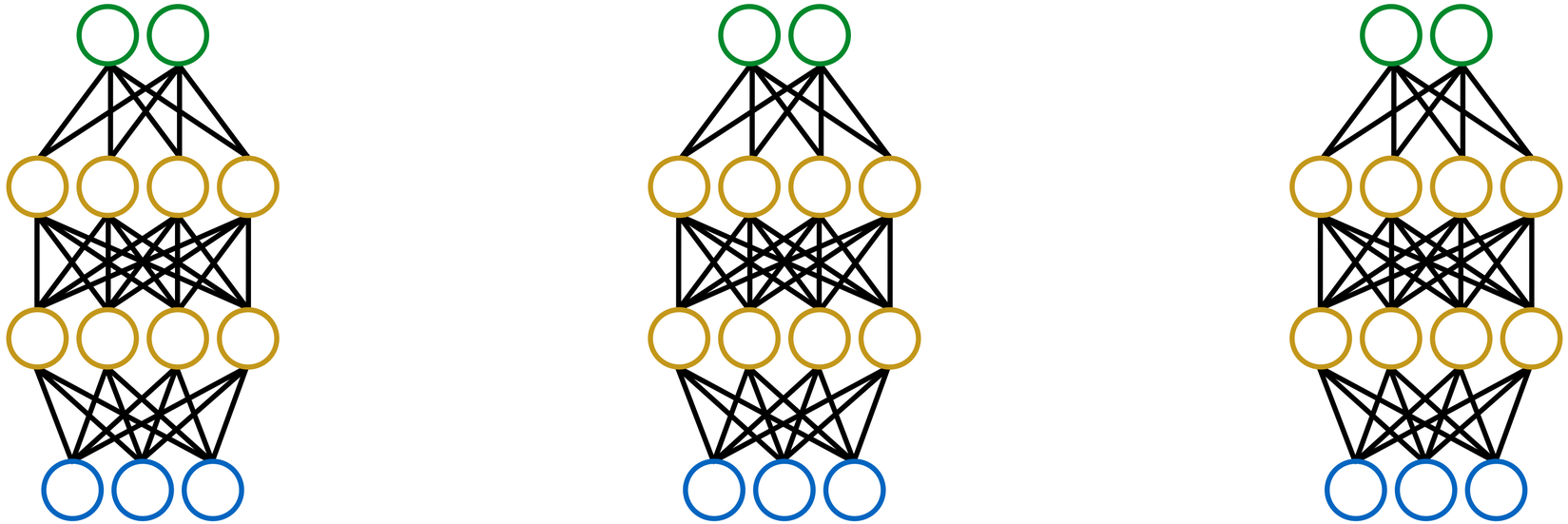
Sequence-to-sequence models

- Class slides

What you should already know

- Converting the linguistic specification into a form suitable for input to Deep Neural Network (DNN)
- The input is now simply a sequence of vectors
- A simple DNN maps one input vector to one output vector (it is *not* aware that the input and output are part of a sequence)

Limitations of processing each time step independently



t- l

t

t+ l

Limitations of processing each time step independently

- Input features
  - Requires assembling all necessary contextual information and placing at current input
  - Features pre-determined using knowledge-driven feature engineering (e.g., quinphones)
- Duration
  - Must be handled separately
- Sequence modelling
  - A constant regression function, time-independent, memoryless
- Output features
  - Predicted using only the input features
  - Output is conditionally-independent of previous/next outputs, given the current input

Orientation

- Input features
  - the model should **learn input feature engineering**
- Duration
  - **integrate** into the model
- Sequence modelling
  - enable the model to pass information between time steps - give it a **memory**
- Output features
  - allow output to **depend** on previous outputs

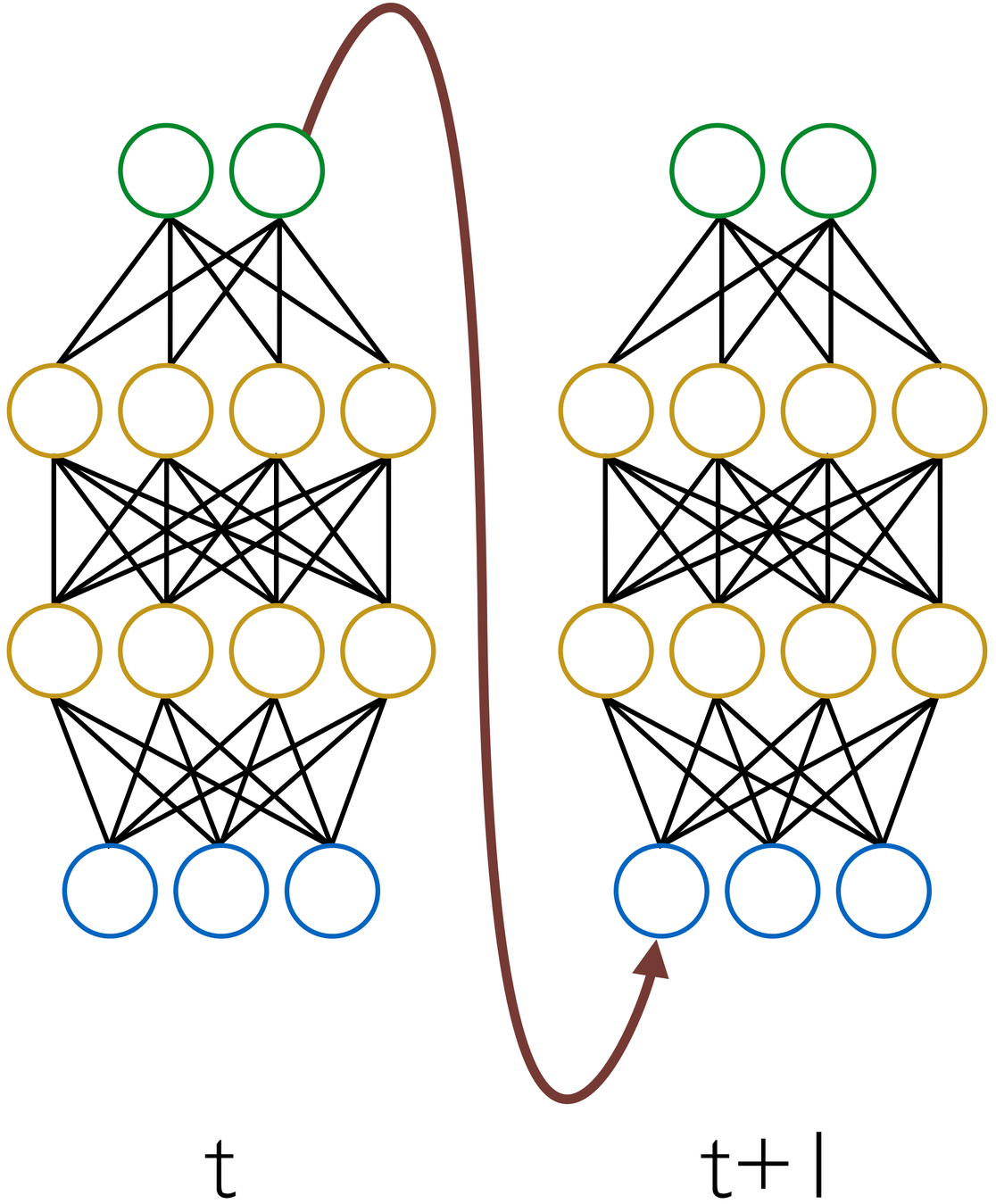
Things to improve next

- Input features
  - the model should **learn input feature engineering**
- Duration
  - **integrate** into the model
- Sequence modelling
  - enable the model to pass information between time steps - give it a **memory**
- Output features
  - allow output to **depend** on previous outputs

Recurrent (naive version)

- Pass some of the **outputs** (or hidden layer activations) forwards in time, typically to the next time step
- A kind of **memory**
- Provides "infinite" left context

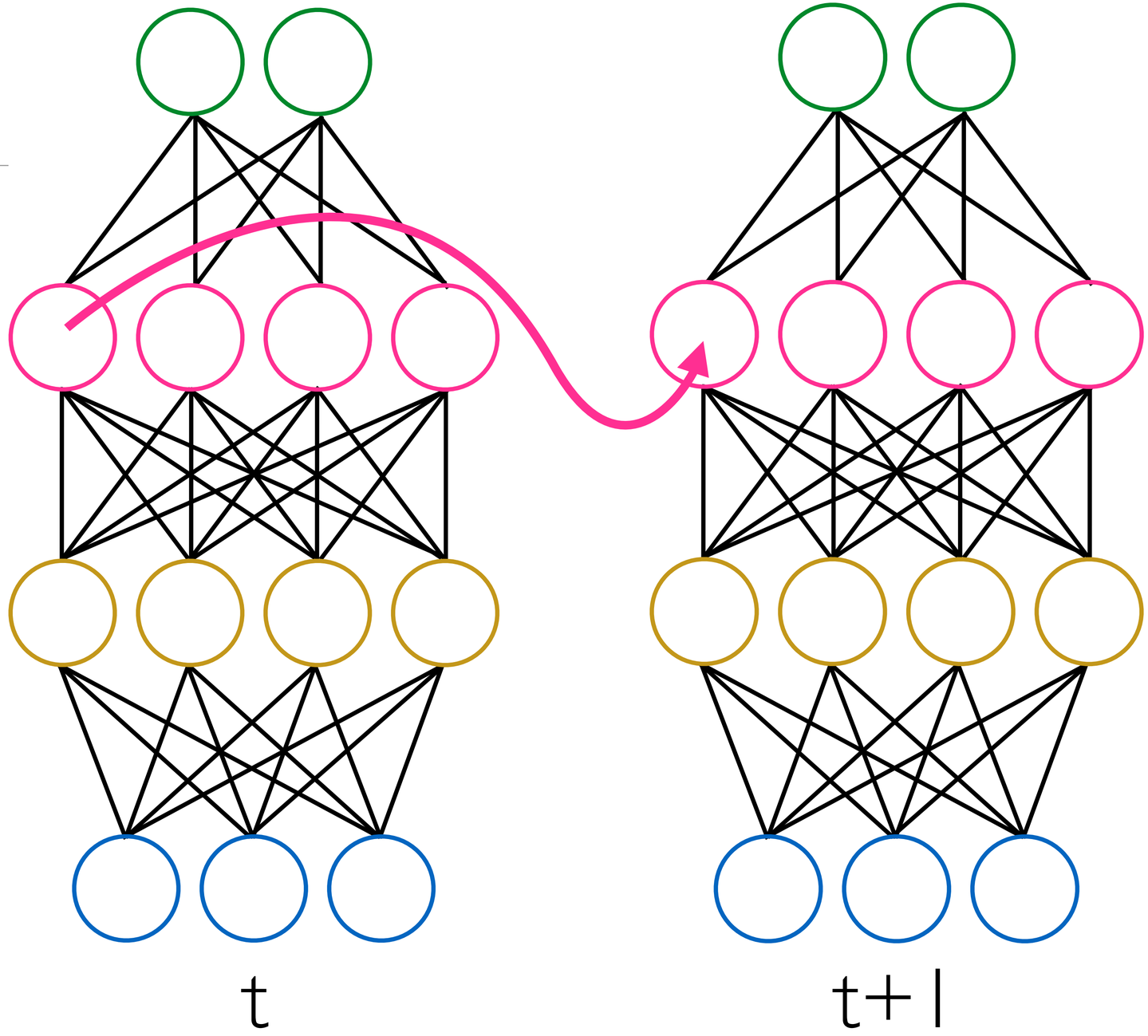
- (could also pass information backwards in time)



Recurrent (naive version)

- Pass some of the outputs (or **hidden layer activations**) forwards in time, typically to the next time step
- A kind of **memory**
- Provides "infinite" left context

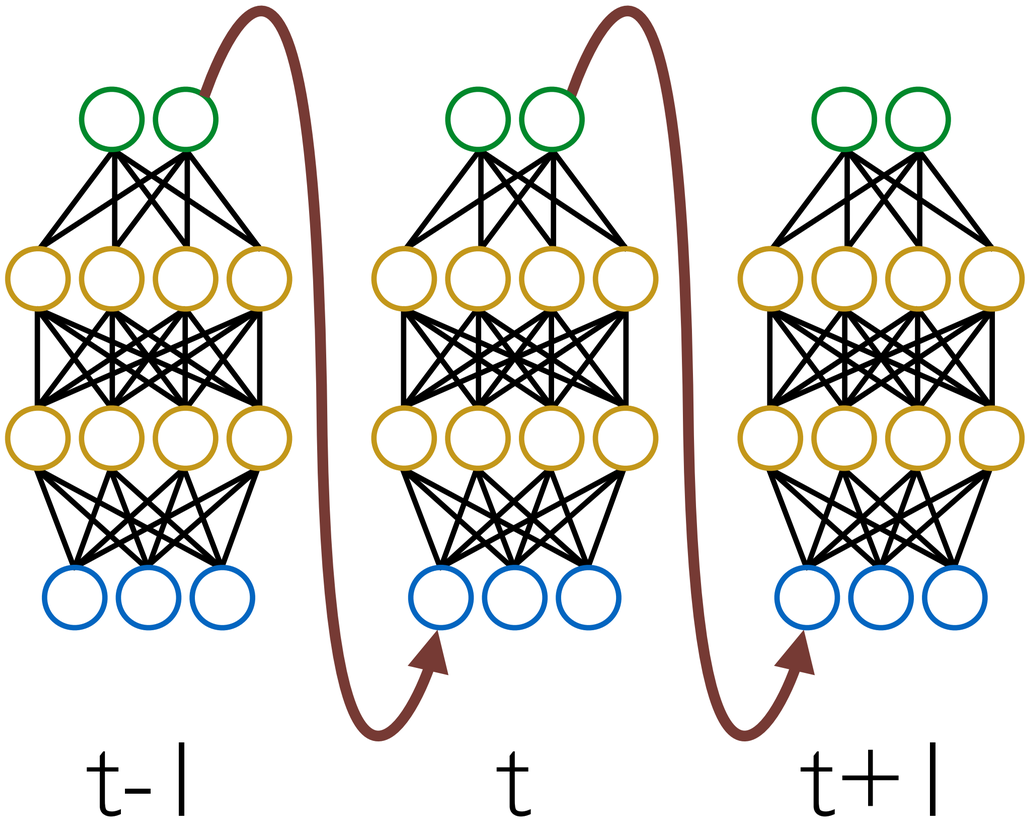
- (could also pass information backwards in time)



Recurrent (naive version)

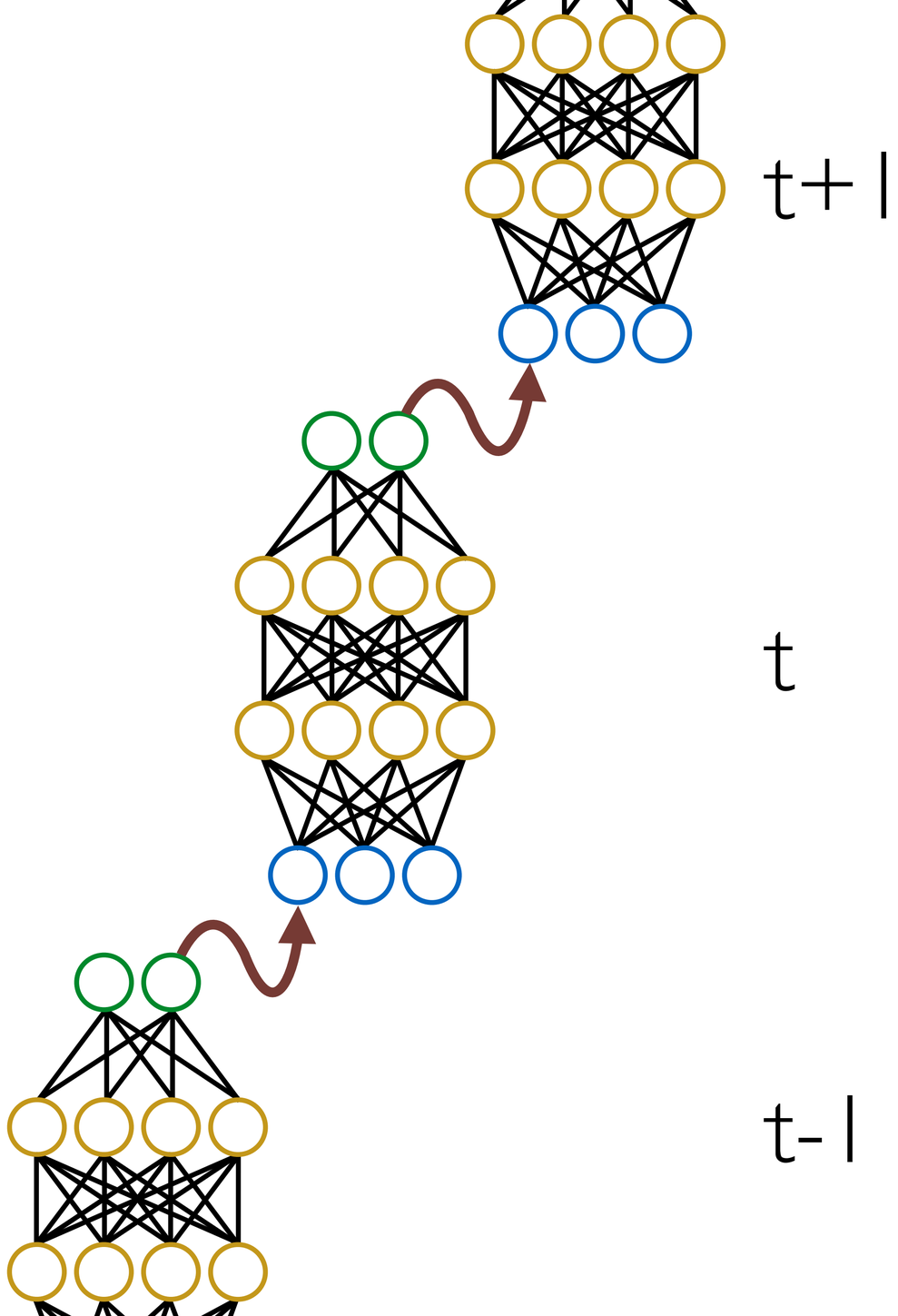
- Pass some of the outputs (or hidden layer activations) forwards in time, typically to the next time step
- A kind of **memory**
- Provides "infinite" left context

- (could also pass information backwards in time)



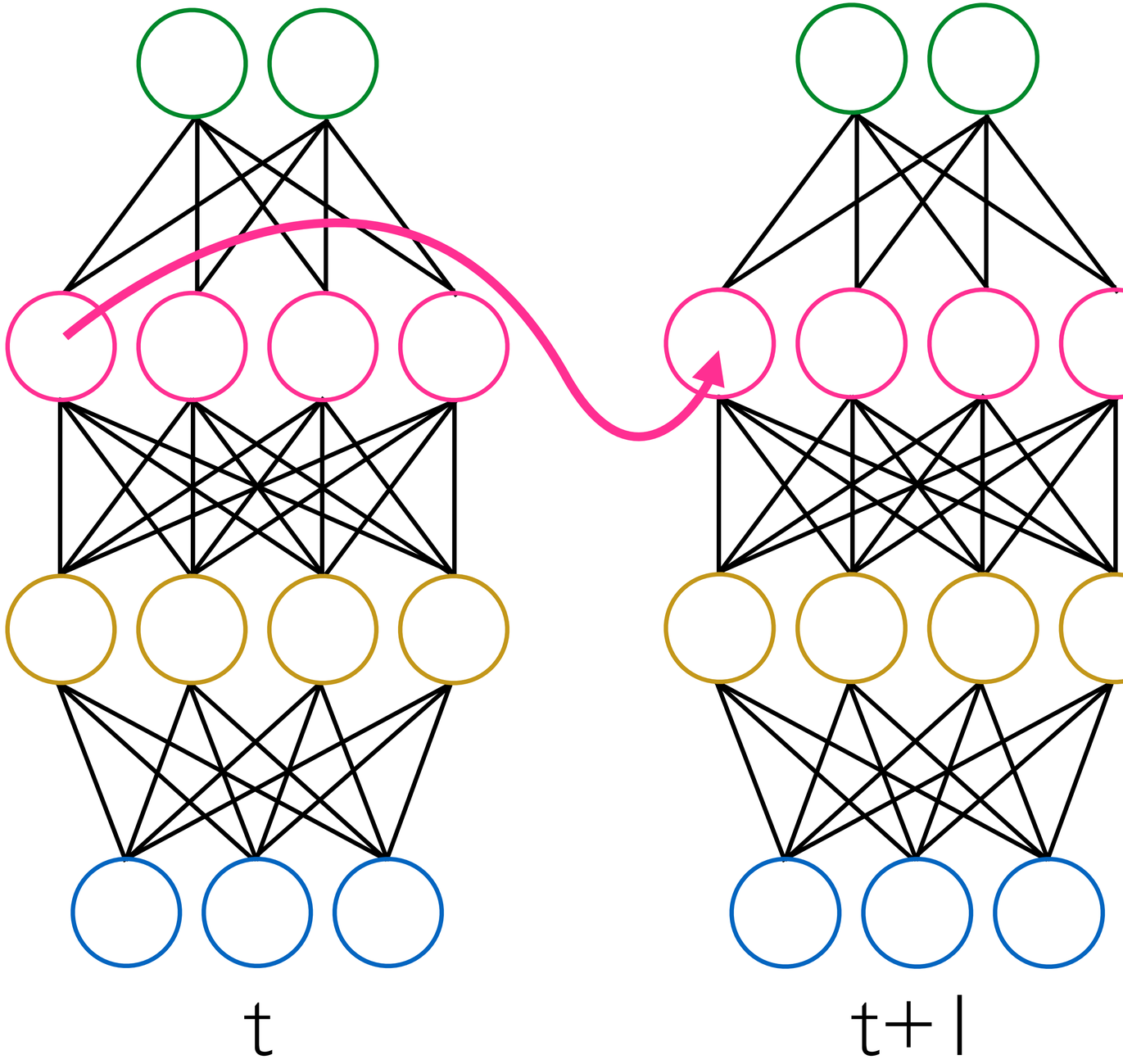
Recurrent

- Simple recurrence is equivalent to a **very deep network**
- To train this network, we have to backpropagate the derivative of the the errors (the **gradient**) through all of the layers
  - "backpropagation through time"
- Suffers from the "**vanishing gradient**" problem, for long sequences



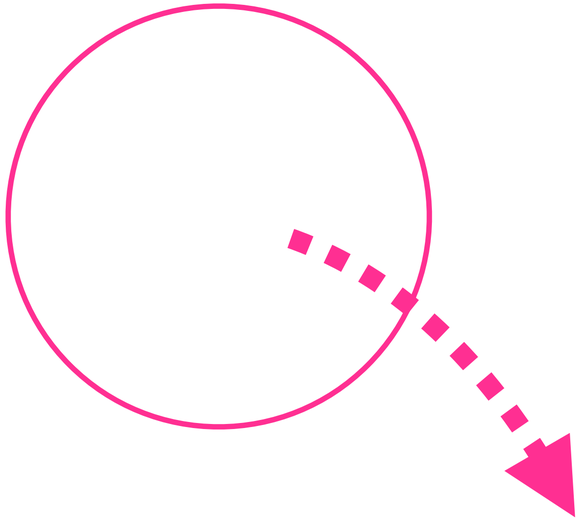
Long short-term memory (a type of recurrence)

- Solves the vanishing gradient problem by using "gates" to control the flow of information
- Conceptually
  - Special LSTM units
    - learn when to **remember**
    - remember information for any number of time steps
    - learn when to **forget**



Long short-term memory (a type of recurrence)

- Solves the vanishing gradient problem by using "gates" to control the flow of information
- Conceptually
  - Special LSTM units
    - learn when to **remember**
    - remember information for any number of time steps
    - learn when to **forget**



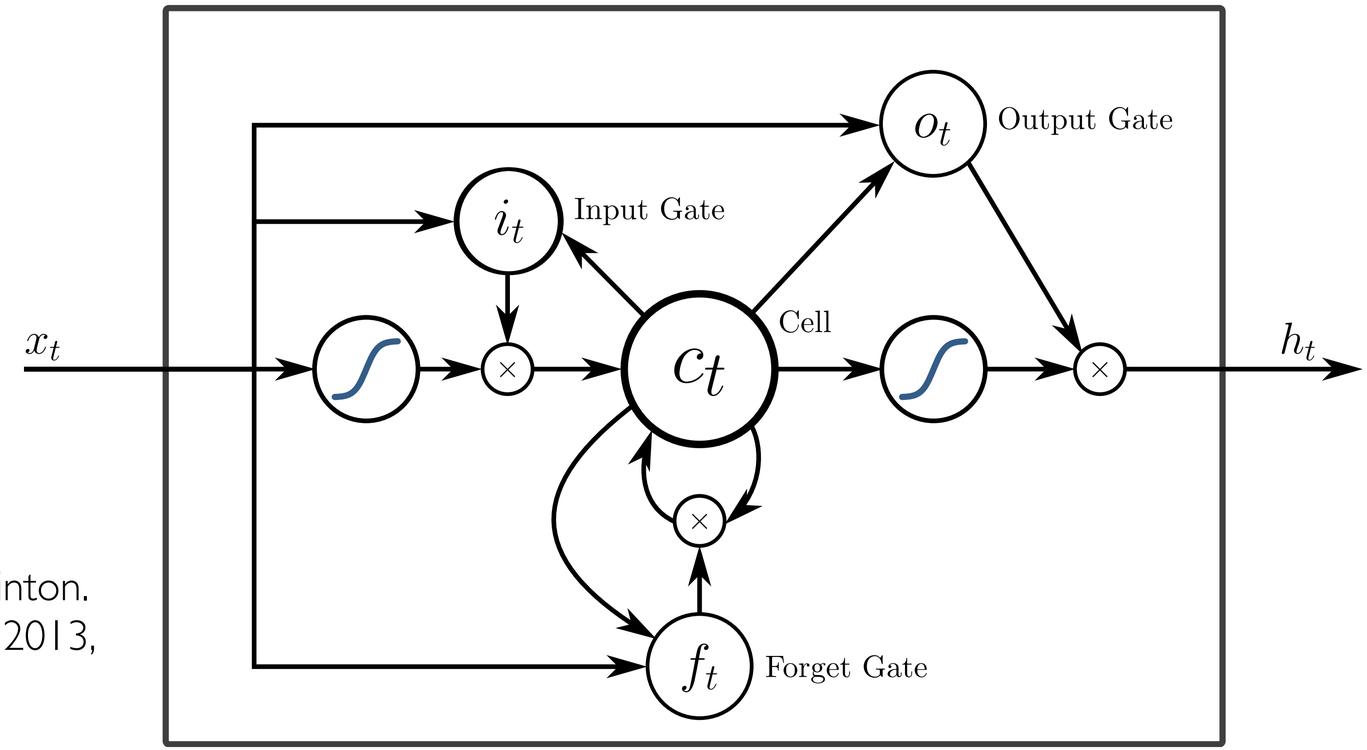


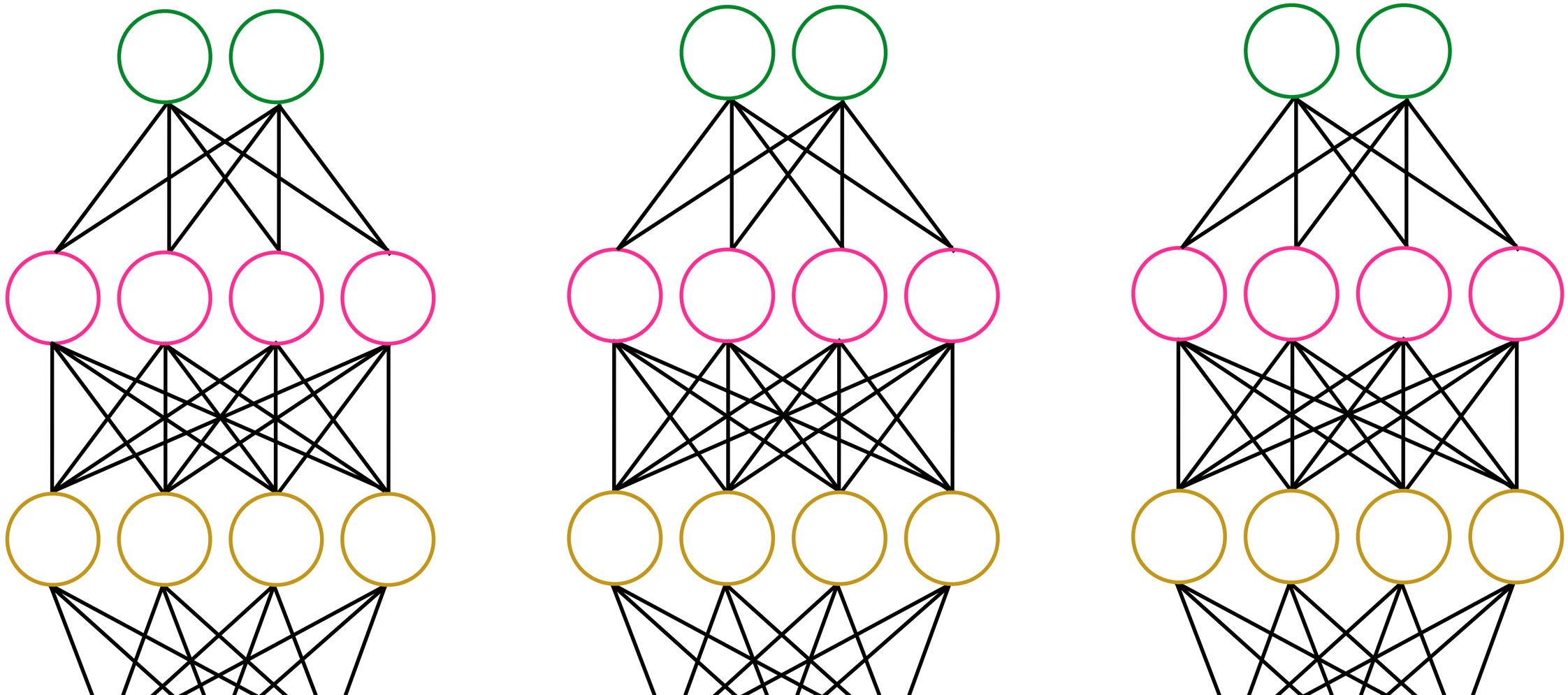
Figure from Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. "Speech recognition with deep recurrent neural networks" ICASSP 2013, redrawn as SVG by Eddie Antonio Santos

LSTM units & Gated Recurrent Units (GRUs)

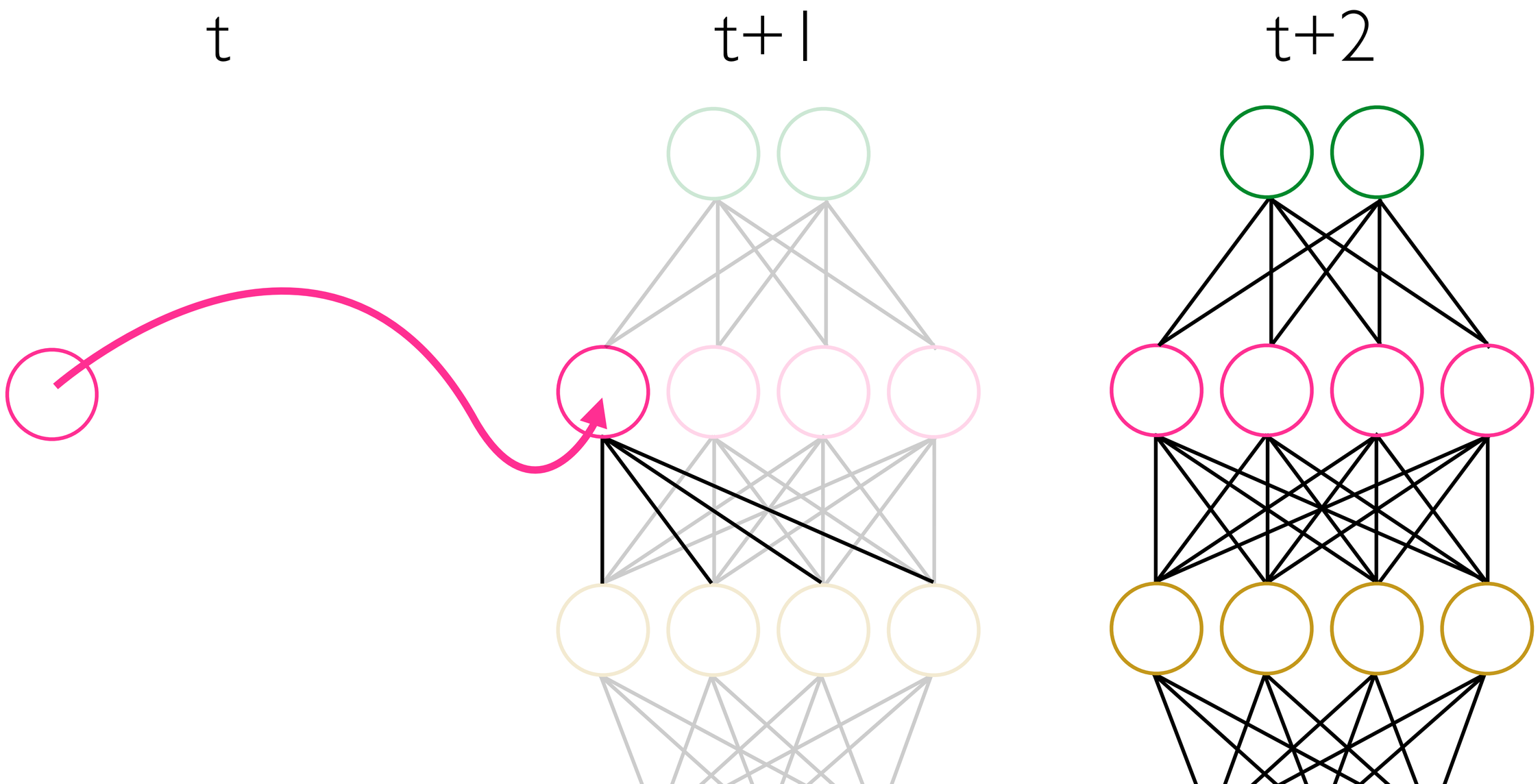
t

t+1

t+2

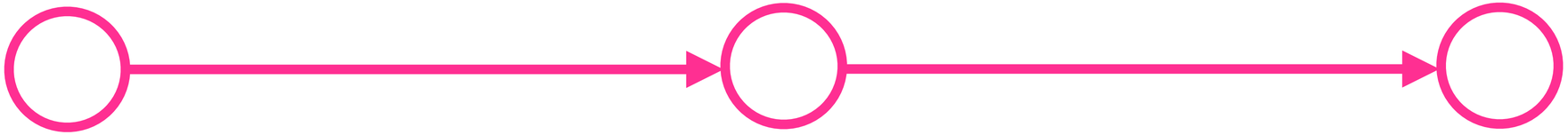


LSTM units & Gated Recurrent Units (GRUs)



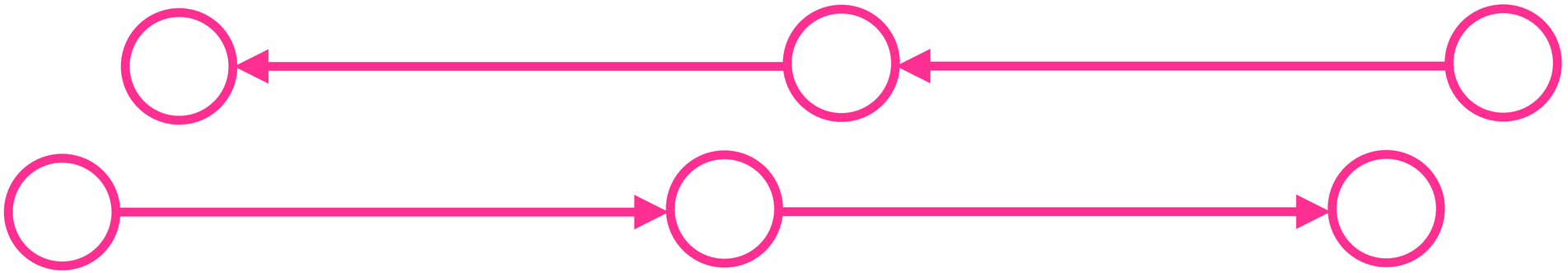
Neural building blocks : (bidirectional) LSTM layer

t t+1 t+2

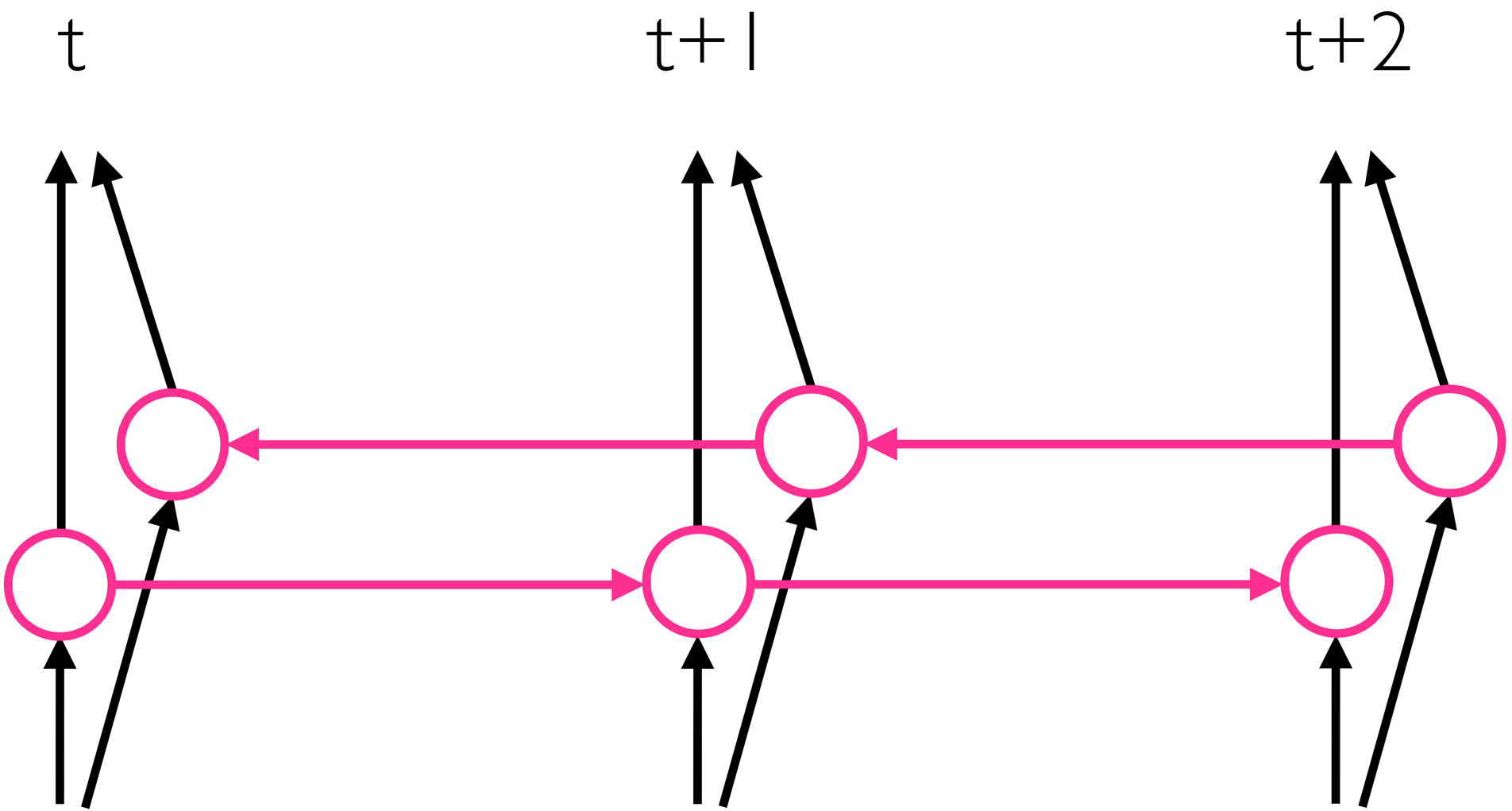


Neural building blocks : (bidirectional) LSTM layer

t t+1 t+2



Neural building blocks : (bidirectional) LSTM layer



Orientation

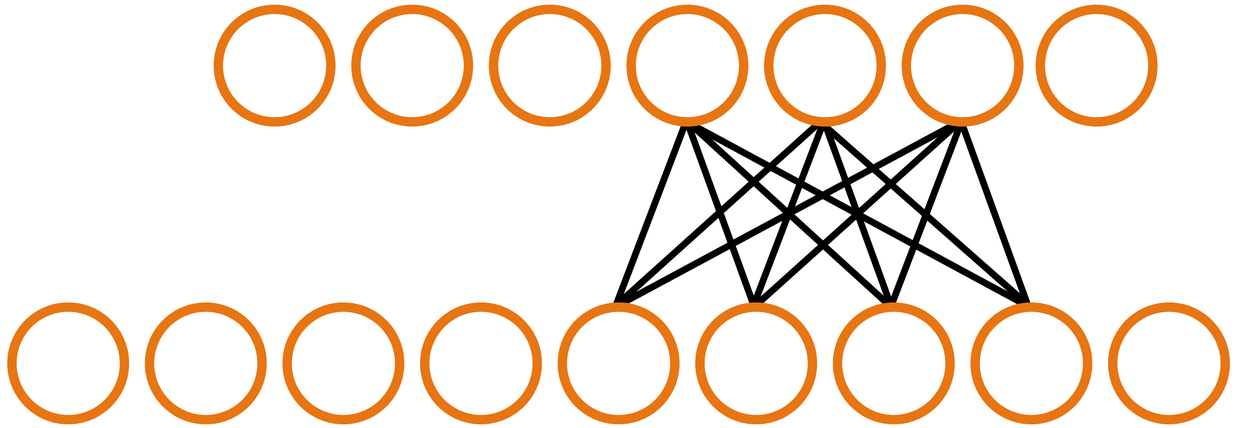
- Input features
  - the model should **learn input feature engineering**
- Duration
  - **integrate** into the model
- Sequence modelling
  - enable the model to pass information between time steps - give it a **memory**
- Output features
  - allow output to **depend** on previous outputs

- Feed-forward architecture
  - no memory
- "Simple" recurrent neural networks
  - vanishing gradient problem
- **LSTMs or GRUs** (which avoid the vanishing gradient problem)

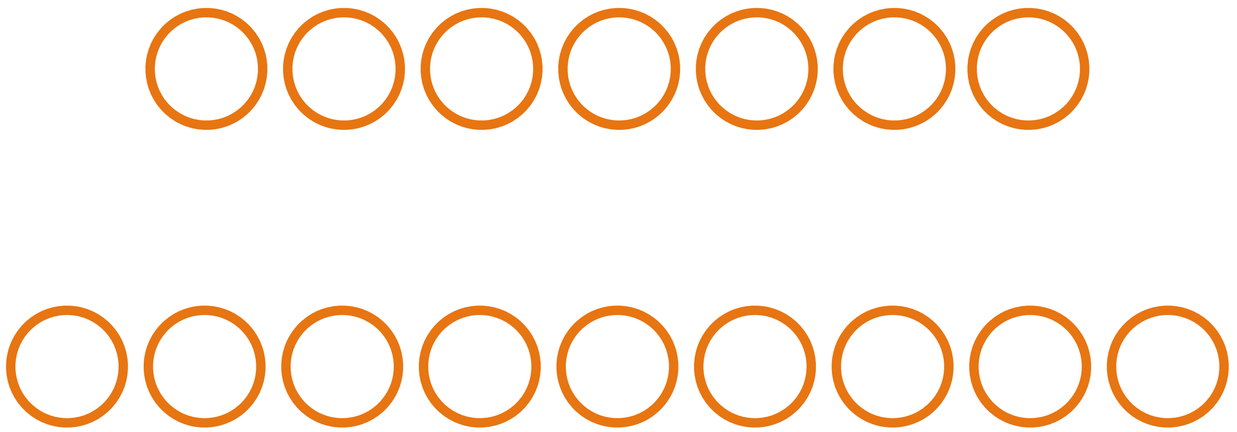
Orientation

- Input features
  - the model should **learn input feature engineering**
- Duration
  - **integrate** into the model
- Sequence modelling
  - enable the model to pass information between time steps - give it a **memory**
- Output features
  - allow output to **depend** on previous outputs

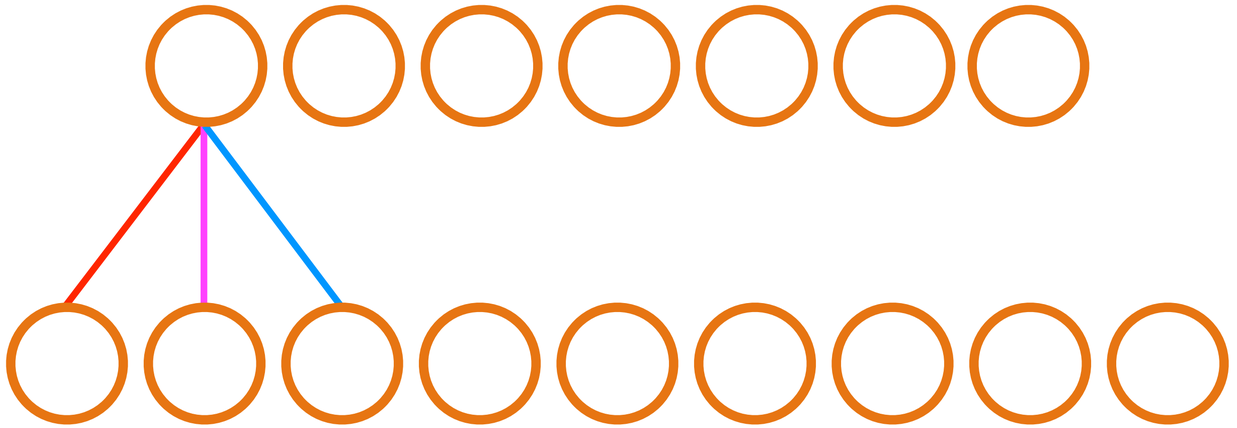
Neural building blocks : fully connected layer



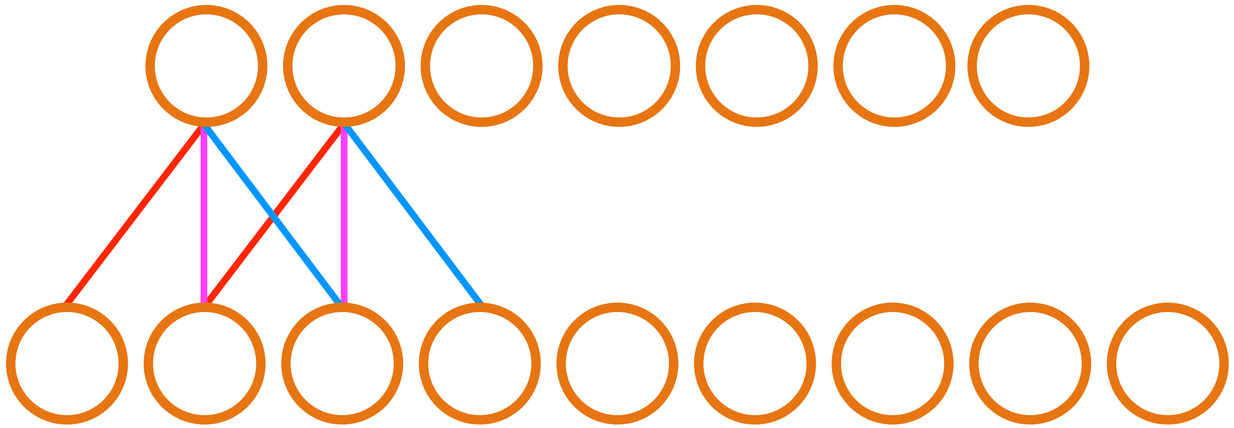
Neural building blocks : convolutional layer



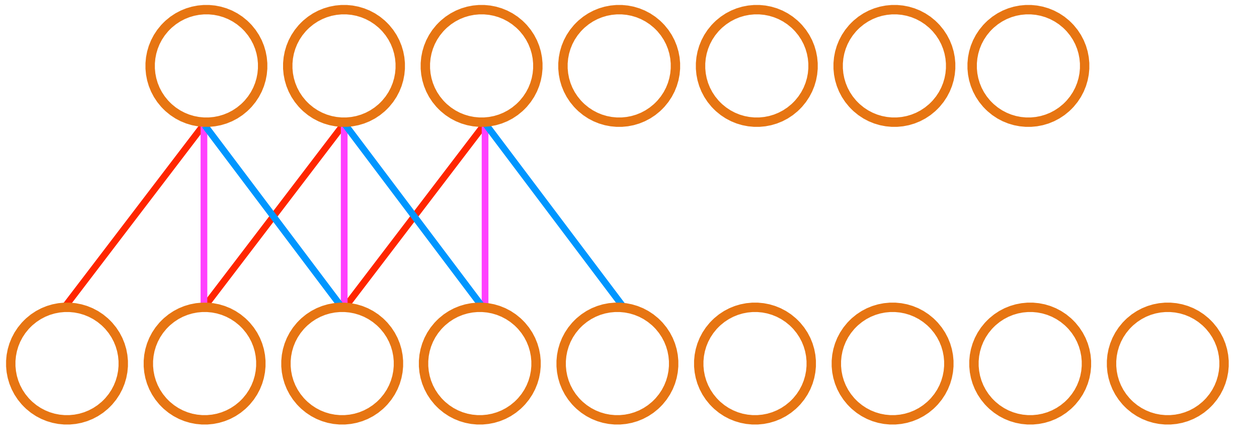
Neural building blocks : convolutional layer



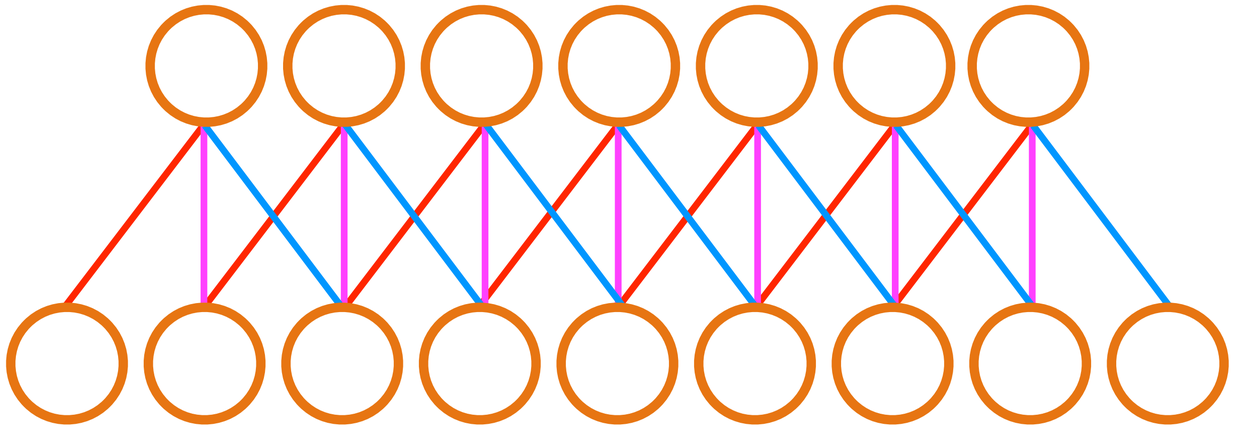
Neural building blocks : convolutional layer



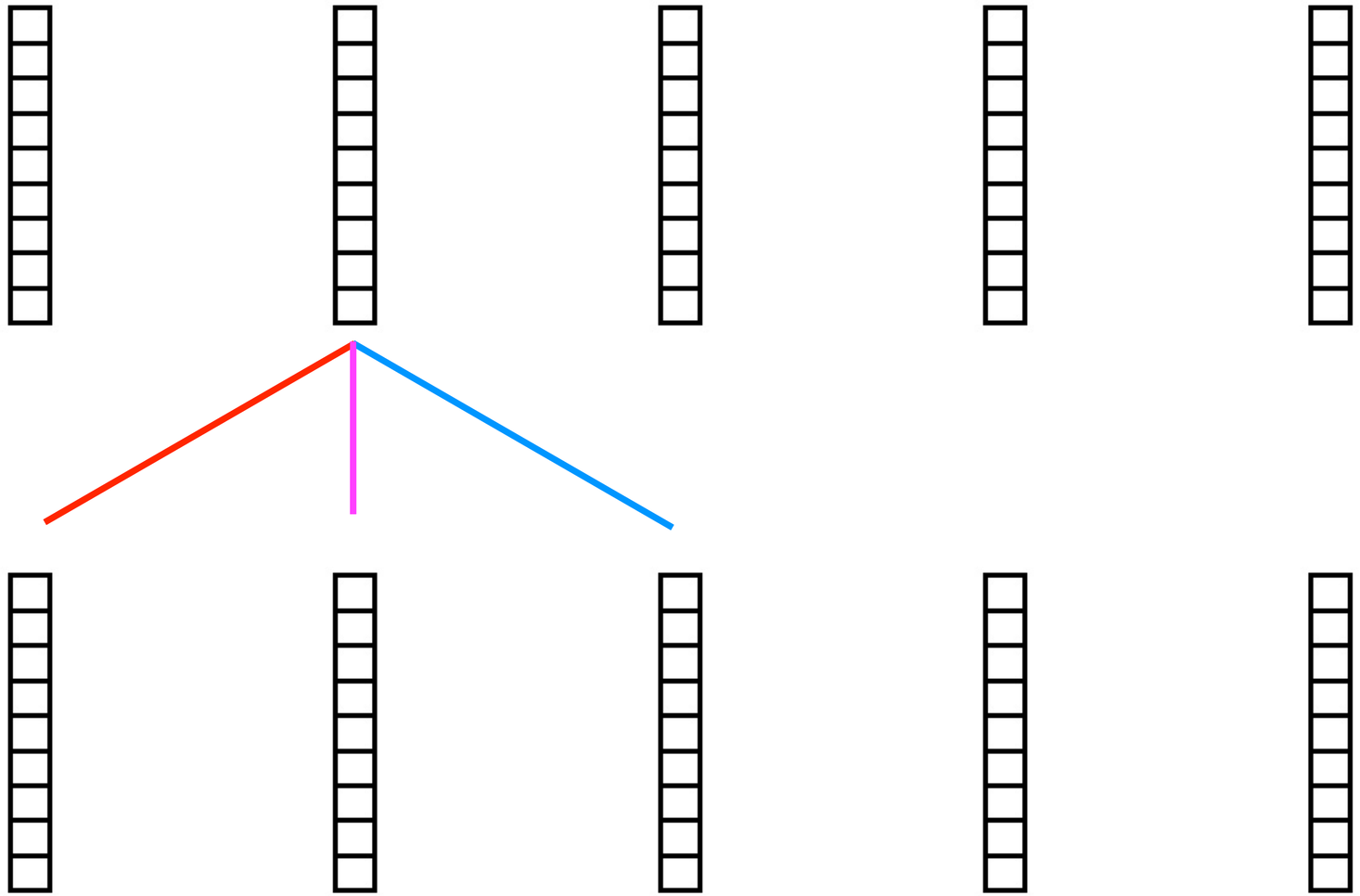
Neural building blocks : convolutional layer



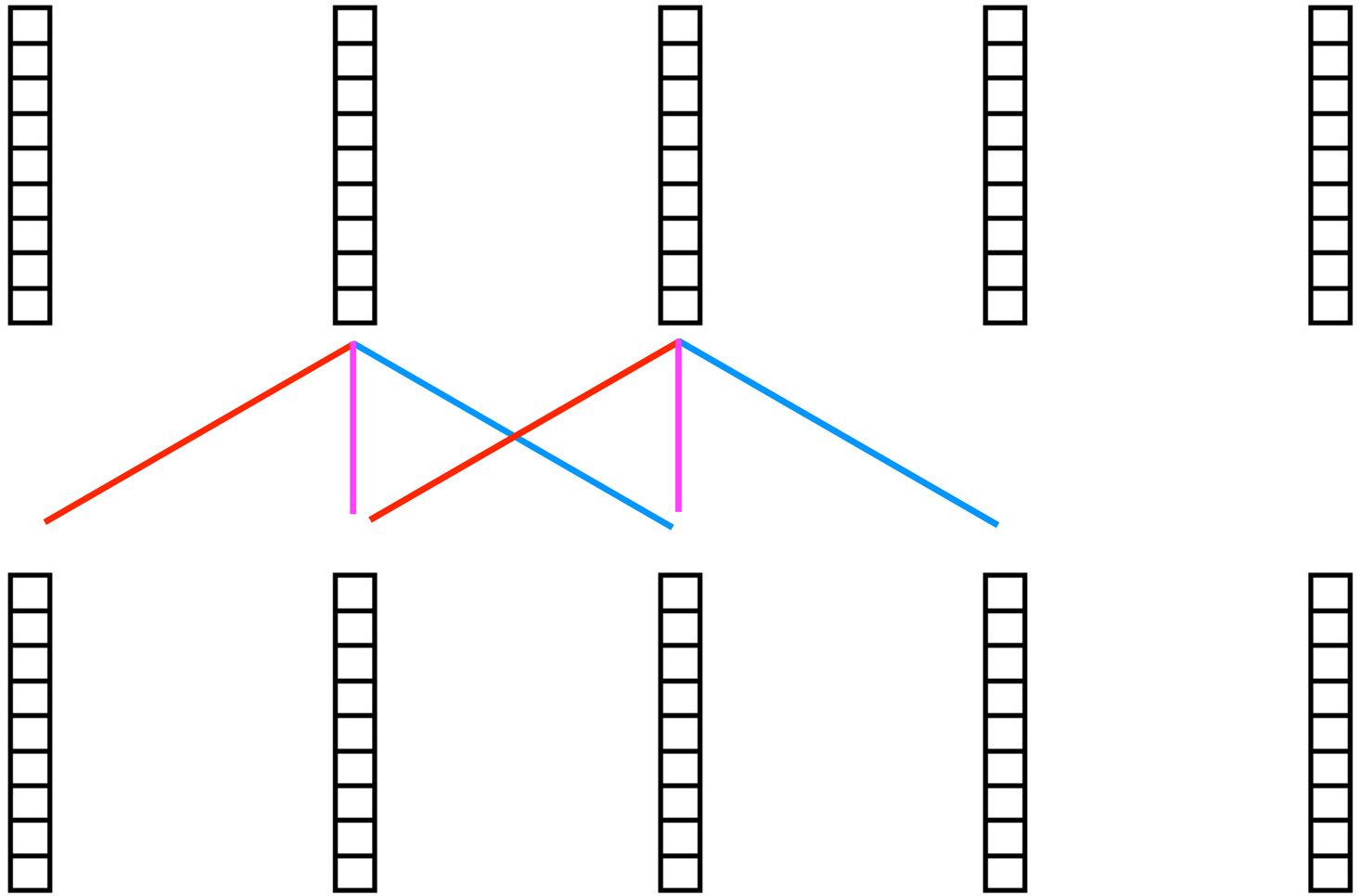
Neural building blocks : convolutional layer



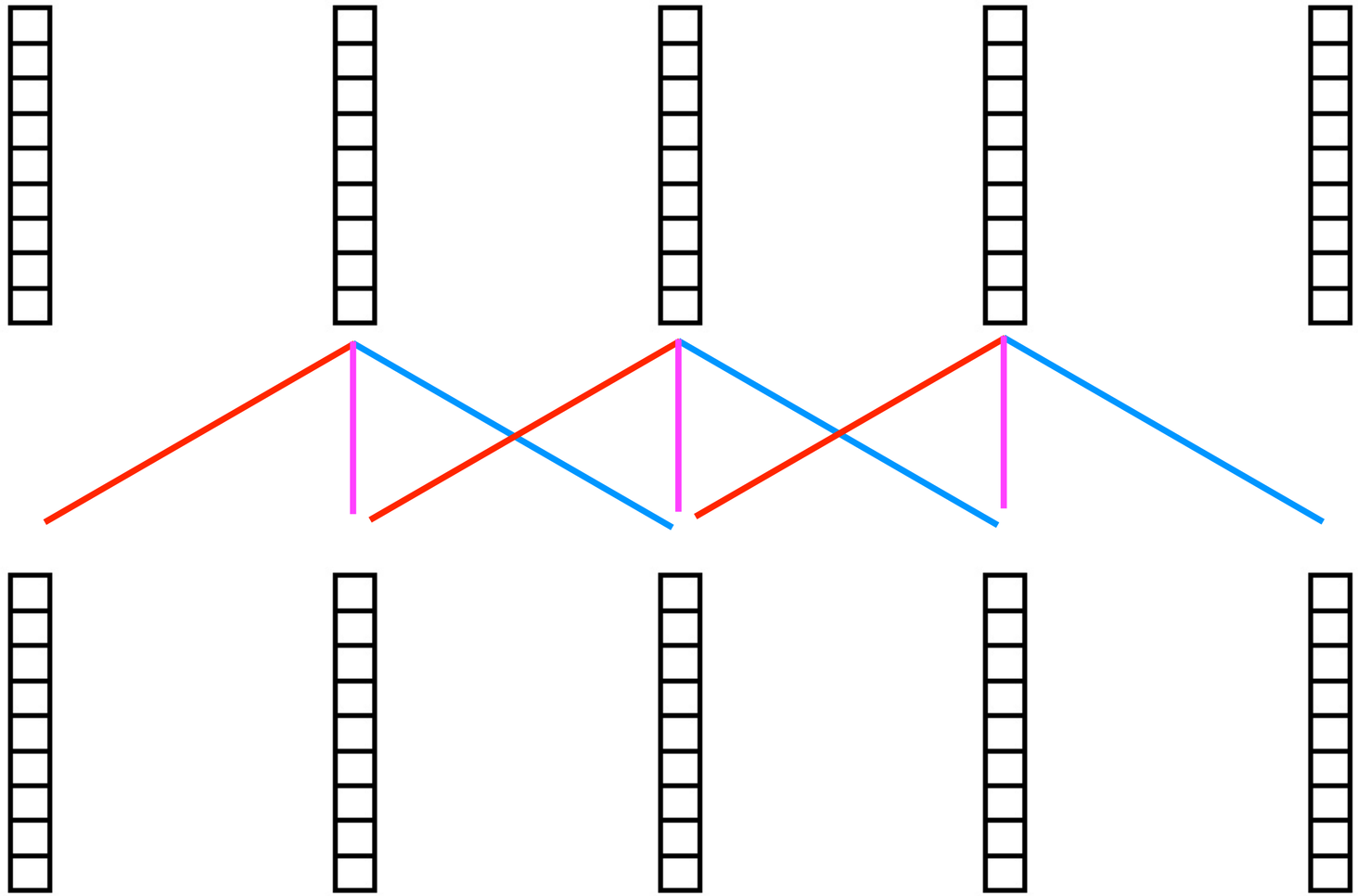
Using convolution to learn input feature engineering



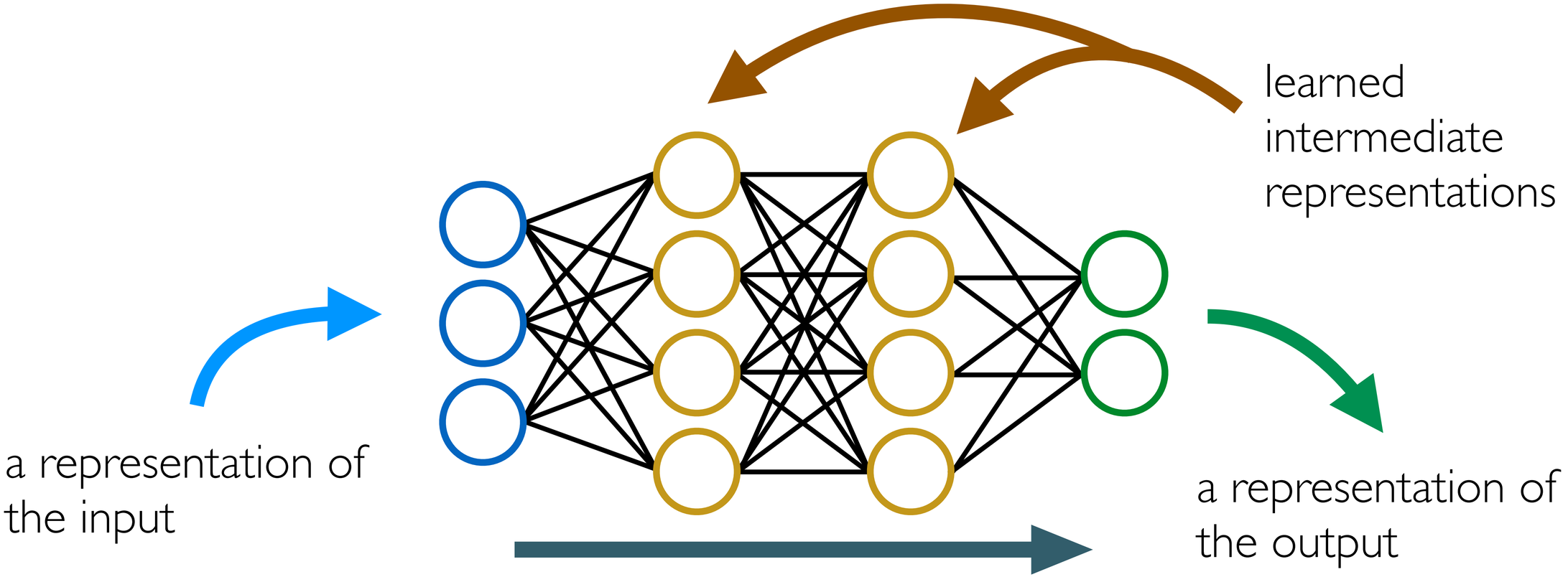
Using convolution to learn input feature engineering



Using convolution to learn input feature engineering



PAUSE! What are all those layers for? Learning **representations**!

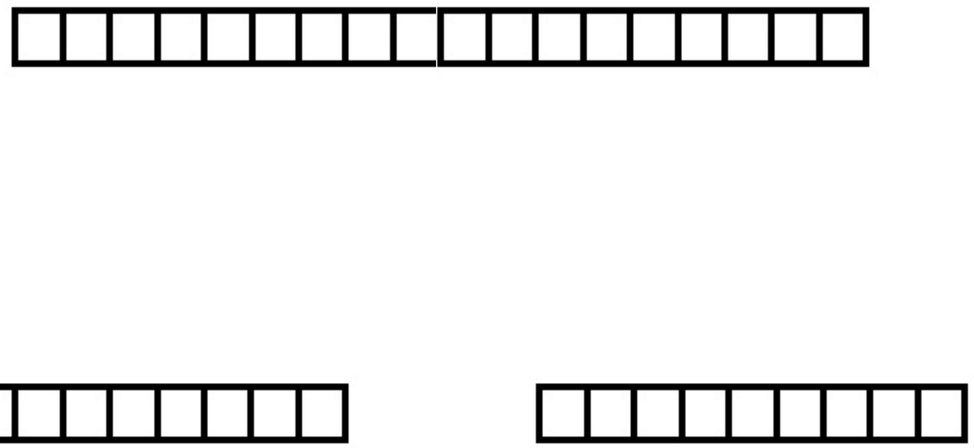


Inputting a one-hot vector into the model: **embedding**

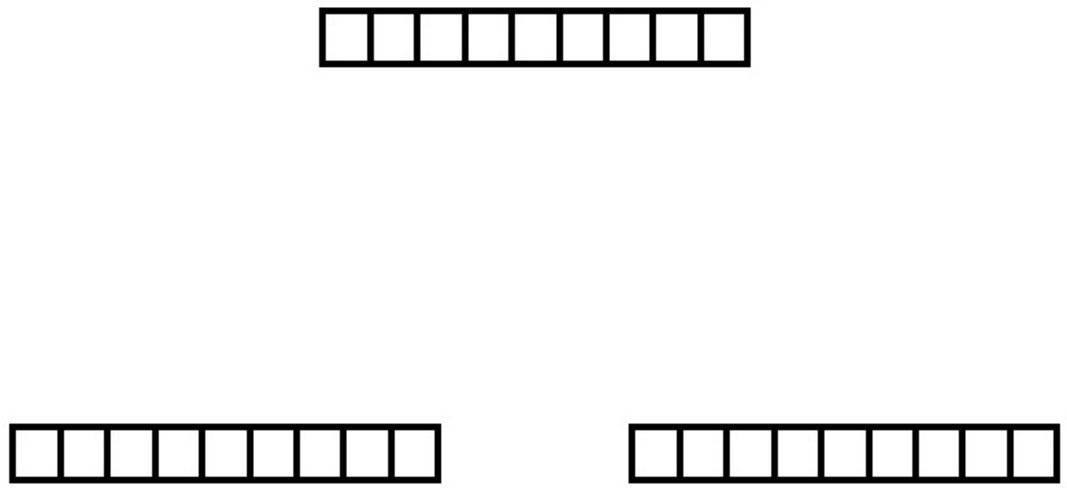
Changing the dimensionality of the representation: **projection**

Combining representations as information flows through the model

Option 1: concatenate

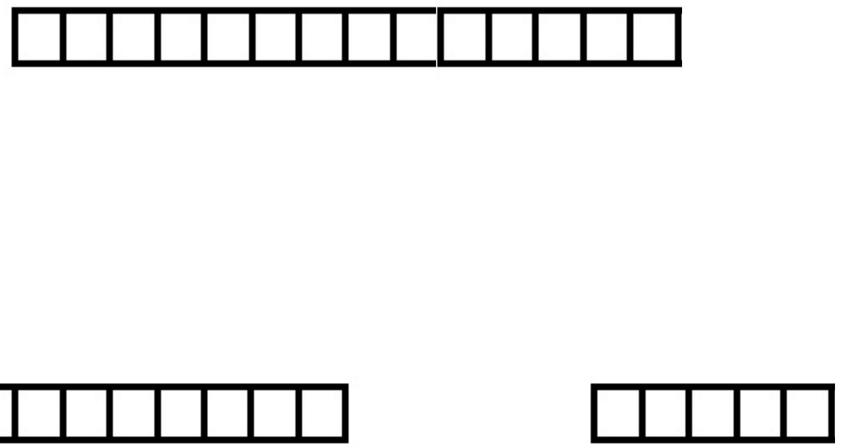


Option 2: sum

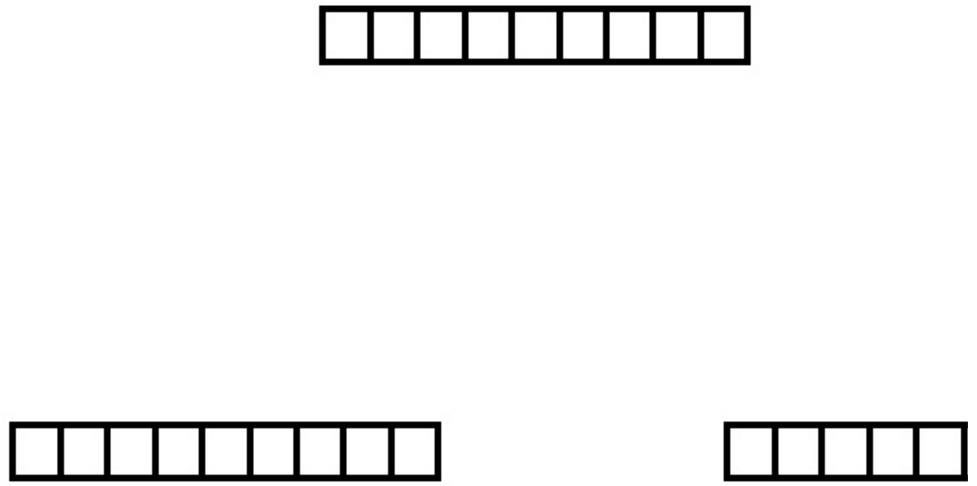


Combining representations as information flows through the model

Option 1: concatenate



Option 2: sum



Combining representations as information flows through the model

Option 1: concatenate

Option 2: sum

Terminology

- types of layer
  - fully-connected (FC)
  - recurrent
  - LSTM, GRU, bidirectional LSTM (BiLSTM)
  - convolutional (conv, conv 1D)

- operations
  - embedding
  - projection
  - sum ($\oplus$) vs. concatenation (concat)

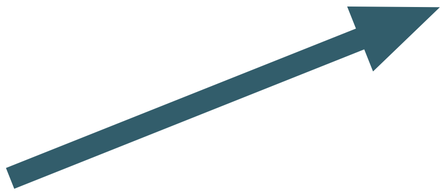
Orientation

- Input features
  - the model should **learn input feature engineering**
- Duration
  - **integrate** into the model
- Sequence modelling
  - enable the model to pass information between time steps - give it a **memory**
- Output features
  - allow output to **depend** on previous outputs

Encode the input using neural building blocks such as convolutional layers

Orientation

- Input features
  - the model should **learn input feature engineering**
- Duration
  - **integrate** into the model
- Sequence modelling
  - enable the model to pass information between time steps - give it a **memory**
- Output features
  - allow output to **depend** on previous outputs



- Solution 1: attention
- Solution 2: explicit duration model

During training: alignment

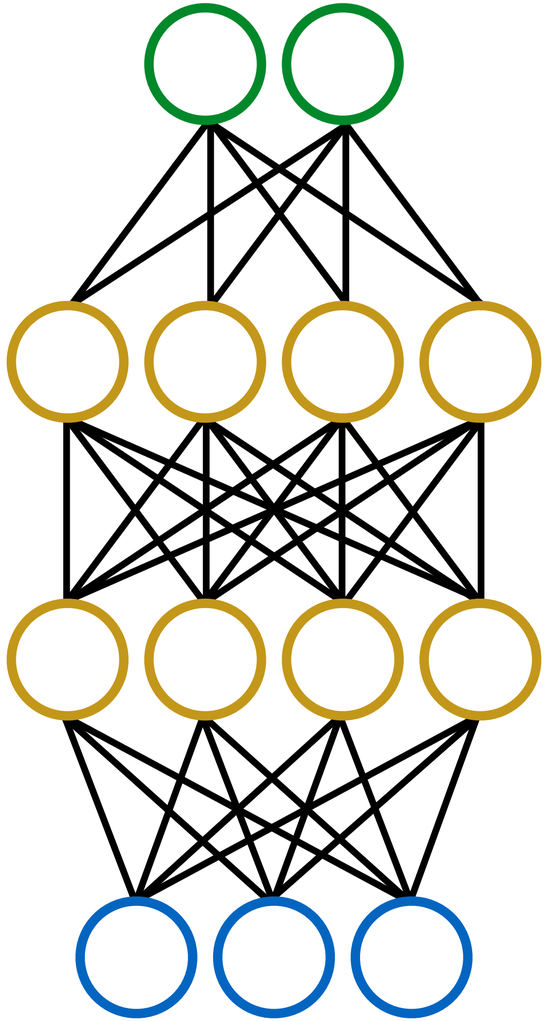
During inference: duration prediction

- Length of input sequence is generally **different** to length of output sequence

- For example
  - input: sequence of phones
  - output: acoustic frames (e.g., a spectrogram, to be input to a vocoder)

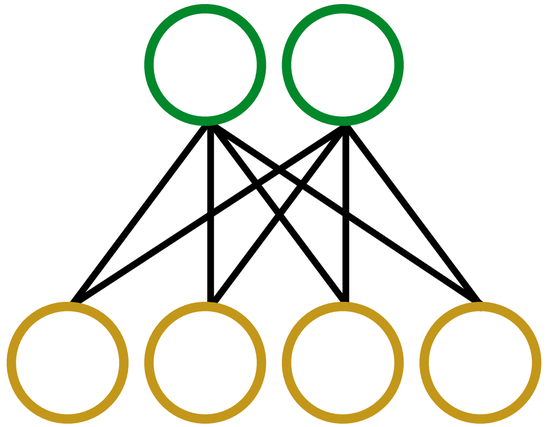
- Conceptually
  - **read** in the input sequence; **memorise** it using a **learned representation**
  - given that representation, **write** the output sequence

output time steps are frames (e.g., of a mel spectrogram)

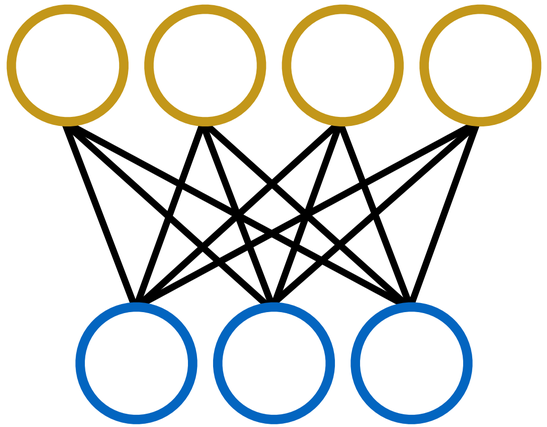


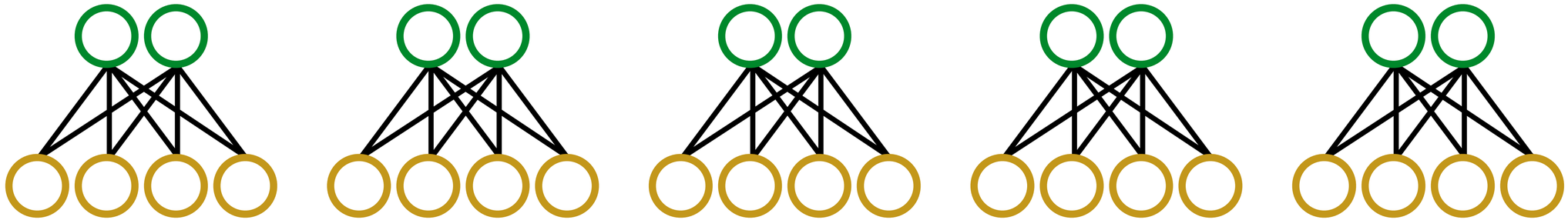
input time steps are linguistic units (e.g., phones)

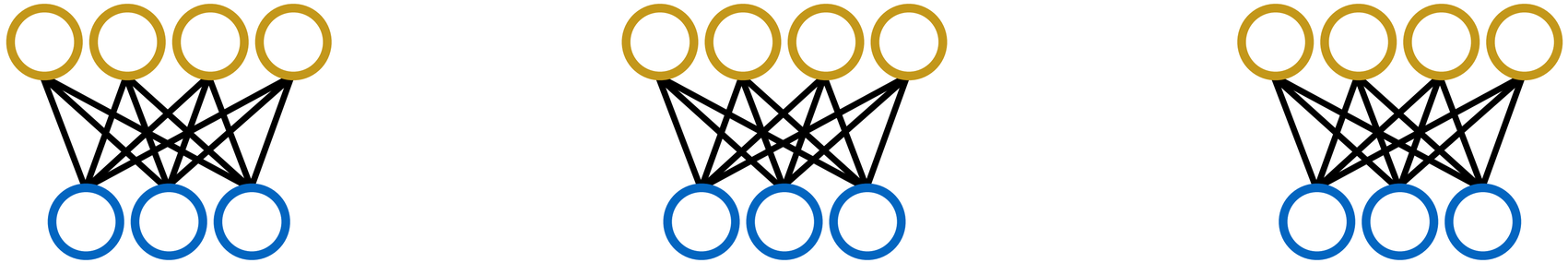
Decoder

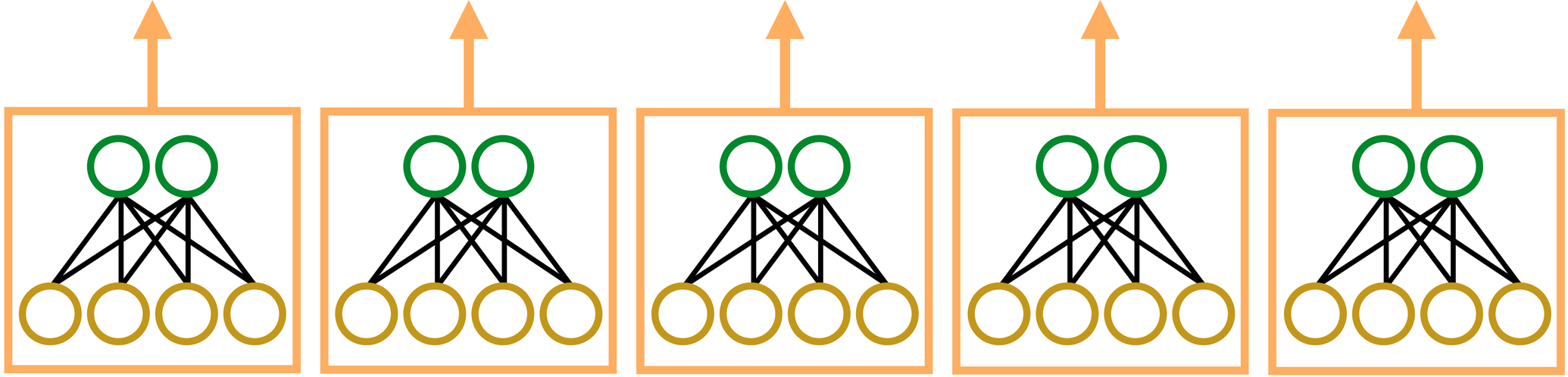


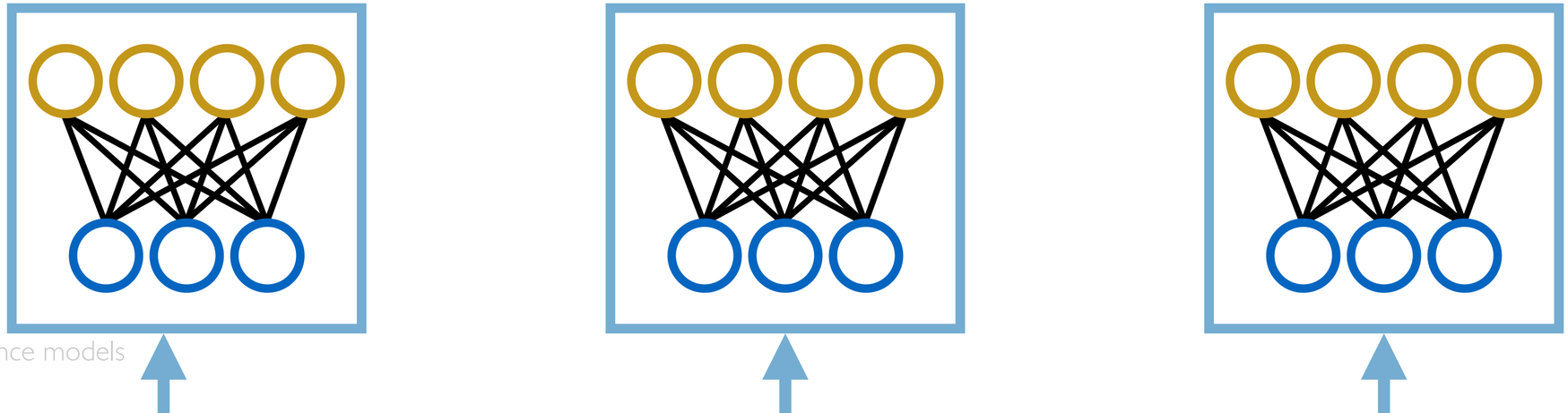
Encoder



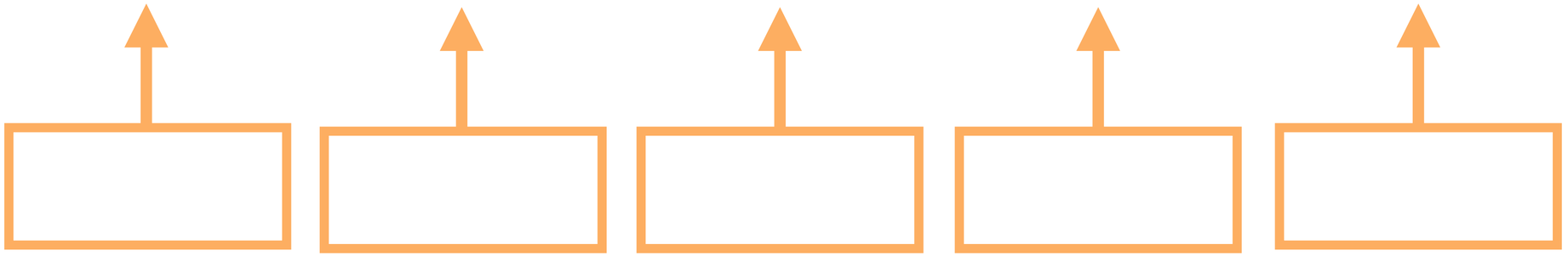






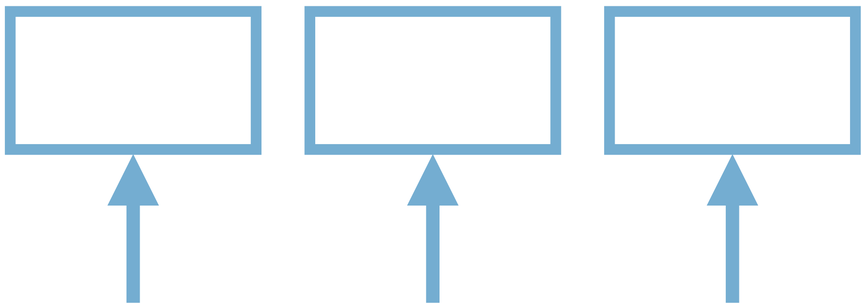


A sequence-to-sequence network using an encoder-decoder architecture

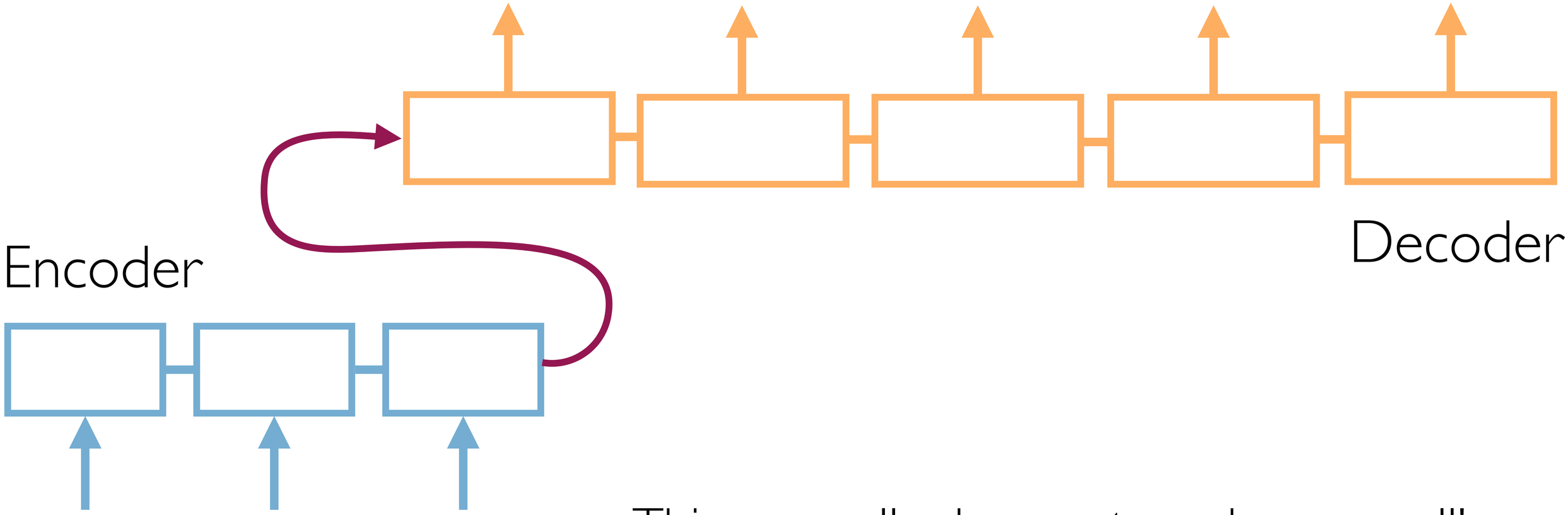


Decoder

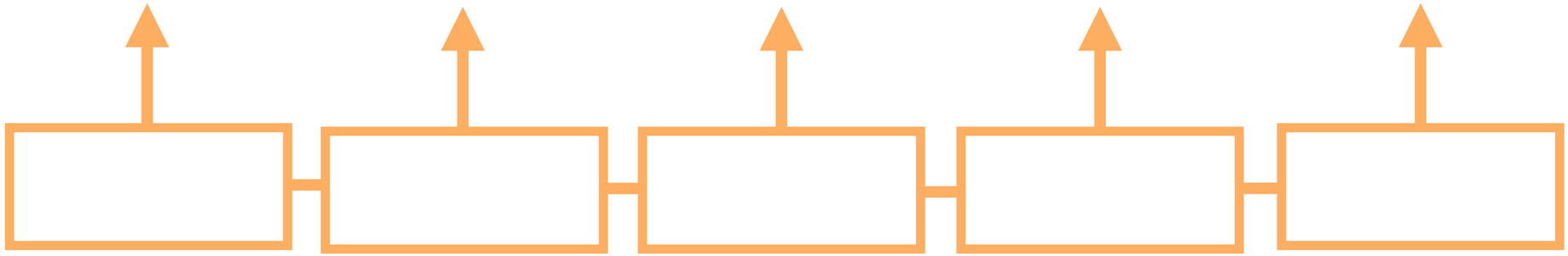
Encoder

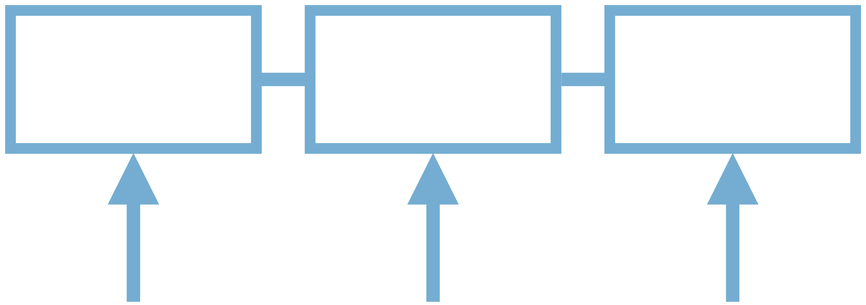


A sequence-to-sequence network using an encoder-decoder architecture

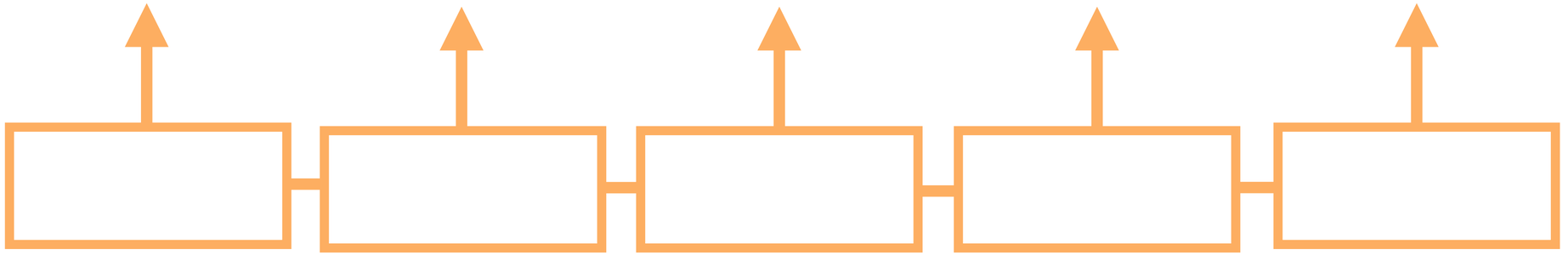


This generally does not work very well!
Why?

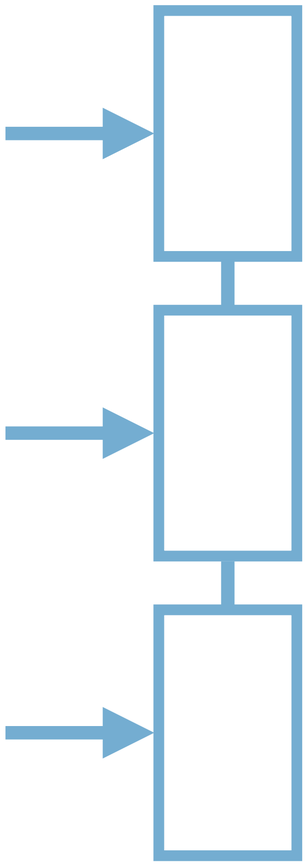




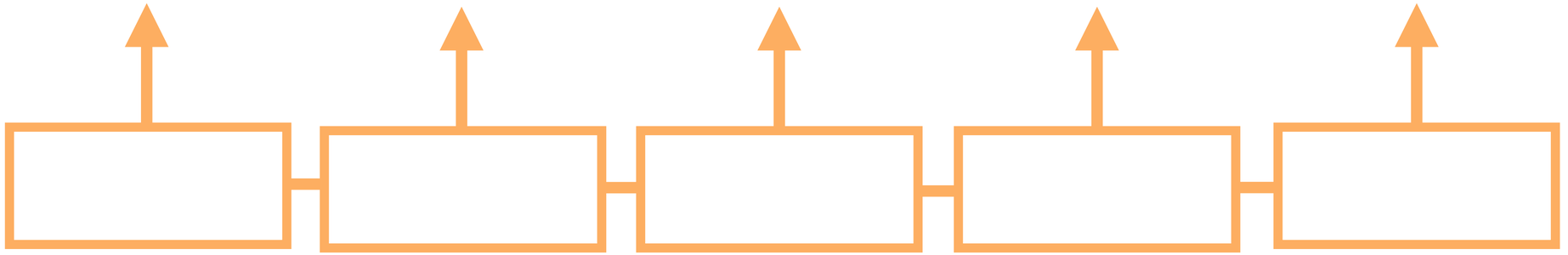
Decoder



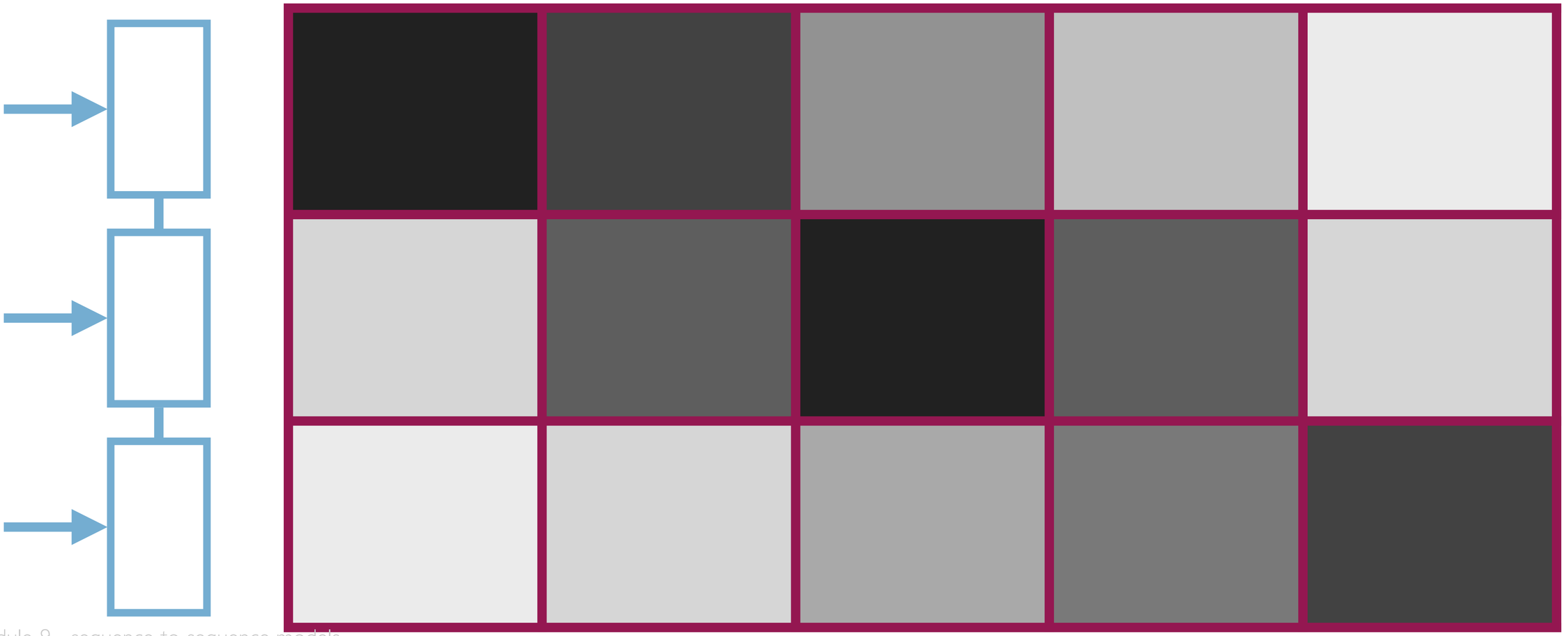
Encoder



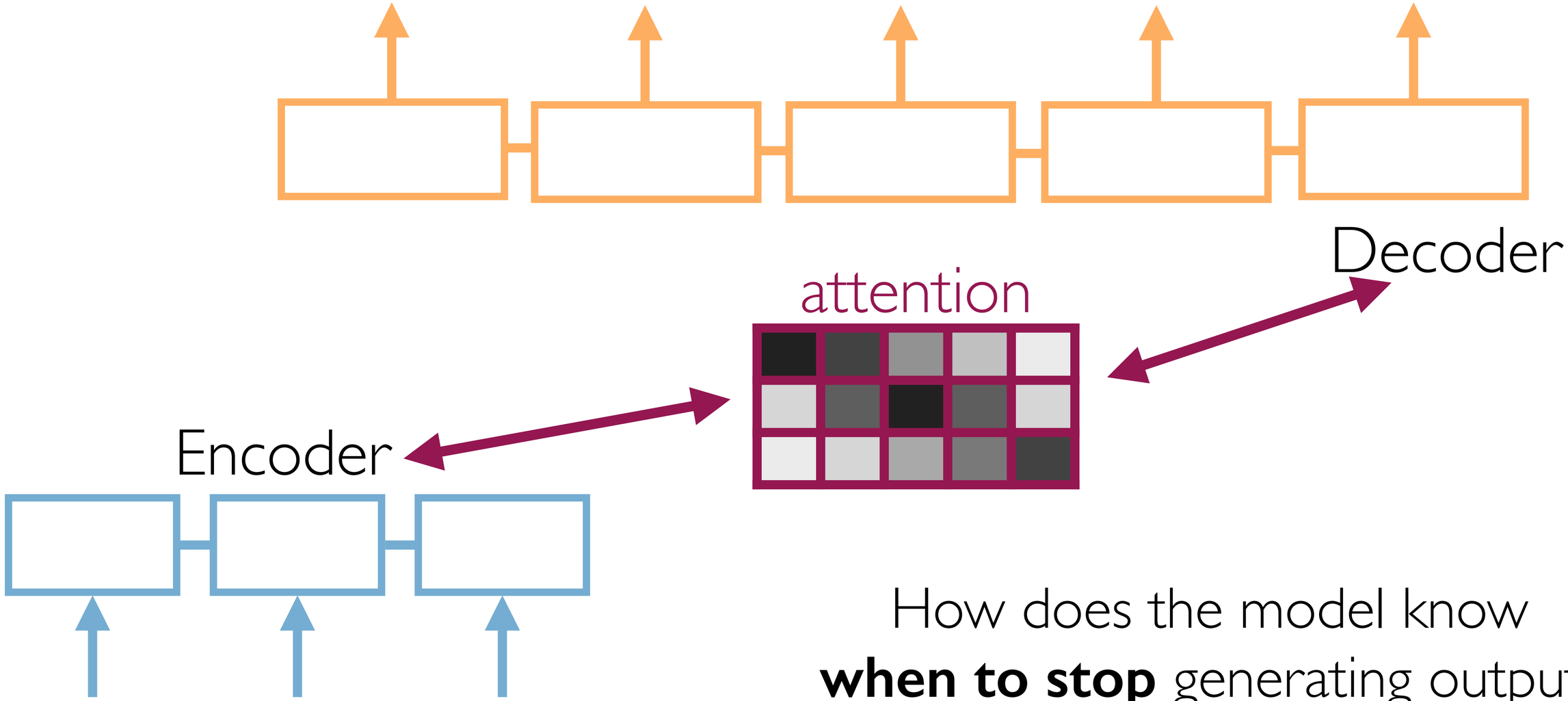
Decoder



Encoder



Encoder-decoder with attention



How does the model know **when to stop** generating output?

Terminology

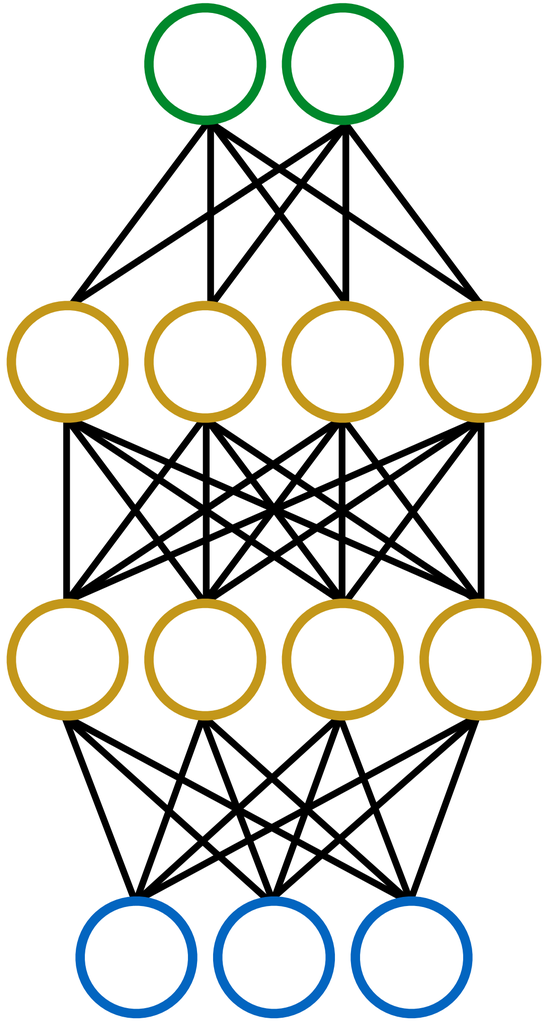
- encoder
- decoder
- attention

Orientation

- Input features
  - the model should **learn input feature engineering**
- Duration
  - **integrate** into the model
- Sequence modelling
  - enable the model to pass information between time steps - give it a **memory**
- Output features
  - allow output to **depend** on previous outputs

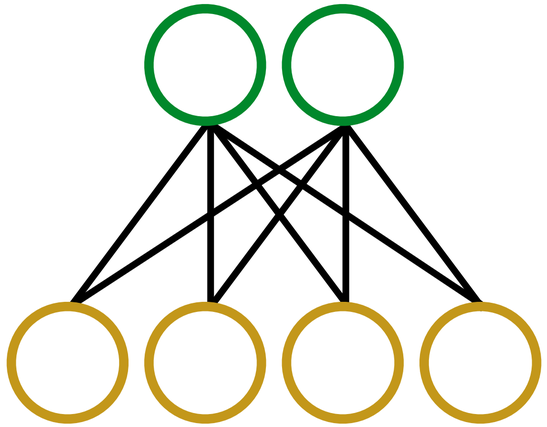
- Solution 1: attention
- Solution 2: explicit duration model

output time steps are frames (e.g., of a mel spectrogram)

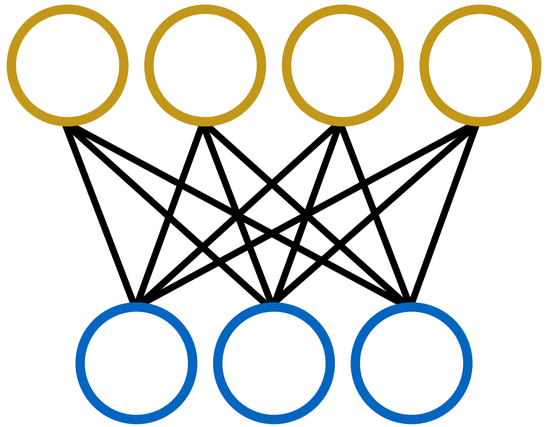


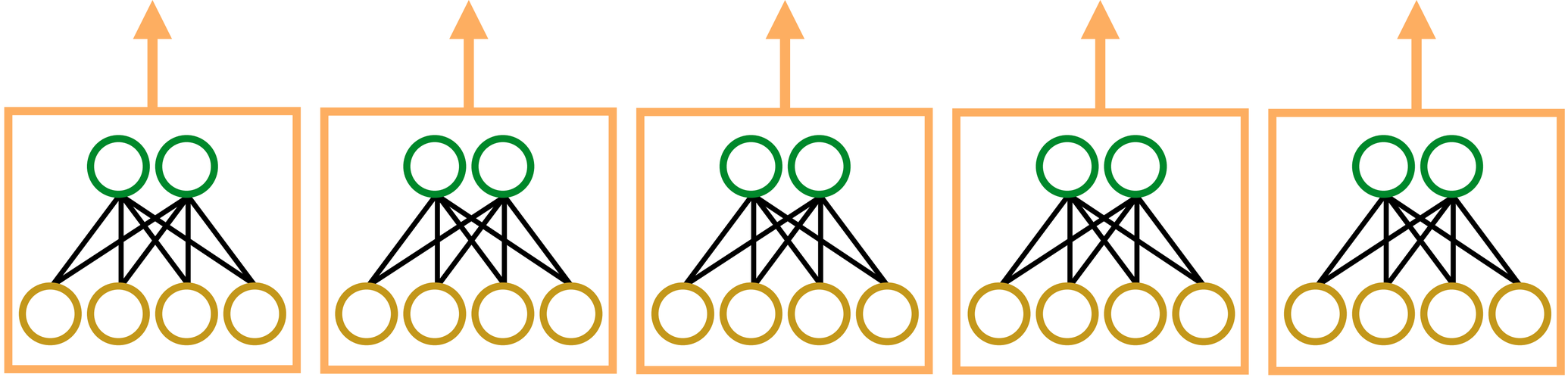
input time steps are linguistic units (e.g., phones)

Decoder

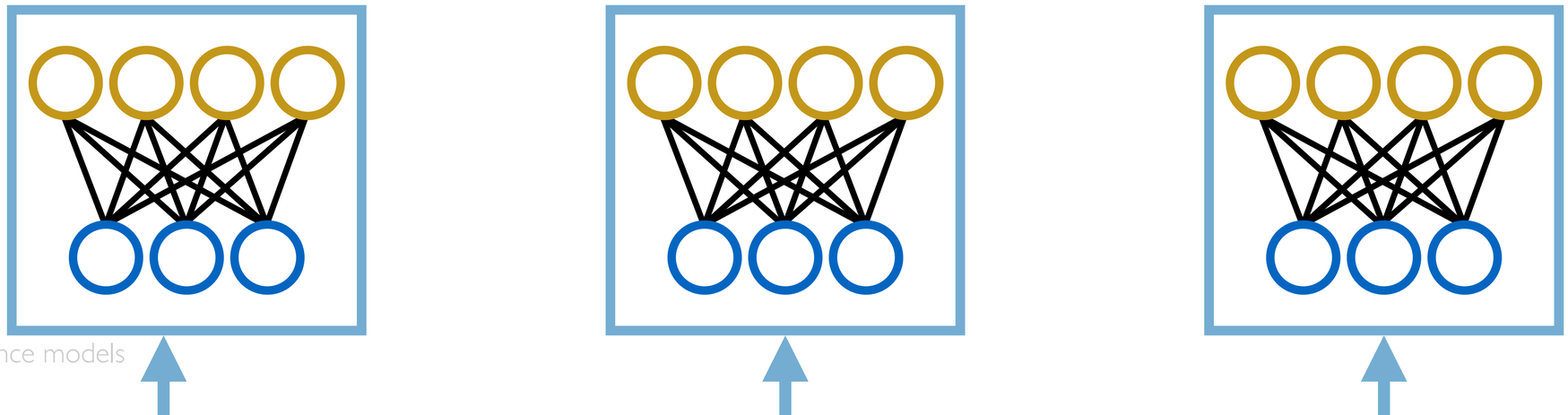


Encoder





predict an explicit duration
for each input time step



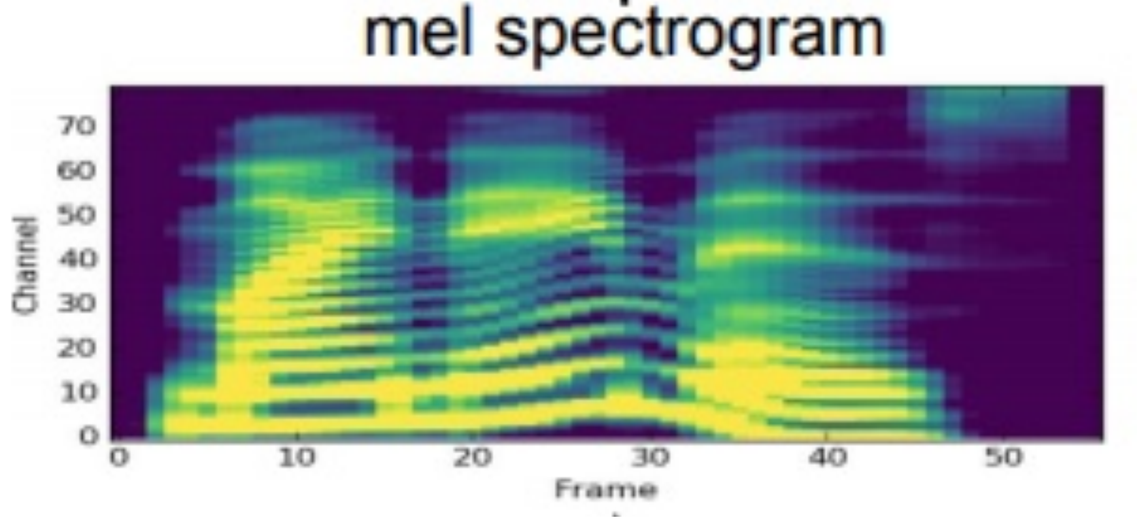
Orientation

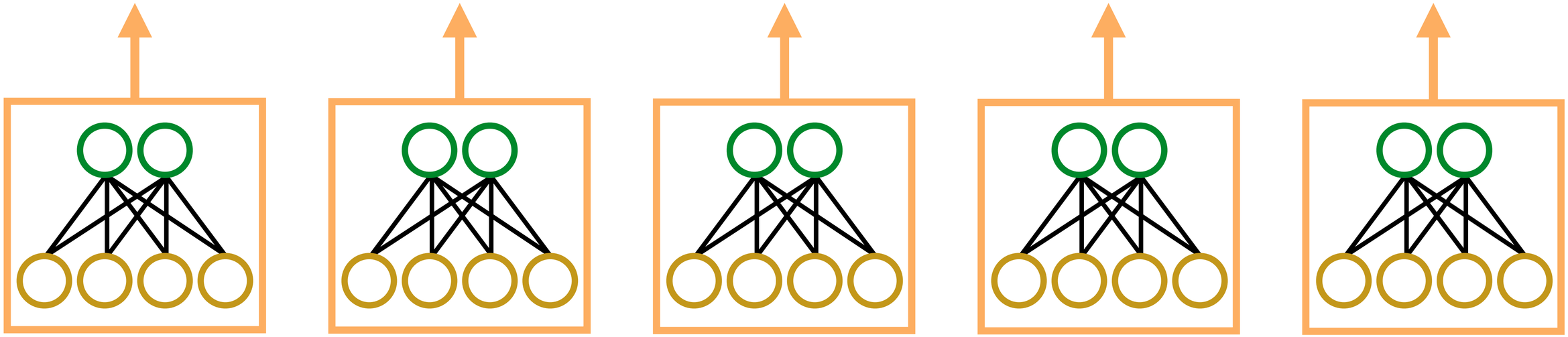
- Input features
  - the model should **learn input feature engineering**
- Duration
  - **integrate** into the model
- Sequence modelling
  - enable the model to pass information between time steps - give it a **memory**
- Output features
  - allow output to **depend** on previous outputs

- Solution 1: attention
- Solution 2: explicit duration model

Orientation

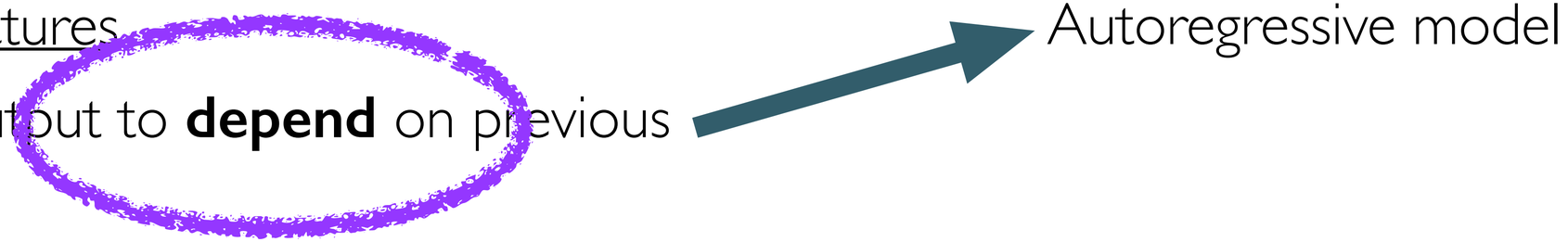
- Input features
  - the model should **learn input feature engineering**
- Duration
  - **integrate** into the model
- Sequence modelling
  - enable the model to pass information between time steps - give it a **memory**
- Output features
  - allow output to **depend** on previous outputs


mel spectrogram
Channel
70
60
50
40
30
20
10
0
0
10
20
30
40
50
Frame



Orientation

- Input features
  - the model should **learn input feature engineering**
- Duration
  - **integrate** into the model
- Sequence modelling
  - enable the model to pass information between time steps - give it a **memory**
- Output features
  - allow output to **depend** on previous outputs

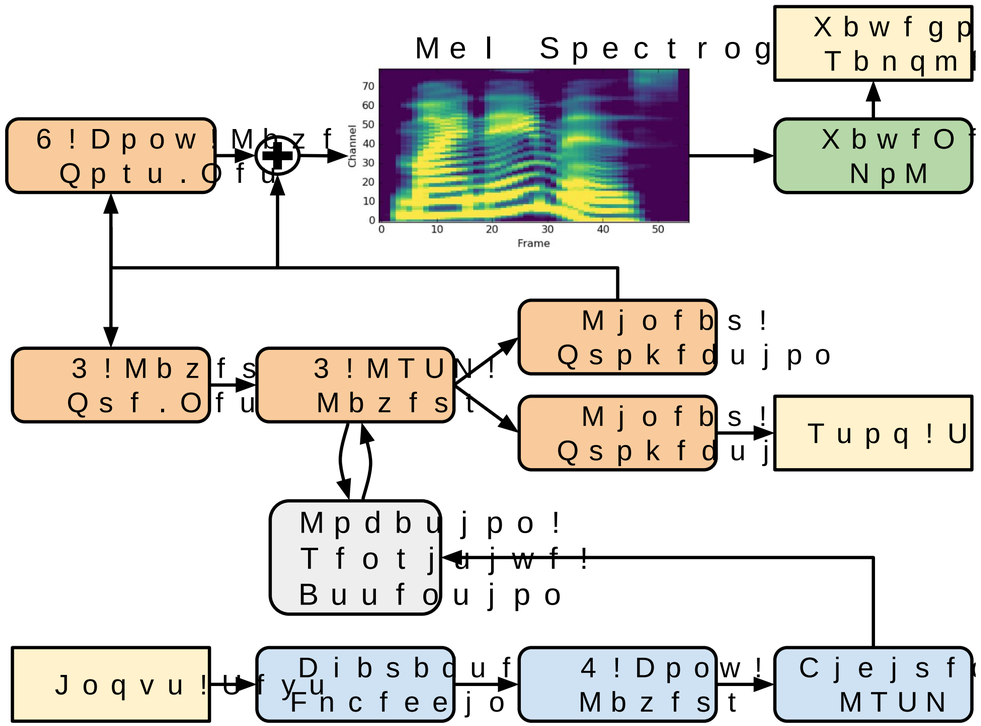


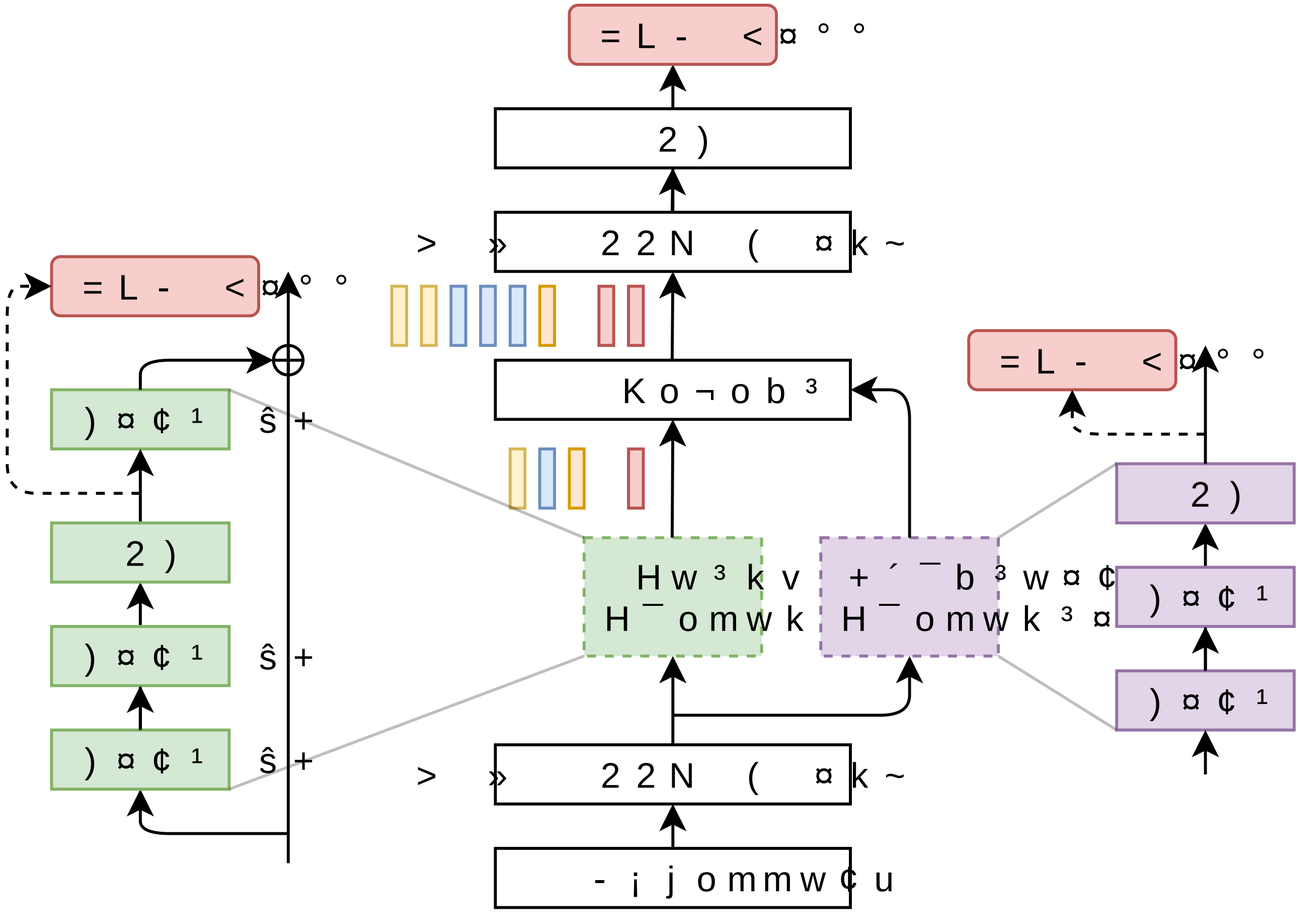
Coming next in this class

- Architecture diagrams

- Two case studies
  - Tacotron 2
  - FastPitch

Architecture diagrams





NATURAL TTS SYNTHESIS BY CONDITIONING WAVENET ON MEL SPECTROGRAM PREDICTIONS

Jonathan Shen[1], *Ruoming Pang*[1], *Ron J. Weiss*[1], *Mike Schuster*[1], *Navdeep Jaitly*[1], *Zongheng Yang*[*2], *Zhifeng Chen*[1], *Yu Zhang*[1], *Yuxuan Wang*[1], *RJ Skerry-Ryan*[1], *Rif A. Saurous*[1], *Yannis Agiomyrgiannakis*[1], *and Yonghui Wu*[1]

[1]Google, Inc., [2]University of California, Berkeley,
{jonathanasdf,rpang,yonghui}@google.com

ABSTRACT

This paper describes Tacotron 2, a neural network architecture for speech synthesis directly from text. The system is composed of a recurrent sequence-to-sequence feature prediction network that maps character embeddings to mel-scale spectrograms, followed by a modified WaveNet model acting as a vocoder to synthesize time-domain waveforms from those spectrograms. Our model achieves a mean opinion score (MOS) of 4.53 comparable to a MOS of 4.58 for professionally recorded speech. To validate our design choices, we present ablation studies of key components of our system and evaluate the impact of using mel spectrograms as the conditioning input to WaveNet instead of linguistic, duration, and F_0 features. We further show that using this compact acoustic intermediate representation allows for a significant reduction in the size of the WaveNet architecture.

Index Terms— Tacotron 2, WaveNet, text-to-speech

1. INTRODUCTION

Generating natural speech from text (text-to-speech synthesis, TTS) remains a challenging task despite decades of investigation [1]. Over time, different techniques have dominated the field. Concatenative synthesis with unit selection, the process of stitching small units of pre-recorded waveforms together [2, 3] was the state-of-the-art for many years. Statistical parametric speech synthesis [4, 5, 6, 7], which directly generates smooth trajectories of speech features to be synthesized by a vocoder, followed, solving many of the issues that concatenative synthesis had with boundary artifacts. However, the audio produced by these systems often sounds muffled and unnatural compared to human speech.

WaveNet [8], a generative model of time domain waveforms, produces audio quality that begins to rival that of real human speech and is already used in some complete TTS systems [9, 10, 11]. The inputs to WaveNet (linguistic features, predicted log fundamental frequency (F_0), and phoneme durations), however, require significant domain expertise to produce, involving elaborate text-analysis systems as well as a robust lexicon (pronunciation guide).

Tacotron [12], a sequence-to-sequence architecture [13] for producing magnitude spectrograms from a sequence of characters, simplifies the traditional speech synthesis pipeline by replacing the production of these linguistic and acoustic features with a single neural network trained from data alone. To vocode the resulting magnitude spectrograms, Tacotron uses the Griffin-Lim algorithm [14] for phase estimation, followed by an inverse short-time Fourier transform. As the authors note, this was simply a placeholder for future neural vocoder approaches, as Griffin-Lim produces characteristic artifacts and lower audio quality than approaches like WaveNet.

In this paper, we describe a unified, entirely neural approach to speech synthesis that combines the best of the previous approaches: a sequence-to-sequence Tacotron-style model [12] that generates mel spectrograms, followed by a modified WaveNet vocoder [10, 15]. Trained directly on normalized character sequences and corresponding speech waveforms, our model learns to synthesize natural sounding speech that is difficult to distinguish from real human speech.

Deep Voice 3 [11] describes a similar approach. However, unlike our system, its naturalness has not been shown to rival that of human speech. Char2Wav [16] describes yet another similar approach to end-to-end TTS using a neural vocoder. However, they use different intermediate representations (traditional vocoder features) and their model architecture differs significantly.

2. MODEL ARCHITECTURE

Our proposed system consists of two components, shown in Figure 1: (1) a recurrent sequence-to-sequence feature prediction network with attention which predicts a sequence of mel spectrogram frames from an input character sequence, and (2) a modified version of WaveNet which generates time-domain waveform samples conditioned on the predicted mel spectrogram frames.

2.1. Intermediate Feature Representation

In this work we choose a low-level acoustic representation: mel-frequency spectrograms, to bridge the two components. Using a representation that is easily computed from time-domain waveforms allows us to train the two components separately. This representation is also smoother than waveform samples and is easier to train using a squared error loss because it is invariant to phase within each frame.

A mel-frequency spectrogram is related to the linear-frequency spectrogram, i.e., the short-time Fourier transform (STFT) magnitude. It is obtained by applying a nonlinear transform to the frequency axis of the STFT, inspired by measured responses from the human auditory system, and summarizes the frequency content with fewer dimensions. Using such an auditory frequency scale has the effect of emphasizing details in lower frequencies, which are critical to speech intelligibility, while de-emphasizing high frequency details, which are dominated by fricatives and other noise bursts and generally do not need to be modeled with high fidelity. Because of these properties, features derived from the mel scale have been used as an underlying representation for speech recognition for many decades [17].

*Work done while at Google.

ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) | 978-1-7281-7605-5/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICASSP39728.2021.9413889

FASTPITCH: PARALLEL TEXT-TO-SPEECH WITH PITCH PREDICTION

Adrian Łańcucki

NVIDIA Corporation

ABSTRACT

We present FastPitch, a fully-parallel text-to-speech model based on FastSpeech, conditioned on fundamental frequency contours. The model predicts pitch contours during inference. By altering these predictions, the generated speech can be more expressive, better match the semantic of the utterance, and in the end more engaging to the listener. Uniformly increasing or decreasing pitch with FastPitch generates speech that resembles the voluntary modulation of voice. Conditioning on frequency contours improves the overall quality of synthesized speech, making it comparable to state-of-the-art. It does not introduce an overhead, and FastPitch retains the favorable, fully-parallel Transformer architecture, with over $900\times$ real-time factor for mel-spectrogram synthesis of a typical utterance.

Index Terms— text-to-speech, speech synthesis, fundamental frequency

1. INTRODUCTION

Recent advances in neural text-to-speech (TTS) enabled real-time synthesis of naturally sounding, human-like speech. Parallel models are able to synthesize mel-spectrograms orders of magnitude faster than autoregressive ones, either by relying on external alignments [1], or aligning themselves [2]. TTS models can be conditioned on qualities of speech such as linguistic features and fundamental frequency [3]. The latter has been repeatedly shown to improve the quality of neural, but also concatenative models [4, 5]. Conditioning on F_0 is a common approach to adding singing capabilities [6]. or adapting to other speakers [5].

In this paper we propose FastPitch, a feed-forward model based on FastSpeech that improves the quality of synthesized speech. By conditioning on fundamental frequency estimated for every input symbol, which we refer to simply as a pitch contour, it matches the state-of-the-art autoregressive TTS models. We show that explicit modeling of such pitch contours addresses the quality shortcomings of the plain feed-forward Transformer architecture. These most likely arise from collapsing different pronunciations of the same phonetic units in the absence of enough linguistic information in the textual input alone. Conditioning on fundamental frequency also improves convergence, and eliminates the need for knowledge distillation of mel-spectrogram targets used in FastSpeech. We would like to note that a concurrently developed FastSpeech 2 [7] describes a similar approach.

Combined with WaveGlow [8], FastPitch is able to synthesize mel-spectrograms over $60\times$ faster than real-time, without resorting to kernel-level optimizations [9]. Because the model learns to predict and use pitch in a low resolution of one value for every input symbol, it makes it easy to adjust pitch interactively, enabling practical applications in pitch editing. Constant offsetting of F_0 with FastPitch produces naturally sounding low- and high-pitched variations of voice that preserve the perceived speaker identity. We conclude that the model learns to mimic the action of vocal chords, which happens during the voluntary modulation of voice.

2. MODEL DESCRIPTION

The architecture of FastPitch is shown in Figure 1. It is based on FastSpeech and composed mainly of two feed-forward Transformer (FFTr) stacks [1]. The first one operates in the resolution of input tokens, the second one in the resolution of the output frames. Let $\boldsymbol{x} = (x_1, \ldots, x_n)$ be the sequence of input lexical units, and $\boldsymbol{y} = (y_1, \ldots, y_t)$ be the sequence of target mel-scale spectrogram frames. The first FFTr stack produces the hidden representation $\boldsymbol{h} = \text{FFTr}(\boldsymbol{x})$. The hidden representation $\boldsymbol{h}$ is used to make predictions about the duration and average pitch of every character with a 1-D CNN

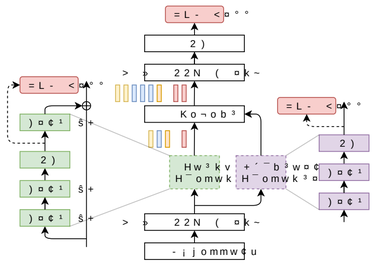
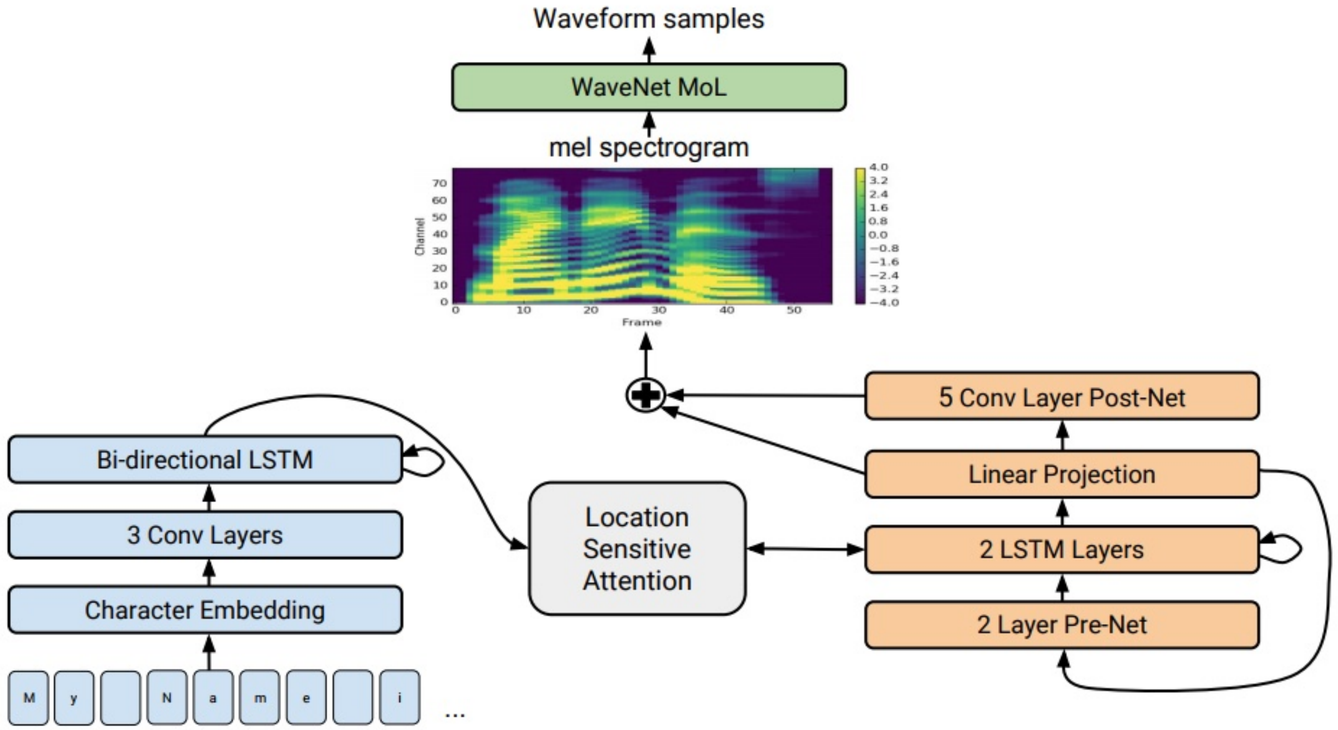


Fig. 1. **Architecture of FastPitch** follows FastSpeech [1]. A single pitch value is predicted for every temporal location.

Understanding architecture diagrams



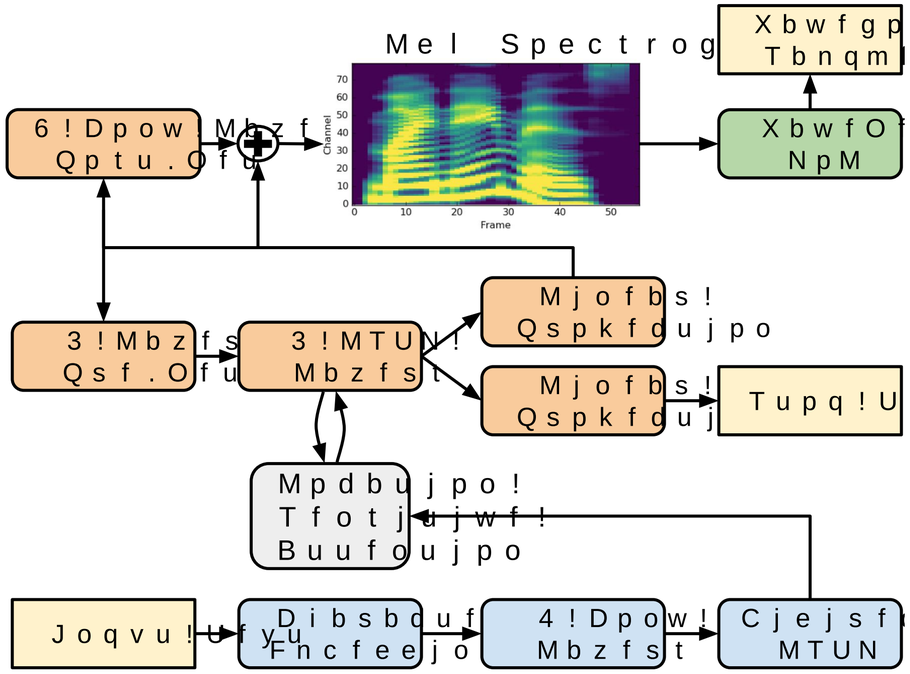
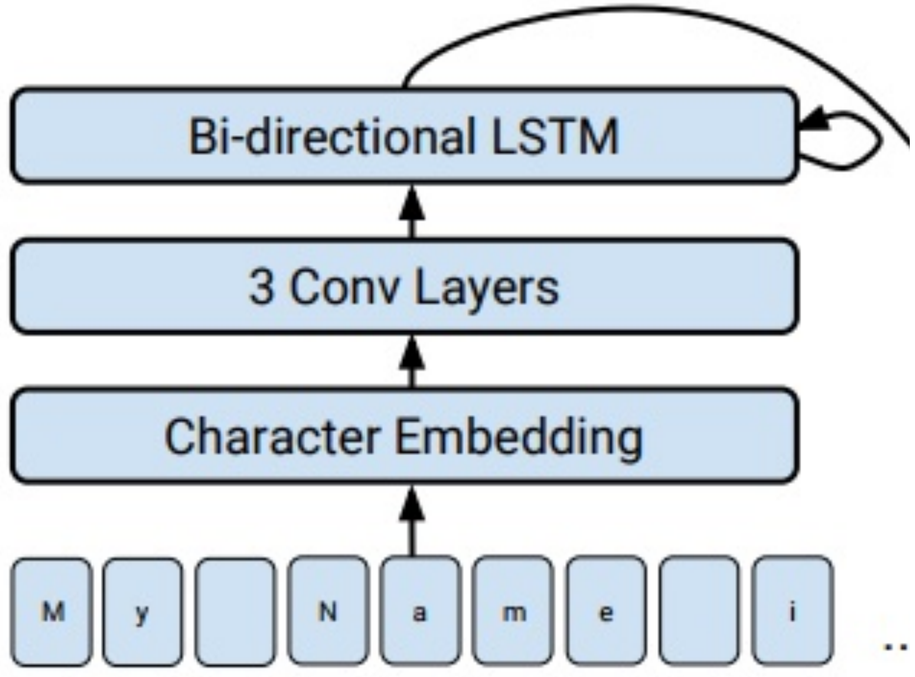
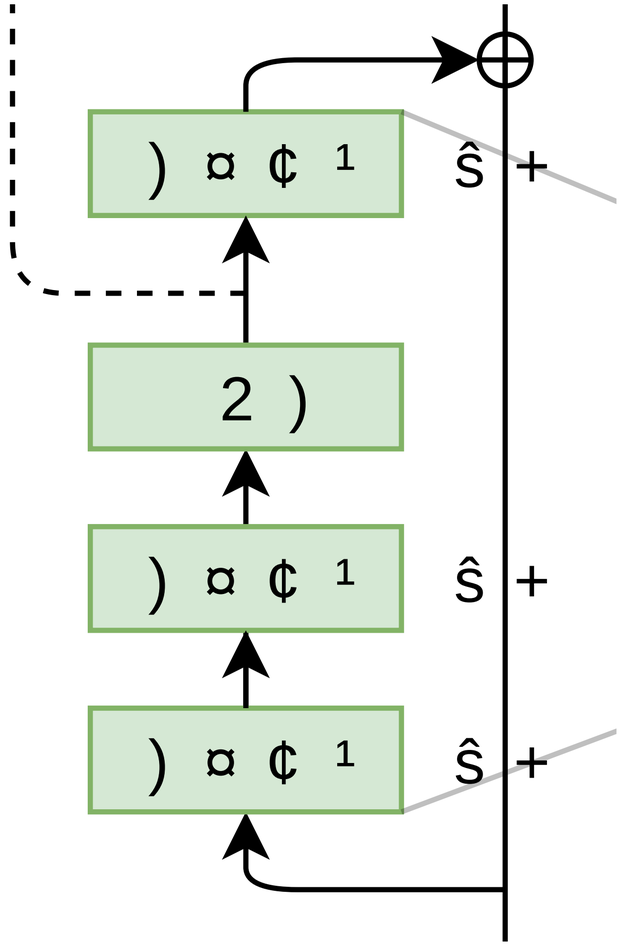


Fig. 1. Block diagram of the Tacotron 2 system architecture.

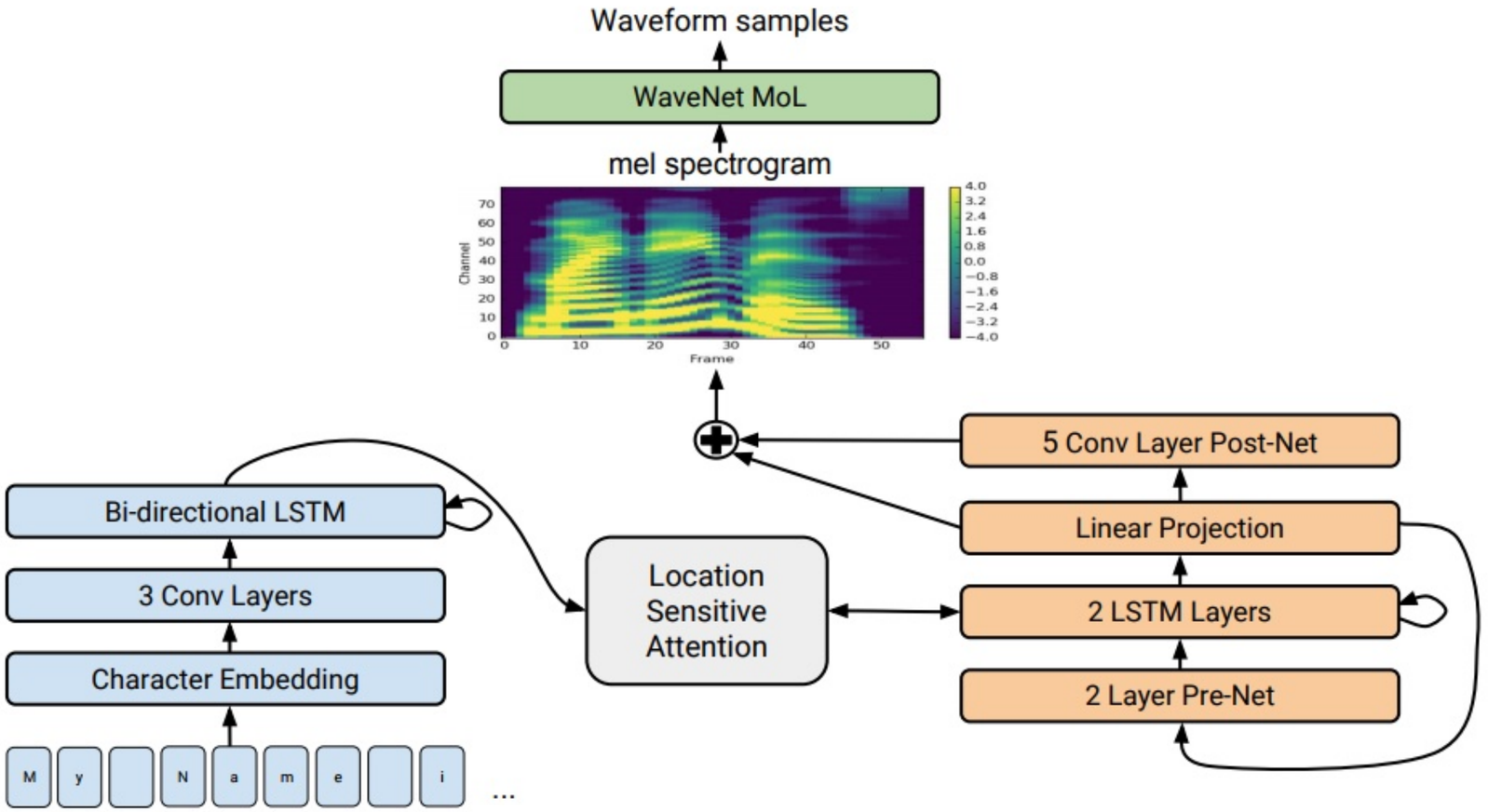
Stacking layers to create more powerful networks



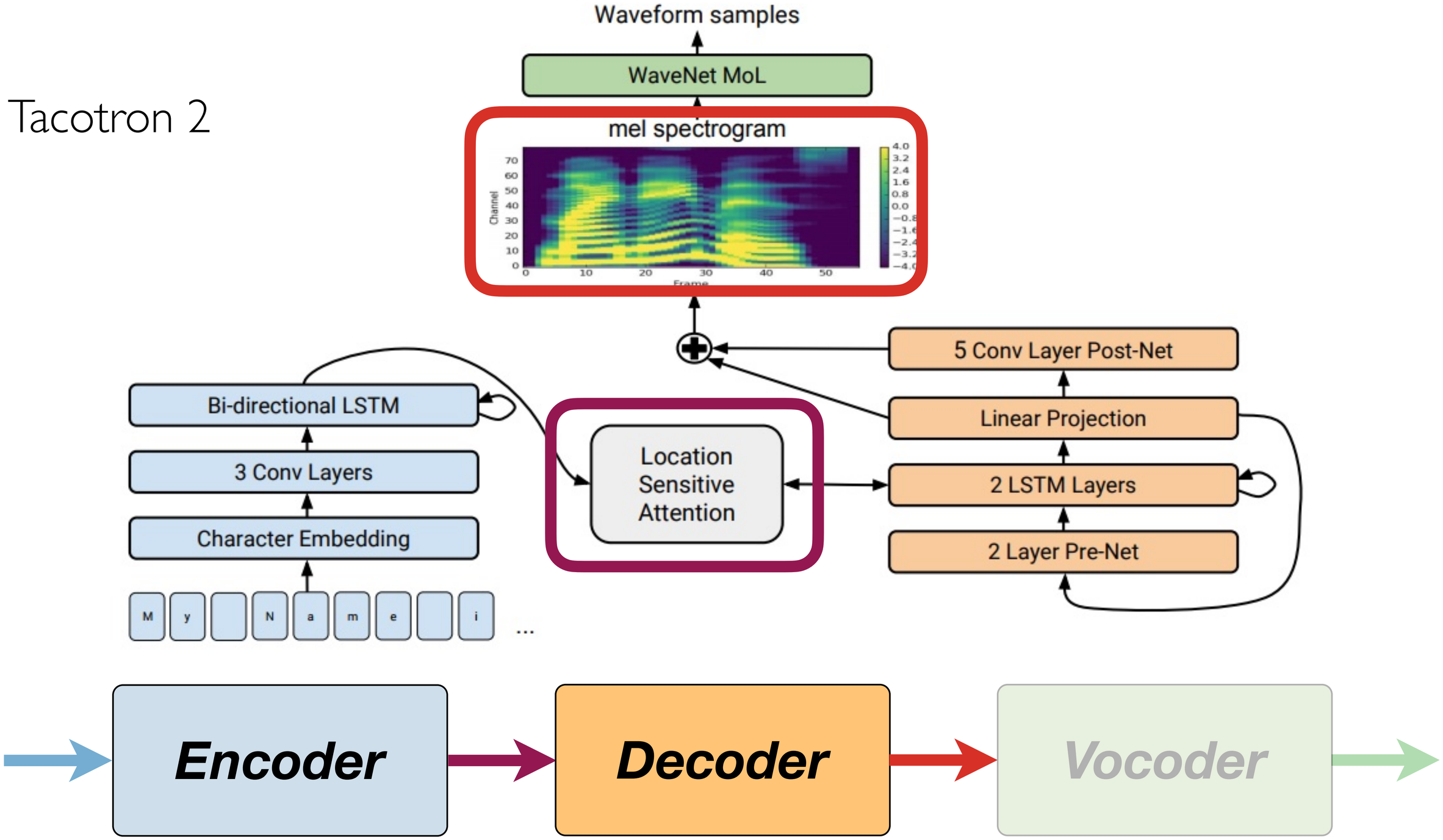


Case study

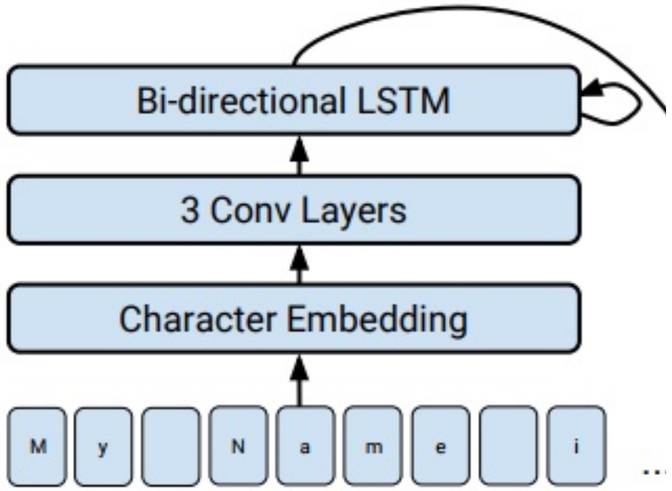
Tacotron 2



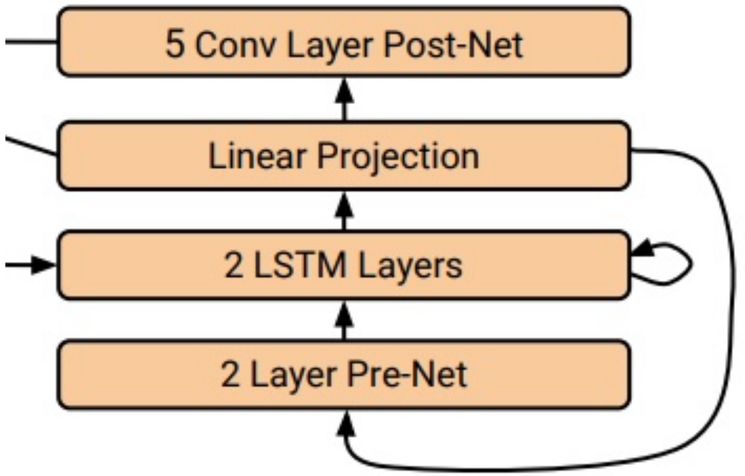
Tacotron 2

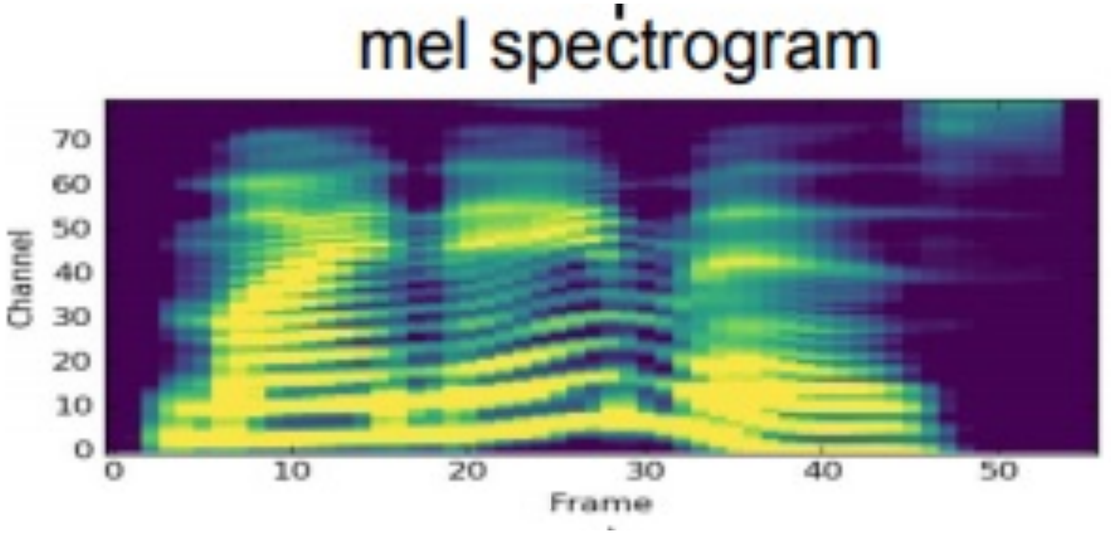

Waveform samples
WaveNet MoL
mel spectrogram
Channel
Frame
5 Conv Layer Post-Net
Linear Projection
2 LSTM Layers
2 Layer Pre-Net
Location Sensitive Attention
Bi-directional LSTM
3 Conv Layers
Character Embedding
M
y
N
a
m
e
i
...
Encoder
Decoder
Vocoder

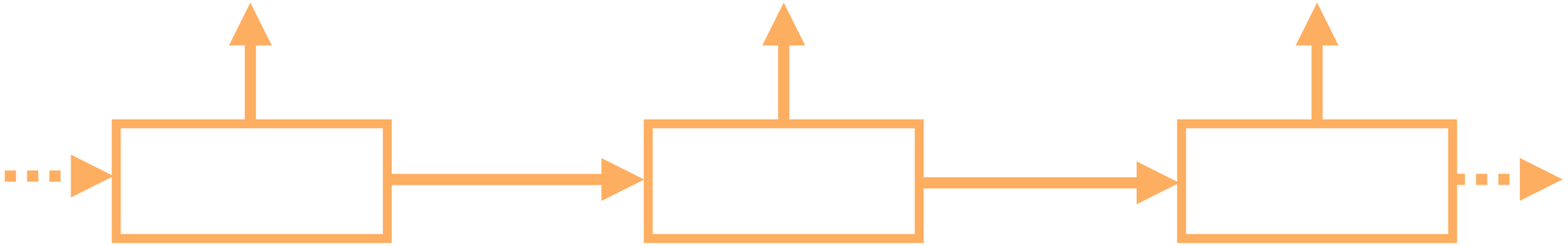
Tacotron 2

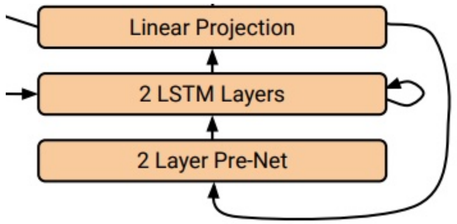

Bi-directional LSTM
3 Conv Layers
Character Embedding
M
y
N
a
m
e
i

Tacotron 2

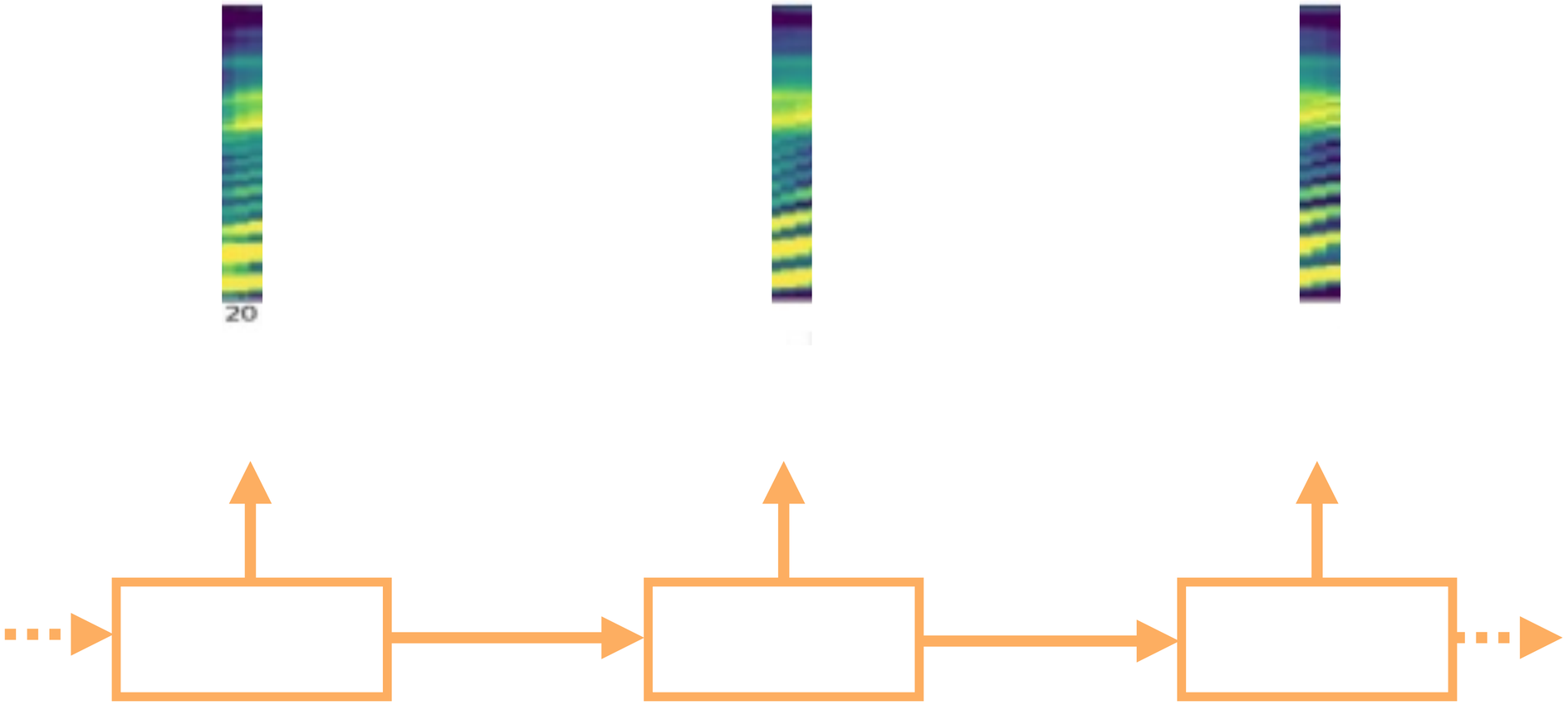

5 Conv Layer Post-Net
Linear Projection
2 LSTM Layers
2 Layer Pre-Net


mel spectrogram
Channel
70
60
50
40
30
20
10
0
0
10
20
30
40
50
Frame




Linear Projection
2 LSTM Layers
2 Layer Pre-Net

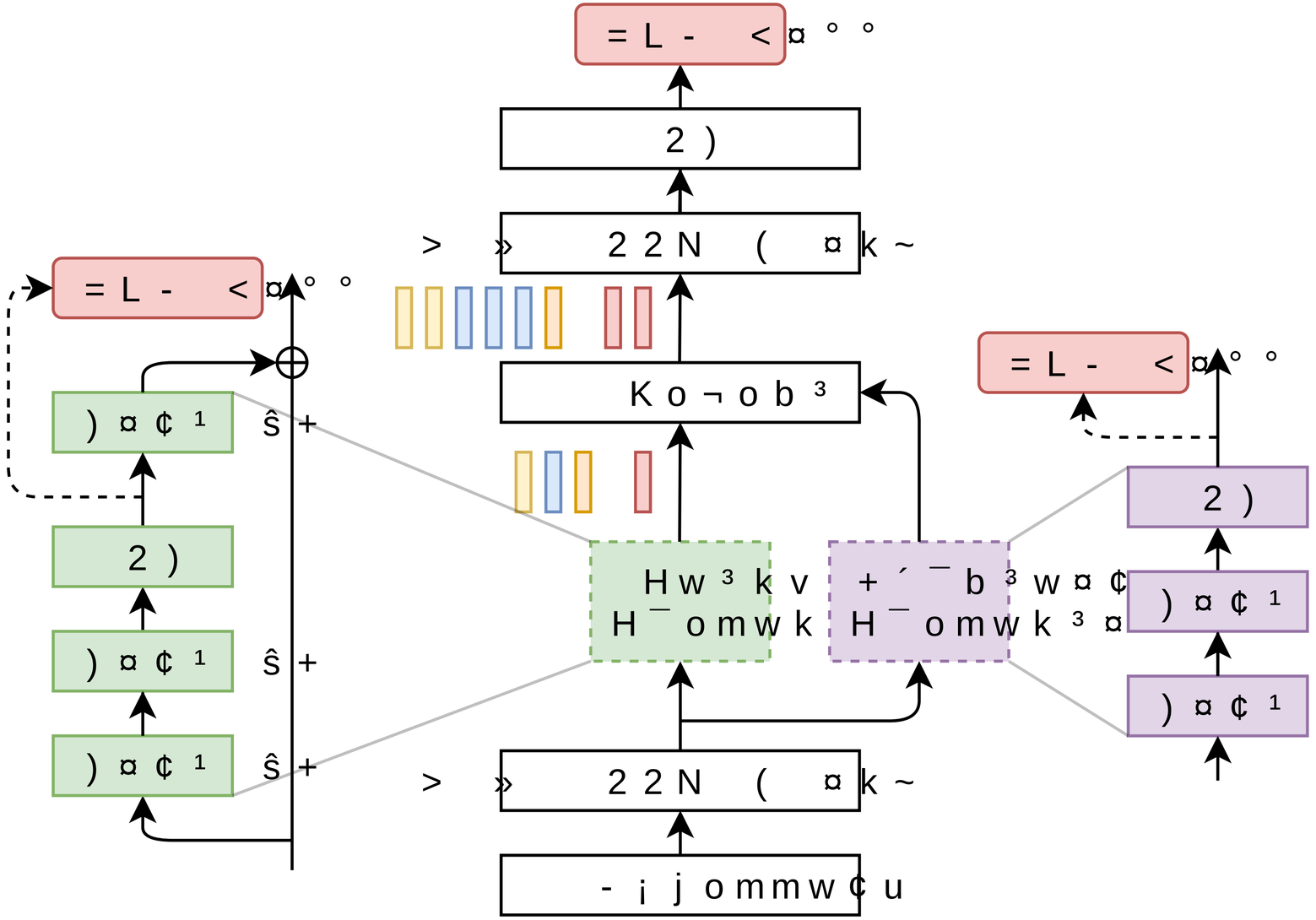
Encoder



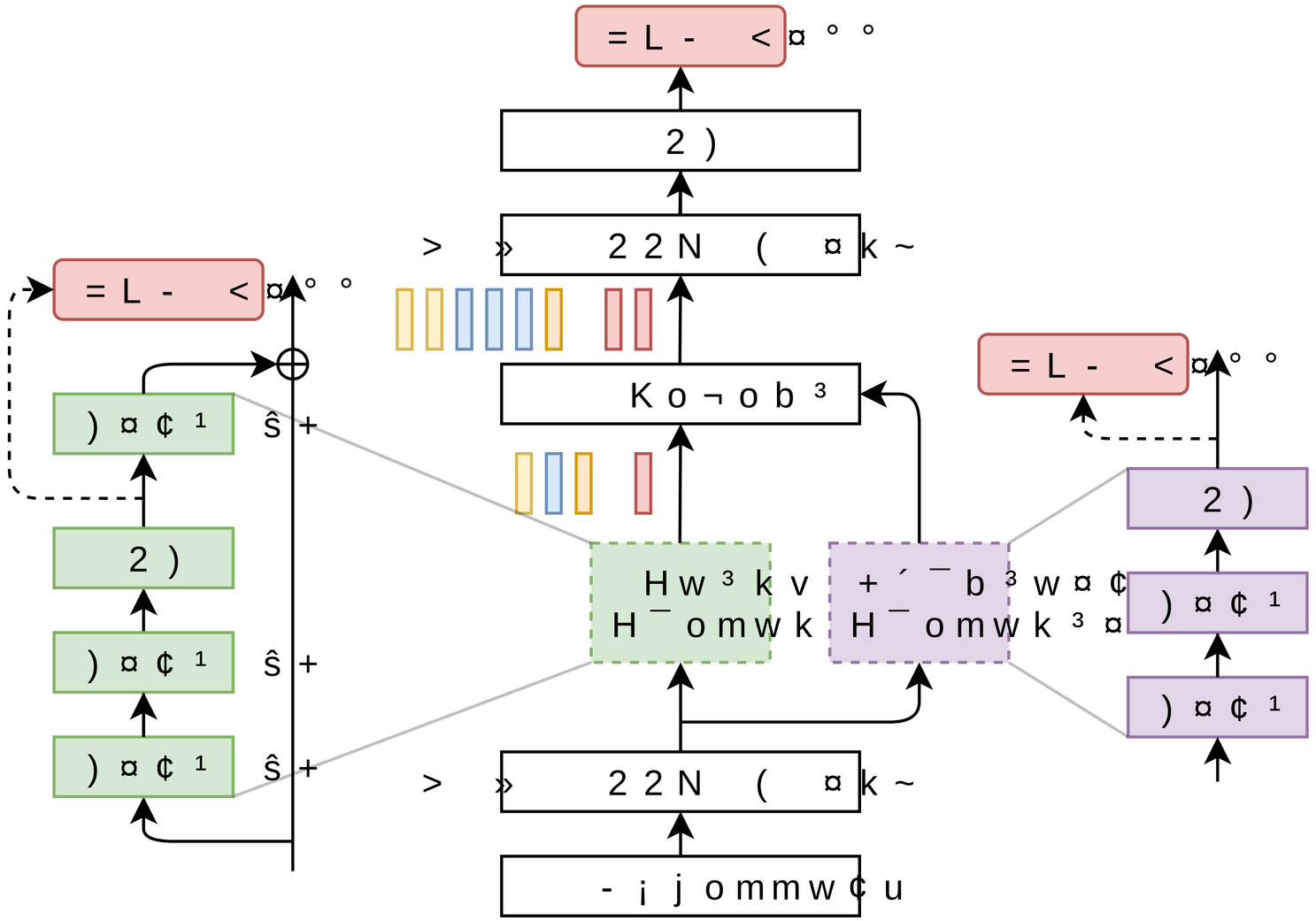
Encoder

Case study

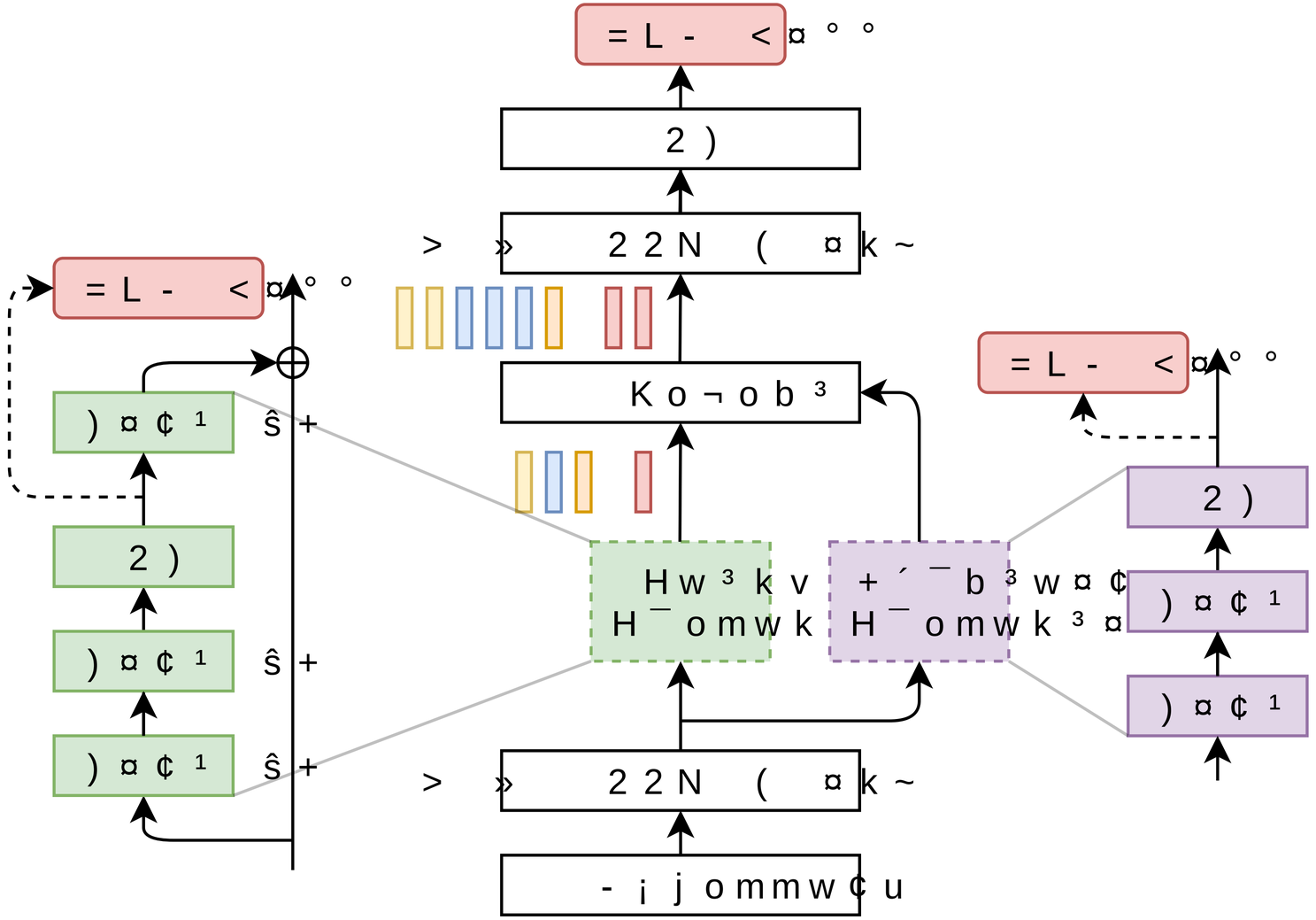
FastPitch



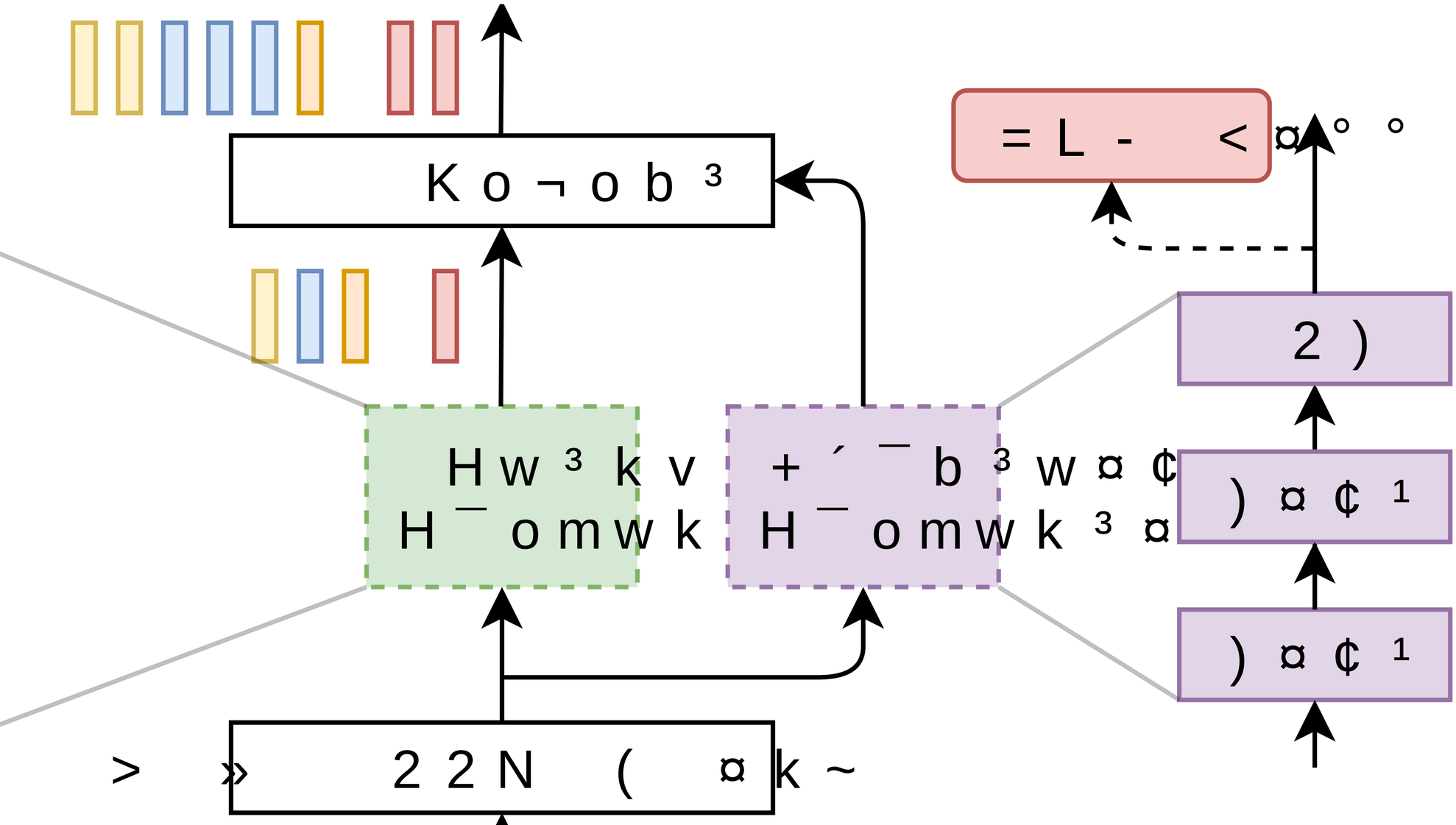
Input and output - add them to the diagram



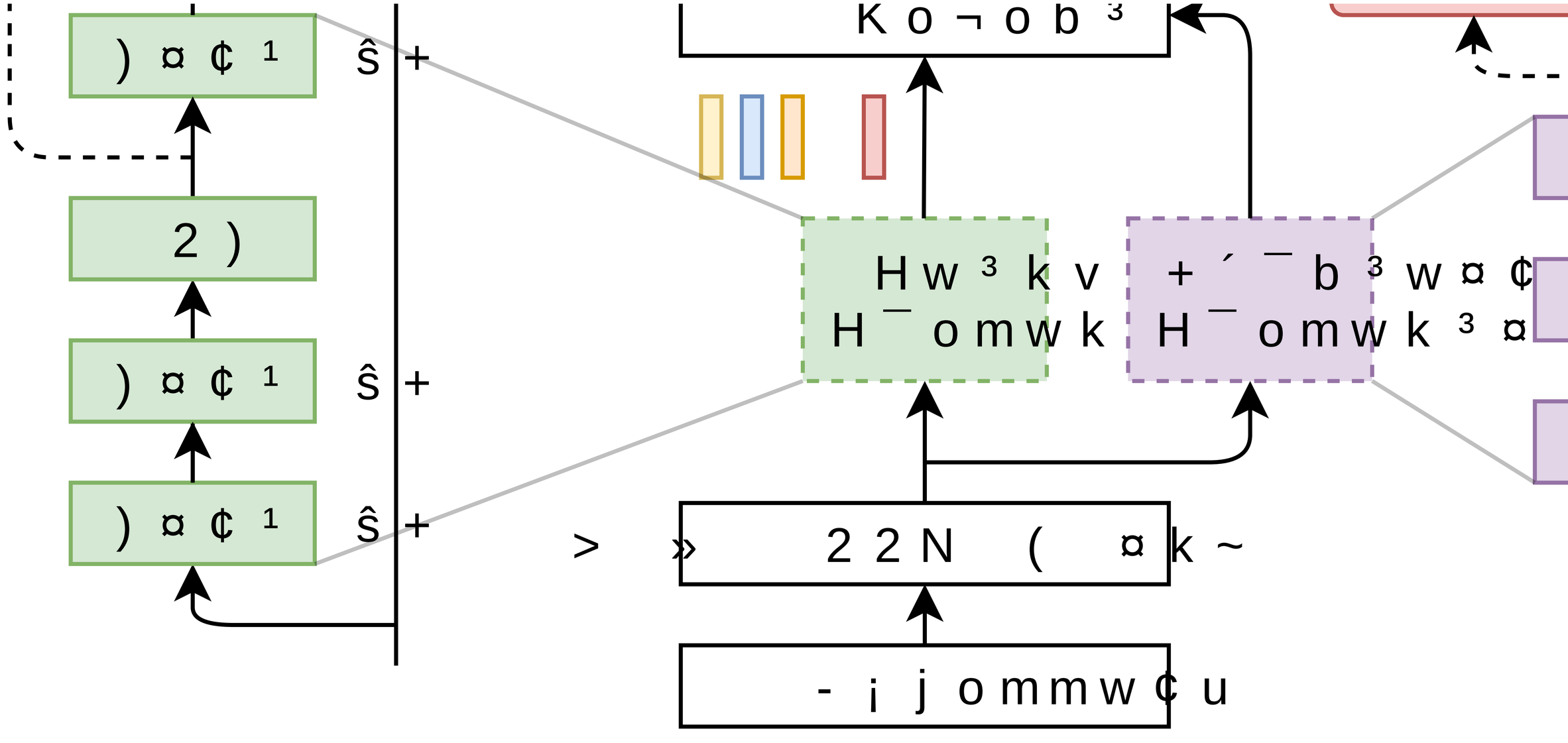
Sequences - describe the time steps and length of sequence at every point



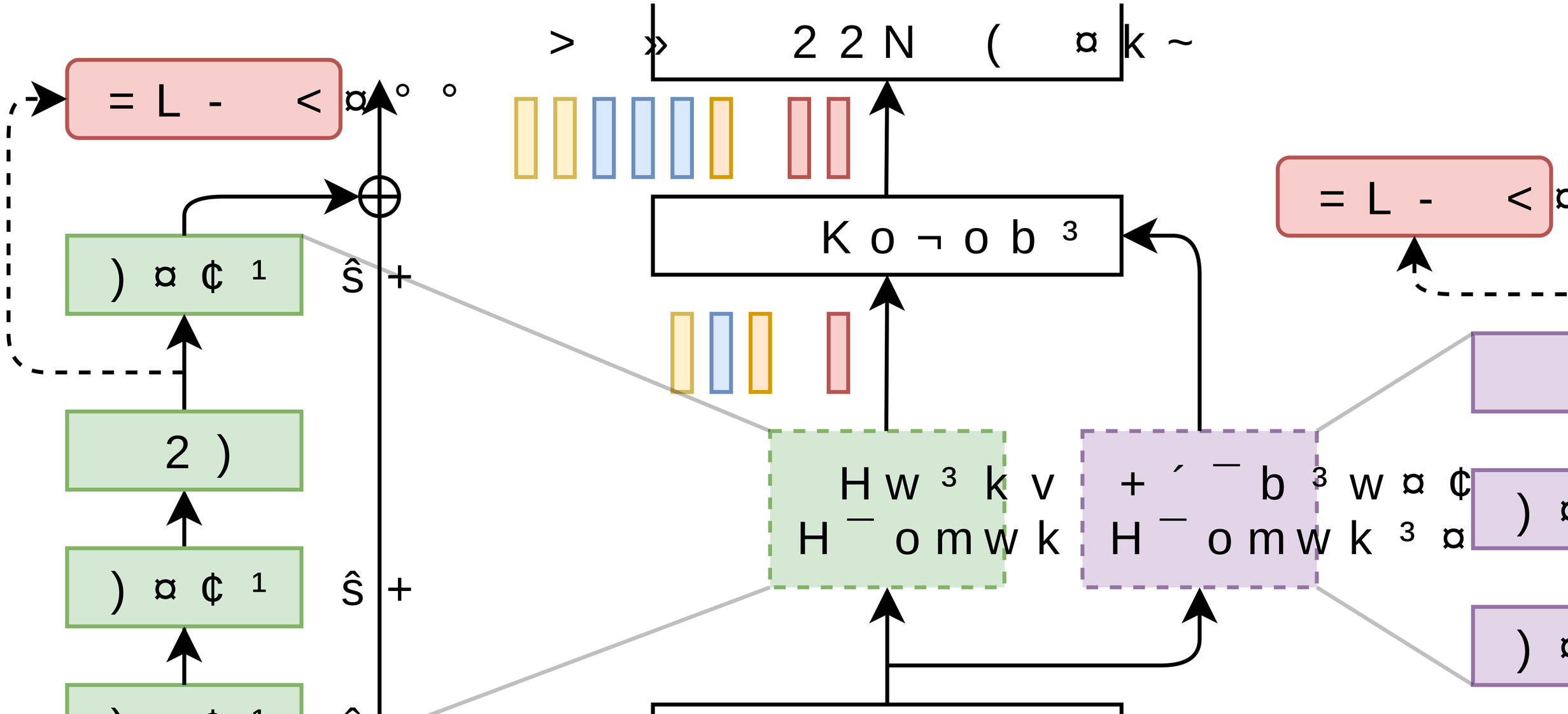
Duration - describe how it is predicted and used during inference



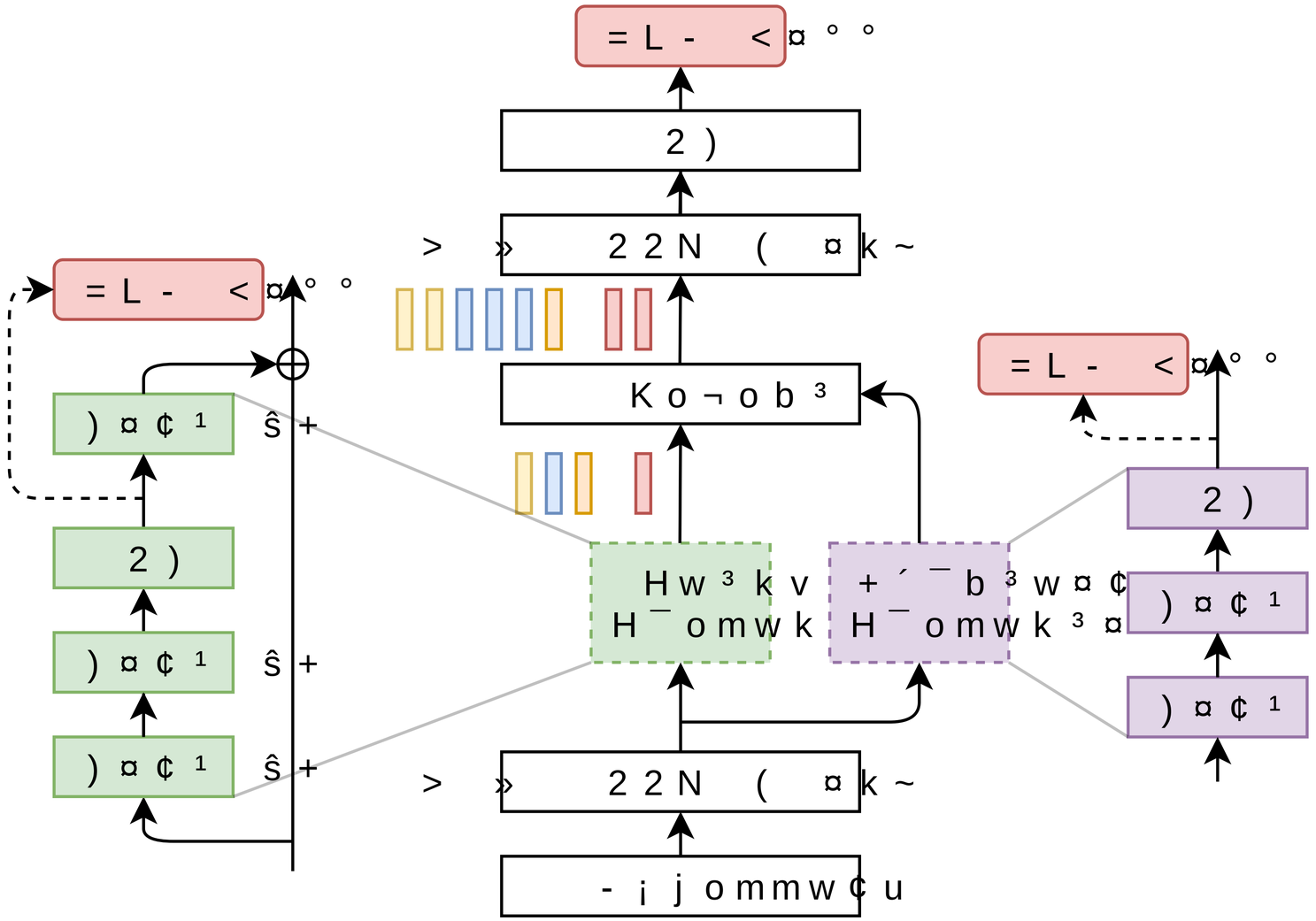
“Pitch” (F0!) - describe how it is **predicted** during inference



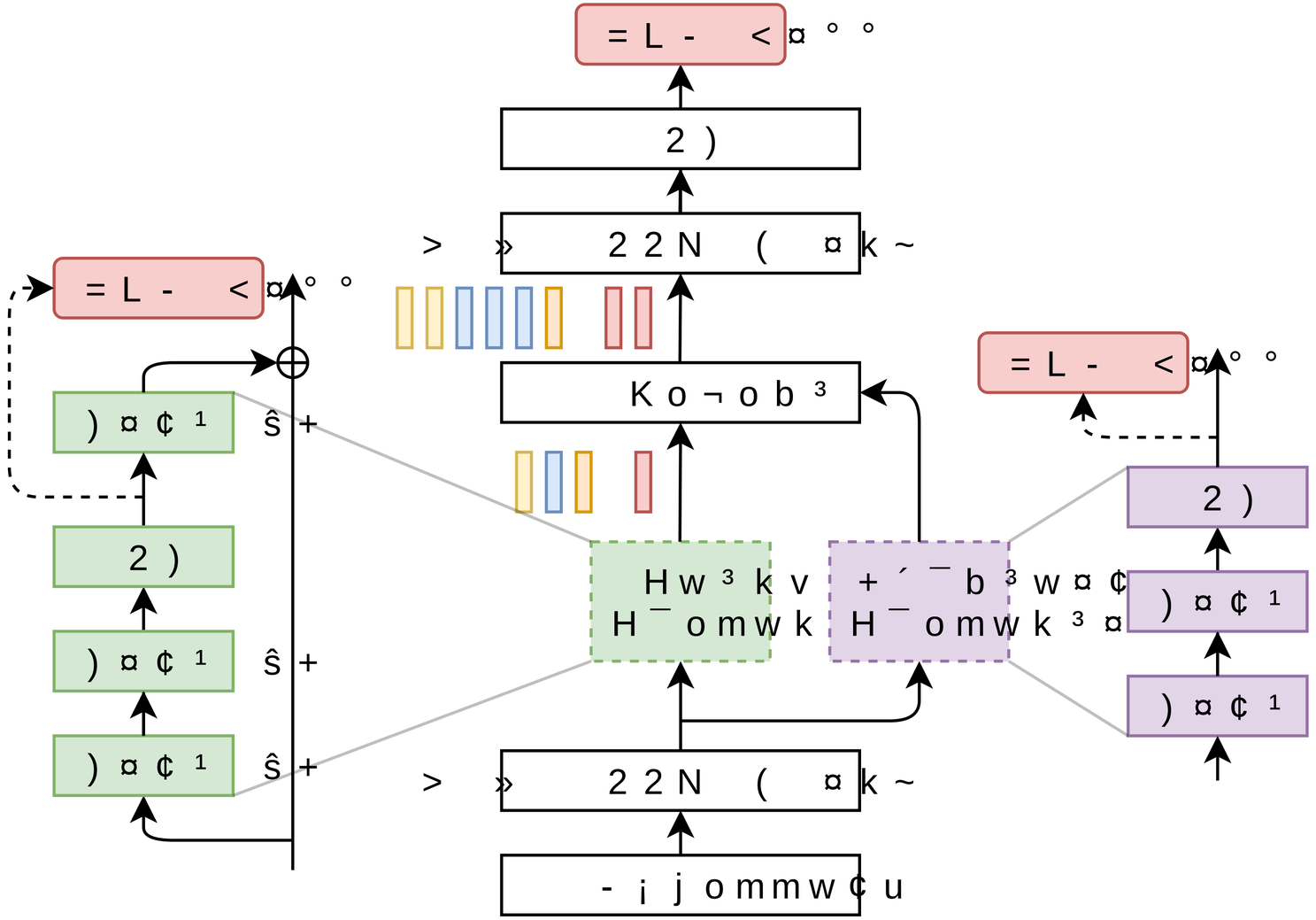
“Pitch” (F0!) - describe how it is **used** during inference



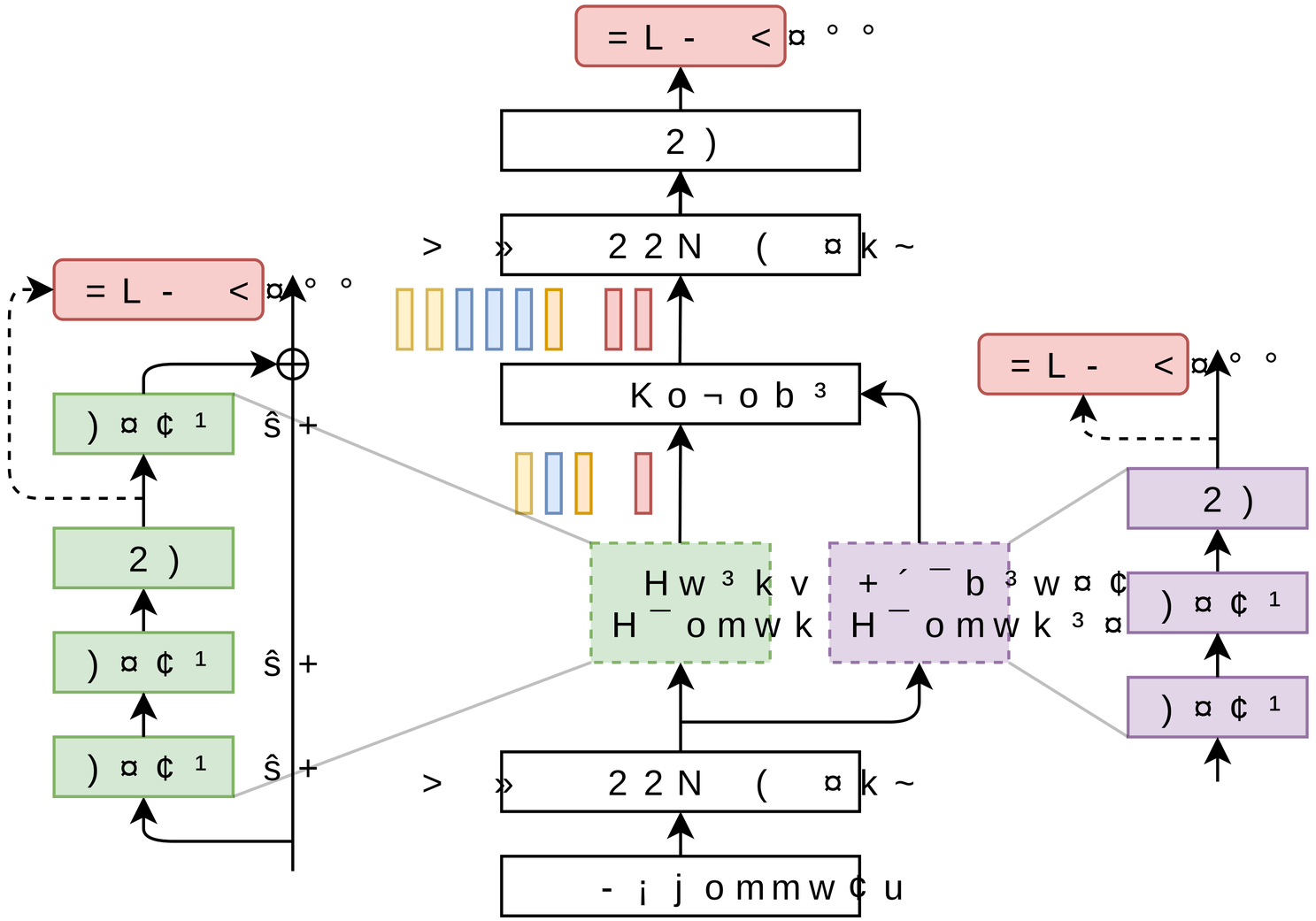
Representations - describe them all; which ones are learned?



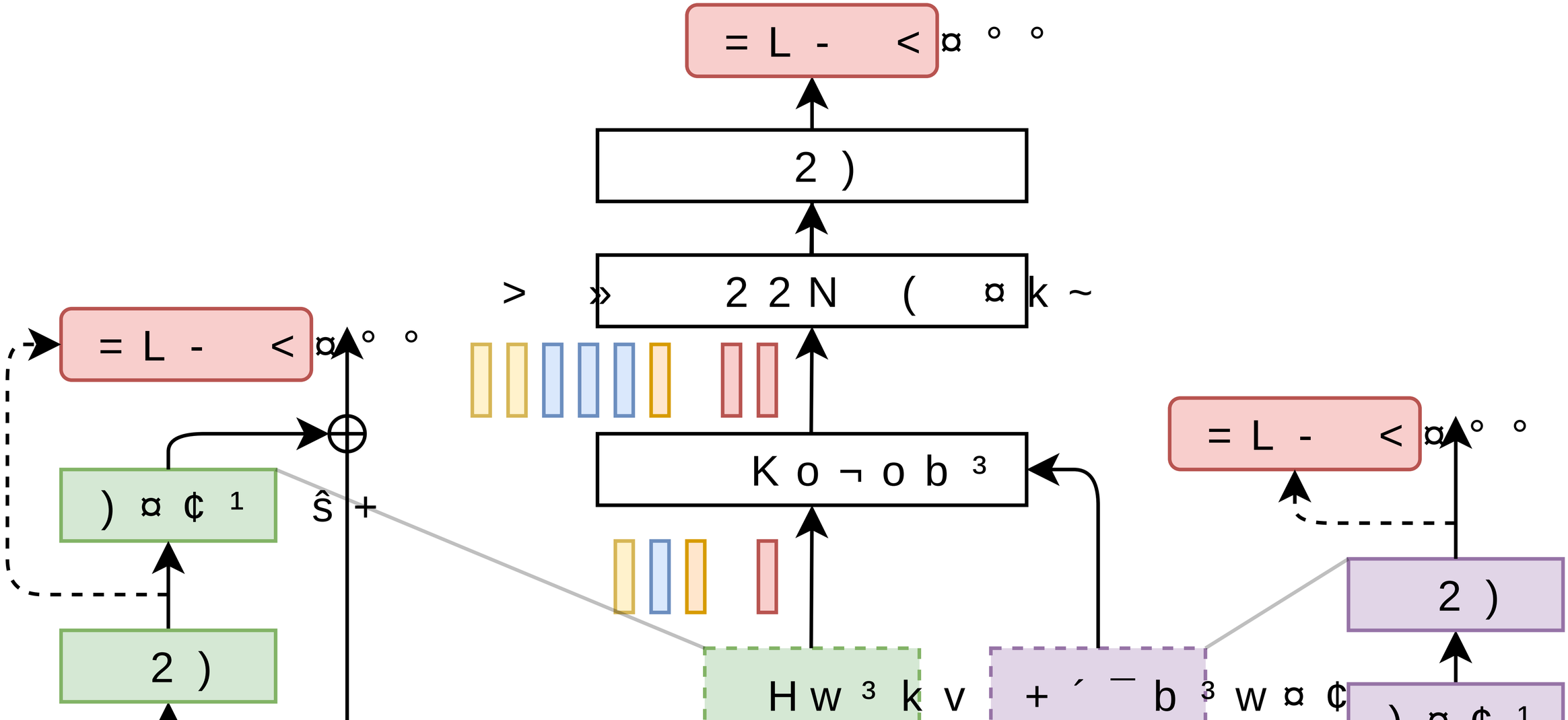
Combining representations - describe every point where this occurs



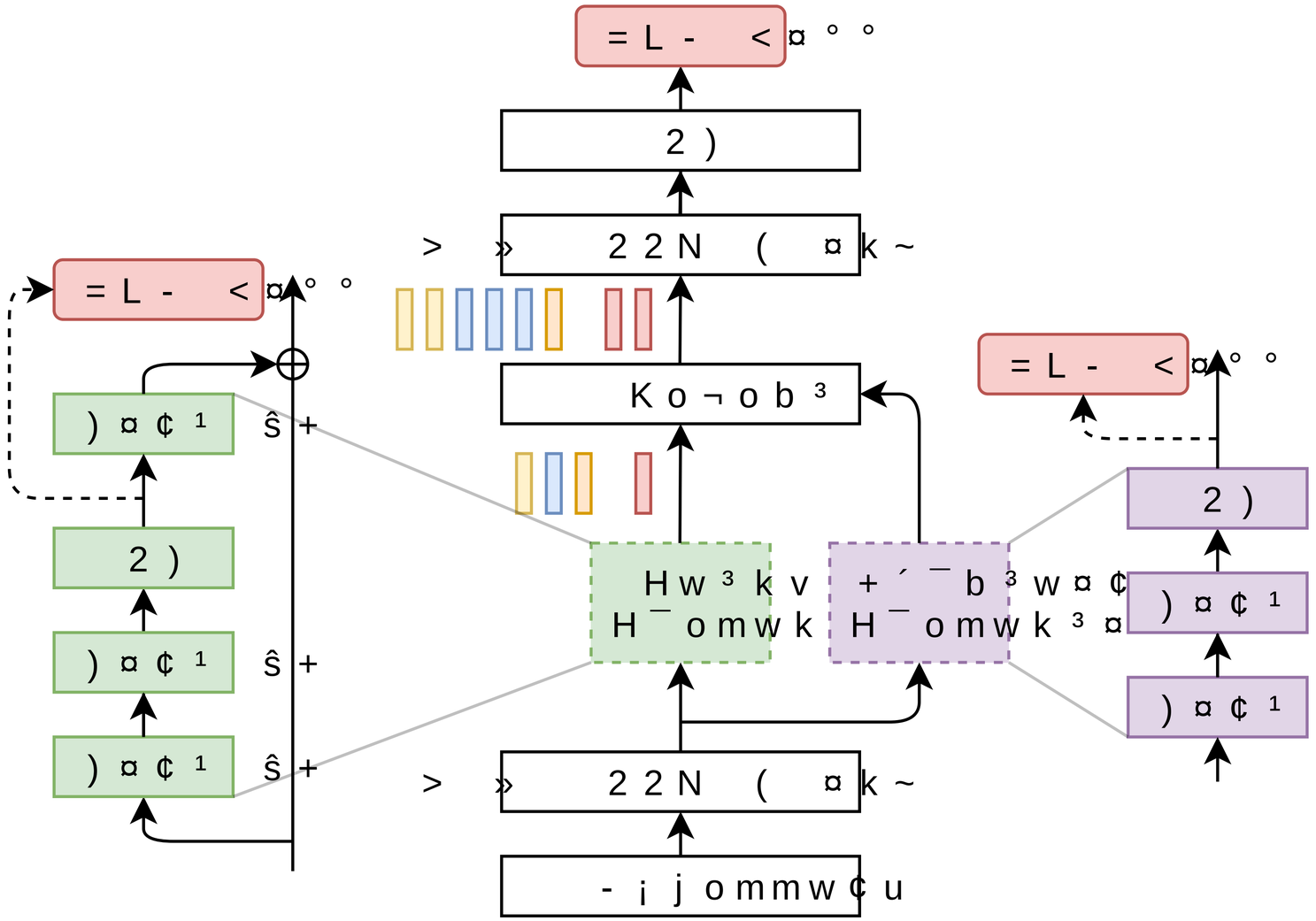
Training vs. **inference** - which one is this diagram describing ?



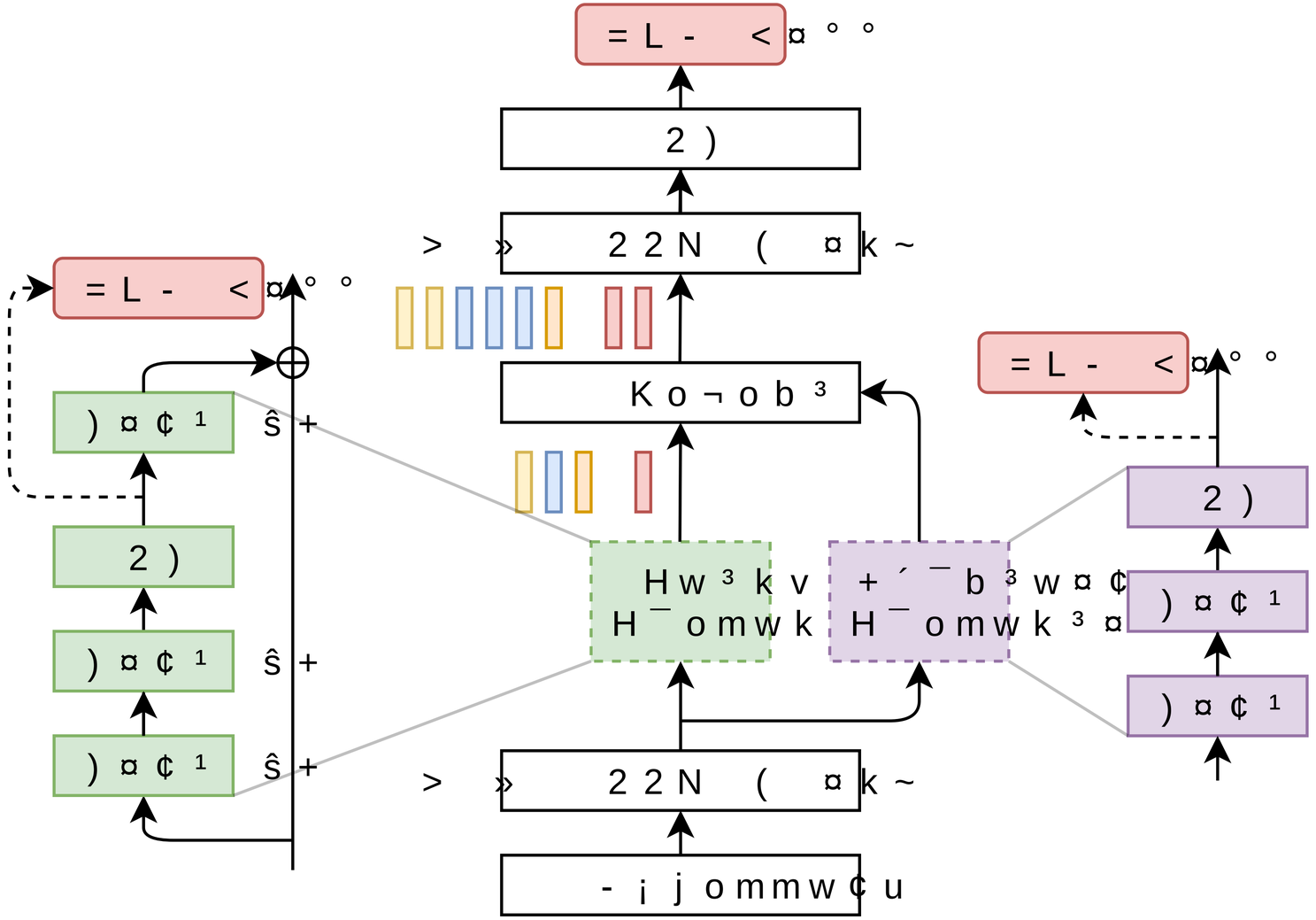
Training the model - what is “MSE Loss”; why are there 3 losses?



Training the model - how is the Pitch Predictor trained?



Training the model - how is the Duration Predictor trained?



Terminology & jargon - make sure you understand all terms

- architectures
  - fully parallel
  - recurrent
  - autoregressive
  - positional encoding
  - causal
- training methods
  - optimiser (e.g., Adam)
  - teacher forcing (for autoregressive models)
- input symbol, lexical unit, character, grapheme
- audio (*never* "audios" please!!)
- ground truth
- ablation study
- real-time factor (RTF)

What next?

- Neural speech processing
- Large Speech Language Models
- Beyond TTS