Active Information Acquisition

He He
University of Maryland, College Park
hhe@umiacs.umd.edu

Paul Mineiro **Nikos Karampatziakis**
Microsoft CISL
{pmineiro,nikosk}@microsoft.com

Abstract

We propose a general model that sequentially and dynamically acquire useful information to solve a task under the Learning to Search framework. By focusing on most prominent parts of each instance, our method obtains promising results on sentiment analysis and image recognition. The model also learns to give harder instances more attention without explicitly being trained to do so.

1 Introduction

Attention is a mechanism that forces humans to work with a limited set of information at any given time. We do not process data all at once and do not pay equal attention to different pieces of information. Instead, a task is usually solved by dynamically seeking information needed most at the moment based on information at hand. Inspired by the dynamic attention in human perception, we propose a general-purpose framework that sequentially processes the input, adaptively selects parts of it and combines the acquired information to make predictions. Our framework can be applied to any base model (e.g. generalized linear models, neural networks) with any information unit (e.g. features, feature groups or pieces of raw input).

Given a prediction task, our goal is to learn a task predictor and an information selector. The task predictor takes information acquired by the selector and generates outputs defined by the specific task, such as object classes for image classification. The information selector acquires pieces of information based on past information and intermediate predictions given by the task predictor. We model this dynamism as a sequential decision-making process as shown in Figure 1 (left). The process stops when the model decides that enough information is obtained and outputs its final prediction. We learn the predictor and the selector by the Learning to Search (L2S) [1] framework, where a learner learns to make decisions by imitating behavior of a reference selector (c.f. Section 3).

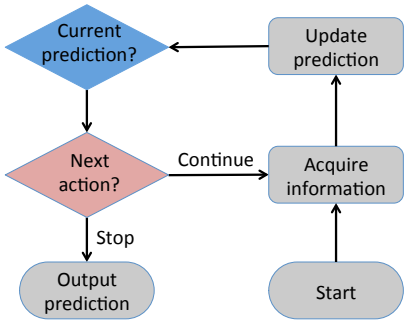
We evaluate our algorithm on a sentiment analysis task with a bag-of-words predictor, and an image classification task with convolutional neural networks (CNN). Our algorithm achieves better results than static information selection baselines on both tasks. Additionally, we show that the dynamic selector learns to acquire more information for hard, confusing examples than easy examples.

2 Framework of Active Information Acquisition

We assume that an input x can be represented by an information set of size n: $X = \{x_1, x_2, \ldots, x_n\}$ (and different instances may have different sizes). We denote a partial input by $x' \in (\{X \cup \varnothing\})^n$, a subset of information with unobserved parts indicated by $\varnothing$. Our model consists of an input feature map I, a task predictor T, a state feature map S and an information selector π.

Task predictor: T takes in the feature representation of x' and outputs a prediction based on partial information: $\hat{y} = T(I(x'))$.

State feature map: Our state includes (partial) input acquired so far and intermediate predictions given by T; S computes a feature representation of the current state, $S(x', T)$.



Algorithm 1 PREDICT (x, T, π, I, S)

$x' \leftarrow \varnothing^n$ ▷ Or initialize heuristically
for $t = 1$ to n **do**
  $\hat{y} \leftarrow T(I(x'))$ ▷ Intermediate prediction
  $a \leftarrow \pi(S(x', T))$ ▷ Selection decision
  if $a =$ STOP **then**
    return $\hat{y}$ ▷ Early stop
  else
    Update x' with new information x_a
return $\hat{y}$

Figure 1: Information acquisition at test time. *Left*: a flowchart of our algorithm. The blue diamond and the red diamond represent the task predictor and the information selector respectively. *Right*: pseudocode of the execution.

Information selector: π takes in the state representation and decides to acquire a new piece of information or to stop and output the current prediction. We denote the decision by an action $a \in \{i \,|\, i \in 1, \ldots, n \wedge x_i = \varnothing\} \cup \{\text{STOP}\}$.

At test time, given an input x, our model sequentially selects information and makes predictions as shown in Figure 1 (right). The implementation of T depends on the specific prediction task and is defined by the user, for example, neural networks. One difference with our predictor T is that it should be able to handle partial inputs. Nevertheless, it is easy to adapt most standard predictors to handle incomplete inputs: e.g. by replacing the missing parts with $\varnothing$ and training on incomplete inputs. The decision model π can also be any learning algorithm. Therefore, our framework covers a wide range of applications. In Section 4, we detail implementations for sentiment classification of reviews and image classification.

3 Learning to Search

Our framework builds on top of the Learning to Search [1] (L2S) paradigm, which allows us to jointly train the (interdependent) information selector and the task predictor via online cost-sensitive classification. The L2S algorithm requires three components:

A **search space** which includes states, actions, and transitions. States and actions have been defined in Section 2. After an action is taken, the current state transitions to a new one as in Algorithm 1.

A **loss function** to evaluate the output given by an action sequence. To learn a trade-off between the amount of information and the quality of the prediction, we define the loss function as

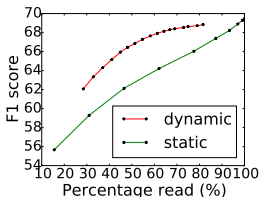
$$\ell(\hat{y}, y, x') = \ell_{task}(\hat{y}, y) + \lambda \cdot |x'|, \tag{1}$$

where ℓ_{task} is the loss function specific to the task, and $|x'|$ denotes the quantified amount of information in x', e.g. number of observed parts. Here λ controls the amount of penalty on acquiring more information. Since the loss function can be applied to results at any time step, we call loss at the end the *terminal loss* and those at earlier time steps the *immediate loss*. L2S optimizes the expected terminal loss.

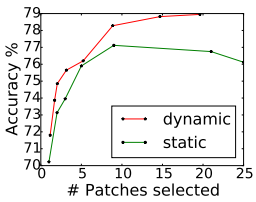
A **reference policy** on the training data that suggests a good action to take in any state during prediction. We use a greedy reference policy that always chooses the next piece of information that yields the lowest immediate loss.

L2S calls the PREDICT function (Algorithm 1) many times to explore different action sequences and figure out the ones that have a low terminal loss, much as in reinforcement learning. However, with a reference policy, L2S can explore the search space more efficiently by initially focusing on areas close to the action sequences generated by the reference policy and gradually deviating away by following the learned policy [1].

We can jointly learn everything with L2S. However, a more effective optimization strategy in practice is to pre-train the task predictor a bit before entering into end-to-end training, similar to curricu-



(a)



(b)

7	6	6	6	7
6	5	3	5	6
6	2	1	4	6
6	5	4	5	6
7	6	6	6	7

(c)

Figure 2: Comparison between pareto frontiers of dynamic (ours) and static selection: (a) sentiment analysis; (b) image classification. (c) Patches selected by the static baseline: the k-th model selects all patches with number smaller or equal to k.

lum learning [2]. We use complete and randomly sampled information for pre-training; and fine-tune the task predictor with inputs generated by the learned information selector after each online update.

4 Experiments

4.1 TL;DR: Sentiment Analysis of Book Reviews

Here we predict a user's rating by reading their review sentence by sentence from the beginning. We use sentences as the units of information. The model dynamically decides whether to continue reading the next sentence or to stop and output the current predicted rating, hence we refer to it as TL;DR. We evaluate TL;DR on book reviews from the Amazon product data [3], where each review has a rating of $\{1, 2, 3, 4, 5\}$. We select reviews with 5 to 10 sentences and split the dataset into three sets: 1M for pre-training the task predictor, 8M for L2S and fine-tuning and 1M for testing. Our task predictor is a linear multiclass classifier using a bag of unigrams and bigrams representation (no author features). We pre-train the predictor on complete reviews and their prefixes.

The state features are the intermediate scores (negative log likelihood) for each class given by the task predictor, the difference between the highest and the next-highest score, i.e. the score margin, the KL-divergence between the current belief and the class prior, [1] the current prediction, and the number of sentences read. Our information selector is a quadratic classifier (using Vowpal Wabbit).

We compare with models that use a static selector that always selects the first k sentences ($k \in [5, 10]$), and a task predictor trained on both the pre-training data and the L2S+fine-tuning data. We sweep over λ to obtain a range of models that reads different numbers of sentences on average. Larger λ discourages the model to use more information. To show the trade-off, we compare performance of our dynamic model with the baseline static model given various amounts of information. The result is shown in Figure 2a and our model completely dominates the static selection method. To examine where the model decides to acquire more information, we compute the average percentage of sentences read for each class/rating. For example, for $\lambda = 0.0125$[2], the result is 61%, 70%, 69%, 59%, 38% for ratings from 1 to 5 respectively. The model reads much fewer sentences for the easy rating-5 (majority) reviews and more for confusing reviews in the middle. This shows that the model learns to acquire information adaptively according to how difficult the example is.

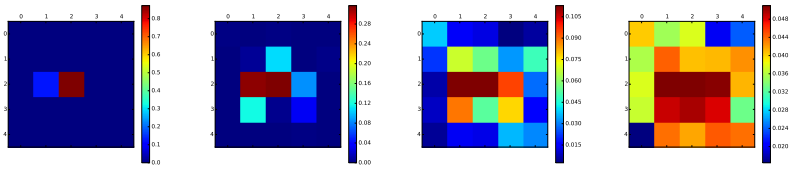
4.2 Image Recognition

In this experiment the goal is to recognize objects by looking at a few patches from an image. This scenario is a toy version of a robot/camera trying to making sense of a scene by deciding where to focus. Under our information acquisition framework, the model starts from an empty image and adaptively selects a sequence of patches to examine until it feels confident about the prediction and stops. We evaluate our algorithm on the image classification task from PASCAL VOC Challenge 2007.

We resize all images to 256×256. Each image is divided into 25 equal-sized and overlapping square patches, thus the information unit is one patch. Our task predictor takes features extracted from the

[1] The prior class distribution is imbalanced in this dataset: more than 50% reviews have a rating of 5.

[2] Other values demonstrate similar behavior



(a) $\lambda = 3.5$ (b) $\lambda = 2.0$ (c) $\lambda = 1.0$ (d) $\lambda = 0.0$

Figure 3: Heat maps of frequencies a patch get selected at different λ.

selected patches and predicts the objects in the image. There are 21 object classes including the background. For simplicity, we focus on the task of predicting whether a person is in the image (the majority class that often co-occurs with other classes). To obtain patch features, we label each patch with its image (multi-)label and fine-tune the pre-trained VGG-16 [4] model from Caffe with the patch examples. We use the predicted probabilities output by the softmax layer of VGG network as the patch features.[3] The state features are based on intermediate scores, similar to TL;DR.

We compare against static selectors that always select a fixed subset of patches. As it is computationally expensive to enumerate all possible subsets, we selected a family of subsets that cover the image from the center to the outer parts. Details are shown in Figure 2c. We obtain similar results to the sentiment analysis task: Figure 2b shows a better trade-off than static selection, for any number of patches. The dataset also contains annotations about hard instances, which we use to confirm that the model learns to use more information for hard examples: For difficult images the average number of patches selected is consistently larger than for common images. For example, when $\lambda = 1$ these averages are 10.4 and 8.8 patches respectively. To examine where the model pays most attention, we show heat maps of the attention of models with different trade-offs in Figure 3. The result is consistent with our intuition: when the amount of information is restricted, look mostly in the center where the object is more likely to be located; when more information is allowed, dynamically explore outer parts.

5 Related Work

This work is most related to the recurrent visual attention model [5]. They model the dynamic search as a recurrent neural network trained with the REINFORCE algorithm. Recently, Luong et al. [6] also proposed the local attention model for machine translation. It is similar to ours in the sense that it sequentially selects local windows to attend to instead of weighing all input words. Our approach, however, is more flexible in defining the decision model and can adaptively choose when to stop instead of taking a fixed number of steps. In addition, our loss function explicitly encode the information-accuracy trade-off.

References

[1] Hal Daumé III, John Langford, and Stéphane Ross. Efficient programmable learning to search. In *arXiv*, 2014.

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of ICML*, 2009.

[3] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of SIGIR*, 2015.

[4] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *arXiv*, 2014.

[5] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent models of visual attention. In *Proceedings of NIPS*, 2014.

[6] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of EMNLP*, 2015.

[3] We have also tried to use features from the fully-connect layer but find it was not helpful.