Progress Log

Jeff Hykin

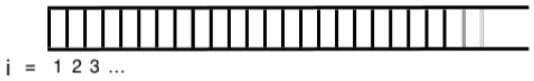
July 14, 2020

July 9th 2020

1 Finding Happiness

1.1 Problem Definition

Consider a video with infinite sequence of frames, where each frame is enumerated with an index $i \; s.t. \; i \in I, \; I \subseteq \mathbb{W}$. Suppose, in addition to being infinite, that any frame can be accessed in a constant amount of time.



In this scenario $H(i)$ is a function $H : \mathbb{W} \mapsto \{0, 1\}$ where
$H(x) = 1$ indicates a frame is a "hit"
$H(x) = 0$ indicates a frame is a "miss"

In our particular application we define a "hit" as a frame containing a face with a happy expression.
Ideally an algorithm $A(n)$ would iteratively select the index with the highest likelihood of being a hit, with n being the number of iterations. This algorithm would return a set of indices $i_a \in A(n)$, where each index was a hit $\forall i_a, H(i_a) = 1$. The problem is framed as maximizing the score of an algorithm $S(A, n)$, which is defined as the proportion of hits per iteration

$$S(A, n) = \frac{\sum^{i \in A(n)} H(i)}{n}$$

In addition to this, it is also desirable to minimize the size of the largest index k that is accessed, as a measurement of "wasted" frames defined as $W(n)$.

$$k \in A(n) \;\; s.t. \;\; k \leq a, \forall a \in A(n)$$

$$W(n) = \sum_{j=1}^{k} H(j) - S(A, n)$$

1.2 Building Probability Distributions

To generate the probability of a hit $P_H(i)$ for an index i, we must work to answer a few questions.

- What is $P_H(i)$, given no other information?
- What is $P_H(i)$, given a nearby frame is a hit?
- What is $P_H(i)$, given a nearby frame is a miss?
- What is $P_H(i)$, given c_h adjacent hits?
- What is $P_H(i)$, given c_m adjacent misses?

Before making any statistical inferences, at least some prior data will be needed. Suppose sub-sequences of various lengths are taken from the video and put into a set R. (Suppose also that these sequences are a representative random sample, and that the hit/miss value has been pre-found for every frame in every sub-sequence.)

To answer the first question, each sequence in R must be broken down into a set of 1-frame sequences, represented as F_1. The probability is then found as the hit ratio HR of the set of frames $HR(F_1)$

$$HR(X) = \frac{\sum^{i \in X} H(i)}{|X|}$$

As for the subsequent questions, lets consider an example of these sequence sets F_1, F_2, F_4.

```
F_1 = {  [ 12233 ],                    [ 1 ],        [ 9 ],            }
F_2 = {  [ 12233,12234 ],              [ 1,2 ],      [ 9,10 ],         }
F_4 = {  [ 12233,12234,12235,12236 ], [ 1,2,3,4 ], [ 9,10,11,12 ],  }
```

(Note, sets with sequences with a count larger than 1 need a method of selection, but we will not address that here.)
For each of these sets $F_j \in F$, lets build a corresponding distribution $D_j \in D$. This can be done by replacing each sequence with the hit-ratio of all indices in that sequence.

```
D_1 = [  0,       1,    1,  ] # assuming H(12233)=0, H(1)=1, H(9)=1
D_2 = [  0.5,     0.5,  1,  ] # assuming H(12234)=1, etc
D_4 = [  0.75,    0.25, 1,  ] # etc
```

Mathematically represented as

$$D_j = [x | x = HR(s), \forall s \in F_i]$$

Given these distributions corresponding to sequences of length i, we can now move on.

1.3 Calculating Probabilities

Given that we have a c_h close-by hits, and c_m close-by misses, we can determine a distribution D_k that would be most suited for prediction. Before this is done however, consider the following analogy.

Instead of merely a ratio, imagine each elment of D_k to be a bag containing k marbles. Given that the bag has a ratio, imagine that ratio b_r as indicating the bag's ratio of green and red marbles, green representing a hit and a red representing a miss. Similarly consider c_h as a quantity of green marbles, and c_m a quantity of red marbles.

By treating each bag as a binomial distribution, we can compute $P_d(b)$ as the likelihood that c_h and c_m were drawn from each bag $b \in D_k$.

$$P_d(b) = b_r^{(c_h)} * (1 - b_r)^{(c_m)}$$

These binomial probabilities can then be normalized and used as weights for each ratio.

$$P_{total} = \sum^{\forall b \in D_k} P_d(b)$$

$$P_H(i|D_k, c_h, c_m) = \sum^{\forall b \in D_k} b_r * (P_d(b)/P_{total})$$

In light of the analogy, the task of selecting a D_k is simplified to selecting the ideal bag-size k. Given that we are asking for the probability that c_h green, c_m red, and one unknown i are being drawn from a bag, k should at least be $k \geq c_h + c_m + 1$. To gain the highest accuracy possible, it would also be ideal to minimze the number of unknown marbles, meaning $k = c_h + c_m + 1$. If such a D_k is not avaible, then the next closest ideal is $min(|k - (c_h + c_m + 1)|)$.

1.4 Infinte Comparison

Finding the frame with the highest likelihood i_m by comparing against all frames in an infinitely long video might be considered computationially expesive. Thankfully, convergence and the Law of Large Numbers, allows us to sidestep this problem entirely. As the sequence size k increases, the ratios in D_k are guarenteed to converge on the population ratio of the entire video. Since the frame with the highest probability is the only one of interest, the confidence needed for convergence towards the average probablilty can be relatively low. This provides both a theoretical and pratcial bound w for creating $|D| = w \ \ s.t. \ D_j \in D$.

1.5 Time Complexity and Optimization

The time complexity of $A(n)$ is then $O(n * (O(H) + m))$ where n is the number of iterations, m is the size of the largest D_j, and $O(H)$ is the time complexity of the hit detection function. A tradeoff can be made to reduce m to nearly 1 at cost of space complexity. This is done by caching the results of $P_H(i|D_k, c_h, c_m)$, until all common combinations needed for convergence have been cached.

1.6 Python Implmentation

((*TODO: add comments to the code, create an online repository, then paste code *))

1.7 Practical Application (Conclusion)

((TODO))

Idea: localized "forgetful" statistics

June 21 2020

```
#
#
# Finding Faces and the Game of Battleship
#
#
```

Optimally finding frames containing faces may seem unrelated to the board game of Battleship. The key hides in the pedestrian observation that frames with faces most often occur in sequences. For empirical validity, this was confirmed based on a data set of 360 randomly selected videos. The mapping between these two problems is quite useful since the game of Battleship is known to have a mathematically optimal strategy for winning, as shown in the article "Battleship and the Negative Hypergeometric Distribution" by Melanie A. Autin and Natasha E. Gerstenschlager. In the game of Battleship, each player has a 10x10 grid of cells along with ships of various lengths that can be placed on the grid. The players take turns guessing the locations of

the opponent's ships, one cell at a time. A sequence of frames containing faces can be thought of as a ship, and the video itself can be thought of as a one dimensional grid. The downside is that, with videos, the ships are of an unknown length, and unknown quantity. On the upside however, the videos don't shoot back. Having a minimum ship length of one, combined with an unknown number of ships, requires a player to guess every possible cell before declaring absolute victory. Thankfully enough however, the case of video processing does not need to find every last rowboat and buoy. If we increase the minimum ship length to merely two units, the absolute victory condition goes from needing every cell, to only needing every-other cell. A minimum duration of 1 second (approximately 32 frames) would likely be useful for training machine learning models. And, with that minimum ship size of 32, along with a single-dimension grid, the game becomes orders-of-magnitude easier to win. Not only in terms of never missing any major ships, but also in terms of having the highest possible ratio of hits per guess.

Optimizing the game of Battleship requires applying a probability density function to every cell in the grid. This involves simulating every possible permutation of all ships in order to find the cells with the highest probability of containing a ship. The end result is that center of the grid (e.g. video) will always be selected as the most likely spot. This is because ships have fewer possible configurations involving only edge-cells, with many more possible ways of being placed in cells closer to the center. After choosing the middle frame, a miss would cause a cascading reduction in the probability of all nearby cells. Furthermore, the sum total of probabilities from all cells would also drop because it is certain that a ship spanning the whole grid (or even just half of the grid) cannot be not present. Supposing that the initial guess is a hit however, would cause the exact inverse reaction. Nearby cells would cascadingly increase in probability, and the sum total of probabilities for the video as a whole will rise significantly. This process can then be repeated continually by always selecting the cell with the highest probability.

Higher Order Battleship

The macro process of finding faces is not a single game of Battleship but instead one game of Battleship per video, all of which are (theoretically) being played simultaneously. If the outcome of a particular game is looking grim, simply start (or resume) a different game. More specifically, if the highest probability of a hit in a particular video is lower than the probability of a hit on a new video, then a new video should be evaluated instead. Beyond this improvement, the average sizes and frequencies of ships can also be empirically estimated as more data is gathered, allowing for an increasingly accurate probability density function. A second-order game of Battleship can also be played by evaluating the probability of ships being near other ships. Most notably, this method can provide an empirical estimation of its own efficiency without needing to be trained on a large data set.

Implementation

A practical implementation of these strategies however, will be much more difficult than it may first appear. Some videos are actually the same image on every frame, or the same image but merely translated or rotated. This will cause face detections on every frame even when the video is undesirable. The probability of inaccurate reporting, such as reporting a miss when it was in fact a hit, will also need to be taken into account. There are likely many other practical barriers yet to be discovered.

June 18 2020

Through the code in this repository we provide access to a undirected graph `G = (V, E)` of YouTube links. Each node `v` $\in$ `V` is a YouTube video, and each edge `e` $\in$ `E` is a relationship to a related video. For every video `v`, in addition to edges, we also provide access to the duration, frame-rate, and frame-data. The graph is stored using a hash map where each key is a unique video ID provided by YouTube. The number of nodes `|V|` in this graph is growing, but can be expected to be in the thousands. The average number of edges per node `|e|` is approximately 22. The details of how to explore the graph are explained in the following source:
https://github.com/jeff-hykin/iilvd_interface/blob/master/README.md

June 16 2020

A 3 page installation/setup/usage guide was created for A&M students to be able to be able to use the video database. The guide is here:
https://github.com/jeff-hykin/iilvd_interface/blob/master/README.md

June 15 2020

The database has now been running on the GPU server with GPU optimizations, which is approximately x8 faster than before. Statistical logging was added to monitor the process as it unfolded.
Download Time: 1% of duration Download Size: 7.2 Mb per minute of duration
Time spent finding faces: (46%) 138 sec per 5 min of video Time spent finding emotion: (22%) 66 sec per 5 min of video Time spend saving to DB: (19%) 57 sec per 5 min of video Other code (13%)

June 13th 2020

The collection system has now been fully running for 48 hours on my local machine. At the current average rates, it will be 17 days before 1000 videos has been collected. The code is running without GPU optimizations, and there are several opportunities for major improvements.

June 9 2020

The system for continually labeling videos is now put in place. The process is as follows:
- A priority system for collecting videos is established
- Videos are served in a stream-like fashion via python generators
- Videos are downloaded as .mp4 files as needed
- Each frame of the video is analyzed individually
- Using dlib detectMultiScale, all faces in a frame are discovered
- Light preprocessing is performed on the cropped face images
- The processed images are then run through 9-scale emotion detection model
- The predicted probabilities for all 9 emotions are recorded
- That data is then saved, over the network, to the database

The system has been tested on various individual data, but requires a day or more of testing before being given the chance to run continually and possibly corrupt the entire database. There is a current format for saving data, however will be updated to better fit with additional data. Also note the emotion detection model used was obtained from a fellow researcher in the Information Innovation Lab.

June 7&8 2020

Work was completed on a dynamic video caching and selection system for getting videos into the emotion detection model. The selection system works as a class 'VideoSelector' that can be given priorities of which videos to retrieve first. For example, 'VideoSelector().is_downloaded.has_related_videos.retrieve()' will return a generator that only outputs videos matching that criteria. That example only has a single priority. This example:
'VideoSelector().is_downloaded.then.has_related_videos.retrieve()' has two priorities, the first one is downloaded videos. Once the list of downloaded videos is exhausted it falls back on its second priority, which is videos that are related to some other videos. Every video that is not already downloaded is automatically downloaded before being returned. This system should be flexible enough to allow for dynamic queues in the future, where videos are ordered based on the evolving likelihood of each video's usefulness.

June 4&5 2020

Updated the database to both be crash-tolerant and reactively update to changes in the code. The query code was improved and now all of the MongoDB query selectors are supported directly in both Ruby (data scraping) and Python (machine learning). This will allow for the next step of quantifying statistics about the data, and validation of data.

June 2 2020

The related video collection script ran successfully for over 24 hours. In that time it exhausted the videos that were in the database and provided relations for 66,074 videos. Code for querying the database was added, and work was done towards data-validation.

June 1 2020

Code was created and tested for recording related_videos. Collected 21218 samples that contain related video data. Validation of data still needed. There are approximately 20 related videos (edges) per video. Current video data format is as follows, loosely following the YAML format.

```
# null means not-yet-checked or unknown
# false means known-to-be-false: aka unavailable
(youtube_video_id): !map_or_null_or_false
    basic_info: !map_or_null
        duration: !seconds (duration)
        fps: !integer (fps)
        height: !pixels (height)
        width: !pixels (width)
        download_error: !true_or_null (download_error)

    related_videos: !map_or_null
        !string (related_video_id): {}  # this is a key-value pair
                                        # because there are different
                                        # kinds of relationships

    facedata_1.2: !map_or_null
        good_faces: !bool_or_null (significant_number_of_big_faces)
        is_picture: !bool_or_null (all_checked_frames_equal)
        max_repeated_face: !integer (lots_of_same_person_or_no)

    frames: !map_or_null # this is a map instead of an array because
                         # not all frames (indices) are checked
        !integer (frame_index):
            facedata_1.2:
                big_faces_>=1: !true_or_false (frame_contained_good_face)
                faces: !list
                    - "is_big?": !true_or_false
                      "x": !integer (x_position_pixels)
                      "y": !integer (y_position_pixels)
                      "width": !integer (width_pixels)
                      "height": !integer (height_pixels)
```

#

#
Database Design and Implementation
#
#
May 25 2020

This entry covers the mathematical structure, the computer science data structure, time complexities, the API implementation, and finally the software implementation of the database of videos being used.

2 Theoretical Structure

The data can be thought of as an undirected graph, where each video is a node and the edges are relationships to other videos. To implement this structure, a hash map is used. Each key in the mapping is a video id, the one provided by YouTube, and the value is a nested hash map and array structure. Here is the structure represented in a JSON-like format:

```
{
    video_1_unique_key: {
        "related_videos": {
            video_2_unique_key: null
        }
    },
    video_2_unique_key: {
        "related_videos": {
            video_1_unique_key: null
        }
    }
}
```

This structure allows for O(1) retrieval times for any node or immediate neighbor, at the cost of having the memory overhead of edges * 2. Which is justifiable since storage is cheap and time is not nearly as cheap. Note that the edges themselves are key-value pairs. This allows for adding future properties such as relationship distance/strength or adding relationship qualities, such as being related in color scheme or being related in transcription/audio content.

3 Software Implementation

This information is for those interested in reading, reproducing, or otherwise using the code. The hash map structures were additionally chosen in order to house additional (unknown) information about the video. Because of this unknown structure, the noSQL database MongoDB was chosen.
GLFS (git large file storage) is used to handle the database files, which are currently stored within the git repo itself. This will likely change if there is any collaboration on the database, but it is ideal to store it locally for a single user. Docker is used to ensure compatibility with all OS's. Within docker containers a MongoDB service is created locally, and a Node.js server is run locally to create easy API for the MongoDB service. By using the API created by Node.js, videos can be added, retrived by id, edited, or deleted by anyone on the local network. Videos that are are on que, but do not have details are stored as a null value (instead of a hash map). Videos that have been attempted, but are unavalible (such as being deleted) are stored as a false value instead of a null value or hash map. All successful videos with downloaded details will be a hash map.

#
#
PowerPoint Script/Draft
(the concept machine)
#

4 Overview

- **Need:** Recognition without examples is hard and common
- **First Obstacle:** Adversarial attacks and systematic weaknesses
- **Second Obstacle:** Meta learning and plasticity
- **Solution:** A recursive network of concepts

5 The Situation: The Tragedy of Alex

Dr. Bach Prahp is a biomedical researcher and his assistant, Alex Nat, joined his lab a few months ago.

Dr. Bach needs Alex to identify WhatchaMaCallIt. This shouldn't be challenging as Dr. Bach can recognize WhatchaMaCallIt without fail. It just so happens that Dr. Bach doesn't have any examples with WhatchaMaCallIt on hand. However, after 1,000,000 attempts, Alex has failed to show Dr. Bach any data related to WhatchaMaCallIt. If only there were some way Dr. Bach could explain to Alex what WhatchaMaCallIt was.

Alas, no such way existed. So Dr. Bach told Mr. WhatchaMaCallIt that the assistant was unable to find any of his records.

6 Obstacle 1: Backpropagation Didn't Slay the Hydra

Explaining to Alex that WhatchaMaCallIt was a person seems to be obvious step, but on the contrary, Alex is a machine. Teaching the Alex-machine what a person is likely means teaching it about faces, limbs, clothing, and each of those concepts would need their own recursive explanations. While it may seem that backpropagation has solved this recursive problem, the work done with adversarial attacks shows that the recursive Hydra was partially side stepped.
turtle vs shotgun
human definition of a gun *human definition of a turtle*
This kind of recognition failure in a human would need a near total collapse of the recursive definition system. Beyond that, such a collapse would be immediately apparent rather than hidden the way it is in many networks.

7 Obstacle 2: Punishment without Explanation

Let us suppose that techniques such as meta learning have sufficiently addressed the recursive definition problem of the last section. The second issue is that Dr. Bach has no means of communicating with Alex.
A human's definition of a shotgun depends on a recursive requirement of a handle, a barrel, a trigger, etc. If the machine's definition of a cat depended on only a few pixels
This is Where one definition is used two or more will take Solving this using back propagation
- Our current machines don't have a good understanding - Meta learning is the best shot, and it has problems
- My solution is If Dr. Bach told Alex that WhatchaMaCallIt was a person, it would It seems trivial that Dr. Bach should tell Alex that WhatchaMaCallIt was a person, but It may be easy to see Dr. Bach as foolish, but let us look at a more literal example.
image
This is a cat.
image
This is a shotgun. Or at least it is if you ask (insert adversarial attacks paper)

#
#
Research Draft #2
(video query tool VQT)
#
March 2020

What is the problem?

Suppose you have a large set of videos, for example months of video footage at a vacation house. Finding a specific event or object in a large set of footage is still an incredibly challenging task. While YouTube, Vimeo, and other services have attempted to help users find interesting videos within seconds, there appears to be a lack of research in the field of searching for actions and content within a data-set of videos where response time is not an issue.

What is the proposed approach?

Rather than creating an powerful agent using big data and significant compute power, we hope to mimic the approach of a company hiring process. Namely the fail-fast nature obviously non-viable candidates, and the process of learning with minimal examples by taking advantage of meta learning. There is assumed to be an initial be a large pool of candidate videos, to which we will apply a series of increasingly complex filters. The initial filter will spend very little time per-candidate, with each subsequent filter gathering more information and spending more time per-candidate.

What will implementation be?

First Filter

Inputs
Similar to a resume for a person, the first filter will utilize metadata for a video. The relevant components of the metadata are:
- title
- description
- thumbnail
- categories
- tags
- duration
- view count
- like count
- dislike count

The textual data (title, description, tags) will be converted into a one-hot encoding with each index representing the presence or absence of a word. The top 10,000 words will be used.

The thumbnail will be converted into a similar format by a model such as Darknet to list out common objects that are present in the image. That data will then be converted from a list of classifications into a one-hot encoded vector.

The numeric data will have a bit of feature engineering to simplify the learning process.

View Count The view count will be placed into logarithmic buckets to show the general magnitude of the video.

Like Count The like count will be replaced with a percentage of views that liked the videos. It will also be converted into buckets, but it will be parabolic to handle the case of very high and very low percentages.

Dislike Count This will be converted into a ratio of likes to dislikes. It will also be converted to buckets that are designed to handle variability. This variability however will more logarithmic in nature with more buckets near 100% and exponentially larger buckets towards 0 and positive infinity.

Metalearning

In addition to the inputs mentioned above, there will be a search query. This will start of as a one-hot encoded word describing what the user is looking for. The initial size of the vector will be 10 elements, however in the future this will be expanded. It is also likely that this will be changed from one-hot to using Google's word2vec to incorporate Zero-Shot learning.

Network

The network will be simple
- 32, fully connected, relu
- 16, fully connected, relu
- 1, sigmoid

Optimized using binary crossentropy. Dropout and batch normalization will be applied as needed to combat overfitting.

Testing and Data Sources

The training and testing will be done with 1000 videos taken from YouTube. There will be 100 videos that contain a basic emotion as defined by a VGG based model for detecting the 9 basic emotions. Of the remaining 900 videos, 450 will be videos taken from the "related" field of the 100 videos with the happy emotion, and the other 450 will be effectively random unrelated videos.

All of the videos will be scanned for the happy frames by taking a sample the frames rather than scanning all possible frames.

Optimizations (footnote)

The video metadata will be stored in a

Future Filters

Subsequent filters will begin by analyzing the content of the transcriptions of the videos, and possibly the content of the video itself using an LSTM.

Services related to this research:
https://smartella.com/index.php?pg=welcome

Reference sources:

- "Deep Learning with Python" by François Chollet
- "Tabular Data Analysis with Deep Neural Nets", https://towardsdatascience.com/tabular-data-analysis-with-deep-neural-nets-d39e10efb6e0

March 21 2020

What is the current direction/objective?
Essentially a smart search for video content. More specifically it is to create a system that creates a smaller and smaller subset of likely videos so that the user is more efficient at searching the likely possibilities.

What will be novel?
While the outcome would be new, there is no doubt YouTube has already be working heavily in field of video search. The novelty is in bootstrapping, bootstrapping the training set, testing set, and the learning process itself with meta-learning. The relatively poor performance of an initial model can be used as a starting point for future models to rule out obviously-inefficient video segments. This will involve some form of an actor refining concepts and attempting to predict when a concept is present in a video.

Why will this work?
Minimal changes in recognition will rapidly accelerate the ability of the next system to learn. The challenge will be the initial recognition of concepts.

What is the minimum example approach?
There is an existing system for finding videos that contain faces, which can be used as a starting point for training a system to recognize viable videos based on their metadata. The video metadata contains tags, a description, title, and voice-to-text transcription that can be used as a simple estimation of the video's content. The initial form of training can be a modified form of existing NLP models trained on predicting whether or not a video will contain faces with a particular emotion before the video is downloaded. The next level of the process will be similar, but will involve the visual recognition of some frames within the video.

Why is there promise in this area?
There are two reasons.
1. Humans observe less than a million hours in a lifetime. Nature's answer to recognition was not more data. Creating a smarter actor based learning system with minimal data will, at minimum, be an approach that has already been proven to work.
2. Failing will still be a success. Meta-learning as well as actor-based understanding of videos is a relatively unexplored in a field that will eventually need to explored. Knowing the approaches that do not work will be critical for future research.

Are there any major concerns with the system?
The lack of ability to reason and have memory will be limiting factors, but that applies to all existing machine learning.

Feb 18 2020

There are two concepts I have been considering for a few months that I believe can be applied with much success to gesture/facial recognition:
1. Dynamic class classification
2. Uncertainty measurements

Dynamic class classification
How can a classification algorithm have a dynamic number of classes?

Well humans can be dynamic classifiers, but you must ask a human a question first before they will classify anything. Why not apply the same to a deep learning model: create a model that allows you to ask it questions. This is nothing complex, simply add an additional input vector and label it as the "question" vector.

How will the "question" vector be connected to the rest of the "normal" network?

This will require some experimentation, however the model will likely benefit if the question vector is connected to every layer of the "normal" network. The question vector may also need a few layers to be interpreted before being applied uniformly to the rest of the network. These interpretation layers would likely benefit the system by allowing for XOR and more complex logic to be performed.

How is this going to help generalization?

By having a question input, we have effectively taken the concept of transfer learning and have made it a built-in feature. The connections between the "normal" layers are still required to do the recognition, however now they are now potentially required to recognize a vastly larger number of classes, and they need to change completely change what it is they are looking for based on the input. The classes need not be mutually exclusive. For example the machine can be asked if a human is in a picture, if a child is in a picture, if the color red is in a picture, or if the picture was taken from a moving camera. These are very different aspects, and by requiring the same network to be capable of recognizing all of them, the network itself must become generalized or it incur penalties of inaccuracy.

How will training work?

When training time occurs, the "question" will stay constant for a particular class. This line of research will also be interesting to see if the model can quickly learn that the "question" vector is constant.

Should the "question" vector be a one-hot encoding?

It is likely the model would benefit if the encoding was more advanced, such as using Google's word2vec encoding. Doing this will potentially allow the network to have Zero-Shot learning since the words with similar meaning will be relatively close to each other.

Uncertainty

How can a machine know when it doesn't know?

There's actually a number of techniques, and experimentation will need to be done to discover which will work best for this application. One of my personal favorites is to measure the amount of total neural activation in a network, and when there is significantly less activation than normal (maybe 2 standard deviations) then that likely indicates the network doesn't understand the input it is receiving. The book "Human-in-the-Loop Machine Learning" by Robert Munro covers an entire arsenal of uncertainty-measuring techniques along with their strengths, weaknesses, and quirks. I believe they can be be used to break into unexplored research territory in a number of ways.

How is this going to help with a smart crawler?

At this point in time, some data needs to be hand labelled.
1. If the model has absolutely no idea what is happening in an image, it will likely benefit immensely from having that labelled by a human. As some research has pointed out (at Texas Data Day) the quality and diversity of the data is at least as important as the amount of data.
2. If the model is orders of magnitudes more certain about a decision than a human, that is a red flag in-

dicating over-fitting. What is fantastic about measured uncertainty is that over-fitting can (theoretically) be adjusted for on an example-by-example basis instead of by changes to the entire model like dropout.

How will uncertainty + dynamic classes work together? By training a model on traditional challenges, such as the 9 basic emotions, the model can be both a jump-start into recognition while also being a long term network that nearly all future classifications can be mixed into. The uncertainty of the traditional model, such as the 9 basic emotions, can be used to identify data that would be useful for creating new classes. These new classes can be labelled and then the network can be trained to recognize the new class and repeat the cycle with more intelligence than before.

February 3 2020

Found and cataloged all the video metadata options, I suspect the data they (even only a few options) would generate would be way more than a single JSON file can handle so the extra options are disabled at the moment. Its already 12Mb of whitespace-less text, so a legit database is probably needed.

Reorganized the repo, its structure is much cleaner now and reflects the 3 existing stages. Many small bugs, like the lack of stdout output for stage 2, were fixed.

Moved all dependencies into Docker and created a docker inter-dependency framework for use on all future projects. This makes all the code server-scaleable, makes it OS independent, and allows us to freeze libraries and prevent them from breaking dependencies.

asdfasdf

January 29 2020

The smart crawler system has been broken up into 3 pieces
1. url collection
2. add meta data
3. add face data

stage 1 is multithreaded and put in a docker container, but stdout isn't displayed
stage 2 is multithreaded but not yet in docker, and must be restarted to incorporate URLs added since the time it was started
stage 3 is single threaded, not in docker, but can be run continually. It is certainly the bottle neck of the system. However, bottleneck of the entire process is still us humans because as incredibly inefficient as stage 3 currently is, it still easily outpaces the number of videos we can label ourselves.

The next stage is likely
4. add recommended clips
This stage would use some methods similar to hill climbing to guess at the distribution of nicely-visible faces. Estimate where faces would be, and pick the most-uncertain areas first to quickly fill in an accurate knowledge graph. The better the estimate of total face-time-per-second, the more willing the algorithm should be to spend time exploring that video.

A lot of time was spent fixing/updating libraries (major libraries) that were working 6 months ago. Putting all these processes inside of a docker container would prevent this kind of breakage in the future.

todo:
- conceptualize an algorithm for video-clip (sub-clip) selection
- create a legitimate database for managing video metadata

- put stage 2 in a docker container
- put stage 3 in a docker container

January 21 2020

Got the url_collector working inside docker Pulls URLs recursively (BFS) by exploring all possible "related" videos" Saves them to a JSON file, with the youtube video ID as the key The only non-working part is that the stdout isn't displayed until after the process is stopped. I'll have to fix that next

#
#
Research Draft
(smart active crawler)
#
January 2020

8 Whats the general problem?

I need a system that doesn't require a Genghis-Khan-sized army of people labeling data. I need generically good data and decent amount of it (don't we all) and I need to label the relevant parts of that data.
For example, I need data about sarcasm, or people attempting to do handstands.

9 So you're proposing an answer?

Well, I'm proposing a practical answer assuming:

- you can at least get your hands on garbage data
- the garbage has a non-zero probability containing useful data
- you don't care too much about compute power

10 So whats the basics of the answer?

Computers only work well because of bootstrapping, computers+software make computers+software better. Same with human learning, learning one thing accelerates the process of learning any number of other concepts. So why not turn the entire machine learning data collection, data labelling, model refinement process into a living self-bootstrapping system guided by humans.
Training one-off models on a known dataset may have its upsides, but I'm here to convince you why and how smart crawlers are the future.

11 Whats that kind of system look like?

I'm so glad you asked. Its a simple 2 cycle system.

Cycle 1 Data Collection

1. Create a never ending stream of garbage data points (videos, pictures, documents, etc) For example, a web scraper pulling in our president's twitter feed.

2. Do a little bit of scripting/programming to add summarizing information to that data (resolution, picture date, document length, etc)
3. Create a automatic/flexible filter on the data. For example a high probablilty of rejecting documents that are over 10,000,000 words long.
4. Take all the models you have (starting with 0 models) and use them as filters. You can apply the models recursively or in parallel (or a mix of both). Recurively means: if one model thinks the data point is good (and generates a more detailed summary) then that data point gets evalutated by another more advanced summary model Parallel means: if two models disagree, then that data point is probably useful for training if both models agree with high confidence, then the data is probably useless
The idea is that obviously-garbage data gets filtered out quickly and doesn't waste processing power.
5. If the data seems useful, send it to a "useful data" database

Cycle 2 Data Query/Selection
1. Use a model(s) (of your choice) to attempt to identify data relevant to your current task based on the "useful data" database.
2. For any data returned, human-label it as approved/disapproved of being useful.
3. If it is useful, human-label it as relevant/irrelevant for your specific task.
4. If it is relevant, human-label it with detailed labels.
5. Train the model you actually care about on the labelled data.
6. If the trained model is good at specific classification, add it to step 1 If the trained model is good at summarizing, add it to the list in Cycle 1 step 4
TODO: add uncertainty into the mix so that data most useful for a model will be selected

12 Why is that novel/useful?

The static databases of MINST, ImageNet, IMDB Reviews, and others have launched machine learning to the incredible height it is at today. Building continuous bootstrapping database streams could be the next generation of data needed for the next wave of breakthroughs.

January 2020

13 EZ Database

14 Whats this sub-problem?

I need a database that's scaleable in terms of number-of-items but is as easy to manipulate as a JSON file

15 Whats the current design plan?

```
- create a docker-database setup:
    - have an init with username+password
    - have a superuser API with:
        - ability to add/remove approved keys
        - ability to create/remove databases (collections)
        - ability to change passwords
    - have an API library for:
        - connecting to a database with an approved key
        - set the value for a key
        - get the value for a key
        - check if key exists
        - filter all keys/values for a specific pattern
```

January 1 2020

(log template)