Language Models and the Dataset

- In the previous lecture, saw above how to map text data into tokens, where these tokens can be words or characters.
- Assume that a text sequence of T tokens: $x_1, x_2, \ldots, x_T$.
- Given such a sequence, the goal of a **language model** is to estimate the joint probability of the sequence

$$P(x_1, x_2, \ldots, x_T).$$

Applications of Language Models

- Could generate natural text just on its own, simply by drawing one token at a time $x_t \sim P(x_t \mid x_{t-1}, \ldots, x_1)$.
- Could generate a meaningful dialog, simply by conditioning the text on previous dialog fragments.
- Help improve Speech Recognition
    - when two words or phrases sound similar (e.g. phrases "to recognize speech" and "to wreck a nice beach")

Learning a Language Model

- Suppose that we tokenize text data at the word level.
- By applying basic probability rules:

$$P(x_1, x_2, \ldots, x_T) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1, x_2)P(x_4 \mid x_1, x_2, x_3)$$

- Example:

$$P(\text{deep, learning, is, fun}) = P(\text{deep})P(\text{learning} \mid \text{deep})P(\text{is} \mid \text{deep, learning})P(\text{fun} \mid \text{deep, learning, is}).$$

- So in order to compute the language model, we need to calculate the probability of words and the conditional probability of a word given the previous few words.
- Such probabilities are essentially the language model parameters.

- The probability of words can be calculated from the relative word frequency of a given word in the training dataset.
    - For example, the estimate $\hat{P}(\text{deep})$ can be calculated by counting all occurrences of the word "deep" and dividing it by the total number of words in the corpus.

- Moving on, we wish to estimate

$$\hat{P}(\text{learning} \mid \text{deep}) = \frac{n(\text{deep, learning})}{n(\text{deep})},$$

  where $n(x)$ and $n(x, x')$ are the number of occurrences of single and consecutive word pairs, respectively.

- Estimating the probability of a word pair is somewhat more difficult, since the occurrences of "deep learning" are a lot less frequent.

- The number of model parameters increases exponentially with the number of tokens we condition upon.

Recurrent Neural Networks

- Rather than modeling $P(x_t \mid x_{t-1}, \ldots, x_{t-n+1})$ it is preferable to use a latent variable model:

$$P(x_t \mid x_{t-1}, \ldots, x_1) \approx P(x_t \mid h_{t-1}),$$

where h_{t-1} is a *hidden state* (also known as a hidden variable) that stores the sequence information up to time step $t-1$.

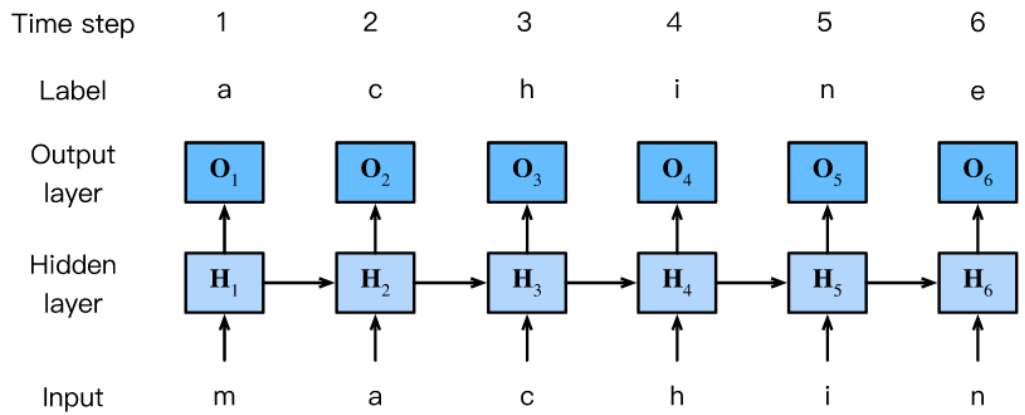
- The hidden state at any time step t can be computed based on both the current input x_t and the previous hidden state h_{t-1}:

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

- **Recurrent neural networks (RNNs)** are neural networks with *hidden states*.

RNN-based Character-Level Language Models

- For simplicity, we tokenize text into characters rather than words.
- The following figure demonstrates how an RNN predicts the next character based on the current and previous characters:



- During the training process, a softmax operation is applied on the output from the output layer for each time step, and cross-entropy loss is used to compute the error between the model output and the label.
- Due to the recurrent computation, the output of time step 3 $\mathbf{O}_3$, is determined by the text sequence "m", "a", and "c".

Neural Networks without Hidden States

- Before introducing the RNN model, we first revisit the MLP model.
    - We take a look at an MLP with a single hidden layer.
    - Let the hidden layer's activation function be ϕ.
- Given a minibatch of examples $\mathbf{X} \in \mathbb{R}^{n \times d}$ with batch size n and d inputs, the hidden layer's output $\mathbf{H} \in \mathbb{R}^{n \times h}$ is calculated as
$$\mathbf{H} = \phi(\mathbf{X}\mathbf{W}_{xh} + \mathbf{b}_h),$$
where, for the hidden layer, we denote the weights by $\mathbf{W}_{xh} \in \mathbb{R}^{d \times h}$, the bias by $\mathbf{b}_h \in \mathbb{R}^{1 \times h}$, and the number of hidden units by h.
- Next, the hidden variable $\mathbf{H}$ is used as the input of the output layer. The output layer is given by
$$\mathbf{O} = \mathbf{H}\mathbf{W}_{hq} + \mathbf{b}_q,$$
where $\mathbf{O} \in \mathbb{R}^{n \times q}$ is the output variable, $\mathbf{W}_{hq} \in \mathbb{R}^{h \times q}$ are the weights, and $\mathbf{b}_q \in \mathbb{R}^{1 \times q}$ is the bias of the output layer.
    - If it is a classification problem, we can use $\text{softmax}(\mathbf{O})$ to compute the probability distribution of the output categories.

Recurrent Neural Networks with Hidden States

- Assume that we have a minibatch of inputs $\mathbf{X}_t \in \mathbb{R}^{n \times d}$ at time step t.
    - In other words, for a minibatch of n sequence examples, each row of $\mathbf{X}_t$ corresponds to one example at time step t from the sequence.
- Denote by $\mathbf{H}_t \in \mathbb{R}^{n \times h}$ the hidden variable of time step t.

- Unlike the MLP, here we save the hidden variable $\mathbf{H}_{t-1}$ from the previous time step and introduce a new weight parameter $\mathbf{W}_{hh} \in \mathbb{R}^{h \times h}$ to describe how to use the hidden variable of the previous time step in the current time step.

- Specifically, the calculation of the hidden variable of the current time step is determined by the input of the current time step together with the hidden variable of the previous time step:

$$\mathbf{H}_t = \phi(\mathbf{X}_t \mathbf{W}_{xh} + \mathbf{H}_{t-1} \mathbf{W}_{hh} + \mathbf{b}_h).$$

- Compared to MLP adds one more term $\mathbf{H}_{t-1} \mathbf{W}_{hh}$

- The hidden variable $\mathbf{H}_t$ captures and retains the sequence's historical information up to their current time step.
    - Such a hidden variable is called a *hidden state*.
- For time step t, the output of the output layer is similar to the computation in the MLP:

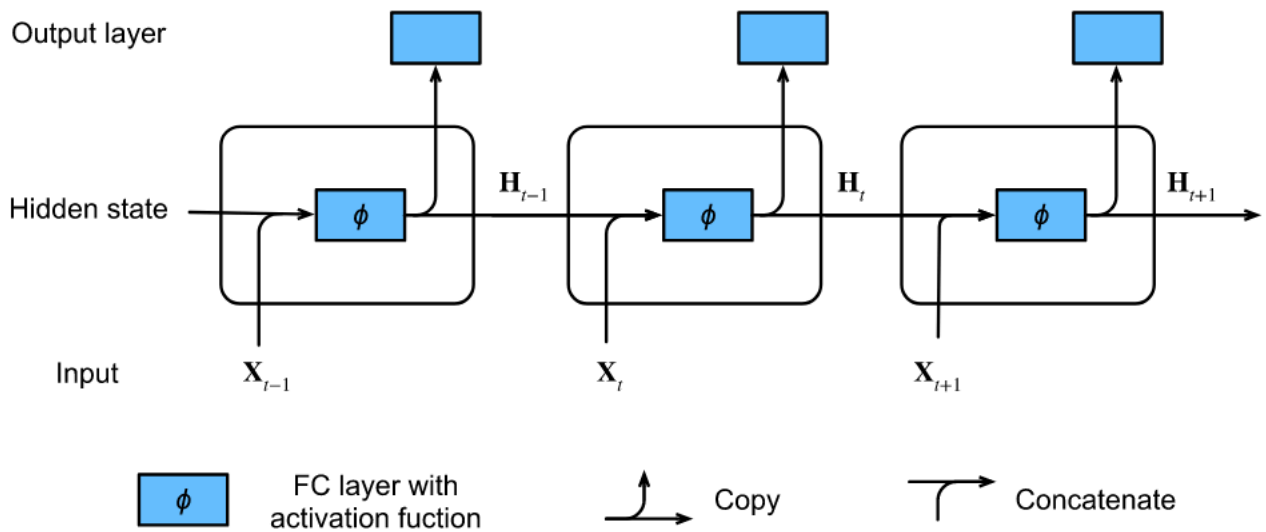
$$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}_{hq} + \mathbf{b}_q.$$

- RNN parameters include:
    - the weights $\mathbf{W}_{xh} \in \mathbb{R}^{d \times h}$, $\mathbf{W}_{hh} \in \mathbb{R}^{h \times h}$, and the bias $\mathbf{b}_h \in \mathbb{R}^{1 \times h}$ of the hidden layer,
    - the weights $\mathbf{W}_{hq} \in \mathbb{R}^{h \times q}$ and the bias $\mathbf{b}_q \in \mathbb{R}^{1 \times q}$ of the output layer.
- Even at different time steps, RNNs always use these model parameters.
    - The parameters of an RNN do not grow as the number of time steps increases.

- At any time step t, the computation of the hidden state $\mathbf{H}_t$ can be done as follows:

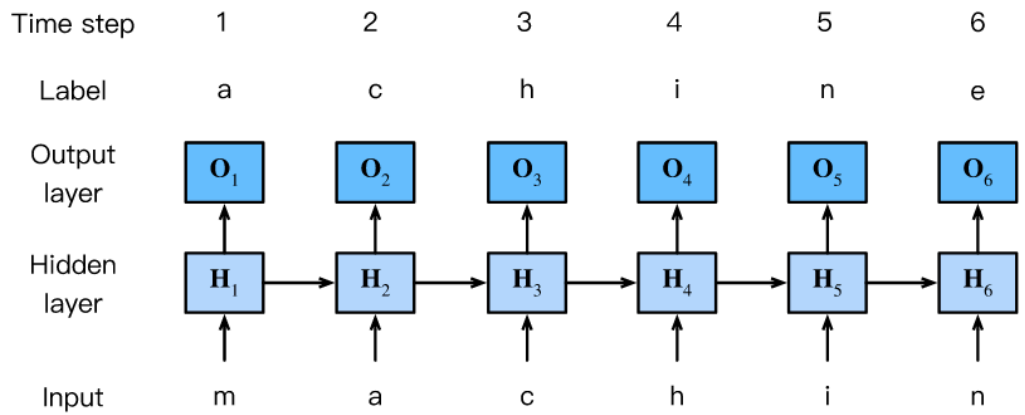
    1. concatenate the input $\mathbf{X}_t$ at the current time step t and the hidden state $\mathbf{H}_{t-1}$ at the previous time step $t-1$;
    2. concatenate the weights $\mathbf{W}_{xh}$ and $\mathbf{W}_{hh}$
    3. Feeding the concatenated input into a fully-connected layer with the concatenated weights (and the activation function ϕ).
    4. The output of such a fully-connected layer is the hidden state $\mathbf{H}_t$ of the current time step t.

- The figure below illustrates the computational logic of an RNN at three adjacent time steps.



RNN-based Character-Level Language Models

- For simplicity, we tokenize text into characters rather than words.
- The following figure demonstrates how an RNN predicts the next character based on the current and previous characters:



- During the training process, a softmax operation is applied on the output from the output layer for each time step, and cross-entropy loss is used to compute the error between the model output and the label.
- Due to the recurrent computation, the output of time step 3 $\mathbf{O}_3$, is determined by the text sequence "m", "a", and "c".

Summary

- A neural network that uses recurrent computation for hidden states is called a recurrent neural network (RNN).
- The hidden state of an RNN can capture historical information of the sequence up to the current time step.
- The number of RNN model parameters does not grow as the number of time steps increases.
- We can create character-level language models using an RNN.