HW3P2 Bootcamp

Utterance to Phoneme Mapping using Sequence Models

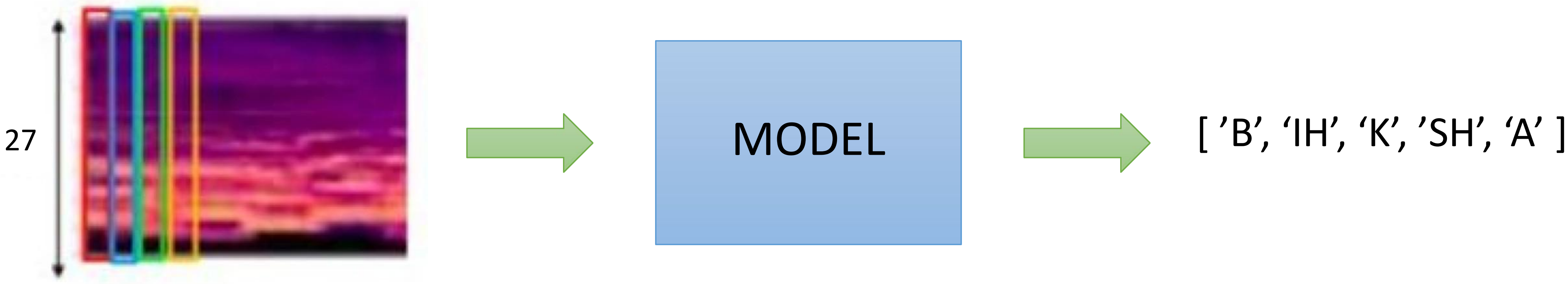
Fall 2022

Eshani Agrawal | Vedant Bhasin

Logistics

- Early submission is due **March 26th, 11:59PM ET**
  - Kaggle submission a with Lev. Dist <= 20
  - Canvas MCQ
- On time submission deadline: **April 7th, 11:59PM ET**
- Constraints: No attention

Problem at hand

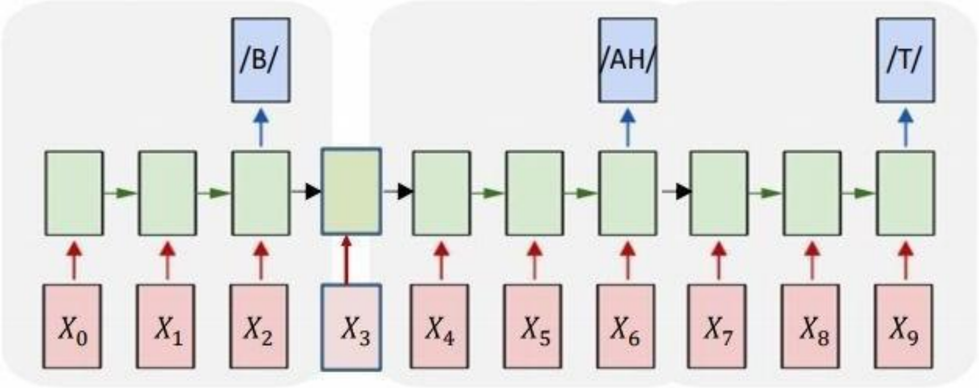


Input Utterance MFCC

Sequence of Phonemes

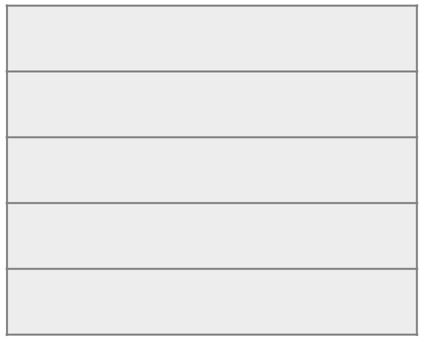
Data and Task

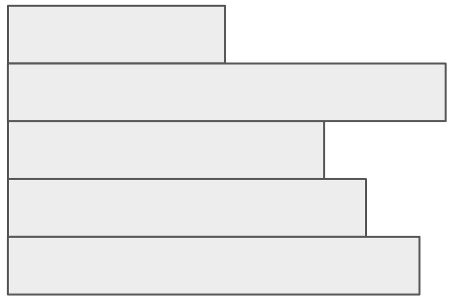
- Features: Same as HW1P2 (27D)
- Labels: Order synchronous but not time synchronous
- Should output sequence of phonemes
  - [‘B’, ‘IH’, ‘K’, ‘SH’, ‘A’] (precisely the indexes)
- Loss: CTCLoss
- Metric: mean Levenshtein distance
  - Can import (given in starter notebook)
  - Sequence of Phonemes -> String and then calculate distance (Use CMUdict and ARPABet)



Batch of Variable Length Inputs: Padding

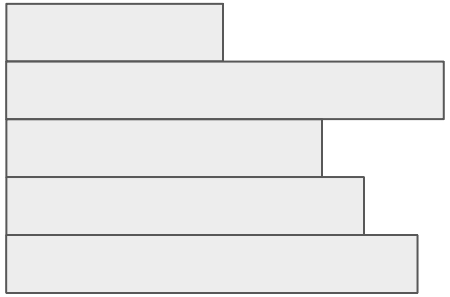
- HW1, HW2: Equal length inputs
- HW3: Variable Length sequences
- Steps:
  - Padding
  - Packing





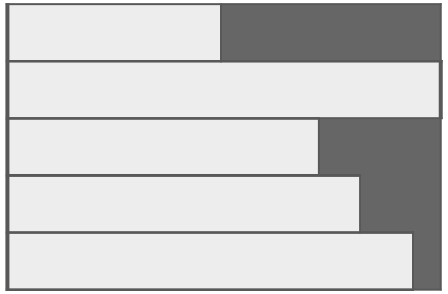
Batch of Variable Length Inputs: Padding

- Padding



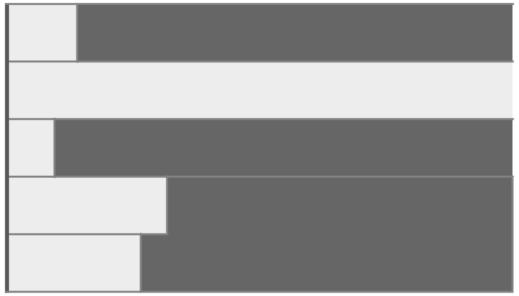
Need to store unpadded lengths as well.
Have the variables *lengths_x, lengths_y* in the starter notebook

Padded to equal lengths



$$(B, *, 27) \rightarrow (B, T, 27)$$

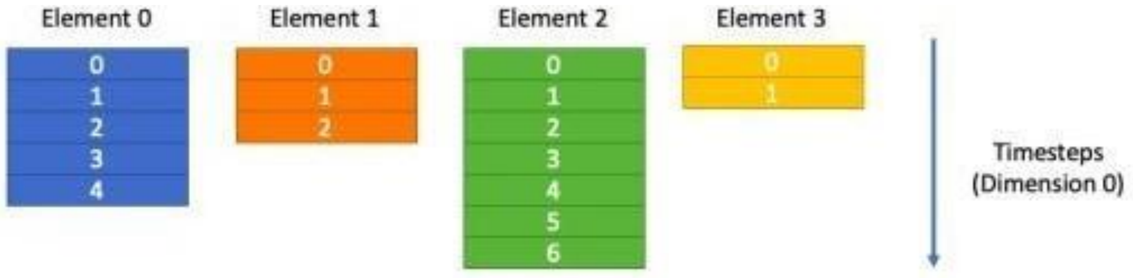
Problematic Example (When padding on whole dataset)



Inefficient with space

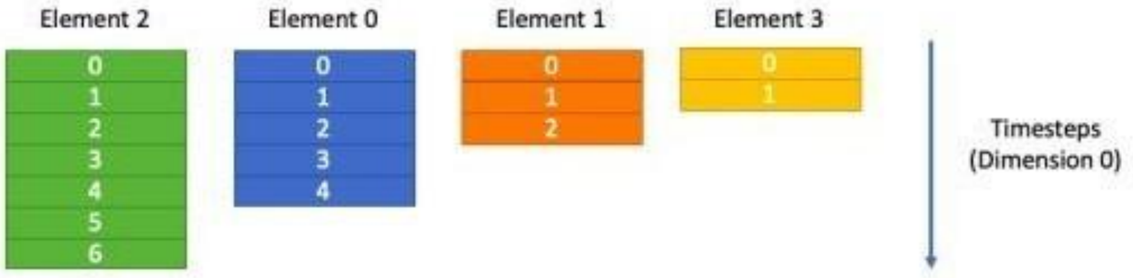
Ref: 11785 Fall 21 Bootcamp

Batch of Variable Length Inputs: Packing



List of Tensors to be packed. Each has same number of features but different time steps.

Figure 2: List of tensors we want to pack



Tensors sorted in descending order based on the number of time steps in each sample.

Figure 3: First we sort the list in a descending order based on number of timesteps in each

Ref: 11785 Fall 21 Bootcamp

Batch of Variable Length Inputs: Packing

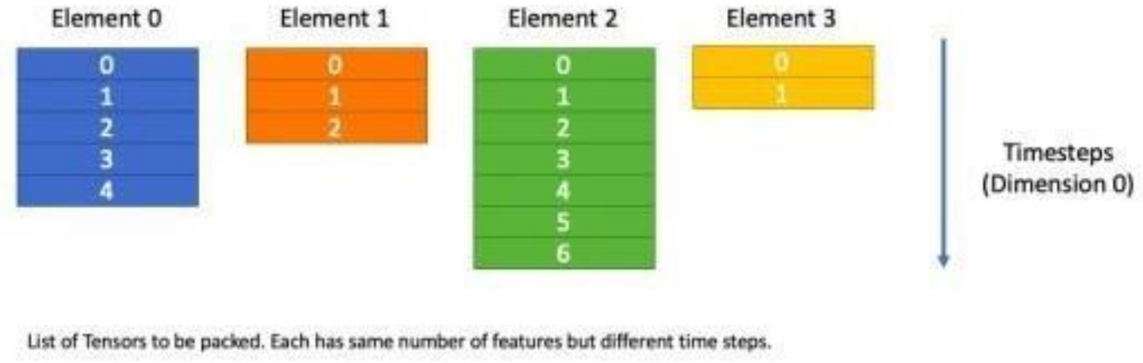


Figure 2: List of tensors we want to pack

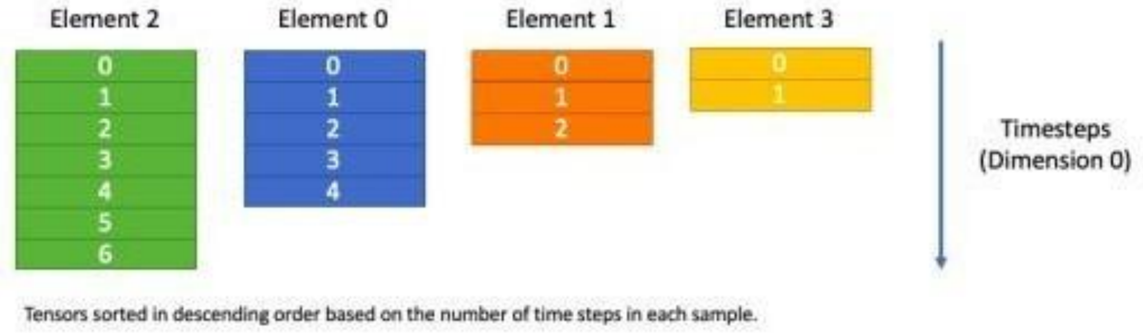


Figure 3: First we sort the list in a descending order based on number of timesteps in each

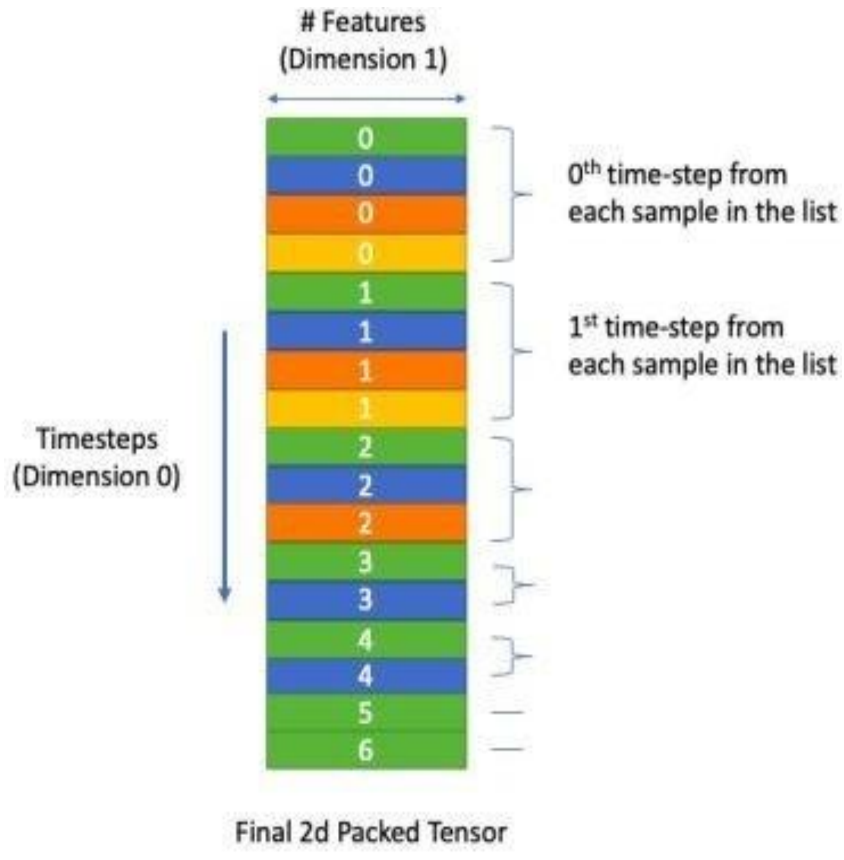


Figure 4: Final Packed 2d Tensor

Ref: 11785 Fall 21 Bootcamp

Packed Sequence

- Pad_sequence()
  - Pads to equal length for batching
- pack_padded_sequence()
  - Packs batch of padded sequences
  - Requires sequences + sequence lengths
- X = pad_packed_sequence()
  - Unpacks back to a batch of padded sequences
  - Outputs sequences + sequence lengths
- Collate Function
  - Dataloader argument
  - Helpful when altering data for batch

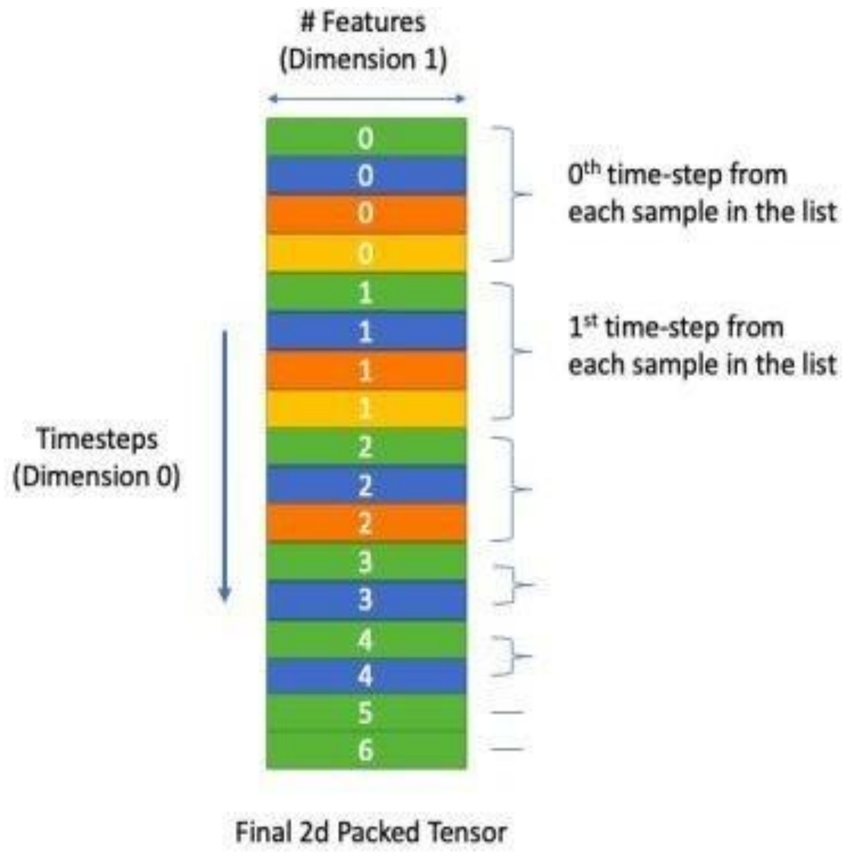
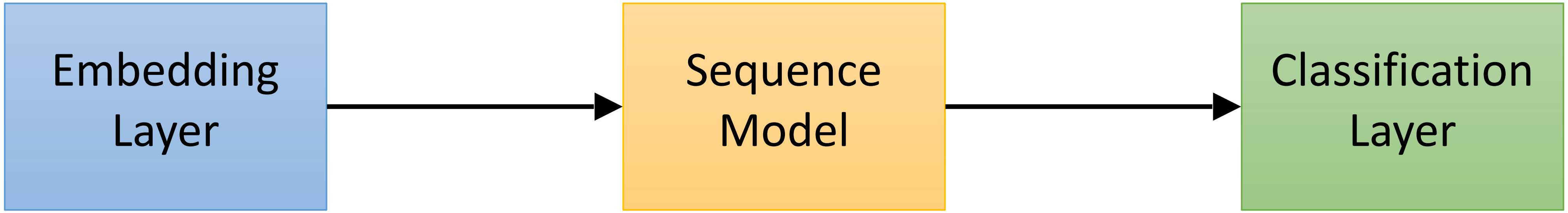
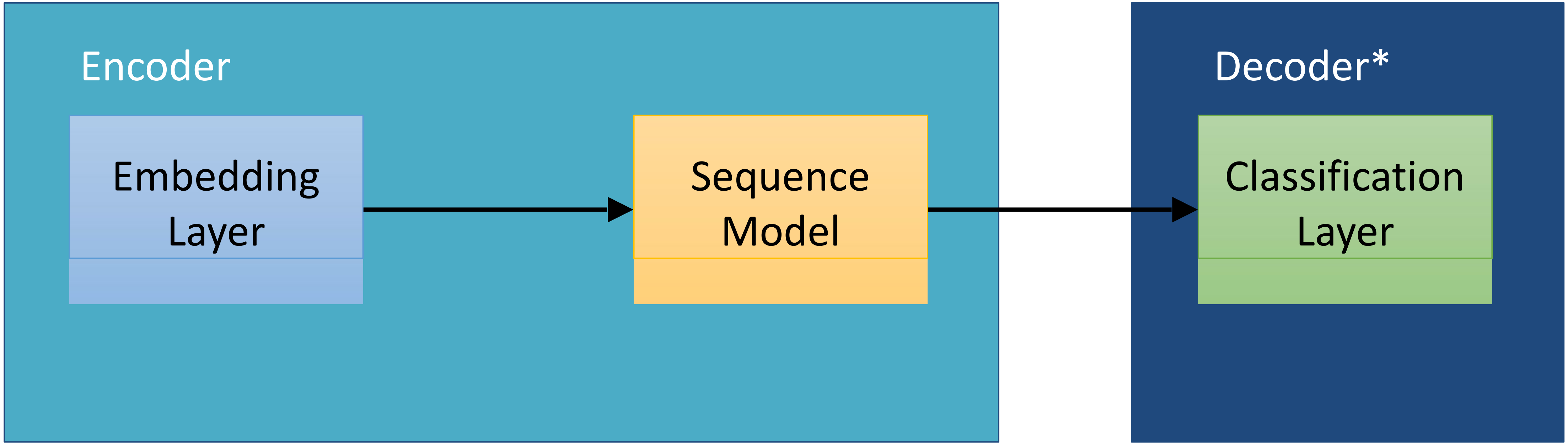


Figure 4: Final Packed 2d Tensor

Parts of a Sequence Model



Encoder - Decoder set up



*Not exactly a decoder in this HW as decoding happens outside the model.

Encoder

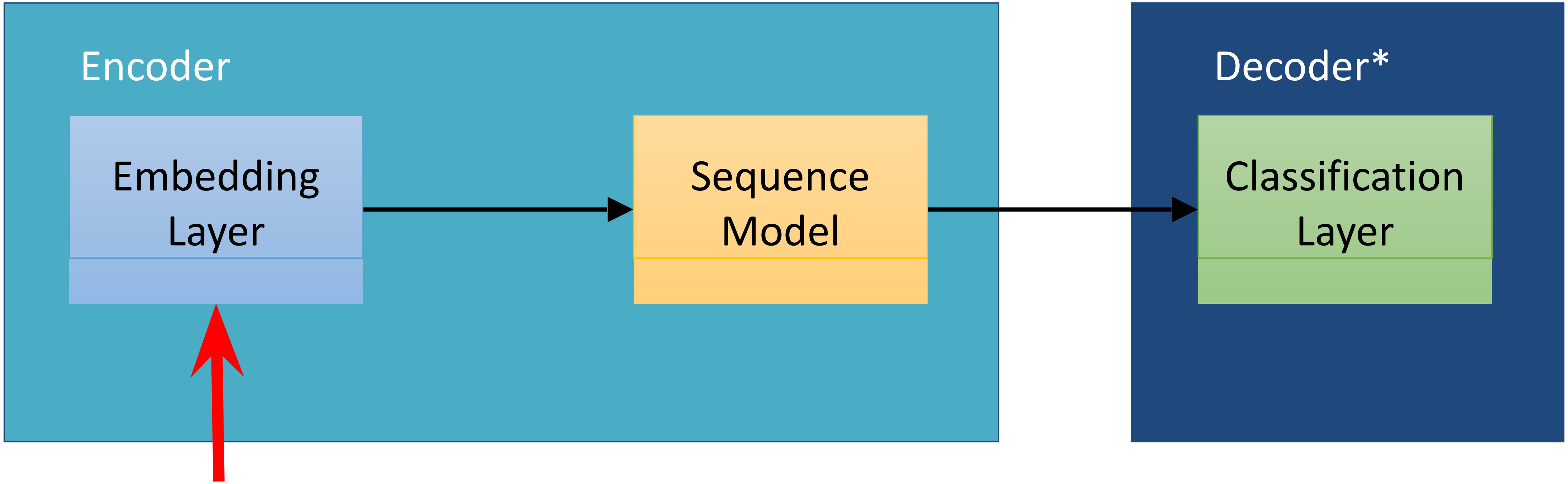
- Typically used to generate high-level representations of given input data.
- There are no labels used to train encoders
- Are trained jointly with decoders.
- Can be any network, CNN, RNN or Linear

Decoder

- It is a network that takes in the feature representation from the Encoder and tries to generate the closest match to the expected output.
- Loss function is applied on the output of the Decoder.
- Can also be trained without encoders, encoders are basically to amplify the results of the decoder

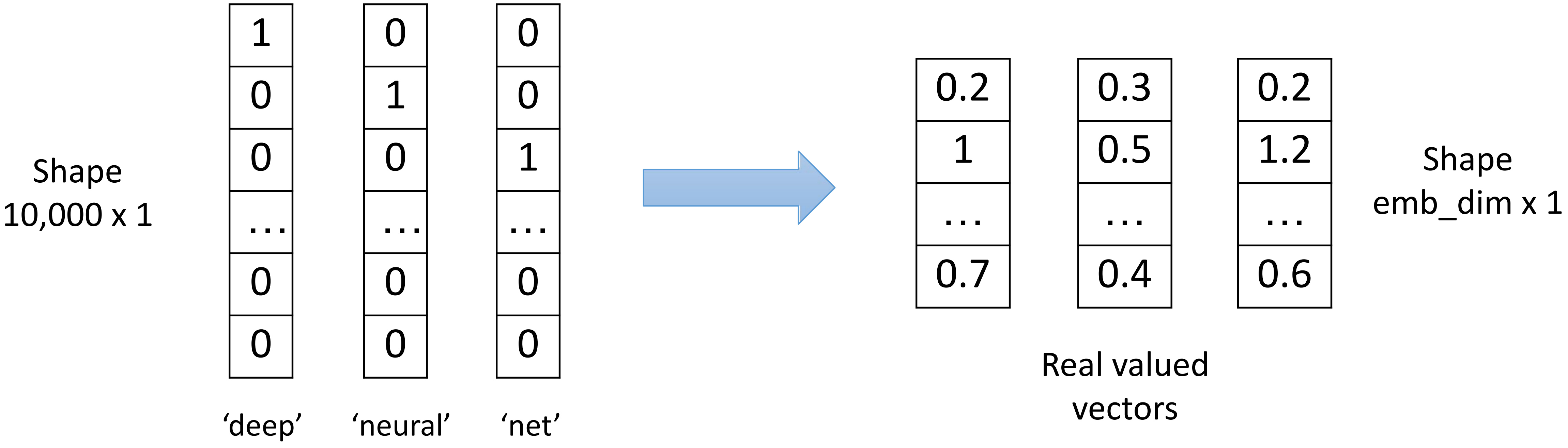
Embedding Layer

- Optional but recommended
- Used to increase/decrease the dimensionality of the input



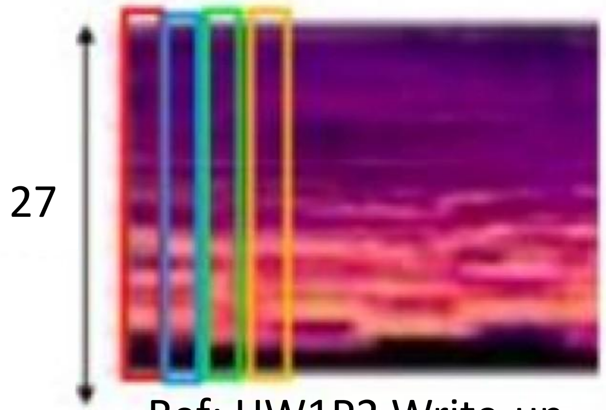
Embedding Layer

- Optional but recommended
- Used to increase/decrease the dimensionality of the input
- Eg. In NLP, 10k vocabulary represented as 1 hot vectors with 10k dim



Embedding Layer

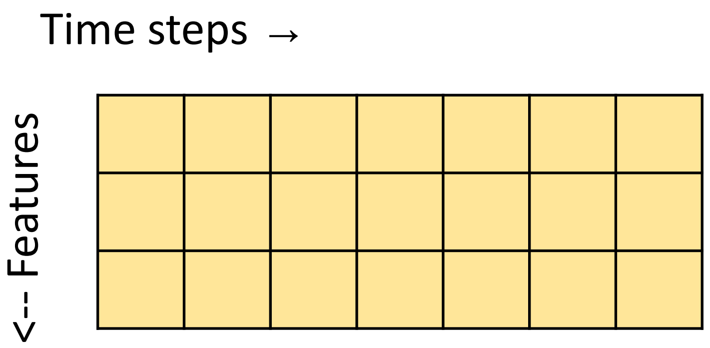
- Optional but recommended
- Used to increase/decrease the dimensionality of the input
- Our task:
  - Input dim = 27
  - Expand to emb_dim > 27 for feature extraction



Ref: HW1P2 Write-up

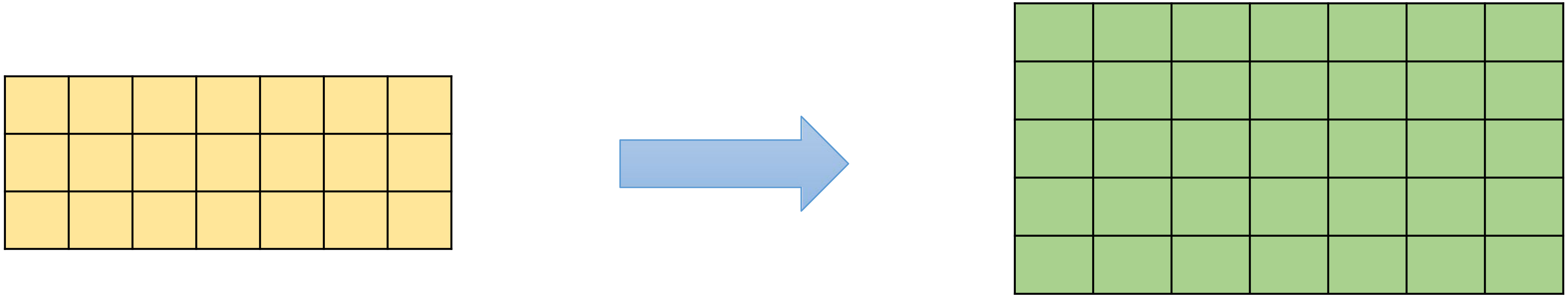
Embedding Layer: Conv1d Layers

- Consider the below as an input having 3 features at each time instant



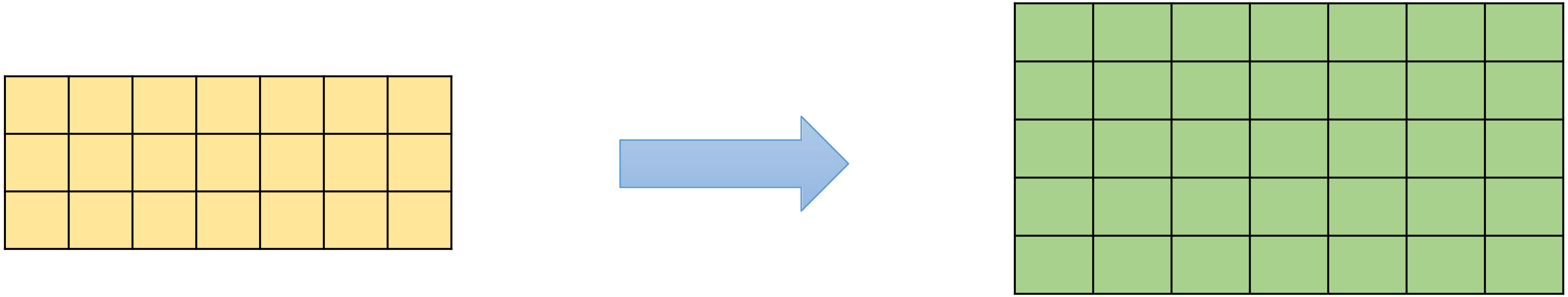
Embedding Layer: Conv1d Layers

- We can use Convolution which increases the channels of the input as we go deeper.



Embedding Layer: Conv1d Layers

- We can use Convolution to which increases the channels of the input as we go deeper.

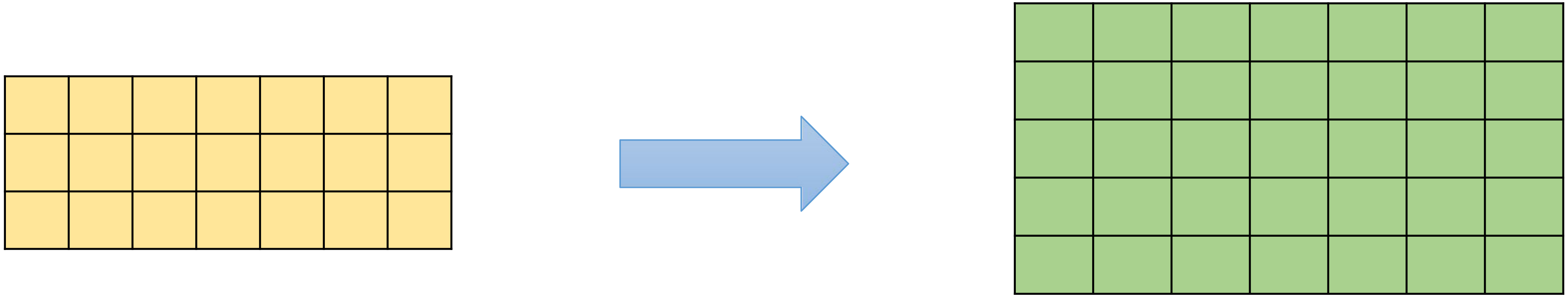


- No. Filters = 5
- Kernel= 3; Padding= 1; Stride= 1
- Kernel= 5; Padding= 2; Stride= 1

(Or anything similar)

Embedding Layer: Conv1d Layers

- We can use Convolution to which increases the channels of the input as we go deeper.



- No. Filters = 5
- Kernel= 3; Padding= 1; Stride= 1 **3D**
- Kernel= 5; Padding= 2; Stride= 1 **5D**

(Or anything similar)

Embedding Layer: Conv1d Layers

- Our input is of shape (B, T, 27) (after padding). How can we change it to (B, T, 64) ?

- Think about what you did in downsampling blocks for HW2P2:
  - increase the number of channels
  - decrease spatial dimensions

Assuming *batch_first = True*
(You may also have it as (T, B, 27)

Embedding Layer: Conv1d Layers

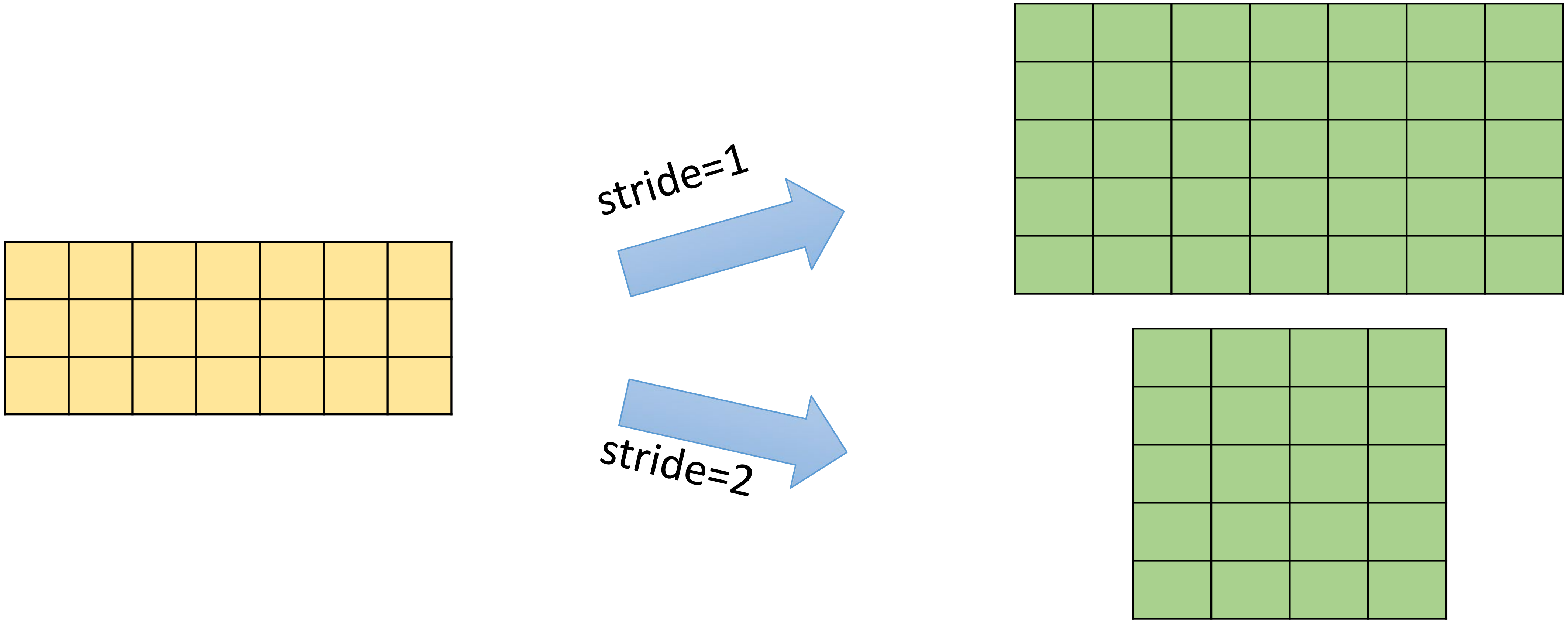
Objective:
change input from (B, T, 27) to (B, T, 64)

- Transpose/Permute:
  - PyTorch conv1d expects tensors of shape (N, C, L)
    i.e. (batch size, in channels, length)
  - Permuting the input aligns the feature dim with C:
    (B, T, 27) → (B, 27, T)
- Apply convolution (B, 27, T) ⟶ (B, 64, T)
- Transpose/Permute: (B, 64, T) ⟶ (B, T, 64)
- Pack and pass to sequence model

Assuming *batch_first = True*
(You may also have it as (T, B, 27)

Embedding Layer: Conv1d Layers

If stride > 1, we effectively reduce the time steps



Embedding Layer: Conv1d Layers

- Stride > 1 reduces computation for LSTM and training is faster.
- However, too much reduction in time steps will lead to loss of information (we don’t recommend downsampling more than 4x)

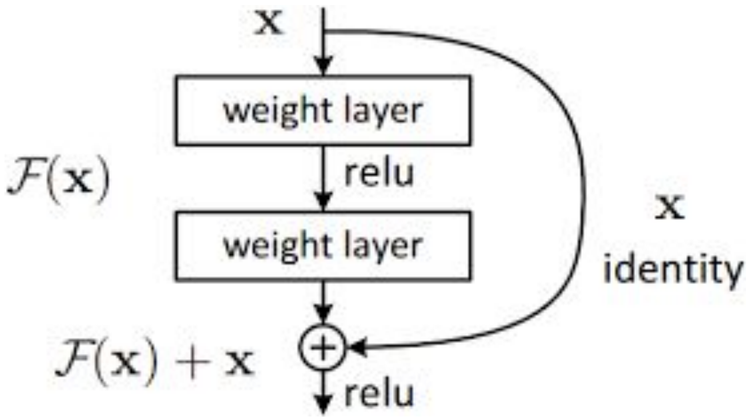
Embedding Layer: Conv1d Layers

- Stride > 1 reduces computation for LSTM and training is faster.
- However, too much reduction in time steps will lead to loss of information (we don’t recommend downsampling more than 4x)

- **Note: Stride > 1 alters number of time steps. You need to change lengths_x accordingly**
  - Use convolution formula ***(X – K + 2*P)//S*** (or)
  - Clamp lengths to length of embedding (torch function)

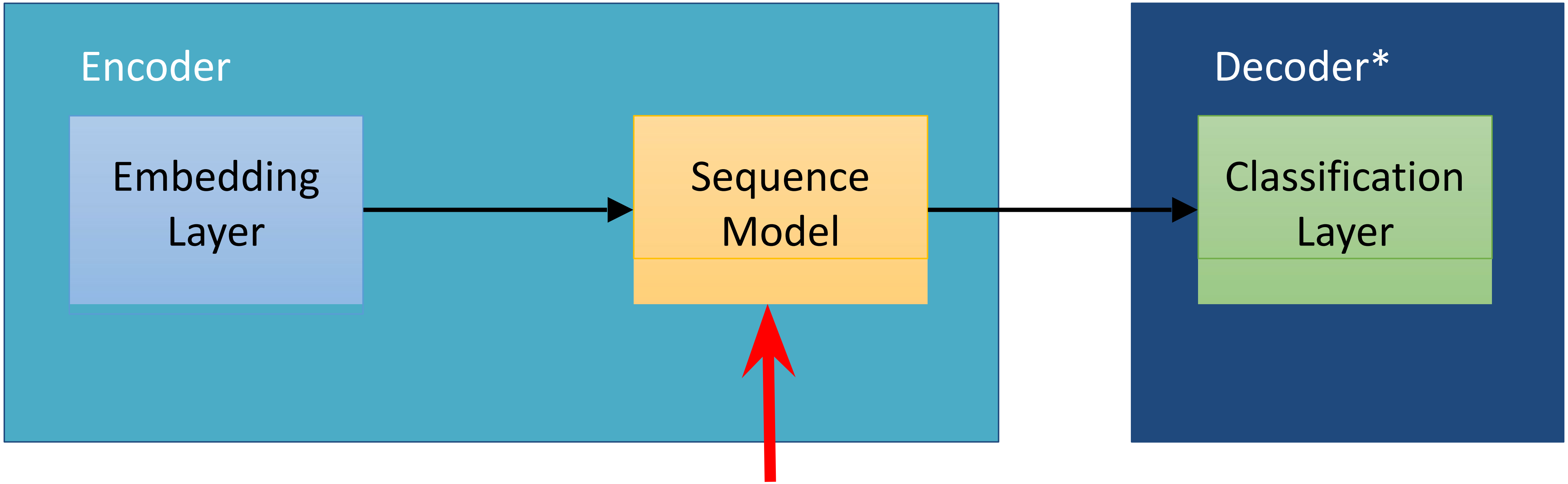
Embedding Layer: Conv1d Layers

- You can try convolution layers based on residual blocks
- Hint: Remember HW2P2!



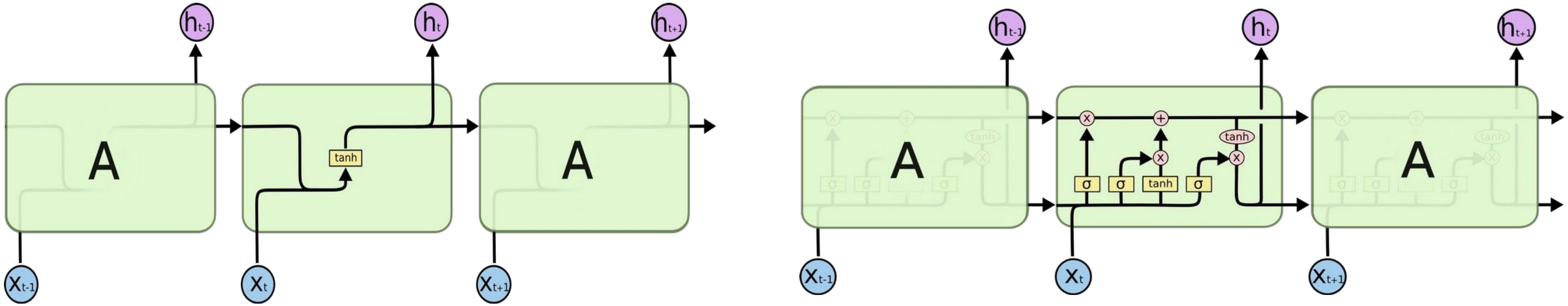
https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf

Sequence Model



Sequence Model

- Can use RNN, GRU, LSTM (recommended) from *torch.nn*



http://colah.github.io/posts/2015-08-Understandin
g- LSTMs/

Sequence Model

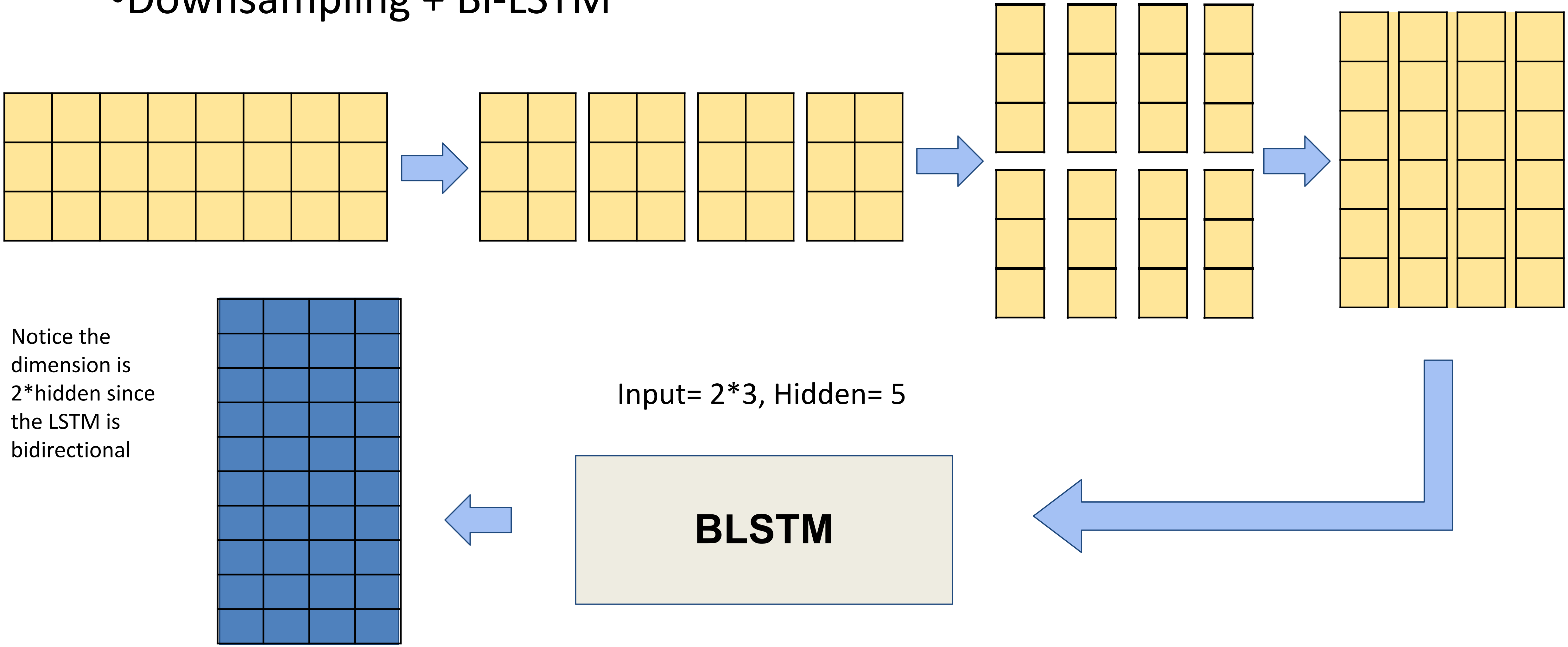
- Important parameters/hyper parameters in *nn.LSTM()*
  - *input_size (27 or embedding_size)*
  - *hidden_dim*
  - *num_layers*
  - *dropout*
  - *bidirectional*
  - Note: when *bidirection = True*, LSTM outputs a shape of *hidden_dim* in the forward direction and *hidden_dim* in the backward direction (in total, *2*hidden_dim*)

pBLSTM

- **p**yramidal **B**i-directional **LSTM.** Described in the Listen-Attend-Spell paper
- The pBLSTM is a variant of Bi-LSTMs that downsamples sequences by a factor of 2 by **concatenating adjacent pairs of inputs** before running a conventional Bi-LSTM on the reduced-length sequence
- This can be implemented using reshape

Pyramidal Bi-LSTM (pBLSTM)

- Downsampling + Bi-LSTM

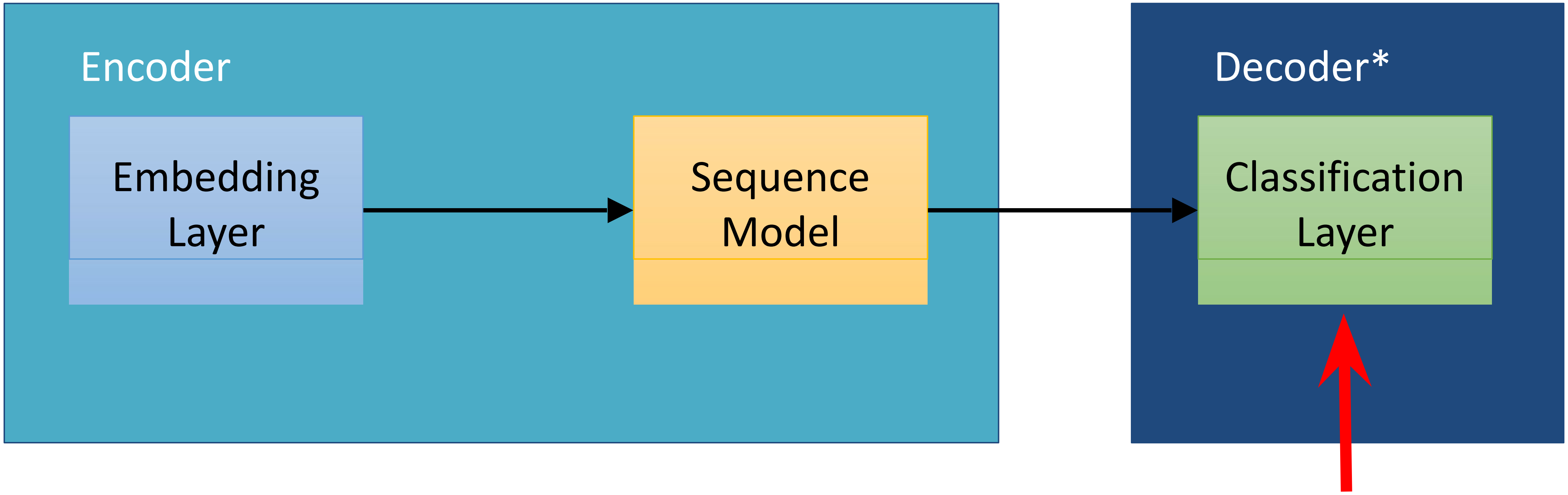


pBLSTM - pseudocode

Listing 1 pBLSTM

```
# X = (batchsize, length, dim) is a minibatch of input sequences, possibly from a previous layer
# Assuming dataloader ensures that all input sequences in the batch are the same length
function O = pBLSTM(X, LSTMwidth, Params)
    # Reshape inputs to have half the length, but twice the dimensionality
    X_downsampled = reshape(X,B,L/2,2*D)
    output = BiLSTM(X_downsampled, LSTMwidth, Params)
    return output
end
```

Classification Layer



Classification Layer

- Same as HW1P2 - just an MLP
- Output from the sequence model goes to the classification layer
- Variations
  - Deeper
  - Wider
  - Different activations
  - Dropout

Hyperparameters and Regularization

- Cepstral Normalization:

$$X \rightarrow (X - \text{mean})/\text{std}$$

- Different weight initialization (for Conv and Linear layers)
- Weight decay with optimizer

Hyperparameters and Regularization

- Scheduler is very important
  - ReduceLRonPlateau (Most of our ablation)
    - Lev distance might start to oscillate at lower values
  - Cosine Annealing
    - Try with higher number of epochs

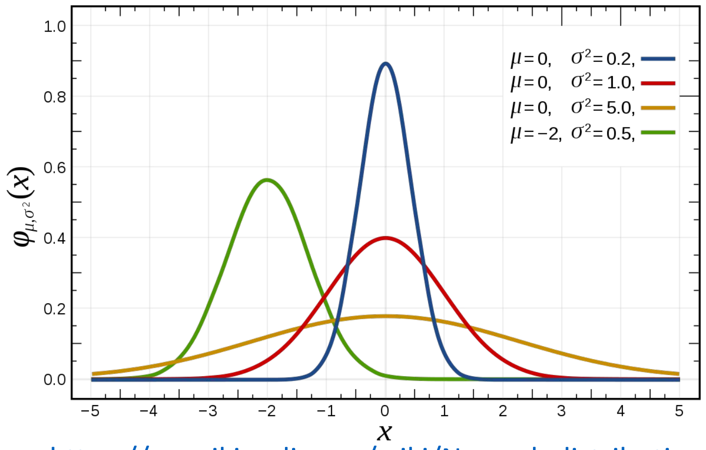
Hyperparameters and Regularization

- Dropout is key
  - Can use dropout in all the 3 layers: Embedding, Sequence model and classification
  - You can also start with a small dropout rate and increase after the model gets trained

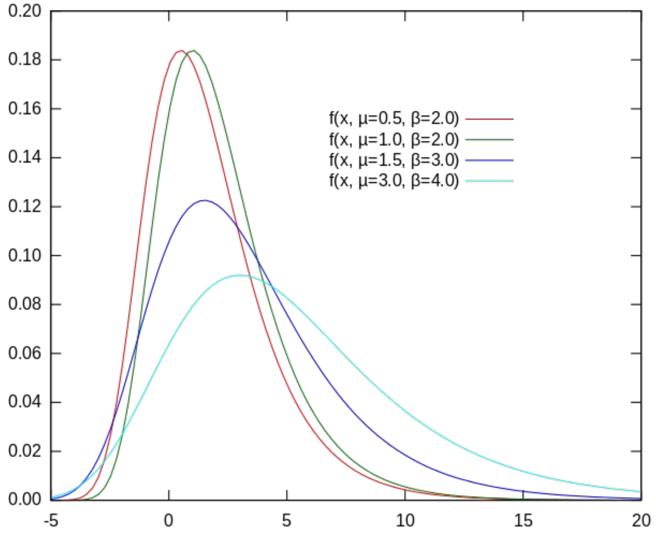
- Locked Dropout for LSTM layer
  - Locked Dropout can be used to apply the same dropout mask to every time step
  - You can refer to PyTorch NLP's implementation of locked dropout here
  - Pay attention to whether modules adhere to batch first format or not

Hyperparameters and Regularization

- Addition of Noise *(only during training)*
  - Gaussian Noise
  - Gumbel Noise
- Need not add to all samples. Implement your module *AddNoise(nn.module)* in such a way that it adds noise to random inputs



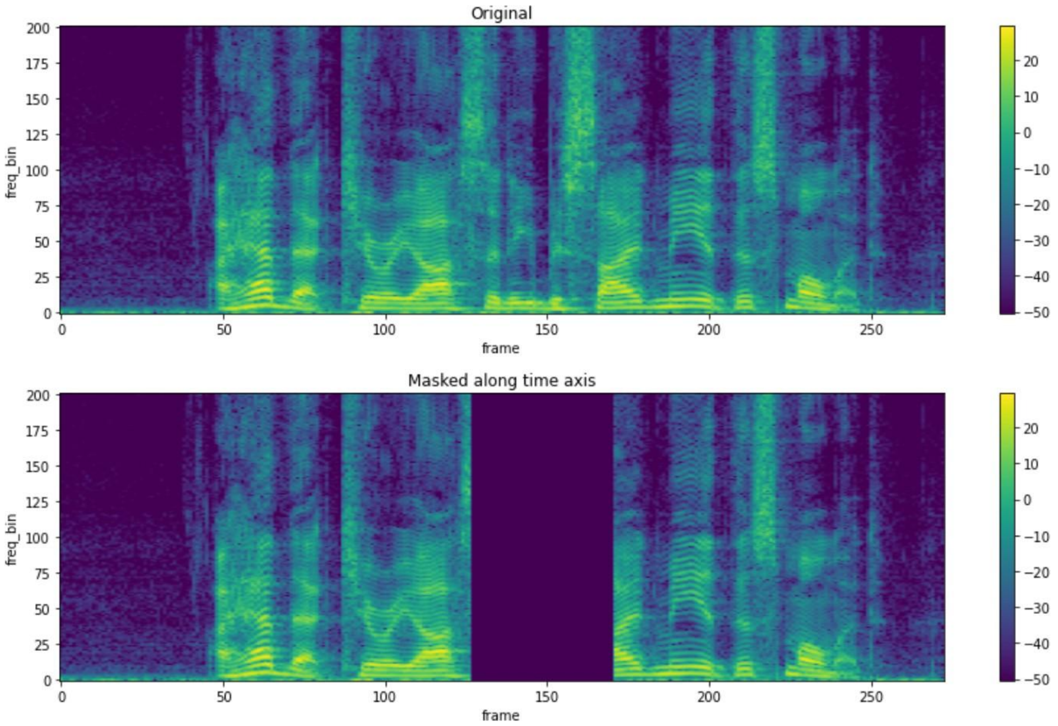
https://en.wikipedia.org/wiki/Normal_distribution

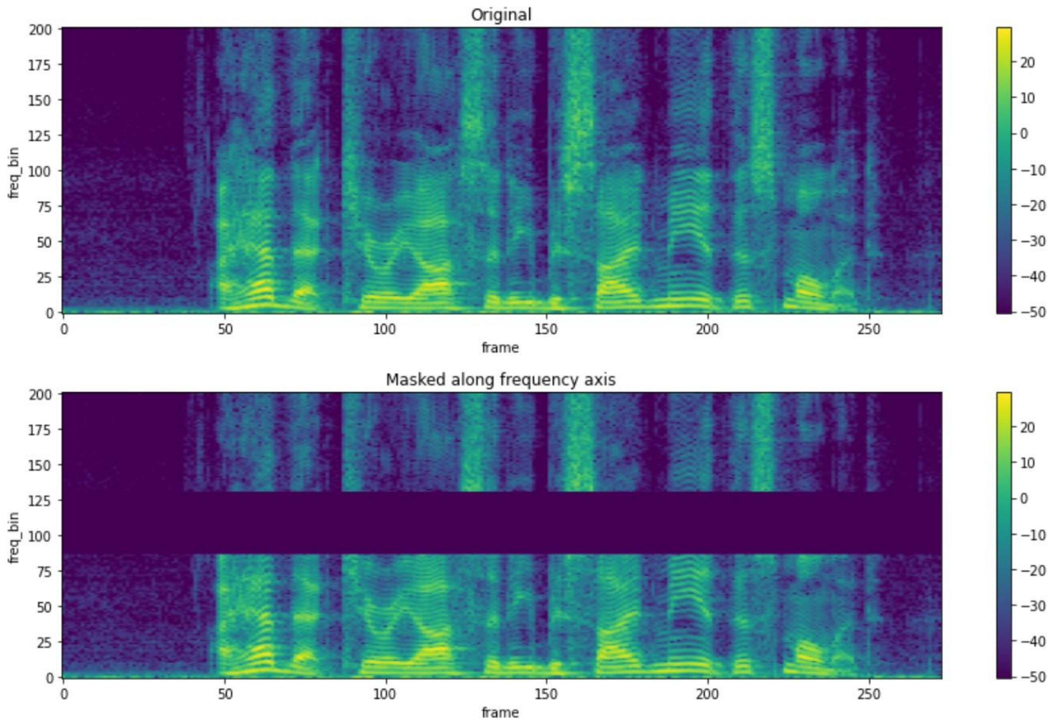


https://en.wikipedia.org/wiki/Gumbel_distribution

Hyperparameters and Regularization

- Torch Audio Transforms [docs]
  - Time Masking
  - Frequency Masking





Hyperparameters and Regularization

- Beam width
  - Higher beam width may give better results (advisable to keep test beam width below 50 for computation purposes)
  - Sometimes bw = 1 (greedy search) also gives good results
  - Tip: Don’t use a high beam width while validating in each epoch (time per epoch will be higher)

Final Tips

- Make sure to split work within your study groups

All the best!