Intro to Neural Nets

Session 6: Working with Text

Session 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

- Standardization, stop words, stemming, tokenization (words), n-grams.
- One-hot-encoding / vectorization.
- Final state is often a Term-Frequency Matrix
- *Q: why is this called a bag-of-words approach?*

	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

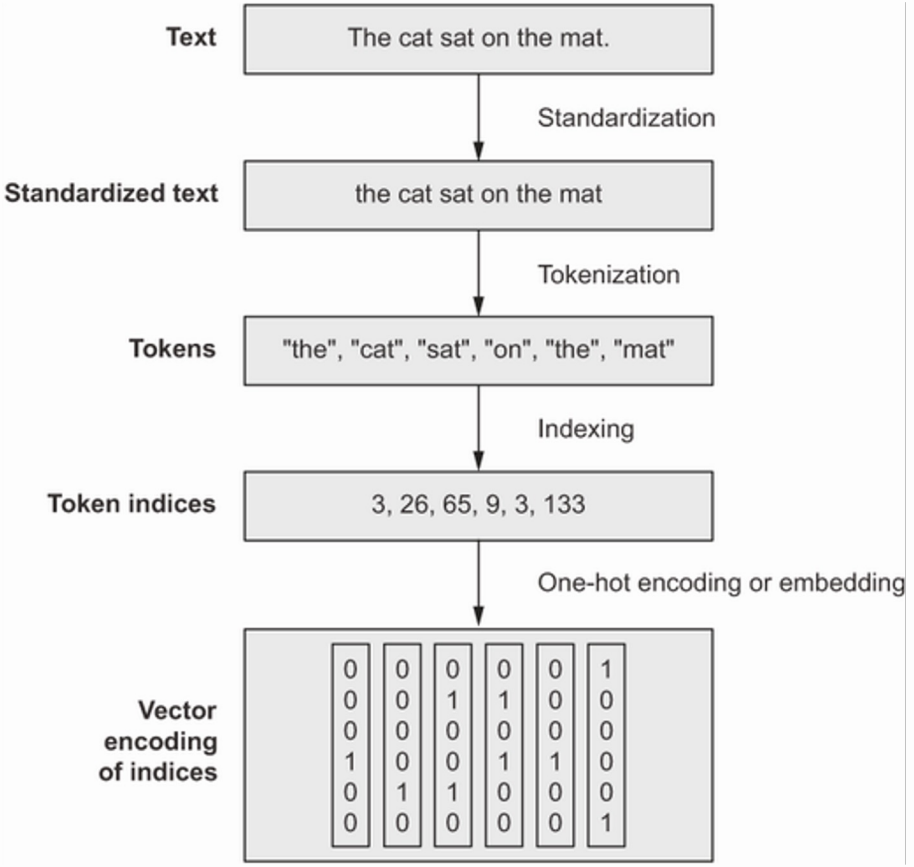
- Standardization, tokenization (words), one-hot-encoding / vectorization.
- The Keras TextVectorization() layer achieves these steps quickly.

Customization

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

Options

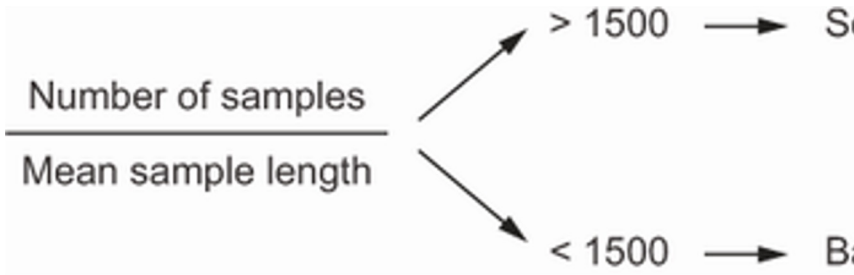
- Include as part of TF Dataset pipeline (more efficient)
- Include as a layer in your Keras model.



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).



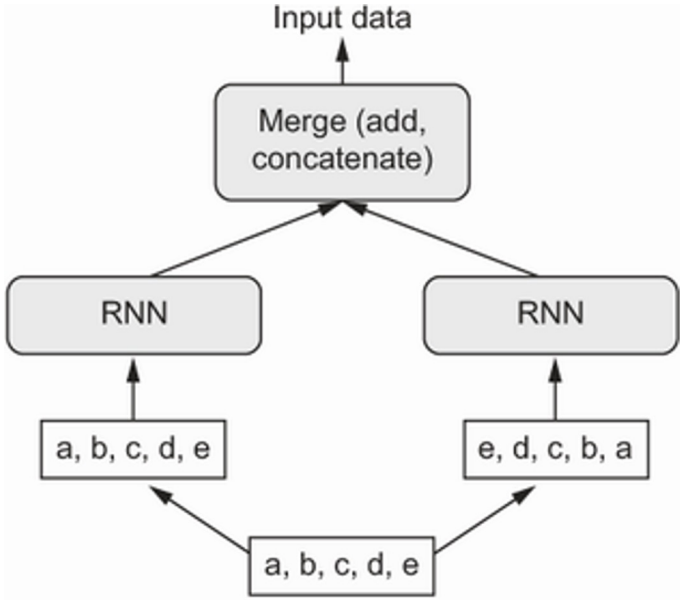
Bidirectional LSTM

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.



Embedding Layer

LSTM Will Still Struggle to Figure Out Semantics

- Despite having sequence, it will struggle with synonyms, grammar, concepts.

Textual Embedding Layer First Provides Dimensionality Reduction

- Cast words into a latent dimensional space – similar vector = similar meaning.
- The Embedding layer is a lookup table that maps tokens to vectors. The vector associations are weights in the network, randomly initialized. Network updates them to learn dimensionality reductions that help with prediction (just like with convolution filters).
- We can pass the output sequences of learned vector representations into our LSTM.

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.

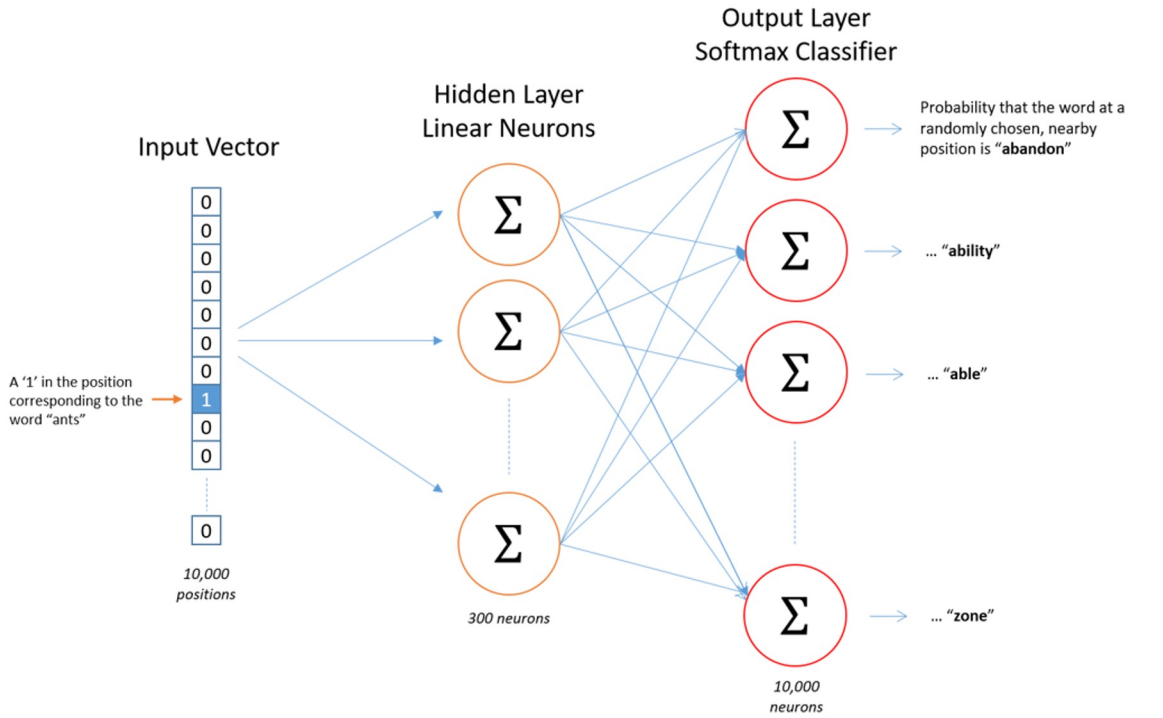
	the	cat	sat	on	mat
the	0	1	0	1	1
cat	1	0	1	0	0
sat	0	1	0	1	0
on	1	0	1	0	0
mat	1	0	0	0	0

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)	



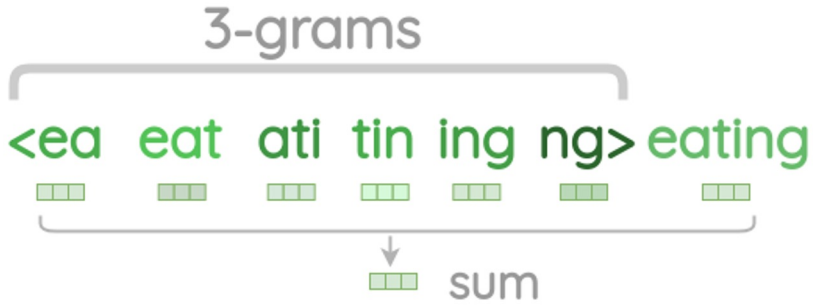
Pre-Trained Embeddings: Limitation

Out of Sample Words

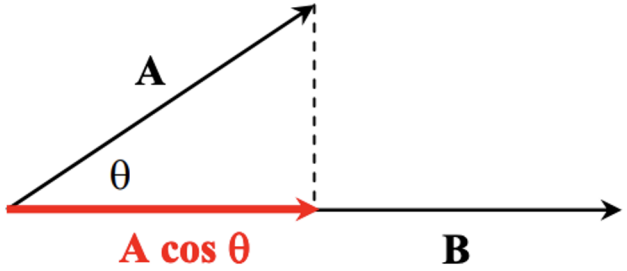
- Both GloVe and Word2Vec are limited to words you've seen before in training. They cannot handle new words. Those words thus get omitted / dropped, or you need to do something different.

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.

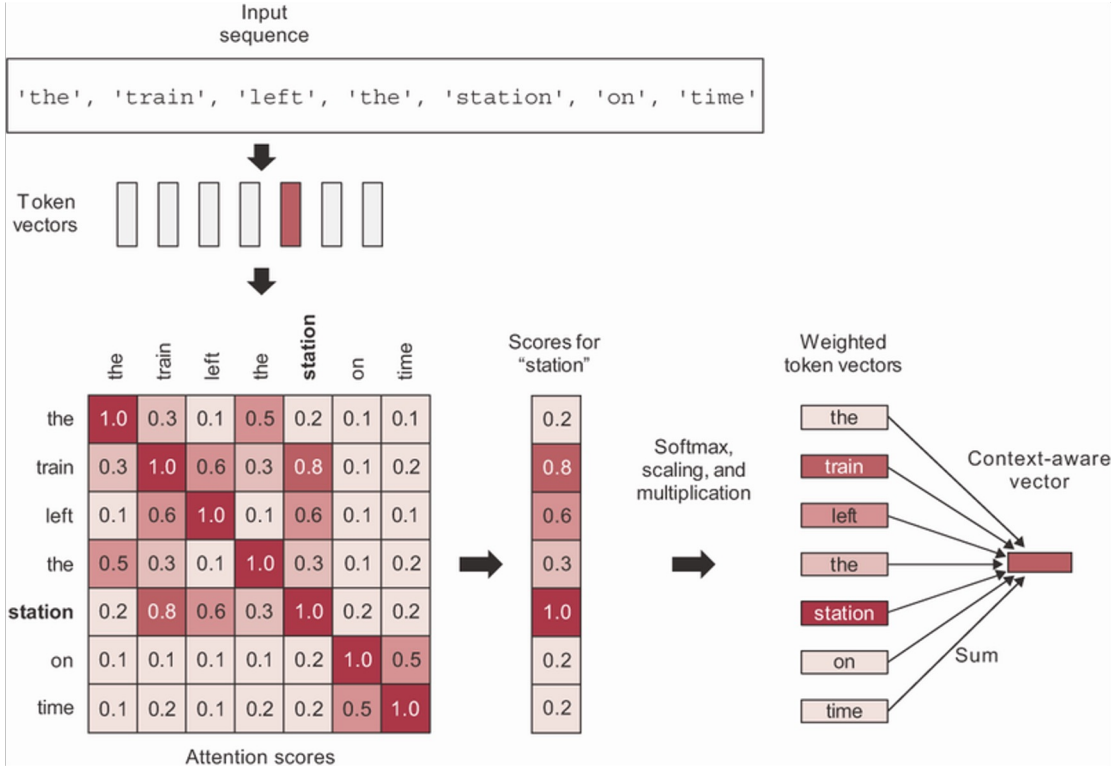


Attention (Building Block of LLMs)



Drawback of LSTM: Tries to Memorize Everything!

- BUT: some pieces of a sequence are more important than others for understanding values at a particular position.
- Self-attention Layer: a dense layer that takes sequences of values as input and implements some mechanism to figure out weights that can be used to amplify or attenuate sequence elements.
- Basically, it gives the network a way to shift focus to certain items that are useful



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
```

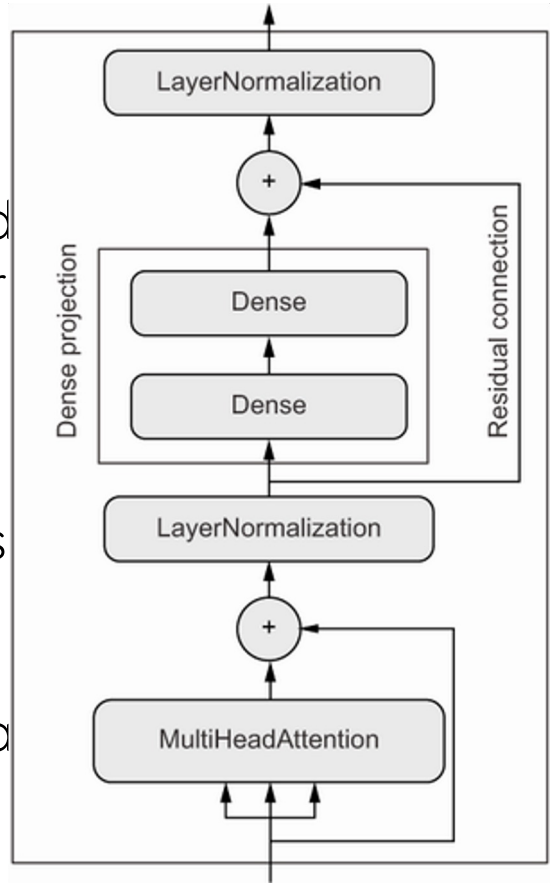
Transformer Architecture

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.



Other Generative Models

Generative Models Have Taken Off

- **Text-to-Image:** Midjourney, Stable Diffusion, DALL-E, etc.
- **Audio + Photo to Video:** D-ID
- **Text to Voice:** ElevenLabs

Generative Adversarial Networks (GANs)

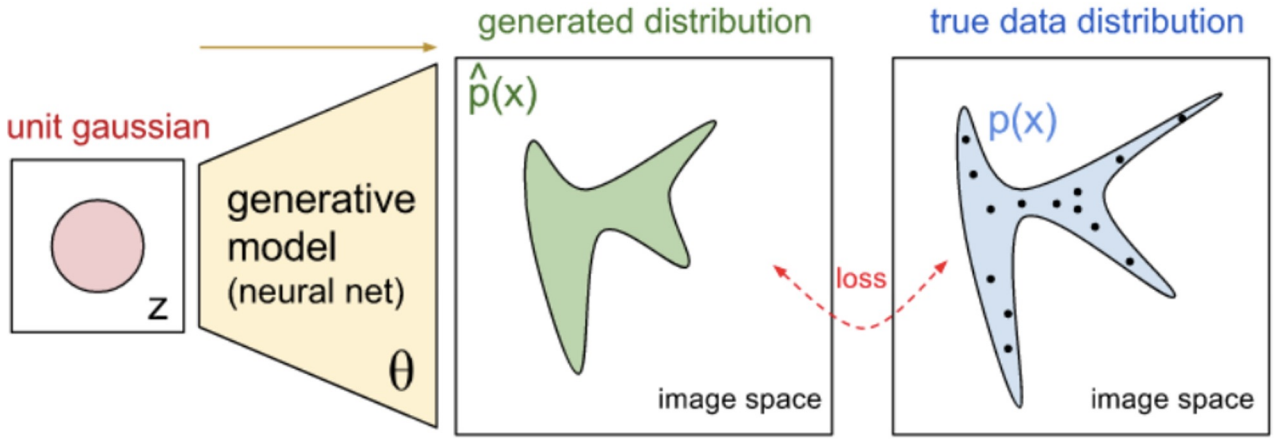
GANs are a Powerful Flexible Tool for Generative Modeling

- What is a GAN? How do GANs work?
- What problems can we address with GANs?
- How do we implement a GAN?

The Goal of a GAN

Synthesize Samples That Conform to the Real Data's Distribution

- We train a network to produce synthetic samples that are indistinguishable from true samples.
- More concrete, this means we train a network to learn the probability distribution of real data.
- For example, learn the joint probability distribution of pixel values in a set of images.
- So, the resulting network could take a random vector of noise as input, and map it to a synthetic output that looks very similar to real data.



How Might We Do This?

A Neural Net That Produces Image Output

- Take in a random vector as input, and have it produce image predictions.
- Next, compare those predictions with real images. But how?
- We don't want it to try to predict a specific image, because then it won't be able to produce new synthetic examples.
- The problem? **The loss function is extremely complicated.**
- A solution… `

Generative Adversarial Nets

Ian J. Goodfellow, Jean Pouget-Abadie[*], Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair[†], Aaron Courville, Yoshua Bengio[‡]
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Adversarial Architecture

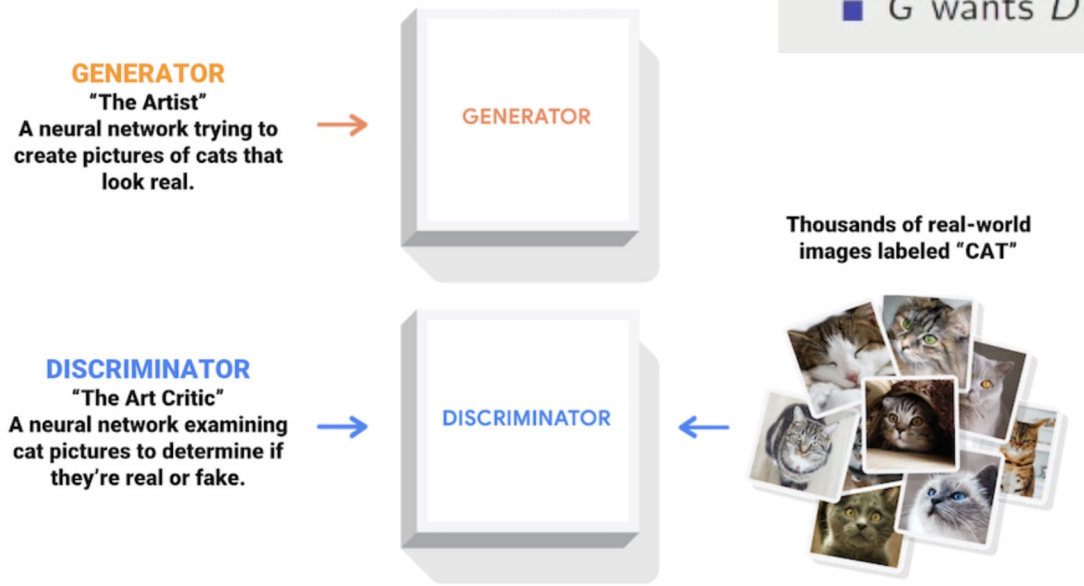
Discriminator

- Serves as an adaptive loss function for the generator.
- It's a throw-away network that is just there to help train the generator.

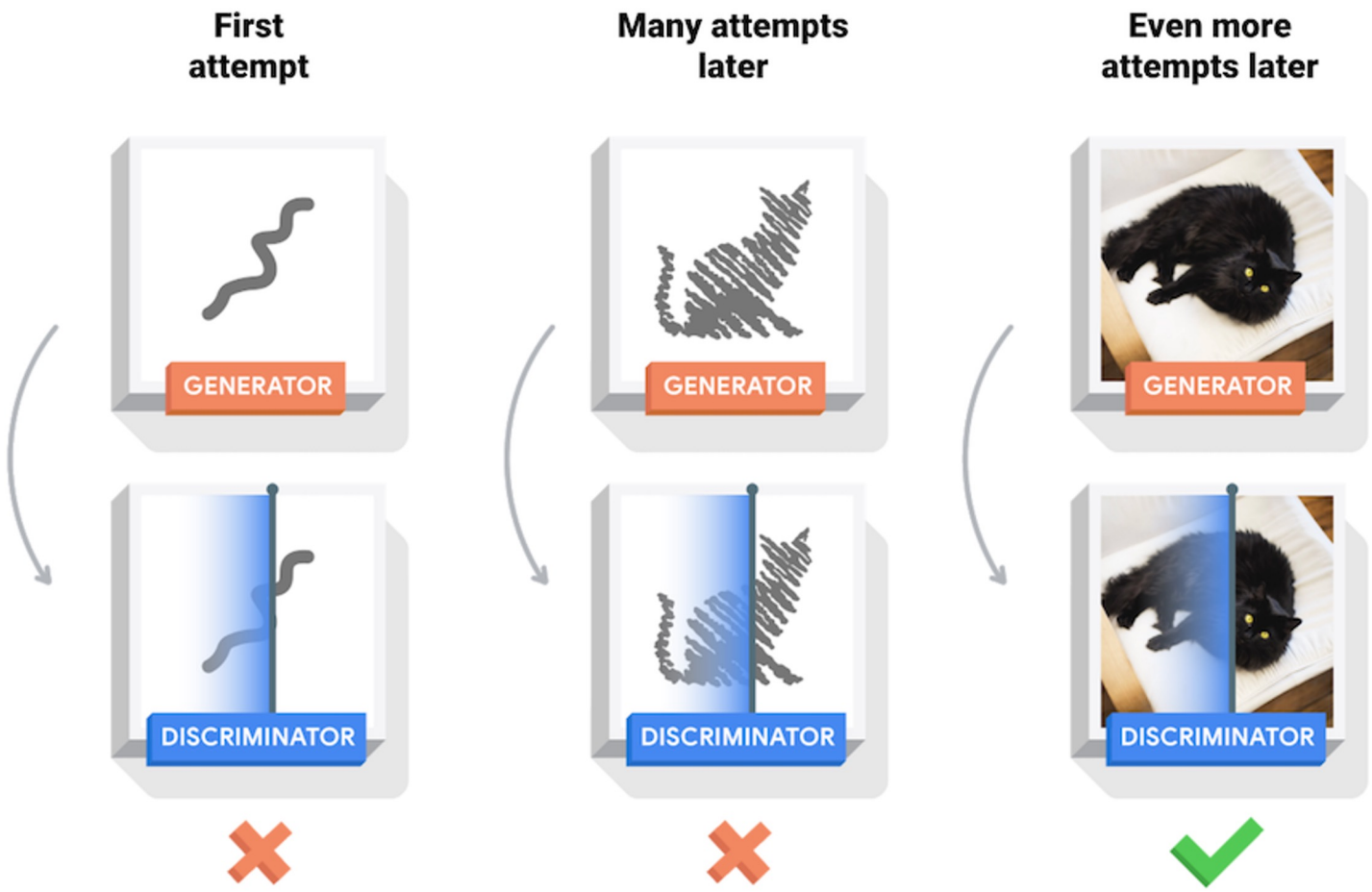
A GAN is defined by the following min-max game

$$\min_G \max_D V(D, G) = \mathbb{E}_X \log D(X) + \mathbb{E}_Z \log(1 - D(G(Z)))$$

- D wants $D(X) = 1$ and $D(G(Z)) = 0$
- G wants $D(G(Z)) = 1$



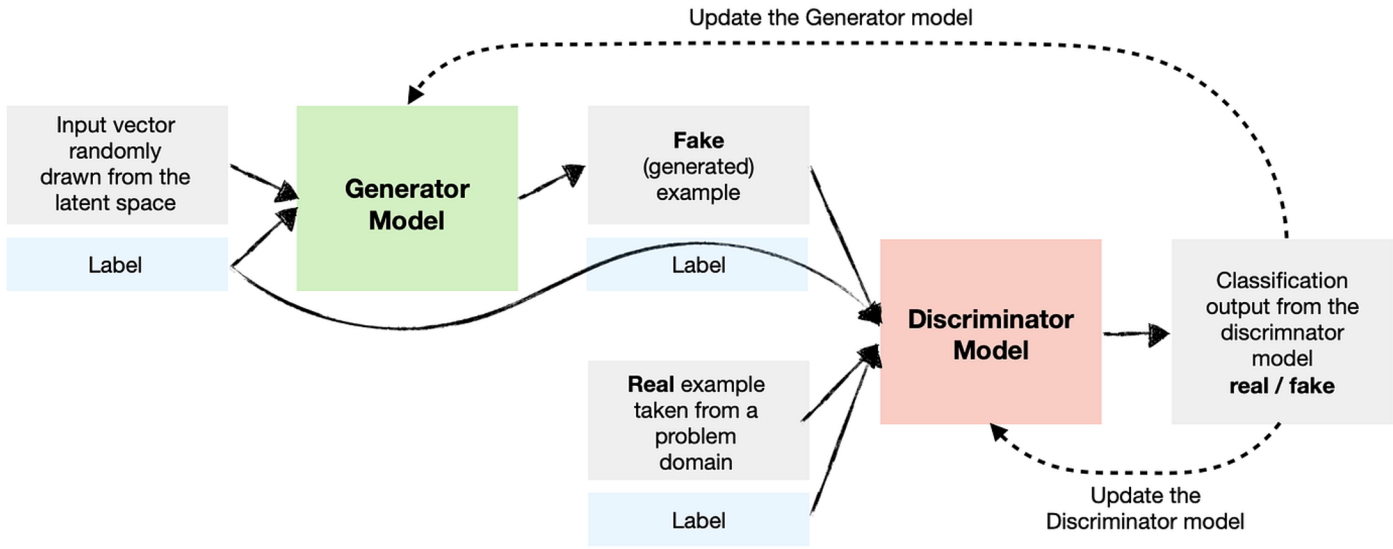
Adversarial Architecture



Conditional GANs

GAN /w Labels as Input Too

- We don't want our GAN to just learn P(X); it needs to learn P(X|Y).
- This is a simple modification; we give two inputs to the generator, a label and a noise vector. So, it tries to produces images that match real images given a particular label, rather than images in general.
- End result is a generator that you can pass a noise vector and a label, and it spits out an image of that label.



Questions?