HW3P1 Bootcamp

RNNs, GRUs, CTC, and Greedy/Beam Search

Spring 2025

Acknowledgement: Fall 2024 TAs

Logistics

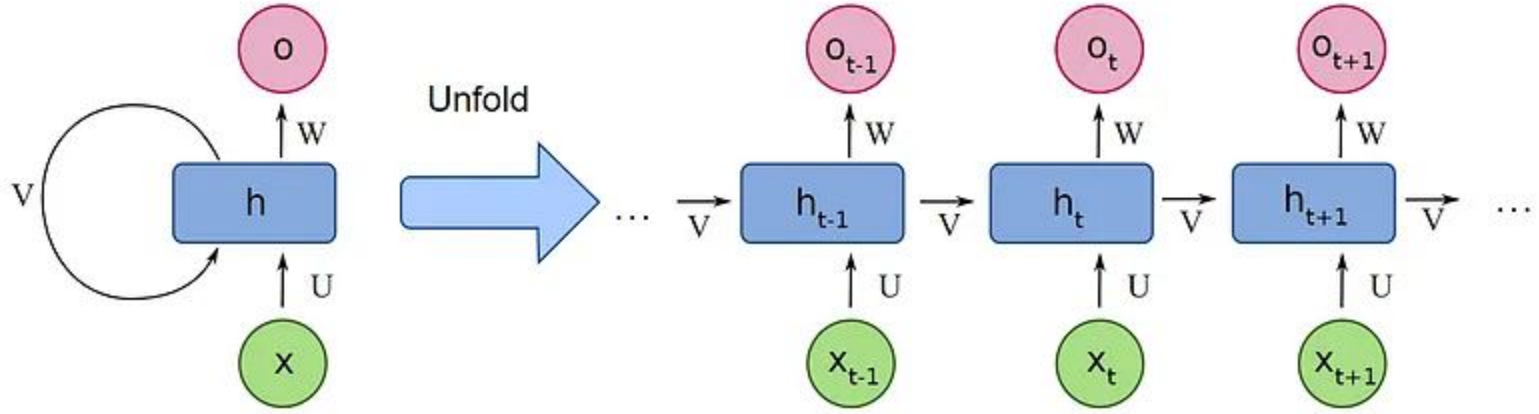
- Early Submission : **March 14th, 11:59pm**
- On-Time Submission: **March 28th, 11:59pm**

Two approaches: **Standard** and **Autograd**

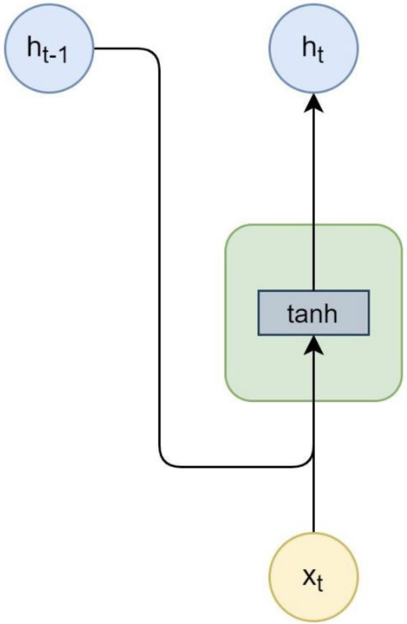
No late days can be used for Homework part 1s, please plan accordingly!

Structure of RNNs

A simple RNN that does seq-to-seq task with one RNN cell



RNN Cell Forward and Backward

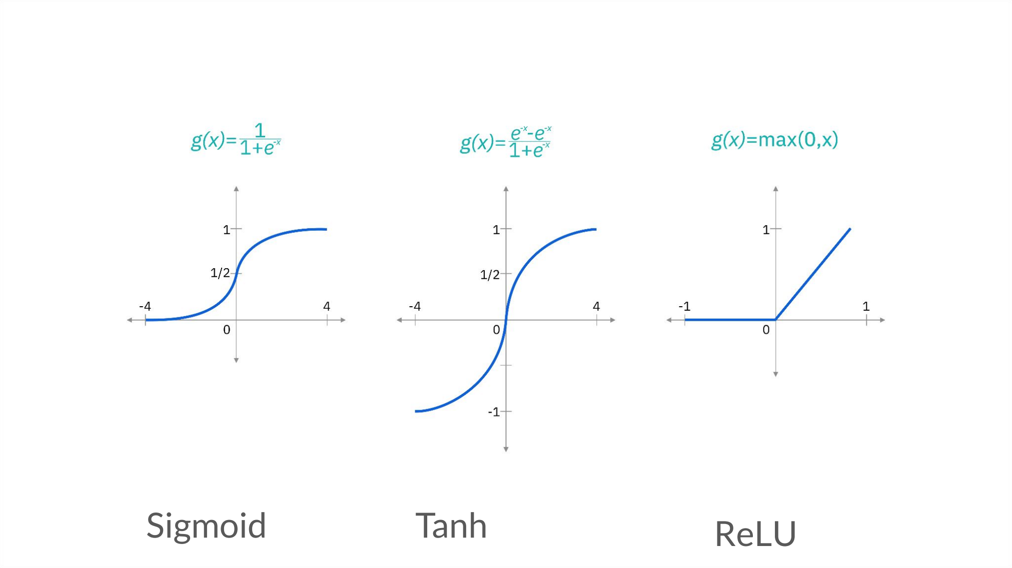


$$h_t = \tanh(W_{ih} x_t + b_{ih} + W_{hh} h_{t-1} + b_{hh})$$

Tip: very similar to **linear.py** in HW1P1.

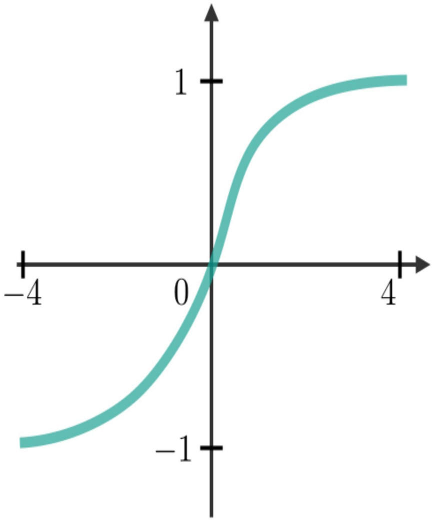
We are just applying a tanh function to linear transformations of x_t and h_{t-1}

Why Tanh though?



RNN Cell Forward and Backward
Why tanh?

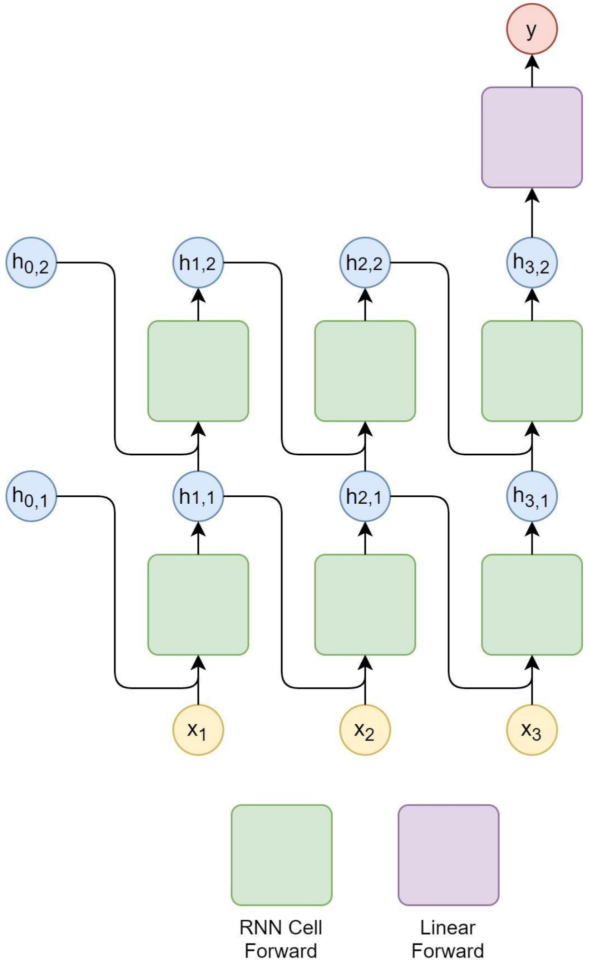
$$h_t = \boxed{tanh}(W_{ih}x_t + b_{ih} + W_{hh}h_{t-1} + b_{hh})$$



1. Non-linearity
2. Tanh is bounded; can mitigate exploding gradient problem

RNN Phoneme Classifier

- **Many-to-one** task
- Input sequence is passed through a few layers of **RNN cells**
- The final hidden state at the final timestamp is passed through a **linear layer** to give us the phoneme class



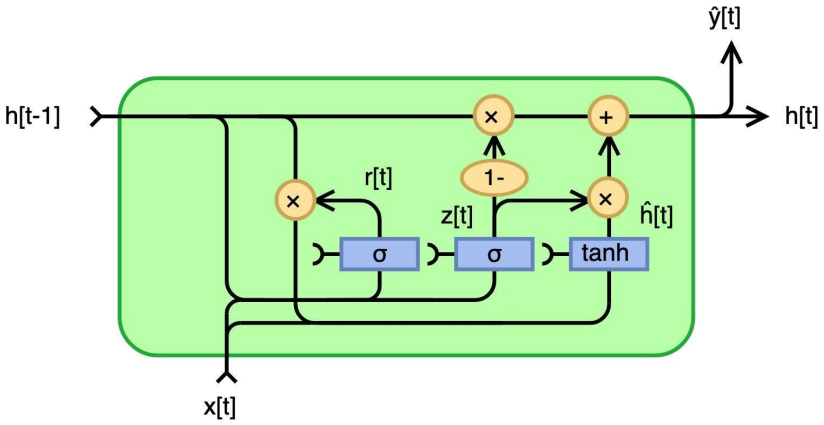
GRU Cell Forward and Backward

Reset Gate: $\mathbf{r}_t = \sigma(\mathbf{W}_{rx} \cdot \mathbf{x}_t + \mathbf{b}_{rx} + \mathbf{W}_{rh} \cdot \mathbf{h}_{t-1} + \mathbf{b}_{rh})$

Update Gate: $\mathbf{z}_t = \sigma(\mathbf{W}_{zx} \cdot \mathbf{x}_t + \mathbf{b}_{zx} + \mathbf{W}_{zh} \cdot \mathbf{h}_{t-1} + \mathbf{b}_{zh})$

Memory Content: $\mathbf{n}_t = tanh(\mathbf{W}_{nx} \cdot \mathbf{x}_t + \mathbf{b}_{nx} + \mathbf{r}_t \odot (\mathbf{W}_{nh} \cdot \mathbf{h}_{t-1} + \mathbf{b}_{nh}))$

Hidden State: $\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{n}_t + \mathbf{z}_t \odot \mathbf{h}_{t-1}$



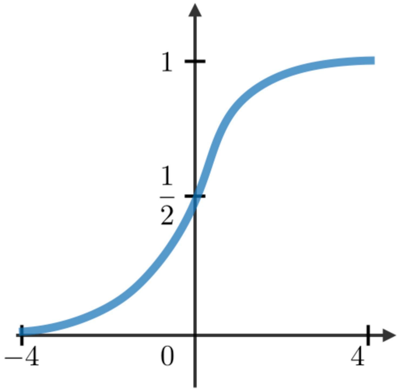
GRU Cell Forward and Backward contin.

Reset Gate: $\mathbf{r}_t = \boxed{\sigma}(\mathbf{W}_{rx} \cdot \mathbf{x}_t + \mathbf{b}_{rx} + \mathbf{W}_{rh} \cdot \mathbf{h}_{t-1} + \mathbf{b}_{rh})$

Update Gate: $\mathbf{z}_t = \boxed{\sigma}(\mathbf{W}_{zx} \cdot \mathbf{x}_t + \mathbf{b}_{zx} + \mathbf{W}_{zh} \cdot \mathbf{h}_{t-1} + \mathbf{b}_{zh})$

Memory Content: $\mathbf{n}_t = tanh(\mathbf{W}_{nx} \cdot \mathbf{x}_t + \mathbf{b}_{nx} + \mathbf{r}_t \odot (\mathbf{W}_{nh} \cdot \mathbf{h}_{t-1} + \mathbf{b}_{nh}))$

Hidden State: $\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{n}_t + \mathbf{z}_t \odot \mathbf{h}_{t-1}$



Why sigmoid?

- Sigmoid is limited to the values 0 to 1 → This can describe how much info to pass
  - Close to 0: we want to “forget”
  - Close to 1: we want to “remember”

GRU Cell Forward and Backward contin.

Reset Gate: $$\mathbf{r}_t = \sigma(\mathbf{W}_{rx} \cdot \mathbf{x}_t + \mathbf{b}_{rx} + \mathbf{W}_{rh} \cdot \mathbf{h}_{t-1} + \mathbf{b}_{rh})$$

Update Gate: $$\mathbf{z}_t = \sigma(\mathbf{W}_{zx} \cdot \mathbf{x}_t + \mathbf{b}_{zx} + \mathbf{W}_{zh} \cdot \mathbf{h}_{t-1} + \mathbf{b}_{zh})$$

Memory Content: $$\mathbf{n}_t = tanh(\mathbf{W}_{nx} \cdot \mathbf{x}_t + \mathbf{b}_{nx} + \boxed{\mathbf{r}_t \odot (\mathbf{W}_{nh} \cdot \mathbf{h}_{t-1} + \mathbf{b}_{nh})})$$

Hidden State: $$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{n}_t + \mathbf{z}_t \odot \mathbf{h}_{t-1}$$

We do an element-wise product with a linear transformation of the previous hidden-state. We are combining the transformation of the inputs from candidate space with the information we are retaining from previous hidden state.

GRU Cell Forward and Backward contin.

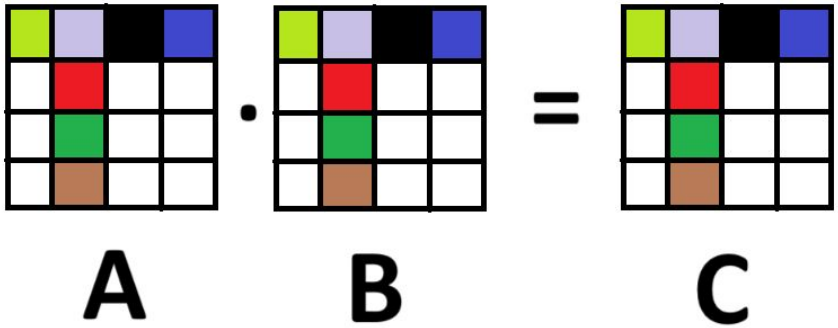
- GRU backward be the longest question in HW3P1
- Tips:
    - All intermediate **dWs** and **dbs** should be correct to make sure that your **dx** and **dh** are correct
    - Useful resource: [How to compute a derivative]
    - Break down complicated equations into unary/binary operations
    - e.g. $f(x) = \tanh(r \odot (Wx+b))$, we want to decompose it into:
        - $Z1 = Wx + b$
        - $Z2 = r \odot (Wx+b)$
        - $f(x) = \tanh(Z2)$
        - Derivative for each step would be easy and lastly we apply chain rule to get our $f'(x)$

Chain rule through element-wise multiplication

Assume that the shape of derivative wrt a matrix is the same as that of the matrix.

Let C = A $\odot$ B (element-wise)

- This means A, B, and C have the same shape
- Only elements of the same position are related to each other → derivatives flow only position-wise.
- Therefore, dLdA = dLdC $\odot$ B and dLdB = dLdC $\odot$ A



Chain rule through matrix multiplication

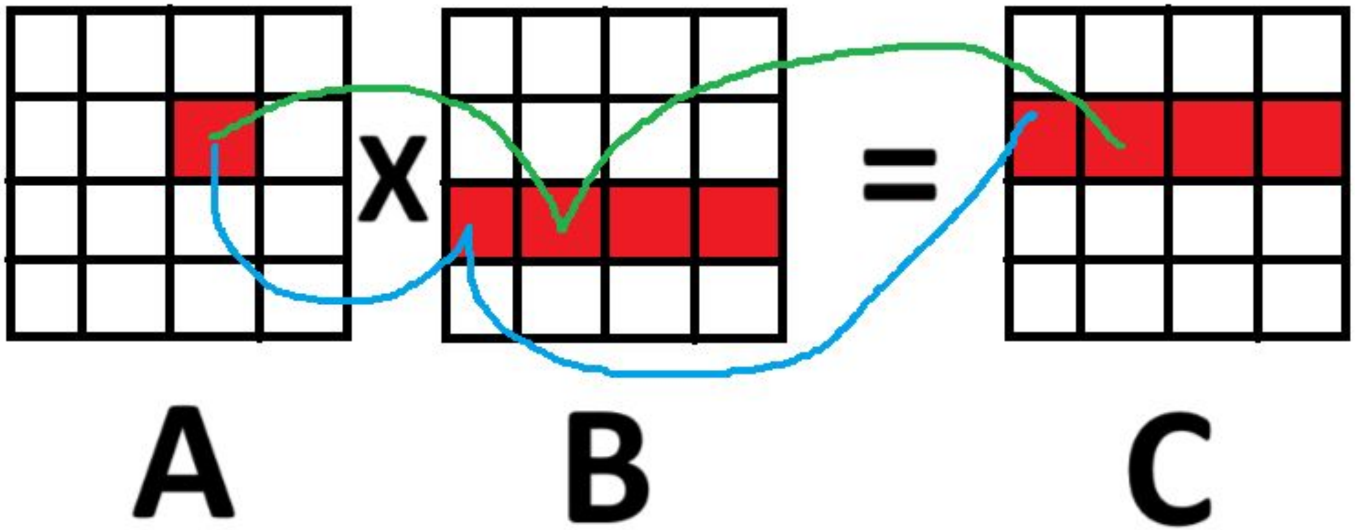
Assume that the shape of derivative wrt a matrix is the same as that of the matrix.

Let C = AB (matrix multiplication). The shapes of A, B, C are a x b, b x c, and c x a respectively.

- Think about which all elements of C does A(i, j) influence.
- It influences all elements of C in row i through multiplication with the j-th element in every column of B.
- So, dLdA(i, j) = sum[k=1 to c] dLdC(i, k)B(j, k)
- Doing this for every element gives dLdA = dLdC X B.T (matrix multiplication)

DON'T JUST MATCH SHAPES. UNDERSTAND HOW VALUES MATCH INSTEAD. SHAPES WILL FOLLOW.

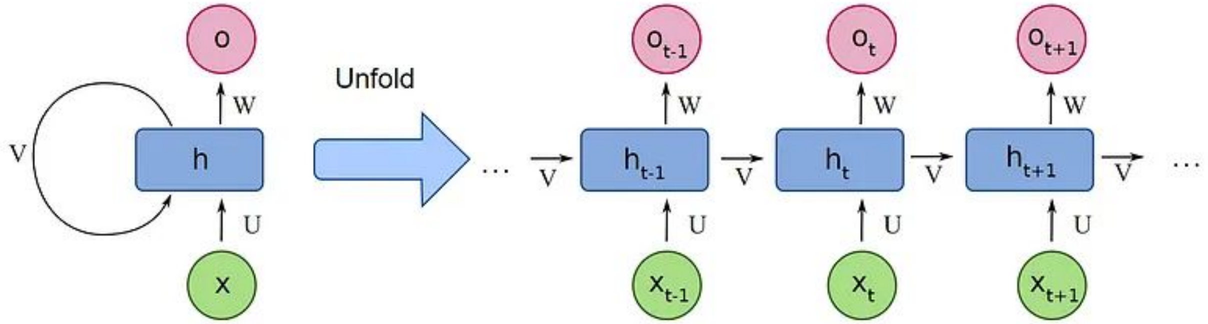
Chain rule through matrix multiplication (contin.)

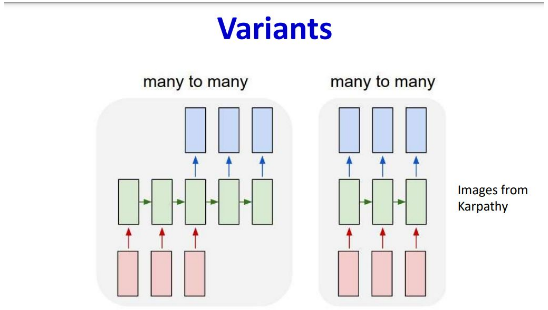


GRU Inference - Character Predictor

Many-to-many task, the model is supposed to have an output for each timestamp.

Different from RNN Phoneme classifier, here we need to pass the hidden state at **each** timestamp to a linear layer to predict the character at each t, instead of just the previous timestamp's hidden state.





- 1: *Delayed* sequence to sequence, e.g. machine translation
- 2: Sequence to sequence, e.g. stock problem, label prediction
- Etc…

Connectionist Temporal Classification (CTC)

1) Why CTC?
2) How does it work?
3) Implementation in HW3 P1

Why CTC?

- CTC is used to calculate loss when our input sequences and output labels have **different lengths and no fixed alignment.**
- This method allows models to learn without needing pre-aligned data, which is crucial for applications like **speech recognition.**

Input (from RNN over time) → "hhheellllooo" (length = 10)

Target sequence → "hello"

Implementation [CTC/CTC.py]

- CTC
  - extend_target_with_blank()
  - get_forward_probs()
  - get_backward_probs()
  - get_posterior_probs()
- CTCLoss

1. Extend target with blank

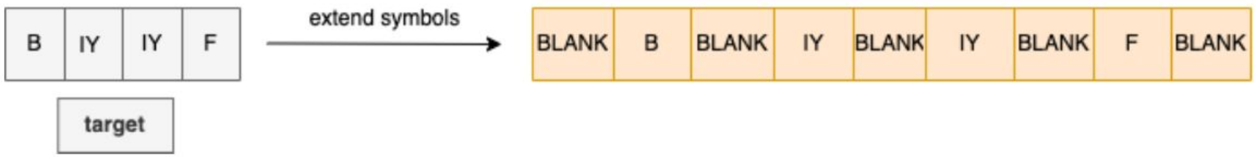


Figure 13: Extend symbols

1. Extend target with blank

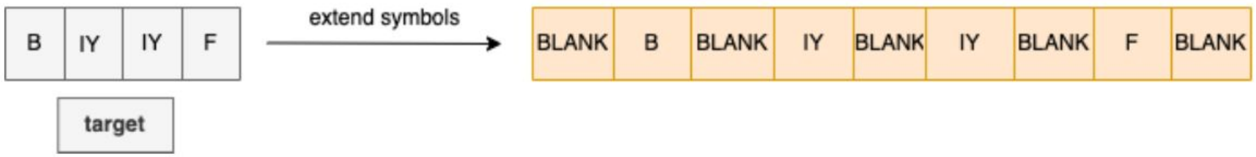


Figure 13: Extend symbols

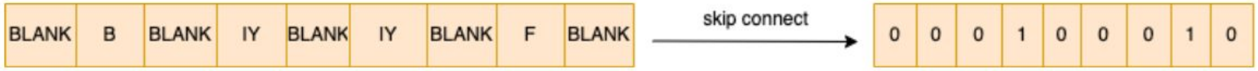
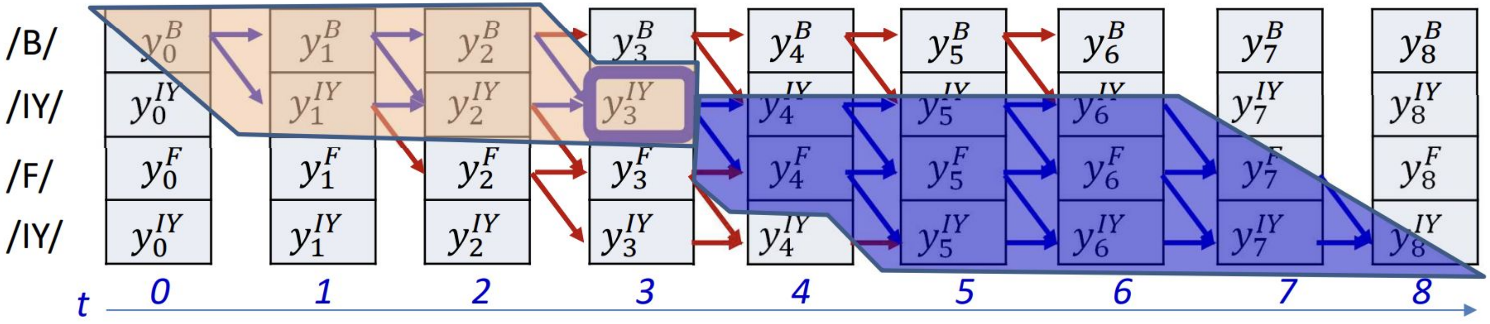


Figure 14: Skip connections

2. Forward Probabilities

$$\alpha(t,r) = P(S_0 ... S_r, s_t = S_r | X) = \sum_{q:S_q \in pred(S_r)} \alpha(t-1,q) y_t^{S_r}$$



2. Forward Probabilities

$$\alpha(t,r) = P(S_0 ... S_r, s_t = S_r | X) = \sum_{q:S_q \in pred(S_r)} \alpha(t-1,q) y_t^{S_r}$$

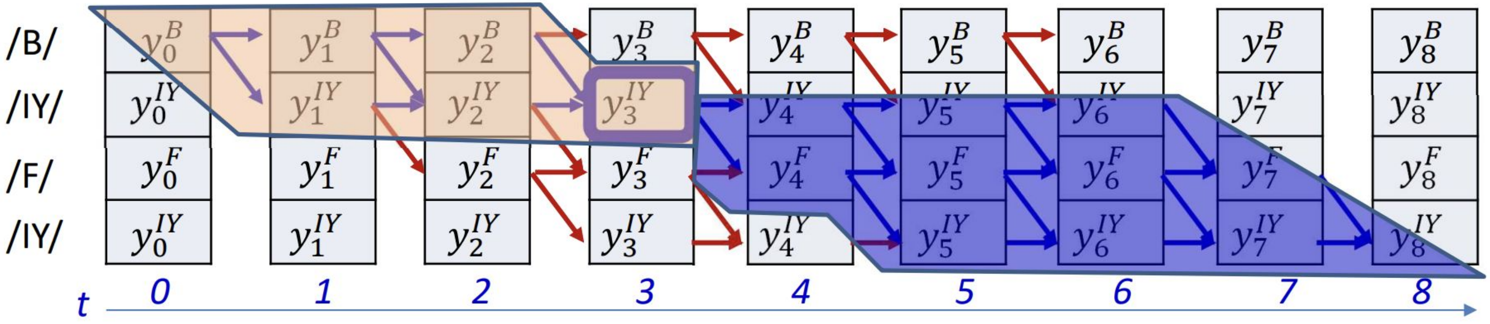
alpha	t = 1	t = 2	t = 3	t = 4	t = 5	t = 6
BLANK	0	0	0	0	0	0
B	.1	.05	.045	.0135	.00945	.000945
BLANK	0	.04	.072	.0234	.02214	0
IY	0	.06	0	.0468	.06696	.01971
BLANK	0	0	.048	.0096	.03384	0
IY	0	0	0	.0192	.02304	.011376
BLANK	0	0	0	0	.01152	0
F	0	0	0	0	.01728	.015552
BLANK	0	0	0	0	0	0
input	x1	x2	x3	x4	x5	x6

Figure 15: Forward Algorithm

3. Backward Algorithm

- Computing probabilities from right to left
- **β(t, r)** represents probability of generating the rest of the sequence starting from time t

$$\beta(t,r) = P(s_{t+1} \in succ(S_r), succ(S_r), ..., S_{K-1}|X) = \sum_{q:S_q \in succ(S_r)} \beta(t+1,q) y_{t+1}^{S_q}$$



4. CTC Posterior Probability

- Represents probability of being in state **r** at time **t** given input sequence.

$$\gamma(t,r) = P(s_t = S_r | S, X) = \frac{\alpha(t,r)\beta(t,r)}{\sum_{r'} \alpha(t,r)\beta(t,r)}$$

5. CTC Loss Computation

- Once we know the posterior probability, we can plug it in to calculate the loss, which will ultimately be used to train your network.

$$\gamma(t,r) = P(s_t = S_r | S, X) = \frac{\alpha(t,r)\beta(t,r)}{\sum_{r'} \alpha(t,r)\beta(t,r)}$$

$\Rightarrow$

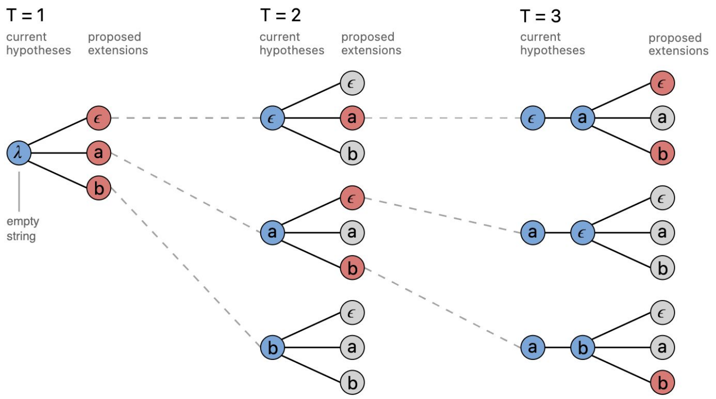
$$DIV = -\sum_t \sum_{s \in S_0 \dots S_{K-1}} P(s_t = s|\mathbf{S},\mathbf{X}) \log Y(t, s_t = s)$$

$$DIV = -\sum_t \sum_r \gamma(t,r) \log y_t^{S(r)}$$

$$\frac{dDIV}{dy_t^l} = -\frac{1}{y_t^l} \sum_{r\,:S(r)=l} \gamma(t,r)$$

CTC Decoding [CTC/CTCDecoding.py]

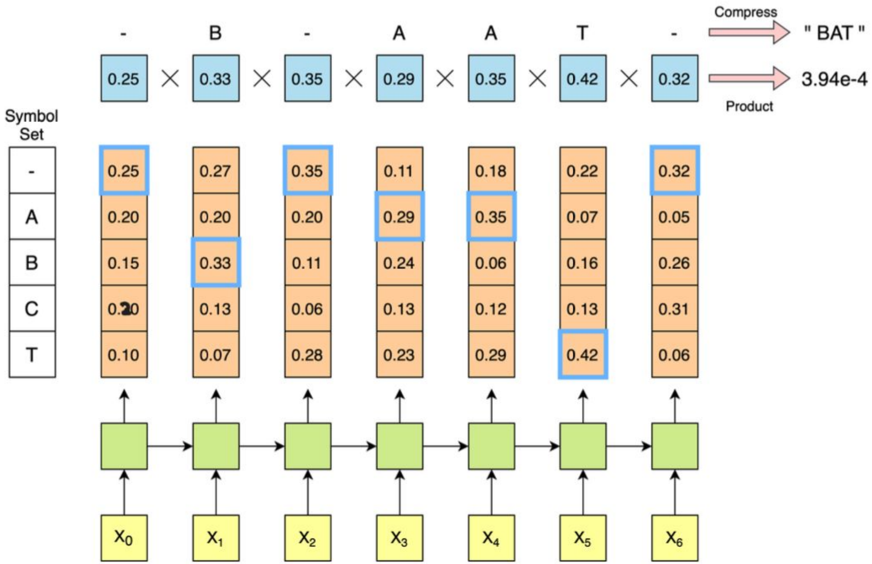
- ❏ CTC loss gives us a **probability distribution over possible alignments**, but it does **not** directly produce the final output sequence.
- ❏ Instead, we need a decoding strategy to transform the model's output into meaningful text. There are two primary methods for this: **Greedy Search** and **Beam Search.**



A standard beam search algorithm with an alphabet of $\{\epsilon, a, b\}$ and a beam size of three.

I. Greedy Search

- Picks highest probability token at each timestep.
- Collapses repeated tokens and removes blanks.



Pseudocode provided in the write-up.

II. Beam Search

Efficient Beam Search:

```
Input: SymbolSets, y_probs, BeamWidth
Output: BestPath, MergedPathScores

    0. Initialize:
        1. BestPaths with a blank symbol path with a score of 1.0.
        2. TempBestPaths as an empty dictionary.
    1. For each timestep in y_probs:
        1. Extract the current symbol probabilities.
        2. For each path, score in BestPaths limited by BeamWidth:
            1. For each new symbol in the current symbol probabilities:
                1. Based on the last symbol of the path, determine the new path.
                2. Update the score for the new path in TempBestPaths.

        3. Update BestPaths with TempBestPaths.
        4. Clear TempBestPaths.

    2. Initialize MergedPathScores as an empty dictionary.

    3. For each path, score in BestPaths:
        1. Remove the ending blank symbol from the path.
        2. Update the score for the translated path in MergedPathScores.
        3. Update the BestPath and BestScore if the score is better.

    4. Return BestPath and MergedPathScores.
```

Remember, when extending a path with a new symbol, you'll encounter three scenarios:

1. The new symbol is the same as the last symbol on the path.
2. The last symbol of the path is blank.
3. The last symbol of the path is different from the new symbol and is not blank.

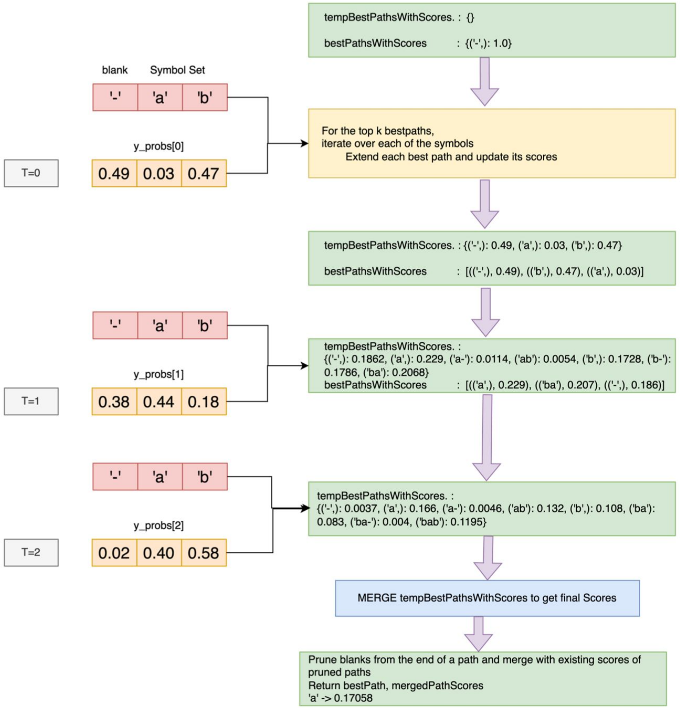
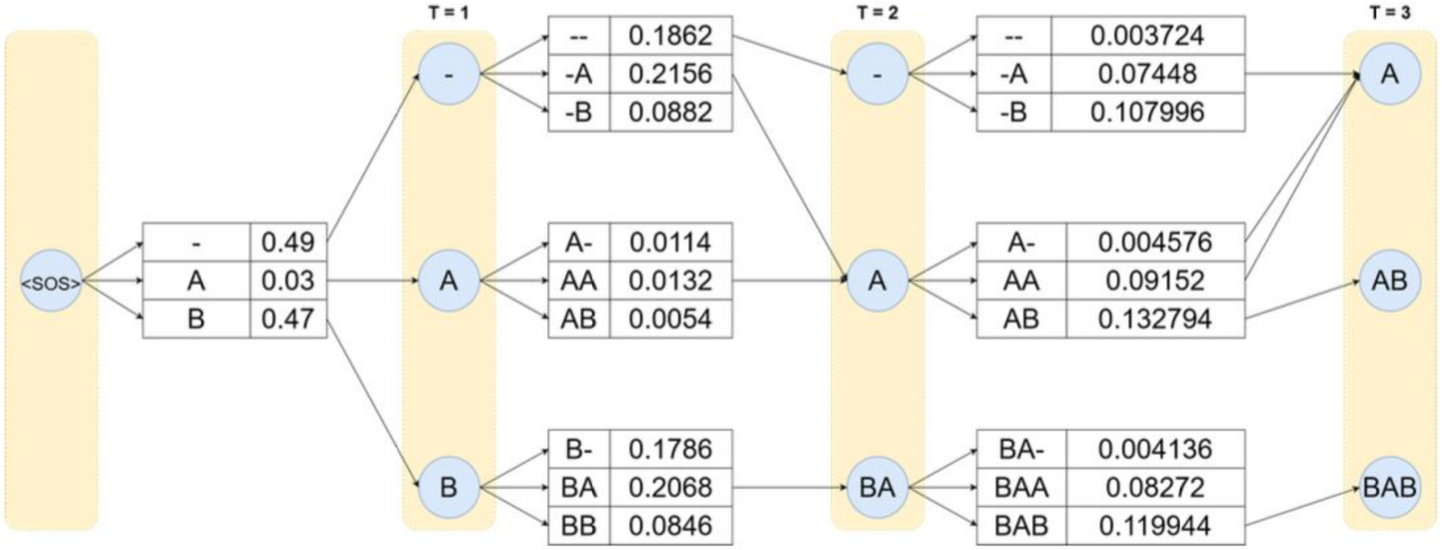


Figure 20: Efficient Beam Search procedure

II. Beam Search

❏



Pseudocode can be found in the write-up and in future lectures slides.

Thank you!

Q&A