory cells that maintain a memory state (cell state) over time and selectively update and read from this memory state using different gates.
- The memory state allows LSTMs to store and remember information over long sequences, making them effective for capturing long-term dependencies.

2. Forget Gate:

- LSTMs have a forget gate f_t that controls how much of the previous memory state c_{t-1} should be retained and how much should be forgotten.
- The forget gate is computed using a sigmoid activation function and takes as input the concatenation of the previous hidden state h_{t-1} and the current input x_t:
$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
- Here, W_f and b_f) are learnable parameters, and σ represents the sigmoid activation function.

3. Input Gate:

- LSTMs have an input gate i_t that controls how much of the new information should be added to the memory state.
- The input gate is computed using a sigmoid activation function and takes as input the concatenation of the previous hidden state h_{t-1} and the current input x_t:
$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$

4. Candidate Memory State:

- At each time step, LSTMs compute a candidate memory state $\tilde{c}_t$ that represents the new information to be added to the memory state.
- The candidate memory state is computed using the hyperbolic tangent activation function and takes as input the concatenation of the previous hidden state h_{t-1} a current input x_t:
$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$

5. Update Memory State:

- LSTMs update the memory state c_t by selectively forgetting information from the previous memory state c_{t-1} and adding new information represented by the cand memory state $\tilde{c}_t$:
$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$

6. Output Gate:

- LSTMs have an output gate o_t that controls how much of the memory state c_t should be exposed as the output.
- The output gate is computed using a sigmoid activation function and takes as input the concatenation of the previous hidden state h_{t-1} and the current input x_t:
$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$

7. Current Hidden State:

- The current hidden state h_t is computed based on the memory state c_t and the output gate o_t:
$h_t = o_t \odot \tanh(c_t)$

Resolved Unresolved @465_f1

Darin Zhen 2 days ago

New book Ch 03 RNN Notes

- RNNs are useful for handling sequential data as they retain memory of previous states. This allows them to understand relationships between events in lengthy input sequences.
- Two common RNN variants are long short-term memory (LSTM) networks and gated recurrent units (GRUs). These address issues like vanishing/exploding gradients in RNNs.
- LSTMs have a more complex architecture with multiple gates (forget, input, cell, output) to regulate information flow and capture long-term dependencies.
- GRUs are a more streamlined version of LSTMs with just two gates (reset and update). Performance is often similar.
- RNNs can handle variable length inputs and different sequence-to-sequence mapping configurations like many-to-many.
- Training uses backpropagation through time (BPTT) to compute gradients along the timeline. Truncated BPTT is common.
- An application of using an LSTM network is to for early prediction of heart failure onset from longitudinal patient records.
- A great pipeline is to use the PyHealth library - data preprocessing, LSTM model initialization, training loop, and evaluation.

helpful! | 1

Reply to this followup discussion

Resolved Unresolved @465_f2

Darin Zhen 2 days ago

Old Book: RNN

- RNNs are good at modeling sequential data like longitudinal patient records, time series data, and text sequences. This makes them useful for healthcare applications involving clinical notes, EHR data, EEG/ECG signals, etc.
- Two major variants of RNNs are long short-term memory (LSTM) and gated recurrent units (GRU). These help address the vanishing gradient problem in basic RNNs, allowing them to learn long-term dependencies.
- Bidirectional RNNs process sequence data in both forward and backward directions, helping capture full context when modeling sequences like clinical notes.
- Sequence-to-sequence models use an encoder-decoder RNN architecture to map input sequences to output sequences, useful for tasks like generating medications from diagnoses.
- Key healthcare applications include early prediction of diseases like heart failure from longitudinal records, de-identifying clinical notes by removing PHI, predicting future diagnoses/medications/visit times, and learning to prescribe medication combinations for patients with multiple conditions.
- A major advantage of RNNs is the ability to handle inputs of varying lengths, like different numbers of patient visits. Parameter sharing also reduces model size.
- Challenges include vanishing gradients and difficulty capturing very long-range dependencies. Attention models can help address some of these limitations.

helpful! | 0

Reply to this followup discussion

Resolved Unresolved @465_f3

Darin Zhen 2 days ago

Lecture 07 RNN

- RNNs are useful for modeling sequence data like text, audio, video, patient records, etc. They have memory to capture long-term dependencies.
- The basic RNN has a simple cell to process the input and hidden state. But it struggles with vanishing gradients over long sequences.
- LSTM and GRU are more sophisticated RNN cells that address the vanishing gradient problem using gating mechanisms.
- RNNs can be trained with backpropagation through time (BPTT) to compute gradients w.r.t. the parameters.
- Bidirectional RNNs process the sequence both forwards and backwards to capture past and future context.
- Sequence-to-sequence models use an encoder-decoder RNN for tasks like translation. The encoder condenses the input sequence to a vector, and the decoder generates the output sequence.
- RNNs are useful in healthcare for early prediction of diseases like heart failure using longitudinal patient records. The models can learn temporal relations.
- RNNs can also generate predictions of next clinical events for a patient and generalize well to new institutions when warm-started.

helpful! | 0

Reply to this followup discussion

Start a new followup discussion

Compose a new followup discussion