Intro to Neural Nets

RNNs for Text

Today's Agenda

Background on NLP
- Use Cases
- Quick review on bag of words approaches, etc.

TextVectorization Layer
- This implements basic standardization and punctuation removal. It assumes 1-grams, then one-hot encodes.
- No stemming or stop word removal, by default.

Sequence vs. Bag-of-Words
- Conceptually

Architectures for Sequences
- Bidirectional LSTM

Quick Review of NLP Concepts

Pre-processing Text

- Lower-casing, stop word removal, stemming, removing punctuation, stripping rare tokens, etc.
- Tokenization (this may be chars, words, sentences, etc.
- Integer encoding / indexing the tokens.
- Finally, I may or may not leverage sequence information.
- *Q: what is a bag of words approach? What are n-grams?*

	Database	SQL	Index	Regression	Likelihood	linear
D1	24	21	9	0	0	3
D2	32	10	5	0	3	0
D3	12	16	5	0	0	0
D4	6	7	2	0	0	0
D5	43	31	20	0	3	0
D6	2	0	0	18	7	6
D7	0	0	1	32	12	0
D8	3	0	0	22	4	4
D9	1	0	0	34	27	25

Weighting Term-Documents: TF-IDF

Not all phrases are of equal importance…

- E.g., David less important than Beckham
- If a term occurs all the time, observing its presence is less informative

Inverse-document frequency (IDF) helps address this.

$$\mathrm{IDF} = \log(N/n_j)$$

- Term 'weighting' is then calculated as Term Frequency (TF) x IDF
- n_j= # of docs containing the term, N = total # of docs
- A term is deemed important if it has a high TF and/or a high IDF.
- As TF goes up, the word is more common generally. As IDF goes up, it means very few documents contain this term.

TextVectorization Layer

Pre-processing Text

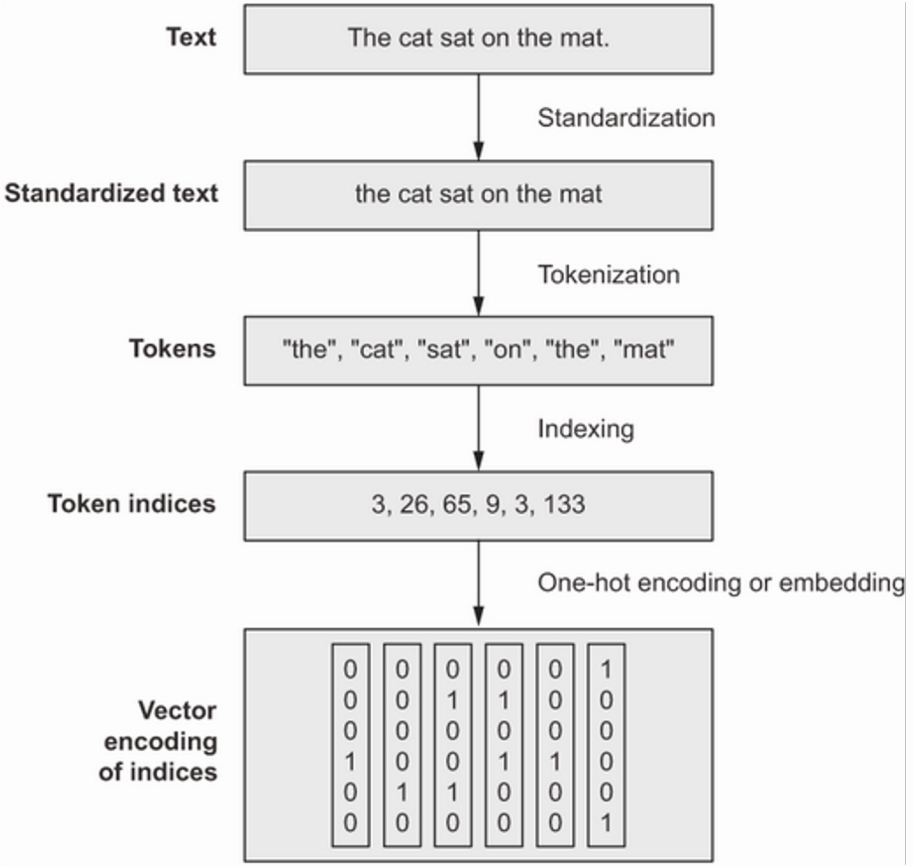
- Standardization, tokenization (words), one-hot-encoding / vectorization.
- The Keras TextVectorization() layer achieves these steps quickly.

Customization

- You can work with n-grams, and do other sorts of pre-processing, using arguments.

Options

- Include as part of TF Dataset pipeline (more efficient)
- Include as a layer in your Keras model.



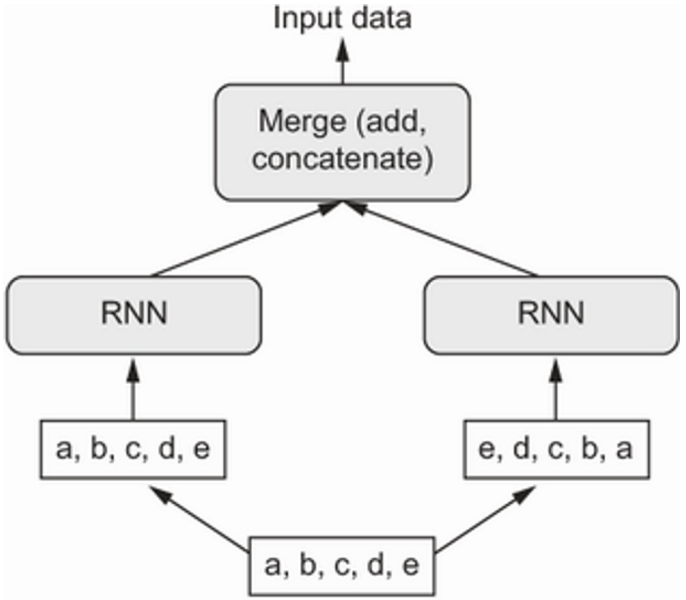
Bidirectional LSTM

We Saw This Last Time

- Take each sequence as input data, as well as a flipped/reversed copy.
- Was state of the art for text processing until relatively recently (transformers now dominate).

Instead of Time Series We Pass…

- Sequences of one-hot-encodings of terms.
- Sequences of pre-trained vector embeddings of terms.



Masking

RNNs with Ragged Lists

- Lists where each element is a sequence of variable length *can* be handled here, but not automatically.
- A batch in Keras must be of a fixed dimensionality; we use padding and truncation to achieve this.

We Deal with Variable Lengths Using a Mask

- The Masking layer “flags” sequence elements that are to be ignored by subsequent layers (e.g., an LSTM) that have the ability to process a mask.
- The output of masking layer passes on the input + a second tensor of the same shape, containing Boolean values (True = process, False = ignore).
- By default values of 0 get ignored, but we can override this.

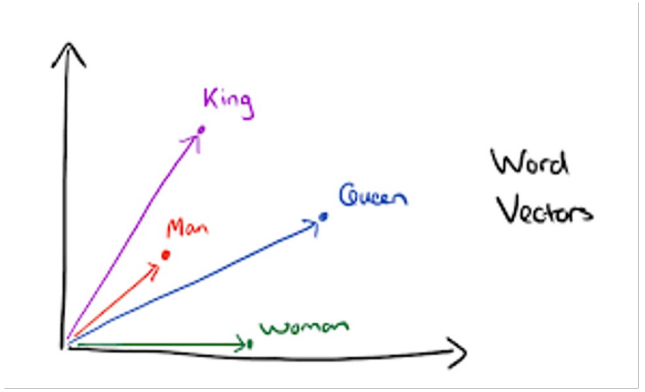
Embedding Layer

With Hot Encodings, Model Will Still Struggle to Figure Out Semantics

- Despite having sequence, the model is "told" that the tokens are orthogonal / independent of one another in their meanings. But that's not true!

Textual Embedding Layer First Provides Dimensionality Reduction

- Represent words into a lower dimensional space – similar vector = similar meaning.
- The Embedding layer is a lookup table that maps tokens to vectors. For each token in the vocabulary, the network learns a vector representation. The vectors are initially random, and the network updates them in training to learn representations that help in prediction (just like with convolution filters!).
- In practice, it is learning semantic relationships…
- This is much better for an RNN than a hot encoding, because 120 values (for example) is << 20,000!



Pre-Trained Embeddings: GloVe

Global Vector Representation

- Based on a giant term-term co-occurrence matrix – rows are vectors of co-occurrence (conditional) probabilities.
- Two terms are similar if their ratios of co-occurrences with *other* terms are about equal.
- Roughly speaking, GloVe learns word vectors, e.g., v_i and v_j, such that the dot product of any pair of vectors is equal to their co-occurrence ratio P(v_j | v_i).
- This is achieved via a gradient-descent optimization.

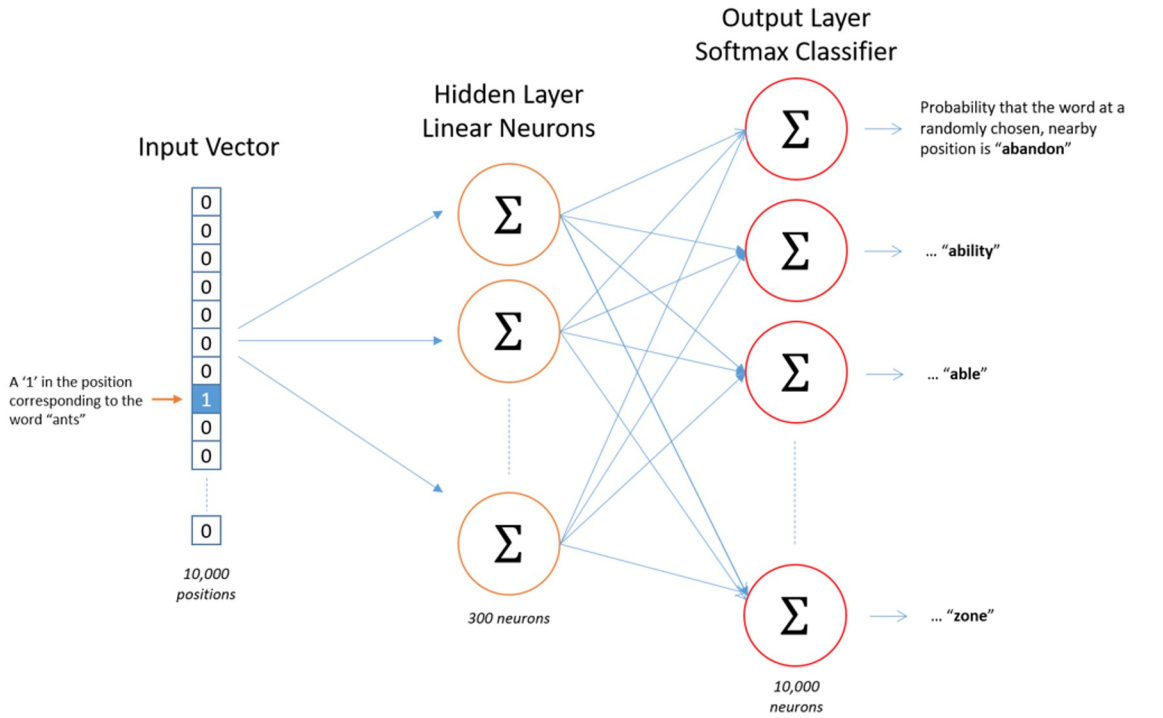
	the	cat	sat	on	mat
the	0	1	0	1	1
cat	1	0	1	0	0
sat	0	1	0	1	0
on	1	0	1	0	0
mat	1	0	0	0	0

Pre-Trained Embeddings: Word2Vec

Word2Vec

- Two types: CBoW and Skipgram
- Construct training examples and labels.

Source Text	Training Samples generated from source text			
I will have orange juice and eggs for breakfast	(will, I)	(will, have)	(will, orange)	
I will have orange juice and eggs for breakfast	(have, I)	(have, will)	(have, orange)	(have, juice)
I will have orange juice and eggs for breakfast	(orange, will)	(orange, have)	(orange, juice)	(orange, and)
I will have orange juice and eggs for breakfast	(juice, have)	(juice, orange)	(juice, and)	(juice, eggs)
I will have orange juice and eggs for breakfast	(and, orange)	(and, juice)	(and, eggs)	(and, for)
I will have orange juice and eggs for breakfast	(eggs, juice)	(eggs, and)	(eggs, for)	(eggs, breakfast)
I will have orange juice and eggs for breakfast	(for, and)	(for, eggs)	(for, breakfast)	



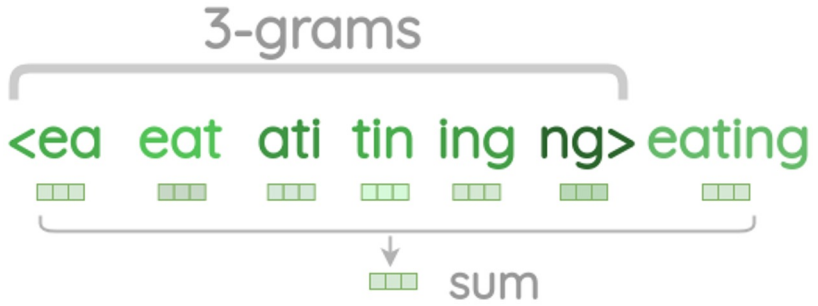
Pre-Trained Embeddings: Limitation

Out of Sample Words

- Both GloVe and Word2Vec are limited to words you've seen before in training. They cannot handle new words. Those words thus get omitted / dropped, or you need to do something different.

FastText

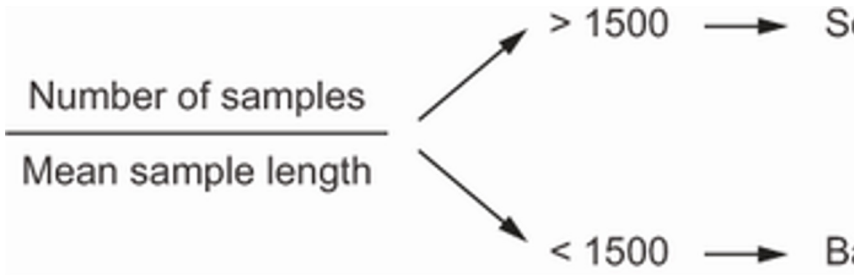
- An extension to Word2Vec which learns character n-grams of words. So, instead of embedding words, we embed portions of words (e.g., a 3-gram character representation would break up the word 'coffee' into 'cof', 'off', 'ffe', … and then learn vector embeddings of each.



Sequence vs. Bag-of-Words

Word-Ordering Contains Information

- We can get a weak representation of language sequences using n-grams, but this can be limited.
- Sequence-models may provide leverage more information from language in prediction tasks (if we have enough examples, and the sequences are short enough).
- We can represent these sequences with RNNs, typically bidirectional RNNs (because word ordering and interpretation is not always linear).



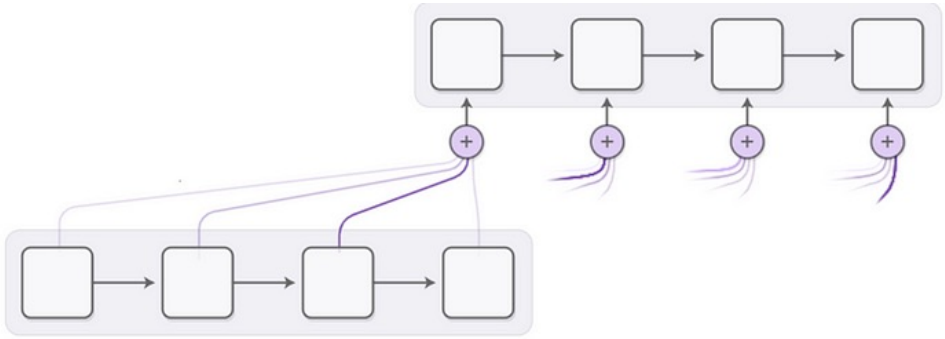
General Attention

Provide the Network with a Way to Learn Where it Should Focus

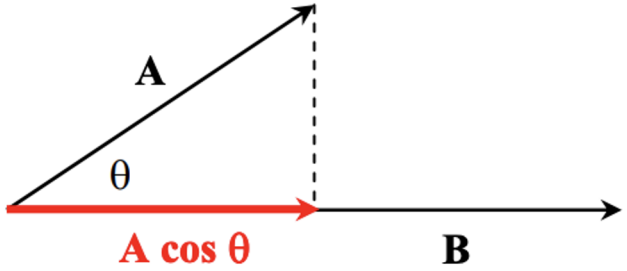
- Rather than having the network assume equal relevance across all inputs it encounters, we can let it learn to focus on different segments of the input.

These ideas apply to inputs that have any sort of structural relation between elements

- In the case of CNNs, we might let our network learn to focus its attention on certain segments of a photo based on the features / values it encounters in said segments.
- In the case of RNNs, we might let our network learn to focus its attention on certain tokens in a string base (e.g., important words or sounds).



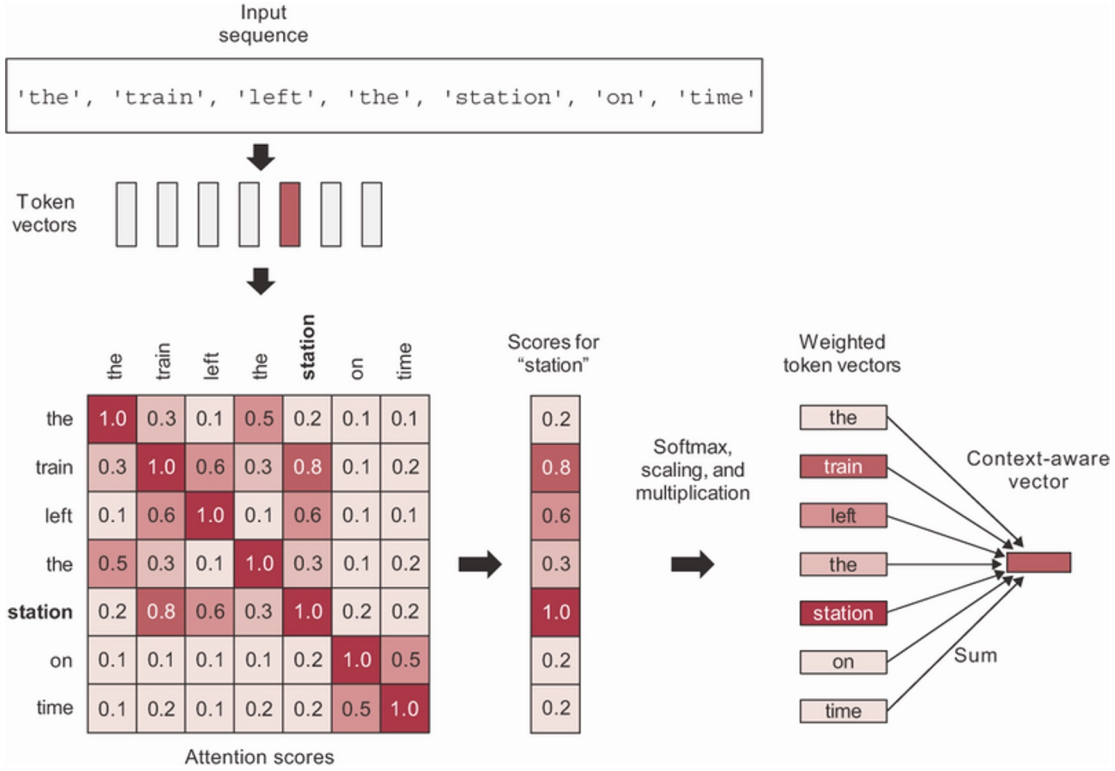
Self-Attention



We Can Also Enable Broader Contextual Consideration

- Self-attention allows the network to evaluate the importance of an input element (e.g., token) based on that that element's value *and all the other element values.*

- Basically, it gives the network a way to shift focus to certain items that are useful, considering informational connections that may be very distant in the input sequence or image.



Self-Attention Layer

```python
def self_attention(input_sequence):

    # Our output will be new vector representations for each word in the sequence.
    output = np.zeros(shape=input_sequence.shape)

    # For each word-vector representation in the input sequence
    for i, pivot_vector in enumerate(input_sequence):

        # Scores will be scalars, one for each word in the sequence.
        scores = np.zeros(shape=(len(input_sequence),))

        # For each word-vector representation in the input sequence (i.e., look at all pairwise combinations of word vectors.)
        for j, vector in enumerate(input_sequence):

            # Take the dotproduct between word i's vector and word j's vector - this value is larger for semantically related words, and smaller for orthogonal words.
            scores[j] = np.dot(pivot_vector, vector.T)

        # Scale the scores - divide the dot products by the root of the dimensionality of the embedding space.
        scores /= np.sqrt(input_sequence.shape[1])

        # Run the results through a softmax. So, for a given word, i, we get a set of scores for all other terms in the sequence, each 0-1, summing to 1.
        scores = softmax(scores)

        # Make a new placeholder vector representation for word i.
        new_pivot_representation = np.zeros(shape=pivot_vector.shape)

        # For all pairwise dot-products, i.e., attention scores, multiply the score by the associted word j, and add them up.
        for j, vector in enumerate(input_sequence):
            new_pivot_representation += vector * scores[j]

        # the vector representation of word i is now shifted toward other terms in the sequence that have similar semantic meaning.
        output[i] = new_pivot_representation

    return output
```

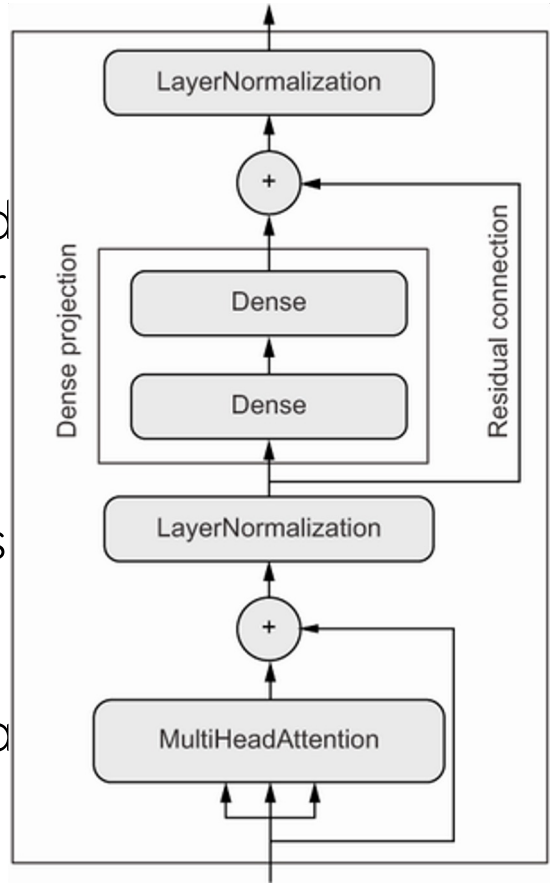
Transformer Architecture

Implement Multiple, Parallel Attention Mechanisms

- This allows the model to figure out different ‘types’ of attention patterns.
- So, maybe the model should pay attention to word 1 and word for one ‘reason’ and it should pay attention to word 3 and wor too, for a different ‘reason’.

Transformer Builds on Multi-Head Attention

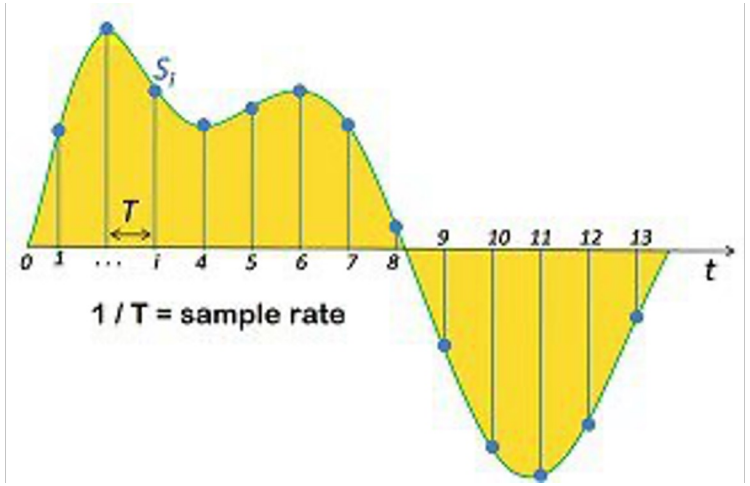
- It stacks the parallel attention layers with normalization layers and dense layers, plus some residual connections to enable better gradient updates.
- LayerNormalization() normalizes within sequence, instead of a the batch.



RNN for Audio

Same Sequence Concepts Work for Audio Data

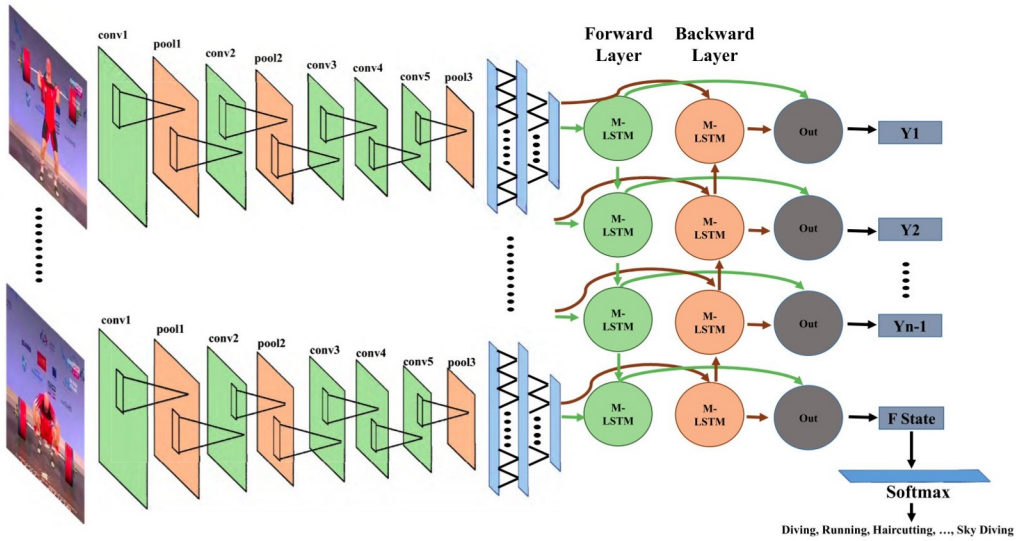
- Audio files are just sequences of numeric values (amplitude), possibly two if it was recorded in stereo.
- Once we recognize this, we realize we can predict things about audio sequences too!



CNN-RNN for Video

Hybrid Topology for Image Sequences

- We Use CNN's to detect features at a given input.
- We feed those feature maps into an RNN architecture, like LSTM.
- We can use this topology to predict things about videos.
- You might pre-process frames using a pre-trained CNN and pass feature maps as sequences to an RNN.



Questions?