Thesis Proposal
Incorporating Structural Bias into Neural Networks

Zichao Yang

Nov 2017

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Eric Xing (co-chair)
Taylor Berg-Kirkpatrick (co-chair)
Alexander Smola
Ruslan Salakhutdinov
Nando de Freitas (DeepMind)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

Abstract

The rapid progress in artificial intelligence in recent years can largely be attributed to the resurgence of neural networks, which enables learning representations in an end-to-end manner. Although neural network are powerful, they have many limitations. For examples, neural networks are computationally expensive and memory inefficient; Neural network training needs many labeled exampled, especially for tasks that require reasoning and external knowledge. The goal of this thesis is to overcome some of the limitations by designing neural network with structural bias of the inputs taken into consideration.

This thesis aims to improve the efficiency of neural networks by exploring structural properties of inputs in designing model architectures. Specifically, this thesis augments neural networks with designed modules to improves their computational and statistical efficiency. We instantiate those modules in a wide range of tasks including supervised learning and unsupervised learning and show those modules not only make neural networks consume less memory, but also generalize better.

Contents

Chapter 1

Introduction

Over the past decades, machine learning models have relied heavily on hand-crafted features to learn well. With the recent breakthrough of neural networks in computer vision [21] and natural language processing [40], feature representations can be learned directly in an end-to-end manner. Minimum human efforts are required to preprocess the data and neural network models are directly resorted to optimize learning the feature representation. Although neural networks are powerful, they still have many limitations. On the one hand, neural networks are computationally expensive and memory inefficient. They can have hundreds of millions of parameters and a model can take several GBs. On the other hand, neural networks, though with large representation capacity, require many labeled examples to work well. This is especially true for tasks that require reasoning and external knowledge, which are two key components in human intelligence. The goal of this thesis is to overcome some of the limitations by designing neural networks with structural bias taken into consideration. Designing neural network according to characteristics of the inputs reduces the capacity of hypothesis space, this not only reduces the memory footprint of neural networks, but also make neural networks learn more efficiently and generalize better.

This thesis is composed of three parts:

- **Incorporating Structural Bias for Computational Efficiency**: Fully connected layer in neural networks are computational expensive and memory inefficient. Reducing the number of parameters would improve the efficiency of neural networks and facilitate its deployment on embedded devices. In this part of thesis, we propose to replace the fully connected layers with an adaptive Fastfood transform, which is a generalization of the Fastfood transform for approximating random features in kernels. We first begin by verifying the expressiveness of Fastfood transform in the context of kernel methods in Chapter 2. This part is largely based on work from [47]. We then embed the Fastfood transform in neural networks to replace fully connected layers to show that it can not only reduce the number of parameters but also has sufficient representation capacity. This part of work appears in Chapter 3 and is largely based on the work from [46].
- **Incorporating Structural Bias for Statistical Efficiency**: Neural network training needs many labeled examples. This is especially true for more advanced tasks that require reasoning and external knowledge, in which labeled examples are more expensive to get. Simple neural network models are not sufficient and it is necessary to design more efficient mod-

ules to accomplish these goals. In this part of the thesis, we augment neural networks with modules that support reasoning and querying external knowledge. We instantiate those models in the context of visual question answering (Chapter 4) and dialogue modeling (Chapter 5). In Chapter 4, we propose a stacked attention network to support multiple step reasoning to find the clues to the answer given an image and question. This chapter is largely based on the work from [49]. In Chapter 5, we study task oriented dialogue modeling. A dialogue model has to make the right recommendation based on an external database according to users' query. We augment a standard sequence to sequence model with a table pointer module to enable searching the table in replying users' query. This chapter is largely based on the work from [48]. We show models with those modules learn more efficiently and generalize better in those tasks.

- **Incorporating Structural Bias for Unsupervised Learning**: Incorporating structural bias is also helpful for unsupervised learning. In Chapter 6, we investigate the variational auto-encoder for text modeling. Previous work has found that LSTM-VAE tends to ignore the information from the encoder and the VAE collapses to a language model. We propose to use a dilated CNN as the decoder, with which we can control the explicit trade off between contextual capacity of decoders and efficient use of encoding information, thus avoiding the training collapse problem observed in previous works. This chapter is largely based on the work from [51].

Part I

Structural Bias for Computational Efficiency

Chapter 2

Learning Fast Kernels

2.1 Kernel Methods

2.1.1 Basic Properties

Denote by $\mathcal{X}$ the domain of covariates and by $\mathcal{Y}$ the domain of labels. Moreover, denote $X := \{x_1, \ldots, x_n\}$ and $Y := \{y_1, \ldots, y_n\}$ data drawn from a joint distribution p over $\mathcal{X} \times \mathcal{Y}$. Finally, let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a symmetric positive semidefinite kernel [28], such that every matrix K with entries $K_{ij} := k(x_i, x_j)$ satisfies $K \succeq 0$.

The key idea in kernel methods is that they allow one to represent inner products in a high-dimensional feature space implicitly, using

$$k(x, x') = \langle \phi(x), \phi(x') \rangle \,. \tag{2.1}$$

While the existence of such a mapping ϕ is guaranteed by the theorem of Mercer [28], manipulation of ϕ is not generally desirable since it might be infinite dimensional. Instead, one uses the representer theorem [19, 35] to show that when solving regularized risk minimization problems, the optimal solution $f(x) = \langle w, \phi(x) \rangle$ can be found as linear combination of kernel functions:

$$\langle w, \phi(x) \rangle = \left\langle \sum_{i=1}^{n} \alpha_i \phi(x_i), \phi(x) \right\rangle = \sum_{i=1}^{n} \alpha_i k(x_i, x).$$

While this expansion is beneficial for small amounts of data, it creates an unreasonable burden when the number of datapoints n is large. This problem can be overcome by computing approximate expansions.

2.1.2 Fastfood

The key idea in accelerating $\langle w, \phi(x) \rangle$ is to find an explicit feature map such that $k(x, x')$ can be approximated by $\sum_{j=1}^{m} \psi_j(x)\psi_j(x')$ in a manner that is both fast and memory efficient. Following the spectral approach of Rahimi and Recht [31], we exploit that for translation invariant kernels $k(x, x') = \kappa(x - x')$ we have

$$k(x, x') = \int \rho(\omega) \exp\left(i \left\langle \omega, x - x' \right\rangle\right) d\omega \,. \tag{2.2}$$

Here $\rho(\omega) = \rho(-\omega) \geq 0$ to ensure that the imaginary parts of the integral vanish. Without loss of generality we assume that $\rho(\omega)$ is normalized, e.g. $\|\rho\|_1 = 1$. A similar spectral decomposition holds for inner product kernels $k(x, x') = \kappa(\langle x, x' \rangle)$ [23, 34].

Rahimi and Recht [31] suggested to sample from the spectral distribution $\rho(\omega)$ for a Monte Carlo approximation to the integral in (2.2). For example, the Fourier transform of the popular Gaussian kernel is also Gaussian, and thus samples from a normal distribution for $\rho(\omega)$ can be used to approximate a Gaussian (RBF) kernel.

This procedure was refined by Le et al. [23] by approximating the sampling step with a combination of matrices that admit fast computation. They show that one may compute *Fastfood* approximate kernel expansions via

$$\tilde{k}(x, x') \propto \frac{1}{m} \sum_{j=1}^{m} \phi_j(x) \phi_j^*(x') \tag{2.3}$$
$$\text{where } \phi_j(x) = \exp\left(i[SHG\Pi HBx]_j\right).$$

The random matrices S, H, G, Π, B are chosen so as to provide a sufficient degree of randomness while also allowing for efficient computation.

B **Binary decorrelation** The entries B_{ii} of this diagonal matrix are drawn uniformly from $\{\pm 1\}$. This ensures that the data have zero mean in expectation over all matrices B.

H **Hadamard matrix** It is defined recursively via

$$H_1 := \begin{bmatrix} 1 \end{bmatrix} \text{ and } H_{2d} := \begin{bmatrix} H_d & H_d \\ H_d & -H_d \end{bmatrix}$$

The recursion shows that the dense matrix H_d admits fast multiplication in $O(d \log d)$ time, i.e. as efficiently as the FFT allows.

Π **Permutation matrix** This decorrelates the eigensystems of the two Hadamard matrices. Generating such a random permutation (and executing it) can be achieved by reservoir sampling, which amounts to n in-place pairwise swaps.

G **Gaussian matrix** G is a diagonal matrix with Gaussian entries drawn iid via $G_{ii} \sim \mathcal{N}(0, 1)$. The result of using it is that each of the rows of $HG\Pi HB$ consist of iid Gaussian random variables. Note, though, that the rows of this matrix are not quite independent.

S **Scaling matrix** This diagonal matrix encodes the spectral properties of the associated kernel. Consider $\rho(\omega)$ of (2.2). There we draw ω from the spherically symmetric distribution defined by $\rho(\omega)$ and use its length to rescale S_{ii} via

$$S_{ii} = \|\omega_i\| \, \|G\|_{\text{Frob}}^{-1}$$

It is straightforward to change kernels, for example, by adjusting S. Moreover, all the computational benefits of decomposing terms via (2.3) remain even after adjusting S. Therefore we can customize kernels for the problem at hand rather than applying a generic kernel, without incurring additional computational expenses.

2.2 À la Carte

In keeping with the culinary metaphor of Fastfood, we now introduce a flexible and efficient approach to kernel learning *à la carte*. That is, we will adjust the spectrum of a kernel in such a way as to allow for a wide range of translation-invariant kernels. Note that unlike previous approaches, this can be accomplished without any additional cost since these kernels only differ in terms of their choice of scaling.

In Random Kitchen Sinks and Fastfood, the frequencies ω are sampled from the spectral density $\rho(\omega)$. One could instead learn the frequencies ω using a kernel learning objective function. Moreover, with enough spectral frequencies, such an approach could learn any stationary (translation invariant) kernel. This is because each spectral frequency corresponds to a point mass on the spectral density $\rho(\omega)$ in (2.2), and point masses can model any density function.

However, since there are as many spectral frequencies as there are basis functions, individually optimizing over all the frequencies ω can still be computationally expensive, and susceptible to over-fitting and many undesirable local optima. In particular, we want to enforce smoothness over the *spectral distribution*. We therefore also propose to learn the scales, spread, and locations of *groups* of spectral frequencies, in a procedure that modifies the expansion (2.3) for fast kernel learning. This procedure results in efficient, expressive, and lightly parametrized models.

2.2.1 Gaussian Spectral Mixture Models

For the Gaussian Spectral Mixture kernels of Wilson and Adams [44], translation invariance holds, yet rotation invariance is violated: the kernels satisfy $k(x, x') = k(x+\delta, x'+\delta)$ for all $\delta \in \mathbb{R}^d$; however, in general rotations $U \in \mathrm{SO}(d)$ do not leave k invariant, i.e. $k(x, x') \neq k(Ux, Ux')$. These kernels have the following explicit representation in terms of their Fourier transform $F[k]$

$$F[k](\omega) = \sum_q \frac{v_q^2}{2} \left[\chi\left(\omega, \mu_q, \Sigma_q\right) + \chi\left(-\omega, \mu_q, \Sigma_q\right)\right]$$

$$\text{where } \chi(\omega, \mu, \Sigma) = \frac{e^{-\frac{1}{2}(\mu-\omega)^\top \Sigma^{-1}(\mu-\omega)}}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}}$$

In other words, rather than choosing a spherically symmetric representation $\rho(\omega)$ as typical for (2.2), Wilson and Adams [44] pick a mixture of Gaussians with mean frequency μ_q and variance Σ_q that satisfy the symmetry condition $\rho(\omega) = \rho(-\omega)$ but not rotation invariance. By the linearity of the Fourier transform, we can apply the inverse transform F^{-1} component-wise to obtain

$$k(x - x') = \sum_q v_q^2 \frac{|\Sigma_q|^{\frac{1}{2}}}{(2\pi)^{\frac{d}{2}}} \exp\left(-\frac{1}{2}\left\|\Sigma_q^{\frac{1}{2}}(x - x')\right\|^2\right) \cos\left\langle x - x', \mu_q\right\rangle \tag{2.4}$$

Lemma 1 (Universal Basis) *The expansion (2.4) can approximate any translation-invariant kernel by approximating its spectral density.*

Proof This follows since mixtures of Gaussians are universal approximators for densities [38], and by the Fourier-Plancherel theorem, approximation in the Fourier domain amounts to approximation in the original domain.

Note that the expression in (2.4) is *not* directly amenable to the fast expansions provided by Fastfood since the distributions are shifted. However, a small modification allows us to efficiently compute kernels of the form of (2.4). The key insight is that shifts in Fourier space by $\pm\mu_q$ are accomplished by multiplication by $\exp\left(\pm i \left\langle \mu_q, x \right\rangle\right)$. Here the inner product can be precomputed, which costs only $O(d)$ operations. Moreover, multiplications by $\Sigma_q^{-\frac{1}{2}}$ induce multiplication by $\Sigma_q^{\frac{1}{2}}$ in the original domain, which can be accomplished as *preprocessing*. For diagonal Σ_q the cost is $O(d)$.

In order to preserve translation invariance we compute a symmetrized set of features. We have the following algorithm (we assume diagonal Σ_q — otherwise simply precompute and scale x):

Preprocessing — Input $m, \{(\Sigma_q, \mu_q)\}$
for each q generate random matrices S_q, G_q, B_q, Π_q
Combine group scaling $B_q \leftarrow B_q \Sigma_q^{\frac{1}{2}}$

Feature Computation — Input $S, G, B, \Pi, \mu, \Sigma$
for $q = 1$ **to** Q **do**
$\zeta \leftarrow \left\langle \mu_q, x \right\rangle$ (offset)
$\xi \leftarrow [S_q H G_q \Pi_q H B_q x]$ (Fastfood product)
Compute features

$$\phi_{q\cdot 1} \leftarrow \sin(\xi + \zeta) \qquad \text{and } \phi_{q\cdot 2} \leftarrow \cos(\xi + \zeta)$$
$$\text{and } \phi_{q\cdot 3} \leftarrow \sin(\xi - \zeta) \qquad \text{and } \phi_{q\cdot 4} \leftarrow \cos(\xi + \zeta)$$

end for

To learn the kernel we learn the weights v_q, dispersion Σ_q and locations μ_q of spectral frequencies via marginal likelihood optimization, as described in section **??**. This results in a kernel learning approach which is similar in flexibility to individually learning all md spectral frequencies and is less prone to over-fitting and local optima. In practice, this can mean optimizing over about 10 free parameters instead of 10^4 free parameters, with improved predictive performance and efficiency. See section 4.2 for more detail.

2.2.2 Piecewise Linear Radial Kernel

In some cases the freedom afforded by a mixture of Gaussians in frequency space may be more than what is needed. In particular, there exist many cases where we *want* to retain invariance under rotations while simultaneously being able to adjust the spectrum according to the data at hand. For this purpose we introduce a novel piecewise linear radial kernel.

Recall (2.2) governs the regularization properties of k. We require $\rho(\omega) = \rho(\|\omega\|) := \rho(r)$ for rotation invariance. For instance, for the Gaussian RBF kernel we have

$$\rho(\|\omega\|_2) \propto \|\omega\|_2^{d-1} \exp\left(-\tfrac{\|\omega\|_2^2}{2}\right). \tag{2.5}$$

For high dimensional inputs, the RBF kernel suffers from a concentration of measure problem [23], where samples are tightly concentrated at the maximum of $\rho(r)$, $r = \sqrt{d-1}$. A fix is

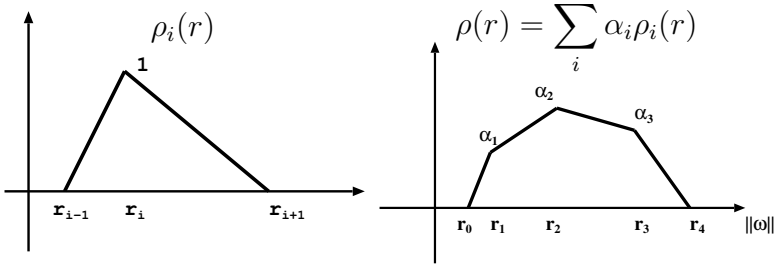


Figure 2.1: Piecewise linear functions. Top: single basis function. Bottom: linear combination of three functions. Additional degrees of freedom are fixed by $\rho(r_0) = \rho(r_4) = 0$.

relatively easy, since we are at liberty to pick any nonnegative ρ in designing kernels. This procedure is flexible but leads to intractable integrals: the Hankel transform of ρ, i.e. the radial part of the Fourier transform, needs to be analytic if we want to compute k in closed form.

However, if we remain in the Fourier domain, we can use $\rho(r)$ and sample *directly* from it. This strategy kills two birds with one stone: we do not need to compute the inverse Fourier transform and we have a readily available sampling distribution at our disposal for the Fastfood expansion coefficients S_{ii}. All that is needed is to find an efficient parametrization of $\rho(r)$.

We begin by providing an explicit expression for piecewise linear functions ρ_i such that $\rho_i(r_j) = \delta_{ij}$ with discontinuities only at r_{i-1}, r_i and r_{i+1}. In other words, $\rho(r)$ is a 'hat' function with its mode at r_i and range $[r_{i-1}, r_{i+1}]$. It is parametrized as

$$\rho_i(r) := \max\left(0, \min\left(1, \frac{r - r_{i-1}}{r_i - r_{i-1}}, \frac{r_i - r}{r_{i+1} - r_i}\right)\right)$$

By construction each basis function is piecewise linear with $\rho_i(r_j) = \delta_{ij}$ and moreover $\rho_i(r) \geq 0$ for all r.

Lemma 2 *Denote by $\{r_0, \ldots, r_n\}$ a sequence of locations with $r_i > r_{i-1}$ and $r_0 = 0$. Moreover, let $\rho(r) := \sum_i \alpha_i \rho_i(r)$. Then $\rho(r) \geq 0$ for all r if and only if $\alpha_i \geq 0$ for all i. Moreover, $\rho(r)$ parametrizes all piecewise linear functions with discontinuities at r_i.*

Now that we have a parametrization, we only need to discuss how to draw ω from $\rho(\|\omega\|) = \rho(r)$. We have several strategies at our disposal:

- $\rho(r)$ can be normalized explicitly via

$$\bar{\rho} := \int_0^\infty \rho(r)dr = \sum_i \frac{\alpha_i}{2(r_{i+1} - r_{i-1})}$$

  Since each segment ρ_i occurs with probability $\alpha_i/(2\bar{\rho}(r_{i+1} - r_{i-1})$ we first sample the segment and then sample from ρ_i explicitly by inverting the associated cumulative distribution function (it is piecewise quadratic).

- Note that sampling can induce considerable variance in the choice of locations. An alternative is to invert the cumulative distribution function and pick m locations equidistantly at locations $\frac{i}{m} + \xi$ where $\xi \sim U[0, 1/m]$. This approach is commonly used in particle filtering

[9]. We choose this strategy, since it is efficient yet substantially reduces the variance of sampling.

The basis functions are computed as follows:

Preprocessing$(m, \{(\{\alpha_i\}_{i=1}^n, \{r_i\}_{i=0}^{n+1}, \Sigma)\})$
Generate random matrices G, B, Π
Update scaling $B \leftarrow B\Sigma^{\frac{1}{2}}$
Sample S from $\rho(\|\omega\|)$ as above

Feature Computation(S, G, B, Π)

$$\phi_1 \leftarrow \cos([SHG\Pi HBx]) \text{ and } \phi_2 \leftarrow \sin([SHG\Pi HBx])$$

The rescaling matrix Σ_q is introduced to incorporate automatic relevance determination into the model. Like with the Gaussian spectral mixture model, we can use a mixture of piecewise linear radial kernels to approximate any radial kernel. Supposing there are Q components of the piecewise linear $\rho_q(r)$ function, we can repeat the proposed algorithm Q times to generate all the required basis functions.

2.2.3 Fastfood Kernels

The efficiency of Fastfood is partly obtained by approximating Gaussian random matrices with a product of matrices described in section 2.1.2. Here we propose several expressive and efficient kernel learning algorithms obtained by optimizing the marginal likelihood of the data in Eq. (**??**) with respect to these matrices:

FSARD The scaling matrix S represents the spectral properties of the associated kernel. For the RBF kernel, S is sampled from a chi-squared distribution. We can simply change the kernel by adjusting S. By varying S, we can approximate any radial kernel. We learn the diagonal matrix S via marginal likelihood optimization. We combine this procedure with *Automatic Relevance Determination* of Neal [30] – learning the scale of the input space – to obtain the **FSARD** kernel.

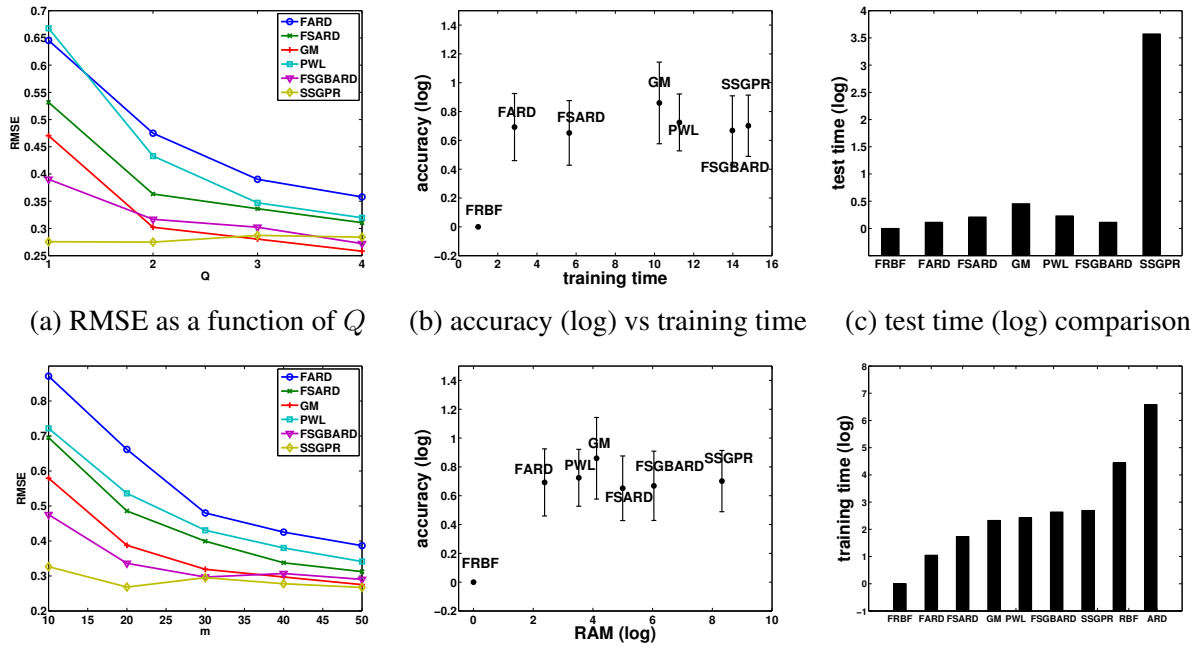
FSGBARD We can further generalize **FSARD** by additionally optimizing marginal likelihood with respect to the diagional matrices G and B in Fastfood to represent a wider class of kernels.

In both **FSARD** and **FSGBARD** the Hadamard matrix H is retained, preserving all the computational benefits of Fastfood. That is, we only modify the scaling matrices while keeping the main computational drivers such as the fast matrix multiplication and the Fourier basis unchanged.

2.3 Experiments

We evaluate the proposed kernel learning algorithms on many regression problems from the UCI repository. We show that the proposed methods are flexible, scalable, and applicable to a large and diverse collection of data, of varying sizes and properties. In particular, we demonstrate scaling to more than 2 million datapoints (in general, Gaussian processes are intractable beyond 10^4 datapoints); secondly, the proposed algorithms significantly outperform standard exact kernel methods, and with only a few hyperparameters are even competitive with alternative methods that

involve training orders of magnitude more hyperparameters.[1] All experiments are performed on an Intel Xeon E5620 PC, operating at 2.4GHz with 32GB RAM. Details such as initialization are in the supplement.



(a) RMSE as a function of Q (b) accuracy (log) vs training time (c) test time (log) comparison

(d) RMSE as a function of m (e) accuracy (log) vs ram (log) (f) training time (log) comparison

Figure 2.2: Fig. 2.2a and Fig. 2.2d illustrate how RMSE changes as we vary Q and m on the *SML* dataset. For variable Q, the number of basis functions per group m is fixed to 32. For variable m the number of clusters Q is fixed to 2. FRBF and FARD are Fastfood expansions of RBF and ARD kernels, respectively. Fig. 2.2b and Fig. 2.2e compare all methods in terms of accuracy, training time and memory. The accuracy score of a method on a given dataset is computed as $\text{accuracy}_{\text{method}} = \text{RMSE}_{\text{FRBF}}/\text{RMSE}_{\text{method}}$. For runtime and memory we take the reciprocal of the analogous metric, so that a lower score corresponds to better performance. For instance, $\text{time}_{\text{method}} = \text{walltime}_{\text{method}}/\text{walltime}_{\text{FRBF}}$. $\log$ denotes an average of the (natural) log scores, across all datasets. Fig. 2.2c and Fig. 2.2f compares all methods in terms of log test and training time (Fig. 2.2f also includes the average log training time for the exact ARD and RBF kernels across the smallest five medium datasets; these methods are intractable on any larger datasets).

RBF and ARD On smaller datasets, with fewer than $n = 2000$ training examples, where exact methods are tractable, we use exact Gaussian RBF and ARD kernels with hyperparameters learned via marginal likelihood optimization. ARD kernels use Automatic Relevance Determination [30] to adjust the scale of each input coordinate. Since these exact methods are intractable on larger datasets, we use Fastfood basis function expansions of these kernels for $n > 2000$.

[1]GM, PWL, FSARD, and FSGBARD are novel contributions of this paper, while RBF and ARD are popular alternatives, and SSGPR is a recently proposed state of the art kernel learning approach.

GM For Gaussian Mixtures we compute a mixture of Gaussians in frequency domain, as described in section 2.2.1. As before, optimization is carried out with regard to marginal likelihood.

PWL For rotation invariance, we use the novel Piecewise Linear radial kernels described in section 2.2.2. PWL has a simple spectral parametrization.

SSGPR Sparse Spectrum Gaussian Process Regression is a kitchen sinks (RKS) based model which individually optimizes the locations of *all* spectral frequencies [**?**]. We note that SSGPR is heavily parametrized. Moreover, SSGPR is a special case of the proposed GM model if $Q = m$, and we set all GM bandwidths to 0 and weigh all terms equally.

FSARD and FSGBARD See section 2.2.3.

In Figures 2.2a we investigate how RMSE performance changes as we vary Q and m. The GM and PWL models continue to increase in performance as more basis functions are used. This trend is not present with SSGPR or FSGBARD, which unlike GM and PWL, becomes more susceptible to over-fitting as we increase the number of basis functions. Indeed, in SSGPR, and in FSGBARD and FSARD to a lesser extent, more basis functions means more parameters to optimize, which is not true with the GM and PWL models.

We further investigate the performance of all seven methods, in terms of average normalised log predictive accuracy, training time, testing time, and memory consumption, shown in Figures 2.2b and 2.2c (higher accuracy scores and lower training time, test time, and memory scores, correspond to better performance). Despite the reduced parametrization, GM and PWL outperform all alternatives in accuracy, yet require similar memory and runtime to the much less expressive FARD model, a Fastfood expansion of the ARD kernel. Although SSGPR performs third best in accuracy, it requires more memory, training time, testing runtime (as shown in Fig 2.2c), than all other models. FSGBARD performs similar in accuracy to SSGPR, but is significantly more time and memory efficient, because it leverages a Fastfood representation. For clarity, we have so far considered log plots.

Chapter 3

Memory Efficient Convnets

3.1 The Adaptive Fastfood Transform

Large dense matrices are the main building block of fully connected neural network layers. In propagating the signal from the l-th layer with d activations $\mathbf{h}_l$ to the $l+1$-th layer with n activations $\mathbf{h}_{l+1}$, we have to compute

$$\mathbf{h}_{l+1} = \mathbf{W}\mathbf{h}_l. \tag{3.1}$$

The storage and computational costs of this matrix multiplication step are both $\mathcal{O}(nd)$. The storage cost in particular can be prohibitive for many applications.

Our proposed solution is to reparameterize the matrix of parameters $\mathbf{W} \in \mathbb{R}^{n \times d}$ with an Adaptive Fastfood transform, as follows

$$\mathbf{h}_{l+1} = (\mathbf{SHG\Pi HB})\,\mathbf{h}_l = \widehat{\mathbf{W}}\mathbf{h}_l. \tag{3.2}$$

. The storage requirements of this reparameterization are $\mathcal{O}(n)$ and the computational cost is $\mathcal{O}(n \log d)$. We will also show in the experimental section that these theoretical savings are mirrored in practice by significant reductions in the number of parameters without increased prediction errors.

To understand these claims, we need to describe the component modules of the Adaptive Fastfood transform. For simplicity of presentation, let us first assume that $\mathbf{W} \in \mathbb{R}^{d \times d}$. Adaptive Fastfood has three types of module:

- $\mathbf{S}, \mathbf{G}$ and $\mathbf{B}$ are diagonal matrices of parameters. In the original non-adaptive Fastfood formulation they are random matrices, as described further. The computational and storage costs are trivially $\mathcal{O}(d)$.
- $\mathbf{\Pi} \in \{0,1\}^{d \times d}$ is a random permutation matrix. It can be implemented as a lookup table, so the storageand computational costs are also $\mathcal{O}(d)$.
- $\mathbf{H}$ denotes the Walsh-Hadamard matrix, which is defined recursively as

$$\mathbf{H}_2 := \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \text{ and } \mathbf{H}_{2d} := \begin{bmatrix} \mathbf{H}_d & \mathbf{H}_d \\ \mathbf{H}_d & -\mathbf{H}_d \end{bmatrix}.$$

  The Fast Hadamard Transform, a variant of Fast Fourier Transform, enables us to compute $\mathbf{H}_d\mathbf{h}_l$ in $\mathcal{O}(d \log d)$ time.

In summary, the overall storage cost of the Adaptive Fastfood transform is $\mathcal{O}(d)$, while the computational cost is $\mathcal{O}(d \log d)$. These are substantial theoretical improvements over the $\mathcal{O}(d^2)$ costs of ordinary fully connected layers.

When the number of output units n is larger than the number of inputs d, we can perform n/d Adaptive Fastfood transforms and stack them to attain the desired size. In doing so, the computational and storage costs become $\mathcal{O}(n \log d)$ and $\mathcal{O}(n)$ respectively, as opposed to the more substantial $\mathcal{O}(nd)$ costs for linear modules. The number of outputs can also be refined with pruning.

3.1.1 Learning Fastfood by backpropagation

The parameters of the Adaptive Fastfood transform ($\mathbf{S}, \mathbf{G}$ and $\mathbf{B}$) can be learned by standard error derivative backpropagation. Moreover, the backward pass can also be computed efficiently using the Fast Hadamard Transform.

In particular, let us consider learning the l-th layer of the network, $\mathbf{h}_{l+1} = \mathbf{SHG\Pi HB h}_l$.

For simplicity, let us again assume that $\mathbf{W} \in \mathbb{R}^{d \times d}$ and that $\mathbf{h}_l \in \mathbb{R}^d$. Using backpropagation, assume we already have $\frac{\partial E}{\partial \mathbf{h}_{l+1}}$, where E is the objective function, then

$$\frac{\partial E}{\partial \mathbf{S}} = \operatorname{diag}\left\{ \frac{\partial E}{\partial \mathbf{h}_{l+1}} (\mathbf{HG\Pi HB h}_l)^\top \right\}. \tag{3.3}$$

Since $\mathbf{S}$ is a diagonal matrix, we only need to calculate the derivative with respect to the diagonal entries and this step requires only $\mathcal{O}(d)$ operations.

Proceeding in this way, denote the partial products by

$$\begin{aligned} \mathbf{h}_S &= \mathbf{HG\Pi HB h}_l \\ \mathbf{h}_{H1} &= \mathbf{G\Pi HB h}_l \\ \mathbf{h}_G &= \mathbf{\Pi HB h}_l \\ \mathbf{h}_\Pi &= \mathbf{HB h}_l \\ \mathbf{h}_{H2} &= \mathbf{B h}_l. \end{aligned} \tag{3.4}$$

Then the gradients with respect to different parameters in the Fastfood layer can be computed recursively as follows:

$$\begin{aligned} \frac{\partial E}{\partial \mathbf{h}_S} &= \mathbf{S}^\top \frac{\partial E}{\partial \mathbf{h}_{l+1}} & \frac{\partial E}{\partial \mathbf{h}_{H1}} &= \mathbf{H}^\top \frac{\partial E}{\partial \mathbf{h}_S} \\ \frac{\partial E}{\partial \mathbf{G}} &= \operatorname{diag}\left\{ \frac{\partial E}{\partial \mathbf{h}_{H1}} \mathbf{h}_G^\top \right\} & \frac{\partial E}{\partial \mathbf{h}_G} &= \mathbf{G}^\top \frac{\partial E}{\partial \mathbf{h}_{H1}} \\ \frac{\partial E}{\partial \mathbf{h}_\Pi} &= \mathbf{\Pi}^\top \frac{\partial E}{\partial \mathbf{h}_G} & \frac{\partial E}{\partial \mathbf{h}_{H2}} &= \mathbf{H}^\top \frac{\partial E}{\partial \mathbf{h}_\Pi} \\ \frac{\partial E}{\partial \mathbf{B}} &= \operatorname{diag}\left\{ \frac{\partial E}{\partial \mathbf{h}_{H2}} \mathbf{h}_l^\top \right\} & \frac{\partial E}{\partial \mathbf{h}_l} &= \mathbf{B}^\top \frac{\partial E}{\partial \mathbf{h}_{H2}}. \end{aligned} \tag{3.5}$$

Note that the operations in $\frac{\partial E}{\partial \mathbf{h}_{H1}}$ and $\frac{\partial E}{\partial \mathbf{h}_{H2}}$ are simply applications of the Hadamard transform, since $\mathbf{H}^\top = \mathbf{H}$, and consequently can be computed in $\mathcal{O}(d \log d)$ time. The operation in $\frac{\partial E}{\partial \mathbf{h}_\Pi}$

is an application of a permutation (the transpose of permutation matrix is a permutation matrix) and can be computed in $\mathcal{O}(d)$ time. All other operations are diagonal matrix multiplications.

3.2 Deep Fried Convolutional Networks

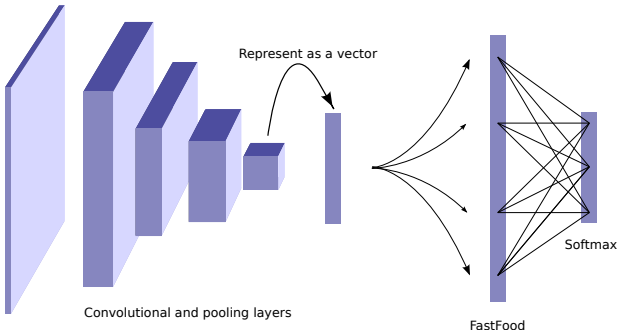


Figure 3.1: The structure of a deep fried convolutional network. The convolution and pooling layers are identical to those in a standard convnet. However, the fully connected layers are replaced with the Adaptive Fastfood transform.

We propose to greatly reduce the number of parameters of the fully connected layers by replacing them with an Adaptive Fastfood transform followed by a nonlinearity. We call this new architecture a deep fried convolutional network. An illustration of this architecture is shown in Figure 3.1.

In principle, we could also apply the Adaptive Fastfood transform to the softmax classifier. However, reducing the memory cost of this layer is already well studied; for example, [33] show that low-rank matrix factorization can be applied during training to reduce the size of the softmax layer substantially. Importantly, they also show that training a low rank factorization for the internal layers performs poorly, which agrees with the results of [8]. For this reason, we focus our attention on reducing the size of the internal layers.

3.3 Imagenet Experiments

We now examine how deep fried networks behave in a more realistic setting with a much larger dataset and many more classes. Specifically, we use the ImageNet ILSVRC-2012 dataset which has 1.2M training examples and 50K validation examples distributed across 1000 classes.

We use the the Caffe ImageNet model[1] as the reference model in these experiments [16]. This model is a modified version of AlexNet [21], and achieves 42.6% top-1 error on the ILSVRC-2012 validation set. The initial layers of this model are a cascade of convolution and pooling layers with interspersed normalization. The last several layers of the network take the form of an MLP and follow a 9216–4096–4096–1000 architecture. The final layer is a logistic regression

[1] `https://github.com/BVLC/caffe/tree/master/models/bvlc_reference_caffenet`

layer with 1000 output classes. All layers of this network use the ReLU nonlinearity, and dropout is used in the fully connected layers to prevent overfitting.

There are total of 58,649,184 parameters in the reference model, of which 58,621,952 are in the fully connected layers and only 27,232 are in the convolutional layers. The parameters of fully connected layer take up 99.9% of the total number of parameters. We show that the Adaptive Fastfood transform can be used to substantially reduce the number of parameters in this model.

ImageNet (fixed)	Error	Params
Dai et al. [7]	44.50%	163M
Fastfood 16	50.09%	16.4M
Fastfood 32	50.53%	32.8M
Adaptive Fastfood 16	45.30%	16.4M
Adaptive Fastfood 32	**43.77%**	32.8M
MLP	**47.76%**	58.6M

Table 3.1: Imagenet fixed convolutional layers: MLP indicates that we re-train 9216–4096–4096–1000 MLP (as in the original network) with the convolutional weights pretrained and fixed. Our method is *Fastfood 16* and *Fastfood 32*, using 16,384 and 32,768 Fastfood features respectively. [7] report results of max-voting of 10 transformations of the test set.

3.3.1 Fixed feature extractor

Previous work on applying kernel methods to ImageNet has focused on building models on features extracted from the convolutional layers of a pre-trained network [7]. This setting is less general than training a network from scratch but does mirror the common use case where a convolutional network is first trained on ImageNet and used as a feature extractor for a different task.

In order to compare our Adaptive Fastfood transform directly to this previous work, we extract features from the final convolutional layer of a pre-trained reference model and train an Adaptive Fastfood transform classifier using these features. Although the reference model uses two fully connected layers, we investigate replacing these with only a single Fastfood transform. We experiment with two sizes for this transform: *Fastfood 16* and *Fastfood 32* using 16,384 and 32,768 Fastfood features respectively. Since the Fastfood transform is a composite module, we can apply dropout between any of its layers. In the experiments reported here, we applied dropout after the $\mathbf{\Pi}$ matrix and after the $\mathbf{S}$ matrix. We also applied dropout to the last convolutional layer (that is, before the $\mathbf{B}$ matrix).

We also train an MLP with the same structure as the top layers of the reference model for comparison. In this setting it is important to compare against the re-trained MLP rather than the jointly trained reference model, as training on features extracted from fixed convolutional layers typically leads to lower performance than joint training [52].

The results of the fixed feature experiment are shown in Table 3.1. Following [52] and [7] we observe that training on ImageNet activations produces significantly lower performance than

of the original, jointly trained network. Nonetheless, deep fried networks are able to outperform both the re-trained MLP model as well as the results in [7] while using fewer parameters.

In contrast with our MNIST experiment, here we find that the Adaptive Fastfood transform provides a significant performance boost over the non-adaptive version, improving top-1 performance by 4.5-6.5%.

3.3.2 Jointly trained model

Finally, we train a deep fried network from scratch on ImageNet. With 16,384 features in the Fastfood layer we lose less than 0.3% top-1 validation performance, but the number of parameters in the network is reduced from 58.7M to 16.4M which corresponds to a factor of 3.6x. By further increasing the number of features to 32,768, we are able to perform 0.6% better than the reference model while using approximately half as many parameters. Results from this experiment are shown in Table 3.2.

ImageNet (joint)	Error	Params
Fastfood 16	46.88%	16.4M
Fastfood 32	46.63%	32.8M
Adaptive Fastfood 16	42.90%	16.4M
Adaptive Fastfood 32	**41.93%**	32.8M
Reference Model	**42.59%**	58.7M

Table 3.2: Imagenet jointly trained layers. Our method is *Fastfood 16* and *Fastfood 32*, using 16,384 and 32,768 Fastfood features respectively. *Reference Model* shows the accuracy of the jointly trained Caffe reference model.

Nearly all of the parameters of the deep fried network reside in the final softmax regression layer, which still uses a dense linear transformation, and accounts for more than 99% of the parameters of the network. This is a side effect of the large number of classes in ImageNet. For a data set with fewer classes the advantage of deep fried convolutional networks would be even greater. Moreover, as shown by [8, 33], the last layer often contains considerable redundancy. We also note that any of the techniques from [5, 6] could be applied to the final layer of a deep fried network to further reduce memory consumption at test time. We illustrate this with low-rank matrix factorization in the following section.

Part II

Structural Bias for Statistical Efficiency

Chapter 4

Stacked Attention Networks

In this chapter, we propose stacked attention networks (SANs) that allow multi-step reasoning for image QA. SANs can be viewed as an extension of the attention mechanism that has been successfully applied in image captioning [45] and machine translation [3]. The overall architecture of SAN is illustrated in Fig. 4.1a. The SAN consists of three major components: (1) the image model, which uses a CNN to extract high level image representations, e.g. one vector for each region of the image; (2) the question model, which uses a CNN or a LSTM to extract a semantic vector of the question and (3) the stacked attention model, which locates, via multi-step reasoning, the image regions that are relevant to the question for answer prediction. As illustrated in Fig. 4.1a, the SAN first uses the question vector to query the image vectors in the first visual attention layer, then combine the question vector and the retrieved image vectors to form a refined query vector to query the image vectors again in the second attention layer. The higher-level attention layer gives a sharper attention distribution focusing on the regions that are more relevant to the answer. Finally, we combine the image features from the highest attention layer with the last query vector to predict the answer.

4.1 Stacked Attention Networks (SANs)

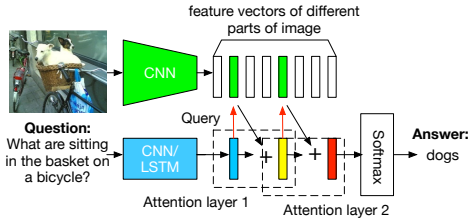
The overall architecture of the SAN is shown in Fig. 4.1a. We describe the three major components of SAN in this section: the image model, the question model, and the stacked attention model.

4.1.1 Image Model

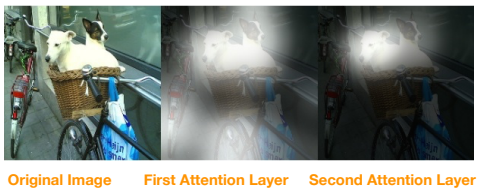
The image model uses a CNN [22, 39, 41] to get the representation of images. Specifically, the VGGNet [39] is used to extract the image feature map f_I from a raw image I:

$$f_I = \text{CNN}_{vgg}(I). \tag{4.1}$$

Unlike previous studies [11, 25, 32] that use features from the last inner product layer, we choose the features f_I from the last pooling layer, which retains spatial information of the original images. We first rescale the images to be 448×448 pixels, and then take the features from the last



(a) Stacked Attention Network for Image QA



(b) Visualization of the learned multiple attention layers. The stacked attention network first focuses on all referred concepts, e.g., `bicycle, basket` and objects in the basket (`dogs`) in the first attention layer and then further narrows down the focus in the second layer and finds out the answer `dog`.

Figure 4.1: Model architecture and visualization

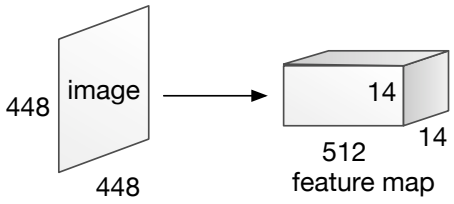


Figure 4.2: CNN based image model

pooling layer, which therefore have a dimension of $512 \times 14 \times 14$, as shown in Fig. 4.2. 14×14 is the number of regions in the image and 512 is the dimension of the feature vector for each region. Accordingly, each feature vector in f_I corresponds to a 32×32 pixel region of the input images. We denote by $f_i, i \in [0, 195]$ the feature vector of each image region.

Then for modeling convenience, we use a single layer perceptron to transform each feature vector to a new vector that has the same dimension as the question vector (described in Sec. 4.1.2):

$$v_I = \tanh(W_I f_I + b_I), \tag{4.2}$$

where v_I is a matrix and its i-th column v_i is the visual feature vector for the region indexed by i.

4.1.2 Question Model

As [10, 37, 40] show that LSTMs and CNNs are powerful to capture the semantic meaning of texts, we explore both models for question representations in this study.

LSTM based question model

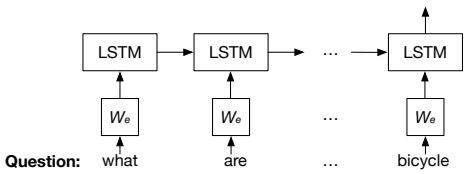


Figure 4.3: LSTM based question model

The essential structure of a LSTM unit is a memory cell c_t which reserves the state of a sequence. At each step, the LSTM unit takes one input vector (word vector in our case) x_t and updates the memory cell c_t, then output a hidden state h_t. The update process uses the gate mechanism. A forget gate f_t controls how much information from past state c_{t-1} is preserved. An input gate i_t controls how much the current input x_t updates the memory cell. An output gate o_t controls how much information of the memory is fed to the output as hidden state. The detailed update process is as follows:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i), \tag{4.3}$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f), \tag{4.4}$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o), \tag{4.5}$$
$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c), \tag{4.6}$$
$$h_t = o_t \tanh(c_t), \tag{4.7}$$

where i, f, o, c are input gate, forget gate, output gate and memory cell, respectively. The weight matrix and bias are parameters of the LSTM and are learned on training data.

Given the question $q = [q_1, ...q_T]$, where q_t is the one hot vector representation of word at position t, we first embed the words to a vector space through an embedding matrix $x_t = W_e q_t$. Then for every time step, we feed the embedding vector of words in the question to LSTM:

$$x_t = W_e q_t, t \in \{1, 2, ...T\}, \tag{4.8}$$
$$h_t = \text{LSTM}(x_t), t \in \{1, 2, ...T\}. \tag{4.9}$$

As shown in Fig. 4.3, the question `what are sitting in the basket on a bicycle` is fed into the LSTM. Then the final hidden layer is taken as the representation vector for the question, i.e., $v_Q = h_T$.

CNN based question model

In this study, we also explore to use a CNN similar to [18] for question representation. Similar to the LSTM-based question model, we first embed words to vectors $x_t = W_e q_t$ and get the question vector by concatenating the word vectors:

$$x_{1:T} = [x_1, x_2, ..., x_T]. \tag{4.10}$$

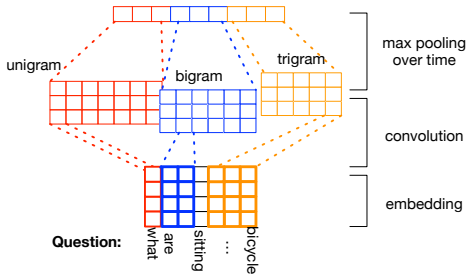


Figure 4.4: CNN based question model

Then we apply convolution operation on the word embedding vectors. We use three convolution filters, which have the size of one (unigram), two (bigram) and three (trigram) respectively. The t-th convolution output using window size c is given by:

$$h_{c,t} = \tanh(W_c x_{t:t+c-1} + b_c). \tag{4.11}$$

The filter is applied only to window $t : t + c - 1$ of size c. W_c is the convolution weight and b_c is the bias. The feature map of the filter with convolution size c is given by:

$$h_c = [h_{c,1}, h_{c,2}, ..., h_{c,T-c+1}]. \tag{4.12}$$

Then we apply max-pooling over the feature maps of the convolution size c and denote it as

$$\tilde{h}_c = \max_t [h_{c,1}, h_{c,2}, ..., h_{c,T-c+1}]. \tag{4.13}$$

The max-pooling over these vectors is a coordinate-wise max operation. For convolution feature maps of different sizes $c = 1, 2, 3$, we concatenate them to form the feature representation vector of the whole question sentence:

$$h = [\tilde{h}_1, \tilde{h}_2, \tilde{h}_3], \tag{4.14}$$

hence $v_Q = h$ is the CNN based question vector.

The diagram of CNN model for question is shown in Fig. 6.1b. The convolutional and pooling layers for unigrams, bigrams and trigrams are drawn in red, blue and orange, respectively.

4.1.3 Stacked Attention Networks

Given the image feature matrix v_I and the question feature vector v_Q, SAN predicts the answer via multi-step reasoning.

In many cases, an answer only related to a small region of an image. For example, in Fig. 4.1b, although there are multiple objects in the image: `bicycles`, `baskets`, `window`, `street` and `dogs` and the answer to the question only relates to `dogs`. Therefore, using the one global image feature vector to predict the answer could lead to sub-optimal results due to the noises introduced from regions that are irrelevant to the potential answer. Instead, reasoning

via multiple attention layers progressively, the SAN are able to gradually filter out noises and pinpoint the regions that are highly relevant to the answer.

Given the image feature matrix v_I and the question vector v_Q, we first feed them through a single layer neural network and then a softmax function to generate the attention distribution over the regions of the image:

$$h_A = \tanh(W_{I,A} v_I \oplus (W_{Q,A} v_Q + b_A)), \tag{4.15}$$

$$p_I = \text{softmax}(W_P h_A + b_P), \tag{4.16}$$

where $v_I \in \mathbf{R}^{d \times m}$, d is the image representation dimension and m is the number of image regions, $v_Q \in \mathbf{R}^d$ is a d dimensional vector. Suppose $W_{I,A}, W_{Q,A} \in \mathbf{R}^{k \times d}$ and $W_P \in \mathbf{R}^{1 \times k}$, then $p_I \in \mathbf{R}^m$ is an m dimensional vector, which corresponds to the attention probability of each image region given v_Q. Note that we denote by $\oplus$ the addition of a matrix and a vector. Since $W_{I,A} v_I \in \mathbf{R}^{k \times m}$ and both $W_{Q,A} v_Q, b_A \in \mathbf{R}^k$ are vectors, the addition between a matrix and a vector is performed by adding each column of the matrix by the vector.

Based on the attention distribution, we calculate the weighted sum of the image vectors, each from a region, $\tilde{v}_i$ as in Eq. 4.17. We then combine $\tilde{v}_i$ with the question vector v_Q to form a refined query vector u as in Eq. 4.18. u is regarded as a refined query since it encodes both question information and the visual information that is relevant to the potential answer:

$$\tilde{v}_I = \sum_i p_i v_i, \tag{4.17}$$

$$u = \tilde{v}_I + v_Q. \tag{4.18}$$

Compared to models that simply combine the question vector and the global image vector, attention models construct a more informative u since higher weights are put on the visual regions that are more relevant to the question. However, for complicated questions, a single attention layer is not sufficient to locate the correct region for answer prediction. For example, the question in Fig. 4.1 `what are sitting in the basket on a bicycle` refers to some subtle relationships among multiple objects in an image. Therefore, we iterate the above query-attention process using multiple attention layers, each extracting more fine-grained visual attention information for answer prediction. Formally, the SANs take the following formula: for the k-th attention layer, we compute:

$$h_A^k = \tanh(W_{I,A}^k v_I \oplus (W_{Q,A}^k u^{k-1} + b_A^k)), \tag{4.19}$$

$$p_I^k = \text{softmax}(W_P^k h_A^k + b_P^k). \tag{4.20}$$

where u^0 is initialized to be v_Q. Then the aggregated image feature vector is added to the previous query vector to form a new query vector:

$$\tilde{v}_I^k = \sum_i p_i^k v_i, \tag{4.21}$$

$$u^k = \tilde{v}_I^k + u^{k-1}. \tag{4.22}$$

That is, in every layer, we use the combined question and image vector u^{k-1} as the query for the image. After the image region is picked, we update the new query vector as $u^k = \tilde{v}_I^k + u^{k-1}$. We repeat this K times and then use the final u^K to infer the answer:

$$p_{\text{ans}} = \text{softmax}(W_u u^K + b_u). \tag{4.23}$$

Fig. 4.1b illustrates the reasoning process by an example. In the first attention layer, the model identifies roughly the area that are relevant to `basket, bicycle`, and `sitting in`. In the second attention layer, the model focuses more sharply on the region that corresponds to the answer `dogs`. More examples can be found in Sec. 4.2.

4.2 Experiments

4.2.1 Data sets

VQA is created through human labeling [1]. The data set uses images in the COCO image caption data set [24]. Unlike the other data sets, for each image, there are three questions and for each question, there are ten answers labeled by human annotators. There are $248,349$ training questions and $121,512$ validation questions in the data set. Following [1], we use the top 1000 most frequent answer as possible outputs and this set of answers covers 82.67% of all answers. We first studied the performance of the proposed model on the validation set. Following [10], we split the validation data set into two halves, val1 and val2. We use training set and val1 to train and validate and val2 to test locally. The results on the val2 set are reported in Table. 4.2. We also evaluated the best model, SAN(2, CNN), on the standard test server as provided in [1] and report the results in Table. 4.1.

4.2.2 Results

We compare our models with a set of baselines proposed recently [1, 25, 26, 27, 32] on image QA.

We follow [1] to use the following metric: $min(\#\text{ human labels that match that answer}/3, 1)$, which basically gives full credit to the answer when three or more of the ten human labels match the answer and gives partial credit if there are less matches.

Table. 4.1 summarizes the performance of various models on VQA. The overall results show that our best model, SAN(2, CNN), outperforms the LSTM Q+I model, the best baseline from [1], by 4.8% absolute. The superior performance of the SANs across all four benchmarks demonstrate the effectiveness of using multiple layers of attention.

As shown in Table. 4.1, compared to the best baseline LSTM Q+I, the biggest improvement of SAN(2, CNN) is in the *Other* type, 9.7%, followed by the 1.4% improvement in *Number* and 0.4% improvement in *Yes/No*. Note that the *Other* type in VQA refers to questions that usually have the form of "what color, what kind, what are, what type, where" etc. The VQA data set has a special *Yes/No* type of questions. The SAN only improves the performance of this type of questions slightly. This could due to that the answer for a *Yes/No* question is very question dependent, so better modeling of the visual information does not provide much additional gains.

	test-dev				test-std
Methods	All	Yes/No	Number	Other	All
VQA: [1]					
Question	48.1	75.7	36.7	27.1	-
Image	28.1	64.0	0.4	3.8	-
Q+I	52.6	75.6	33.7	37.4	-
LSTM Q	48.8	78.2	35.7	26.6	-
LSTM Q+I	53.7	78.9	35.2	36.4	54.1
SAN(2, CNN)	**58.7**	**79.3**	**36.6**	**46.1**	**58.9**

Table 4.1: VQA results on the official server, in percentage

Methods	All	Yes/No 36%	Number 10%	Other 54%
SAN(1, LSTM)	56.6	78.1	41.6	44.8
SAN(1, CNN)	56.9	78.8	42.0	45.0
SAN(2, LSTM)	57.3	78.3	**42.2**	45.9
SAN(2, CNN)	**57.6**	78.6	41.8	**46.4**

Table 4.2: VQA results on our partition, in percentage

This also confirms the similar observation reported in [1], e.g., using additional image information only slightly improves the performance in *Yes/No*, as shown in Table. 4.1, Q+I vs Question, and LSTM Q+I vs LSTM Q.

Chapter 5

Reference-aware Language Models

5.1 Reference to databases

NAME	PRICE RANGE	FOOD	AREA	ADDRESS	POST CODE	PHONE
ali baba	moderate	lebanese	centre	59 Hills Road City Centre	CB 2, 1 NT	01462 432565
the nirala	moderate	indian	north	7 Milton Road Chesterton	CB 4, 1 UY	01223 360966

Table 5.1: Fragment of database for dialogue system.

We consider the application of task-oriented dialogue systems in the domain of restaurant recommendations. To better understand the model, we first make a brief introduction of the data set. We use dialogues from the second Dialogue State Tracking Challenge (DSTC2) [14]. Table. 5.2 is one example dialogue from this dataset.

M:	Hello, welcome to the Cambridge restaurant system? You can ask for restaurants by `area`, `price range` or `food type` . How may I help you?
U:	i want a `moderately priced` restaurant in the `north` town
M:	`the nirala` is a nice restaurant in the north of town in the moderate price range
U:	what is the `address`
M:	Sure , `the nirala` is on `7 Milton Road Chesterton`

Table 5.2: Example dialogue, M stands for Machine and U stands for User

We can observe from this example, users get recommendations of restaurants based on queries that specify the area, price and food type of the restaurant. We can support the system's decisions by incorporating a mechanism that allows the model to query the database to

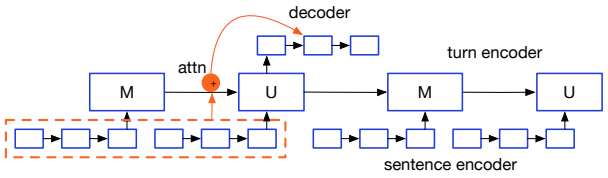


Figure 5.1: Hierarchical RNN Seq2Seq model. The red box denotes attention mechanism over the utterances in the previous turn.

find restaurants that satisfy the users' queries. A sample of our database (refer to data preparation part on how we construct the database) is shown in Table 5.1. We can observe that each restaurant contains 6 attributes that are generally referred in the dialogue dataset. As such, if the user requests a restaurant that serves "indian" food, we wish to train a model that can search for entries whose "food" column contains "indian". Now, we describe how we deploy a model that fulfills these requirements. We first introduce the basic dialogue framework in which we incorporates the table reference module.

Basic Dialogue Framework: We build a basic dialogue model based on the hierarchical RNN model described in [36], as in dialogues, the generation of the response is not only dependent on the previous sentence, but on all sentences leading to the response. We assume that a dialogue is alternated between a machine and a user. An illustration of the model is shown in Figure 5.1.

Consider a dialogue with T turns, the utterances from a user and a machines are denoted as $X = \{x_i\}_{i=1}^{T}$ and $Y = \{y_i\}_{i=1}^{T}$ respectively, where i is the i-th utterance. We define $x_i = \{x_{ij}\}_{j=1}^{|x_i|}$, $y_i = \{y_{iv}\}_{v=1}^{|y_i|}$, where x_{ij} (y_{iv}) denotes the j-th (v-th) token in the i-th utterance from the user (the machines). The dialogue sequence starts with a machine utterance and is given by $\{y_1, x_1, y_2, x_2, \ldots, y_T, x_T\}$. We would like to model the utterances from the machine

$$p(y_1, y_2, \ldots, y_T | x_1, x_2, \ldots, x_T) = \prod_i p(y_i | y_{<i}, x_{<i}) = \prod_{i,v} p(y_{i,v} | y_{i,<v}, y_{<i}, x_{<i}).$$

We encode $y_{<i}$ and $x_{<i}$ into continuous space in a hierarchical way with LSTM: **Sentence Encoder**: For a given utterance x_i, We encode it as $h^x_{i,j} = \text{LSTM}_\text{E}(W_E x_{i,j}, h^x_{i,j-1})$. The representation of x_i is given by the $h^x_i = h^x_{i,|x_i|}$. The same process is applied to obtain the machine utterance representation $h^y_i = h^y_{i,|y_i|}$. **Turn Encoder**: We further encode the sequence $\{h^y_1, h^x_1, ..., h^y_i, h^x_i\}$ with another LSTM encoder. We shall refer the last hidden state as u_i, which can be seen as the hierarchical encoding of the previous i utterances. **Decoder**: We use u_{i-1} as the initial state of decoder LSTM and decode each token in y_i. We can express the decoder as:

$$s^y_{i,v} = \text{LSTM}_\text{D}(W_E y_{i,v-1}, s_{i,v-1}),$$
$$p^y_{i,v} = \text{softmax}(W s^y_{i,v}).$$

We can also incoroprate the attetionn mechanism in the decoder. As shown in Figure. 5.1, we use the attention mechanism over the utterance in the previous turn. Due to space limit, we don't present the attention based decoder mathmatically and readers can refer to [2] for details.

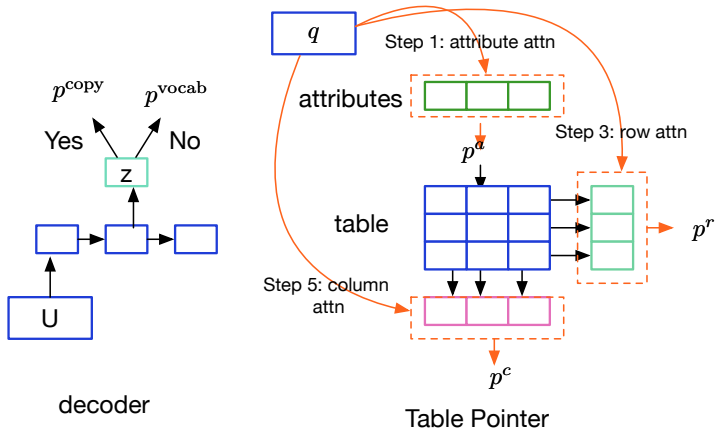


Figure 5.2: Decoder with table pointer.

Incorporating Table Reference

We now extend the decoder in order to allow the model to condition the generation on a database. **Pointer Switch**: We use $z_{i,v} \in \{0, 1\}$ to denote the decision of whether to copy one cell from the table. We compute this probability as follows:

$$p(z_{i,v}|s_{i,v}) = \text{sigmoid}(W s_{i,v}).$$

Thus, if $z_{i,v} = 1$, the next token $y_{i,v}$ is generated from the database, whereas if $z_{i,v} = 0$, then the following token is generated from a softmax. We now describe how we generate tokens from the database.

We denote a table with R rows and C columns as $\{t_{r,c}\}, r \in [1, R], c \in [1, C]$, where $t_{r,c}$ is the cell in row r and column c. The attribute of each column is denoted as s_c, where c is the c-th attribute. $t_{r,c}$ and s_c are one-hot vector.

Table Encoding: To encode the table, we first build an attribute vector and then an encoding vector for each cell. The attribute vector is simply an embedding lookup $g_c = W_E s_c$. For the encoding of each cell, we first concatenate embedding lookup of the cell with the corresponding attribute vector g_c and then feed it through a one-layer MLP as follows: then $e_{r,c} = \tanh(W[W_E t_{r,c}, g_c])$.

Table Pointer: The detailed process of calculating the probability distribution over the table is shown in Figure 5.2. The attention over cells in the table is conditioned on a given vector q, similarly to the attention model for sequences. However, rather than a sequence of vectors, we now operate over a table.

Step 1: Attention over the attributes to find out the attributes that a user asks about, $p^a = \text{ATTN}(\{g_c\}, q)$. Suppose a user says `cheap`, then we should focus on the `price` attribute.

Step 2: Conditional row representation calculation, $e_r = \sum_c p^a_c e_{r,c} \quad \forall r$. So that e_r contains the price information of the restaurant in row r.

Step 3: Attention over e_r to find out the restaurants that satisfy users' query, $p^r = \text{ATTN}(\{e_r\}, q)$. Restaurants with `cheap` price will be picked.

Step 4: Using the probabilities p^r, we compute the weighted average over the all rows $e_c = \sum_r p_r^r e_{r,c}$. $\{e_r\}$ contains the information of `cheap` restaurant.
Step 5: Attention over columns $\{e_r\}$ to compute the probabilities of copying each column $p^c = \text{ATTN}(\{e_c\}, q)$.
Step 6: To get the probability matrix of copying each cell, we simply compute the outer product $p^{\text{copy}} = p^r \otimes p^c$.
The overall process is as follows:

$$\begin{aligned} p^a &= \text{ATTN}(\{g_c\}, q), \\ e_r &= \sum_c p_c^a e_{r,c} \quad \forall r, \\ p^r &= \text{ATTN}(\{e_r\}, q), \\ e_c &= \sum_r p_r^r e_{r,c} \quad \forall c, \\ p^c &= \text{ATTN}(\{e_c\}, q), \\ p^{\text{copy}} &= p^r \otimes p^c. \end{aligned}$$

If $z_{i,v} = 1$, we embed the above attention process in the decoder by replacing the conditioned state q with the current decoder state $s_{i,v}^y$.

5.2 Experiments

5.2.1 Data sets

Dialogue: We use the DSTC2 data set. We only use the dialogue transcripts from the data set. There are about 3,200 dialogues in total. The table is not available from DSTC2. To reconstruct the table, we crawled TripAdvisor for restaurants in the Cambridge area, where the dialog dataset was collected. Then, we remove restaurants that do not appear in the data set and create a database with 109 restaurants and their attributes (e.g. food type). Since this is a small data set, we use 5-fold cross validation and report the average result over the 5 partitions. There may be multiple tokens in each table cell, for example in Table. 5.1, the name, address, post code and phone number have multiple tokens, we replace them with one special token. For the name, address, post code and phone number of the j-th row, we replace the tokens in each cell with _NAME_j, _ADDR_j, _POSTCODE_j, _PHONE_j. If a table cell is empty, we replace it with an empty token _EMPTY. We do a string match in the transcript and replace the corresponding tokens in transcripts from the table with the special tokens. Each dialogue on average has 8 turns (16 sentences). We use a vocabulary size of 900, including about 400 table tokens and 500 words.

5.2.2 Results

We compare our model with baselines that do not model reference explicitly. We compare our model with basic seq2seq and attention model. We also apply attention mechanism over the table

for dialogue modeling as a baseline. We train all models with simple stochastic gradient descent with gradient clipping. We use a one-layer LSTM for all RNN components. Hyper-parameters are selected using grid search based on the validation set.

The results for dialogue are shown in Table 5.3. The findings for dialogue are shown in Table 5.3. Conditioning table performs better in predicting table tokens in general. Table Pointer has the lowest perplexity for tokens in the table. Since the table tokens appear rarely in the dialogue transcripts, the overall perplexity does not differ much and the non-table token perplexity are similar. With attention mechanism over the table, the perplexity of table token improves over basic Seq2Seq model, but still not as good as directly pointing to cells in the table, which shows the advantage of modeling reference explicitly. As expected, using sentence attention improves significantly over models without sentence attention. Surprisingly, Table Latent performs much worse than Table Pointer. We also measure the perplexity of table tokens that appear only in test set. For models other than Table Pointer, because the tokens never appear in the training set, the perplexity is quite high, while Table Pointer can predict these tokens much more accurately. This verifies our conjecture that our model can learn reasoning over databases.

Model	All	Table	Table OOV	Word
Seq2Seq	1.35±0.01	4.98±0.38	1.99E7±7.75E6	1.23±0.01
Table Attn	1.37±0.01	5.09±0.64	7.91E7±1.39E8	1.24±0.01
Table Pointer	**1.33±0.01**	**3.99±0.36**	**1360 ± 2600**	**1.23±0.01**
Table Latent	1.36±0.01	4.99±0.20	3.78E7±6.08E7	1.24±0.01
+ Sentence Attn				
Seq2Seq	1.28±0.01	3.31±0.21	2.83E9 ± 4.69E9	**1.19±0.01**
Table Attn	1.28±0.01	3.17±0.21	1.67E7±9.5E6	1.20±0.01
Table Pointer	**1.27±0.01**	**2.99±0.19**	**82.86±110**	1.20±0.01
Table Latent	1.28±0.01	3.26±0.25	1.27E7±1.41E7	1.20±0.01

Table 5.3: Dialogue perplexity results. Table means tokens from table, Table OOV denotes table tokens that do not appear in the training set. **Sentence Attn** denotes we use attention mechanism over tokens in utterances from the previous turn.

Part III

Structural Bias for Unsupervised Learning

Chapter 6

Variational Autoencoders for Text Modeling

6.1 Background on Variational Autoencoders

Neural language models [29] typically generate each token x_t conditioned on the entire history of previously generated tokens:

$$p(\mathbf{x}) = \prod_t p(x_t|x_1, x_2, ..., x_{t-1}). \tag{6.1}$$

State-of-the-art language models often parametrize these conditional probabilities using RNNs, which compute an evolving hidden state over the text which is used to predict each x_t. This approach, though effective in modeling text, does not explicitly model variance in higher-level properties of entire utterances (e.g. topic or style) and thus can have difficulty with heterogeneous datasets.

Bowman et al. [4] propose a different approach to generative text modeling inspired by related work on vision [20]. Instead of directly modeling the joint probability $p(\mathbf{x})$ as in Equation 6.1, we specify a generative process for which $p(\mathbf{x})$ is a marginal distribution. Specifically, we first generate a continuous latent vector representation $\mathbf{z}$ from a multivariate Gaussian prior $p_\theta(\mathbf{z})$, and then generate the text sequence $\mathbf{x}$ from a conditional distribution $p_\theta(\mathbf{x}|\mathbf{z})$ parameterized using a neural net (often called the generation model or decoder). Because this model incorporates a latent variable that modulates the entire generation of each whole utterance, it may be better able to capture high-level sources of variation in the data. Specifically, in contrast with Equation 6.1, this generating distribution conditions on latent vector representation $\mathbf{z}$:

$$p_\theta(\mathbf{x}|\mathbf{z}) = \prod_t p_\theta(x_t|x_1, x_2, ..., x_{t-1}, \mathbf{z}). \tag{6.2}$$

To estimate model parameters θ we would ideally like to maximize the marginal probability $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{z})p_\theta(\mathbf{x}|\mathbf{z})d\mathbf{z}$. However, computing this marginal is intractable for many decoder

choices. Thus, the following variational lower bound is often used as an objective [20]:

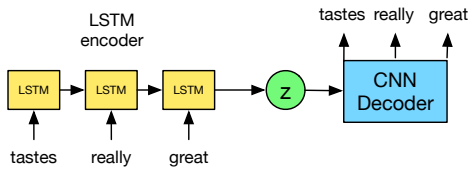
$$\begin{aligned}\log\ p_\theta(\mathbf{x}) &= -\log \int p_\theta(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) d\mathbf{z} \\ &\geq\ \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})).\end{aligned}$$

Here, $q_\phi(\mathbf{z}|\mathbf{x})$ is an approximation to the true posterior (often called the recognition model or encoder) and is parameterized by ϕ. Like the decoder, we have a choice of neural architecture to parameterize the encoder. However, unlike the decoder, the choice of encoder does not change the model class – it only changes the variational approximation used in training, which is a function of both the model parameters θ *a*nd the approximation parameters ϕ. Training seeks to optimize these parameters jointly using stochastic gradient ascent. A final wrinkle of the training procedure involves a stochastic approximation to the gradients of the variational objective (which is itself intractable). We omit details here, noting only that the final distribution of the posterior approximation $q_\phi(\mathbf{z}|\mathbf{x})$ is typically assumed to be Gaussian so that a re-parametrization trick can be used, and refer readers to [20].

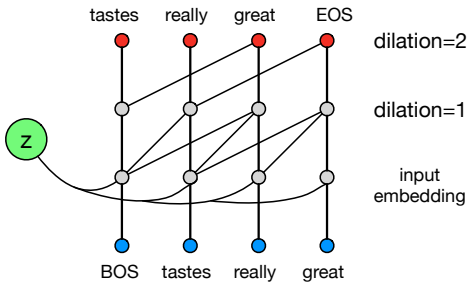
6.2 Training Collapse with Textual VAEs

Together, this combination of generative model and variational inference procedure are often referred to as a variational autoencoder (VAE). We can also view the VAE as a regularized version of the autoencoder. Note, however, that while VAEs are valid probabilistic models whose likelihood can be evaluated on held-out data, autoencoders are not valid models. If only the first term of the VAE variational bound $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]$ is used as an objective, the variance of the posterior probability $q_\phi(\mathbf{z}|\mathbf{x})$ will become small and the training procedure reduces to an autoencoder. It is the KL-divergence term, $\mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))$, that discourages the VAE memorizing each $\mathbf{x}$ as a single latent point.

While the KL term is critical for training VAEs, historically, instability on text has been evidenced by the KL term becoming vanishingly small during training, as observed by Bowman et al. [4]. When the training procedure collapses in this way, the result is an encoder that has duplicated the Gaussian prior (instead of a more interesting posterior), a decoder that completely ignores the latent variable $\mathbf{z}$, and a learned model that reduces to a simpler language model. We hypothesize that this collapse condition is related to the contextual capacity of the decoder architecture. The choice encoder and decoder depends on the type of data. For images, these are typically MLPs or CNNs. LSTMs have been used for text, but have resulted in training collapse as discussed above [4]. Here, we propose to use a dilated CNN as the decoder instead. In one extreme, when the effective contextual width of a CNN is very large, it resembles the behavior of LSTM. When the width is very small, it behaves like a bag-of-words model. The architectural flexibility of dilated CNNs allows us to change the contextual capacity and conduct experiments to validate our hypothesis: decoder contextual capacity and effective use of encoding information are directly related. We next describe the details of our decoder.



(a) VAE training graph using a dilated CNN decoder.



(b) Digram of dilated CNN decoder.

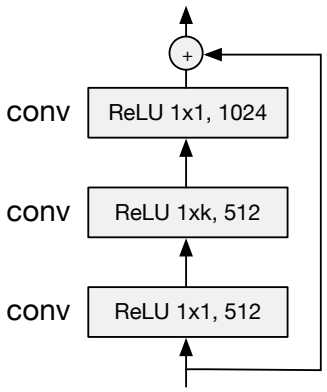
Figure 6.1: Our training and model architectures for textual VAE using a dilated CNN decoder.

6.3 Dilated Convolutional Decoders

The typical approach to using CNNs used for text generation [17] is similar to that used for images [13, 22], but with the convolution applied in one dimension. We take this approach here in defining our decoder.

One dimensional convolution: For a CNN to serve as a decoder for text, generation of x_t must only condition on past tokens $x_{<t}$. Applying the traditional convolution will break this assumption and use tokens $x_{\geq t}$ as inputs to predict x_t. In our decoder, we avoid this by simply shifting the input by several slots [43]. With a convolution with filter size of k and using n layers, our effective filter size (the number of past tokens to condition to in predicting x_t) would be $(k-1) \times n + 1$. Hence, the filter size would grow linearly with the depth of the network.

Dilation: Dilated convolution [53] was introduced to greatly increase the effective receptive field size without increasing the computational cost. With dilation d, the convolution is applied so that $d-1$ inputs are skipped each step. Causal convolution can be seen a special case with $d = 1$. With dilation, the effective receptive size grows exponentially with network depth. In Figure 6.1b, we show dilation of sizes of 1 and 2 in the first and second layer, respectively. Suppose the dilation size in the i-th layer is d_i and we use the same filter size k in all layers, then the effective filter size is $(k-1)\sum_i d_i + 1$. The dilations are typically set to double every layer $d_{i+1} = 2d_i$, so the effective receptive field size can grow exponentially. Hence, the contextual capacity of a CNN can be controlled across a greater range by manipulating the filter size, dilation size and network depth. We use this approach in experiments.



Residual connection: We use residual connection [13] in the decoder to speed up convergence and enable training of deeper models. We use a residual block (shown to the right) similar to that of [17]. We use three convolutional layers with filter size $1 \times 1, 1 \times k, 1 \times 1$, respectively, and ReLU activation between convolutional layers.

Overall architecture: Our VAE architecture is shown in Figure 6.1a. We use LSTM as the encoder to get the posterior probability $q(\mathbf{z}|\mathbf{x})$, which we assume to be diagonal Gaussian. We parametrize the mean μ and variance σ with LSTM output. We sample $\mathbf{z}$ from $q(\mathbf{z}|\mathbf{x})$, the decoder is conditioned on the sample by concatenating $\mathbf{z}$ with every word embedding of the decoder input.

6.4 Experiments

6.4.1 Data sets

We use two large scale document classification data sets: Yahoo Answer and Yelp15 review, representing topic classification and sentiment classification data sets respectively [42, 50, 54]. The original data sets contain millions of samples, of which we sample 100k as training and 10k as validation and test from the respective partitions. The detailed statistics of both data sets are in Table 6.1. Yahoo Answer contains 10 topics including Society & Culture, Science & Mathematics etc. Yelp15 contains 5 level of rating, with higher rating better.

Data	classes	documents	average #w	vocabulary
Yahoo	10	100k	78	200k
Yelp15	5	100k	96	90k

Table 6.1: Data statistics

6.4.2 Results

We use an LSTM as an encoder for VAE and explore LSTMs and CNNs as decoders. For CNNs, we explore several different configurations. We set the convolution filter size to be 3 and gradually increase the depth and dilation from [1, 2, 4], [1, 2, 4, 8, 16] to [1, 2, 4, 8, 16, 1, 2, 4, 8, 16]. They represent small, medium and large model and we name them as SCNN, MCNN and LCNN. We also explore a very large model with dilations [1, 2, 4, 8, 16, 1, 2, 4, 8, 16, 1, 2, 4, 8, 16] and name it as VLCNN. The effective filter size are 15, 63, 125 and 187 respectively. We use the last hidden state of the encoder LSTM and feed it though an MLP to get the mean and variance of $q(\mathbf{z}|\mathbf{x})$, from which we sample $\mathbf{z}$ and then feed it through an MLP to get the starting state of decoder. For the LSTM decoder, we follow [4] to use it as the initial state of LSTM and feed it to every step of LSTM. For the CNN decoder, we concatenate it with the word embedding of every decoder input.

The results for language modeling are shown in Table 6.2. We report the negative log likelihood (NLL) and perplexity (PPL) of the test set. For the NLL of VAEs, we decompose it into reconstruction loss and KL divergence and report the KL divergence in the parenthesis.

We first look at the LM results for Yahoo data set. As we gradually increase the effective filter size of CNN from SCNN, MCNN to LCNN, the NLL decreases from 345.3, 338.3 to 335.4. The NLL of LCNN-LM is very close to the NLL of LSTM-LM 334.9. But VLCNN-LM is a little bit worse than LCNN-LM, this indicates a little bit of over-fitting.

We can see that LSTM-VAE is worse than LSTM-LM in terms of NLL and the KL term is nearly zero, which verifies the finding of [4]. When we use CNNs as the decoders for VAEs, we can see improvement over pure CNN LMs. For SCNN, MCNN and LCNN, the VAE results improve over LM results from 345.3 to 337.8, 338.3 to 336.2, and 335.4 to 333.9 respectively. The improvement is big for small models and gradually decreases as we increase the decoder model contextual capacity. When the model is as large as VLCNN, the improvement diminishes and the VAE result is almost the same with LM result. This is also reflected in the KL term, SCNN-VAE has the largest KL of 13.3 and VLCNN-VAE has the smallest KL of 0.7. When LCNN is used as the decoder, we obtain an optimal trade off between using contextual information and latent representation. LCNN-VAE achieves a NLL of 333.9, which improves over LSTM-LM with NLL of 334.9.

We find that if we initialize the parameters of *LSTM encoder* with parameters of LSTM language model, we can improve the VAE results further. This indicates better encoder model is also a key factor for VAEs to work well. Combined with encoder initialization, LCNN-VAE improves over LSTM-LM from 334.9 to 332.1 in NLL and from 66.2 to 63.9 in PPL. Similar results for the sentiment data set are shown in Table 6.2b. LCNN-VAE improves over LSTM-LM from 362.7 to 359.1 in NLL and from 42.6 to 41.1 in PPL.

Model	Size	NLL (KL)	PPL
LSTM-LM	$< i$	334.9	66.2
LSTM-VAE**	$< i$	342.1 (0.0)	72.5
LSTM-VAE** + init	$< i$	339.2 (0.0)	69.9
SCNN-LM	15	345.3	75.5
SCNN-VAE	15	337.8 (13.3)	68.7
SCNN-VAE + init	15	335.9 (13.9)	67.0
MCNN-LM	63	338.3	69.1
MCNN-VAE	63	336.2 (11.8)	67.3
MCNN-VAE + init	63	334.6 (12.6)	66.0
LCNN-LM	125	335.4	66.6
LCNN-VAE	125	333.9 (6.7)	65.4
LCNN-VAE + init	125	**332.1 (10.0)**	**63.9**
VLCNN-LM	187	336.5	67.6
VLCNN-VAE	187	336.5 (0.7)	67.6
VLCNN-VAE + init	187	335.8 (3.8)	67.0

(a) Yahoo

Model	Size	NLL (KL)	PPL
LSTM-LM	$< i$	362.7	42.6
LSTM-VAE**	$< i$	372.2 (0.3)	47.0
LSTM-VAE** + init	$< i$	368.9 (4.7)	46.4
SCNN-LM	15	371.2	46.6
SCNN-VAE	15	365.6 (9.4)	43.9
SCNN-VAE + init	15	363.7 (10.3)	43.1
MCNN-LM	63	366.5	44.3
MCNN-VAE	63	363.0 (6.9)	42.8
MCNN-VAE + init	63	360.7 (9.1)	41.8
LCNN-LM	125	363.5	43.0
LCNN-VAE	125	361.9 (6.4)	42.3
LCNN-VAE + init	125	**359.1 (7.6)**	**41.1**
VLCNN-LM	187	364.8	43.7
VLCNN-VAE	187	364.3 (2.7)	43.4
VLCNN-VAE + init	187	364.7 (2.2)	43.5

(b) Yelp

Table 6.2: Language modeling results on the test set. ** is from [4]. We report negative log likelihood (NLL) and perplexity (PPL) on the test set. The KL component of NLL is given in parentheses. Size indicates the effective filter size. VAE + init indicates pretraining of only the encoder using an LSTM LM.

Chapter 7

Proposed work

7.1 Unsupervised Natural Language Learning

To continue the line of work of unsupervised learning, I would like to investigate the natural language generation problem. Previously natural language generation mainly relies on language models. Recurrent neural networks (RNNs) have achieved the state of the art in language modeling. However, RNN alone lacks the ability to generate languages with desired attributes. Generating sentences with desired attributes are useful for many applications. For example, in review generations, we would like to condition the generation on the ratings. For dialogue modeling, the response is generated conditioning on given intents.

Variational auto-encoders are potential solutions to generate sentences with desired attributes. But there are some inherent limitations with variational autoencoders for text modeling, as mentioned in Chapter 6. [15] is a preliminary work in this direction. It combines language modeling with generative adversarial network [12]. The generated sentences are fed to a discriminator and we can directly use backpropagation to ensure that sentences have certain attributes. Based on the success, we would like to investigate style transfer for text, dialogue modeling with intent etc.

7.2 Timeline

- Now - Feb 2018: work on natural language generation.
- Feb 2018 - May 2018: wrap up work into a final thesis.
- June 2018: thesis defense.

Bibliography

[1] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. *arXiv preprint arXiv:1505.00468*, 2015. 4.2.1, 4.2.2, **??**

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL `http://arxiv.org/abs/1409.0473`. 5.1

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. 4

[4] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015. 6.1, 6.2, 6.4.2, 6.2

[5] Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. 2015. 3.3.2

[6] Maxwell D. Collins and Pushmeet Kohli. Memory bounded deep convolutional networks. Technical report, University of Wisconsin-Madison, 2014. 3.3.2

[7] B. Dai, B. Xie, N. He, Y. Liang, A. Raj, M. Balcan, and L. Song. Scalable kernel methods via doubly stochastic gradients. 2014. **??**, 3.1, 3.3.1

[8] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. pages 2148–2156, 2013. 3.2, 3.3.2

[9] A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, 2001. 2.2.2

[10] Hao Fang, Saurabh Gupta, Forrest Iandola, Rupesh Srivastava, Li Deng, Piotr Dollár, Jianfeng Gao, Xiaodong He, Margaret Mitchell, John Platt, et al. From captions to visual concepts and back. *arXiv preprint arXiv:1411.4952*, 2014. 4.1.2, 4.2.1

[11] Haoyuan Gao, Junhua Mao, Jie Zhou, Zhiheng Huang, Lei Wang, and Wei Xu. Are you talking to a machine? dataset and methods for multilingual image question answering. *arXiv preprint arXiv:1505.05612*, 2015. 4.1.1

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 7.1

[13] 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. 6.3, 6.3

[14] Matthew Henderson, Blaise Thomson, and Jason Williams. Dialog state tracking challenge 2 & 3, 2014. 5.1

[15] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. Controllable text generation. *arXiv preprint arXiv:1703.00955*, 2017. 7.1

[16] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *ArXiv*, 1408.5093, 2014. 3.3

[17] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016. 6.3, 6.3

[18] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014. 4.1.2

[19] G. S. Kimeldorf and G. Wahba. A correspondence between Bayesian estimation on stochastic processes and smoothing by splines. *Annals of Mathematical Statistics*, 41:495–502, 1970. 2.1.1

[20] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 6.1, 6.1

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. pages 1106–1114, 2012. 1, 3.3

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 4.1.1, 6.3

[23] Q.V. Le, T. Sarlos, and A. J. Smola. Fastfood — computing hilbert space expansions in loglinear time. In *International Conference on Machine Learning*, 2013. 2.1.2, 2.2.2

[24] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014*, pages 740–755. Springer, 2014. 4.2.1

[25] Lin Ma, Zhengdong Lu, and Hang Li. Learning to answer questions from image using convolutional neural network. *arXiv preprint arXiv:1506.00333*, 2015. 4.1.1, 4.2.2

[26] Mateusz Malinowski and Mario Fritz. A multi-world approach to question answering about real-world scenes based on uncertain input. In *Advances in Neural Information Processing Systems*, pages 1682–1690, 2014. 4.2.2

[27] Mateusz Malinowski, Marcus Rohrbach, and Mario Fritz. Ask your neurons: A neural-based approach to answering questions about images. *arXiv preprint arXiv:1505.01121*, 2015. 4.2.2

[28] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philos. Trans. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci.*, A 209:

415–446, 1909. 2.1.1, 2.1.1

[29] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010. 6.1

[30] Radford M. Neal. Assessing relevance determination methods using delve. In C. M. Bishop, editor, *Neural Networks and Machine Learning*, pages 97–129. Springer, 1998. 2.2.3, 2.3

[31] A. Rahimi and B. Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. 2009. 2.1.2, 2.1.2

[32] Mengye Ren, Ryan Kiros, and Richard Zemel. Exploring models and data for image question answering. *arXiv preprint arXiv:1505.02074*, 2015. 4.1.1, 4.2.2

[33] Tara N. Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. pages 6655–6659, 2013. 3.2, 3.3.2

[34] I. Schoenberg. Positive definite functions on spheres. *Duke Math. J.*, 9:96–108, 1942. 2.1.2

[35] B. Schölkopf, R. Herbrich, and A. J. Smola. A generalized representer theorem. In D. P. Helmbold and B. Williamson, editors, *Proc. Annual Conf. Computational Learning Theory*, number 2111 in Lecture Notes in Comput. Sci., pages 416–426, London, UK, 2001. Springer-Verlag. 2.1.1

[36] Iulian V Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, 2016. 5.1

[37] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Gregoire Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 101–110. ACM, 2014. 4.1.2

[38] B. W. Silverman. *Density Estimation for Statistical and Data Analysis*. Monographs on statistics and applied probability. Chapman and Hall, London, 1986. 2.2.1

[39] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 4.1.1

[40] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014. 1, 4.1.2

[41] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014. 4.1.1

[42] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *EMNLP*, pages 1422–1432, 2015. 6.4.1

[43] Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information*

Processing Systems, pages 4790–4798, 2016. 6.3

[44] A. G. Wilson and R. P. Adams. Gaussian process kernels for pattern discovery and extrapolation. In *Proceedings of the 30th International Conference on Machine Learning*, 2013. 2.2.1

[45] Kelvin Xu, Jimmy Ba, Ryan Kiros, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015. 4

[46] Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. Deep fried convnets. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1476–1483, 2015. 1

[47] Zichao Yang, Andrew Wilson, Alex Smola, and Le Song. A la carte–learning fast kernels. In *Artificial Intelligence and Statistics*, pages 1098–1106, 2015. 1

[48] Zichao Yang, Phil Blunsom, Chris Dyer, and Wang Ling. Reference-aware language models. *arXiv preprint arXiv:1611.01628*, 2016. 1

[49] Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alex Smola. Stacked attention networks for image question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 21–29, 2016. 1

[50] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of NAACL-HLT*, pages 1480–1489, 2016. 6.4.1

[51] Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, and Taylor Berg-Kirkpatrick. Improved variational autoencoders for text modeling using dilated convolutions. *arXiv preprint arXiv:1702.08139*, 2017. 1

[52] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? pages 3320–3328. 2014. 3.3.1

[53] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015. 6.3

[54] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *arXiv preprint arXiv:1509.01626*, 2015. 6.4.1