Conditional Random Fields for the Classification of Digital Logic Circuits

July 2006

Jason Fennell
Harvey Mudd College
jfennell@hmc.edu

Max Pflueger
Harvey Mudd College
mpflueger@hmc.edu

Abstract

Sketches are an important and natural means of communication that historically have been of little use as a way of interacting with computers. A sketch based interface to computers would increase both ease of use and usefulness of computers for design. We have tackled the stroke-level recognition piece of this problem by constructing a Conditional Random Field to take advantage of the context in which strokes are drawn to do recognition. We have successfully used this tool to distinguish between wires in gates in sketches of digital logic circuits, which are more complex than the domains considered in previous work.

1 Introduction

Sketches are a natural medium of communication used regularly in engineering, computer science, and many other fields. However, they are severely limited as tools for anything but discussion by the inability of computers to recognize and analyze them. The ease of use of computers for design, and many other processes, could be improved if sketches became as natural of a way of communicating with computers as they are for communicating with other humans.

Our research focuses on the problem of recognizing a freely drawn sketch with the eventual goal of having a natural computer interface through sketches. The problem presents us with a host of challenges. First and foremost is compensating for the variability and sloppiness inherent in sketches. In order for software to accurately recognize sketches, it must have both the specificity necessary to distinguish individual elements of a sketch and the generality to recognize different styles of drawing a given object. Second is the problem of segmenting, or logically grouping individual pen strokes (strokes are considered to be a continuous motion of the pen across the tablet surface) into a larger object. If the problem of segmentation were solved, sketch recognition would be a much easier problem, since the number of individual elements to be recognized would shrink and there would be more information contained in each of these elements. As it stands, it is difficult to determine exactly how to partition the strokes of a sketch into the objects the user intends to represent, so we must assume that any real world task would have to be performed with unsegmented data.

The project that our group is focusing on is the creation of a toolchain that can take as input a sketch of a digital circuit and output a fully recognized version of that circuit in a design program. We restrict ourselves to recognizing digital circuits to limit the complexity of our problem. Additionally, to give the user a natural interface we do not contstrain the them in any way during the sketching process. Adding interface requirements, such as having users indicate segmentation of their sketch for us, would make our task significantly easier. Our primary goal is a natural interface, however, making such requirements unacceptable.

This paper focuses on the problem of recognizing individual strokes in a sketch, a low-level step in our toolchain. The only preprocessing is fragmentation of the sketch into lines and arc-segments to give us more strokes to train on, and simpler strokes to recognize. For more information on the fragmentation, see the report by Aaron Wolin [Wolin]. Once we have finished recognition, another program uses the recognized strokes to segment our drawing into logical parts by grouping sets of strokes that make up gates or wires together. This segmenter goes back and forth with an object-level recognizer that attempts to recognize the objects the segmenter found. For more information on segmentation and recognition, see the report by Devin Smith [Smith]. Once this final recognition step is complete, the circuit undergoes analysis and is passed to a design program, fully understood by the computer.

The goal of our stroke-level recognition is to inform a segmentation tool and improve its accuracy. The sparsity of information contained within individual strokes presents the major challenge for this type of recogntion. The stroke making up the back of an AND gate can look exactly the same as a small piece of wire. However, the context in which a stroke finds itself can provide significantly more information about the nature of a stroke. Context is a stroke's relationship to the strokes near it in time and space, or to all of the other strokes in the input. Context is also they key to our solution of the stroke-level recognition problem. A straight line that is connected on both ends to an arc reveals itself as the back of an AND gate, whereas another straight line that looks exactly the same is recognized as a wire when it has two different straight lines touching either end.

It is with the importance of context in mind that we have constructed a Conditional Random Field, or CRF, to perform stroke-level recognition. A CRF uses the context of a stroke to classify the stroke. The CRF consists of two parts: a graph that quantifies the probabilistic dependencies between the elements of the CRF input (strokes in our case), and an associated set of potential functions that measure the compatibility between labels and strokes over cliques in the graph. The CRF incorporates contextual information through the potential functions, which measure compatability of individual labels and strokes as well as compatibility between groups of mutually dependent labels and strokes. For instance, a potential function could quantify if a pair of neighboring strokes were compatible with the first being labeled wire and the second being labeled gate. Once the graph and potentials are constructed, the CRF can be given to an inference algorithm that calculates the probability of each possible label on every input. This probabilities are used to select the most likely set of labels for the input.

Our goal for the CRF is to correctly classify strokes in digital logic circuits to provide a major piece of a sketch recognition toolchain. We have extended two-label classification into the domain of digital circuits—which is a more complex domain than considered in previous research—with a high level of accuracy. We have also made headway with multi-label classification in the same domain. With this in mind, we begin with a closer look at CRF's in general before delving into our implementation and the results it gave us.

2 Background

Szummer and Qi offer an excellent explanation of Conditional Random Fields in their paper "Contextual Recognition of Hand-drawn Diagrams with Conditional Random Fields" [Szummer and Qi 2004]. We summarize their exposition below.

A Conditional Random Field is an undirected graphical model that seeks to find the joint probability distribution $P(\boldsymbol{y}|\boldsymbol{x})$ where $\boldsymbol{x}$ is a set of input data and $\boldsymbol{y}$ is a set of labels for that input data. The actual CRF consists of a graph $G = (V, E)$ and an associated set of potential functions. Each node in V corresponds to an element of the input $\boldsymbol{x}$, and each edge in E quantifies a probabilistic dependence between nodes. G thus obeys the Markov property, which means that the probability that a node has a label is dependent only on neighbors. Formally, $P(y_i|\boldsymbol{x}, \boldsymbol{y}_{V-i}) = P(y_i|\boldsymbol{x}, \boldsymbol{y}_{\mathcal{N}_i})$, where $\boldsymbol{y}_{V-i}$ is all of the nodes other than i in the graph, and $\boldsymbol{y}_{\mathcal{N}_i}$ is the neighborhood of i [Szummer and Qi 2004].

There are two types of potential functions: site potentials that are associated with each node, and interaction potentials that are associated with each pair of nodes. In theory, potentials could exist over cliques larger than 2, but this makes the CRF prohibitively complex. The intuitive understanding of a site potential function is that it measures the compatibility between a label and a node. Similarly, an interaction potential measures the compatability between a pair of labels and a pair of adjacent nodes. Both types of potentials measure compatability by linearly combining parameters with a set of feature functions and passing the result through a non-linearity.

A feature function takes as input some subset of the input data and returns a numerical representation of a charactaristic of that data. As with potentials, feature functions come in two varieties, site features and interaction features. Site features are functions of a single node, and interaction features are a function of an adjacent pair of nodes. An important property of CRF feature functions is that while they are functions measuring compatibility between up to two labels and nodes, they have access to all of the input data $\boldsymbol{x}$, not just the data for the nodes in question.

Parameters are associated with every feature function, and quantify how important each feature is to the classification of a node or pair of nodes with a certain label. If l is the number of labels the CRF has to decide between, each site feature has l parameters associated with it and each interaction feature has l^2 parameters associated with it, one for every label or pair of labels that can be applied. Putting this all together, potential functions take the form:

$$\begin{aligned} \text{Site:} \qquad \Phi(y_i, \boldsymbol{x}; \boldsymbol{\theta}) &= e^{\boldsymbol{\theta}(y_i)\cdot \boldsymbol{g}_i(\boldsymbol{x})} \\ \text{Interaction:} \qquad \Omega(y_i, y_j, \boldsymbol{x}; \boldsymbol{\theta}) &= e^{\boldsymbol{\theta}(y_i, y_j)\cdot \boldsymbol{f}_{ij}(\boldsymbol{x})} \end{aligned}$$

Where $\boldsymbol{\theta} = [\boldsymbol{w}\ \boldsymbol{v}]$ and $\boldsymbol{w}$ is the set of parameters for site features and $\boldsymbol{v}$ is the set of parameters for interaction features. Also, $\boldsymbol{g}_i$ is the vector of site features for node i and $\boldsymbol{f}_{ij}$ is the vector of interaction features between nodes i and j. The probability distribution $P(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$ that we are interested in can be factored into a product of these potential functions using the graph G. This gives the following probability distribution that the CRF models:

$$P(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{i\in V} \Phi_i(y_i,\boldsymbol{x};\boldsymbol{\theta}) \prod_{(i,j)\in E} \Omega_{i,j}(y_i,y_j,\boldsymbol{x};\boldsymbol{\theta})$$

$$Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{y}} \left(\prod_{i\in V} \Phi_i(y_i,\boldsymbol{x};\boldsymbol{\theta}) \prod_{(i,j)\in E} \Omega_{i,j}(y_i,y_j,\boldsymbol{x};\boldsymbol{\theta}) \right)$$

The probability above can be calculated using an inference algorithm which will calculate $P(y_i|\boldsymbol{x},\boldsymbol{\theta})$ which can then be used to calculate $P(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta})$.

2.1 Training

Training the CRF consists of finding the set of parameters (a point in parameter-space) that maximizes the log of the likelihood $P(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta})$, which takes the form

$$\mathcal{L} = \sum_{i\in V} \boldsymbol{w}(y_i)\cdot\boldsymbol{g}_i(\boldsymbol{x}) + \sum_{(i,j)\in E} \boldsymbol{v}(y_i,y_j)\cdot\boldsymbol{f}_{ij}(\boldsymbol{x}) - \log Z(\boldsymbol{\theta}) - \frac{||\boldsymbol{\theta}||^2}{2\sigma^2}.$$

After initializing $\boldsymbol{\theta}$ with random Gaussian priors, we train using Nonlinear Conjugate Gradient Descent by following the gradient of $\mathcal{L}$ toward the critical point of our surface. Other work, such as [Szummer and Qi 2004] has also had success with the quasi-Newtonian BFGS method. The gradient of $\mathcal{L}$ takes two different forms depending on whether differentiation is done with respect to site or interaction parameters.

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}(y_i)} = \sum_{i\in V} \boldsymbol{g}_i(\boldsymbol{x}) - E\left(\sum_{i\in V} \boldsymbol{g}_i(\boldsymbol{x})\right) - \frac{\boldsymbol{w}(y_i)}{\sigma^2}$$

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{v}(y_i,y_j)} = \sum_{(i,j)\in E} \boldsymbol{f}_{ij}(\boldsymbol{x}) - E\left(\sum_{(i,j)\in E} \boldsymbol{f}_{ij}(\boldsymbol{x})\right) - \frac{\boldsymbol{v}(y_i,y_j)}{\sigma^2}$$

Where E denotes the expected value of a function with respect to the current model distribution $P(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta})$. Note that this is simply shorthand, as $E\left(\sum_{i\in V} \boldsymbol{g}_i(\boldsymbol{x})\right) = \sum_{\boldsymbol{y}}\left(\sum_{i\in V} \boldsymbol{g}_i(\boldsymbol{x})P(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta})\right)$. There is a similar formula for interaction features.

2.2 Similar Problems

Earlier research has used CRFs and variants on CRFs to classify hand drawn diagrams, however, we are not aware of anyone who has used it for such a complex set of possible labels. Previous research by Szummer and others has shown great success in using CRF's to classify strokes in organizational charts involving connectors (lines) and containers (simple geometric shapes, usually rectangles) [Szummer and Qi 2004], [Szummer 2005], [Cowans and Szummer 2005]. These papers, especially [Szummer and Qi 2004], form the jumping off point where our own project began.

For a more in depth treatment of CRF's, refer to [Wallach 2004], [Sutton and McCallum 2006]

3 Approach

Our purpose in implementing a CRF is to correctly label the individual strokes in a digital logic circuit sketch to allow for further recognition of the sketch. With this in mind we begin our discussion of the specific implementation of CRF that we have created, and the important aspects of the design.

3.1 Fragmenter

Before the CRF begins to work on our data, the strokes have to be fragmented. The purpose of the fragmenter is to divide strokes up into lines and arcs. Additionally, and perhaps more importantly, in the occasional case when one stroke will be used to draw two distinct elements of the diagram, it is the job of the fragmenter to divide that stroke between those two elements so they can be properly recognized by the CRF. The quality of the results obtained from our CRF is highly dependent on the quality of the fragmentation of the data. Further discussion of this is in the results section.

3.2 General CRF

Our implementation of a CRF allows for a completely general graph to encode the probabilistic dependencies of the input. The graph is constructed by making a node for every stroke in the input sketch, and placing edges between nodes that differ by less than a temporal or spacial theshold. These thresholds are currently rather arbitrary (about 2 seconds and 1 centimeter respectively). Hopefully later research will be able to fully optimize these values.

There are a number of simplifying assumptions that can be made about the structure of a CRF's graph to lessen computational complexity. CRF implementations often assume that the graph will take the form of a linear chain, or a two-dimensional lattice. However, neither of these limited models can accuratly represent the interactions of a hand-drawn sketch, and hence these assumptions are unacceptable for our application.

3.3 Multi-Label

We designed our CRF to handle an arbitrary number of labels. A multi-label CRF is very similar to a two label CRF mathematically, but computationally there are some notable differences. First, the number of parameters grows as $O(l^2)$ where l is the number of possible labels. The number of parameters grows this way because the interaction features on an edge must be calculated for every possible label combination, of which there are l^2.

An individual CRF is represented by its parameter space. The parameter space comes in two parts, the site parameters and the interaction parameters. If the number of labels is l and the number of site feature functions is s, the number of site parameters is $l \times s$ because there is a parameter for each site feature function associated with each label. If the number of interaction feature functions is i, then the number of interaction parameters is $l^2 \times i$ because there is a parameter for each interaction feature combined with a pair of labels.

For a system with m possible states there are $m - 1$ decision boundaries, therefore there are $l - 1$ decision boundaries on nodes for each feature function and $l^2 - 1$ decision boundaries on edges for each feature function. The number of parameters must match the number of decision boundaries, otherwise you have redundant parameters that could lead to wandering parameter values. To avoid this, we zero out the set of site parameters for the first label (label 0) and the set of interaction parameters for labels (0,0).

3.4 Feature functions

One of the most important aspects of the performance of a CRF is the selection and design of feature functions. As mentioned previously, feature functions can either be site features of nodes or interaction features of a pair of adjacent nodes. A typical site feature may compute something such as the length of a stroke, or a stroke's degree of curvature. A typical interaction feature may compute something such as how close the ends of two strokes are to each other or whether or not two strokes cross each other.

Unfortunately, to get good results it is not as simple as just outputting values such as length or curvature. The CRF behaves in a linear fashion on the feature functions, but the relationships between these functions and the useful data about the sketch is almost always nonlinear. For example, consider a feature function that measures how much a stroke curves across its length. If a particular stroke does a half-turn, then it may be the front of an AND gate. However, if that stroke turns more it is certainly not more of an AND gate than before, and a full turn is not twice as much AND gate as a half-turn. In order to deal with this we broke up features, such as curvature, into groups of features, each associated with a certain region of the output of the function in question. For example, we might have one function that indicates a very small curvature, another that indicates a medium curvature, and yet another that indicates a large curvature. Breaking up scalar valued feature functions in this way allows us to have more complex recognition behavior across the range of the function. The decisions about where to break up the scalar valued features were made based upon preliminary observations of the data. The output of these new feature functions is a value in the range of 1 to -1 to represent true or false. Most of the time this value is very close to 1 or -1, but for input near the boundary of the function we used a smooth transition between 1 and -1.

If scalar valued feature functions are to be included, they must be normalized so they do not exceed a reasonable range, in our case something like a maximum absolute value of 10 to 15. By breaking up the scalar valued feature functions we get the added benefit of fixing the range of our feature functions. Later in the algorithm we compute the exponential of the feature functions. This means that, if the values from feature functions become too large, the exponential will very quickly exceed the range of a double, causing the program to fail.

3.5 Inference

Exact algorithms for inference on graphs with loops are intractable for even very small graphs. To perform inference on our CRF we used the loopy belief propagation algorithm, as implemented by Kevin Murphy in his CRF Toolbox [?]. Loopy BP is an approximate inference algorithm adapted for graphs with loops from the standard belief propagation algorithm. It is not guaranteed to converge, and occasionally causes problems during execution.

3.6 Training

In this section we outline the salient details of the training methods we used on our CRF. First we mention our use of the Conjugate Gradient method to do training.

We then explain one of the problems that we have had in training and our implementation of ShakeIt! to fix that problem. Next is an explanation of the regularization of our gradient and log-liklihood functions, and finally is an explanation of the method of preconditioning which we used to greatly increase how well our training worked.

By adjusting the parameters $\boldsymbol{\theta}$, training seeks to minimize $-\mathcal{L}$ the negative log-likelihood of our CRF, so that the set of correct labels for the training dataset are the most likely. We used Non-linear Conjugate Gradient Descent for training our CRF. This is an extension of the simple gradient descent algorithm, so the basic concept behind the algorithm is to use the gradient of a function to step toward a critical point of the surface in question. For more information about Non-linear Conjugate Gradient Descent, see "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain" by Jonathan Shewchuk [Shewchuk 1994].

Note that the gradient and log-liklihood functions, and thus training, need $\log Z(\boldsymbol{\theta})$ to be calculated. This value is in turn found as part of the Loopy BP calculation. Occasionally in the course of training a negative or otherwise obviously wrong value of the $\mathcal{L}$ function is returned. We suspect that these values are due to Loopy BP not converging and returning an incorrect value for the $\log Z(\boldsymbol{\theta})$ term. However, regardless of the source of these errors, uncorrected they will cause the training to fail. Hence, we added a piece of code that looked for such erroneous values, and fixed them by slightly randomizing the current set of parameters and recalculating until an acceptable value was arrived upon. We call this the ShakeIt! code.

Adding regularization factors to our log-likelihood and gradient functions improved our training significantly. This is the $\frac{||\boldsymbol{\theta}||^2}{2\sigma^2}$ term in $\mathcal{L}$, the $\frac{\boldsymbol{w}(y_i)}{\sigma^2}$ term in $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}(y_i)}$, and the $\frac{\boldsymbol{v}(y_i,y_j)}{\sigma^2}$ term in $\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}(y_i,y_j)}$. These terms serves two purposes: without it the values of the parameters have the tendency to explode and cause numerical overflow errors, and it also prevents the training algorithm from overfitting. We have set σ^2 to a hard value for our implementation, but others recommend more complex methods for finding its value [Szummer 2005].

Because of the difficulty in getting the CRF parameters to converge to a good value in a single run, we use preconditioning by training first on smaller sets of data to get parameters that are close to the optimal parameters. We will also sometimes run training on smaller datasets between training runs on full datasets to perturb the parameters slightly in the hope that the next full training set will converge better. The imperfect nature of nonlinear conjugate gradient descent and the possibility that Loopy BP will not converge mean that the CRF parameters do not always converge all the way, in fact, usually they stop well before their optimal point. The process running the same set of CRF parameters through successive rounds of training on differing data sets has the effect of defeating these imperfections by starting closer to the optimal point at the beginning of each round of training.

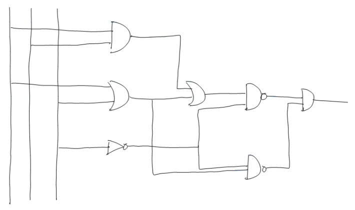
Figure 1: First example subjects were asked to draw.

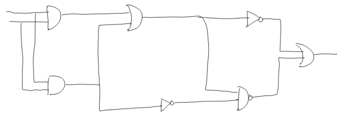
Figure 2: Second example subjects were asked to draw.

4 Results and Discussion

To obtain data on which to run CRF training and classification, we surveyed other computer science researchers working nearby us. We asked the for their name, "How much experience do you have with digital circuits?", and "How much have you used a tablet computer before?". Generally our subjects had a passing familiarity with digital circuits and little to no experience with tablet computers. After the questions, the subjects were asked to draw copies of the circuits in Figure 1 and Figure 2, then to make a circuit of their own choosing, restricting their gate choices to AND, OR, NOT, and NAND. This restriction is so we could run 5-label (as opposed to 8-label) testing on this data. The final circuit did not have to have any formal meaning other than being correct (only one input to a NOT gate, for example). Subjects were told that they were not going to be judged on performance.

We gathered 3 circuits each from 17 different subjects. We trained on a subset of the circuit sketches, generally using 17 samples for full training and 4 or fewer samples for conditioning. Classification was run on all of the collected data, but recognition rates were only recorded for those circuits that were not used in training. We also excluded the third sketch of one of our users because it is apparent that they were trying to draw a smiley face with circuit components rather than trying to draw an actual circuit.

4.1 Fragmenter

The recognition performance of the CRF is highly dependent on the how the sketch is fragmented. An under-fragmented sketch, or a sketch that is not fragmented at key points may be difficult for the CRF to recognize because the strokes may not come in predictable shapes, or, even worse, one stroke may be used in two different elements. Going the other direction, and over-fragmented sketch will be difficult to recognized because the strokes may be decimated down to the point of not being individually significant, and therefore giving the CRF very little to work with in terms of evidence per node. Fragmenting is a difficult problem in and of itself, and we will not be discussing the precise details of that project here, but one can reference the paper by Aaron Wolin [Wolin] to see a discussion of the fragmenter that we used.

In our own experience, changes in performance due to changes in the fragmenter have been dramatic. During development, the change that took our CRF from very mediocre (approximately chance) 2-label recognition to good (finding most gates) 2-label recognition was an updated fragmenter. Future researchers should keep this in mind when attempting to do sketch recognition.

As an extension of this principle, it is logical to assume that changes in the performance of the CRF will have similarly large effects on the performance of other algorithms later in the tool chain.

4.2 2-label classification

We had three major training runs on our data using two label classification, all three of which followed the general procedure outlined at the start of this section. On the first run we chose all of the sketches of Figure 1 to train on, and four sketches of Figure 2 to condition on. On our second run we trained on the same datasets, but noticed that the standard deviation of the parameters in the output of the first training run was about .3. We wished to see if changing our σ values to reflect this would affect recognition, we modified the way our gaussian priors were selected so that the standard deviation was $\frac{1}{3}$ instead of 1, and we also changed the σ^2 in our regularization term in $\mathcal{L}$ and gradient so that $\sigma = \frac{1}{3}$. Our third training run used the original σ values from the first run. We chose our 17 training examples randomly from the sketches of Figures 1 and 2, and chose our 4 conditioning examples similarly. Finally, our fourth run... Running classification as outlined at the start of the section gave us the results seen in Table 1.

Train on:	Mean Recognition Rate
Fig 1	$93 \pm 2\%$
Fig 1, $\sigma = \frac{1}{3}$	$93 \pm 2\%$
Rand. subset of Fig 1 & 2	$92 \pm 1\%$
Rand. subset all sketches	$96 \pm 1\%$

Table 1: Results

There was no statistically significant change in recognition rates when we changed σ, or opened up the training set to both the first and second sketches. However, the best performance was gleaned from taking a random subset from all of our data, which makes sense as it gave the training algorithm a greater variety of samples to work with.

One qualitiative think we noticed about classification was that features that detected if a two strokes were originally part of the same stroke (before fragmentation) and other features that found if strokes were grouped together in time/space, strongly (parameters bigger than most others) supported the labeling of those strokes with the same label. Another feature function that seemed to be important (large parameter values) was the T-junction checking feature function.

4.3 5-label classification

Our five-label classification tried to correctly label strokes as Wire, AND, OR, NOT, or NAND. Since we asked our survey subjects to restrict themselve to these gates, we were able to use the survey data for this also. We were only able to make one successful training run for five-label classification. The CRF usually crashed during training for five labels, but after enough tries we were able to finish the training process. We think the crashing comes from the large data sets passed to the CRF and the problem that if loopyBP doesn't converge, then the CRF will crash. The training process for the five label cases followed nearly the same format as the two label case. We conditioned on three sketches from the second page of the survey data, trained on the 17 first pages of the survey data, conditioned on three more sketches from the second page of survey data (one of which was conditioned on before), then finished by training again on the same 17 sketches.

Classification following the rules outlined at the start of this section yielded a recognition rate of $39 \pm 2\%$. While this is disappointing for us, this level of recognition is still twice the 20% accuracy that random classification should yeild in the five label case. Also, quite frequently there were gates in the survey input that could not be recognized accurately by humans because of the sloppiness of the drawing, so it it not altogether surprising that the computer is not able to recognize the circuits well. Finally, although this will be a future area of work, our toolchain should be able to function quite well with only the two label classification, as there will be an object level recognizer higher in the toolchain that will focus more on differentiating between gates.

4.4 Performance

The code has been written to move as much work as possible out of gradient and log-likelihood, as they are called most frequently. The creates a heavy cost at the start of a program run, but make the rest of training go much faster. The largest slowdown that we regularly run into is the current implementation of loopy belief propagation. It is currently a C# interface to call a matlab function, which requires all sorts of overhead in the C# code to repackage all of the information for the matlab code. There is a fairly large constant time slowdown to initialize the matlab interface. This also requires anyone who wants to use this package to have matlab on their system. The most obvious performance and code-niceness increase that can currently be performed on the CRF is to reimplement loopy belief propagation in C# and remove the need for the matlab call.

4.5 Parameter explosion

When our CRF fails it is usually due to something we called parameter explosion. This is an error that occurs during training (we don't really have errors during classification), where the training algorithm takes a step along what is nominally the gradient of $\mathcal{L}$, and then the CRF falls apart because all of its potential functions have overflow errors and have Positive Infinity for values. This overflow happens because somehow a bunch of the CRF parameters were inflated to very large values by the first step along the gradient. These sort of overflow errors have been the bane of our existence, as we have a lot of exponential functions in the code, and when the exponent is anything but small we run into overflow very quickly. We think that this is why the regularization helped our CRF performance so much. We however do not know why these parameter explosions occur, though we do know that they cause all the exponential functions to overflow out of doubles when they do occur. This is something to watch for and hopefully fix in the future.

4.6 Discussion

Here we should talk about what we think of these results overall and what they mean. This may get moved to other subsections later.

It seems like the CRF we have developed is capable of very impressive recognition results, at least for the 2-label recognition case. Unfortunately, it is difficult to get the CRF to converge to a good set of parameters in one training session. There is a tendency for the gradient descent process to get stuck after training has continued for a number of iterations. It is difficult for us to say exactly why this happens. We suspect that the gradient function may not be returning a precisely correct value, either due to an algorithm problem in the gradient function, or a convergence problem in the inference algorithm. A solution to this problem would be very desirable, as it would result in a faster and better training process.

5 Related Work

Conditional random fields were originally introduced by John Lafferty et. al. in the the paper "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data" [Lafferty et al. 2001]. A major jumping off point for our work has been the paper by Szummer and Qi, "Contextual Recognition of Hand-drawn Diagrams with Conditional Random Fields" [Szummer and Qi 2004]. The paper provides a detailed approach and much of the theory necessary to apply a general, dynamically generated CRF to sketch recognition. A similar model to conditional random fields known as hidden random fields is used by Szummer in his paper "Learning Diagram Parts with Hidden Random Fields" [Szummer 2005]. This approach is derived from to conditional random fields, and does not seem to show a significant gain in performance over CRFs. In our approach we chose to do only labeling with our CRF, and segment the strokes later. However, not everyone has taken this approach. Cowans and Szummer have created a graphical model that simultaneously does partitioning and labeling [Cowans and Szummer 2005]. Although their approach shows significant promise in accuracy, the computational cost is very high.

6 Future Work

There are many different paths that future research on this subject can take, largely with the hope of improving the overall recognition accuracy of the CRF. The first and simplest way in which classification could be improved is by doing a study of the behavior of our current CRF and its feature functions, and tweaking the large number of arbitrary values in the code. These arbitrary values primarily crop up as thresholds in our binary feature functions that find if a scalar valued feature function falls in a certain region. We would hope to optimize the borders of these regions. Also, the algorithm that creates edges in the CRF's graph uses hard time and space threshold values, which we would also like to study and optimize.

A major aid to any future work with CRFs would be a solid theory of how different types of feature functions effect recognition. As it stands, intuition is the only guide we have when determining what feature functions we should use. This study would involve examining how performance in training and classification changed as the feature functions were altered over a fixed set of data. Individual features would be tried first, and then various groupings of features would be tried, with the goal of finding an optimal combination of feature functions. In the course of this study, there are a number of areas that should be addressed.

- The optimal return values for binary features. Should they return 1 on a positive result and 0 on a negative result, or 1 on a positive result and -1 on a negative result. A value of 1 would always represent the feature being true, but it is unclear what the value should be when the feature is false. A value of zero would mean that an untrue feature would have no effect on the CRF, whereas, -1 would have the exact opposite of true effect on the results. The decision for us was an easy one, since features the regularly return 0 lead to parameter explosion errors in our code. However, if this problem were fixed, it would be useful to know what the optimal false return value is. Also, should there be some combination of these two schemes based on the type of feature function? Should binary features instead replace their two return values with a transfer function between 1/0 (or 1/-1) that indicates the strength of the feature toward one end or the other, or does this simply turn a binary feature into a bounded scalar feature?

- An as yet unsolved problem for us is the decision of where to break up the scalar valued feature functions. The easy solution would be to simply create a very large number of regions, but this would be computationally expensive, and would make training more difficult as there would be a larger number of parameters to properly fit. Our feature functions were broken up with preliminary observations about where logical regions would exist. A method of verifying the usefulness of these regions would be desirable.

- Normalization of feature functions. Without small valued features, numerical overflow errors cause the CRF to fail. It seems like normalization is necessary for the CRF to work. However, is there a way to get the CRF to work without individually normalizing the feature functions, and how do different normalization schema effect classification? Also, what about region-based binary features versus scalar valued features? Do binary features that decide if a scalar valued function is within a given region or not improve upon simple scalar valued feature functions for classification?

- Symmetry of interaction features. Are interaction features between nodes required to be symmetric? Is it enough to have asymmetric feature functions that come in symmetric pairs / groups? For instance, a T-Junction function that detects if the first stroke passed to it is the T and the other stroke is the stem is not symmetric. However, another T-Junction function that detects if the first stroke is the stem and the second is the T would pair with the first function to create a symmetry. Is this sort of pairing even necessary or can we have totally asymmetric feature functions with no problem?s

- Combination feature functions. How does it affect classification to combine multiple active (also being used by the CRF) features together in some way in another feature. Will this allow for more flexible classification (hoping that this would allow two individual features that have a negative effect on a given label to show a positive effect on the label if they were both negative and that meant some positive effect).

The most obvious and useful extension of the CRF would be to extend it to be able to classify on an arbitrary number of labels [Hopefully move to results]

[IMPLEMENTATION SUGGESTION] It would be useful to change the format that we use to store trained CRFs to be tolerant of changes in the feature functions. XML is a possibility, but other formats could work as well. This would allow the program to only have to generate random parameters for new feature functions when they are added, instead of losing all the progress of previous training sessions.

Finally, there are a few other long term possibilities for improving CRF performance. Adding the ability for there to be a different set (and number) of feature functions associated with each label would be somewhat nasty to code, but has a good chance of improving classification. Adding interaction features dealing with cliques bigger than size two is more a matter of curiosity than anything, and probably would be very painful to implement. Also, the CRF should be tried on other sketch domains.

7 Conclusion

All that again, but really fast. Not exactly sure what to say.

Acknowledgments

Thanks to all the E85er's and all of the summer 2006 CS researchers on 2nd floor Sprauge for providing us data. We also thank Martin Szummer for guiding us through some aspects of CRF implementation.

References

Cowans, P. J., and Szummer, M. 2005. A graphical model for simultaneous partitioning and labeling. In *2005 AI & Statistics, Tenth International Workshop on Artificial Intelligence and Statistics.*

Lafferty, J., McCallum, A., and Pereira, F. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, Morgan Kaufmann, San Francisco, CA, 282–289.

Murphy, K. Crf toolbox for matlab. Matlab code, published online at http://www.cs.ubc.ca/~murphyk/Software/CRF/crf.html.

Shewchuk, J. R. 1994. An introduction to the conjugate gradient method without the agonizing pain. This paper is available online at http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf.

Smith, D. Segmenting labeled strokes. Title not final, this paper is unfinished as of the time of this writing.

Sutton, C., and McCallum, A. 2006. An introduction to conditional random fields for relational learning. In *Introduction to Statistical Relational Learning*, L. Getoor and B. Taskar, Eds. MIT Press.

Szummer, M., and Qi, Y. 2004. Contextual recognition of hand-drawn diagrams with conditional random fields. In *2004 9th Intl. Workshop on Frontiers in Handwriting Recognition (IWFHR).*

Szummer, M. 2005. Learning diagram parts with hidden random fields. In *2005 Intl Conf Document Analysis and Recognition (ICDAR).*

Wallach, H. M. 2004. Conditional random fields: An introduction. Tech. Rep. MS-CIS-04-21, Department of Computer and Information Science, University of Pennsylvania.

Wolin, A. Fragmenter. The title for this paper is not final, nor is it done being written at the time of this writing.