TITLE OF YOUR THESIS

A Dissertation
Presented to
The Academic Faculty

By

George P. Burdell

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Fiction

Georgia Institute of Technology

January 1927

TITLE OF YOUR THESIS

Approved by:

Dr. Burdell, Advisor
School of Myths
Georgia Institute of Technology

Dr. Two
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Three
School of Electrical Engineering
Georgia Institute of Technology

Dr. Four
School of Computer Science
Georgia Institute of Technology

Dr. Five
School of Public Policy
Georgia Institute of Technology

Dr. Six
School of Nuclear Engineering
Georgia Institute of Technology

Date Approved: January 11, 2000

A great quote to start the thesis

George P. Burdell

A great dedication goes here.

ACKNOWLEDGEMENTS

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

CHAPTER 1

INTRODUCTION

Dictionaries and Dictionary Learning

Convolutional Dictionaries

Convolutional Neural Networks

Multi-Layer Dictionaries

Contributions and Organization of Dissertation

Table 1.1: This is an example Table.

x	f(x)	g(x)
1	6	4
2	6	3
3	6	2
4	6	2

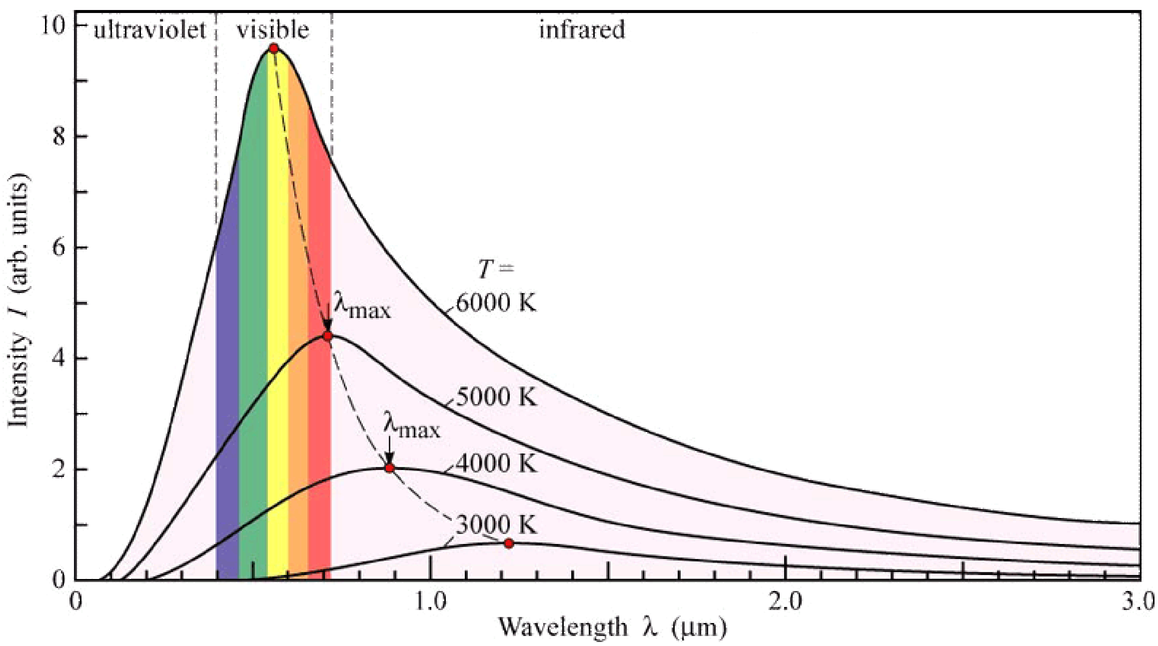

Figure 1.1: This is an example Figure.

CHAPTER 2

LEARNING DICTIONARIES FOR MULTI-CHANNEL SIGNALS

Introduction

When using a multi-layer dictionary model, the coefficients corresponding to a dictionary from one layer become the ”signal” for the subsequent layer. The number of channels for this ”signal” is the number of dictionary filters from the previous layer. Much of the literature on learning convolutional dictionaries is tailored to applications with signals that only have a small number of channels. This chapter presents a novel method for learning convolutional dictionaries from and for multi-channel signals.

Dictionary Types

There are many ways to construct a convolutional sparse representation of a multi-channel signal, but broadly the distinctions reduce down to if and how signal channels share dictionaries and coefficients, and if and how those non-shared entities interact across channels.

It is common in many applications for dictionary models to share dictionaries across channels, which requires the use multi-channel coefficients. If such models were used in a multi-layer dictionary model, the tensor rank would increase with each subsequent layer.

For this work, I focus instead on the multi-channel dictionary with shared coefficients. This structure matches that of convolutional neural networks, and the number of channels for a subsequent dictionary is the number of filters for the dictionary from the previous layer.

Literature Review

Convolutional Sparse Coding

ADMM with Low-Rank Updates

Conclusion

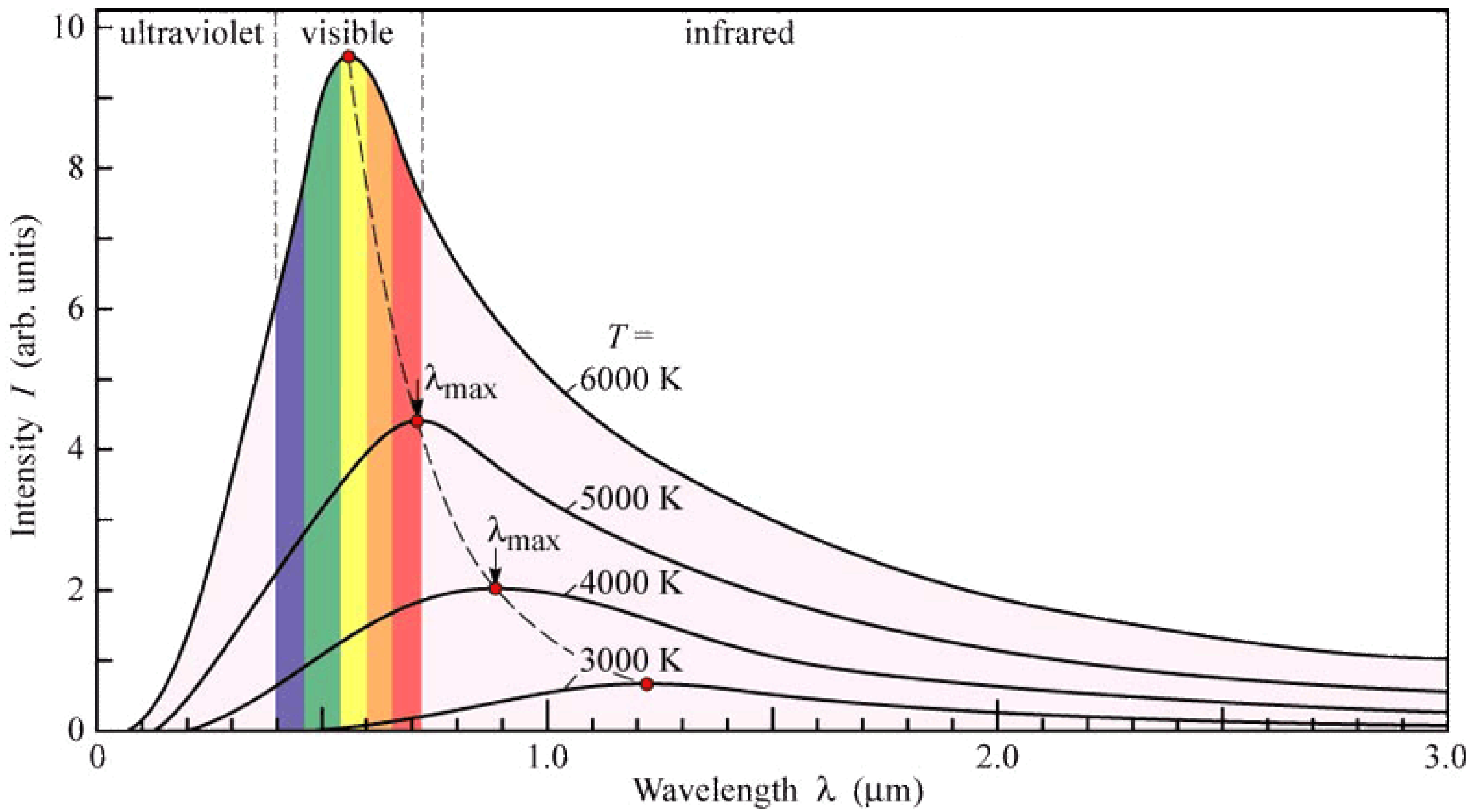

Figure 2.1: This is another example Figure, rotated to landscape orientation.

CHAPTER 3

LEARNING MULTI-LAYER DICTIONARIES

Introduction

A multi-layer dictionary model is composed of multiple dictionaries; the model treats the dictionary coefficients of a previous layer as the signal for the subsequent layer. This model dates back to Zeiler's Deconvolutional Neural Networks [1] and can be thought of as a deep autoencoder [2, Chapter 14][3]. Some researchers have interpreted convolutional neural networks as multi-layer dictionary models, the convolution and its corresponding rectified linear units serving as a crude pursuit algorithm [4]. In this chapter, I explain how to apply the novel dictionary learning algorithm from the prior chapter to the multi-layer dictionary learning problem.

Literature Review

In 2010, Zeiler et al. proposed a multi-layer dictionary model termed a deconvolutional network. The learning process for dictionary filters is entirely unsupervised, and they learn their filters layer-by-layer. Their algorithm is greedy in the sense that there is no feedback from subsequent layers to influence the learning process on the previous layer. This approach was tested both on the task of removing added gaussian noise to images, and also as a feature extraction method for object recognition on the Caltech-101 dataset [5]. While this research drew a lot of attention at the time, as the success of alternative models like convolutional neural networks grew, the popularity of deconolutional networks decreased.

Multi-layer dictionaries also appear in Bayesian models, going by names such as hierarchical convolutional factor analysis [6][7] and deep deconvolutional learning [8]. These networks use probabilistic models to prune network architecture and provide interpretable dictionaries. Inference can be slow.

Multi-Layer ADMM with Low-Rank Updates

Summary

CHAPTER 4

JPEG ARTIFACT REMOVAL

Introduction

Despite the existance of better compression algorithms, use of the JPEG compression algorithm is ubiquitous: it is the most commonly used image compression algorithm. Overzealous JPEG compression can produce visible distortions, and image restoration from these distortions is a challenging problem. There are two aspects of JPEG compression which make the restoration process more challenging than simpler restoration problems like deblurring or removing salt-and-pepper noise: JPEG's block-based approach is not spatially invariant, and the quantization is nonlinear. This chapter describes a novel approach to address the challenges of JPEG image restoration using the ADMM-based convolutional sparse coding for a multi-layer dictionary model.

JPEG Algorithm

The JPEG compression process begins with an RGB image input, and consists of five steps. The first is a color transformation, transitioning from RGB to YUV. Then, the U and V color channels are downsampled. The DCT for each 8×8 block is computed (separately for each channel). The DCT coefficients are then quantized using a quantization matrix determined by a user-chosen JPEG quality factor. Finally, these quantized coefficients are reodered and encoded using a lossless variable length coding process.

The standard reconstruction process reverses the lossless encoding, computes the IDCT of the blocks, upsamples the color channels, and reverses the color transform.

Literature Review

Modelling Compressed JPEG Images

Handling Quantization

Experiments

Experiment Setup

Results

Conclusion

CHAPTER 5

PRACTICAL CONSIDERATIONS CONCERNING TENSORFLOW

Boundary Handling

Removing Low-Frequency Signal Content

JPEG Artifact Removal

Tensorflow and Keras

Most of the computations for my research rely on TensorFlow version 2.3.1 [9], a Python library for machine learning specializing in building models with differentiable, parameterizable composite functions and learning model parameters using gradient descent or other gradient-based optimization methods. TensorFlow is a common platform for researchers and developers working on artificial neural netwokrs, and there are many tutorials and exampes freely available online, so I will not replicate that work here. This chapter section the reader already has some familiarity with TensorFlow and Keras [10] (a high-level library inside TensorFlow). The goal of this section is to provide the reader with the tools and workarounds to be able to replicate my work without resorting to hacking things together with gradient tape and/or TensorFlow-1-style code.

Why Not Use Gradient Tape and TensorFlow-1-Stye Code?

Keras offers a high-level environment. Code written in Keras's framework is easier to integrate with other work. Gradient tape is great for hacking something together or debugging, but promotes styles of coding that are less readable, less maintainable, and less portable. Keras also has a lower learning curve than the broader TensorFlow library.

Shared Weights Between Layers

Trainable TensorFlow variables declared outside of any Keras layer will not be automatically added to a Keras model's list of trainable variables. In most cases, this limitation is not a problem; it is intuitive to declare a layer's weights inside that layer. However, sometimes the same variable is needed in multiple distinct layers. To be include a variable in the model's trainable variables, it is sufficient to declare the variable in one layer and pass the variable (or the layer it was initialized in) as an input argument to the __init__ function of the other layers that share that variable. This will work even if the Keras model does not use the layer that declared the variable. [1]

Custom Partial Gradients

TensorFlow offers a well-documented means of replacing TensorFlow's gradient computations of an operation with specified custom gradient computations. However, if the operation involves multiple tensors that are inputs or trainable variables, the standard approach replaces all the gradients with custom gradients. If TensorFlow's gradient computations are sufficient for some tensors but not others, a workaround is necessary. This workaround is best explained by example.

Suppose the operation is the following:

```
z = f(x,y)
```

for which the standard TensorFlow gradient computations of f are desired in respect to x, but the custom gradient computations desired in respect to y are specified in function $g(\nabla_z \mathcal{L})$. This can be rewritten as the following:

[1] One could instead declare the variable outside any layers, pass it into the __init__ functions of all the variables that depend on it, and then manually add the variable to the model's list of trainable variables, but I do not recommend this approach. The resulting code will be less readable and much less maintainable.

```
@tf.custom_gradient
def h(z,y):
   def grad_fun(grad):
      return (tf.identity(grad),g(grad))
   return z,grad_fun
z = f(x,tf.stop_gradient(y))
z = h(z,y)
```

The function h does nothing on the forward pass, but in the backward pass computes the custom gradient in respect to y as intended.

Updating TensorFlow Variables After Applying Gradients

To update TensorFlow Variables after applying gradients, it is necessary to track which variables are affected and what their corresponding update functions are. To accomplish this, I store the update functions in a Python dictionary using variable names as the dictionary keys. This Python dictionary needs to be widely accessible so that layers can add update functions when they are initialized; a simple way to do this is to make the update function Python dictionary a class attribute. The keys need to be unique, but TensorFlow variable names can conflict. It is easy to avoid this problem by checking for conflicts before adding a new update function.

```
class PostProcess:
   update = {}
   def add_update(varName,update_fun):
      assert varName not in PostProcess.update
      PostProcess.update[varName] = update_fun
```

In the standard Keras training paradigm, models are trained using the fit function, a method in the Keras model object. The fit function calls the function train_step, where gradients are applied. To update TensorFlow Variables after gradients are applied, train_step is the function to modify. The only change that needs to be made is adding a function call to all update functions that correspond to the model's list of trainable variables.

```
class Model_subclass(tf.keras.Model):
   def train_step(self,data):
      trainStepOutputs =
         tf.keras.Model.train_step(self,data)
      update_ops = []
      for tv in self.trainable_variables:
         if tv.name in PostProcess.update:
            PostProcess.update[tv.name]()
      return trainStepOutputs
```

Changes to Tensorflow variables in the update function must use the assign command (or its variants: assign_add, assign_sub, ect). Otherwise, TensorFlow will detect that computations lie outside of its computational graph and throw an error. Note that using the assign command on Python variables that are not TensorFlow variables will produce some very cryptic error messages, so be sure to use the assign command correctly. If the value change of one TensorFlow variable depends on the value of another TenorFlow variable value pre-update, it may be necessary to use the Tensorflow control_dependencies command to get TensorFlow to track that dependency. TensorFlow has a useful tool called TensorBoard that helps visualize TensorFlow's dependencies, but a workaround is required to use TensorBoard on update functions that are called after applying gradients. To use TensorBoard to visualize dependencies in an update function, temporarily call the update function in the layer's call method, use TensorBoard to verify all necessary dependancies are being tracked, then remove the update function call from the layer's call method.

The TensorFlow Probability version 0.11.1 [11] is an extension of TensorFlow mosly used for probabilistic models. The library contains a Cholesky update function, but the function does not properly handle complex inputs. To compute Cholesky updates for complex inputs, users should either write their own implementation or use my code (included in supplementary material). Similarly, the Randomized SVD algorithm in the Python scikit-learn library does not properly handle complex inputs.

Errors like these are fairly common, so when dealing with complex data, researchers and practitioners should carefully verify that the function libraries they rely on are properly handling complex numbers.

Appendices

APPENDIX A

EXPERIMENTAL EQUIPMENT

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

APPENDIX B
DATA PROCESSING

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

REFERENCES

[1] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *2010 IEEE Computer Society Conference on computer vision and pattern recognition*, IEEE, 2010, pp. 2528–2535.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[3] A. Rangamani, A. Mukherjee, A. Basu, A. Arora, T. Ganapathi, S. Chin, and T. D. Tran, "Sparse coding and autoencoders," in *2018 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2018, pp. 36–40.

[4] V. Papyan, Y. Romano, and M. Elad, "Convolutional neural networks analyzed via convolutional sparse coding," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 2887–2938, 2017.

[5] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," in *2004 conference on computer vision and pattern recognition workshop*, IEEE, 2004, pp. 178–178.

[6] B. Chen, G. Polatkan, G. Sapiro, L. Carin, and D. B. Dunson, "The hierarchical beta process for convolutional factor analysis and deep learning," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 361–368.

[7] ar, "Deep learning with hierarchical convolutional factor analysis," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1887–1901, 2013.

[8] Y. Pu, X. Yuan, and L. Carin, "Generative deep deconvolutional learning," *ArXiv preprint arXiv:1412.6039*, 2014.

[9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *Tensorflow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015.

[10] F. Chollet *et al.*, *Keras*, `https://keras.io`, 2015.

[11] J. V. Dillon, I. Langmore, D. Tran, E. Brevdo, S. Vasudevan, D. Moore, B. Patton, A. Alemi, M. Hoffman, and R. A. Saurous, "Tensorflow distributions," *ArXiv preprint arXiv:1711.10604*, 2017.

VITA

Vita may be provided by doctoral students only. The length of the vita is preferably one page. It may include the place of birth and should be written in third person. This vita is similar to the author biography found on book jackets.