CONVOLUTIONAL LINEAR DISCRIMINANT ANALYSIS TECHNIQUES FOR AUDIO VISUALIZATION

A PREPRINT

Truman Purnell
Department of Mathematics
San Jose State University
truman.purnell@sjsu.edu

May 13, 2020

ABSTRACT

Visualizing audio samples in embedding space is a difficult task, its characteristics (alignment, pitch, etc.) typically rendering lower-dimensional representations too messy for analysis. In this paper, I present a novel approach to clustering such samples. Taking 35,000 clips across 35 words from the Speech Commands dataset, each sample is first converted into a 64-frequency spectrogram. These spectrograms are then pushed through a vanilla Convolutional Neural Network (CNN) before Linear Discriminant Analysis (LDA) is applied to the final layer. We see clean clustering across starkly different words, and some overlap when investigating those sharing phonetic features (e.g. rhyming).

1 Introduction

Recognizing human speech, with its rich and complex waveforms, has forever represented a sort of holy grail for computer scientists. Many decades of work by legions of dedicated engineers has, until very recently, produced very little in the way of understanding. Only with the advent and adoption of Deep Neural Networks (and their much relied upon GPUs) has the problem become tractable. In this paper, I will demonstrate novel techniques to reduce the dimensionality of raw audio samples taken from the Speech Commands dataset. In reducing dimensionality, we are able to visualize and cluster such samples in simple, meaningful diagrams. These diagrams may function as upstream systems for eventual speech recognition algorithms, or provide a view into the underlying, higher-dimensional structure of our data.

2 The Problem

Dimensionality reduction naively applied to human speech produces a garbled mess. Consider a matrix $\mathbb{R}^{m\times l}$ with m audio clips each comprised of l samples. Employing a standard dimensionality reduction technique such as Principle Component Analysis (PCA) yields the results in Figure 1. We see minimal clustering, the datapoints overlaying each other meaninglessly as expected. This approach was doomed from the start, which we can see by considering the columns of our data matrix. Each column corresponds to a particular time step (one of 16,000) which define, per the PCA formulation, features of our audio samples. We should expect to see clustering if similar features have similar values, and that's the problem. Most simply, because phonetic features can appear at any time during our sample, we should not expect to find meaningful clusters.

How can we meaningfully reduce the dimensionality of our dataset so as to enable visualizations and clustering? This is the problem we are faced with.

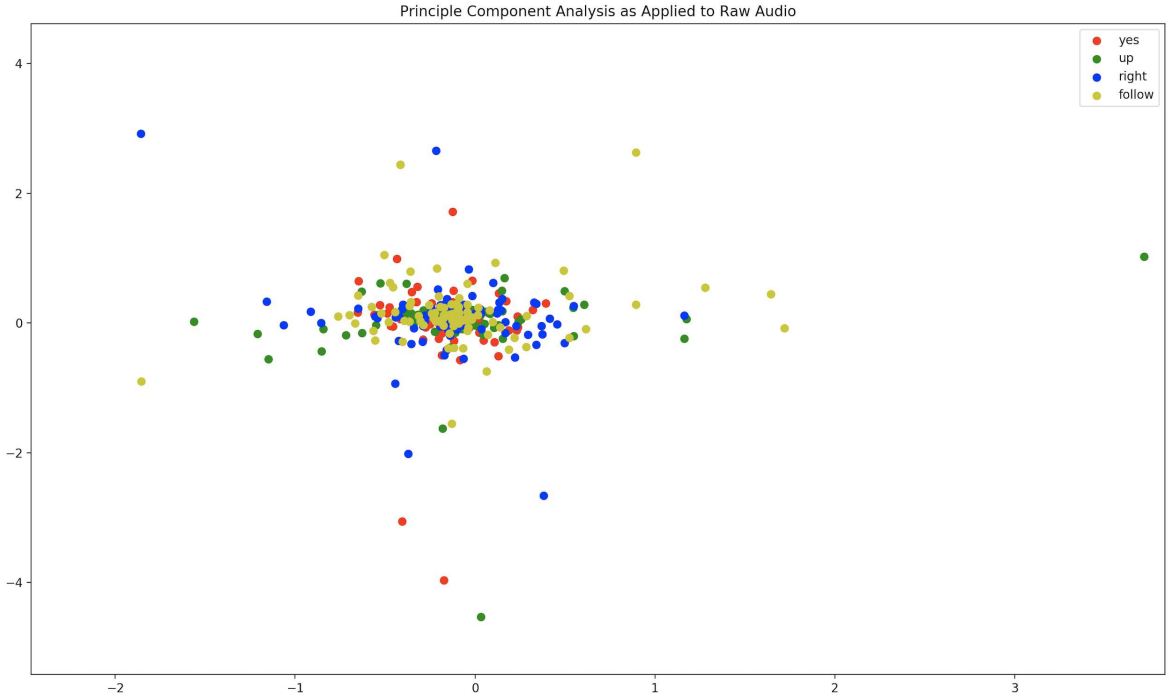

Figure 1: Principle Components of 4 words from the SpeechCommands dataset

3 The Dataset

The dataset used in this paper is known as Speech Commands. Consisting of 35 words with roughly 2000 single-second audio clips per, the dataset is moderately sized and provides a nice entry point for those looking to get started with machine learning. Table 1 contains a list of all provided words and their example counts.

The speech samples come from thousands of individuals, collected through the curator's website. As such, there is enormous variability in the recordings. A few clips are simply unintelligible, others scratchy, which poses a significant challenge for our learning algorithms. The dataset for `Speech Commands` may be found at

`http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz`

As stated previously, working with unprocessed audio is not ideal for dimensionality reduction. The shift, pitch and potentially dilation invariant nature of such data must be dealt with before we can achieve reasonable embeddings.

4 The Algorithm

I propose a three part algorithm to reduce the dimensionality of any arbitrary subset of k words ($k < 10$) from the Speech Commands dataset. We limit $k < 10$ due to the visualization constraints imposed by such high-dimensional data. Any more than 10 samples and our plots become nonsense. We first convert our unprocessed audio clips into spectrograms, running these through a 5-layer Convolutional Neural Network (CNN) before their activations meet the Linear Discriminant Analysis (LDA) algorithm in the final step.

4.1 Spectrograms

Spectrograms offer a much richer view into our audio samples. It was Joseph Fourier, among others, who first demonstrated that essentially any signal can be expressed as a sum of sinusoids of differing frequencies. As such, we can take each audio clip $A \in \mathbb{R}^l$ and transform it, using the Discrete Fourier Transform, into a form $A' \in \mathbb{C}^{f \times l}$ where f represents the granularity of our frequency components and is a parameter of our choosing. A large f means we would like to distinguish between a very fine range of frequencies, whereas a smaller f indicates we are not as concerned with

Table 1: Word Frequencies

backward	1664
bed	2014
bird	2064
cat	2031
dog	2128
down	3917
eight	3787
five	4052
follow	1579
forward	1557
four	3728
go	3880
happy	2054
house	2113
learn	1575
left	3801
nine	3934
no	3941
off	3745
on	3845
one	3890
right	3778
marvin	2100
seven	3998
sheila	2022
six	3860
stop	3872
three	3727
tree	1759
two	3880
up	3723
visual	1592
wow	2123
yes	4044
zero	4052

precisely which frequency contributed to our signal, but rather which general band it came from. We chose $f = 64$ in our experiments, mostly to speed up training.

The Discrete Fourier Transform is defined

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{\frac{-i2\pi kn}{N}} \tag{1}$$

where X_k is known as the Fourier coefficient associated with frequency k. I chose arbitrarily to simply take the norm of each complex number in our resulting matrix, discarding some phase information but bringing us back to $\mathbb{R}^{f \times l}$ for subsequent calculations. Notice we have gained information on the order of f for each sample. See Figure 2 for an example of spectrogram.

4.2 Convolutional Neural Networks

To address at this point remain the nasty invariances associated with our representation of the data. The spectrograms, though denser informationally, suffer yet from potential shifts and dilations. Simply, the target word could appear at any timestep, at any pitch, and any rate (slowed or quickened in its utterance). Convolutional Neural Networks, or CNNs, theoretically solve each of these issues.

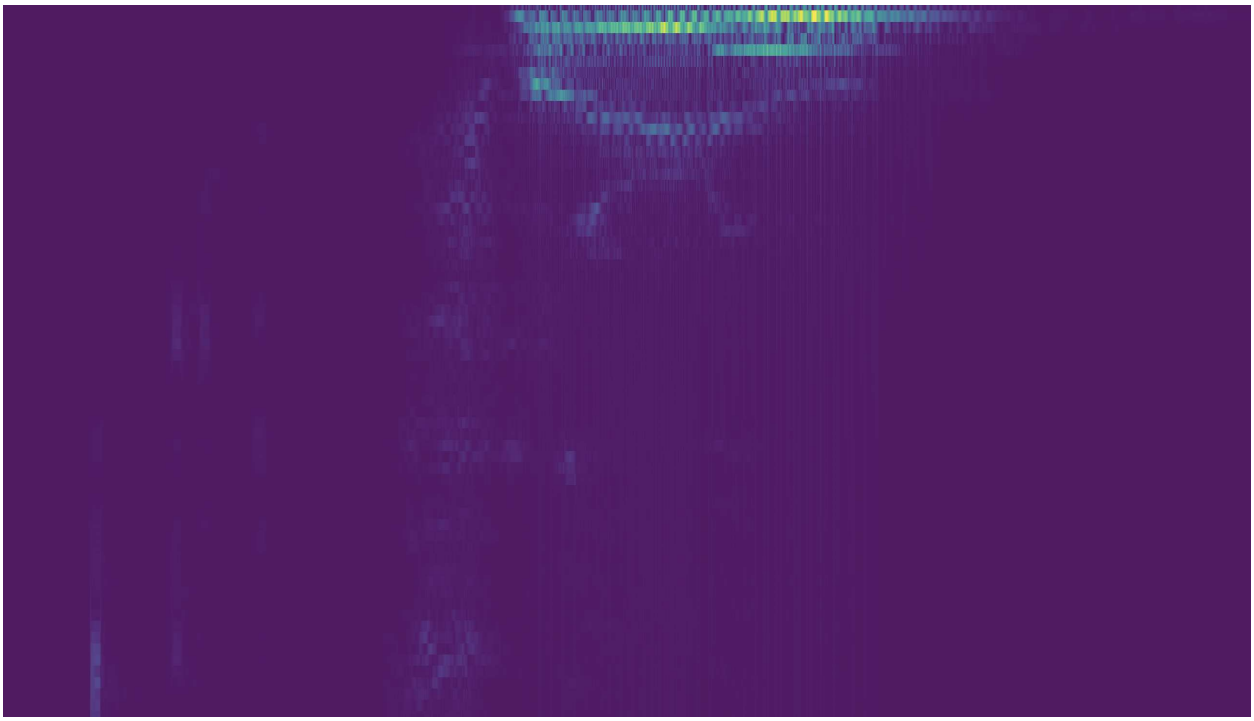

Figure 2: A spectrogram of the word "follow" with $f = 64$ frequency bins

CNNs are represented as layered, parametric feature masks (each layer containing a stack of such masks) which are scanned across an input and multiplied element-wise at each step. The results of these multiplications (activations, as they are termed) themselves define a layer, which after applying some sort of non-linear function (e.g. $f(x) = max(x, 0)$) become the inputs to downstream layers. The parametric feature masks have their values iteratively adjusted until some loss function is minimized. In these experiments, I chose to minimize the categorical cross-entropy loss, designed for datasets wherein each sample has only one correct, unambiguous label. This may not be the best loss function for speech data in general, as words may (and perhaps should) be confused with each other (rhymes, or homonyms) quite naturally. But as this dataset is largely free from such pitfalls, categorical cross-entropy loss was a reasonable choice. The loss is defined

$$L(y, \hat{y}) = -\sum_{i=0}^{N}\sum_{j=0}^{c} y_{ij} \cdot log(\hat{y}_{ij}) \tag{2}$$

where y is the true label for each word's spectrogram (defined as a one-hot vector) and $\hat{y}$ is the networks current prediction of which word a particular spectrogram belongs. In this case, c is the number of words in our training set and N is the number of examples we choose to train on.

The weights, once adjusted to predict the correct word from its spectrogram, have a very nice interpretation. They are themselves small bits of their input layers, maximally excited (or suppressed) by some set of features therein. And, as these features masks are slid across their entire input layer, these features can be detected anywhere (or anytime). This allows the CNN to detect input word characteristics at any pitch, at any time step, and at any dilation (provided it has seen enough dilated examples). Table 2 contains a list of the layers' sizes, activations, and weight volumes. All of these values are hyperparameters and can be tuned to increase the performance of our model.

4.3 Linear Discriminant Analysis

Once our 5-layer CNN has learned reasonable representations for our input data, we turn to Linear Discriminant Analysis (LDA) to find 2-dimensional embeddings. LDA is a dimensionality reduction technique for labeled training data. It attempts to find a k dimensional subspace wherein the variance for embedded points with like classes is small,

Table 2: Summary of CNN model used for initial experiments

Layers (type)	Output Shapes	Parameters
conv2d (Conv2D)	(64, 64, 32)	320
activation (Activation)	(64, 64, 32)	0
conv2d_1 (Conv2D)	(62, 62, 32)	9248
activation_1 (Activation)	(62, 62, 32)	0
max_pooling2d (MaxPooling2D)	(31, 31, 32)	0
dropout (Dropout)	(31, 31, 32)	0
conv2d_2 (Conv2D)	(31, 31, 64)	18496
activation_2 (Activation)	(31, 31, 64)	0
conv2d_3 (Conv2D)	(29, 29, 64)	36928
activation_3 (Activation)	(29, 29, 64)	0
max_pooling2d_1 (MaxPooling2	(14, 14, 64)	0
dropout_1 (Dropout)	(14, 14, 64)	0
flatten (Flatten)	(12544)	0
dense (Dense)	(35)	439075
activation_4 (Activation)	(35)	0

and the variance for centroids of unlike classes is large. In doing so, tight interclass clusters spaced far apart from each other are encouraged.

In the context of our problem, we are interested in reducing the dimensionality of the matrix of activations (outputs after running our examples) from the final layer (dense) of our trained CNN. We call this matrix $E \in R^{n \times c}$ where n is the number of examples passed through (roughly 35,000, see experiments section) and c is the number of words that particular network was asked to distinguish between. It is this matrix to which we apply LDA. Before describing the results, I will briefly outline the general LDA algorithm. Letting

$$x_1, \ldots, x_n \in R^d$$

we are looking for a unit vector $\mathbf{v}$ that discriminates between c classes per the description above. As such, we define

$$a_j = \mathbf{v}^T \mathbf{x}_j,\ j = 1, \ldots, n$$

$$\mu_i = \frac{1}{n_i} \sum_{\mathbf{x}_j \in C_i} a_j$$

$$s_i^2 = \sum_{\mathbf{x}_j \in C_i} (a_j - \mu_i)^2$$

to be the projections of all data points onto onto $\mathbf{v}$, the projected mean (centroid) associated with class i, and variance of the projected points (also associated with class i). From here, we establish the formulation (using $c = 2$ as an example)

$$\max_{\mathbf{v}:\|\mathbf{v}\|=1} \frac{(\mu_1 - \mu_2)^2}{s_1^2 + s_2^2}$$

which we further massage

$$\begin{aligned}(\mu_1 - \mu_2)^2 &= (\mathbf{v}^T \mathbf{m}_1 - \mathbf{v}^T \mathbf{m}_2)^2 = (\mathbf{v}^T (\mathbf{m}_1 - \mathbf{m}_2))^2 \\ &= \mathbf{v}^T (\mathbf{m}_1 - \mathbf{m}_2) \cdot (\mathbf{m}_1 - \mathbf{m}_2)^T \mathbf{v} \\ &= \mathbf{v}^T \mathbf{S}_b \mathbf{v}\end{aligned}$$

where

$$\mathbf{S}_b = (\mathbf{m}_1 - \mathbf{m}_2) \cdot (\mathbf{m}_1 - \mathbf{m}_2)^T \in R^{d \times d}$$

which we call the between-class scatter matrix. The purpose of these manipulations is to get our data into a convenient form for industry software (matlab, NumPy) to chew on. We now look to make similar progress collecting in-class projections into a matrix. Continuing

$$\begin{aligned} s_i^2 &= \sum_{\mathbf{x}_j \in C_i} (a_j - \mu_i)^2 = \sum_{\mathbf{x}_j \in C_i} (\mathbf{v}^T \mathbf{x}_j - \mathbf{v}^T \mathbf{m}_i)^2 \\ &= \sum_{\mathbf{x}_j \in C_i} \mathbf{v}^T (\mathbf{x}_j - \mathbf{m}_i) \cdot (\mathbf{x}_j - \mathbf{m}_i)^T \mathbf{v} \\ &= \mathbf{v}^T \left[\sum_{\mathbf{x}_j \in C_i} (\mathbf{x}_j - \mathbf{m}_i) \cdot (\mathbf{x}_j - \mathbf{m}_i)^T \right] \mathbf{v} \\ &= \mathbf{v}^T \mathbf{S}_i \mathbf{v} \end{aligned}$$

where $\mathbf{m}_i$ is the unprojected centroid for class i and

$$\mathbf{S}_i = \sum_{\mathbf{x}_j \in C_i} (\mathbf{x}_j - \mathbf{m}_i) \cdot (\mathbf{x}_j - \mathbf{m}_i)^T$$

is denoted the within-class scatter matrix again for class i. To arrive at the cumulative within-class scatter matrix across all c classes, we simply sum them up. We call this cumulative scatter matrix $\mathbf{S}_w$

$$\mathbf{S}_w = \sum_{i=1}^{c} \mathbf{S}_i$$

Now, with positive semi-definite matrices $\mathbf{S}_w$ and $\mathbf{S}_b$ in hand, we can properly formulate our problem

$$\max_{\mathbf{v}:\|\mathbf{v}\|=1} \frac{\mathbf{v}^T \mathbf{S}_b \mathbf{v}}{\mathbf{v}^T \mathbf{S}_w \mathbf{v}}$$

Simply, find the direction $\mathbf{v}$ so as to maximize the between-class centroid variance and minimize the within-class sample variance. Problems of this form are known as generalized Rayleigh quotients and can be solved fairly easily assuming $\mathbf{S}_w$ is non-singular. The solution is given as

$$\mathbf{S}_w^{-1} \mathbf{S}_b \mathbf{v}_1 = \lambda_1 \mathbf{v}_1$$

In our case, $\mathbf{S}_w$ is in fact stably invertible (with a reasonably small condition number). If $\mathbf{S}_w$ is singular, there are a few approaches. Perhaps the simplest takes the pseudoinverse of $\mathbf{S}_w$, yielding

$$\mathbf{S}_w^{+} \mathbf{S}_b \mathbf{v}_1 = \lambda_1 \mathbf{v}_1$$

This derivation of LDA assumed two classes, but what if we are dealing with more? The intuition is exactly the same. The only difference comes with $\mathbf{S}_b$, which becomes a weighted average of the projected centroids

$$\mathbf{S}_b = \sum n_i (\mathbf{m}_i - \mathbf{m}) \cdot (\mathbf{m}_i - \mathbf{m})^T$$

with $\mathbf{m}$ in this case being the global centroid of the dataset.

5 Experiments and Results

I ran three main experiments on the Speech Commands dataset. Firstly, as alluded to earlier, I ran Principle Component Analysis (PCA) on a truncated version of the raw data matrix $G \in R^{35000 \times 16000}$. This produced the results show in Figure 1, which were lackluster to say the least. This occurs for a variety of reasons, but it's the three invariances, pitch, time, and dilation that give PCA fits.

Secondly, I wanted to observe, as a baseline, how LDA performed on the same raw data matrix G. Perhaps the idiosyncrasies of audio were cured simply with labels. This produced the plot show in Figure 3. The figure is interesting. Clearly, the information provided by the labels is enough to give LDA a fighting chance. We spot clear clustering of the words "Sheila" and "Tree". But upon further inspection, is this really what we want? These words, on the level of phonemes, share a prominent 'ee' sound. It seems they really should have some overlap. Looking closer, we see the rest of the words stacked atop each other in the familiar, mushy style of PCA. Surely there is something we can do to more cleanly separate the classes and preserve phonetic features.

In the third experiment, I dug into the trained CNN to extract pre-embeddings from the output of its second to last layer. The CNN, as discussed earlier, does a wonderful job retaining our sought after low-level features, which theoretically grow ever more descriptive as the data flows through its layers. Remember, the CNNs task was to discriminate between classes itself! It therefore created it's own representations to do so. After taking roughly 35,000 training examples and sending them through our fully-trained network, we end up with our previously defined matrix $E \in R^{n \times c}$ where again $n = 35,000$ and $c = 35$ (number of classes). We then compute LDA. The results at this point are much improved. Although the in-class variance is larger and between-class centroid distance is smaller, the plot in Figure 4 demonstrates the preservation of important audio characteristics that any downstream system would appreciate. Words that rhyme e.g. "go" and "no" overlap considerably.

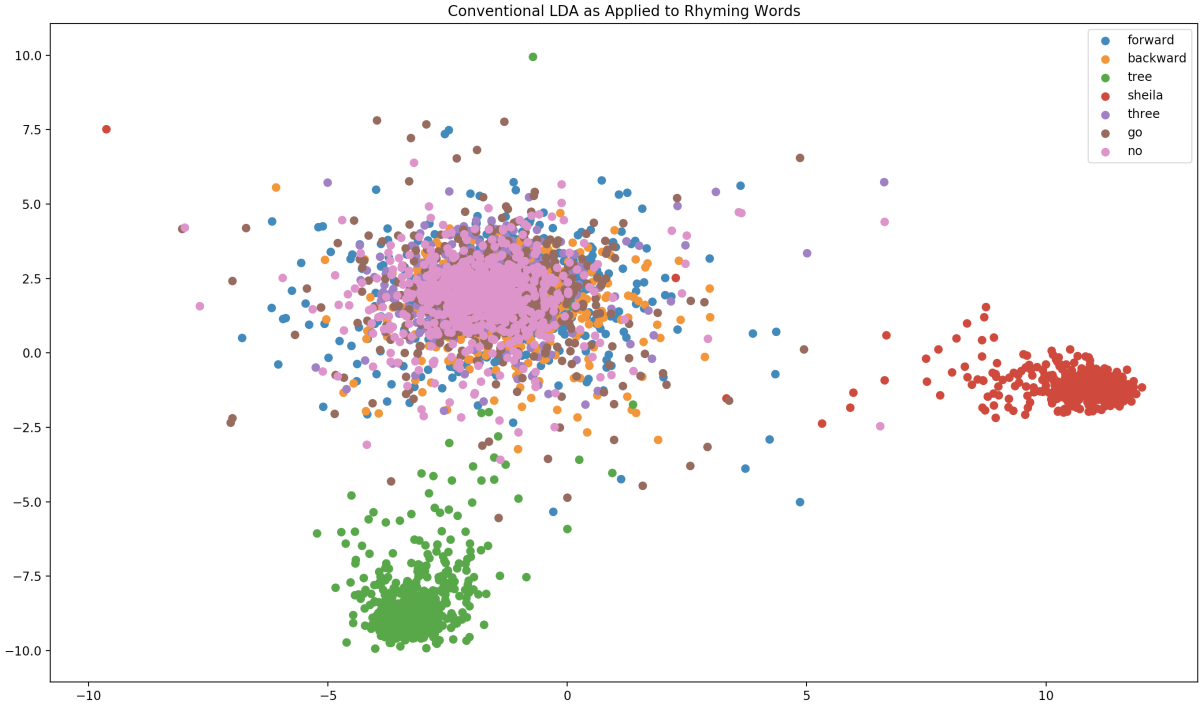

Figure 3: A plot of the first and second discriminating directions for rhyming words in the Speech Commands dataset from Conventional LDA

6 Conclusion

We have seen through preparation and experiment the difficulties presented by raw audio. The paper outlined significant preprocessing and compute-heavy convolutional models all to barely differentiate between quasi-rhyming words. That said, we were able to spot interesting trends in our data, and learn perhaps a thing or two about dimensionality reduction and modeling along the way.

In the proposal stage of this project, I indicated an approach to speech clustering that failed to pan out. After creating spectrograms, I ran PCA on the transposed results. The spectrograms, recall, took the shape of a matrix $\in C^{f \times l}$ where $f = 64$ and $l = 16,000$ before preprocessing. With each timestep a row in the transposed spectrogram, the projections formed a sort of arc tracing out wildly in two dimensions. Though intriguing, and potentially a line of future research, I

Figure 4: A plot of the first and second discriminating directions for rhyming words in the Speech Commands dataset from Convolutional LDA

could not seem to draw any meaningful clusters from such a representation. Additional ideas for further experimentation include removal of the labels. They are a crutch at the end of the day, one that humans almost certainly do not rely upon.

7 Tools

Tensorflow, Matplotlib, and NumPy were used extensively throughout my research.

References

[1] Keras-Team. A simple deep CNN on the CIFAR10 small images dataset. https://github.com/keras-team/keras/blob/master/examples/cifar10_cnn.py

[2] Peltarion. Categorical cross-entropy loss function description. https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy

```python
from tools.loaders import SpeechCommands
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

# Utility Functions
def show(title):
    plt.title(title)
    plt.legend()
    plt.show()


def sample(items, size):
    sample_range = range(len(items))

    return [(items[i], i) for i in np.random.choice(sample_range, size, False)]


# Setup
words = SpeechCommands.all_words

rhymes = [('forward', 9), ('backward', 0), ('tree', 28),
          ('sheila', 24), ('three', 27), ('go', 11), ('no', 17)]

colors = ['tab:blue', 'tab:orange', 'tab:green',
          'tab:red', 'tab:purple', 'tab:brown', 'tab:pink']

# Parameters
samples = 4
dim_embed = 2
num_files = 1000
num_classes = len(words)
num_examples = num_classes * num_files
labels = np.arange(num_examples) // num_files


# Training Data
x_raw = SpeechCommands.load_raw(words, num_files)
x_train = tf.expand_dims(SpeechCommands.load_spec(words, num_files), -1)
y_train = tf.keras.utils.to_categorical(labels)

# Model
model = tf.keras.Sequential()

model.add(tf.keras.layers.Input(shape=x_train.shape[1:]))

model.add(tf.keras.layers.Conv2D(32, (3, 3), padding='same'))
model.add(tf.keras.layers.Activation('relu'))

model.add(tf.keras.layers.Conv2D(32, (3, 3)))
model.add(tf.keras.layers.Activation('relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.Dropout(0.25))

model.add(tf.keras.layers.Conv2D(64, (3, 3), padding='same'))
model.add(tf.keras.layers.Activation('relu'))

```

```
model.add(tf.keras.layers.Conv2D(64, (3, 3)))
model.add(tf.keras.layers.Activation('relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.Dropout(0.25))

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(num_classes))
model.add(tf.keras.layers.Activation('softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy')
model.fit(x=x_train, y=y_train, batch_size=100, epochs=8)

# Convolutional LDA Plot
layer_model = tf.keras.Model(inputs=model.input,
                             outputs=model.layers[-2].output)  # dense

predictions = layer_model.predict(x_train).reshape(num_examples, -1)

lda = LDA(n_components=dim_embed)
embeddings = lda.fit_transform(predictions, labels)

for color, (word, i) in zip(colors, rhymes):
    plt.scatter(embeddings[labels == i, 0],
                embeddings[labels == i, 1], c=color, label=word)

show('Convolutional LDA as Applied to Rhyming Words')

# Conventional LDA Plot
raw_clips = np.vstack([x_raw[labels == i] for _, i in rhymes])
raw_labels = np.vstack([labels[labels == i] for _, i in rhymes]).ravel()

lda = LDA(n_components=dim_embed)
raw_embeddings = lda.fit_transform(raw_clips, raw_labels)

for color, (word, i) in zip(colors, rhymes):
    plt.scatter(raw_embeddings[raw_labels == i, 0],
                raw_embeddings[raw_labels == i, 1], c=color, label=word)

show('Conventional LDA as Applied to Rhyming Words')

```