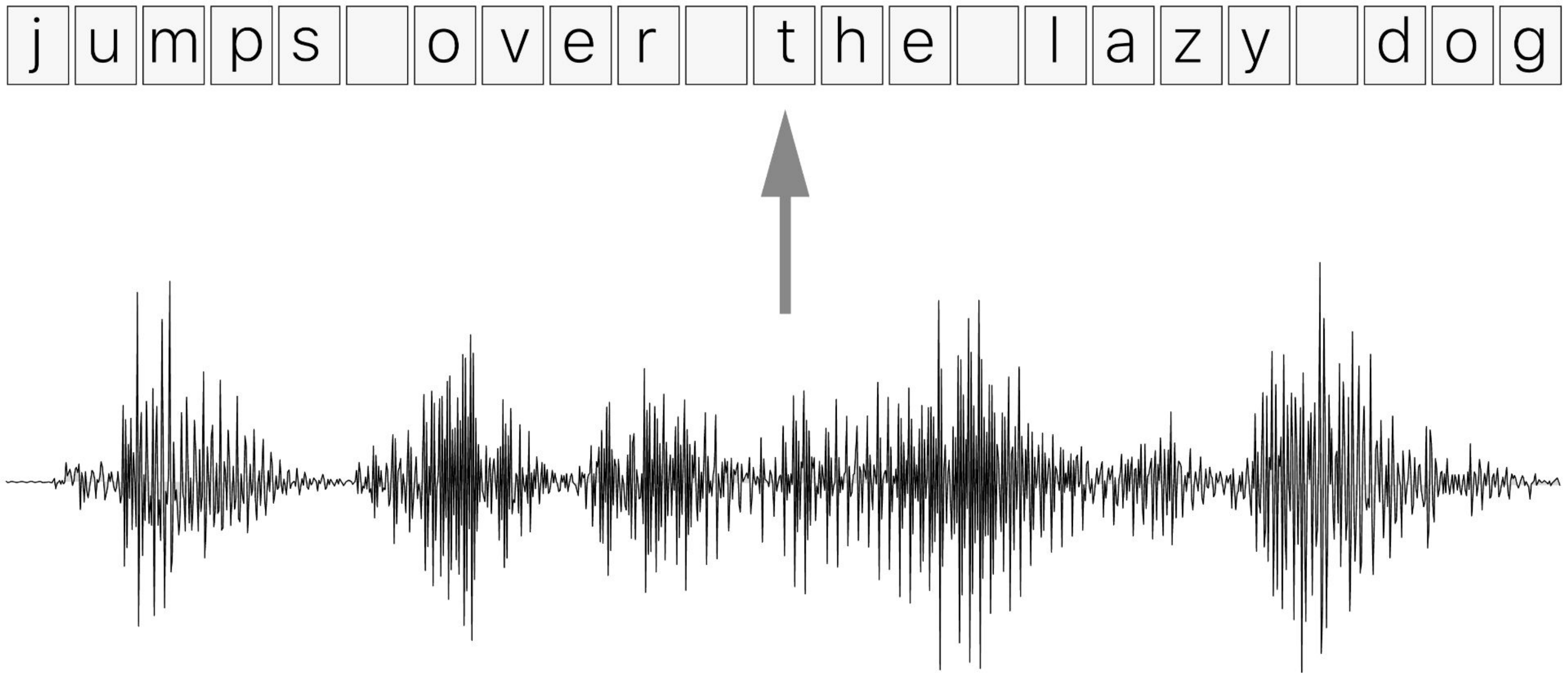
CTC



Speech recognition: The input can be a spectrogram or some other frequency based feature extractor.

Connectionist Temporal Classification (CTC): is a way to get **around not knowing the alignment** between the input and the output. As we'll see, it's especially well suited to applications in speech recognition.

Mapping input sequences such as audio, to corresponding output sequences

$$X = [\, x1 \,,\, x2 \;\,, \ldots, \;\, xT \,]$$
$$Y = [\, y1 \,,\, y2 \;\,, \ldots, \;\, yU \,]$$

There are challenges which get in the way of us using simpler supervised learning:

- Both X and Y can vary in length.
- The ratio of the lengths of X and Y can vary.
- We don’t have an accurate alignment (correspondence of the elements) of X and Y.

Overcomes these challenges

The **CTC algorithm overcomes** these challenges.

For a given X **it gives us an output distribution over all possible Y’s.**

We can use this distribution either to *infer* a likely output or to assess the *probability* of a given output.

Loss Function

We'd like to train our model to **maximize** the probability it assigns to the right answer.

We'll need to efficiently compute the conditional probability $p(Y|X)$.

The function $p(Y|X)$ should also be differentiable, so we can use gradient descent.

Inference

Naturally, after we’ve trained the model,
we want to use it to infer a likely Y given an X. This means solving

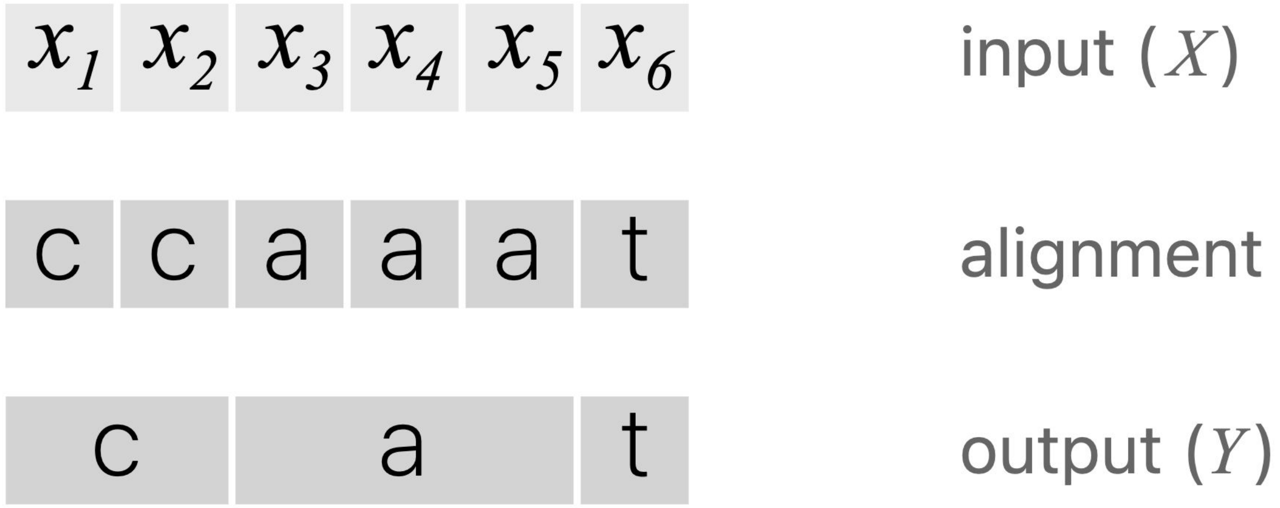
$$Y^* = \underset{Y}{\operatorname{argmax}} \; p(Y \mid X).$$

Algorithm

The CTC algorithm is *alignment-free*
it doesn’t require an alignment between the input and the output.
However, to get the probability of an output given an input, CTC works by summing over the probability of all possible **alignments** between the two.

Alignment

First consider a naive approach. Let’s use an example. Assume the input has length six and Y= [c, a, t]. One way to align X and Y is to assign an output character to each input step and collapse repeats.



This approach has two **problems.**

- It doesn’t make sense to force every input step to align to some output.
- We have no way to produce outputs with **multiple characters in a row.** Consider the alignment [h, h, e, l, l, l, o]. Collapsing repeats will produce “helo” instead of “hello”.

Blank Token

$$\varepsilon$$

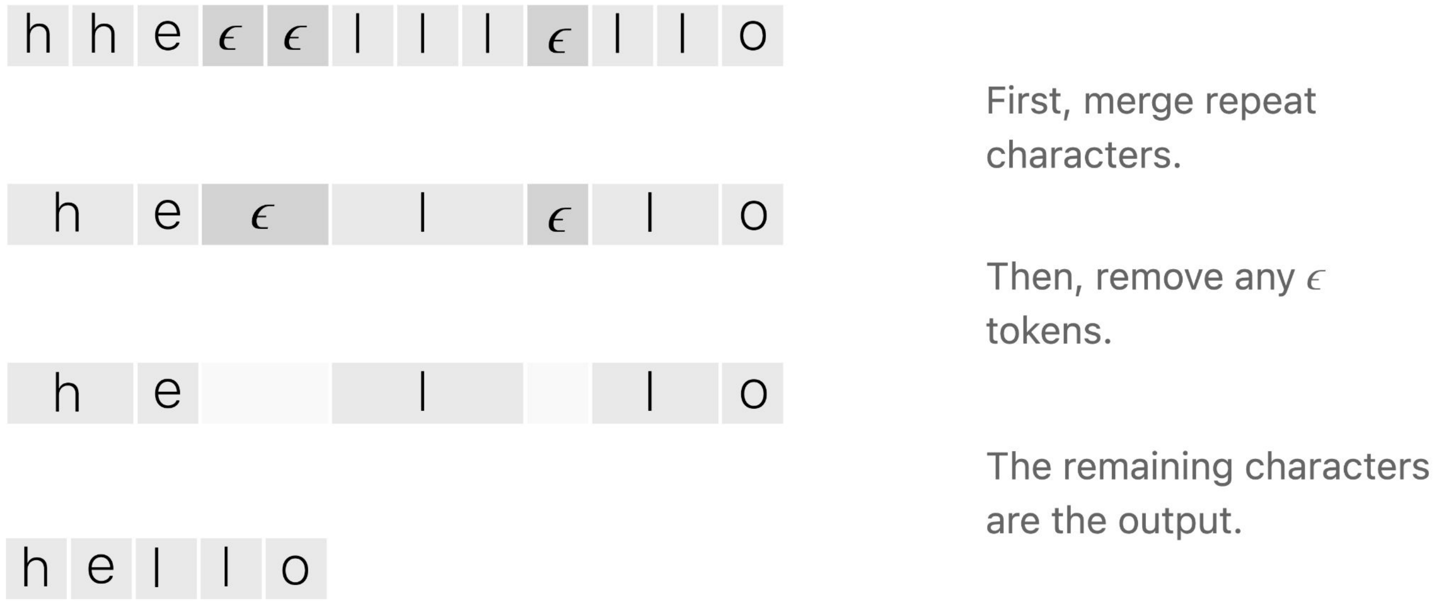
To get around these problems, CTC introduces a new token to the set of allowed outputs.

This new token is sometimes called the ***blank* token.** We'll refer to it here as ϵ**.**

The ϵ token doesn't correspond to anything and is simply removed from the output.

The alignments allowed by CTC are the **same length as the input.**

We allow **any alignment which maps to Y** after merging repeats and removing ϵ tokens

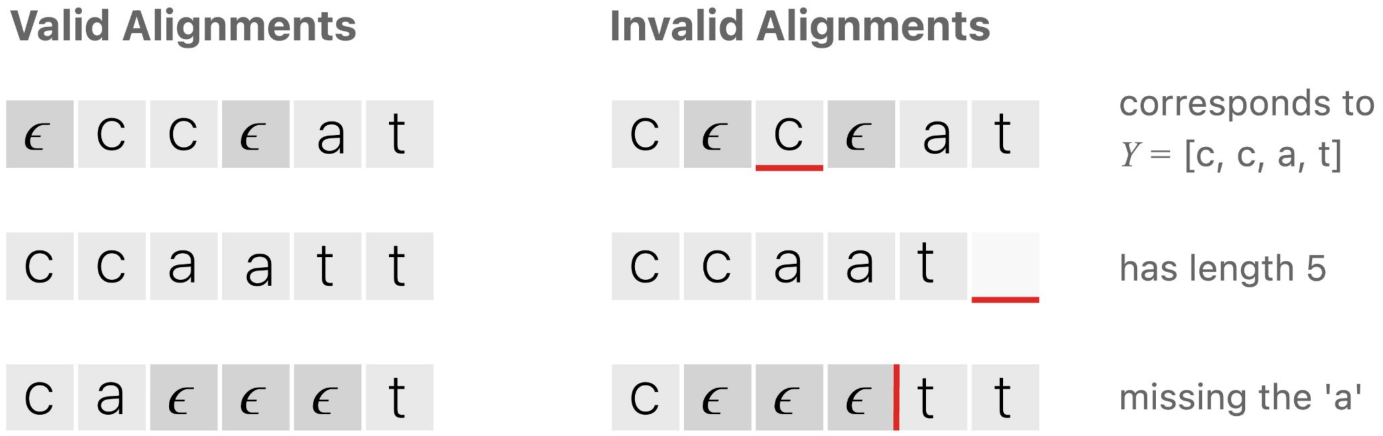

h h e ϵ ϵ l l l ϵ l l o
First, merge repeat characters.
h e ϵ l ϵ l o
Then, remove any ϵ tokens.
h e l l o
The remaining characters are the output.
h e l l o

If Y has two of the same character in a row, then a valid alignment **must have an ϵ** between them.

With this rule in place, we can differentiate between alignments which collapse to

“hello” and those which collapse to “helo”.

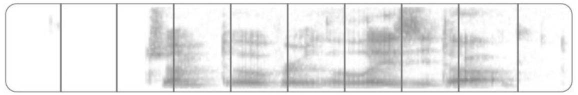
Let’s go back to the output [c, a, t] with an input of length six.
Here are a few more examples of valid and invalid alignments.



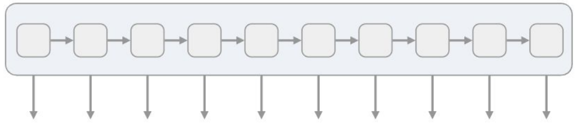
First, the allowed alignments between X and Y are monotonic. If we advance to the next input, we can keep the corresponding output the same or advance to the next one.

A second property is that the alignment of X to Y is many-to-one. One or more input elements can align to a single output element but not vice-versa.

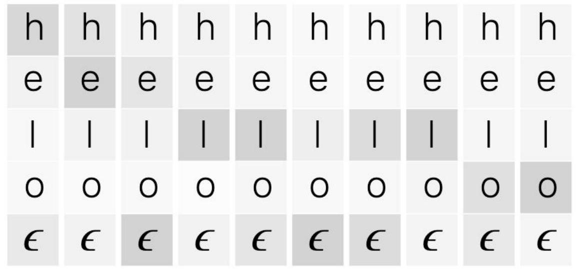
This implies a third property: the length of Y cannot be greater than the length of X.



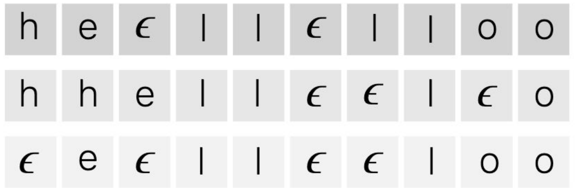
We start with an input sequence, like a spectrogram of audio.



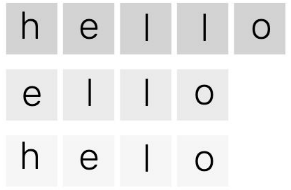
The input is fed into an RNN, for example.


h h h h h h h h h h
e e e e e e e e e e
l l l l l l l l l l
o o o o o o o o o o
ϵ ϵ ϵ ϵ ϵ ϵ ϵ ϵ ϵ ϵ

The network gives $p_t(a \mid X)$, a distribution over the outputs {h, e, l, o, ϵ} for each input step.


h e ϵ l l ϵ l l o o
h h e l l ϵ ϵ l ϵ o
ϵ e ϵ l l ϵ ϵ l o o

With the per time-step output distribution, we compute the probability of different sequences



By marginalizing over alignments, we get a distribution over outputs

$$p(Y \mid X) \quad = \quad \sum_{A \in \mathcal{A}_{X,Y}} \quad \prod_{t=1}^{T} p_t(a_t \mid X)$$

The CTC conditional **probability**

marginalizes over the set of valid alignments

computing the **probability** for a single alignment step-by-step.

CTC Score

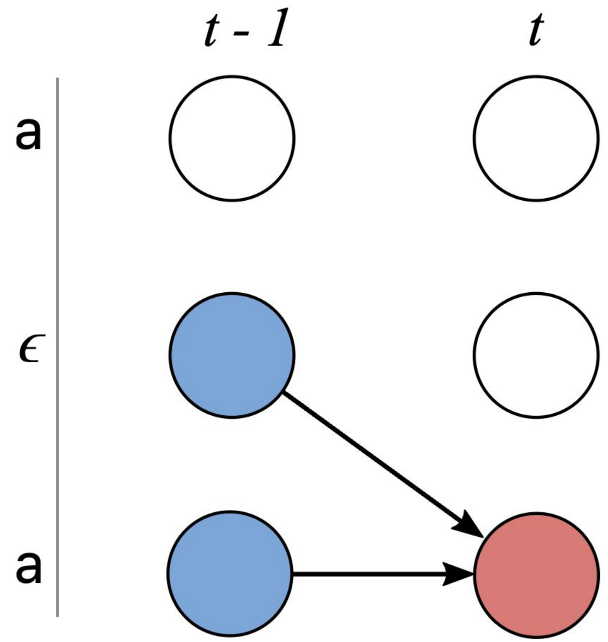
Since we can have an ϵ before or after any token in Y, it's easier to describe the algorithm using a sequence which includes them. We'll work with the sequence

$$Z \quad = \quad [\epsilon,\ y_1,\ \epsilon,\ y_2,\ \ldots,\ \epsilon,\ y_U,\ \epsilon]$$

which is Y with an ϵ at the beginning, end, and between every character.

Let's let α be the score of the merged alignments at a given node. More precisely, $\alpha_{s,t}$ is the CTC score of the subsequence $Z_{1:s}$ after t input steps. As we'll see, we'll compute the final CTC score, $P(Y \mid X)$, from the α's at the last time-step. As long as we know the values of α at the previous time-step, we can compute $\alpha_{s,t}$. There are two cases.

Case 1



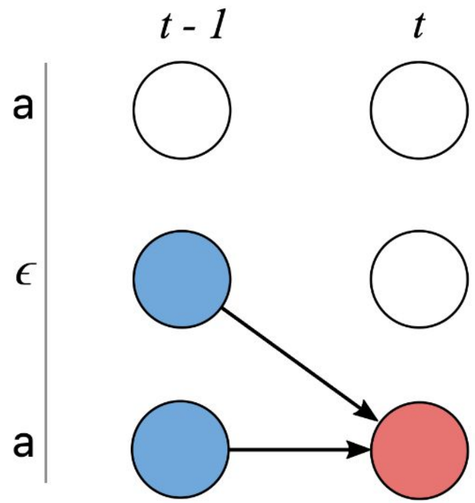
$$\alpha_{s,t} = (\alpha_{s-1,t-1} + \alpha_{s,t-1}) \cdot p_t(z_s \mid X)$$

The CTC probability of the two valid subsequences after $t-1$ input steps.

The probability of the current character at input step t.

Case 1:

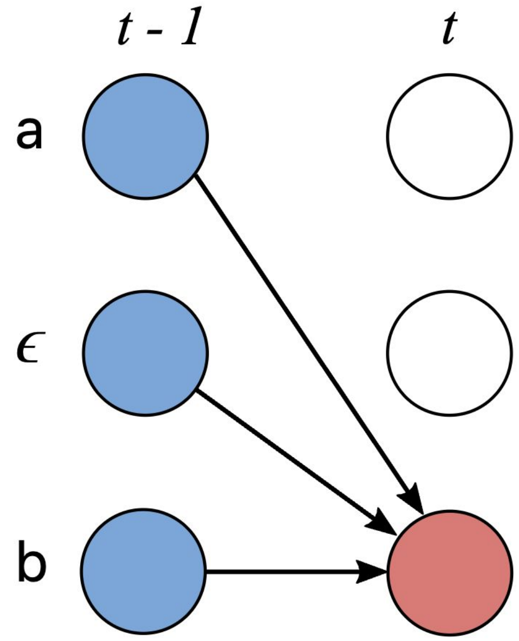
In this case, we can't jump over z_{s-1}, the previous token in Z. The first reason is that the previous token can be an element of Y, and we can't skip elements of Y. Since every element of Y in Z is followed by an ϵ, we can identify this when $z_s = \epsilon$. The second reason is that we must have an ϵ between repeat characters in Y. We can identify this when $z_s = z_{s-2}$.



To ensure we don't skip z_{s-1}, we can either be there at the previous time-step or have already passed through at some earlier time-step. As a result there are two positions we can transition from.

$$\alpha_{s,t} = \underbrace{(\alpha_{s-1,t-1} + \alpha_{s,t-1})}_{\text{The CTC probability of the two valid subsequences after } t-1 \text{ input steps.}} \cdot \underbrace{p_t(z_s \mid X)}_{\text{The probability of the current character at input step } t.}$$

Case 2



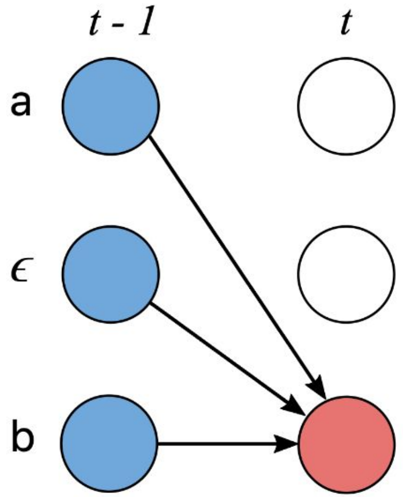
$$\alpha_{s,t} = (\alpha_{s-2,t-1} + \alpha_{s-1,t-1} + \alpha_{s,t-1}) \cdot p_t(z_s \mid X)$$

The CTC probability of the three valid subsequences after $t-1$ input steps.

The probability of the current character at input step t.

Case 2:

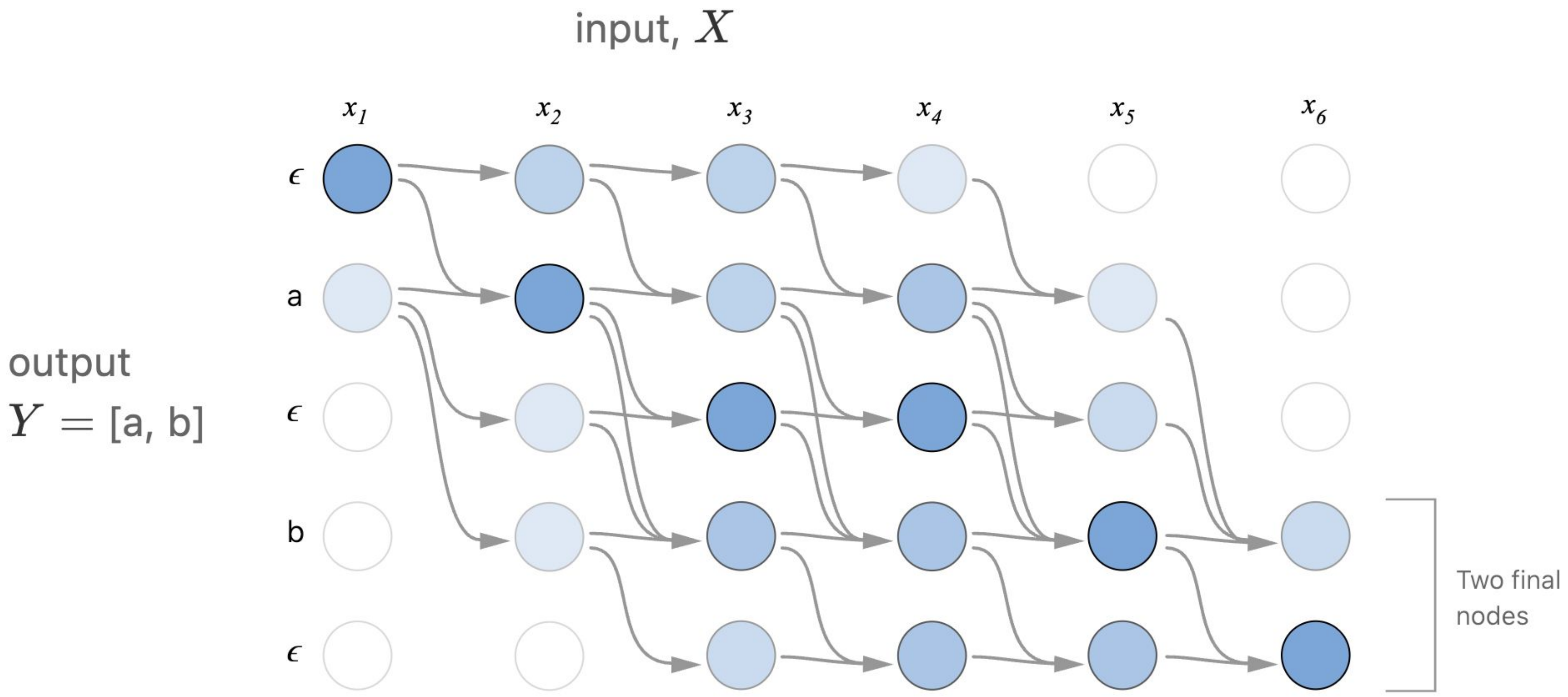
In the second case, we're allowed to skip the previous token in Z. We have this case whenever z_{s-1} is an ϵ between unique characters. As a result there are three positions we could have come from at the previous step.



$$\alpha_{s,t} \quad = \quad \underbrace{(\alpha_{s-2,t-1} + \alpha_{s-1,t-1} + \alpha_{s,t-1})} \quad \cdot \quad \underbrace{p_t(z_s \mid X)}$$

The CTC probability of the three valid subsequences after $t-1$ input steps.

The probability of the current character at input step t.



Node (s, t) in the diagram represents $\alpha_{s,t}$ – the CTC score of the subsequence $Z_{1:s}$ after t input steps.

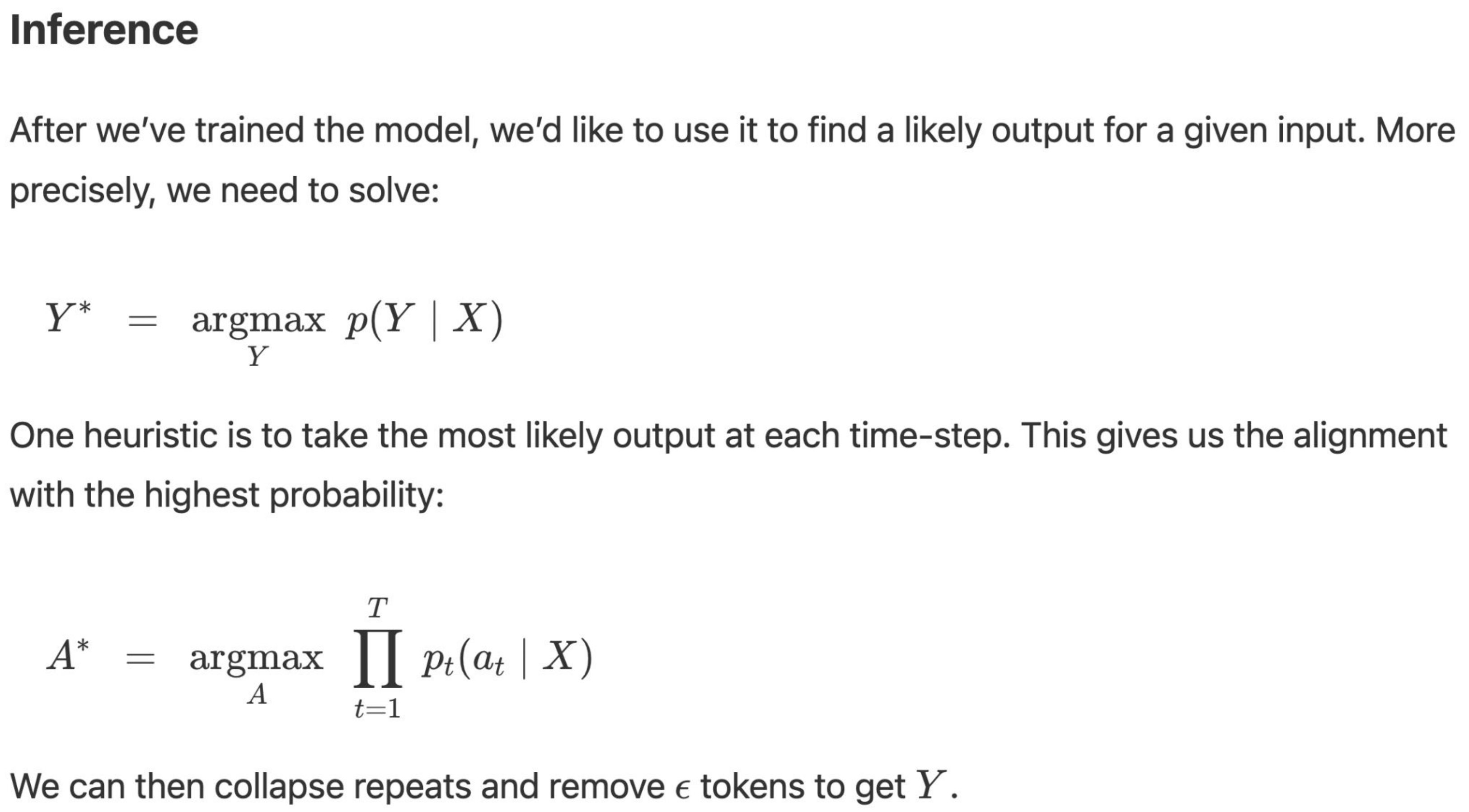
CTC Loss Function

For a training set $\mathcal{D}$, the model's parameters are tuned to minimize the negative log-likelihood

$$\sum_{(X,Y)\in\mathcal{D}} -\log\ p(Y \mid X)$$

instead of maximizing the likelihood directly.

Inference



Inference

After we've trained the model, we'd like to use it to find a likely output for a given input. More precisely, we need to solve:

$$Y^* \quad = \quad \underset{Y}{\operatorname{argmax}} \; p(Y \mid X)$$

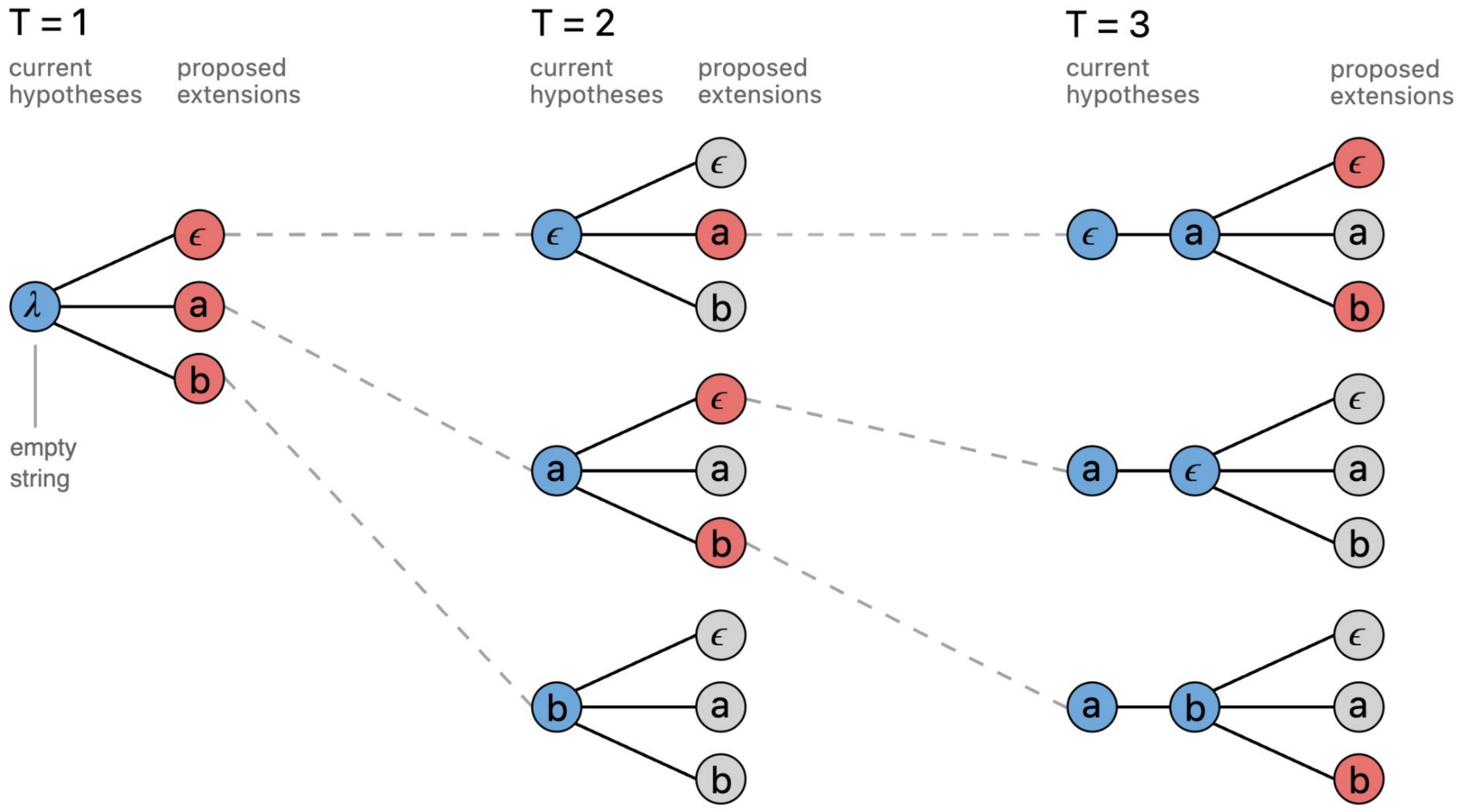
One heuristic is to take the most likely output at each time-step. This gives us the alignment with the highest probability:

$$A^* \quad = \quad \underset{A}{\operatorname{argmax}} \; \prod_{t=1}^{T} p_t(a_t \mid X)$$

We can then collapse repeats and remove ϵ tokens to get Y.

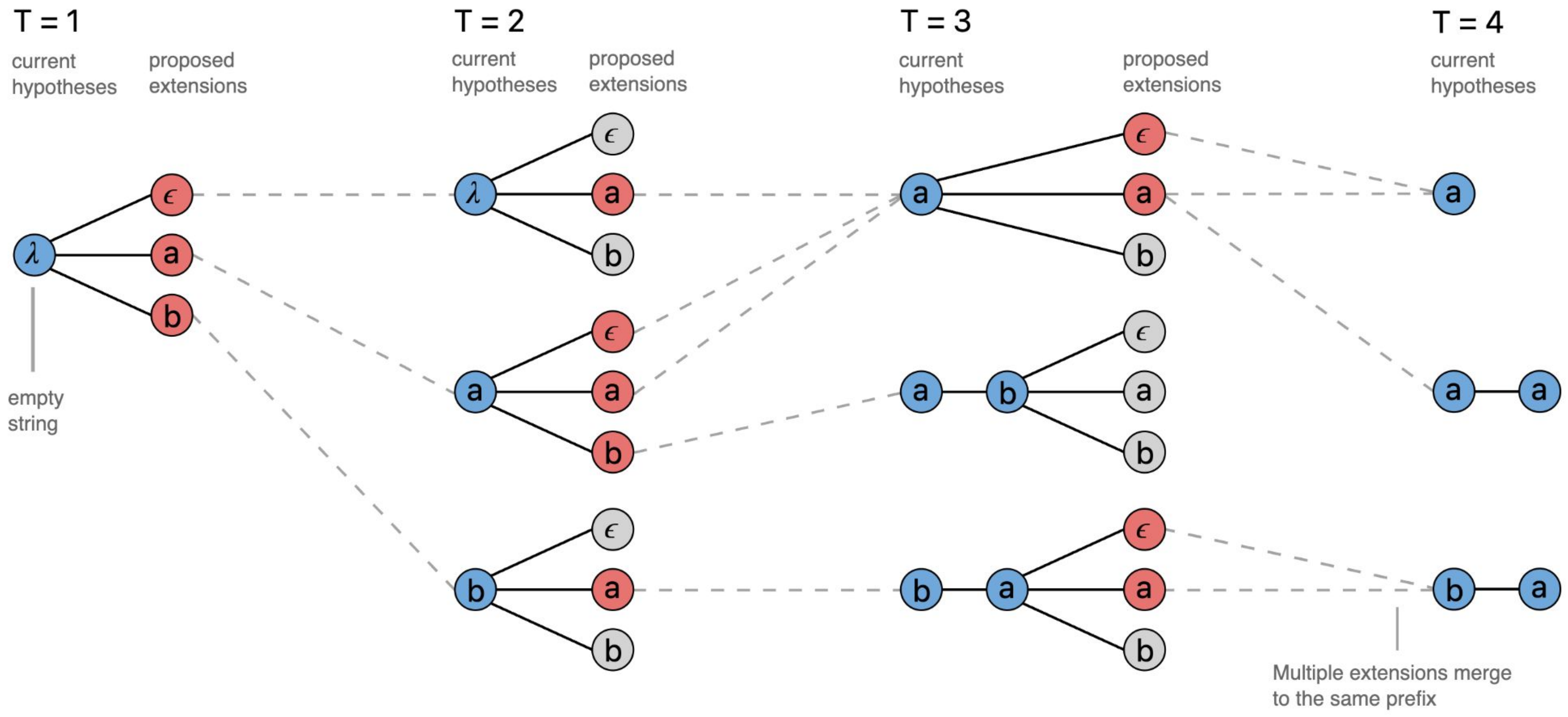
For many applications this heuristic works well,
especially when most of the probability mass is allocated to a single alignment.
However, this approach can sometimes miss easy to find outputs with much higher probability.

A regular beam search computes a new set of hypotheses at each input step.



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

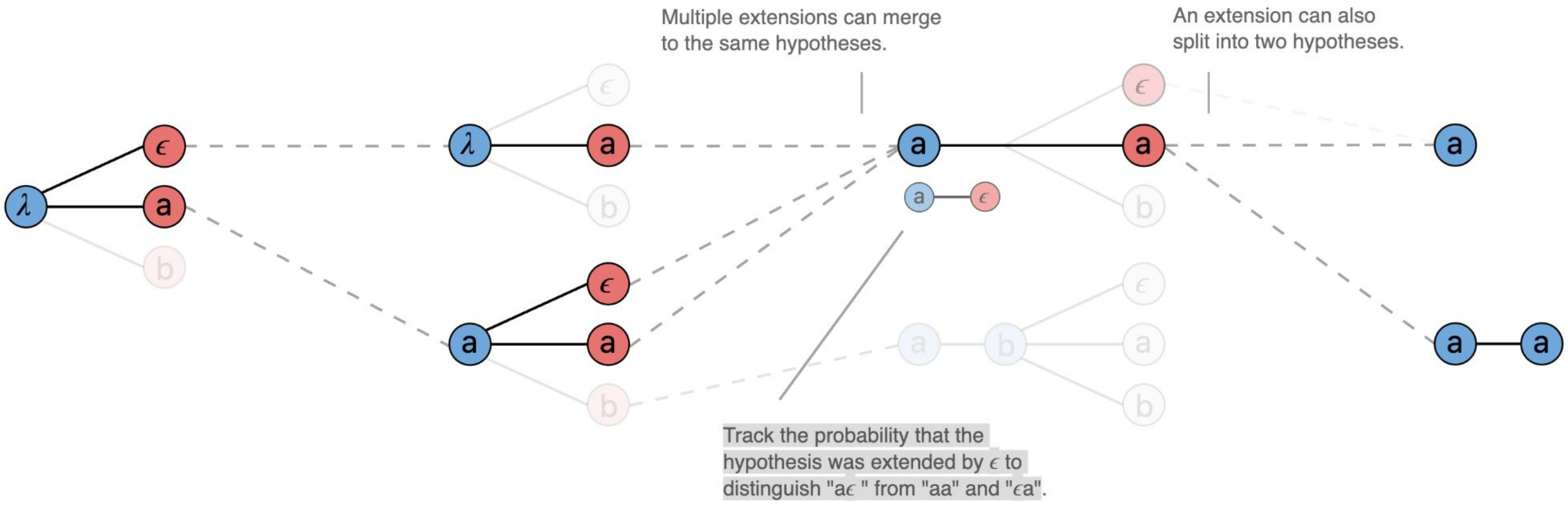
We can modify the vanilla beam search to handle multiple alignments mapping to the same output. In this case instead of keeping a list of alignments in the beam, we store the output prefixes **after collapsing repeats and removing ϵ characters.**



The CTC beam search algorithm with an output alphabet $\{\epsilon, a, b\}$ and a beam size of three.

We have to keep track of two probabilities for each prefix in the beam.

The probability of all alignments which end in ϵ and the probability of all alignments which don't end in ϵ.


Multiple extensions can merge to the same hypotheses.
An extension can also split into two hypotheses.
Track the probability that the hypothesis was extended by ϵ to distinguish "aϵ " from "aa" and "ϵa".
λ
ϵ
a
b
a
b