Document Analysis with Transducers

Léon Bottou, Yoshua Bengio, Yann Le Cun
AT&T Laboratories

DRAFT DOCUMENT

July 3rd, 1996

1 Introduction

We present here a general architecture for building Automatic Document Analysis Systems. This architecture is composed of a succession of modules transforming graphs describing lower-level hypotheses on the documents into graphs describing higher level hypotheses. This architecture generalizes techniques used in Neural Networks, Optical Character Recognition, Natural Language Processing and Speech Recognition.

Section 2 presents the motivation for this work. We discuss in section 3 how we use weighted graphs for describing a document at various levels. Section 4 explains how these modules are built using generalized transducers. Section 5 explains how we can adapt simultaneously the parameters of all modules.

Finally section 6 briefly presents how these concepts were successfully integrated into an existing check amount reading system and suggests future directions. More details about this check reading applications can be found in (Haffner et al., 1996).

2 Motivation

Document image analysis applications always involve a succession of steps. These steps make more and more refined *statements* about the document. For instance, a check amount reading system needs to locate the amount, recognize the amount writing style (e.g. whether the amount is decimal or fractional), remove unnecessary ink (i.e. remove amount boxes, clean up image defects), segment individual characters, recognize each character and finally interpret the sequence of characters as an amount.

Computer science practice suggests to solve each step independently and assemble the resulting modules. This strategy however leads to poor performance. For instance, locating the check amount consists in identifying a part of the document image that fulfills two kinds of criterions:

- Stage specific criterions: the ink shape must display the general features of a check amount (width, height, moments, etc.).
- System wide criterions: the ink shape must look like adequately located characters that the next stages can interpret as a check amount.

Overlooking the system wide criterions lead to poor accuracy. Any erroneous early decision will indeed prevent the system from working properly. Checking the system wide criterions by specific code in each stage provides better accuracy with an increased complexity. This complexity leads to practical problems which finally impact the system accuracy just as well.

Most character recognition systems nowadays tightly combine the segmentation and recognition stages (Denker and Burges, 1995) (Bengio et al., 1995). The high complexity of combining all steps of a real document image analysis system however has limited this trend.

Methods for combining segmentation and recognition have been pioneered by the speech recognition community. Hidden Markov Models (HMMs) for instance, simultaneously perform segmentation and recognition of the speech signal while observing the constraints of a language model (Rabiner and Juang, 1986). Recent advances with transducers (Pereira, Riley and Sproat, 1994) have provided an elegant framework for designing such systems. This framework has also been suggested for character recognition applications (Guyon, Schenkel and Denker, 1996).

We present in the next sections a transducer based architecture for efficiently combining all the steps of a document analysis system while maintaining a handy modularity between the various stages of the system.

3 Representing Documents with Weighted Graphs

There is a straightforward approach for combining several stages in such a system. Instead of returning the top scoring statement, the downstream stages can just give scores to a set of hypotheses. For instance, the amount location module of a check amount reader can output several hypotheses with scores based on the stage specific criterions only. The following stages will then be called for all hypotheses and will generate hypotheses (about the writing style, about segmentation, etc.) and cumulate the scores.

The exponentially growing number of hypotheses forbids to use such an approach directly. The rich structure of the hypotheses themselves however provide a solution. A segmentation hypothesis, for instance, splits the field image into several possible character images. All segmentation hypotheses however are likely to share a same set of candidate cuts between characters. Segmentation hypotheses only differ by the subset of cuts to consider. Once we have decided to consider a given cut, the

segmentation problems on both sides of the cut are quite independent. For this reason, system combining segmentation and recognition often represent segmentation hypotheses using Markov models or more simply weighted oriented graphs (Denker and Burges, 1995).

The proposed architecture *extends this graph representation to all stages of the system.* Each arc of these graphs has a score and a data structure attached. Each stage of the system takes a weighted graph as its input and produces a new weighted graph. These graphs in fact represent the *system's understanding of the document after each processing stage.* Each path between the start node and the end node of a graph *fully describes an hypothesis concerning the interpretation of the document.* This description relies on data structures attached to the arcs. The score of each path (i.e. the sum of the scores of the corresponding arcs) indicates the relative system's confidence in each hypothesis.

In other words, we represent the hypotheses produced by each stage of the system by a sequential language whose words are the data structures associated to the arcs. We can compute a score for each legal sentence of this language by summing the scores of the individual arcs of the graph. Since these scores comply with Bellman conditions, we can use efficient graph algorithms and therefore reduce the computational complexity of the system to acceptable levels.

Unlike speech signal, documents are laid out on a two dimensional surface with no natural order. Yet we represent hypothesis on the document by a sequential language. The criterion for selecting an ordering should be the size of the resulting graphs. Although the complexity of this task is difficult to assess, humans seem very good at picking reading orders that produce computationally efficient graph based representations.

3.1 Illustration

Let us illustrate this representation by describing a simplified architecture for reading checks. Real check reading systems however tend to be more complex and less elegant.

- The system is provided with an *initial graph* whose single arc points to a data structure representing the check image (see figure 1).
- The amount location stage analyzes the check image and computes a score for several candidate fields for the check amount (see figure 2). The *field graph* encodes these candidate fields as alternative arcs. Each arc points to a data structure describing each field.
- The modelization stage analyzes each field and produces hypotheses about the field writing style (i.e. machine printed, handwritten decimal, or handwritten fractional). In the case of a handwritten amount, the modeling stage searches for possible fractional bars and locates the dollar amount and the numerator of the fraction. The *modelization graph* represents all these alternatives. For each field, a field marker arc carries whatever field related information may be useful for the next stages. The following arcs then describe subfields that we are going to segment and submit to the isolated character recognizer (see figure 3).
- The segmentation stage identifies candidate cuts in each subfield. Segmentation hypotheses are then encoded by the *segmentation graph* (see figure 4). The segmentation graph basically

replicates the modelization graph but replaces the subfield arcs by a part marker (representing whatever subfield information may be used by the following stages) followed by a subgraph whose arcs represent the candidate character images.

- The composition (cf. section 4.1) of the segmentation graph and of a predefined grammar graph produces the interpretation graph (see figure 5).

  Each path of the *grammar graph* describes a legal sequence of markers and characters that can be interpreted as a check amount. Each arc of the grammar describes (a) the marker or character observed on the image and (b) symbolic information that will ease its interpretation.

  The *interpretation graph* contains the subset of the grammar graph paths that are compatible with the segmentation graph. An isolated character recognizer is called whenever a candidate character image arc in the segmentation graph matches a character arc in the grammar graph. The scores of the resulting arc in the interpretation graph is the sum of the segmentation graph score, of the grammar graph score and of the score returned by the isolated character recognizer.

- Finally a search algorithm (e.g. the Viterbi's algorithm) extracts the best scoring path in the interpretation graph. This best path is encoded as a single path *Viterbi graph* (see figure 6). The amount read is then obtained by parsing the symbolic information copied from the grammar graph and attached to the arcs of the Viterbi graph.

3.2 Probabilistic Interpretation of Scores

The probabilistic interpretation of the scores in discriminant pattern recognition systems combining segmentation and recognition (speech recognition or optical character recognition alike) has been a puzzling problem for years (Devijver, 1985) (Bourlard and Wellekens, 1989). This problem consists in transforming our scores into quantities that fulfill the basic properties[1] of a probability (Bridle, 1989).

a) Probabilities must be positive. The probability that any one of several disjoint events occurs is the sum of the probability of the events. The probability of the simultaneous occurrence of independent events is the product of the probabilities of these events.

We can easily enforce this positiveness constraint by considering that the scores are in fact the logarithms of the actual probabilities. This encoding moreover improves the numerical

[1]We refer here to Kolmogorov's definition of a probability: A probability defined on a space of outcomes (called *the universe*) is a function that associates a number $P(E)$ to certain subsets E of the universe (called the *events*). This function must satisfy three basic axioms:

a) A probability is always positive,

b) The probability of the universe is exactly 1,

c) The probability of the union of two events A and B is $P(A \cup B) = P(A) + P(B) - P(A \cap B)$.

Moreover, we can further restrict the set of allowable probabilities by declaring that certain pairs of events are independent. The probability of the intersection of two independent events A and B is $P(A \cap B) = P(A)P(B)$.

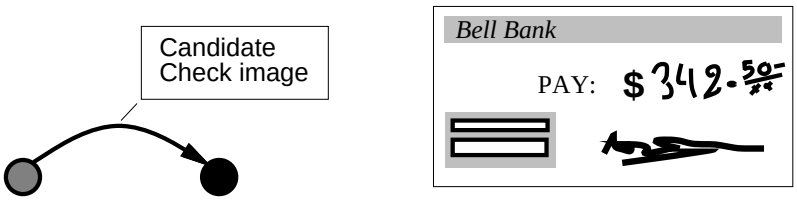

Figure 1: *The initial graph just contains the check image.*

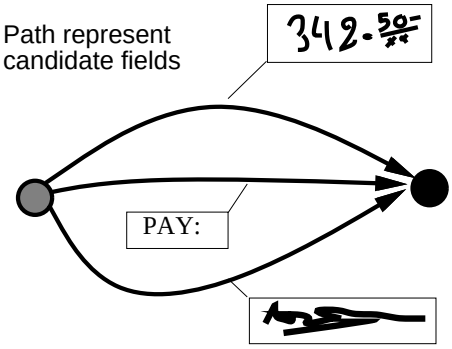

Figure 2: *The field graph scores all possible fields for the amount.*

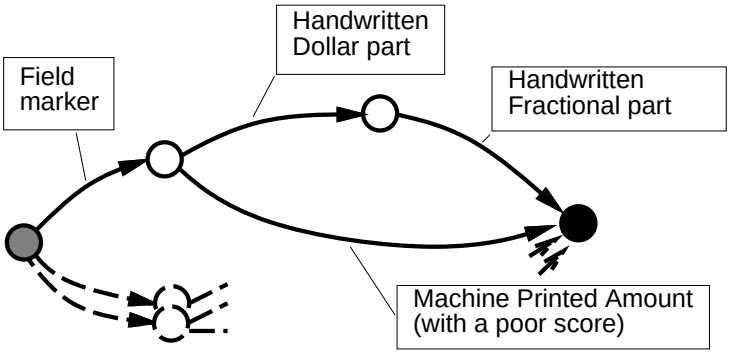

Figure 3: *The modelization graph scores elementary hypotheses on the amount style (machine printed, handwritten decimal, handwritten fractional, etc.) Special marker arcs carry field related information.*

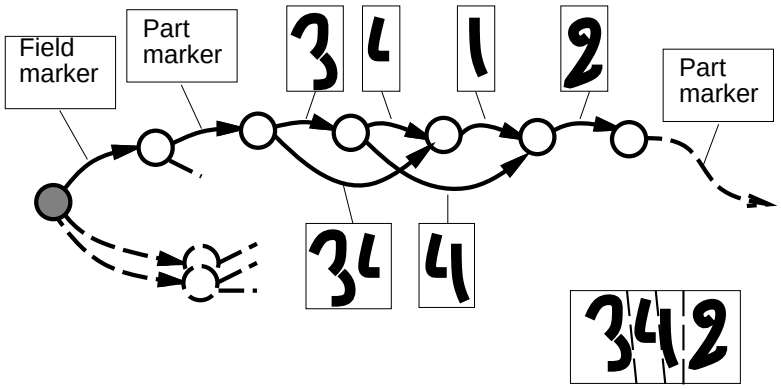

Figure 4: *The segmentation graph scores segmentation hypotheses. Special marker arcs carry field and modeling related information.*

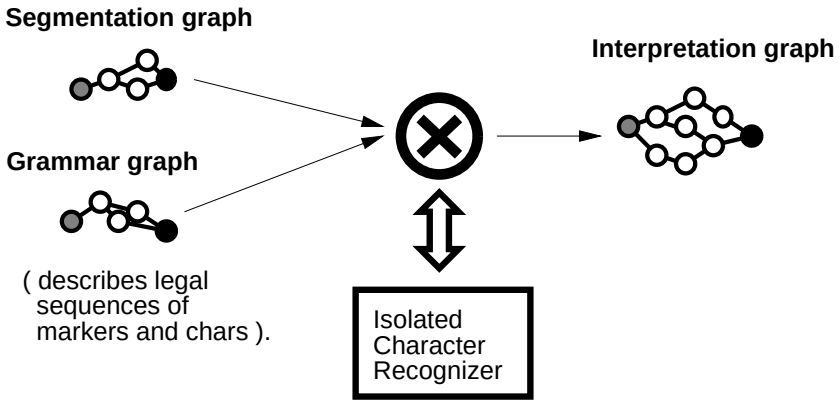

Figure 5: *The interpretation graph is obtained by composing the segmentation graph and a global grammar graph. This composition triggers the calls to the isolated character recognizer. The resulting graph provides a score and an interpretation for all legal sequence of markers and characters.*

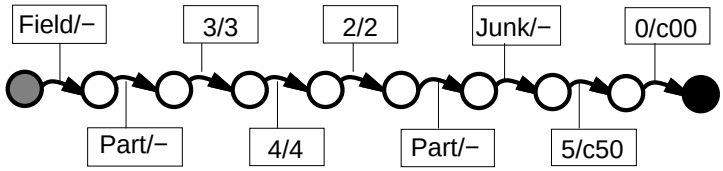

Figure 6: *The Viterbi graph contains a single path representing the most probable interpretation of the document image. Each arc label specifies the recognized character or marker and its interpretation: (character–or–marker / interpretation).*

stability of repeated computations. We can indeed perform direct computation on the scores by log-adding or adding scores instead of adding or multiplying probabilities. The log-sum of two values x and y is:

$$x \oplus y = \log(e^x + e^y) = \max(x, y) - \log(1 + e^{|x-y|})$$

b) Probabilities must be normalized. This normalization in fact means that we are delimiting a priori the set of outcomes. This set is then given probability 1. Assigning a probabilistic meaning to scores therefore implies *a non-trivial assumption* whose consequences must be clearly understood.

To illustrate this point, imagine for instance a check reading system that produces, for each check image I, a score $s(I, X)$ for all possible amounts X. If our scores are meant to represent the posterior probabilities $P(X|I)$, we must normalize our scores by subtracting the log-sum of the scores for all amounts.

$$P(X|I) = \frac{\exp(s(I,X))}{\sum_{X'} \exp(s(I,X'))} = \exp\left(s(I,X) - \bigoplus_{X'} s(I,X')\right)$$

We therefore make the *implicit assumption* that all check images bear an amount. The normalization indeed erased the information concerning incorrect checks.

Like (Bourlard and Wellekens, 1989), many authors choose to perform this normalization as early as possible. Scores are then combined using standard probabilistic methods (i.e. marginalization and conditionment). All scores in the system therefore have a clear probabilistic interpretation (as in a standard Hidden Markov Models).

Early normalization however may delete much needed information. Assume for instance that we normalize the scores returned by our isolated character recognizer. Since we are searching for the best segmentation, the recognizer will be submitted images that do not represent valid characters. Although the best scoring class for these images will have a very low score, this score may be higher than the scores of the other classes (e.g. all vertical bars look like character "1"). In such a case, the normalization would hide the fact that the image does not display a valid character. We would then select a wrong segmentation and produce an erroneous answer (Denker and Burges, 1995).

Following (Denker and Burges, 1995), we prefer to postpone the normalization as much as possible. Therefore we handle the arc scores and path scores as unnormalized log-probabilities (i.e. satisfying properties *(a)* but not property *(b)*.)

We can however compute a probability for each hypothesis represented by a graph if we assume that the graph represents all probable alternatives. We just normalize the log-sum of scores of the paths representing this hypothesis by subtracting the log-sum of the scores of all paths in the graph.

4 Graph Transduction

The proposed architecture is thus composed of a sequence of modules taking weighted graphs as input and returning new weighted graphs as output. We explain now how these modules share the same algorithmic structure.

Graph transformations have a limited number of uses:

a) Graph refinement inserts new alternatives into an existing graph. In our check reading example, the field detection (cf. figure 2), the modelization (cf. figure 3) and the segmentation (cf. figure 4) are graph refinement operations.

b) Graph matching produces a graph representing the paths common to two or more graphs. In our check reading example, the interpretation graph (cf. figure 5) represents paths matching both the segmentation graph and the grammar.

c) Graph pruning eliminates the low scoring paths of a graph. The last stage of our check reader, for instance, creates a graph whose single path is the best scoring path (cf. figure 6).

Graph pruning (c) algorithms are usually derived from well known graph search algorithms (e.g. Viterbi, Stack Decoding, Dijkstra, Beam Search, etc.). We will not discuss these algorithms any further. Both graph refinement operations (a) and graph matching (b) operations are elegantly described using *transducers* and *graph composition* (Pereira, Riley and Sproat, 1994).

4.1 Transducers and Graph Composition

Let us briefly recall the basic ideas behind graph composition and transducers. We assume here that the data structures attached to the arcs of the graphs carry *discrete information*. This discrete information can be represented by a finite number of symbols. We also distinguish a special symbol referred to as the *null symbol*.

We are interested in two kind of graphs, namely *acceptors* and *transducers*.

- Each arc of an acceptor is labelled with a possibly null *input symbol* and a score probability. Each path linking the start node and the end node of an acceptor represents a sequence of non null symbols. This sequence of symbols is obtained by concatening the non null symbols attached to the arcs of the path. The score of the path is the sum of the scores of the corresponding arcs.

  An acceptor therefore embodies a probability law (cf. section 3.2) on the possible sequences of non null symbols.

- Each arc of a transducer is labelled with an *input symbol*, *an output* symbol and a score. Each path linking the start node and the end node of a transducer represents two sequences of non null symbols: an input sequence and an output sequence.

A transducer therefore represents a joint probability on sequences of input symbols and sequences of output symbols. This joint probability describes a probabilistic rule for transforming sequences of input symbols into sequences of output symbols.

Given a transducer and an acceptor, we can compute the conditional probability law of the output sequences given that input sequences follow the law defined by the acceptor. The *graph composition* algorithm computes a new acceptor that embodies this conditional probability law.

Imagine two tokens sitting each on the start nodes of the two graphs (i.e. the acceptor and the transducer). The tokens can freely follow any arc labelled with a null input symbol. A token can follow an arc labelled with a non null input symbol if the other token also follows an arc labelled with the *same* input symbol.

We have an acceptable trajectory when both tokens reach the end nodes of their graphs (i.e. the tokens have reached the terminal configuration). This trajectory represents a sequence of input symbols that complies with both the acceptor and the transducer. We can then collect the corresponding sequence of output symbols along the trajectory of the transducer token.

The mere recursive enumeration of the acceptable joint token trajectories leads to a tree representing all acceptable sequences of output symbols. Elementary Bayesian computations provide transition probabilities for the arcs of this tree.

At any point of this enumeration, the future alternatives for our tokens are completely determined by the position of the tokens in each graph. It is thus easy to generate a graph instead of a tree by remembering which token configuration we have already visited and creating arcs to the node of the composed graph corresponding to this token configuration.

This algorithm calls for a few remarks:

- The enumeration of the joint token trajectories often leads to non terminal configurations for which no token transition is allowed (i.e. a dead end). Such trajectories must be discarded because they do not represent sequence of input symbols acceptable for both graphs. The proper implementation of dead end avoidance presents interesting book-keeping problems.
- We really care about null symbols because they provide the only way for matching sequences of symbols of different lengths. The proper handling of all combinations of null and non null symbols also presents interesting book-keeping problems.
- This algorithm can be generalized to the composition of any number of acceptors with a transducer. The composition of zero acceptors and one transducer is a trivial operation of little interest.

Despite these intricacies, graph composition can be made very efficient. Tranducers indeed have been used in natural language processing and speech recognition applications (Pereira, Riley and Sproat, 1994).

Let us now imagine that the data structures attached to the arcs of our graphs are easily represented

by a finite number of symbols. We could then implement all graph refinement and graph matching operations as a composition of the input graph and an appropriate transducer.

4.2 Generalized Graph Transduction

The data structures attached to the arcs of our graphs however often contain images. We must therefore take dispositions to avoid enumerating all possible images within a huge transducer. We present here a solution for this problem.

Instead of dealing with two categories of graphs whose arcs carry one or two symbols, we are managing many kinds of graphs whose arcs carry complex data structures. Composing graphs then requires some additional information:

- When we are examining arcs (one from each graph), we need a criterion to decide whether the data structures pointed to by these arcs match.
- When some arcs satisfy this criterion, we must know how to build the corresponding arc in the composed graph. We can build a single arc, build several arcs, or even build a small graph with several nodes and arcs.

We have thus introduced another object which holds this extra information. We call this object *transformer*. A transformer is a stateless object that implements two methods:

- Method `check(arc1, arc2)` returns a boolean indicating if the data structures pointed to by arcs `arc1` (from the first graph) and `arc2` (from the second graph) match.
- Method `build(ngraph, upnode, downnode, arc1, arc2)` is called when arcs `arc1` and `arc2` match. This method creates whatever arcs and nodes are required between nodes `upnode` and `downnode` to represent this match in the composed graph `ngraph`.

Figure 7 shows a simplified generalized graph transduction algorithm. For the sake of simplicity, this simplified algorithm neither handles null transitions nor implements dead end avoidance.

The safest way for implementing dead-end avoidance consists in running a preliminary pass for identifying the token configurations from which we can reach the terminal configuration (i.e. both tokens on the end nodes). This is easily achieved by enumerating the trajectories in the opposite direction. We start on the end nodes and follow the arcs upstream. During the main pass, we only build the nodes that allow the tokens to reach the terminal configuration.

The management of null transitions is a simple modification of the token simulation function. Before enumerating the possible non null joint token transitions, we loop on the possible null transitions of each token, recursively call the token simulation function, and finally call method `build`.

Ordinary graph composition is easily and efficiently implemented as a generalized graph transduction. The transformer method `check` just compares the graph input symbols. The transformer method `build` creates a single arc whose symbol is the transducer output symbol.

```
Function simple_transduction(PGRAPH graph1, PGRAPH graph2, PTRANS trans)
Returns PGRAPH
{
  // Create new graph
  PGRAPH ngraph = new_graph()

  // Create map between token positions and nodes of the new graph
  PNODE map[PNODE,PNODE] =  new_empty_map()
  map[endnode(graph1), endnode(graph1)] = endnode(newgraph)

  // Recursive subroutine for simulating tokens
  Function simtokens(PNODE node1, PNODE node2)
  Returns PNODE
  {
    // Check if configuration has already been visited
    PNODE nnode = map[node1, node2]
    If (nnode == nil)
      // Record new configuration
      nnode = ngraph->create_node()
      map[node1, node2] = nnode
      // Enumerate the possible non null joint token transitions
      For ARC arc1 in down_arcs(node1)
        For ARC arc2 in down_arcs(node2)
          If (trans->check(arc1, arc2))
            PNODE newnode = simtokens(down_node(arc1), down_node(arc2))
            trans->build(ngraph, currentnode, newnode, arc1, arc2)
    // Return node in composed graph
    Return nnode
  }

  // Perform token simulation
  simtokens(startnode(graph1), startnode(graph2))
  Delete map
  Return ngraph
}
```

Figure 7: *Pseudo-code for a simplified generalized transduction algorithm. For simplifying the presentation, we do neither handle null transitions nor implement dead end avoidance. The two main component of the composition appear clearly here: (a) the recursive function* `simtoken()` *enumerating the token trajectories, and, (b) the associative array* `map` *used for remembering which nodes of the composed graph have been visited.*

Like the graph composition algorithm, the generalized graph transduction algorithm can be extended to the transduction of any number of graphs using an appropriate transformer. Unlike graph composition however, the transduction of a single graph is not a trivial operation because the method `build` of the transformer is allowed to create several arcs or even a complete subgraph for each arc of the initial graph[2].

Single graph transduction indeed provides a cost effective way to implement graph refinement operations. In our check reading example, the field location, the modelization and the segmentation are implemented as single graph transduction.

More complex transductions are also useful. In our check reading example, the interpretation graph however is produced by transducing the segmentation graph and the grammar graph. The transformer calls the isolated character recognizer as a subroutine during the computation of the scores of the interpretation graph arcs.

4.3 Document Analysis Architecture

All stages of the proposed document analysis architecture are implemented as generalized graph transductions. Each graph in the transduction cascade represent the successive hypotheses considered by the system. We can search these graphs for the most probable hypothesis. We can also prune these graphs in order to reduce the computation time at will.

This regular architecture has many benefits:

- Each stage of the system is implemented as a well defined graph transformer. The communications between the various stages of the system is entirely carried out through intermediate graphs. We can thus easily replace any stage in the system by another compatible stage implementing new heuristics.
- The generation and selection of hypotheses is performed by a few generic graph functions (i.e. generalized graph composition and graph search) instead of random code all over the program. It is therefore considerably easier to optimize, analyze and debug this important part of a document analysis system.
- The complete system is organized like a multi-layer network (cf. figure 8). Instead of connection layers transforming a vector of states into another vector of states, we have transduction layers transforming graphs into graphs. This simple structure is well adapted to adaptive training algorithms.

[2]Single graph transduction is theoretically equivalent to the composition of our graph by a transducer belonging to a particular class. The equivalent transducer includes all subgraphs built by the method `build` for input symbols representing data structures accepted by the method `check`.

5 Global Training

All stages of a document analysis system contain adjustable parameters. In the check reading example, the field location, the modelization and the segmentation compute scores for each hypothesis on the basis of image features and adjustable parameters. The isolated character recognizer is a neural network with adjustable weights. The grammar graph may penalize misleading sequences of characters.

Most parameters affect the scores stored on the arcs of the successive graphs of the system. A few threshold parameters determine whether an arc appears or not in the graph. Since non existing arcs are equivalent to arcs with very bad scores, we only consider the case of parameters affecting the scores.

We can use training algorithms to ensure that each stage of the system performs its assigned task (e.g. field location, isolated character recognition, etc.) as well as possible. Both practice and theory suggest however that it is more efficient to bias certain stages in order to compensate the weaknesses of other stages.

Simultaneously training all stages of the system has been already explored in speech recognition and character recognition. Early papers (Bourlard and Wellekens, 1989) propose techniques that embed neural network training within the mathematics of Hidden Markov Models. Global training has then been formalized as the minimization of a single cost function (Bottou and Gallinari, 1991).

Most global training experiments combine segmentation and recognition, either in speech recognition (Bourlard and Morgan, 1990) (Haffner, Franzini and Waibel, 1991) or in character recognition (Denker and Burges, 1995).

Experiments with a precursor of our architecture (Bengio et al., 1995) have shown that global training significanty improves the system accuracy. We first trained the isolated character recognizer to provide *good recognition* on a database of isolated characters. The global training algorithm then taught the isolated character recognizer to return scores that provide *good recognition and good segmentation* on a database of character strings, taking into account the interactions of these local character scores at the level of a sentence or a word.

There is another reason for implementing global training for our modular architecture. We can indeed replace any transformer by a compatible transformer based on new heuristics. This capability has little value unless we have an automated way to get all the modules in tune.

The input of our transduction cascade is the image of a document. The output is a graph scoring all probable interpretations of our document, namely the interpretation graph. We want to train all parameters of the system by presenting examples composed of (a) the image of a document (e.g. the check image), and (b) the true interpretation of this document (e.g. the check amount).

As explained in section 3.2, we can compute a log-probability for each interpretation of a document. We can adapt the parameters w of the system by maximizing the log-probability of the correct interpretations I_k of the example documents D_k. This objective function is equivalent to the criterions

used in (Denker and Burges, 1995) and (Bengio et al., 1995).

$$E(w) = \log \prod_{k=1}^{K} P_w(I_k|D_k) = \sum_{k=1}^{K} \log P_w(I_k|D_k)$$

where the sum is over the set of K training examples.

The simplest algorithm for performing this maximization is the online gradient descent. We may also use online variant of the conjugate gradient algorithm. Global training requires large databases; online algorithms usually outperform batch algorithms.

The regular structure of the transduction cascade simplifies the computation of the gradient. The transduction cascade indeed looks like a multi-layer perceptron. The successive layers of a multi-layer network transform vectors of states into new vectors of states. The successive stages of a transduction cascade tranform weighted graphs into new weighted graphs.

There is an efficient back-propagation algorithm for such multi-stage systems (Bottou and Gallinari, 1991). For each example (I_k, D_k), we can compute the gradient of the loss function (i.e. in our case the loss is the negative log-likelihood $-\log P(I_k|D_k)$) in a single backward pass over the stages of the system. We can then use these gradient for adapting the system parameters.

The back-propagation algorithm consists in three steps (cf. figure 8):

a) We compute the constrained interpretation graph by selecting the paths of the interpretation graph that comply with the correct interpretation of our current example.

b) We compute the gradient of the objective function with respect to the scores of the interpretation graph by calling the forward-backward algorithm (Rabiner and Juang, 1986) on the constrained interpretation graph and on the unconstrained interpretation graph.

c) We loop backwards over all stages of the system. Given the gradient of the loss with respect given the scores of the output graph of each stage, we compute (a) the gradient of the loss with respect to the transformer trainable parameters, and (b) the gradient of the loss with respect to the scores of the input graph.

The implementation of this back-propagation algorithm again relies on the transformer objects. During the generalized transduction operation, the transformer records information relative to the computation of each individual score in the composed graph. An additional method `bprop` takes advantage of this information to perform perform step *(c)* of the above algorithm.

6 Implementation

A complete description of the HCAR50 system (Haffner et al., 1996) is clearly out of the scope of this paper. We give however a brief overview of this system as a running proof that the advocated architecture can provide breakthrough in automatic document analysis.

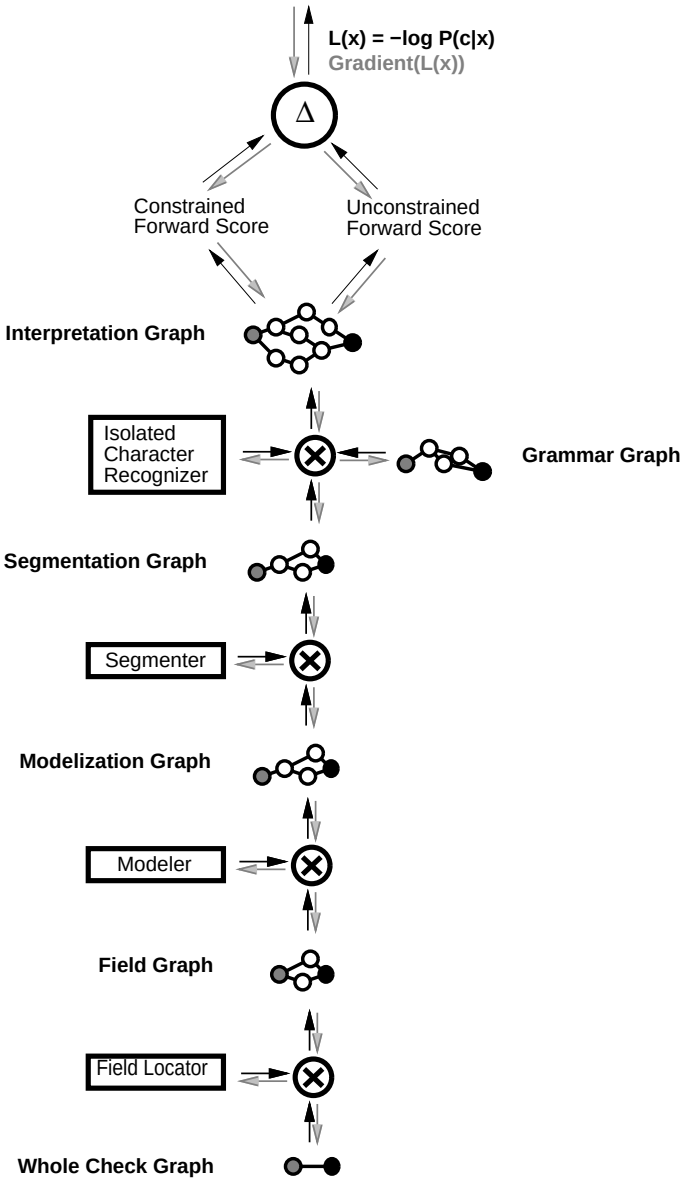

Figure 8: *Discriminant training is performed by maximizing the probability of the correct amount given the image. This probability is easily computed in the interpretation graph by computing the log-added score of all paths compatible with the correct amount and normalizing it with the log-added score of all paths of the graph. We can then backpropagate the gradient of this probability in all parts of the system and adapt parameters everywhere.*

The Holmdel Courtesy Amount Reader (HCAR) is a software system for reading an amount out of check images. This system was initially developed by AT&T Bell Laboratories to fulfill the needs of NCR customers. Performance measurements were independently carried out by NCR. The HCAR development team is now with Lucent Bell Labs Innovation.

The first serious system (HCAR30) relied on a conventional design. It used to read about 30% of the checks with less than 1% error. A systematic trial and error strategy has improved this performance level to 40% by the end of 1994. The resulting system (HCAR40) however is very complex and its performance was locked in a local maximum.

The redesign of HCAR as a transduction cascade was decided during the summer 1995. There was also an urgent need for a better system within a short delay. The HCAR50 system is an hybrid answer to these two concerns.

Although HCAR50 still relies on slightly modified versions of the old amount location and modelization modules, the rest of the recognition engine is completely new.

- The old machinery provides fields representing machine printed amounts, decimal handwritten amount, dollar part of fractional amounts, or cent part of handwritten fractional amounts. These fields are packaged in an initial graph (see figure 1) and fed to the segmentation module.
- The segmentation module runs the HCAR40 cut detection software and produces the segmentation graph (see figure 4).
- This graph is then composed (see figure 5) with a specific grammar graph for each field type. The isolated character recognizer (Lenet5) is described in (Le Cun et al., 1995).
- The resulting interpretation graph is then analyzed with the Viterbi algorithm (see figure 6).

The early modules of HCAR40 used to perform extensive image cleanup (i.e. box removal, underline removal). These early cleanups caused many high scoring errors but were also required by the old segmentation and recognition modules. The considerable flexibility of the HCAR50 transduction cascade allowed us to disable most early cleanups and thus *improve the recognition rate* to about 50% accepts for less than 1% errors.

The graph machinery expands to 25000 out of 200000 lines of C code. Less than 5% of the check processing time is spent in the generalized graph composition subroutine.

During comparative tests carried out by NCR, this system outperformed all other systems. NCR found this system accurate and reliable enough for *actual installation in check processing centers*. HCAR50 may process one millions checks per day before the end of 1996.

7 Conclusion

We have presented a modular architecture for document analysis.

This architecture departs significantly from previous work in Optical Character Recognition or Automatic Speech Recognition:

- We describe all hypotheses on the document using weighted graphs. This representation provides an unified way to select the best hypothesis using global criteria. All stages of the system are built using the same algorithmic structure.
- The global training algorithm simultaneously adjust all parameters of the system on the basis of examples composed of a document image and its interpretation.
- We rely on generalized transductions to achieve low level tasks. Graphs often deal with images rather than discrete symbolic information. Each arc of the graph carries a complex data structure instead of a symbol index.
- We assume no definite probabilistic interpretation to the scores of the arcs. We can however compute conditional probabilities for each hypothesis represented by the graphs.

Finally, a successful check reading application has demonstrated the soundness of this architecture. In less than twelve months, we have been able to significantly improve the accuracy of a real-life state-of-the-art document analysis software.

Acknowledgements

During this work, Leon Bottou was supported by the Adaptive Information Retrieval Systems (AIRS), formerly with AT&T Bell Laboratories and now in with Lucent Technology Bell Labs Innovation.

The HCAR50 team (Jane Bromley, Chris Burges, Troy Cauble, Patrick Haffner, Craig Nohl, Mike Stanton, Charles Stenard and Pascal Vincent) bravely gave us the opportunity to apply our architecture to real world problems (Haffner et al., 1996). The maturation of this paper is a fruit of our collaboration.

References

Bengio, Y., LeCun, Y., C., C. N., and Burges, C. (1995). LeRec: A NN/HMM Hybrid for On-Line Handwriting Recognition. *Neural Computation*, 7(5).

Bottou, L. and Gallinari, P. (1991). A Framework for the Cooperation of Learning Algorithms. In Touretzky, D. and Lippmann, R., editors, *Advances in Neural Information Processing Systems*, volume 3, Denver. Morgan Kaufmann.

Bourlard, H. and Morgan, N. (1990). A continuous speech recognition system embedding MLP into HMM. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*, volume 2, pages 186–187, Denver. Morgan-Kaufmann.

Bourlard, H. and Wellekens, C. (1989). Links between Markov models and multilayer perceptrons. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*, volume 1, pages 186–187, Denver. Morgan-Kaufmann.

Bridle, J. (1989). Probabilistic interpretation of feedforward classification networks outputs, with relationship to statistical pattern recognition. In Fogelman, F., Herault, J., and Burnod, Y., editors, *Neurocomputing, Algorithms, Architectures and Applications*, Les Arcs, France. Springer.

Denker, J. S. and Burges, C. J. (1995). Image Segmentation and Recognition. In *The Mathematics of Induction*. Addison Wesley.

Devijver, P. (1985). Baum's forward-backward algorithm revisited. *Patern Recognition Letters*, 3:369–373.

Guyon, I., Schenkel, M., and Denker, J. (1996). Overview and synthesis of on-line cursive handwriting recognition techniques. In Wang, P. S. P. and H., B., editors, *Handbook on Optical Character Recognition and Document Image Analysis*. World Scientific.

Haffner, P., Bottou, L., Bromley, J., Burges, C. J. C., Cauble, T., Le Cun, Y., Nohl, C., Stanton, M., Stenard, C., and Vincent, P. (1996). The HCAR50 Check Amount Reading System. Technical report, Lucent Technologies, Bell Labs Innovation. Forthcoming publication.

Haffner, P., Franzini, M., and Waibel, A. (1991). Integrating time-alignment and neural networks for high performance continuous speech recognition. In *Proc. of ICASSP 91*. IEEE.

Le Cun, Y., Jackel, L. D., Bottou, L., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Muller, U. A., Sackinger, E., Simard, P., and Vapnik, V. (1995). Learning Algorithms For Classification: A Comparison On Handwritten Digit Recognition. In Oh, J. H., Kwon, C., and Cho, S., editors, *Neural Networks: The Statistical Mechanics Perspective*, pages 261–276. World Scientific.

Pereira, F., Riley, M., and Sproat, R. (1994). Weighted rational transductions and their application to human language processing. In *ARPA Natural Language Processing workshop*.

Rabiner, L. R. and Juang, B. H. (1986). An introduction to hidden Markov models. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, 3 (1).