CPSC 540 Assignment 5 (due April 5 at midnight)

The assignment instructions are the same as for the previous assignment.

1. Name(s): Dingqing Yang, Junjie Zhu
2. Student ID(s):38800141, 30921167

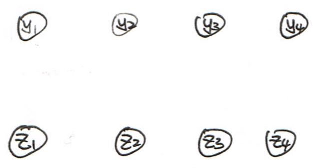
1 Undirected Graphical Models

1.1 Conditional UGM

Consider modeling the dependencies between sets of binary variables x_j and y_j with the following UGM which is a variation on a stacked RBM: Computing univariate marginals in this model will be NP-hard in general, but the graph structure allows efficient block updates by conditioning on suitable subsets of the variables (this could be useful for designing approximate inference methods). For each of the conditioning scenarios below, draw the conditional UGM and informally comment on how expensive it would be to compute univariate marginals (for all variables) in the conditional UGM.

1. Conditioning on all the x and h values.

1.

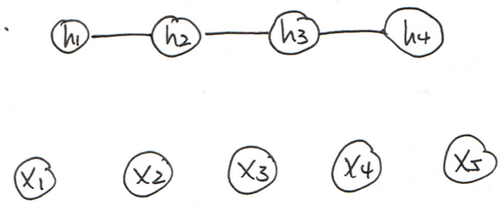


- all variables are independent conditioned on x and h

∴ easy/not expensive to compute univariate marginals.

2. Conditioning on all the z and y values.

2.



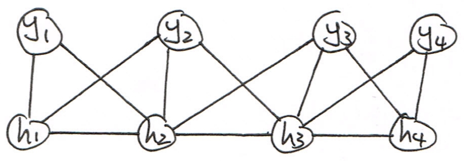
L24, page 15

- x's are independent, and h's form a tree-structured (no cycle)

∴ easy/not expensive to compute univariate marginals.

3. Conditioning on all the x and z values.

3.



- has many cycles in ~~#~~ a UGM

∴ computing univariate marginals will be NP-hard

1.2 Fitting a UGM to PINs

The function *example_UGM.jl* loads a dataset X containing samples of PIN numbers, based on the probabilities from the article at this URL: `http://www.datagenetics.com/blog/september32012`.[1]

This function fits a UGM model to the dataset, where all node/edge parameters are untied and the graph is empty. It then performs decoding/inference/sampling in the fitted model. The decoding is reasonable (it's $x = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$) and the univariate marginals are reasonable (it says the first number is 1 approximately 40% of the time and the last number is 4 approximately 20% of the time), but because it assumes the variables are independent we can see that this is not a very good model:

1. The sampler doesn't tend to generate the decoding ($x = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$) as often as we would expect. Since it happens in more than 1/10 of the training examples, we should be seeing it in more than 1/10 of the samples.
2. Conditioned on the first three numbers being 1 2 3, the probability that the last number is 4 is only around 20%, whereas in the data it's more than 90% in this scenario.

In this question, you'll explore using (non-degenerate UGMs) to try to fix the above issues:

1. Write an equation for $p(x_1, x_2, x_3, x_4)$ in terms of the parameters w being used by the code.

   Answer: $p(x_1, x_2, x_3, x_4) = \frac{1}{Z} \exp\left(\sum_{j=1}^4 x_j w_j\right)$, where $Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \exp\left(\sum_{j=1}^4 x_j w_j\right)$

2. How would the answer to the previous question change (in terms of w and v) if we use `E = [1 2]`?

   Answer: $p(x_1, x_2, x_3, x_4) = \frac{1}{Z} \exp\left(x_1 x_2 v_{1,2} + \sum_{j=1}^4 x_j w_j\right)$,
   where $Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \exp\left(x_1 x_2 v_{1,2} + \sum_{j=1}^4 x_j w_j\right)$

3. Modify the demo to use a chain-structured dependency. Comment on whether this fixes each of the above 2 issues.

   Answer: It improves compared to the independent model, but the chain-structured graph does not solve the problem completely.

   - probability of sampler to generate $x = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$ has improved from 0.003 to 0.048 by generating 1000 samples.
   - $p(x_4 = 4 | x_1 = 1, x_2 = 2, x_3 = 3)$ has improved from 0.19 to 0.531

4. Modify the demo to use a completely-connected graph. Comment on whether this fixes each of the above 2 issues.

   Answer: It improves compared to the chain-structured model, but the complete graph still does not solve the problem completely.

[1] I got the probabilities from reverse-engineered heatmap here: `http://jemore.free.fr/wordpress/?p=73`.

- probability of sampler to generate $x = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$ has improved from 0.048 to 0.072 by generating 1000 samples.
- $p(x_4 = 4 | x_1 = 1, x_2 = 2, x_3 = 3)$ has improved from 0.531 to 0.748

5. What would the effect of higher-order potentials be? What would a disadvantage of higher-order potentials be?

   Answer: Higher order potential can help us capture dependence beyond 2 variables (a pair), but it will introduce a lot more parameters in the model and are more likely to overfit.

If you want to further explore UGMs, there are quite a few Matlab demos on the UGM webpage:
`https://www.cs.ubc.ca/~schmidtm/Software/UGM.html`
that you can go through which cover all sorts of things like approximate inference and CRFs.

2 Bayesian Inference

2.1 Conjugate Priors

Consider a $y \in \{1, 2, 3\}$ following a multinoulli distribution with parameters $\theta = \{\theta_1, \theta_2, \theta_3\}$,

$$y \mid \theta \sim \text{Mult}(\theta_1, \theta_2, \theta_3).$$

We'll assume that θ follows a Dirichlet distribution (the conjugate prior to the multinoulli) with parameters $\alpha = \{\alpha_1, \alpha_2, \alpha_3\}$,

$$\theta \sim \mathcal{D}(\alpha_1, \alpha_2, \alpha_3).$$

Thus we have

$$p(y \mid \theta, \alpha) = p(y \mid \theta) = \theta_1^{I(y=1)} \theta_2^{I(y=2)} \theta_3^{I(y=3)}, \quad p(\theta \mid \alpha) = \frac{\Gamma(\alpha_1 + \alpha_2 + \alpha_3)}{\Gamma(\alpha_1)\Gamma(\alpha_2)\Gamma(\alpha_3)} \theta_1^{\alpha_1 - 1} \theta_2^{\alpha_2 - 1} \theta_3^{\alpha_3 - 1}.$$

Compute the following quantites:

1. The posterior distribution,

$$p(\theta \mid y, \alpha).$$

1. $$p(\theta|y,\alpha) = \frac{p(\theta, y|\alpha)}{p(y|\alpha)} = \frac{p(y|\theta,\alpha)\cdot p(\theta|\alpha)}{p(y|\alpha)}$$

$$\propto \theta_1^{I_{y=1}} \theta_2^{I_{y=2}} \theta_3^{I_{y=3}} \cdot \cancel{p(\alpha)} \cdot \theta_1^{\alpha_1-1} \theta_2^{\alpha_2-1} \theta_3^{\alpha_3-1}$$

$$= \theta_1^{I_{y=1}+\alpha_1-1} \cdot \theta_2^{I_{y=2}+\alpha_2-1} \cdot \theta_3^{I_{y=3}+\alpha_3-1}$$

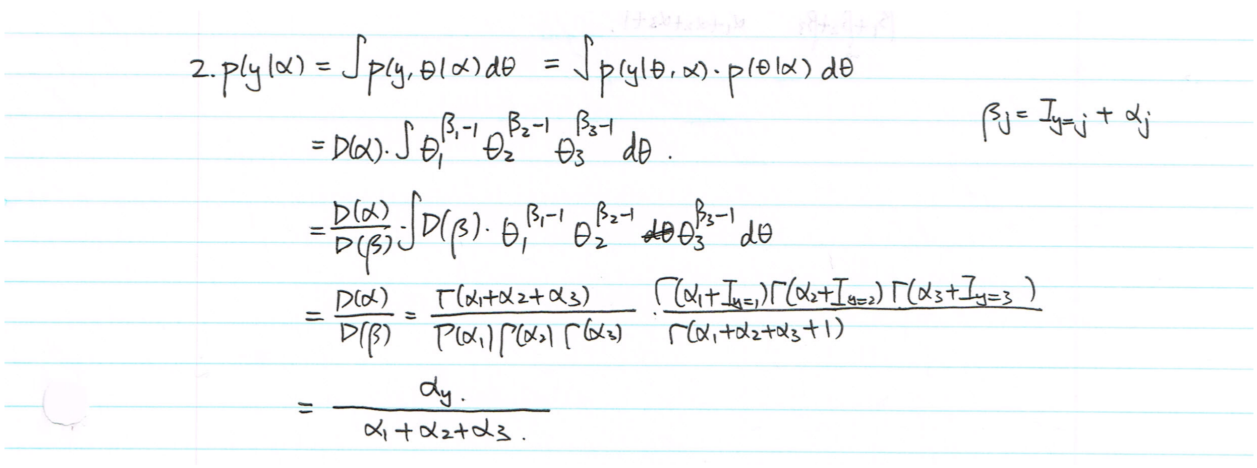
$$\therefore \theta|y,\alpha \sim \mathcal{D}(I_{y=1}+\alpha_1-1,\ I_{y=2}+\alpha_2-1,\ I_{y=3}+\alpha_3-1).$$

$$p(\theta|y,\alpha) = \frac{\Gamma(\alpha_1+\alpha_2+\alpha_3+1)}{\Gamma(\alpha_1+I_{y=1})\Gamma(\alpha_2+I_{y=2})\Gamma(\alpha_3+I_{y=2})} \cdot \theta_1^{I_{y=1}+\alpha_1-1} \theta_2^{I_{y=2}+\alpha_2-1} \theta_3^{I_{y=3}+\alpha_3-1}.$$

$$= \frac{(\alpha_1+\alpha_2+\alpha_3)\Gamma(\alpha_1+\alpha_2+\alpha_3)}{\alpha_y \Gamma(\alpha_1)\Gamma(\alpha_2)\Gamma(\alpha_3)} \cdot \theta_1^{\alpha_1-1} \theta_2^{\alpha_2-1} \theta_3^{\alpha_3-1} \cdot \theta_y .$$

2. The marginal likelihood of y given the hyper-parameters α,

$$p(y \mid \alpha) = \int p(y, \theta \mid \alpha) d\theta,$$



$$2.\ p(y|\alpha) = \int p(y, \theta|\alpha) d\theta = \int p(y|\theta, \alpha) \cdot p(\theta|\alpha)\, d\theta$$

$$\beta_j = I_{y=j} + \alpha_j$$

$$= D(\alpha) \cdot \int \theta_1^{\beta_1 - 1} \theta_2^{\beta_2 - 1} \theta_3^{\beta_3 - 1} d\theta.$$

$$= \frac{D(\alpha)}{D(\beta)} \int D(\beta) \cdot \theta_1^{\beta_1 - 1} \theta_2^{\beta_2 - 1} \cancel{d\theta}\, \theta_3^{\beta_3 - 1} d\theta$$

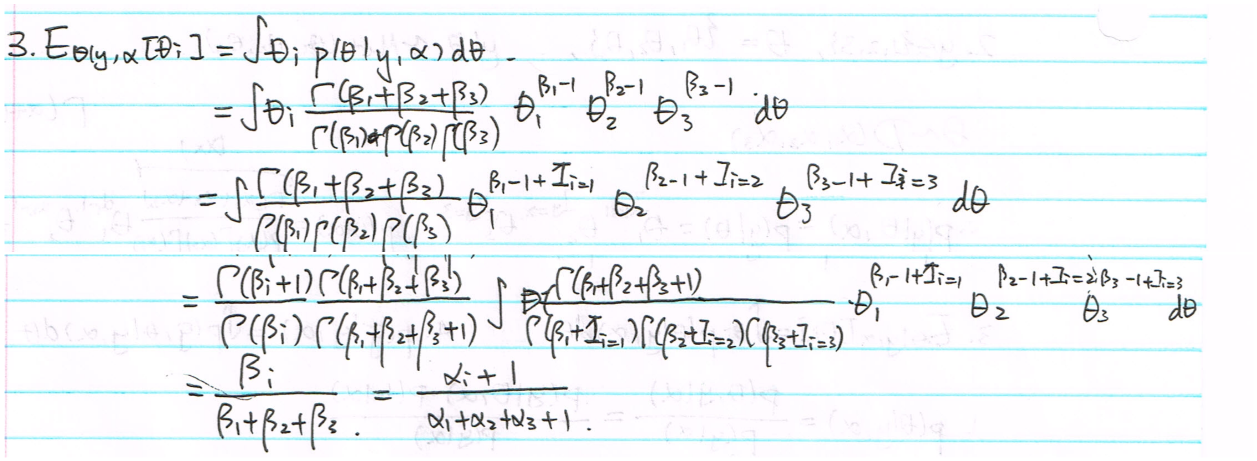
$$= \frac{D(\alpha)}{D(\beta)} = \frac{\Gamma(\alpha_1 + \alpha_2 + \alpha_3)}{\Gamma(\alpha_1)\Gamma(\alpha_2)\Gamma(\alpha_3)} \cdot \frac{\Gamma(\alpha_1 + I_{y=1})\Gamma(\alpha_2 + I_{y=2})\Gamma(\alpha_3 + I_{y=3})}{\Gamma(\alpha_1 + \alpha_2 + \alpha_3 + 1)}$$

$$= \frac{\alpha_y.}{\alpha_1 + \alpha_2 + \alpha_3.}$$

3. The posterior mean estimate for θ,

$$\mathbb{E}_{\theta \mid y, \alpha}[\theta_i] = \int \theta_i p(\theta \mid y, \alpha) d\theta,$$

which (after some manipulation) should not involve any Γ functions.



$$3.\ E_{\theta|y,\alpha}[\theta_i] = \int \theta_i\, p(\theta|y,\alpha)\, d\theta.$$

$$= \int \theta_i \frac{\Gamma(\beta_1 + \beta_2 + \beta_3)}{\Gamma(\beta_1)\Gamma(\beta_2)\Gamma(\beta_3)} \theta_1^{\beta_1 - 1} \theta_2^{\beta_2 - 1} \theta_3^{\beta_3 - 1} d\theta$$

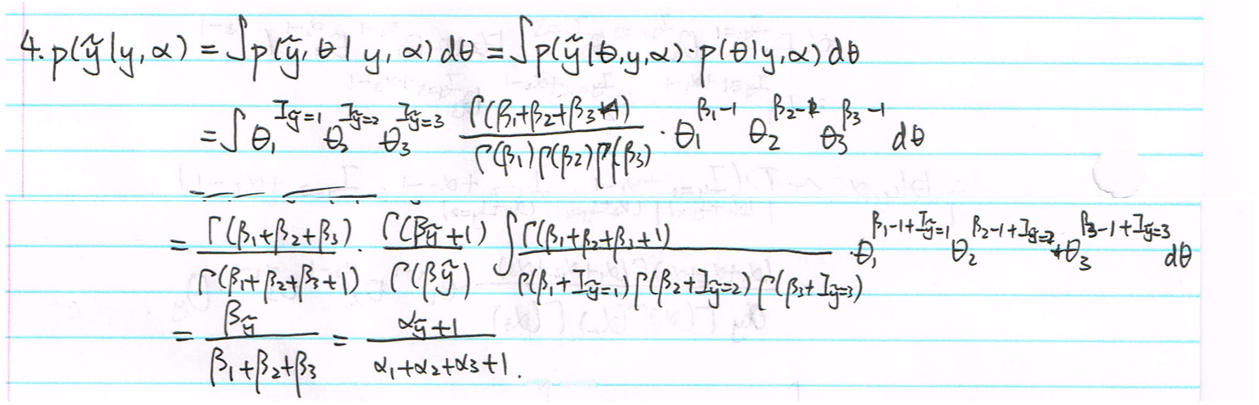
$$= \int \frac{\Gamma(\beta_1 + \beta_2 + \beta_3)}{\Gamma(\beta_1)\Gamma(\beta_2)\Gamma(\beta_3)} \theta_1^{\beta_1 - 1 + I_{i=1}} \theta_2^{\beta_2 - 1 + I_{i=2}} \theta_3^{\beta_3 - 1 + I_{i=3}} d\theta$$

$$= \frac{\Gamma(\beta_i + 1)}{\Gamma(\beta_i)} \frac{\Gamma(\beta_1 + \beta_2 + \beta_3)}{\Gamma(\beta_1 + \beta_2 + \beta_3 + 1)} \int \frac{\Gamma(\beta_1 + \beta_2 + \beta_3 + 1)}{\Gamma(\beta_1 + I_{i=1})\Gamma(\beta_2 + I_{i=2})\Gamma(\beta_3 + I_{i=3})} \cdot \theta_1^{\beta_1 - 1 + I_{i=1}} \theta_2^{\beta_2 - 1 + I_{i=2}} \theta_3^{\beta_3 - 1 + I_{i=3}} d\theta$$

$$= \frac{\beta_i}{\beta_1 + \beta_2 + \beta_3}. = \frac{\alpha_i + 1}{\alpha_1 + \alpha_2 + \alpha_3 + 1}.$$

4. The posterior predictive distribution for a new independent observation $\tilde{y}$ given y,

$$p(\tilde{y} \mid y, \alpha) = \int p(\tilde{y}, \theta \mid y, \alpha) d\theta.$$



$$4.\ p(\tilde{y}\mid y,\alpha) = \int p(\tilde{y},\theta\mid y,\alpha)\,d\theta = \int p(\tilde{y}\mid\theta,y,\alpha)\cdot p(\theta\mid y,\alpha)\,d\theta$$

$$= \int \theta_1^{I_{\tilde{y}=1}}\theta_3^{I_{\tilde{y}=2}}\theta_3^{I_{\tilde{y}=3}}\frac{\Gamma(\beta_1+\beta_2+\beta_3\cancel{+1})}{\Gamma(\beta_1)\Gamma(\beta_2)\Gamma(\beta_3)}\cdot\theta_1^{\beta_1-1}\theta_2^{\beta_2-\cancel{1}}\theta_3^{\beta_3-1}\,d\theta$$

$$= \frac{\Gamma(\beta_1+\beta_2+\beta_3)}{\Gamma(\beta_1+\beta_2+\beta_3+1)}\cdot\frac{\Gamma(\beta_{\tilde{y}}+1)}{\Gamma(\beta_{\tilde{y}})}\int\frac{\Gamma(\beta_1+\beta_2+\beta_3+1)}{\Gamma(\beta_1+I_{\tilde{y}=1})\Gamma(\beta_2+I_{\tilde{y}=2})\Gamma(\beta_3+I_{\tilde{y}=3})}\cdot\theta_1^{\beta_1-1+I_{\tilde{y}=1}}\theta_2^{\beta_2-1+I_{\tilde{y}=2}}\theta_3^{\beta_3-1+I_{\tilde{y}=3}}\,d\theta$$

$$= \frac{\beta_{\tilde{y}}}{\beta_1+\beta_2+\beta_3} = \frac{\alpha_{\tilde{y}}+1}{\alpha_1+\alpha_2+\alpha_3+1}.$$

Hint: You can use $D(\alpha) = \frac{\Gamma(\alpha_1)\Gamma(\alpha_2)\Gamma(\alpha_3)}{\Gamma(\alpha_1+\alpha_2+\alpha_3)}$ to represent the normalizing constant of the prior and $D(\alpha^+)$ to give the normalizing constant of the posterior. You will also need to use that $\Gamma(\alpha+1) = \alpha\Gamma(\alpha)$. For some calculations you may find it a bit cleaner to parameterize the posterior in terms of $\beta_j = I(y=j)+\alpha_j$, and convert back once you have the final result.

3 Very-Short Answer Questions

Give a short and concise 1-sentence answer to the below questions.

1. Why is it appealing to use tree-structured blocks in block-ICM?

   Answer: (L26) It is still easy to update inference for tree-structured block, and it is much faster than updating one node at a time.

2. Why don't we need to compute Z when we use pseudo-likelihood?

   Answer: (L27) The objective is changed so that it does not involve Z.

3. Since deep belief networks are just a big DAG, why is it hard to train them?

   Answer: (L24) It is not tree-structured, so to do anything is NP-hard.

4. What is the advantage of using a CRF, modeling $p(y \mid x)$, rather than treating supervised learning as special case of density estimation (modeling $p(y, x)$).

   Answer: (L28) We get to use complicated features x that makes the task easier.

5. What is an advantage of using residual connections within an RNN?

   Answer: (L31) It gives us option if we don't want to do things sequentially and pick out important details in the middle.

6. What is a setting where we would want to use dilated convolutions?

   Answer: (L31) We need to keep the size of the fileters small, but want to extract informations from cells that are not neighbours.

7. What is the difference between the posterior and posterior predictive distributions?

   Answer: (L32) Posterior is the probability that the parameters are correct, while the posterior predictive distribution is the probability of new data given old data and prior.

8. What is the relationship between Bayes factors and empirical Bayes?

Answer: (L32) Empirical Bayes is to use the data to enhance the prior, while Bayes factors are used to test hypothesis. So using Bayes factors is one method of empirical Bayes

9. What is a hyper-hyper-parameter?

   Answer: (L33) It's a type III maximum likelihood parameters that are the parents of hyper-parameters in DAG.

10. Why do topic models use the Dirichlet prior?

    Answer: (L34) It is multivariate, which can be used to model models with words as features. Also it has nice conjugate prior.

11. In what setting is it unnecessary to include the q function in the Metropolis-Hastings acceptance probability?

    Answer: (L35) If $q(x^t|\hat{x}^t) = q(\hat{x}^t|x^t)$, $\forall x^t, \hat{x}^t$.

4 Literature Survey

Reading academic papers is a skill that takes practice. When you first start out reading papers, you may find that you need to re-read things several times before you understand them, or that details will still be very fuzzy even after you've put a great amount of effort into trying to understand a paper. Don't panic, this is normal.

Even if you are used to reading papers from your particular sub-area, it can be challenging to read papers about a completely different topic. Usually, people in different areas use different language/notation and focus on very different issues. Nevertheless, many of the most-successful people in academia and industry are those that are able to understand/adapt ideas from different areas. (There are a ton of smart people in the world working on all sorts of amazing things, it's good to know how to communicate with as many of them as possible.)

A common technique when trying to understand a new topic (or reading scientific papers for the first time) is to read and write notes on 10 papers on the topic. When you read the first paper, you'll often find that it's hard to follow. This can make reading through it take a long time and might still leave you feeling that many things don't make sense; keep reading and trying to take notes. When you get to the second paper, it might still be very hard to follow. But when you start getting to the 8th or 9th paper, things often start making more sense. You'll start to form an impression of what the influential works in the area are, you'll start getting to used to the language and jargon, you'll start to understand what the main issues that people who work on the topic care about, and you'll probably notice some important references that weren't on your initial list of 10 papers. Ideally, you'll also start to notice how the topic has changed over time and you may get ideas of future work that you could do on the topic.

To help you make progress on your project or to give you an excuse to learn about a new topic, for this part you should write a literature survey of at least 10 academic papers on a particular topic. While your personal notes on the papers may be longer, the survey should be at most 4 pages of text (excluding references/tables/figures) in a format similar to the one for this document. Some logical components of a literature survey might be:

- A description of the overall topic, and the key themes/trends across the papers.
- A short high-level description of what was explored in each paper. For example, describe the problem being addressed, the key components of the proposed solution, and how it was evaluated. In addition, it is important to comment on the *why* questions: why is this problem important and why would this particular solution method make progress on it? It's also useful to comment on the strengths and

weaknesses of the various works, and it's particularly nice if you can show how some works address the weaknesses of prior works (or introduce new weaknesses).

- One or more logical "groupings" of the papers. This could be in terms of the variant of the topic that they address, in terms of the solution techniques used, or in chronological terms.

Some advice on choosing the topic:

- The most logical/easy topic for your literature survey is a topic related to your course project, given that your final report will need a (shorter) literature survey included.
- If you are an undergrad, or a masters student without a research project yet, you may alternately want to choose a general area (like variance-reduced stochastic gradient, non-Gaussian graphical models, recurrent neural networks, matrix factorization, neural artistic style transfer, Bayesian optimization, transformer networks, etc.) as your topic.
- If you are a masters student that already has a thesis project, it could make sense to do a survey on a topic where ML intersects with your thesis (or where ML *could* intersect your thesis).
- If you are a PhD student, I would recommend using this an excuse to learn about a *completely different* topic than what you normally work on. Choose something hard that you would like to learn about, but previously haven't been able to justify spending the time exploring carefully. This can be invaluable to your future research, because during/after your PhD it often becomes hard to allocate time to learn completely new topics.

Neural Architecture Search Review

Dingqing Yang (38800141), Junjie Zhu (30921167)
Department of Electrical and Computer Engineering, Department of Mathematics
University of British Columbia
`dingqingy@ece.ubc.ca`, `jzhu@math.ubc.ca`

Abstract

The success of deep learning has dominated the shift from feature engineering to architecture engineering. However, designing and deploying neural networks is still a time-consuming process and requires human expertise. We need an end-to-end solution that abstracts away details about architecture designing and hyperparameter tuning. In this survey, we exploit different approaches that enable automatic *Neural Architecture Search (NAS)*. Remaining challenges and possible future directions are also discussed.

1 Introduction

Deep learning has achieved state-of-the-art performance in many Artificial Intelligence (AI) applications. One reason for the success of deep neural networks (DNNs) can be regarded as automating the process of feature engineering. For example, in the field of computer vision (CV), we used to manually design features like SIFT [Lowe, 2004]. Nowadays, convolutional neural networks (CNN) are able to learn filters end-to-end in a data-driven fashion. However, it turns the problem of how to design good features into designing good neural network architectures.

Network architecture design space is huge such that exhaustive search is unrealistic. Therefore, DNN design relies heavily on different heuristics and human priors, which may lead to sub-optimal performance. Even with these human priors, finding a good network architecture still needs multiple iterations of cross-validation and hyperparameter tuning, which is very time-consuming. In order to overcome the aforementioned problems, researchers start to develop automatic NAS method in a data-driven approach. The first work that triggers extensive research on NAS is [Zoph and Le, 2016]. Zoph and Le [2016] demonstrate both state-of-the-art results on vision and language task using a reinforcement learning (RL) agent to automate the design process. However, the search process requires thousands of GPU days, which is too expensive for common machine learning practitioners. Many recent works (summarized in section) improve upon this and reduce computational cost up to single GPU days.

In this survey, we make the following contributions:

- We summarize techniques used to enable automatic architecture search and ways to reduce its computational cost.
- We review 10 representative recent works and summarize their key contributions and drawbacks.
- We further discuss open questions and future trends of the area.

The rest of the survey is organized as follows: Section 2 summarize NAS method powered by RL agents. Section 3 introduces one-shot architecture search methods that efficiently reduce computational cost. Section 4 specifies other techniques used to boost NAS efficiency. Section 5 describes several works that generalize NAS in other settings. Section 6 concludes the survey.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

2 RL-Based NAS

2.1 High-Level Overview

Neural architecture search space[1] is discrete and non-differentiable. It is quite challenging optimizing over non-differentiable search space. Therefore, researchers formulate NAS as an RL problem or using evolutionary algorithm. We will mainly focus on the RL approach in this survey due to space limitation.

An overview of RL-based NAS can be shown in Figure 1. A recurrent network controller can sample an architecture specified by a variable length sequence that consists of different architecture hyperparameters (e.g. number of hidden neurons in MLP, filter size, and stride in CNN, etc.) A child network will be built given the architecture specification, trained and validated using normal cross-validation. The validation score is used as the reward signal to guide the controller update using the policy gradient method. The updated controller is more likely to sample a good architecture, and we repeat this loop many times until we are satisfied with the validation accuracy of the child network.

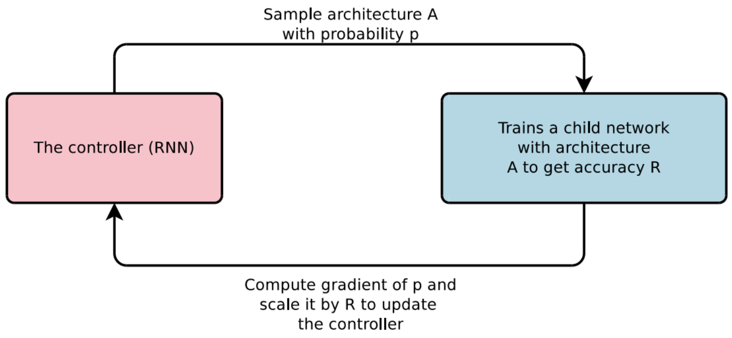


Figure 1: An overview of RL-based NAS methods [Zoph and Le, 2016]

2.2 Review of Major Contributions

[Zoph and Le, 2016] is the first work (to the best of our knowledge) that use RL-based NAS and achieves state-of-the-art 3.65% test error on CIFAR-10 and 62.4 test set perplexity on Penn Treebank. Their method is described in Figure 1. However, the main disadvantage of [Zoph and Le, 2016] is its severe computational cost. The authors report the whole search process uses 800 GPUs and find the desired model after 3 to 4 weeks. The main bottleneck is the sample generation part: generating one sample is equivalent to train a complete network to convergence (which can take days depending on the size of the dataset and available compute resource).

Zoph et al. [2018] realize the drawback of [Zoph and Le, 2016] and attempt to reduce computational cost by limiting search space. Motivated by GoogLeNet [Szegedy et al., 2015] and ResNet [He et al., 2016], Zoph et al. [2018] inject a human prior that deep networks with repeated motifs (i.e. a cell) as the macro architecture tend to work well. Therefore, they use a human-designed macro and only use NAS to search the structure of a single cell . They obtain 2.4% test error on CIFAR-10. Moreover, with the same cell searched on CIFAR-10 but stacked deep, their model achieves 96.2% top-5 accuracy on ImageNet. This demonstrates the great scalability of cell-based search methods. The authors use 500 GPUs for their experiment and take 4 days to find the best cell. The computational cost is reduced compared to [Zoph and Le, 2016] but still intractable for common ML practitioner.

2.3 Learning Curve Prediction

A specific technique that can greatly benefit the efficiency of sample generation for the aforementioned RL approach is learning curve prediction. [Domhan et al., 2015] proposed a probabilistic approach to this extrapolation of the learning curve and employed statistical decision theory, with probability

[1]Search space defines which architectures might be found in principle [Elsken et al., 2018]

computed using Markov chain Monte Carlo. Compared to previous work, this model increases the representation accuracy by considering linear combinations of functions instead of just using one, and account for uncertainty in the data and model parameters. The runtime is reduced by half when training cells in DNN, CNN, and large CNN on CIFAR-10 and MINST, and the model found is at or near the state-of-art. However, it is yet to be proved why this approach could be applied to other models with other data-sets.

3 One-Shot Architecture Search

3.1 High-Level Overview

The aforementioned RL-based method in Section 2 has the common problem that sampling generations is too expensive. The massive amount of effort is spent to train a child network with all trained weights discarded but only a validation accuracy needed. Inspired from multi-task learning, one-shot architecture search enforces parameter tieing across child model. We can represent a DNN model using a computational graph. A super-graph (one-shot model) that includes the complete search space is trained once, and the child model (which is a sub-graph of the one-shot model) is evaluated with shared weights from the super-graph of the same edges.

3.2 Review of Major Contributions

Pham et al. [2018] propose ENAS and use an RL controller to sample the sub-graph. The training is split into two interleaved processes. First, they fix the RL policy and optimized the parameters of the one-shot model that is approximated by the MC estimate of sampled child networks. Then, the weights of the one-shot model are fixed, the policy is updated using Policy Gradient method as other methods summarized in Section 2 but without training each child network, which greatly improves the efficiency. ENAS achieves 2.89% test error on CIFAR-10 and 55.8 test perplexity on Penn Treebank. ENAS computational cost is reduced to less than a single GPU day, which is much more efficient compared to previous NAS methods.

[Bender et al., 2018] is an analysis paper that tries to simplify and understand the one-shot model. The authors get rid of the RL controller that used in [Pham et al., 2018] and sample child network from a fixed distribution. Their main insight is to frame the one-shot model as a large powerful network with many redundant operations. Search best child candidate model is viewed as pruning the least useful operations with respect to validation accuracy. They show that samples generated from the one-shot model are strongly correlated with models that trained from scratch. Therefore, the one-shot model can accurately rank the relative performance between different child models even without fine-tuning them. This is the source of the efficiency of one-shot NAS model.

[Liu et al., 2018b] is a very promising recent work that tackles the problem of non-differentiable search space as well as takes advantage of the one-shot model. The author finds that using black-box optimization methods such as RL or evolutionary algorithm over a discrete domain is another source of inefficiency in previous work. Each edge is formulated as a mixture of different operation with different mixture probabilities. The authors jointly optimize both network weights and mixture probabilities using gradient descent and choose the most probable operation of each edge after training complete.

4 Other strategies that boost efficiency of NAS

4.1 surrogate model

[Liu et al., 2018a] propose PNAS that builds upon [Zoph et al., 2018] and construct architectures in a hierarchical and progressive manner. We introduce the concept of the cell before in Section 2.2, and a cell can be further broken down to multiple blocks. The authors limit the search space down to block level and add blocks progressively. A surrogate model is trained during block searching to predict validation performance. Due to the progressive nature of the algorithm, the surrogate model is able to predict well when only a single block is added in the architecture. This greatly improves efficiency since surrogate model evaluation is much cheaper compared to training a complete child network. The surrogate model is only used when expanding search space by adding another block.

After eliminating many architectures using the surrogate model, the remaining child model is trained to complete as normal and the surrogate model is retrained to match the performance of remaining child architectures. This approach is much efficient compared to RL based approach specified in Section 2, but much slower compared to the one-shot model described in Section 3. [Pérez-Rúa et al., 2018] is proposed to combine PNAS with the one-shot model to further boost efficiency.

4.2 Network Morphism

Another way to speed up sample generation of child network is to reuse already trained child network instead of training every child network from scratch. Network morphism is one successful technique that can be used for this setting. Chen et al. [2015] apply function-preserving operations on a small network to grow a deeper or wider network. The authors show that training the transformed bigger network is an order of magnitude fast compared to training from scratch. Wei et al. [2016] build upon [Chen et al., 2015] and expand possible morphing operation beyond simple Identity mapping used in [Chen et al., 2015].

5 Applications of NAS

5.1 Multi-Objective Learning

All previous works aim at finding good architectures with high validation score, but none of them take into account the underlying required resource utilization. Zhou et al. [2018] propose RENA that can find network architecture that works well with tight resource budgets. Edge devices like smartphones are constrained by the storage size and battery life, so small energy efficient networks with minimal accuracy degradation are favored. The authors take into account several factors that impact resource utilization (such as model size and computational intensity) and add those factors into a multi-objective reward and treat this as the feedback signal to update the RL agent.

5.2 Automatic Model Compression

Another interesting application is to apply NAS techniques to automate the process of DNN compression. Model compression techniques can greatly facilitate the DNN deployment on edge devices. However, human-designed compressed networks are usually based on heuristics and can lead to sub-optimal compression ratio. [He et al., 2018] propose AMC that uses a DDPG agent to propose a compression ratio for each layer. The resulting child compressed model is evaluated and its validation accuracy is returned to RL agents.

6 Conclusion

In this survey, we examine various NAS techniques and many approaches to further improve search efficiency. We believe approached that similar to DARTs [Liu et al., 2018b] is the most promising direction since it utilizes the one-shot model for efficiency and relaxes the non-differentiable search space to use the gradient-based algorithm. Although we have already had much success using NAS to automatically design DNNs, a number of challenges still remains. First, cells designed by NAS are very complex, which makes it more difficult to interpret. Second, we lack the theory to explain why certain searched architecture works well in certain tasks. In addition, we still rely on some human prior to efficiently reduce search space, but some people may argue this can lead to sub-optimal architectures. Also, complicated branches designed by NAS create irregular access pattern that is not friendly for data-parallel architecture such as GPUs and various DNN hardware accelerators like TPUs [Jouppi et al., 2017]. Finally, we are looking forward to seeing more extensions on NAS that enable automation of other complex tasks.

References

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pages 549–558, 2018.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12. IEEE, 2017.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018a.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018b.

David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

Juan-Manuel Pérez-Rúa, Moez Baccouche, and Stephane Pateux. Efficient progressive neural architecture search. *arXiv preprint arXiv:1808.00391*, 2018.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. In *International Conference on Machine Learning*, pages 564–572, 2016.

Yanqi Zhou, Siavash Ebrahimi, Sercan Ö Arık, Haonan Yu, Hairong Liu, and Greg Diamos. Resource-efficient neural architect. *arXiv preprint arXiv:1806.07912*, 2018.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.