Homework 3

Luke Weber, S.I.D. 11398889
CptS 580 — Structured Prediction
Prof. Jana Doppa, Ph.D.

April 17, 2017

1 Recurrent Classifier Error Propagation

For a learned recurrent classifier, at each time step $1 \leq t \leq T$, for T total steps, we have a probability of mistake ϵ. We are to quickly show why the number of mistakes grows quadratically $O(T^2)$, rather than linearly $O(T)$.

An assumption to be made is that when an incorrect (i.e. non-optimal) input $(x, \langle \hat{y}_1, \hat{y}_2, ..., \hat{y}_{t-1} \rangle)$ is fed into the classifier to make a prediction $\hat{y}_t$ at time step t, the probability of error is higher than if a correct input $(x, \langle y_1, y_2, ..., y_{t-1} \rangle)$ was used, as seems to be the case in practice. We'll say that the probability of prediction error (i.e. $\hat{y}_t \neq y_t$) increases by a constant c for each previous mislabelling. Another assumption we make is that if the total probability of mistake grows quadratically, then the total *number* of mistakes grows quadratically.

Let us define the number of errors in T in the following way:

$$N_E(T) = \lfloor T \times P_E(T) \rfloor.$$

This is equation is based on the idea that multiplying a quantity (T) by a probability on that quantity ($P_E(T)$) will result in the sub-quantity affected by that probability. We define that probability of error $P_E(T)$ in all T steps as

$$P_E(T) = c \times \epsilon \times P_E(T-1)$$
$$P_E(T=1) = \epsilon$$

for some $c \geq 1$ and some constant $0 \leq \epsilon \leq 1$. As you can see, $P_E(T)$ will recurse for T times, making it $O(T)$ as ϵ and c are constants. Substituting, this gives us

$$N_E(T) = \lfloor T \times O(T) \rfloor \simeq T \times O(T) = O(T^2),$$

which shows that the number of errors/mistakes in T steps under a recurrent classifier is quadratic in T.

2 Unified Framework for an Imitation Learning Algorithm

From this framework (see Algorithm 1), there is a lot of freedom for implementing any of the following algorithms: Exact Imitation, Forward Training, SEARN, DAgger, and AggreVaTe.

For instance, if we wanted to implement SEARN with this framework, we would set the policy choosing function γ to return the oracle policy for $i = 0$, and gradually introduce learned multi-class policies from the aggregate learned classification set L as $i > 0$. On top of that, we could set the trajectory generating function μ to perform the Monte Carlo cost estimates, and generate mulit-class examples associated with these losses via μ followed by adding these examples to L with ξ.

Concerning Forward Training, an instantiation of the framework would have one main difference as compared to the standard Exact Imitation algorithm. In order to train each successive classifier over each iteration i over L, conditioned on the predictions from the previous classifier, we would need to have all the corresponding steps take place within the trajectory generating function μ, since we produce a new classifier on each labelling decision step.

As SEARN and Forward Training are the most comlex to implement with this framework, we leave Exact Imitation, DAgger, and AggreVaTe up to the reader to instantiate, as they are not too difficult.

Algorithm 1: General Imitation Learning Framework

```
1  Training set $D$, number of iterations $I$, policy choosing function $\gamma$, trajectory generating function
   $\mu$, example updating function $\xi$, and policy returning function $\rho$;
   // Initial variable setup
2  $\pi^*$ := oracle policy function;
3  $H_\pi$ := policy history list (empty);
4  $L$ := classification example list (empty);
5  $i$ := iteration number (1);
   // Run through $I$ training iterations
6  for $i \leq I$ do
       // Pick or create policy for iteration $i$
7      $\hat{\pi}_i := \gamma(L, H_\pi, i)$;
8      for $x$, $y$ in $D$ do
           // Generate decision/labelling trajectories
9          $T = \mu(\hat{\pi}_i, H_\pi, x, y)$;
           // Update classification set
10         $L = \xi(L, T)$;
11     end
       // Add policy to history
12     $H_\pi := H_\pi + \hat{\pi}_i$;
13     $i := i + 1$;
14 end
   // Return or create policy according to either $H_\pi$ or $L$
15 return $\rho(H_\pi, L)$
```

3 Holes in Training Data Dilemma

We assume that missing labels in the training data means precisely, for some input $(x, \langle y_1, y_2, ..., y_T \rangle) \in D$, one or more y_t is missing, for $1 \leq t \leq T$ where T is the (ideal) number of labels for the corresponding input. It is also assumed that we are given the exact locations of each missing label. Let's call the set of missing labels for any given input to be M.

Adapting the DAgger algorithm to this data dilemma isn't as simple as dropping out those labels and not generating that example $(f_{t-1}, y_t) \in L$ — a training example to add to the set of classification examples at iteration t where y_t would be a missing label. Here f_{t-1} represents the features generated from both x and each label in $\langle y_1, ..., y_{t-1} \rangle$. Initially, it was thought that we could discover the missing labels via similarity between examples in the same training data, because this would work with the OCR and Nettalk data given for the programming assignment. But, this might not be the best approach for every training data set, so we consider three more ideas to mitigate this dilemma:

(a) Partition the labels into $|M|-1$ sections and train on each of those sections independently, omitting the missing labels which seperate these sections.

(b) Replace each missing label with a unique dummy label.

(c) Throw out each $(x, \langle y_1, y_2, ..., y_T \rangle)$ where there are at least one missing labels (i.e. $|M| > 0$).

Now, (c) is a bit of a cop-out so we can disregard it. We'll likely lose a lot of useful data. However, concerning (a) we may be able to improve the understanding for the algorithm of specific features of text — despite them being disjoint — and if we only use a context history of 2-3 examples, this is likely the most data recovery we can get with missing labels present. Lastly, (b) seems like a less promising idea because examples in L using the dummy labels as context will have features for a label which will never be seen in the training data, making a mismatch between the training and testing distributions. Furthermore, those examples for which a dummy label would be predicted are essentially training for failure (i.e. labelling mistake) in the testing set.

These ideas seem to apply to both the DAgger and AggreVaTe algorithms, but I could be mistaken.

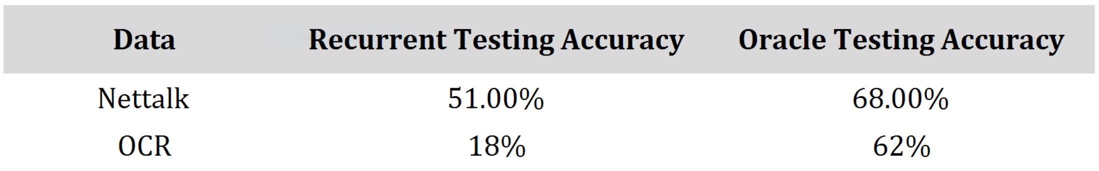
4 Implementation of Recurrent Classifiers: Imitation and DAgger

Please view the GitHub repository `https://github.com/lukedottec/StructuredPrediction` for the complete Python code. Note that this repository has been built-up over all previous assignments, so the files of interest are `recurrent.py`, `model.py`, and `main.py` in the `src/` directory.

See Figure 1 for a table for the oracle and recurrent error from the Imitation classifier on the testing data.

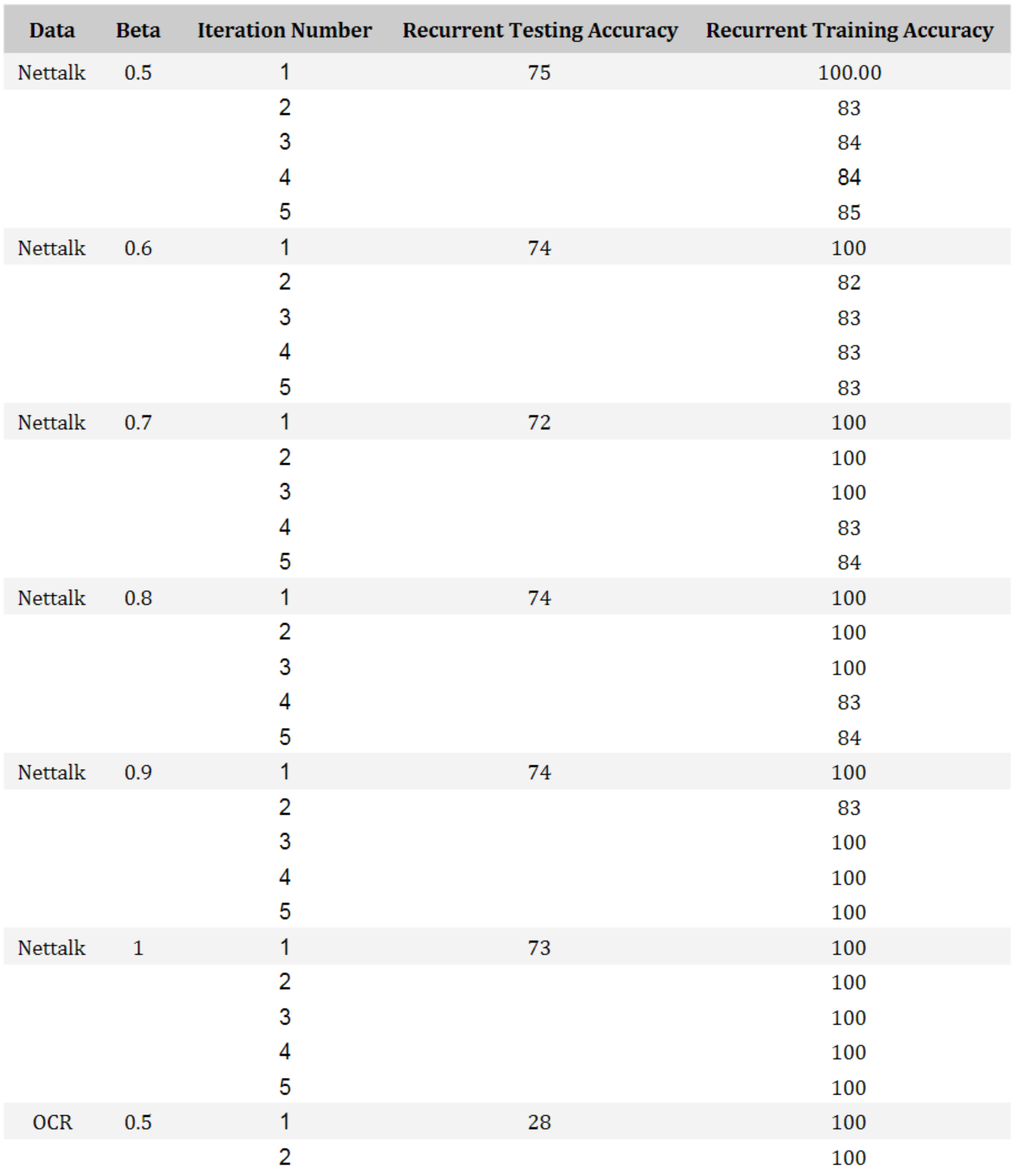
See Figure 2 and 3 for a table for the oracle error of the DAgger classifier on the training and testing data.

Finally, see Figure 3 and 4 for the one and only graph displaying the recurrent error throughout DAgger iterations for each data set.



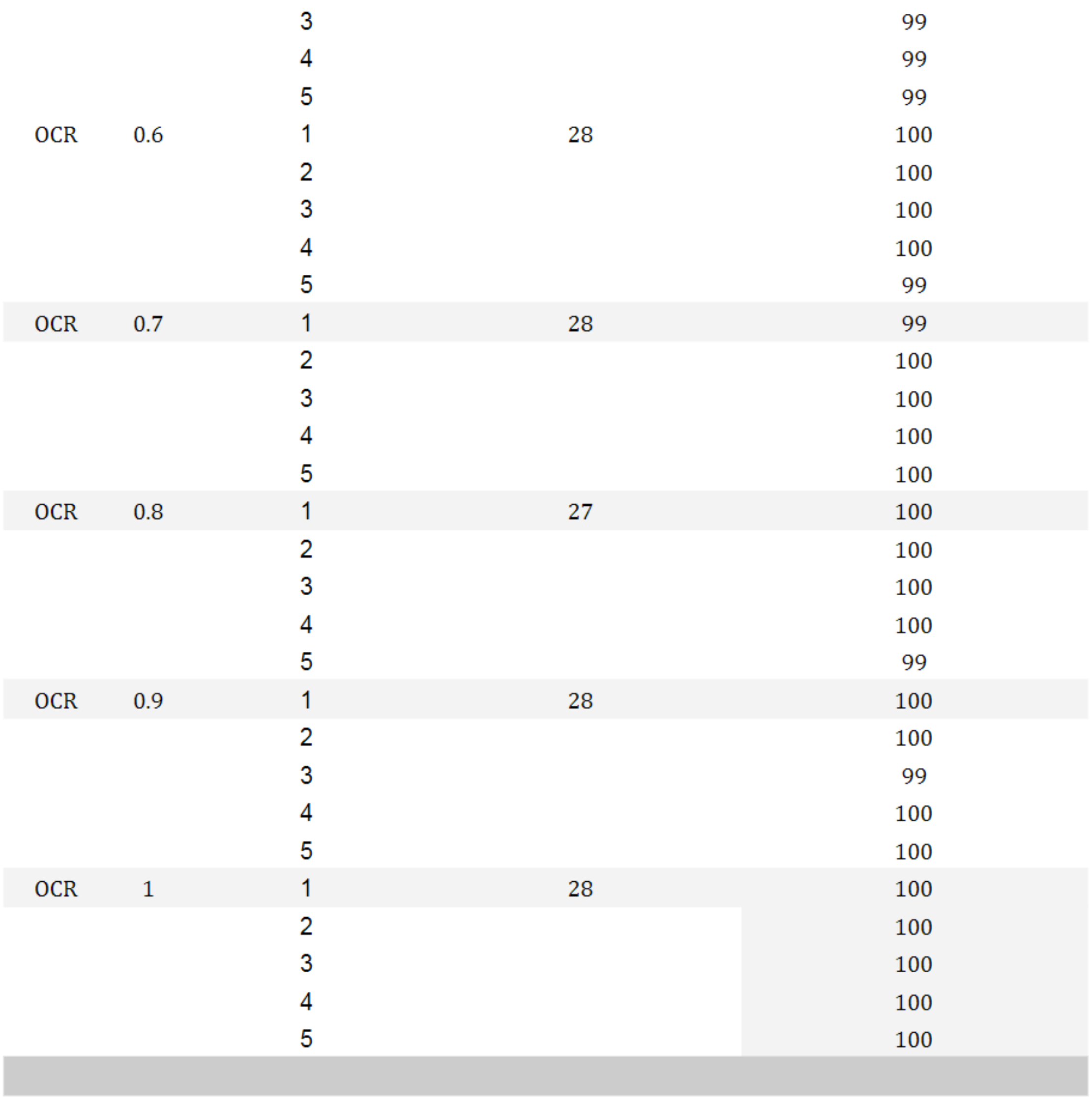
Data	Recurrent Testing Accuracy	Oracle Testing Accuracy
Nettalk	51.00%	68.00%
OCR	18%	62%

Figure 1: Recurrent and oracle testing accuracy of Exact Imitation learning algorithm over handwritten (OCR) and text-to-speech (Nettalk) data.



Data	Beta	Iteration Number	Recurrent Testing Accuracy	Recurrent Training Accuracy
Nettalk	0.5	1	75	100.00
		2		83
		3		84
		4		84
		5		85
Nettalk	0.6	1	74	100
		2		82
		3		83
		4		83
		5		83
Nettalk	0.7	1	72	100
		2		100
		3		100
		4		83
		5		84
Nettalk	0.8	1	74	100
		2		100
		3		100
		4		83
		5		84
Nettalk	0.9	1	74	100
		2		83
		3		100
		4		100
		5		100
Nettalk	1	1	73	100
		2		100
		3		100
		4		100
		5		100
OCR	0.5	1	28	100
		2		100

Figure 2: DAgger training and testing recurrent accuracy over different initial β values over 5 DAgger iterations each.



		3		99
		4		99
		5		99
OCR	0.6	1	28	100
		2		100
		3		100
		4		100
		5		99
OCR	0.7	1	28	99
		2		100
		3		100
		4		100
		5		100
OCR	0.8	1	27	100
		2		100
		3		100
		4		100
		5		99
OCR	0.9	1	28	100
		2		100
		3		99
		4		100
		5		100
OCR	1	1	28	100
		2		100
		3		100
		4		100
		5		100

Figure 3: Continuation of Figure 2.

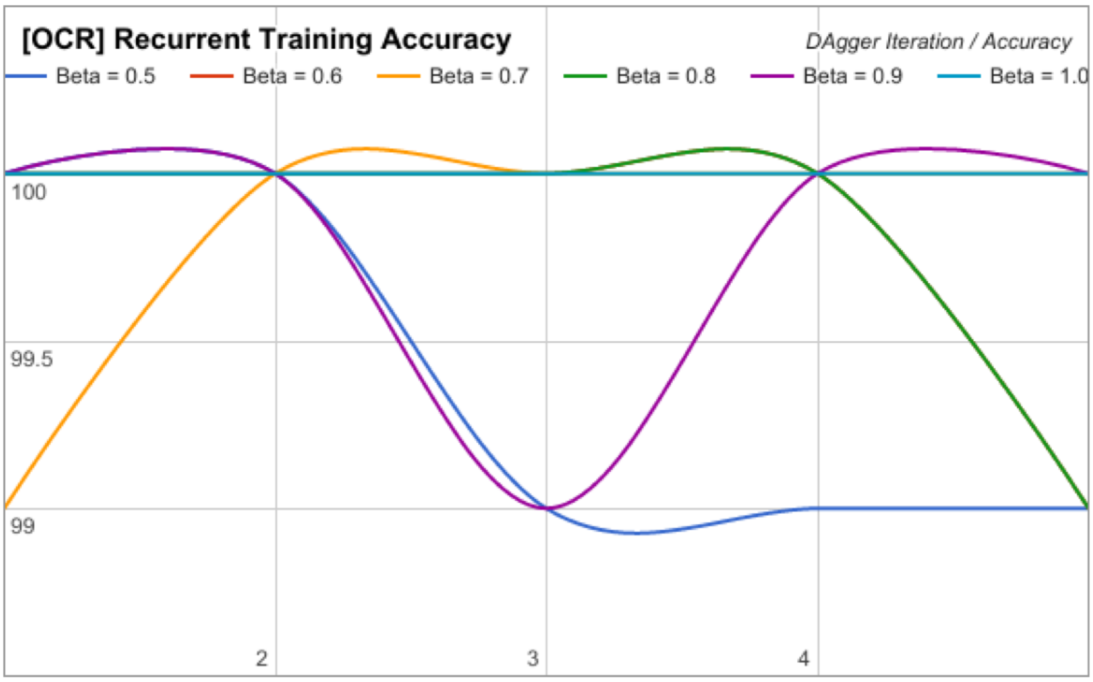


Figure 4: Representation of the recurrent training accuracy, as displayed in Figure 2 and 3, over 5 iterations with varying starting values of β. Over the OCR data set.

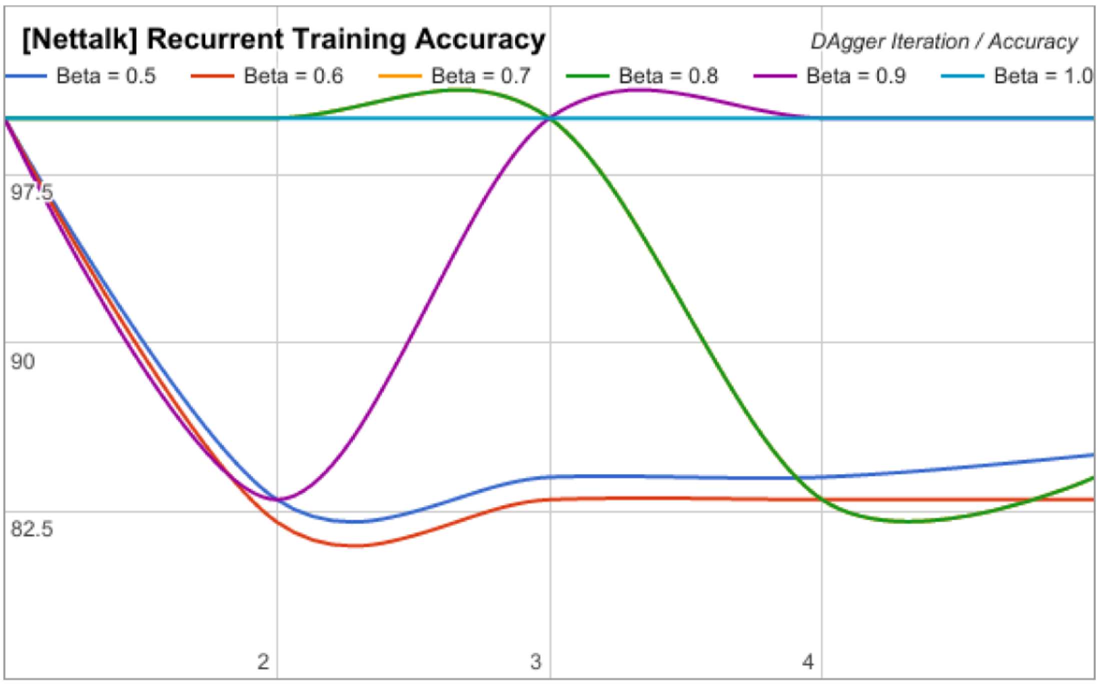


Figure 5: Similar to Figure 4, but over the Nettalk data set.