Exploring Massive Learning via a Prediction System

Omid Madani
Yahoo! Research,
3333 Empire Ave
Burbank, CA 91504
madani@yahoo-inc.com

Abstract

We describe the functionality of a large scale system that, given a stream of characters from a rich source, such as the pages on the web, engages in repeated prediction and learning. Its activity includes adding, removing, and updating connection weights and category nodes. Over time, the system learns to predict better and acquires new useful categories. In this work, categories are strings of characters. The system scales well and the learning is *massive*: in the course of 100s of millions of learning episodes, a few hours on a single machine, hundreds of thousands of categories and millions of prediction connections among them are learned.

Introduction

Categories (recurring patterns) are fundamental to intelligence, as they are the means of abstraction and reduction. However, sophisticated intelligence necessitates many categories, in the millions and beyond, to allow for making the required distinctions. A fundamental question is then how can an entity acquire so many complex inter-related categories? In recent work (Mad07), we explore a framework based on unsupervised learning. Unsupervised here means that the learning does not involve direct human involvement, whether in defining the classes to learn or in providing feedback (training data). In the abstraction provided (Mad07), a system is fed an unlimited stream of bits from a rich source, such as the available online text or audio and video streams, enjoying many regularities. The system engages in playing the game of prediction, repeatedly hiding parts of the stream and guessing the hidden parts. In the course of playing the game, different processes working in concert should lead to learning of myriad categories and their inter-relations. That work focuses on a *specification* level, exploring desiderata, challenges and requirements, for instance in terms of the major learning functionalities that should be useful to support and the many challenges posed by efficiency and long term learning.

Here, we present an implementation of an important subset of the desired functionalities. The system is fed a stream of characters from such sources as news articles. It begins predicting at the individual character level. Over time, it builds larger strings (categories) to be predicted and to help predict. We describe and discuss various aspects, including:

- The game, system organization and the nature of algorithms
- What is learned and the scale of it
- A preliminary study of category generation methods
- Challenges: memory and time efficiency, nonstationarity and noise, robust learning over long periods, ...

The current system performs two of the three major tasks outlined in (Mad07): prediction and composition (or concatenation), corresponding to capturing Part-Of relations, leaving grouping (abstraction or Is-A relations) to future work. The goal of this paper is to overview the work, describing the kind of considerations behind the system and algorithms, and making more concrete some of the challenges. There are numerous problems and subproblems for which our current treatment can be improved, and some functionalities can be enhanced significantly (*e.g.*, our composition currently performs simple concatenation). The exact manner that the task is defined here, the way the game is played, will likely change too. Our first goal is to provide evidence that massive scale learning systems, performing multiple learning activities is a very real possibility, and that this particular avenue is promising.

Our experience with the system is encouraging: hundreds of thousands of recurring categories (strings) are learned during millions of learning episodes. These categories connect to one another via tens of millions of weighted prediction edges. Learning of this scale takes a few hours only, but the task is memory intensive. The system does not show signs of slowing down over time. Interestingly, while the task is unsupervised, at the heart of the predictions are supervised, *i.e.*, feedback driven, learning algorithms at work. Here, the world serves as a teacher, *i.e.*, the provider of feedback, and the system takes an active part in building its own concepts. In terms of traditional supervised learning, the task can be viewed as one in which the sets of classes and features grow over time.

The paper is organized as follows. In the next section we describe a simplified game, and the system and its algorithms. The following section presents our experimental results. We then discuss related work, and conclude.

Basic Mode of Operation:
Repeat
1. Move window one category to the right
2. Update context vector and the target (category)
3. Predict using the context vector,
and update the system based on the outcome

Figure 1: The basic cycle of the prediction system. Update here consists of adjusting connection weights used in prediction and updating composition related statistics and connections. Adjusting connections includes strengthening and weakening weights and adding and dropping connections. Updating composition statistics includes adding new categories.

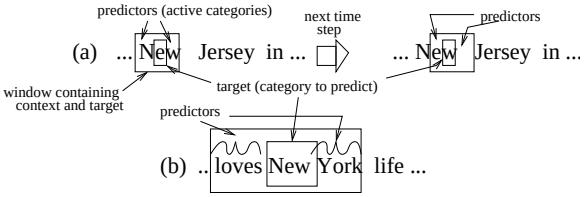


Figure 2: Reading from left to right and hiding and predicting categories. (a) The games begin at a low level, here single characters. First category *e* is the target (category to predict). The context (in this picture) is composed of two categories, one on each side: we say *N* is active at position -1 and *w* is active at position +1. In the next episode, *w* becomes the target. (b) Gradually larger categories are learned and used as targets to be predicted and as predictors (features). Here, *New* is the target and *loves* and *York* are the predictors.

The Game and the System

The objective of playing prediction games is to improve prediction performance, subject to time and space constraints. The basic mode of operation of the current system is given in Figure 1. A stream of characters, for example from all the news articles or pages on the web, can be fed to the system. No preprocessing (*e.g.*, segmentation or tokenization) is necessary. Our system, at a high level, performs two related learning tasks: prediction and composition. In every episode, a *target* category is hidden, and the categories appearing in its context, the *active* categories (or predictors) are used to predict it. Then appropriate prediction edges are updated (some edges may be added or dropped), and relevant composition statistics are updated. A new category (composed of two previous categories) may be added to the system (assigned the next available id). The system begins at the level of predicting single characters: characters form the context and targets. In Figure 2, *e* is the target and *N*, at position -1 (before target), and *w*, at position +1, are the active categories (predictors) comprising the context. The context can be larger or limited to one side of the target. After each episode, the window (containing context and target) moves right to incorporate the next category (and drop the left most category). The next target in the picture is *w*. Later in the game, bigger categories, such as *New* have been acquired, and the context is composed of bigger categories too. The stream is segmented into categories incrementally, using the currently largest category that matches. For example, when processing "New York Mets game..", if *New York* is already an acquired category, but not *New York Mets*, then *New York* is the category that is picked, when the window is moved right (instead of *N* or *New*), and (its id) used for target as well as the context for previous or subsequent categories. The function of finding the longest category is currently implemented efficiently via a trie data structure.

Each category is represented as a node in a sparse directed weighted graph. Each node has 0 or more out-edges, connecting to categories that it predicts well. Edges are weighted and edges are grouped by the position of the category in the context: the edge corresponding to position +1 before the target is different from the edge corresponding to position 2 before or 1 after, even if they both predict the same category. Edges that belong to the same group are updated together (Section). Each category may also keep a short list of candidate categories with which it could compose at some point to form a new category (this depends on the method used for category composition). The graph is sparse: while there can be millions of categories, each category connects to a tiny fraction (in our experiments, in the order of 10s).

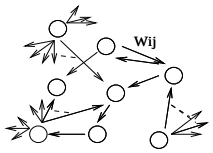
We next describe evaluation and the operations of prediction and composition in more detail.

Prediction

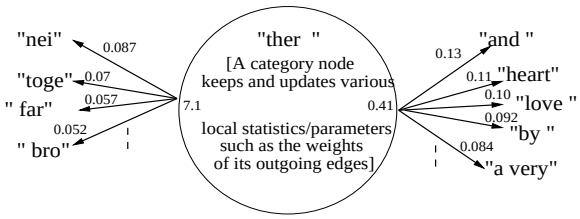
During prediction the out-edges in the appropriate position for each active category become *active*. These edges are used to score candidate categories. The score of a candidate category is simply the sum of the weights of its incoming active edges. For instance, say the active category *New*, at position -1, has only two out-edges, and assigns a score of 0.1 to the category *York*, and 0.07 to *Jersey*. Say the active *Nets* is at position +1 and assigns a 0.05 to *Jersey* ("New Jersey Nets" is a professional team) and 0 to *York* (effectively). Thus *Jersey* obtains a higher score than *York* (0.05+0.07 versus 0.1). The candidate categories are then ranked (sorted) by their score. The prediction of the system is the top-ranked category.

Performance Evaluation

We define the prediction performance, that we will refer to as the *matching performance*, to be the number of characters correctly predicted (matched) per prediction action. The system is successful so long as the "picture" that it predicts matches reality well. Intuitively, we seek a system that strives to predict bigger and bigger portions of its reality or "future" events. Predicting bigger parts or events in one's world (in each prediction action), not only implies that one has learned more about one's world, but it also implies one is faster in reacting to events (as long as we assume that *categories are atomic*: predictions and weight adjustments cost roughly the same no matter how big the the extent of the category predicted). This matching performance is further discussed and motivated in (Mad07). The prediction of the system is the category with the highest received score. If it is the target category, then the system obtains a reward equal to the target string length, otherwise 0 reward. We report the average of this measure as



(a) The sparse directed weighted graph of categories.



(b) A catregory node.

Figure 3: (a) Categories (category nodes) and their connections form a sparse directed weighted graph or network. The graph evolves and grows in nodes and edges. (b) A category node has a number of weighted out-edges, grouped by position. The category updates local statistics, such as most recent time seen, as well as edge weights when activated. Here we show an example node corresponding to "ther " from processing Jane Austen's six online novels. In position -1, "ther "connects to (predicts) "and ", "heart", etc, and in position +1, it connects to "nei" (which makes "neither"), etc. It has a rating of 7.1 (signifi cantly more than 1, implying a good predictor) for position +1 (it predicts its previous category well).

the prediction performance. Relaxed versions of this all-or-nothing performance measure may be more useful for future purposes[1].

On the finite data sets that we report performance on, in addition to reporting periodically on the online performance of the algorithm, we keep a portion of the data (such as a single file from a set of files) as test. This is so we can measure the effect of multiple passes on training data on performance on train as well as test data (how much fitting and overfitting occurs). This mode of experimentation is useful in a number of scenarios, for instance for specializing a generically trained system (*e.g.*, on all classical musics) on a subset (the music of Beethoven only). In this regime, we seek a system that learns as quickly as possible, while avoiding overfitting.

We have also monitored other statistics, some of which are related to prediction performance, such as the number of categories acquired by the system, and the average length of the target category in the input stream. We also report on these measures.

Learning to Predict

A high level description of the algorithm for edge weight updates is given in Figure 4. Versions of this algorithm are proposed and extensively empirically studied in (MC07).

[1]For example, the system could be partially rewarded for partial matches or when the best matching category isn't ranked highest. We have monitored these other measures in our experiments, and for our current system the simple all-or-nothing version appears adequate.

In each episode after prediction (scoring and ranking),
if update is necessary:
For each predictor (active category):
Update its predictiveness rating.
Increase the weight of its active edge to target.
Weaken weights for active (same position) edges.
Drop edge weights below a threshold (weak connections).

Figure 4: Each predictor performs local weight adjustments to improve its predictions (ranking of categories) over time.

Let c be the true category, and i be the predictor category.
$\forall$ **concepts** $j, w_{ij} \leftarrow w_{ij}(1-\beta) + \beta[j = c]$
$\forall$ **concepts** $j, w_{ij} \leftarrow w_{ij}[w_{ij} \geq \beta/5]$

Figure 5: An example form of edge updates for category i. Here, $0 < \beta < 1$, and in our experiments $\beta = 0.05$. w_{ij} denotes the prediction weight from category i to category j. Notation $[j = c]$ is the Kronecker delta (if the condition in brackets is true the value is 1, and otherwise 0). Thus, all edges are weakened (those with weight 0 need not be processed), except that the connection to the correct category c is boosted with β. If a weight falls below $\beta/5$, it is set to 0 (removed).

Edge Weight Updates Categories begin with no edges (*i.e.*, implicitly 0 weights). The essential steps in an update include, for each predictor (category in the context), strengthening the weight of the active edge connecting the predictor to the target (if not present, the edge is added), and weakening other edge weights in the same group (position) of that predictor. Thus the target would obtain a higher score by each individual active category, if a similar situation (context) occurs, and the target category may be ranked higher next time. An example of such an update is shown in Figure 5, for proper choice of *boost* or learning rate β, $0 < \beta < 1$. Note that the update need only examine the edges already connected to the category, plus possibly one additional edge (to true category), thus it is efficiently implemented: the out-degree (number of edges) for each prediction position is at most $5/\beta$. We used $\beta = 0.05$ in our experiments.

The weights are upper bounded by 1.0, and lower bounded by zero. A fixed range is important for long-term learning, as we don't desire the weights to increase or decrease without bound. Let the conditional probability $P(c_i$ is the target $|c_j$ is in position $k)$ or $P(c_i|c_{j,k})$ for short, be the conditional probability that the target category is c_i given the predictor category c_j appears in context in position k. The weight for the edge emanating from c_j, in group (position) k, connecting predictor c_j to c_i, corresponds roughly to $P(c_i|c_{j,k})$. Below we explain what the weight exactly is. This probabilistic semantics makes it convenient to assess the importance of edges: edges with weight lower than a threshold (0.01 in our experiments) are removed. This reduction is very important for keeping memory consumption in check. Just as important, this also preserves prediction speed and overall processing time (for updates, etc). Also, as a conseuqnce, edge groups have finite-memory: a new connection can quickly gain enough weight

to beat the currently highest weight connection. This aspect is important to adaptation to non-stationarity or drifts[2].

Not every episode involves an update: If the target category received a score significantly greater than the closest contender (false category), then there is no update (MC07). This is a mistake-driven update, akin to update policies in other online algorithms. This update policy accounts for possible dependencies among features and improves prediction performance in general (compared to updating in every episode). The mistake-driven update is the main reason that the weights are conditional probabilities for the substream of learning episodes containing c_j as a predictor: the substream for which the connection of c_j are updated. Another source of approximation to the conditional probability is the rounding of tiny weights to 0, and the granularity that updates by β imposes.

Predictiveness Ratings We have also observed that keeping and updating a predictiveness rating for each category is very useful for online long-term learning. The rating is a measure of how good the category is in predicting. It is initialized at 1.0, when the category first appears and a different rating is kept for each of its positions. The rating is adjusted (increased or decreased) in each episode the category is active, based on how the category scored the target category, compared to the average score the target category received (from all the active categories). The rating is used in two ways: to adjust the score the category assigns to its predictions[3], as well as adjust the allowance for the number of categories it may predict (for the corresponding position). Over time, uninformative categories (*e.g.*, non-semantic or functional categories such as white space) obtain low ratings (below 1), while better than average categories obtain high ratings. Note that this mechanism can potentially improve both accuracy and speed and space-efficiency, and in general may lead to better utliziation of space for the goal of prediction. Uninformative categories tend to appear more frequently, and since they tend to obtain low ratings and thus few prediction edges, they are processed quicker. We have verified this efficiency gains to be the case in our experiments, when we compare to the case where we don't adjust ratings (all ratings remain at 1).

Discussion There are likely to be numrous improvements to the algorithms presented. For instance, the objective of improving matching performance may perhaps be more directly incorporated into how weight updates or category rankings are performed. Adjusting predicitveness ratings may also be improvable.

Fundementally, a generic problem is how to efficiently allocate and reallocate space (manage space) for the various components. Such management decisions can include whether to add a new candidate category to the set of categories in the system, and whether to add or remove prediction edges to a predictor.

We have borrowed our current learning algorithms from our work on the problem of *large-scale many-class learning* (MGKS07; MC07). In these problems, the number of classes can be very large (1000s and beyond), and the dimensionality the number of instances is also high (possibly unbounded). The algorithms may be viewed as effectively learning a sparse feature-category index or what we have referred to as a *recall-system*: each feature (predictor) "indexes" a few categories (possibly 0), and the index, the patterns of connections, is efficently learned through experience. In that work we discuss the relation of the algorithms presented to previous online learning methods. We have observed that in terms of ranking quality, this indexing approach is competitive or better than other supervised methods (binary classifier based, *e.g.*, SVMs) in text classification, and it offers highly superior efficiency/scalability advantages (time as well as memory) for large scale learning (MC07).

Category Composition

What are good categories? For instance, having seen *n*, and then immediately after it *e*, should the system acquire the new category *ne*? Our objective is maximizing prediction accuracy (on unseen data) subject to efficiency constraints. Good categories include those that are predicted well and that help in prediction. Categories can also serve as cues for control mechanisms, *e.g.*, to alter or reset the context for prediction. We will explore the prediction role here.

The longer the category (simply string length here), if predicted well, the higher the prediction performance. Longer categories can help define the context better too. There is a tradeoff, however, between length and learning sample size. Longer categories require more bits of training, since longer categories simply take more bits to specify in the input, and the number of possible unique categories grows with category length. It is also possible that longer categories may be inherently more complex. For instance, a randomly picked category of length 4 (four characters), may require say on average in the order of a 1000 learning episodes (in which it's a target), to be predicted well (within say 95% of the maximum achievable accuracy), and a category of length 1 may require in the order of 100.

What are the criteria for candidate category generation processes? We seek efficient processes that generate good categories at high rates. Thus, we eschew from a process that requires explicit examination of a long history to determine goodness of a category, but some aspects of history may be summarized in a few statistics that can be efficiently updated. A category generation process may employ two types of methods or criteria, that we will refer to as 1) *pre-filtering*, and 2) *post-filtering*.

A pre-filtering criterion evaluates a category before the category is treated as a normal category and used for prediction. In exploring category generation, we have focused on methods that consider candidate categories that have appeared in the input stream at least once. This way, we avoid combinatorial blow up of considering all possibilities, though efficient alternatives may be possible. However, generating a new category for every co-location may still be too much: the system may run out of memory, and many such categories may be of little

[2]Note however: ultimately there is a tradeoff between plasticity (quick adaptation or change in weights) and stability.

[3]This way, candidate rankings become biased towards the ranking of highly rated predictors

use (*e.g.,* too infrequent). We next describe the pre-filtering methods we have experimented with, and report on their performance in the next section. In this discussion c_2 is the target and c_1 appears in position -1 (immediately before). The question is whether the composition, c_1c_2, should be added.

The **FRAC** (fraction) method adds c_1c_2 to the set of categories with probability p_s (independent of others). Lowering p_f avoids a blow up.

The **mutual information (MU)** method uses the pointwise mutual information between c_1 and c_2 (MS99): $\frac{P(c2|c1)}{P(c_2)}$, where $P(c_2|c_1)$ denotes the conditional probability that c_2 appears after c_1. These probabilities can be computed efficiently online by keeping track of appropriate statistics and short lists for each category (of which categories come after). If the ratio exceeds a threshold, and both c_1 and c_2 have certain minimum frequencies (*e.g.,* 20), the category is added.

The **IMPROVE** method attempts to check whether adding the category may improve prediction and the lengths into account as follows: Let $l_i = |c_i|$ (string length), and $p = P(c_2|c_1)$. For the second criterion, we ask whether it is better to predict c_1 alone or to have the option of predicting c_1c_2 together. When we are about to predict c_1 and are wondering whether to predict c_1c_2, if c_1 doesn't appear, the two possibilities are both equally bad. When c_1 appears, we get l_1 and then we may predict c_2 and obtain l_2. We use p as an estimate of the probability of predicting c_2 and being correct, thus our reward per action is roughly $\frac{l_1+pl_2}{2}$. If we predict c_1c_2 our reward is roughly $p(l_1 + l_2)$. The difference $p(l_1 + l_2) - \frac{l_1+pl_2}{2} \geq i_t$, for choice of threshold i_t, is our length heuristic.

The **BOUND** uses the FRAC method with $p_f = 1$, but does not generate categories with length exceeding a threshold l_t. We devised this method after examining the distributions over categories (their length and frequency in training set) generated by RAND and IMPROVE methods.

A post-filtering (or recycling) method applies criteria after the candidate category has been added to the set of categories for prediction purposes. A criterion, such as how many times a category's (training) *frequency*, *i.e.,* the number of times it has been seen as target so far during training, and how well the category is predicted or predicts, may be used to decide whether the category should be kept. A post filtering method may be executed periodically to lower its overhead. So far, we have experimented with the criterion of frequency: if the frequency is below a threshold, from the last time the recycling was executed, remove the category.

A category generation methods may employ a mixture of prefiltering and post-filtering. In our current system, updates for category composition can occur independently of the updates for prediction, as the criteria we have experimented with do not require prediction. On finite data sets, category generation can occur before the learning of prediction weights begins.

Global Clock A global clock is incremented with each prediction (learning) episode. Each category keeps a number of statistics related to time, such as the first time it was created, the last time it was seen (most recent time it became the target), the number of times it has been seen, and the number of times it has been seen recently. These statistics are efficiently updated when the category is seen, or periodically (*e.g.,* after every 10 million episodes), as appropriate. Post-filtering methods can then use these statistics for prunning appropriate categories (*e.g.,* infrequent categories).

Categories and Nonlinearity Consider the predictions of the category *new* (in position -1) versus the sum of the predictions of the three consecutive categories *n* (position -3), *e* (position -2) and *w* (position -1). Intuitively, the category *new* affects the distribution of the characters that follow it more accurately than the sum of the predictions of the three categories (treated individually) would indicate. This is a kind of nonlinearity: while the system's prediction are linear (achieved by linear, that is, simple aggregation or summing operations), higher level categories achieve a kind of nonlinearity. This nonlinearity is with respect to the lower level categories: if we stopped category generation at the level of single words, with respect to individual words, predictions would still be linear.

Experiments

We have experimented with various text sources such as online novels (*e.g.,* Jane Austen's), the Reuters collection of news articles (RSW02), and web-search query logs. No segmentation or tokenization was performed: A unix command for running the system would simply be a pipe, such as: *"cat textFiles | predictionSystem"*. Often, we splitted a data set into test and training, and report on testing as well as training measures of performance. While learning prediction weights is online, multiple passes on the same data set can improve accuracy (matching performance), although it can lead to overfitting too.

The Reuters data is a collection of roughly 800,000 news articles from 13 months, spanning a year. The files are in XML format and our reporting focuses on the result of feeding the system only the news body portion. This amounts to 1.2 billion characters. As XML is somewhat regular, matching performance increases if we include it (see below). We kept month 8, 1996, for test. Other files are processed in a fixed order one after another. Figure 6 plots, the average length of a category observed on training as well as test files, and the (matching) performance, *i.e.,* the average number of characters successfully predicted per prediction, against number of episodes. In this experiment, we used the IMPROVE heuristic for category generation, with a threshold of $i_t = -3$. Each pass took about 3 hours on a 2.4GHz AMD Opteron processor with 64GB of RAM. We stopped category generation after the first pass (otherwise, "memorization" can occur, see below). The first pass takes 134 million episodes, and subsequent passes take just over 110 million. Note that if the system continued segmenting at the single character level, then the number of episodes would be over a billion. We see that average category length on test grows to 9-10 range, while test performance improves to close to 2 characters per episode on average. Training performance (reaching about 6 characters)

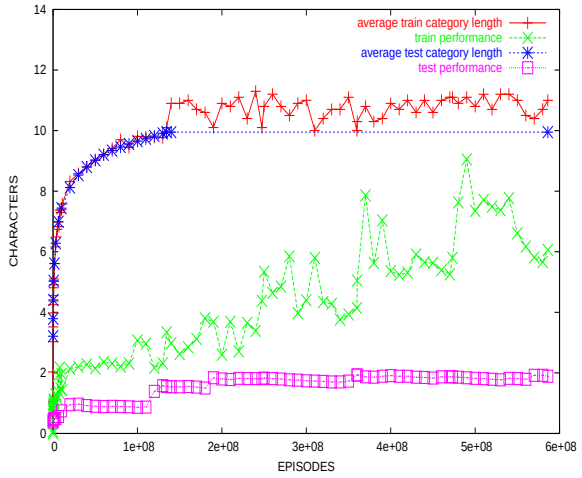


Figure 6: The growth in the average length of target category as well as matching performance (average number of matched characters), on train as well as test, as a function of number of episodes, for 5 passes. Category generation is stopped after pass 1 (about 150 million episodes). Improvements in test performance subsides after pass 3. While training performance continues to grow.

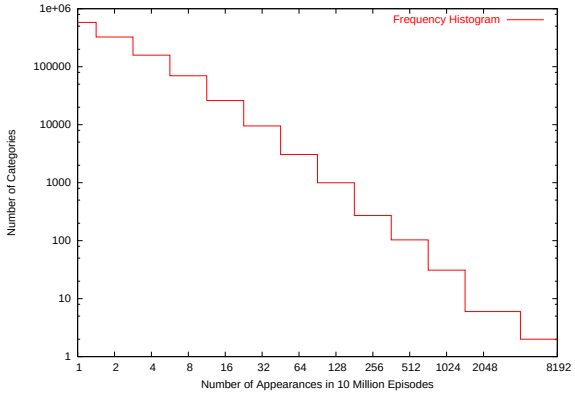


Figure 7: The histogram of category frequencies just before ending the first pass (on Reuters). Over 500k categories appear exactly once (after generation). Over 200k are seen more than 4 times.

is significantly better than test (training performance is an online average of the last 10000 episodes).

A total of over 14 million categories are generated in the first pass. Not all the categories generated are useful. As a measure of utility we consider the occurrence rate [4]. In every period of 10 million episodes, we have seen that over 500k of the categories are seen again (in the first pass, after first time generation), and over 200k are seen at least 8 times (in the second pass, these numbers jump to 1.4mil and 800k respectively).

We have observed that performance on test improves as we lower the threshold i_t, however the memory capacity of the machine does not allow us to go beyond[5] 20mil. The number of prediction edges after the first pass is just over 200mil, it climbs to over 300 million in the second pass, and stays there. The number of edges touched during prediction *per predictor* climbs to around 20 at end of pass 1 (for a total of just over 120 per episode, as there are 6 predictors), and stays there for the remaining passes. This is a promising sign: it provides some evidence that the system does not slow down over time. Of course, more experience is needed to verify this (and as we add more functionality).

While the average target category length is around 10, the longest category found and seen at least 5 times after generation is over 200 characters long. An example is: "In its three-grade rating system, the research institute assigns a B rating to ... Stock Exchange first-section TO". This category gets a relatively high rating of 1.7 (1 is the initial assigned rating) as a predictor, and at position -1 its best prediction is "PIX index over the next six months.". Other types of long categories included table headers and other structural categories. A category such as ", " is not informative, and obtains the minimum rating of 0.1 [6]. The maximum out-degree ranges in the 100s and categories such as " Net ", " Current ", and " Rev " get such high out-degrees, connecting to numbers.

As expected the performance degrades somewhat, from 1.8 down to 1.6-1.7 characters, if we limit the context size to 1 category on each side. It drops to under 1.5 if we limit the context to one side (but keep three categories): thus the effective of having both sides is more pronounced. If we included all of the files (with XML), average prediction goes just above 2. Finally, if we did not compose (kept prediction at the character level only), we get performance around 0.4.

How to Generate Categories?

We found, somewhat unexpectedly, that the system can exhibit a kind of overfitting or memorization if we allow category generation to continue for several passes. In this case, longer categories that apply only to the training data are learned, and the system learns prediction weight values that only apply to such categories. On the test set, these categories do no occur (or not as often), and prediction parameters for the shorter categories would not have been learned as well. Indeed if we allow category generation for several passes on online novels of Jane Austen[7], the average category length obtained on the training stream would increase to above 40, and training prediction performance would be very close to it[8], while the test performance would be only 0.2 characters (less than the performance at single character level!). The best test performance

[4]Other utility indicators include how well a category predicts and generalizes to somewhat different text.

[5]IMPROVE is likley not the best category generation method however. See Figure 8.

[6]The minimum rating allowed is set to 0.1 to allow categories to quickly "bounce back" if they improve in performance

[7]For Jane Austen (4.3mil characters), we trained on 6 novels and tested on MANSFIELD PARK. Interestingly, the category "her" gets the lowest predictiveness rating in this data set.

[8]We have not tested whether the size of the system is small enough to achieve a significant compression of the novels.

Data Sets	FRAC	IMPROVE	BOUND
Jane Austen	0.65	0.65	1.0
Reuters 1 month	1.3	1.5	1.8
Reuters 12 months	1.5	1.88	2.2

Figure 8: A comparison of prediction performance using various category generation methods. The second row reports on training on one month instead of all 12 training files.

is around 1.0 on Jane Austen.

Among the methods we have experimented with for category generation, MU (point-wise mutual information) performed the worst (*e.g.*, performance of 0.6 characters on Reuters compared to performance exceeding 1.5 for better methods. The probabilities used for MU do not take the drift in category co-locations into account, *e.g.*, after *New York* is picked as a new category, *New Jersey* should then be next, but it may take a while for the MU ratio to reflect this. We have shown the performance of the other methods in Figure 8, for their best choice of thresholds. The IMPROVE heuristic often out-performed FRAC. We next explain some of the underlying reasons.

Consider randomly picking a target category from the test stream. One factor that determines the performance on the picked category is the length of the category. Another is how well the system can predict such a category, in particular compared to its subcategories. An important aspect for the second factor is the category's (training) frequency, t_c, *i.e.*, the number of times the system has seen that category, as target, in the training set, so that it has had the opportunity to learn to predict it well. We have observed that, for the same average target length (by tuning a threshold), the length heuristic yields categories with higher training frequency t_c than the simple FRAC heuristic (*e.g.*, a difference of average 1000 versus 600 times).

We have also observed, on Jane Austen and Reuters, with a fixed l_t (tested for $l_t \in \{1, 2, 3\}$, that when average frequency t_c of a category seen in test reaches in the 1000 to 2000, the performance reaches 90% to 95% of its maximum. This may imply that complexity of learning, the number of learning episodes, does not increase or perhaps only slowly increases with average category length. Furthermore, a rule of thumb that may hold is: for a given data set, the best choice of l_t is such that average frequency t_c is between a few thousand. If l_t is so low that average frequency is say above 10,000, we may be "underfitting": we could generate longer categories for better performance, while if l_t is such that $t_c < 1000$, the system probably has generated too many categories with insufficient training episodes for each. These observations, and the rough estimates depend of course on data, as well as the learning algorithms used. Figure 9 show the number of training episodes required (sample size) so that the average category frequency (of a randomly picked category on test) first reaches 1000 on Reuters. We see a steady increase in sample size as we increase l_t. The average statistics are probably not sufficient to determine what the best set of categories to generate and keep is. Pruning (or not generating) categories exceeding a length threshold, *i.e.*, the simple BOUND heuristic, appears to exhibit better prediction performance than IMPROVE.

$l_t = 1$	$l_t = 2$	$l_t = 3$	$l_t = 4$	5	6	7
20k	200k	500k	800k	1.1 mil	1.4 mil	1.8mil

Figure 9: The number of test episodes on Reuters until the average (training) frequency on test set exceeds 1000, under different l_t thresholds for the BOUND method.

Our major conclusion is that FRAC, a simple but plausible category generation procedure, can be significantly improved upon. We also experimented with use of FRAC in conjunction with periodic post-filtering (recycling), but simple heuristics such as removing categories with low frequency (*e.g.*, below say 10) did not improve accuracy, though the post-filtering does reduce the number of categories significantly. We feel effective category generation remains an open problem, and we are further exploring better category generation methods.

Related Work and Discussion

Related work includes various types of neural networks, feature and concept discovery (or induction), and language modeling. We share motivations or part of functionality with previous approaches, while we believe there are important differences. In particular, we think what sets this work apart in terms of goals is first, the scale, *i.e.*, considerations of space/time efficiency for long-terms and very large-scale learning, and second, supporting a type of functionality, extensive/flexible in some ways and restricted in others, that we believe will prove very useful.

Online pattern (sequence) discovery and use is not new. In particular, Pfleger's online system shares some similarities and goals (Pfl04), but the approach is very different. The primary goal there reduces to finding (approximately) the k most frequent sequences for a range of lengths n (for some desired k). The prediction methods, an extension of Boltzman machines (that attempt to learn joint distributions), and objectives differ as well. Recurrent networks (*e.g.*, (HKP91)) also attempt to learn what comes next, although the features or classes (the inputs) are fixed. Deep belief networks (HS06) are multi-layered networks that also learn in part in an unsupervised manner, and in part use labeled data and back-propagation to learn or improve weights. Much work on neural networks deal with static networks, but variants that involve explicit edge or node addition/deletion exist (SM02; Alp94; Pfl04). For instance, Neat, is a general optimization procedure (SM02), involves adapting a population of neural networks (nodes and links added may be added to the a network) via evolutionary algorithms. Note that learning during a lifetime, our work here, is different from changes in structure/algorithms over generations. Explicit edge or node deletion has been less common. Feature reduction (common in machine learning techniques) can be viewed as edge deletion, as well as removing nonzero edge weights close to say 0. Valiant, in order to implement several learning functionalities in his network model of neocortex (Val94), also makes an assumption that each node has more programming capabilities that what is assumed in typical work on neural networks.

Our prediction task in text shares similar goals to statistical language modeling (SLM) (Ros00; Goo01), but the goal is to significantly widens the scope (Mad07), as we seek to integrate efficient online learning of categories enjoying sophisticated structure and interaction. In our current system, categories remain fairly simple (they can be prefixes of one another, not typical in SLM). The most widely used method for SLM remains computing n-gram models. This involves counting word sequences (of certain length), interpolation for infrequent patterns, and making the independence assumption. The method is highly scalable and highly parallelizable, but not as powerful as more advanced supervised (discriminative) methods, such as perceptrons and linear SVMs. This point has been made before (*e.g.*, (EZR00)). But importantly, the methods we employ are also very scalable, and have been shown to be competitive in accuracy compared to perceptrons and SVMs in several text classification tasks (MC07).

In cognitive science and developmental literature composition has been referred to at times as chunking, and it is believed that at least one type of chunking, perceptual chunking, is an automatic and continuous process (GLC^+01).

For many learning problems, such as many text classification problems (RSW02), the choice of classes is determined by humans, and the training data is also procured by explicit human labeling. In others, while the set of classes of interest are determined (to a good extent) by humans, humans are not explicitly involved in procuring the training data. For instance, the data may be obtained by the humans' "natural" activity. An example here is typical language modeling where individual words can be treated as classes. Here, we are exploring an extreme version of many-class learning: neither the choice of classes nor training data involves explicit human guidance. As human involvement lessens, and thus machine autonomy in learning increases, the amount of traing data and the number of classes tend to increase, but so does noise. We expect that this autonomous learning offers fertile ground for research.

Conclusions

We presented a system that functions in an online manner and autonomously learns by trying to improve its predictions over a stream of bits fed from a rich source. The learning is massive, *i.e.*, millions of pattern and prediction weights between them are acquired, but efficient. We overviewed some of the issues for learning of this scale and extent. We plan to extend the functionality, in particular, to integrate grouping mechanisms (or IS-A relations) into new category formation and to incorporate the learning and use of functional cues. There also exist many other problems that require closer study. We highlighted the problem of how best to generate new compositions. We also want to understand the nature of the networks learned and their evolution as the learning progresses. Finally, we like to compare against existing language modeling techniques and experiment with this basic approach in domains other than text.

References

E. Alpaydin. GAL: networks that grow when they learn and shrink when they forget. *International Journal of Pattern Recognition and Artificial Intelligence*, 8, 1994.

Y. Even-Zohar and D. Roth. A classification approach to word prediction. In *Annual meeting of the North Amercian Association of Computational Linguistics (NAACL)*, 2000.

F. Gobet, P. Lane, S. Croker, P. Cheng, G. Jones, I. Oliver, and J. Pine. Chunking mechanisms in human learning. *TRENDS in Cognitive Sciences*, 5, 2001.

J. T. Goodman. A bit of progress in language modeling. *Computer Speech and Language*, 15(4):403–434, October 2001.

J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Adison-Wesley, 1991.

G.E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 2006.

O. Madani. Prediction games in infinitely rich worlds. Technical report, Yahoo! Research, 2007.

O. Madani and M. Connor. Ranked recall: Efficient classification via learning indices that rank. Technical report, Yahoo! Research, 2007.

O. Madani, W. Greiner, D. Kempe, and M. Salavatipour. Recall systems: Efficient learning and use of category indices. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2007.

C. D. Manning and H. Schutze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

K. Pfleger. Online cumulative learning of hierarchical sparse n-grams. In *International Conference on Development and Learning*, 2004.

R. Rosenfeld. Two decades of statistical language modeling: Where do we go from here? *IEEE*, 88(8), 2000.

T. Rose, M. Stevenson, and M. Whitehead. The Reuters Corpus Volume 1 - from yesterday's news to tomorrow's language resources. In *Third International Conference on Language Resources and Evaluation*, 2002.

K. O. Stanley and R. Miikkuliainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

L. G. Valiant. *Circuits of the Mind*. New York: Oxford University Press, 1994.