Auditory Deep Q Networks

Austin Ray 1,2  Do-Hyoung Park 2  Vignesh Venkataraman 1

Abstract

Automatic speech recognition (ASR) has become a ubiquitous part of daily life in an increasingly voice-driven world, with practical applications in digital assistants (Siri, Google Assistant, Cortana), smart speakers (Alexa, Google Home), customer support hotlines, and other spaces of active research and product development. Reinforcement learning has also gained traction in recent years (as a function of renewed research interest and massive increases in processing capabilities) as a very popular method of training agents to perform increasingly difficult tasks, with the latest state-of-the-art deep reinforcement learning systems and agents capable of playing complex games like Pong and Go. This paper explores a combination of the two fields, specifically with regards to digit recognition in the context of a simple, solvable reinforcement learning environment, and our results show that ASR and RL techniques can be combined in an end-to-end model that achieves high scores on an RL task.

1. Introduction

Our project has two distinct but interleaved AI components: automatic speech recognition in the context of digit recognition, and reinforcement learning in the context of tasks provided by the OpenAI Gym framework, specifically the FrozenLake environment.

The primary goal of our project is to demonstrate the feasibility of combining ASR and RL techniques by training an agent capable of “solving” a FrozenLake environment (reaching a goal state from a start state) in which state representations are replaced by audio – that is, instead of an agent being given an integer representation of its state, it is told verbally where it is in the world. A related subgoal of this is to determine what the agent actually learned from an ASR standpoint (i.e., did it actually learn to classify digits as part of its learning?) by taking the underlying model and adapting it into a classifier, which would be very useful in a general sense, not just within the limited confines of the FrozenLake environment. Finally, our project sought to make these results generalizable from an ASR standpoint: As the state space of possible audio recordings is far larger than any sort of training set, we would like to ensure that our trained agent would be able to cope with previously unheard audio samples and still have good performance.

The idea behind this task from a reinforcement learning perspective is to determine whether it is possible for an agent to use human speech as the inputs for a RL system. This is clearly of interest for real-world applications in which vocal directions could be given by a user to teach an agent some policy. In the Atari DeepMind paper, it was determined that by using a convolutional neural network structure, it was possible for an agent to learn from an environment using the visual state of the environment as the input to a Deep-Q Network (DQN) (Mnih et al., 2013). With deciphering speech inputs being a difficult task on its own, we wondered: Can these two individually difficult tasks be handled together by one system? Additionally, would the performance of this single system be able to match the amalgamation of the state-of-the-art in each individual task?

2. Background/Related Work

2.1. Playing Atari with Deep Reinforcement Learning

In this famous paper, Mnih et al. “present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning.” Specifically, the authors of this paper used a model that is a “convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards.”

Our project was therefore an analog of this paper, with
our high-dimensional sensory input being the MFCC features of raw audio and our learning model being a Recurrent Neural Network (RNN) trained with a variant of Q-Learning and other RL algorithms.

One of the main victories in this paper was in overcoming the difficulties that usually come with training neural networks in Reinforcement Learning (RL) settings, such as non-i.i.d. samples, changing data distributions with policy, non-convergence of the model, and vanishing gradients due to delayed rewards. The paper explicitly defined an “experience replay mechanism” that was useful for combating the first and second of the four issues mentioned above, which implemented as part of this project.

A third victory was a reduction in training time due to having the neural network output all Q values for all state-action pairs in one forward pass, rather than computing the Q value of each individually.

In general, this paper was the main inspiration for our project. That said, we understand that our problem space is different, most importantly in the sense that its state space is small and discrete while the Atari paper’s environments had large, continuous state spaces. This manifested in the results that our neural networks did not require as much complexity.

2.2. Connectionist Temporal Classification (Graves et al., 2006)

This paper discusses a new and novel technique for training recurrent neural networks on noisy, unsegmented, and unprocessed sequence data, with specific applications pertaining to ASR. The key idea of Connectionist Temporal Classification (CTC) is to create a recurrent neural network whose outputs can be thought of as a probability distribution over the various label sequences. With some clever arrangement, this network can be used to model the likelihood of encountering a particular label (or not) at a particular time within the sequence, and with a standard maximum-likelihood-derived loss function, this network can be trained using standard gradient-descent-based backpropagation algorithms.

CTC-based models are becoming the go-to state-of-the-art solutions at major corporations for various ASR-related tasks. These models don’t require pretraining, don’t require major preprocessing of the data, operate in an end-to-end manner (without predefined alignments), and ditch the canonical HMM/GMM/hybrid models that have dominated ASR to this date. This paper presents results for CTC on the TIMIT dataset against baseline HMM and HMM-RNN models; these results show a great deal of promise, making models of this variety good choices for our ASR tasks. Moreover, CTC fits exactly what we are trying to accomplish, in that it fits labels to sequence data; in fact, our use case for CTC would be to directly fit digit labels to sequence data.

3. Approach

3.1. Dataset: TIDIGITS

In image recognition, one of the classic “beginner’s tasks” is to build a handwritten digit recognizer on the MNIST image dataset (Lecun et al., 1998). This dataset is free, public, and large; it is available to download for free from Yann LeCun’s website and has 60K training and 10K test examples. The closest match to these criteria for spoken digit recognition is the TIDIGITS dataset, which was collected by Texas Instruments in the 1980s and is available for free to Stanford students upon request. The dataset contains about 25,000 spoken-digit utterances (historically packaged on three different CD-ROMs) by 326 speakers of varying ages, genders, and accents (Leonard & Dod-dington, 1993). Other free, publicly available spoken digits datasets do exist (Jackson, 2017), and we encourage readers wanting reproduce our results to utilize these publicly available datasets if they cannot get access to TIDIGITS.

TIDIGITS is rich, robust, and well documented, but one place it falls short is in the number length-two digit sequences within the range of 00-15 (there are roughly 20-40 total utterances fitting this pattern, of which half are in the test set), which is less of a shortcoming of the dataset itself and more of an undesirable quality with regard to our specific task. Solving this shortcoming was critical in achieving our stated project goals.

In order to better manage usage of this dataset, we parsed the dataset’s hierarchical directory structure and wrote the path to each file, along with the metadata (speaker identifier, speaker type, digit sequence, train/test, etc.) into a sqlite3 database. This allowed us to perform fairly complex queries on the dataset to retrieve any information we needed without writing ad-hoc directory-traversing scripts.

3.2. OpenAI Gym (Greg Brockman, 2016)

OpenAI is a non-profit organization dedicated to the advancement of artificial intelligence in a safe manner. OpenAI Gym is the organization’s generic, open source, and extensible framework for evaluating the performance of reinforcement learning agents on a collection of curated environments and state spaces, with options ranging from classic control-loop problems like balancing a pole on a cart to more esoteric examples like the computer game Doom. Fundamentally speaking, reinforcement learning is the art of teaching an agent to observe (either fully or partially) some notion of state and take an action based on that state.
in order to achieve some rewards, either immediate or deferred. OpenAI Gym simply formalizes that learning and evaluating framework into a standardized API, wherein you query an environment for some state, respond with an action, and receive some reward (or penalty) at some point in the future. Critically, OpenAI’s existing environments can be tweaked in meaningful ways by simply subclassing them and implementing one’s own twists on the needed methods, which was a fundamental part of our project.

3.3. FrozenLake (OpenAI, 2016)

The 4x4 FrozenLake environment. The agent starts in the top left-hand corner and attempts to find an optimal “policy” of actions in each state that will take it to the goal state in the bottom right-hand corner without falling into a hole, marked as ‘H.’ Any episode that results in the agent falling into a hole yields a reward of 0, while a successful goal-reaching episode will yield a reward of +1.

The environment we chose as our sandbox is canonically known as FrozenLake. The premise of this environment is simple: per the source code comments, the agent’s task is to walk across a variable-sized (either 4x4 or 8x8) frozen lake grid (starting from the start state at the top left of the grid), to retrieve a frisbee that is situated at the goal state (on the bottom right of the grid). What the agent observes is the state (an index, from 0-15 or 0-63); what the agent doesn’t know is that certain grid locations are “holes” in the frozen lake, and landing on these holes will cause the agent to fall into the lake and suffer an untimely demise. This is subsequently given no reward, while achieving the goal (and retrieving the proverbial frisbee) will give the agent a reward of +1.

The final twist to this task is that taking an action may not have the desired result in the “stochastic” version of the environment (as the lake is “frozen,” the agent may slip and move in a different direction than what was intended). There is also a deterministic version of the environment, in which moving up/down/left/right is guaranteed to have that result. The OpenAI community has agreed that “solving” the stochastic version of this environment is achieving an average reward of 0.78 over 100 consecutive trials, while solving the deterministic version of the environment should logically yield an average reward of 1.0.

3.4. AudioFrozenLake and MfccFrozenLake

The extension we made to FrozenLake was to replace the state returned by the environment to the agent with a representation of the state in audio form; that is, a recording, culled from TIDIGITS, of someone speaking the state as a zero-padded, zero-indexed, two-digit number (e.g. “zero-zero” for the start state, “one-five” for the goal state). This task required wiring up an API for accessing the TIDIGITS data by desired state and desired usage, and writing a wrapper around the FrozenLake environment that respected the internals of the environment but obfuscated the state-level output such that it would return raw audio corresponding to the state instead of the actual state value – we called this environment AudioFrozenLake.

Additionally, as raw audio has traditionally almost never used as an input to any sort of machine learning task (Jaitly & Hinton, 2011) (in stark contrast to computer vision tasks, where normalized images are commonly fed directly into convolutional neural networks), we wrote an additional wrapper around AudioFrozenLake that computed the MFCC features for the relevant audio segment selected by the underlying AudioFrozenLake environment. We called this environment MfccFrozenLake. With these environments prepared, we defined our task as “solving” (per the OpenAI Gym guidelines) the deterministic and stochastic versions of MfccFrozenLake, which would mean achieving an average reward of 0.78 on the slippery MFCC-feature-enabled environment and an average reward of 1.0 on the deterministic environment.

4. Experiments

4.1. Benchmark: Monte Carlo Policy Evaluation with Various State Error Rates

Early on in our project, we evaluated the theoretical performance of an agent on Deterministic-4x4-FrozenLake-v0 and Stochastic-4x4-FrozenLake-v0 with varied observational error rates contributing noise in order to get an idea of how well we should expect our agents to perform in the theoretical limit. Specifically, we first derived an optimal policy for each environment via Value Iteration, then evaluated these policies using Monte Carlo Policy Evaluation (MCPE) with varying values of “State Error Rate” (SER), which we define here to be the probability the agent receives an incorrect observation of the current state as an integer. The results of these calculations for SER values from 0 % to 100 % can be seen in Fig. 2.
Auditory Deep Q Networks

4.2. Benchmark: Two-Step Model (Digit Classifier + Q-learning)

The simplest model one might use to solve the auditory FrozenLake environment would be a spoken digit classifier combined with the standard tabular Q-learning model, effectively doing the ASR and RL tasks in isolation. Specifically, we first train a spoken digits classification model, namely a Recurrent Neural Network (RNN) model in TensorFlow with a single GRU layer of 64 units using Connectionist Temporal Classification (CTC) loss (see Fig. 3 for CTC Loss and WER vs. training iteration for the various models we trained). We then train and test a standard tabular Q-learning model on MfccFrozenLake using the digit classifier as a translator between the audio observation given by the environment and the integer observation required by the Q-learning model to update its Q table and choose its next action.

A quick aside about the CTC digit classification model: The ASR system that we elected to deploy was based on the concept of Connectionalist Temporal Classification (CTC), which was discussed in further detail in the Background/Related Works section. This type of recurrent neural network architecture has shown great promise with regards to automatic speech recognition and is quickly becoming the state-of-the-art both in research and in consumer-facing production environments. Our CTC architectures relied on Gated Recurrent Unit (GRU) cells feeding into an affine output layer, with a special CTC loss function used to train the network to recognize sequences of digits and a beam-search decoder to select the most likely sequence of digits from network outputs (Graves et al., 2006).

This “Two-Step Model” performed quite well on MfccFrozenLake (both deterministic and stochastic), achieving results close to those of a normal Q-Learning Model trained and tested with fully-observable, integer observations. Importantly, the Two-Step Model would always learn the optimal policy for the environment when using a digit classifier with state-of-the-art WER (0.3%). Furthermore, our (limited) experiments with Two-Step Models on Deterministic-MfccFrozenLake show that, for digit classifiers with low WER $w$, a Two-Step Model using this digit classifier achieves an average reward approximately equal to that of the MCPE result produced using an SER of $w$ (see Table 1). That is, SER for Deterministic-FrozenLake is approximately equivalent to WER for Deterministic-MfccFrozenLake when each uses a tabular Q-Learning model (and when SER/WER is sufficiently low). This finding is interesting on its own and proves useful in discussing the results of our ADQN later in this paper.

<table>
<thead>
<tr>
<th>Two-Step Model</th>
<th>MCPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SER(WER) = 0.3%</td>
<td>1.000</td>
</tr>
<tr>
<td>SER(WER) = 8.9%</td>
<td>0.853</td>
</tr>
<tr>
<td>SER(WER) = 27.9%</td>
<td>0.700</td>
</tr>
</tbody>
</table>

Table 1. A comparison of the average rewards attained by our two-step model and the theoretical maximum long-run performance by an ideal agent acting on the deterministic 4x4 FrozenLake environment. The agent is seen to outperform the theoretical maximum in two of the three cases; this isn’t unexpected, as the MCPE trials were conducted under the assumption that each state was uniformly likely to be erroneously detected, which is not the case in the case of an actual ASR classifier.
Figure 3. The learning curves (CTC loss and word error rate, plotted against training iteration) for the digits classifiers that we trained specifically for the ASR task as part of the two-step model baseline, which combined an ASR trained in isolation with bare-bones tabular Q-learning in order to establish a baseline for our the auditory deep Q-learning model we eventually worked toward.

Note that the only important heuristic here is the performance of the digit classifier on the data subset used by MfccFrozenLake during training and testing of the Q-Learning agent, as we’re dealing with simulated environments. That said, state-of-the-art CTC-based spoken digit classifiers are able to achieve generalized WERs lower than 0.3% on both their training and their test sets (He & Deng, 2008), which corresponds to average rewards higher than 0.995 on Deterministic-4x4-MfccFrozenLake and higher than 0.802 on Stochastic-4x4-MfccFrozenLake in the theoretical limit. These quantities are important to keep in mind when analyzing the generalizability of ADQN models later in this paper.

4.3. The Auditory Q-Network Model

4.3.1. OVERVIEW

The key creation of our project is a neural network model architecture we call the “Auditory Deep Q-Network” or “ADQN”. The ADQN is based on the famous Deep Q-Network (DQN) from DeepMind’s famous paper “Playing Atari with Deep Reinforcement Learning.” The core goals of the ADQN and DQN are aligned: learn to perform well in an environment in which observations are offered in the form of high-dimensional media. For the DQN, this media is visual, in the form of two-dimensional images a human player might see when playing an Atari game. For the ADQN, this media is auditory, in the form of audio clips of humans speaking the digits representing the current location of the agent in the FrozenLake environment (these are the observations returned by MfccFrozenLake).

In both cases, the model is a complex function approximator that converts high-dimensional state observations to a vector of predicted future rewards (Q values) for the various actions it can take. During testing, the agent receives an observation of state, feeds this observation through its end-to-end model, receives Q values for its various actions, and takes the action with the highest Q value. Thus, the goal of our process is to train a model that is able to predict the Q value for each action at each state such that the Bellman update rule is satisfied, given a complex observation of that state. The overall Q-learning algorithm will, in turn, “teach” the model the highest possible Q value for the optimal action in each state.

Specifically, the DQN’s model architecture consists of several convolutional neural network (CNN) layers followed by a fully-connected (FC) layer with Rectified-Linear-Unit
**END-TO-END ADQN ARCHITECTURE**

Components: ASR / RL

**INPUT:**

\[ s(timesteps \times mfcc\_features=13) \]

**OUTPUT:**

\[ argmax \]

\[ Q(s, a) \text{ values} \]

\[ (1 \times num\_actions=4) \]

\[ \text{fully\_connected\_2} \]

\[ W^{(1)} \]

\[ \text{matmul} \]

\[ \text{ReLU} \]

\[ \text{fc\_out} \]

\[ (1 \times num\_hidden\_fc) \]

\[ \text{last\_state} (1 \times num\_hidden) \]

\[ \text{fully\_connected\_1} \]

**Figure 4.** The model architecture used for the end-to-end auditory deep Q-network (ADQN). The states are fed to the network in an array of size \( timesteps \times mfcc\_features \), which is then fed to an RNN layer of GRU cells. The final state from that RNN layer is then fed through two fully connected layers, with a ReLU activation as part of the first fully connected layer, to return a tuple of \( Q(s, a) \) values for all actions that can be taken from the agent’s current state.

(ReLU) activation followed by a linear projection onto a vector with a length equal to the number of possible actions. In essence, the DQN’s first CNN layers convert its complex image-based input into high-level features about the observation that its FC layers can use to accurately output \( Q \) values. The DQN specifically uses CNN layers to convert its image-based input into high-level features because this is what CNNs are known to be good at (Karpathy, 2016).

Drawing inspiration from the DQN, we decided to use RNN layers for the first layers of the ADQN model, as RNNs are known to be good at converting audio-based input into high-level features (Graves et al., 2013). Note that this is the only core difference between the ADQN and DQN model architectures, which makes sense, as the only difference in their respective tasks are the type of media they receive as state observations. Our full model architecture is presented in Fig. 4.

**4.3.2. TASK 1: PERFORMANCE IN SIMULATION**

At first, the ADQN was trained and evaluated under the assumption of a “fully contained” environment – that is, the audio samples the agent “hears” during validation/testing are drawn from the same set of samples it heard during training, which is akin to the simulated Atari environments learned by the DQN.

Since we wanted to first replicate the results of the Atari DQN paper with auditory input, our first experiment involved using the entire TIDIGITS dataset for training and using that same set for evaluation, without doing a training/validation/test split. Given that the TIDIGITS database contains myriad utterances from people of different ages, genders, and regional accents, we felt that even by doing so, our system would generalize well.

Though we discuss the “synthesized digits audio” approach for “Task 2: Generalization” below, we did not need to use this approach for “Task 1: Performance in Simulation,” since the main goal of the approach was to improve generalizability, which is not an issue in Task 1.

**4.3.3. TASK 2: GENERALIZATION**

The main difference between the Atari environments and our MfccFrozenLake environment is that we get to choose a subset of media to give our environment to serve to the agent, raising the question of generalizability. In other words, while the goal of the DQN is simply to learn to play a game that lives entirely within a simulation, the nature of the ADQN’s task lends itself to grander goals, such as performing well in the “real world” on unseen media and inputs, particularly since utterances by most people tend to be unique. This goal is an interesting deviation from the DQN paper and something for which we have exciting, quite positive results.

Before moving on, we want to explain this difference a little
further. Let’s look at a specific Atari game – Pong. During evaluation of the DQN, even if the ball happens to be in a position it hasn’t been in during training, the essence of the state is the same – the sizes, shapes, colors, and orientations of the paddles and ball remain the same, the background is the same color, the image is the same shape, etc. – meaning that generalizing to these “unseen” states really isn’t asking for that much more from the agent, and thus, if such “unseen” states do exist, the agent should be expected to handle such states gracefully. This is why the DQN paper treats training and evaluation error the same way – there is no difference. On the other hand, we could train an ADQN-based agent using a set of ten human speakers to perform some RL task well and then expect it to perform this same task well when spoken to by a different set of ten human speakers. One can see how this generalizability might be important in a variety of practical applications for a DQN-based RL agent, so we thought it was equivalently important to explore in our project.

Specifically, we chose to evaluate the generalizability of the ADQN by creating three MfccFrozenLake environments (i.e. train, val, and test environments) at the beginning of training, each embedded with a non-intersecting subset of digits audio samples from the TIDIGITS dataset. Importantly, we made sure that the test-MfccFrozenLake environment was embedded with audio samples from a completely distinct set of human speakers from the train-MfccFrozenLake and val-MfccFrozenLake environments. That is, during evaluation on the test-MfccFrozenLake environment, the ADQN will be encountering audio from speakers it has never heard from before, a core requirement for proving generalizability on spoken audio datasets.

Here, the limitations of TIDIGITS came to the fore, as each two-digit sequence representing state (the utterances “zero-zero” through “one-five”) contained only 20-40 examples across both train and test sets, and thus, only 8-16 training examples alongside 2-4 validation examples. Attempting to train on such little data produced fairly comical results (for instance, a training reward of 0.9 to go along with a validation reward of 0.1). We clearly needed more data, and an open question was how we would acquire it, as there were, to our knowledge, basically no other accessible and reputable spoken digit datasets.

Our solution was to leverage the enormous numbers of utterances for each individual digit (all 326 speakers in the dataset said each of the single-digit numbers, from zero through nine, once) and concatenate them together to create “synthesized” audio samples. That is, to represent the state “one-two”, we’d randomly sample a recording of someone in the train/val/test dataset for each of the two digits “one” and “two.” We would then concatenate the audio frames together and feed them through an MFCC extractor as a single entity. In taking this step, we went from potentially single-digit training examples for each state to over 60,000 options per state, a massive increase in scale and scope.

4.4. Transfer Learning

4.4.1. Overview

In the spirit of exploring the intersection between ASR and RL, we thought it would be interesting to explore transfer learning between our ADQN and our CTC digit classifier. Specifically, we wanted to see if the first GRU layer from the ADQN could be successfully used as the first GRU layer for the CTC digit classifier (and vice versa), either (1) used purely as initialization for the GRU layer weights but subject to change during training or (2) used as an initialization and then frozen during training, effectively making the destination model use the exact same first GRU layer as the source model throughout training and testing.

5. Results + Discussion

5.1. Auditory Q-Network Model

5.1.1. Performance in Simulation

For the discussion below, it’s important to keep a couple things in mind. The first is that, for our “performance in simulation” experiments, we used standard two-digit audio samples from TIDIGITS, not the two-digit audio samples synthesized from individual one-digit audio samples. The second is that we both trained and tested on all two-digit samples from TIDIGITS, since generalization is irrelevant here.

For a variety of model hyperparameter choices, the ADQN is able to achieve maximum performance on Deterministic-4x4-MfccFrozenLake, or an average reward of 1. Put another way, the model learns to map input MFCC features to the proper actions corresponding to the integer represented by those features, which means there is some measure of combined success in ASR and RL. This finding is pretty neat, as it means our agent was able to solve the audio version of the Atari DQN problem (albeit on a less complex environment).

On Stochastic-4x4-MfccFrozenLake, our best ADQN was able achieve an average reward of 0.64, meaning it reached the goal state around 64% of the time. Note that a Two-Step Model with 0.3% WER achieves an average score of 0.802 on Stochastic-4x4-MfccFrozenLake, and while we didn’t fully achieve this target (owing mainly to time and compute constraints), we are optimistic that we could reach it with further hyperparameter tuning and more extensive training.
The success of the ADQN model in simulation is interesting in that it replicates the success of the Atari DQN model in a slightly different setting. The ADQN-based agent managed to perform perfectly in the deterministic simulation and near-perfectly in the stochastic simulation. This finding is rather delightful and we encourage others to replicate and expand upon these results, possibly finding other environments on which to unleash the ADQN.

5.1.2. Generalization

First, it must be noted that no generalization occurred when the standard two-digits audio samples within TIDIGITS were used. We implemented models with Dropout and/or $L_2$-regularization to try to improve model generalization, but the model’s average reward on the validation and test sets remained near 0. Our conclusion was that there are simply not enough audio samples per state (on the order of 10-20 samples per state) to allow the model to generalize. This is the reason we created the synthesized two-digits audio samples discussed above, which ended up improving generalization quite drastically, as we’ll see below.

Most notable amongst our results was the performance of the ADQN on the deterministic MfccFrozenLake environment using synthesized digits audio and withholding validation and test datasets during training. Our best ADQN model for this task contained 2 GRU layers with 128 hidden units followed by a FC-ReLU layer with 16 hidden units and a final affine layer. We trained this model for 25,000 iterations (steps in the environment) using Adam optimization and the same loss function as the Atari DQN paper:

$$L(\theta) = \mathbb{E}_{s,a,r,s'} \left[ r + \gamma \max_{a'} Q_{\theta}(s', a') - Q_{\theta}(s, a) \right]^2$$

After training, we evaluated the ADQN on the training, validation, and test sets, on which it achieved average rewards of 1.000, 0.987, and 0.997, respectively. You can see a demo video of the ADQN being evaluated on the test set at this link: http://bit.ly/2re6QbG (make sure your sound is on). We also evaluated this model on the original, non-synthesized two-digits audio samples in TIDIGITS (which it never encountered during training), on which the model achieved an average score of 0.878.

Here, we see that the ADQN was able to achieve near-perfect generalization performance on the deterministic environment when trained with the relatively large synthesized digits audio dataset. Additionally, the ADQN was able to achieve high performance on non-synthesized digits audio, which qualifies as another type of generalization. As we’ve mentioned before, we can’t compare these results to those of the Atari DQN paper because the Atari paper deals only with simulated environments. That is, images are generated for Atari games the same way during training and testing, so there is no concept of “generalizability” in that world. As such, our results are significant in that they extend the Atari paper to show that the DQN doesn’t only perform well on simulated, self-contained environments with high-dimensional inputs, but also can perform well when given inputs it hasn’t seen before. This is exciting, as it shows that the ADQN had to learn something about how to distinguish digits by sound at a core level, otherwise it wouldn’t have been able to “understand” the speakers in the test set it had never heard from before.

We have only shown generalization for auditory inputs, but we don’t think it’s a stretch to say this generalizability would apply to visual inputs as well. For example, if the environment fed a certain image class to the DQN for each state in FrozenLake (e.g. “dog for state 0, “airplane” for state 6) instead of audio of the digits, we imagine the DQN would be able to generalize to a separate set of held-out images of each class and perform well on an environment that only used those images to represent state. We think this would be an interesting future experiment to show the DQN has good generalized performance for multiple types of high-dimensional inputs, not just audio.

Acknowledging the generalization success of the ADQN on the Deterministic environment, we must concede that almost no generalization was achieved by the ADQN on the stochastic environment. We tried many different model architectures, but none were able to generalize. The highest test set performance we achieved on Stochastic-4x4-MfccFrozenLake was 0.079, corresponding to a model with a train set performance of 0.133. This result might appear strange, considering the ADQN achieved a simulation performance of 0.64 in the previous section. We attribute this drastic drop in perfor-
mance to the fact that the ADQN in simulation was dealing with the standard digits audio dataset, which had so few samples per state that the ADQN could essentially “memorize” each sample for each state in order to perform well on the environment, while this would be quite difficult for the ADQN dealing with synthesized digits audio.

Lastly, why was the ADQN able to learn how to understand digits so well as to navigate to the goal state in the Deterministic environment but not the Stochastic one? Though we have no proof, we believe that introducing stochasticity between the action taken and the state transition that occurs ends up “confusing” the ADQN, making it significantly more difficult to train, especially when memorizing audio samples for each state is not an option, as it was in the simulation task. Relevantly, the Q-learning agent in the Two-Step Model took more iterations to converge to an optimal policy when acting in the Stochastic environment than the Deterministic environment. We imagine this difference is amplified for the ADQN, which starts of not knowing anything about speech, introducing even more stochasticity into the problem and making it even more difficult to train.

We leave the problem of generalization performance on Stochastic-4x4-FrozenLake with synthesized digits audio as an open problem and encourage the ambitious reader to try their hand at it.

5.2. Transfer Learning

When we took the GRU layer of our best ADQN and used it to initialize the GRU layer of a digit classifier using CTC loss before training, we found that training did not differ significantly from normal. We did not explore freezing the layer.

When we took the GRU layer of our best digit classifier (0.3% WER on train set and 72.9% WER on test set, both sets being non-synthesized digits audio) and used it to initialize the first and only layer (with no layer freeze) of the ADQN before training on Deterministic-4x4-MfccFrozenLake using synthesized digits, we saw that training also did not differ significantly from normal. The most interesting results were when we decided to freeze the transferred GRU layer. We tried this freezing tactic with an ADQN with a single GRU layer of 64 units, which was unable to learn anything. We then tried this freezing tactic with an ADQN with two GRU layers of 64 units, initializing the first of these layers with the pretrained digit classifier GRU weights and then freezing only that layer during training. Surprisingly, this model was able to achieve scores almost as high as our best ADQN model, with 0.994 train, 0.978 val, and 0.957 test.

The result of our successful transfer-ADQN model is interesting, in that it shows the GRU layer from the digit classifier provided sufficiently useful features for the input MFCCs for the rest of the ADQN layers to be able to accurately output Q values. This adds to our previous conclusion that the vanilla ADQN is indeed learning something about how digits sound at a core level.

6. Conclusions + Future Work

In summary, this paper presents a logical extension of Deep Q-Networks, a modern reinforcement learning staple, towards the field of automatic speech recognition: the ADQN. Particularly, using a recurrent neural network as part of the function approximator in a Deep Q-Network yielded very promising results on the combined TIDIGITS + FrozenLake environment. The resulting agent was able to fully solve the deterministic FrozenLake environment and showed early promise in the stochastic environment. Future work in this field would focus on achieving higher scores on the stochastic environment (especially with regards to generalization), attempting to solve more difficult environments (the 8x8 FrozenLake environment comes readily to mind), and using the trained agents for more complex peripheral tasks. Overall, the results of this paper show that the work of the DeepMind group on using high-dimensional media for inputs to a Q-learning system can be extended to other inputs, given the proper guidance during training, and can be used for ASR tasks effectively.

One possible model extension is that our state is only partially observable to our agent – that is, the agent will rely on an observation instead of accurate state information to judge the actions that it will take. In the spirit of recent work done by Matthew Hausknecht on his so-called “Deep Recurrent Q System,” we also made an effort to try and introduce historical context by replacing the first fully connected layer following the GRU RNN output with a LSTM layer in order to try and see if we could encode some recurrence and time context in order to further mitigate the state uncertainty of our agent, given the partial observability of our system based on the inability for even the best ASR systems to be “sure” of a state. Hausknecht showed that his DRQN performs comparably to the DeepMind Atari DQN on standard, fully observable environments, and performs much more effectively on a POMDP environment (e.g. “Flickering Pong,” in which the screen is sometimes blanked out) (Hausknecht & Stone, 2015).

However, this did not yield good results for us when used on MfccFrozenLake (i.e. the agent did not learn at all), and given time limitations, we were unable to further experiment with this to any significant extent. With more time to experiment with various model architectures, this might be a promising area worth further work due to the uncertainty inherent in our world.


OpenAI. Frozenlake-v0, 2016. URL https://gym.openai.com/envs/FrozenLake-v0.
Task
The FrozenLake environment is an OpenAI Gym construct that can be used to train an agent to "walk" across a slippery, hole-filled lake (represented as a 4x4 or 8x8 grid) to retrieve a frisbee from the goal state. While state is communicated noise-free to the agent by default, this project extends the FrozenLake environment by replacing integer-valued state with audio representations of state (e.g. state 12 is represented with an audio clip of a speaker saying "one-two"). We seek to "solve" this extended FrozenLake environment by combining automated speech recognition techniques with reinforcement learning and deep learning approaches to get the agent to the goal.

Dataset
TIDIGITS
● Collected by Texas Instruments in the 1980s
● 326 speakers, over 25K utterances
● Two-digit sequences augmented by a "synthesized" dataset
  ○ E.g. to speak state "one-two", sample "one" and "two" from the single-digit corpus and concatenate
  ○ Over 65K synthesized utterances per relevant two-digit sequence

Previous Work
Playing Atari with Deep Reinforcement Learning [1]
The authors of this famous paper developed a deep Q-learning agent to use on various Atari games, by taking in the visual representation of the game at each timestep as the state information, passing it through CNN layers to extract information, and using fully connected layers to extract Q(s, a) values for each state. This was the primary model inspiration for our project.

Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks [2]
This paper presents a novel approach for training recurrent neural networks on noisy, unsegmented, and unprocessed sequence data. The RNN architectures expounded upon in this paper were used in our project as part of the function approximators in Deep Q-Learning, as well as in direct ASR with pretrained weights from RL tasks.

Deep Recurrent Q-Learning for Partially Observable MDPs [3]
This paper dealt with issues of partial observability in the system (that is, the inability to know the state of the world with certainty, instead relying on observations). We have an analogous issue — we have a belief of our system based on the sound clips. The authors mitigate this issue by replacing one of the DQN fully connected layers with an LSTM layer that helped the system remember temporal context for each state, reducing uncertainty. We ultimately tried this within our architecture but it didn't yield results — perhaps with more experimentation, this would be a good extension.

Model
Baseline
ASR: Connectionist Temporal Classification (CTC)
● Under a canonical CTC model, our recognizer was able to achieve average train and test edit distance scores of under 0.05 on the full TIDIGITS dataset.
RL: Vanilla Q-Learning
● Stochastic FrozenLake is considered solved with an average reward of 0.78 over 100 consecutive trials. Deterministic FrozenLake is considered solved when average reward is 1.0. Both of these are achievable on standard FrozenLake without deep Q-function approximation (i.e. simple Q(s,a) lookup table).

Combined Baseline
● We plugged a pretrained digit recognizer (error: sub-0.5%) into our augmented FrozenLake environment, where it struggled, especially with stochasticity. Due to extended iterations, even a single mistake was enough to throw the agent into a hole, yielding poor rewards (full results pending). Not robust.

Auditory Q-Network (AQN)
END-TO-END AQN
ARCHITECTURE
Components: ASR / RL

Input:
● timesteps x mfcc_features

Output:
● optimal_action

Other architectural tweaks/experimentation
● LSTM layer — DRQN for POMDP [3]
● Batch Norm/Dropout
● CNN
● Replay Buffer

Clip Preprocessing
Each audio clip was fed through a mel-frequency cepstral coefficient (MFCC) extractor, which produced a matrix of dimensionality timesteps x mfcc_features. MFCC features are extremely popular in ASR. We set mfcc_features to be 13.

Training Details
Condition 1: Fully Observed Environment
● All audio clips are available during training
● No unobservable possibilities
  ○ Analogous to Atari games
Condition 2: Subset Environment
● Training environment is some subset of the state space
● There are audio clips describing state that might not be present in training but may be encountered in testing
  ○ More accurate description of the real world

Target network, replay buffer from Atari paper used to speed training [1]
Loss function:
\[ L(\theta) = E_{s,a,r,s'}[\gamma^{\text{max}(Q(s',a') - Q(s,a))}] \]
\[ \theta = \theta + \alpha(Q(s,a) + \gamma \text{max}(Q(s',a')) - Q(s,a)) \nabla Q(s,a) \]

Results
Evaluation Metrics
Average reward (RL): +1 per episode if goal reached, 0 if hole reached
Edit distance (ASR): Levenshtein distance between predicted, actual sequences

Fully Contained Environment
Very successful at solving both the Deterministic and Stochastic 4x4 FrozenLake environments with a wide range of model architectures and hyperparameters. Average reward of 1.0 on deterministic, 0.7 on stochastic. Further hyperparameter tuning could easily boost average reward in stochastic environment.

Subset Environment
Also successful at solving the Deterministic 4x4 FrozenLake environment, but only after extended training and experimentation with model architectures and hyperparameters. Average reward of > 0.95 on train, validation, and test sets with single- and double-layer architectures. Stochastic 4x4 FrozenLake in progress.

Conclusions
● Fully contained environment "reproduces" the results of the Atari paper. Given sufficient training time and training enhancements like the replay buffer, agent can learn enough about TIDIGITS audio to play MfccFrozenLake as well as an agent can play standard FrozenLake.
● Subset environment proves generalizability — even without hearing all possible clips, agent is able to generalize to new speakers and new utterance structures and still solve the environment.

ASR works, but needs more tuning and more investigation of implications.

Future Work
● Try tougher scenarios (8x8 FrozenLake, other OpenAI Gym environments)
● What other parts of the RL process can be replaced with speech inputs/
  ○ Example: Speech feedback as "reward input" (e.g. "good job"/"bad job")

References

An End-to-End Model for RL Using Speech Inputs
Austin Ray (aray7@), Do-Hyoun Park (dpark027@), Vignesh Venkataraman (viggy@)
CS 224S / CS 234 • Spring 2017