Pareto-Efficient Training of Multi-Task Deep Networks

THESIS

Submitted in Partial Fulfillment of

the Requirements for

the Degree of

MASTER OF SCIENCE (Electrical Engineering)

at the

NEW YORK UNIVERSITY

TANDON SCHOOL OF ENGINEERING

by

Alp Aygar

September, 2019

Pareto-Efficient Training of Multi-Task Deep Networks

THESIS

Submitted in Partial Fulfillment of
the Requirements for
the Degree of

MASTER OF SCIENCE (Electrical Engineering)

at the

NEW YORK UNIVERSITY
TANDON SCHOOL OF ENGINEERING

by

Alp Aygar

September, 2019

Approved:

_____

Advisor Signature

_____

Date

_____

Department Chair Signature

_____

Date

University ID: N14049800
Net ID: aa3250

Vitae

Alp Aygar was born in Ankara, Turkey. He received his Bachelor's degree at his hometown from Bilkent University in Electrical and Electronics Engineering in June 2016. During his studies he was awarded Merit scholarship. After graduation he has been engaged in the Master of Science in Electrical Engineering at New York University, Tandon School of Engineering in Brooklyn, New York. In January 2017 he started working as a graduate asistant working on problems related to machine learning, computer vision and medical imaging. He did an internship at Apple Inc. where he worked on Deep Learning applications in computer vision.

Acknowledgements

First and foremost, I would like to thank my advisor Prof. Yao Wang whose kind support, valuable insight and vast knowledge has been a brilliant guide to me through the path of academic research and life.

I am grateful to Prof. Elza Erkip and Prof. Siddharth Garg for devoting their time to serve on my committee.

I have the sincerest gratitude for Chuanmin Jia, who worked on this project with me. Without the path he paved before me, this project would not have been possible.

I express my gratitude for now graduated senior PhD members of our lab, Shervin Minaee and Fanyi Duanmu for their experienced help and guidance in introducing me the ways of academic research. I thank my peers at the NYU Video Lab; Valentin, Amir, Jack, Ziming and Nitin for providing a positive, fun and supportive lab environment.

I was fortunate to have a family that never failed to make their support felt from half a world away.

I thank my uncle, Soner for showing me first hand the alternative paths in life. Finally, my friends scattered across the world pursuing degrees of their own, who have found time to spare a piece of their mind whenever I asked.

for Nazlı, without whom this thesis would have never started

Pareto-Efficient Training of Multi-Task Deep Networks

ABSTRACT

Pareto-Efficient Training of Multi-Task Deep Networks
by
Alp Aygar

Advisor: Prof. Yao Wang

Submitted in Partial Fulfillment of the Requirements for
the Degree of Master of Science (Electrical Engineering)

by

Alp Aygar

September, 2019

Multi-task training of neural networks potentially yield better performance over training for the individual tasks while resulting in computationally more efficient structures than separate networks. Effective optimization methods for multi-task network is an ongoing area of research. Multiple-gradient descent algorithm (MGDA) is a good candidate for training networks with shared parameters. It finds solutions on the Pareto frontier by taking gradient descent steps in Pareto optimal directions. In this work we build upon the theoretical work done by Peitz et al. [1] who introduced an inexact version of MGDA that conditionally updates the parameters based on the gradient uncertainties. In deep learning uncertainty in gradients arise from minibatch sampling. We simplify the inexact MGDA and propose a version that is well-suited for training multi-task deep networks. We further introduce an approximation for estimating gradient inexactness through

training and propose a novel adaptive MGDA that updates the inexactness bound as the training progresses. We test the new algorithm on three separate multi-task training networks and compare its performance with the classical MGDA, single task training and optimizing jointly for the weighted sum of losses associated with multiple tasks.

Contents

List of Figures

Chapter 1

Multi-Objective Optimization

A multi-objective optimization problem can be expressed in the form

$$\min_{\substack{\theta^{\text{sh}} \\ \theta^1,\ldots,\theta^k}} \{\mathcal{L}^i(\theta^{sh}, \theta^i)\}_{i=1}^k \tag{1.1}$$

where we try to jointly optimize the parameter set $\theta = \theta^{sh} \cup \bigcup_{i=1}^k \theta^i$ for k loss functions $\mathcal{L}^i : \mathbb{R}^{n_i} \to \mathbb{R}$. θ^{sh} denotes the set of parameters that is shared by all the loss functions and every loss function indexed i has its own set of parameters θ^i. In deep learning literature jointly optimized objectives are thought to provide better generalized solutions for every objective compared to individual optimizations of the objectives [3]. There are two explanations given for this phenomenon; (1) the knowledge learned by a single task can be leveraged by other tasks and (2) tasks can behave as regularization constraints for each other which enforce the network to learn more generic features [3]. However better generalization may not be gained by joint optimization when the tasks are competing. Often times it is difficult to distinguish competing and cooperating tasks and this distinction is not explored in this work. Another motivation for multi-objective optimization (MOP) in deep learning is by using a shared set of parameters for multiple tasks one can save memory and time when deploying the tasks. An important concept in MOP is the

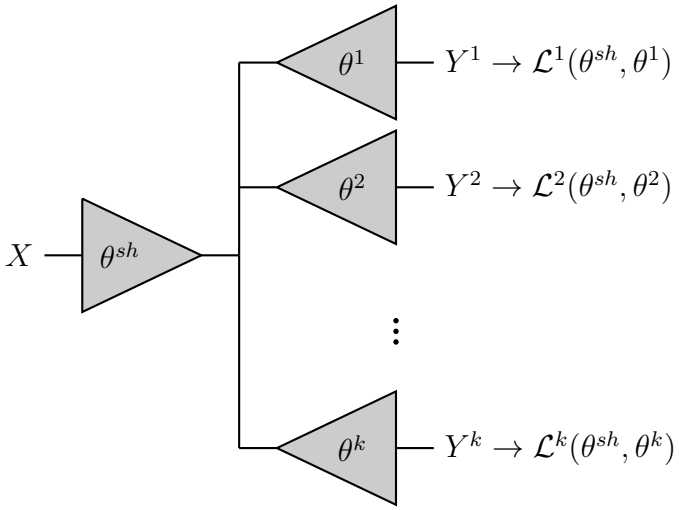


Figure 1.1: A common deep learning framework for multi-objective optimization problems in the form of equation (1.1) is shared encoder multi decoder structure. A single input X is fed through a shared encoder which extracts generic features for all tasks and a separate output Y^i is generated for each task through the decoders. The network is jointly optimized for loss functions $\mathcal{L}^i, i = 1, \ldots, k$.

Pareto optimality.

Definition 1. Pareto Optimal Set

- A point $\theta^* \in \mathbb{R}^n$ dominates the point $\theta \in \mathbb{R}^n$ if $\mathcal{L}^i(\theta^*) \leq \mathcal{L}^i(\theta)$ for all $i \in \{1, \ldots k\}$ and $\mathcal{L}^i(\theta^*) < \mathcal{L}^i(\theta)$ for at least one $i \in \{1, \ldots k\}$.
- A point $\theta^* \in \mathbb{R}^n$ is a Pareto optimal point if there exists no point $\theta \in \mathbb{R}^n$ that dominates θ^*.
- The set of non-dominated points is called a Pareto optimal set $\mathcal{P}_s$ and its image is called the Pareto front $\mathcal{P}_F$.

It is often desired for a solution to the MOP be a Pareto optimal point; otherwise we could find another solution that is better off in at least one task

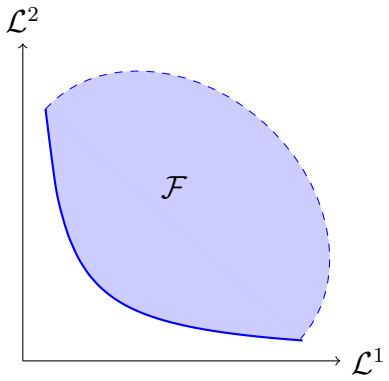


Figure 1.2: The bold line denotes the Pareto frontier for the 2-task optimization problem. $\mathcal{F}$ denotes the feasibility set.

without being worse off in any other task. A common method in deep learning for finding a Pareto optimal solution is optimizing all the parameters jointly for a single objective obtained by the weighted sum of the other objectives.

$$\min_{\substack{\theta^{\text{sh}} \\ \theta^1,\ldots,\theta^k}} \sum_{i=1}^{k} \lambda_i \mathcal{L}^i(\theta^{sh}, \theta^i) \tag{1.2}$$

where $\lambda_i > 0$ for $i \in 1, \ldots, k$. Ideally one can approximate the Pareto frontier by finding the optimal solution for every point in the λ space. However there are two major drawbacks to this method (1) the optimal solution distribution is not uniform, and (2) optimal solutions in non-convex regions of Pareto frontier are not detected [4]. Both of these drawbacks are more apparent for deep learning applications where the objective functions and Pareto frontiers are non-convex with a high dimensional parameter space and gradient descent is used to obtain the solutions. The most common optimization techniques in deep learning applications involve using gradient descent variant algorithms; in which a descent direction in the parameter space is determined by the gradient with respect to the objective function. For the case of weighted sum of losses in equation (1.2), this direction is the weighted sum of individual

gradients

$$q(\theta) = -\sum_{i=1}^{k} \lambda_i \nabla \mathcal{L}^i(\theta) \tag{1.3}$$

Then the parameters are updated by the gradient descent update rule

$$\theta \leftarrow \theta + \eta q(\theta) \tag{1.4}$$

where η denotes the learning rate.

Lemma 1. *For a sufficiently small learning rate η, a descent direction $q(\theta)$ will be non-increasing for objective function $\mathcal{L}^i : \mathbb{R}^{n_i} \to \mathbb{R}$ iff $q(\theta) \cdot \nabla \mathcal{L}^i(\theta) \leq 0$*

Proof. Using the first order Taylor approximation around the updated parameters

$$\begin{aligned} \mathcal{L}^i(\theta + \eta q(\theta)) &= \mathcal{L}^i(\theta) + \eta q(\theta) \cdot \nabla \mathcal{L}^i(\theta) + \epsilon_i \|\eta q(\theta)\|_2^2 && (1.5) \\ &= \mathcal{L}^i(\theta) + \eta (q(\theta) \cdot \nabla \mathcal{L}^i(\theta) + \epsilon_i \|\eta q(\theta)\|_2^2) && (1.6) \\ &\leq \mathcal{L}^i(\theta) && (1.7) \end{aligned}$$

where $\epsilon_i \to 0$ as $\eta \to 0$. Equation (1.7) holds iff

$$q(\theta) \cdot \nabla \mathcal{L}^i(\theta) + \epsilon_i \eta \|q(\theta)\|_2^2 \leq 0 \tag{1.8}$$

Since $\epsilon \|q(\theta)\|_2^2 \geq 0$ it must hold that $q(\theta) \cdot \nabla \mathcal{L}^i(\theta) \leq 0$ and for a sufficiently small η there is an ϵ satisfying $-q(\theta) \cdot \nabla \mathcal{L}^i(\theta) \geq \epsilon_i \eta \|q(\theta)\|_2^2$. □

Notice that lemma 1 holds for any function, including non-convex functions which is often the case in deep learning optimization. Based on this observation we can define a Pareto optimal descent direction as

Definition 2. A descent direction $q(\theta)$ is a Pareto optimal descent direction when $-q(\theta) \cdot \mathcal{L}^i(\theta) \geq 0, \forall i \in \{1, \ldots, k\}$

Notice that the solution to the optimization problem is Pareto optimal when weighted sum of losses is optimized as in equation (1.2) [4] however when gradient descent is used to obtain the solution, there are drawbacks that may prevent us from getting to a Pareto optimal point when gradient descent is used with the weighted sum of losses. In this work we address two of these drawbacks (1) gradient descent direction is not Pareto optimal ($-\nabla\mathcal{L}^j(\theta)\cdot\sum_{i=1}^{k}\lambda_i\nabla\mathcal{L}^i(\theta)\geq 0$ is not necessarily true for all $i,j\in\{1,\ldots,k\}$) and (2) the gradients are sampled by mini-batching the dataset which causes inexactness. A popular method to address the first drawback is the Multiple Gradient Descent Algorithm (MGDA) developed by Désideri et al. [5]. It was applied in deep learning setting by Şener et al. [2]. The second drawback is addressed by a modified version of this algorithm using inexact gradients. The theoretical work for the inexact-MGDA has been published by Peitz et al. [1]. Our contributions are (1) experimentally applying this algorithm in a deep learning setting and (2) formulating an empirical method for inexactness estimation.

Chapter 2

Multiple Gradient Descent Algorithm

2.1 Standard MGDA

Each gradient vector defines a half-space by the hyperplane perpendicular to the gradient vector. The cone formed by the intersection of the half-spaces across the gradients that satisfy $-q(\theta) \cdot \nabla \mathcal{L}^i(\theta) \geq 0$ is the cone of Pareto optimal descent directions. Thus, if a single Pareto optimal descent direction exists then there must be a cone in which infinitely many more directions lie. If the Pareto optimal descent direction does not exist then the intersection of the half-spaces is an empty region. Goal of MGDA is to find the common steepest descent direction [5]. An auxiliary optimization problem is suggested for finding the common steepest direction

$$\min_{\alpha \in \mathbb{R}^k} \left\{ \left\| \sum_{i=1}^{k} \alpha_i \nabla \mathcal{L}^i(\theta) \right\|_2^2 \;\middle|\; \alpha_i \geq 0, i = 1, \ldots, k, \sum_{i=1}^{k} \alpha_i = 1 \right\} \tag{2.1}$$

$$q(\theta) = -\sum_{i=1}^{k} \alpha_i \nabla \mathcal{L}^i(\theta) \tag{2.2}$$

Following the derivations from [2] we first derive the solution for $k = 2$ tasks case.

$$\min_{\alpha} \|\alpha\nabla\mathcal{L}^1(\theta) + (1-\alpha)\nabla\mathcal{L}^2(\theta)\|_2^2 \quad \alpha \in [0,1] \tag{2.3}$$

$$\alpha = \text{clip}_{[0,1]}\left[\frac{(\nabla\mathcal{L}^2(\theta) - \nabla\mathcal{L}^1(\theta))^{\mathrm{T}}\nabla\mathcal{L}^2(\theta)}{\|\nabla\mathcal{L}^1(\theta) - \nabla\mathcal{L}^2(\theta)\|_2^2}\right] \tag{2.4}$$

In order to find the solution for a higher number of tasks we follow the

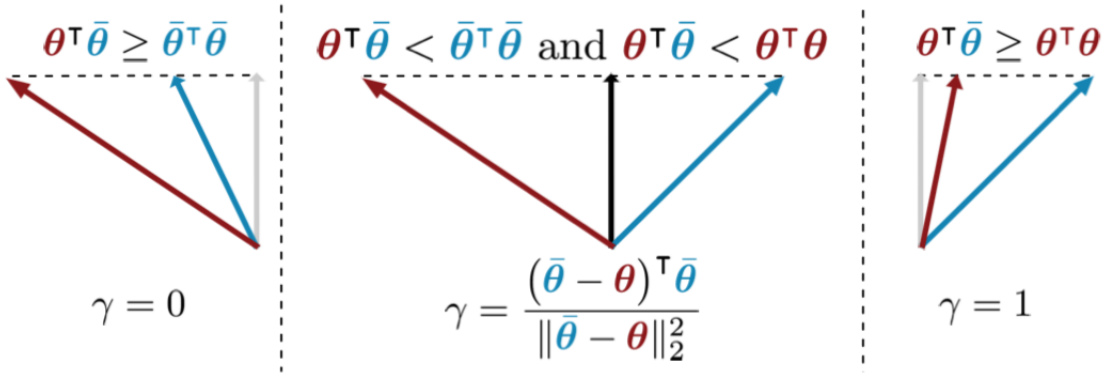


Figure 2.1: Figure by Şener et al. [2] illustrating MGDA solution in 2D with 2 tasks. Their notation matches us when $\gamma = \alpha$, $\theta = \nabla\mathcal{L}^1(\theta)$ and $\hat{\theta} = \nabla\mathcal{L}^2(\theta)$.

procedure suggested in [2]. Define matrix M

$$\mathbf{M}_{i,j} = \nabla\mathcal{L}^i(\theta)^{\mathrm{T}}\nabla\mathcal{L}^j(\theta) \tag{2.5}$$

then we can rewrite the optimization objective in vector quadratic form

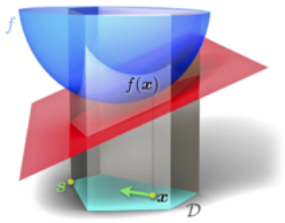


Figure 2.2: Frank-Wolfe algorithm visualized for a two variable function. Figure by Stephanie Stutz from wikimedia under creative commons license.

$$\begin{aligned} &\min_{\alpha \in \mathbb{R}^k} \alpha^{\mathrm{T}} \mathbf{M} \alpha \\ &\text{subject to } \sum_{i=1}^{k} \alpha_i = 1 \\ &\qquad \alpha_i \geq 0 \, \forall i \in \{1, \ldots, k\} \end{aligned} \tag{2.6}$$

These type of contrained convex optimization problems can be solved by Frank-Wolfe iteration solver [6]. Algorithm iteratively takes a linear approximation of the objective function until convergence. Frank-Wolfe solves problems in the following form for a convex function $f(x)$.

$$\begin{aligned} &\min_{x} f(x) \\ &\text{subject to } x \in \mathcal{D} \end{aligned} \tag{2.7}$$

for our problem $x \leftarrow \alpha$, $f(\alpha) \leftarrow \alpha^T \mathbf{M} \alpha$ and $\mathcal{D} = \{\alpha | \sum \alpha_i = 1, \ \alpha_i \geq 0\}$.

procedure FRANK-WOLFE($\mathbf{M}$)
  initialize $\{\alpha_1, \ldots, \alpha_k\} \leftarrow \{1/k, \ldots, 1/k\}$
  while α not converged **do** ▷ Check for convergence
    $\hat{t} = \text{argmin}_r \sum_t \alpha_t \mathbf{M}_{rt}$ ▷ find direction
    $\gamma = \text{argmin}_\gamma((1-\gamma)\alpha + \gamma e_{\hat{t}})^{\mathrm{T}} \mathbf{M}((1-\gamma)\alpha + \gamma e_{\hat{t}})$ ▷ find step size with line search, two task solution in equation (2.4)
    $\alpha \leftarrow (1-\gamma)\alpha + \gamma e_{\hat{t}}$ ▷ update parameters
  end while
  return α
end procedure

The gradient descent algorithm is updated using the Frank-Wolfe algorithm.

procedure MULTIPLE GRADIENT DESCENT ALGORITHM($\theta, \mathcal{L}$)
  while θ not converged **do** ▷ Check for convergence
    for $i \in \{0, \ldots, k\}$ **do**
      for $j \in \{0, \ldots, i\}$ **do**
        $\mathbf{M}_{ij} \leftarrow \nabla\mathcal{L}^i(\theta) \cdot \nabla\mathcal{L}^j(\theta)$
        $\mathbf{M}_{ji} \leftarrow \mathbf{M}_{ij}$ ▷ Sample gradients and form $\mathbf{M}$
      end for
    end for
    $\alpha \leftarrow$ FRANK-WOLFE($\mathbf{M}$)
    $\theta \leftarrow \theta - \eta \sum_{i=1}^{k} \alpha_i \nabla\mathcal{L}^i(\theta)$
  end while
end procedure

2.2 Inexact MGDA

Gradient descent used in deep learning results in inexact gradients as the data is fed in random batches. Empirical evidence suggest that models trained with smaller batches usually achieve better generalization [7]. We expect the inexactness of the gradients to grow as batchsize gets smaller. This

leads to confusion about whether it is desired to rid the models of all the inexactness that arise from sampling batches. We believe MGDA grants a unique opportunity to study the effect of inexact batch gradients because the existence of multiple gradients can be utilized to perform inexactness-aware multiple-gradient descent. The theoretical foundations of an inexact MGDA has been studied by Peitz et al. [1]. In this section we introduce a simplified version of their algorithm that well-suited for deep learning applications. Assume the gradients sampled for every task contains a bounded error

$$\|\nabla \mathcal{L}^i(\theta) - \nabla \tilde{\mathcal{L}}^i(\theta)\|_2 \leq \epsilon_i \tag{2.8}$$

Exact gradient $\nabla \mathcal{L}(\theta)$ is the gradient of whole epoch and inexact gradient

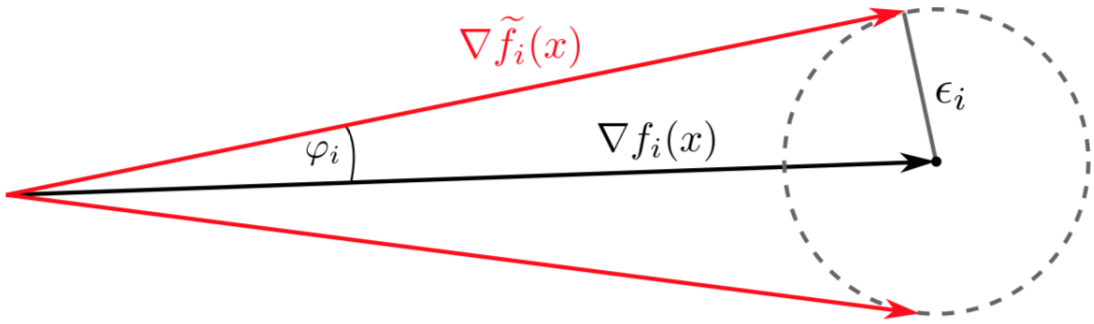


Figure 2.3: Figure by Peitz et al. [1]. $f_i = \mathcal{L}^i$ for our notation.

$\nabla \tilde{\mathcal{L}}(\theta)$ is the gradient of the batch. In the following sections we will discuss various methods for ϵ_i estimation, but for the theoretical discussions in this section we assume the value to be known. The angle of inexactness φ_i can be defined geometrically

$$\varphi_i = \arcsin\left(\frac{\epsilon_i}{\|\nabla \mathcal{L}^i(\theta)\|_2}\right) \tag{2.9}$$

Going back to the Pareto optimal descent direction from definition (1) that defines a cone when multiple gradients are introduced $Q = \{q(\theta) : -q(\theta) \cdot$

$\mathcal{L}^i(\theta) \geq 0, i \in \{1, \ldots, k\}\}$. This cone can be updated when inexactness is introduced Angle $gamma_i$ is the angle from the descent direction to the

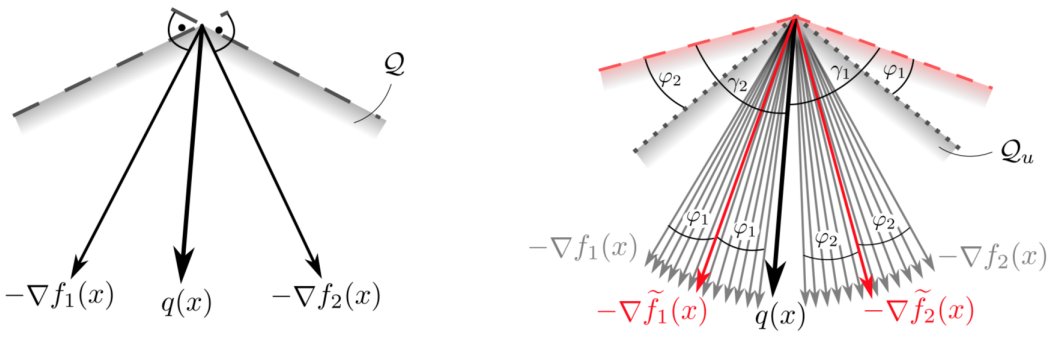


Figure 2.4: Figure by Peitz et al. [1] (a) the exact cone Q defined by gradients in MGDA (b) illustrating the updated cone Q_u when inexactness is taken into account.

updated cone boundary hyperplane introduced by gradient i. Geometrically we can define it as

$$\gamma_i = \frac{\pi}{2} - \arccos \frac{q(\theta) \cdot (-\nabla \mathcal{L}^i(\theta))}{\|q(\theta)\|_2 \|\nabla \mathcal{L}^i(\theta)\|_2} \tag{2.10}$$

Based on this we extend the Pareto optimal descent direction rule such that for every task

$$\gamma_i \geq \varphi_i \geq 0, \ i = 1, \ldots, k \tag{2.11}$$

Assume we apply the original MGDA to come up with the set of α_i and descent direction $q(\theta) = \sum_{i=1}^{k} \alpha_i \nabla \mathcal{L}^i(\theta)$. Then we can use the definitions of γ_i and φ_i in equations (2.10) and (2.9) and check if this descent direction is Pareto optimal descent using the updated rule in equation (2.11). Plugging the definitions in, after some algebraic operations we can write the rule in

terms of α_i.

$$\alpha_i \geq \alpha_{min,i} =: \frac{1}{\|\nabla \mathcal{L}^i(\theta)\|} \left(\|q(\theta)\|_2 \epsilon_i - \sum_{\substack{i \neq j \\ j=1}}^{k} \alpha_j \big(\nabla \mathcal{L}^j(\theta) \cdot \nabla \mathcal{L}^i(\theta) \big) \right) \tag{2.12}$$

For the MGDA descent direction to be Pareto optimal we require every α_i yielded by Frank-Wolfe algorithm to be greater than the α_{min} defined in equation (2.12). The inexact-MGDA is proposed as doing regular MGDA while checking the condition of equation (2.12) at every step and skipping the updates that do not satisfy this condition.

procedure INEXACT-MGDA($\theta, \mathcal{L}, \epsilon$)
    Sample gradients $\nabla \mathcal{L}(\theta)$ and form $\mathbf{M}$
    $\alpha \leftarrow$ FRANK-WOLFE($\mathbf{M}$)
    Calculate $\alpha_{min,i}$ using α_i and ϵ_i in equation (2.12)
    if $\alpha_i \geq \alpha_{min,i}, i = 1, \ldots, k$ **then**
        $\theta \leftarrow \theta - \eta \sum_{i=1}^{k} \alpha_i \nabla \mathcal{L}^i(\theta)$ ▷ do descent update
    end if
    return θ
end procedure

Inexact-MGDA requires an upper bound on inexactness ϵ_i to be defined for each task. In the following sections we discuss how to estimate this value for deep learning problems.

2.3 Inexactness Upper Bound Estimation

ϵ_i can be regarded as a new hyperparameter introduced by inexact MGDA: $\epsilon_i \geq \|\nabla \mathcal{L}^i(\theta) - \nabla \tilde{\mathcal{L}}^i(\theta)\|_2$ that quantifies the uncertainity of the gradients. We make some speculations about some expected properties of this new hyperparameter ϵ.

- Very large ϵ_i definitely satisfies the definition but results in large $\alpha_{min,i}$ which causes MGDA to skip most of the updates. We only want to

skip very uncertain updates

- Very small ϵ_i makes the uncertainity analysis dsyfunctional: $\alpha_{min,i} \to 0$ as $\epsilon_i \to 0$ so inexact MGDA becomes same with regular MGDA
- Optimum ϵ_i value probably changes through training as uncertainity is different when parameters are near or far from convergence
- Optimum ϵ_i value should be closely related to the batchsize, as less inexactness is expected from larger batches. Note that small inexactness may not always be a desired property in deep networks as previous works suggest that smaller batches usually yield better generalization [7]

Based on these speculations we propose to estimate and update uncertainty ϵ_i after each epoch. Our goal is to quantify uncertainty in the gradients during an epoch say e and use that uncertainty to make an estimate on ϵ_i on epoch $e+1$. Assume we label the minibatches contained within an epoch with index p where N is the number of all minibatches. Let the gradient of task i in minibatch p as $\nabla\tilde{\mathcal{L}}_i^p$ and true epoch gradient as $\nabla\mathcal{L}_i^E$.
By definition the gradient error in a epoch can be determined by ϵ_i

$$\epsilon_i^2 = (\frac{1}{N}\sum_p \|\nabla\tilde{\mathcal{L}}_i^p - \nabla\mathcal{L}_i^E\|_2^2) \tag{2.13}$$

In reality we do not know the true epoch gradient as we perform parameter update after each minibatch based on the minibatch gradient. Here we propose to simply use the average of the minibatchg gradients, $\mu_{\nabla L_i}$

$$\mu_{\nabla L_i} \approx \frac{1}{N}\sum_p \nabla\tilde{\mathcal{L}}^p \tag{2.14}$$

So we update ϵ_i using

$$\hat{\epsilon}^2 = \frac{1}{N}\sum_p (\|\nabla\tilde{\mathcal{L}}^p - \mu_{\nabla\mathcal{L}}\|_2^2) \tag{2.15}$$

$$= \frac{1}{N}\sum_p \nabla\tilde{\mathcal{L}}^p{}^{\mathrm{T}}\nabla\tilde{\mathcal{L}}^p - \mu_{\nabla\mathcal{L}}{}^{\mathrm{T}}\mu_{\nabla\mathcal{L}} \tag{2.16}$$

Notice that $\frac{1}{N}\sum_p(\nabla\tilde{\mathcal{L}}^{\mathrm{T}}\tilde{\nabla\mathcal{L}})$ term is a scalar and can easily be tracked with $O(1)$ memory cost. The term $\mu_{\nabla\mathcal{L}}{}^{\mathrm{T}}\mu_{\nabla\mathcal{L}}$ requires $\mu_{\nabla\mathcal{L}}$ to be updated at every gradient descent update so it induces an extra memory cost of $O(P)$ for each task where P is the total number of parameters in the model. However this mean should be tracked for every task so the total increased cost is $O(kP)$. This increase in cost is comparable to popular optimizers like Adam and Momentum, where moving averages of gradients are kept through training. Note that the pseudo-mean gradient introduced here is closely related to the momentum terms used in popular optimizers like Adam, with a key main difference. The pseudo-mean is taken over a single epoch whereas the momentum terms are calculated by a running weighted average over the whole training session.

Finally a new hyperparameter scaling factor $\epsilon_i = \lambda\hat{\epsilon}_i$. Our experiments have shown that without the scaling factor the estimated inexactness is usually an order of magnitude larger than the optimum ϵ_i value obtained by training the network with a different ϵ_i that remains constant throughout training session. Putting these together we introduce a novel algorithm adaptive inexact-MGDA.

procedure ADAPTIVE-ϵ-MGDA($\theta, \mathcal{L}, \lambda$)
    Initialize $\epsilon_i \leftarrow 0, i = 1, \ldots, k$
    while θ not converged **do**
        $\mu_{\nabla\mathcal{L}^i} \leftarrow 0, i = 1, \ldots, k$ ▷ Adds memory cost $O(kP)$ where P is total size of network
        $\mu_{\nabla\mathcal{L}^{i\mathrm{T}}\nabla\mathcal{L}^i} \leftarrow 0$ ▷ Only a scalar, adds memory cost $O(k)$ which is negligible
        $t \leftarrow 0$
        while epoch has not ended **do**
            Sample gradients $\nabla\mathcal{L}^i$
            $\mu_{\nabla\mathcal{L}^i} \leftarrow \mu_{\nabla\mathcal{L}^i} + \nabla\mathcal{L}^i, i = 1, \ldots, k$
            $\mu_{\nabla\mathcal{L}^{i\mathrm{T}}\nabla\mathcal{L}^i} \leftarrow \mu_{\nabla\mathcal{L}^{i\mathrm{T}}\nabla\mathcal{L}^i} + \nabla\mathcal{L}^{i\mathrm{T}}\nabla\mathcal{L}^i, i = 1, \ldots, k$
            INEXACT-MGDA($\theta, \nabla\mathcal{L}, \epsilon$)
            $t \leftarrow t + 1$
        end while
        $\mu_{\nabla\mathcal{L}^i} \leftarrow \mu_{\nabla\mathcal{L}^i}/t, i = 1, \ldots, k$
        $\mu_{\nabla\mathcal{L}^{i\mathrm{T}}\nabla\mathcal{L}^i} \leftarrow \mu_{\nabla\mathcal{L}^{i\mathrm{T}}\nabla\mathcal{L}^i}/t, i = 1, \ldots, k$
        $\epsilon_i^2 \leftarrow \lambda(\mu_{\nabla\mathcal{L}^{i\mathrm{T}}\nabla\mathcal{L}^i} - \mu_{\nabla\mathcal{L}^i}{}^T\mu_{\nabla\mathcal{L}^i}), i = 1, \ldots, k$
    end while
end procedure

An alternative to using an adaptive ϵ estimation is to simply use a constant throughout training, which means is regarded as a hyperparameter to be tuned.

We compare the adaptive inexact MGDA with the constant inexact MGDA and the original MGDA without considering inexactness in the experimentation section.

Chapter 3

Experimental Results

We setup 3 different multi-task networks to compare the suggested algorithms. Two of these networks are directly following the work of Sener et al. [2] where we do (1) MNIST overlapping digit estimation and (2) joint semantic segmentation, instance segmentation and depth estimation. For the third task we introduce a joint saliency and compression network. All of these tasks follow the single encoder multi decoder architecture in the form of Figure 1.1.

3.1 MNIST Overlapping Digit Classification

The goal of the network is to separately classify overlapping digits in a single image. The images are created by sampling two random digits form the MNIST dataset and pasting one on the upper left and one on the lower right of the image. A shared encoder and duo-decoder architecture is used where one decoder tries to classify upper left and the other decoder tries to classify the lower right digit. Decoders are fully connected layers with 10 unit sigmoid outputs. MNIST dataset with 10000 28x28 images are used for generating

Figure 3.1: Sample input images to be classified by the network.

the images. Cross entropy loss is used to train the network [8].

$$\mathcal{L} = -\log(p_t) \tag{3.1}$$

where

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{if } y = 0 \end{cases} \tag{3.2}$$

and y indicates whether the digit's class corresponds to the one-hot vector location. Network is trained over 100 epochs with learning rate 0.001 using

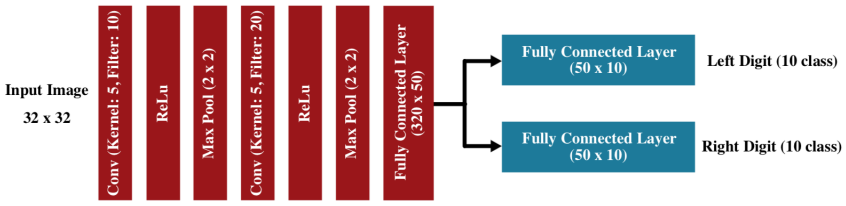


Figure 3.2: MNIST network by Şener et al. [2].

Nesterov's momentum 0.9.

3.1.1 Results

We evaluate the left and right digit accuracies on 5000 validation images.

We observe in figure (3.3) that for most values of ϵ classical MGDA outper-

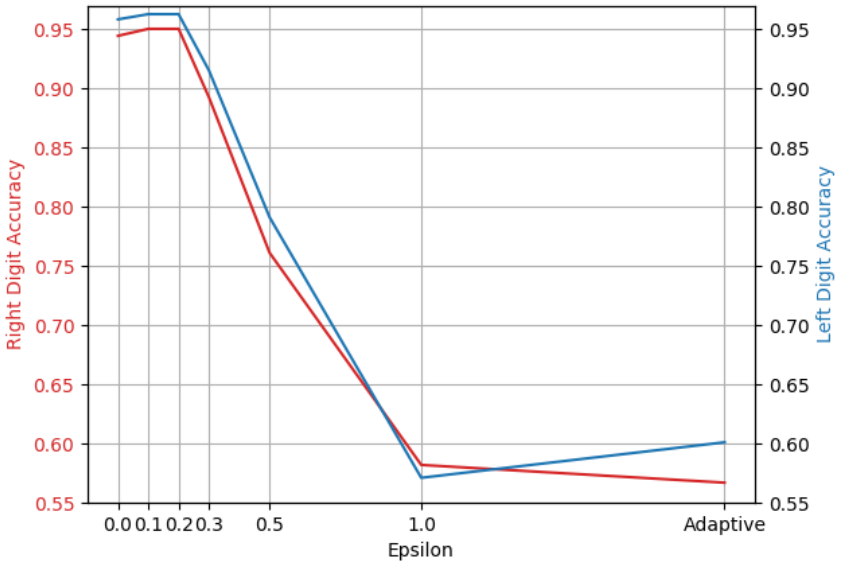


Figure 3.3: Tasks' performance for different ϵ values. $\epsilon = 0.0$ corresponds to the original MGDA and adaptive MGDA point has $\lambda = 1$.

forms inexact-MGDA but for a fine tuned ϵ we can see a slight improvement in the digit accuracies. Notice that directly using the ϵ estimate in equation (2.13) with $\lambda = 1$ does not yield a good training point. So we make a separate sweep on λ parameters on Figure (3.4) and compared it with the best performing constant ϵ point for $\epsilon = 0.2$. Adaptive-MGDA outperforms both classical and constant MGDA when tuned with the right λ.

We have also tracked the estimated value during adaptive-MGDA before scaling in figure (3.5), as well as the change of α in Figure (3.6).

The tracked uncertainty values in Figure (3.5) illustrate a couple of interesting observations.

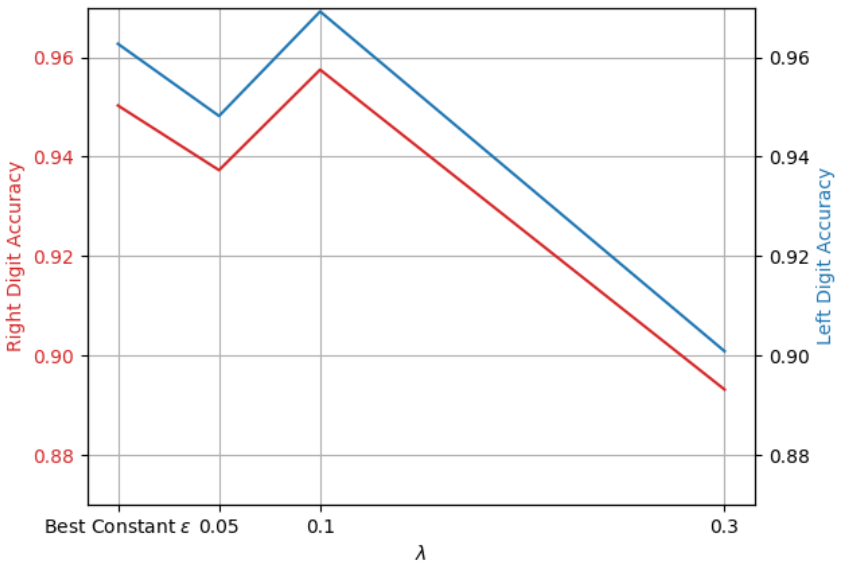
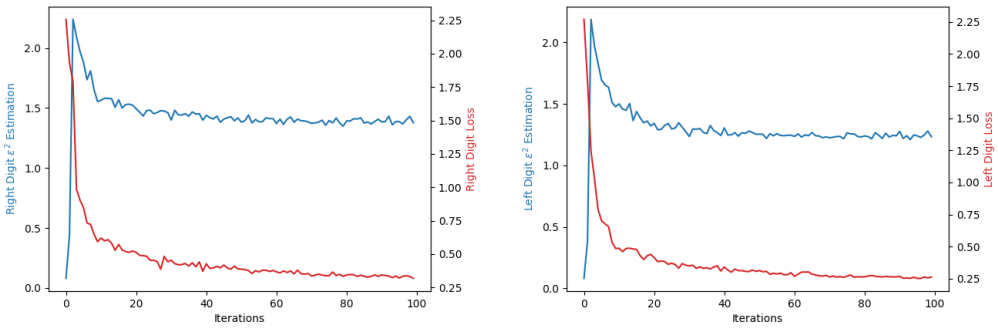


Figure 3.4: Adaptive ϵ training results for different λ values. Best constant ϵ point is $\epsilon = 0.1$ point from figure (3.3).



(a) Left digit loss and ϵ^2 estimation (b) Right digit loss and ϵ^2 estimation

Figure 3.5: Training loss per task and ϵ^2 estimation by the adaptive algorithm.

- The gradient inexactness peak during the steepest descent of the losses

- The uncertainty converges as the loss converge
- The adaptive $\hat{\epsilon}$ tracked is an order of magnitude larger than the optimum ϵ obtained by the sweep. Moreover the optimum $\lambda = 0.1$ seems to correct this mismatch.

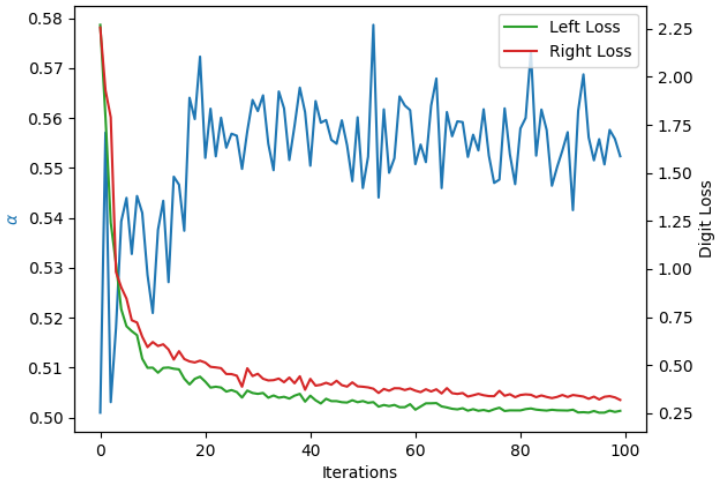


Figure 3.6: Change of α through training during adaptive MGDA

We have also observed that α values in Figure (3.6) oscillate less as the network nears convergence and there is slight more weighting on the right digit loss throughout training which we believe is due to right digit loss being consistently higher than the left. On the MNIST experiments we have observed that using large ϵ or λ in training causes the early termination of the training where updates are not being done.

3.2 Joint Semantic Segmentation, Instance Segmentation and Depth Estimation

We follow the Resnet-50 [9] based shared encoder and pyramid pooling network [10] tri-decoder architecture put together by Sener et al. [2]. Network

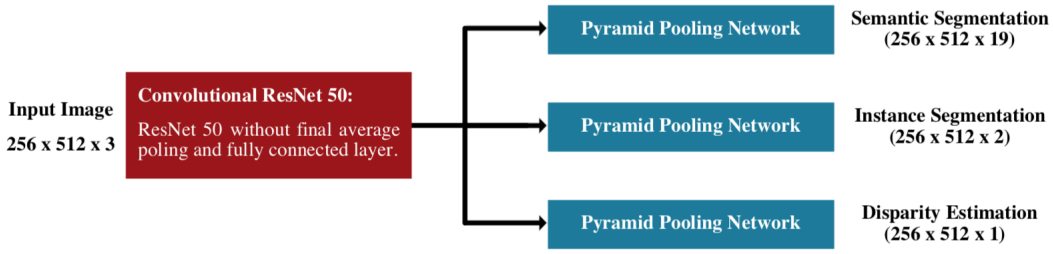


Figure 3.7: Network used for joint semantic segmentation, instance segmentation and depth estimation by Şener et al. [2].

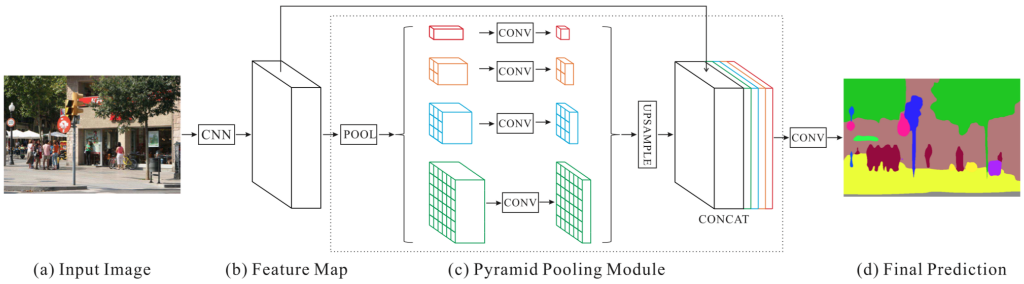


Figure 3.8: Pyramid network as the decoder layers of the 3 task network by Zhao et al. [10].

is trained using the fine annotated cityscapes dataset [11] consisting of 3750 training images and 500 validation images. Dataset consists high resolution images taken over 30 cities in Germany annotated for semantic, instance and depth maps.

3.2.1 Semantic Segmentation

Goal of semantic segmentation is to assign a classification label for every pixel on the input image based on the object that the pixel belongs to. Output is a $H \times W \times C$ image where C corresponds to the number of objects in the dataset. For the cityscapes dataset $C = 19$. There are many popular metrics to evaluate semantic segmentation performance, in this work we use the mean intersection over union metric (mIoU). For every class

$$IoU = \frac{y \cap \hat{y}}{y \cup \hat{y}} \tag{3.3}$$

where y denotes the ground truth regions that belong to the class and $\hat{y}$ is the predicted pixels to be belonging to the given class. Mean intersection over union is obtained by averaging IoU scores over all C classes. The loss function used is 1 minus sum of IoU scores for each pixel.

3.2.2 Instance Segmentation

Instance segmentation task is similar to semantic segmentation where each pixel is labeled based on the class of the object it belongs to. Difference is in instance segmentation when multiple separate objects of the same class are present, different labels should be assigned for different instances. For example when there are two dogs in the image each dog should be classified differently. Therefore the output space of the instance segmentation task would be undefined if we were to try an output map sized $H \times W \times C$ in the same form of semantic task because number of instances can change for each image. Kendall et al. [12] suggests a proxy task as a workaround of this problem. Output maps are of $H \times W \times 2$ where each pixel at location (x, y) contains a vector $(\delta x, \delta y)$ that points at the center of mass of the object such that $(x + x, y + \delta y)$ is the mean location of all the pixels that belong to the given object. Using the proxy problem, ground truth maps can be formed

	Disparity Error	**Instance Error**	**mIOU**
MGDA	2.78	11.24	56.54
Constant ϵ-MGDA	2.72	10.82	58.16
Adaptive ϵ-MGDA $\lambda = 0.1$	2.61	10.36	58.22

Table 3.1: Comparison of MGDA variations on 3 task network. For mIoU higher is better. Optimum ϵ values found by the sweep were 0.35, 0.2 and 0.25 for depth estimation, instance segmentation and semantic segmentation respectively.

and network is directly trained on the MSE between the ground truth and predicted maps. We report the MSE of the validation dataset.

3.2.3 Depth Estimation

Depth estimation task aims to estimate the distance of each pixel from the camera. Cityscapes dataset contains the depth maps in the form $H \times W \times 1$ where each pixel directly contains the ground truth depth. Training is done using the MSE with the ground truth. We report the MSE of the validation dataset.

3.2.4 Joint Training and Results

Due to the large size of the network, a full sweep of all the potential λ and ϵ values was not feasible. Best performing values was determined by a sweep on a smaller dataset containing 150 images. Later a full training was performed with the same parameters. All training instances use Adam optimizer with 0.001 learning rate. Learning rate is halved every 30 epochs. Training was ran for 100 full epochs and the latest non-dominated point has been selected from the validation performances. Based on the comparison on Table (3.1) inexact MGDA outperforms the regular MGDA with adaptive-MGDA yielding the best result. However like the MNIST experiment the λ scaling factor is significantly small. When we consider that a large ϵ results in gradient

updates being skipped for most of the inexact gradients the low optimum λ value supports our conclusion that some inexactness helps the network generalize but filtering out gradients with too much inexactness is still beneficial for the training.
As before we tracked the estimated $\hat{\epsilon^2}$ for each task together with corresponding loss function.

We have observed very similar gradient inexactness and loss relation to

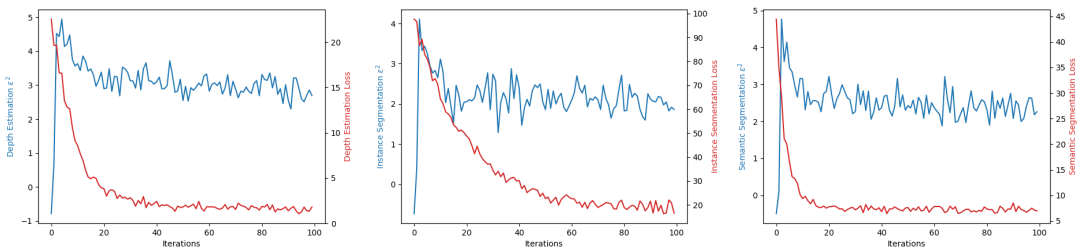


Figure 3.9: Task loss and ϵ^2 estimation by the adaptive algorithm for 3-task network.

the MNIST experiments on Figure (3.9) eventhough the network was larger and 3 tasks were employed. The nature of tasks were significantly different than each in this example compared to the MNIST example so we see more variation in the gradient inexactness range and convergence time.

3.3 Joint Saliency and Compression Network

3.3.1 Image Compression

The most representative models for image compression typically adopted the auto-encoder (AE) like structure whose bottleneck layers, or latent representations, are quantized and entropy coded [13, 14, 15, 16, 17, 18, 19]. For

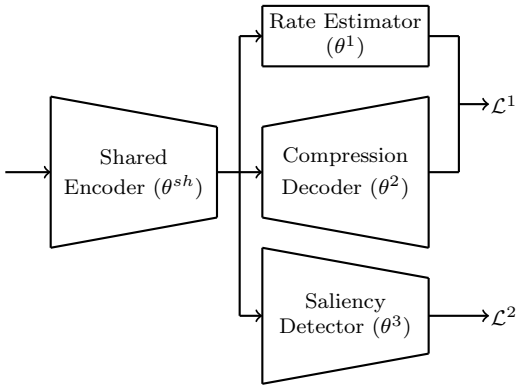


Figure 3.10: Simple illustration of proposed simultaneously learning framework.

example, Toderici *et al.* [15] proposed an end-to-end image coding model based on the RNN, where the original and residual images are iteratively compressed using RNN structure. During each RNN iteration, the better reconstruction with a commensurate bitrate cost will be produced. However, the RNN based method might have limitations in representing high-frequency residuals. Additionally, the lack of entropy estimation during RNN training also constraints its overall R-D performance. To address this issue, Theis *et al.* [14] presented a CNN based AE structure, where the entropy model was approximated using Gaussian distribution during optimization. In [20], an inpainting base learning approach was proposed for image compression. To enhance the visual quality, the generative adversarial network (GAN) based learning strategies were investigated to the CNN based framework [16, 17, 18] where the perceptual quality of the reconstructed images could be extremely promoted. Furthermore, the generalized divisive normalization (GDN) [21] together with variational auto-encoder was introduced as a substitute of the nonlinear activation in [13], which significantly improves the coding performance to be competitive with JPEG2000 standard. When comparing with other activation functions, the core advantage of GDN is its fully reversibility which guarantees nearly no information loss for the transform coding. And

its successor [22], a novel parameters estimation for entropy model has shown that additional coding gain could be obtained by extend the GDN with a hyperprior that captures the fact that spatially neighboring elements of the latent representations tend to vary together in their scales. In [23], Alemi *et al.* presented a theoretical framework for understanding representation learning using latent variable models in terms of the R-D tradeoff. Tschannen *et al.* studied to optimize the R-D tradeoff under the constraint that the reconstructed samples follow the distribution of the training data [18].

3.3.2 Saliency Prediction

Image saliency has embraced considerable development due to spread of deep learning (DL) techniques and the current state-of-the-art methods are mainly based on deep learning. One of these frameworks is the Ensemble of Deep Networks (eDN) model by proposed by Vig *et al.* [24], who consists of three convolutional layers followed by a linear classifier that blends feature maps coming from the previous layers. After this work, Kummerer *et al.* [25] proposed two deep saliency prediction networks: the first, called DeepGaze I, was based on the AlexNet model [26], while its successor, DeepGaze II [27], was built upon the VGG-19 network [28]. Liu *et al.* [29] presented a multi-resolution CNN (Mr-CNN) fine-tuned over image patches centered on fixation and non-fixation locations. It is well known that deep learning approaches strongly depend on the availability of sufficiently large datasets. The publication of a large-scale eye-fixation dataset, SALICON [30], indeed contributed to a big progress of deep saliency prediction models. SALICON dataset consists of 480×640 images with ground truth binary saliency eye fixation points (S^f) as well as the continuous saliency maps (S^m). There are 10000 training, 5000 validation and 5000 test images in the dataset. Ground truth fixation and saliency maps are publicly available for the training and validation images.

More recently, Kummerer *et al.* show that no single saliency map can

perform well under all metrics and proposed a principled approach to solve the benchmarking problem by separating the notions of saliency models, maps and metrics [31].

3.3.3 Proposed Framework

We now consider the problem of joint learning for image compression and saliency detection. The general framework is depicted in Fig. 3.11. Following [13], we adopt an AE structure for image compression, which consists of an encoder (with parameter θ^{sh}), a decoder (with parameter θ^1), and an entropy estimator (R). The encoder takes an image and generates a set of feature maps; The decoder takes the quantized feature maps and generate the reconstructed image. The entropy coder takes the quantized feature maps and estimates their entropy as a surrogate for the bit rate needed to code the feature maps. The learning task for compression is to minimize the compression loss defined as

$$\mathcal{L}^1 = ||I - \hat{I}||_2^2 + \lambda_c R, \tag{3.4}$$

where λ_c depends on the target bit rate, I and $\hat{I}$ are original and reconstructed images, R indicates the entropy of the latent variables.

In our current work, we adopt the compression framework proposed in [13]. The encoder utilizes a stack of three convolution layers separated by two GDN layers as non-linearity function while the decoder part has mirror operations with the encoder. Regarding the entropy model and arithmetic coding engine for compressing the latent variables (X), we deploy the density model with factorized-prior proposed in [22]. It is worthy noting that although the image compression needs to quantize (round) the X to get finite entropy for bit-rate, thanks to [13], the end-to-end learning of variational AEs could be achieved by adding the i.i.d uniform noise ($\Delta \sim (-0.5, 0.5)$) to X to simulate the quantization process such that the entire framework could be optimized

end-to-end. The detailed illustration of network architecture is shown in Appendix. In the following discussion, the parameter θ^{sh} capsules the encoder part of AE in Fig. 3.11 and θ^{t} denotes the parameters in the two specific tasks.

For saliency detection, we adopt the architecture proposed in [32], which uses initial Resnet [9] and VGG-16 [28] layers to generate features as inputs to an attentive convolutional long-short-term-memory (ConvLSTM) architecture. Following [32], we remove the intial VGG-16 and Resnet layers and directly use the encoder layers from the compression architecture followed by three 3×3 stride 1 zero-padded convolutional layers with ReLu activation functions to generate the inital visual features. We also reduce the number of total feature maps from 512 to 192. We use a loss function that is a weighted average of multiple metrics. The saliency detector (with parameter θ^2) takes the quantized feature maps X and generates a saliency map ($\hat{S}$). The learning task is to minimize the loss expressed as,

$$\mathcal{L}^2 = c_1\mathrm{NSS}(\hat{\mathrm{S}}, \mathrm{S^f}) + \mathrm{c_2KL}(\hat{\mathrm{S}}, \mathrm{S^m}) + \mathrm{c_3CORR}(\hat{\mathrm{S}}, \mathrm{S^m}), \tag{3.5}$$

where $\mathrm{NSS}(\mathrm{S^f}, \hat{\mathrm{S}})$ is the Normalized Scanpath Saliency loss specifically defined for the evaluation of generated saliency map ($\hat{S}$) who is to quantify the saliency map values at the binarized label fixation locations (S^f) and to normalize it with the saliency map variance [33],

$$\mathrm{NSS}(\hat{\mathrm{S}}, \mathrm{S^f}) = \frac{1}{\mathrm{M}} \sum_{\mathrm{i}} \frac{\hat{\mathrm{S}}_\mathrm{i} - \mu(\hat{\mathrm{S}})}{\sigma(\hat{\mathrm{S}})} \mathrm{S^f_i}, \tag{3.6}$$

where the $\mu(\cdot)$ and $\sigma(\cdot)$ are the mean and variance for the generated saliency map respectively and M is the number of elements of saliency map. While the remaining two items in Eqn. (3.5) denote the Kullback-Leibler (K-L) divergence,

$$KL(\hat{S}, S^m) = \sum_i S_i^m \log(\frac{S_i^m}{\hat{S}_i}). \tag{3.7}$$

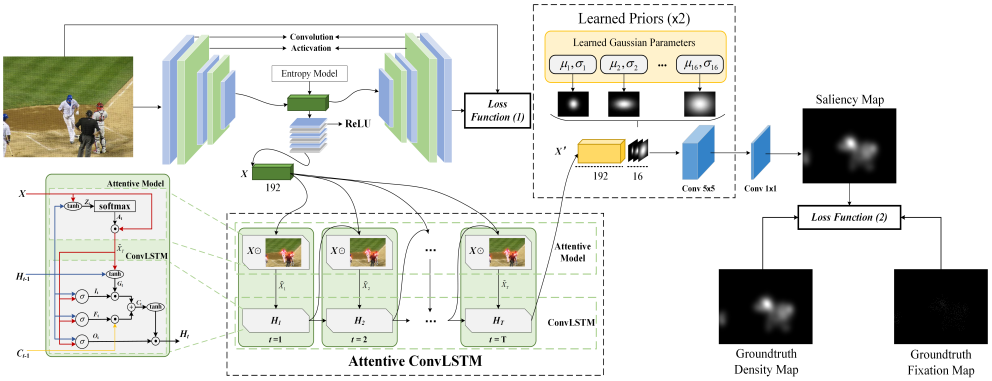


Figure 3.11: The proposed framework for simultaneously learning for image compression and saliency. The compression network follows [13]. The saliency detection network follows the [32] with small modifications.

and linear Correlation Coefficient

$$CORR(\hat{S}, S^m) = \frac{\sigma(\hat{S}, S^m)}{\sigma(\hat{S}) \cdot \sigma(S^m)}. \tag{3.8}$$

respectively, where the numerator is the co-variance between $\hat{S}$ and the continuous label S^m. The three $c_i, i \in \{1, 2, 3\}$ are used to balance the output saliency map.

3.3.4 Training

The shared parameters θ^{sh} of our proposed model consists of the parameters for GDN and the convolution kernels and bias. We use a 4 layer encoder, with a 5×5 convolutional kernel with stride 2 followed by GDN operation in each layer. First 3 layers use 192 convolution channels and the final latent representations have 320 channels. *Same-zero* padding is always used in our framework to ensure the feature dimension consistency. For the compression decoder network we use the mirror structure of the shared encoder with 4 upsampling convolutional layers followed by inverse GDN activations. Following [32], we built the saliency framework using the attentive convolutional

LSTM cells, using the structure illustrated in Fig. 3.11. Different from [32], the latent variables are obtained using the shared encoder. These latent variables then go through 3 additional convolutional layers with 320 channels, each activated by rectified linear unit (ReLU). The attentive convoltional LSTM model is subsequently applied to predict a set of feature maps, which are then combined by 1×1 convolutional layers with multiple learned gaussian priors to model the tendency of humans to fixiate in the center region of the image. The gaussian parameters are learnable by the network.

Our joint training procedure consists of the following three phases.

- **Phase 1 compression training:** Only the encoder parameters (θ^{sh}) and the decoder parameters (θ^1) are trained using the compression loss in Eqn. (3.4). We do not train the model until convergence on this phase to allow for joint fine tuning on later training phases.
- **Phase 2 saliency training:** The encoder parameters (θ^{sh}) are frozen and the saliency detector parameters (θ^2) are optimized for the loss in Eqn. (3.5).
- **Fine tuning phase:** All the parameters are further refined jointly using two different approaches.
  - **Weighted average training :** We use the weighted sum of the loss the functions

$$\mathcal{L} = \beta\mathcal{L}^1 + (1-\beta)\mathcal{L}^2, \tag{3.9}$$

    where β is a constant weight.
  - **MGDA:** Updating θ^{sh} by the classical MGDA. θ^1 and θ^2 are then updated based on the losses $\mathcal{L}^1$ and $\mathcal{L}^2$ respectively.
  - **Adaptive ϵ-MGDA**
  - **Constant-ϵ-MGDA**

For evaluation purposes, we have also performed a separate saliency training and a separate compression training.

- **Compression priority training:** This training starts with the continuation of the **Phase 1** training, where we continue minimizing L^1 loss until convergence on the compression task. Then θ_{sh} parameters are frozen and training is performed only on the θ_2 parameters with L^2 loss.

- **Saliency priority training:** This model is trained using only $\mathcal{L}^2$ loss over the parameters $\theta^{sh} \cup \theta^2$ until convergence on the saliency task. Then $\theta^{sh} \cup \theta^2$ are frozen and only the θ^1 parameters are trained using L^1.

Different instances of the saliency only training and full compression training were used to determine a viable λ_c in Eqn. (3.4) and viable $\lambda_1, \lambda_2, \lambda_3$ in Eqn. (3.5). Then these parameters were kept as constant through all the phases of training.

For all phases of the training we have used the SALICON training and validation dataset [30]. During Phase 1, the model is trained self-supervised on 256×256 random crops from the training images normalized to $[0, 1]$ with i.i.d. uniform noise $\Delta \sim (-0.001, 0.001)$ added for better generalization.

We have used an SGD optimizer with momentum with batchsize of 8 for all the training phases, halving the learning rate every 15 epochs. Phase 1 training lasted for 1500 epochs, phase 2 training lasted for 50 epochs and fine tuning training was done for 200 epochs on both the proposed and the weighted average procedure. Although the weighted average fine tuning lasted for the entire 200 epochs we have observed convergence much faster compared to the proposed MGDA method. However training was done for the full 200 epochs for fairness. Saliency priority training was ran for 50 epochs and compression priority training was ran for 2000 epochs.

3.3.5 Results

We report our results on the SALICON validation dataset at different phases of training. Since the parameters θ^{sh} are initially trained only on $\mathcal{L}^1$ loss, our model performs much better in the compression task than the saliency task at the end of Phase 2. However with the fine tuning phase this discrepancy between the tasks greatly reduces. Tables 3.2 and 3.3 show that proposed MGDA algorithm significantly outperforms the weighted average algorithm. It was able to reach significantly lower $\mathcal{L}^1$ and $\mathcal{L}^2$ than weighted average approach for three choice of the weighting factor; For $\beta = 0.2$, it achieved significantly lower $\mathcal{L}^1$ and only slightly higher $\mathcal{L}^2$. Note that the MGDA training was able to further reduce on the saliency loss, without increasing the compression loss. We further compare the proposed method with the constant-weight averaged training methods on Fig. 3.12. The red points are generated by using weighted average training with different weights. Blue point corresponds to MGDA training. Black points are the saliency and compression priority training results, which can be considered as special cases of weighted average training with $\beta = 0$ and $\beta = 1$, respectively. This figure clearly shows that the MGDA algorithm is able to reach lower compression loss and saliency loss than the weighted average method for most weighting parameters, except for one point where saliency loss is slightly lower but compression loss is significantly higher. Overall, the MGDA solution has a significantly better trade-off between the two losses than all possible solutions achievable by varying the weighting factor in the entire range. Furthermore, the MGDA solution has a compression loss very close to the compression-priority training, but has a saliency loss that is still quite higher than that by saliency-priority training. This is likely because MGDA started with an initial solution that favors the compression task. Had we start with a solution that is more balanced between the two tasks, MGDA may yield a solution that has worser compression performance and better saliency performance.

We have also compared the training losses of the exact and inexact MGDA

	$\mathcal{L}^1$	**PSNR**	**BPP**
End of Phase 1	2.226	29.131	0.742
End of Phase 2	2.226	29.131	0.742
Original MGDA	2.013	29.291	0.685
Adaptive MGDA $\lambda = 0.01$	2.146	29.191	0.719
Adaptive MGDA $\lambda = 0.05$	2.211	29.276	0.725
Adaptive MGDA $\lambda = 0.1$	2.359	27.972	0.742
$\beta = 0.2$	3.428	25.374	0.771
$\beta = 0.4$	2.642	27.456	0.737
$\beta = 0.6$	2.221	28.923	0.694
$\beta = 0.8$	2.139	29.273	0.689
Compression priority	1.845	31.150	0.669
Saliency priority	8.123	26.152	6.123

Table 3.2: Compression model performances for joint training. BPP is the average bits per pixel calculated using rate estimation network from [22]. MGDA is our proposed method.

	$\mathcal{L}^2$	**NSS**	**CORR**
End of Phase 2	-2.143	1.642	0.671
Original MGDA	-3.002	1.856	0.791
Adaptive MGDA $\lambda = 0.01$	-3.038	1.921	0.772
Adaptive MGDA $\lambda = 0.05$	-2.973	1.803	0.722
Adaptive MGDA $\lambda = 0.1$	-2.614	1.752	0.769
$\beta = 0.2$	-3.101	2.002	0.813
$\beta = 0.4$	-2.617	1.745	0.781
$\beta = 0.6$	-2.364	1.682	0.768
$\beta = 0.8$	-2.164	1.648	0.763
Saliency priority	-3.441	2.163	0.811
Compression priority	-2.152	1.654	0.681

Table 3.3: Saliency model performances for different joint training instances.

points in Table (3.4). Inexact-MGDA has resulted in slightly better training loss performance which may be because the by making sure batch gradients are closer to the whole epoch gradient, the network optimizes better on the

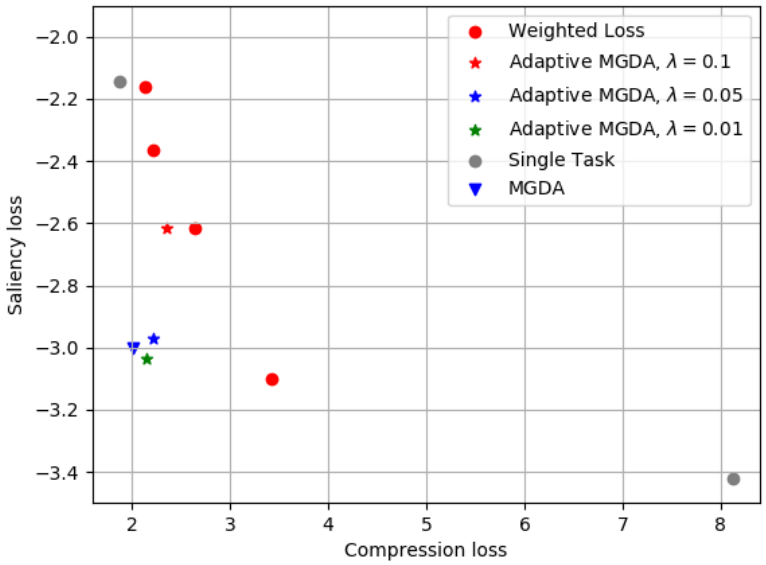


Figure 3.12: The validation loss values for saliency and compression. Black points are the saliency and compression priority training results.

	Compression Loss	**Saliency Loss**
Original MGDA	1.513	-3.247
Adaptive MGDA $\lambda = 0.01$	1.502	-3.298
Adaptive MGDA $\lambda = 0.05$	1.527	-3.256
Adaptive MGDA $\lambda = 0.1$	1.711	-2.924

Table 3.4: Compression and saliency training losses compared based on MGDA variants.

training data but losses the generalization benefits of a smaller batchsize. In compression and saliency experiments by varying β we observe a Pareto frontier on Figure (3.12) and MGDA variant algorithms yielding better solutions than the Pareto frontier. However this network have not seen improvement by using the inexact MGDA, in fact using the scaling factor $\lambda = 0.1$ similar to the previous experiments have in fact significantly lowered the network performance. In order to get results similar to regular MGDA very

small λ values had to be used. This not very surprising as $\lambda \to 0$ inexact MGDA acts like regular MGDA. We speculate some explanations about why the network have not gained from the inexact MGDA.

- Out of the three experiments performed, saliency-compression was the only one where the multiple tasks are not very compatible. The entropy minimization of the compression was surely in direct contrast with the saliency detection task and MSE minimzation does not take into account which pixels are salient.

- The relationship between ϵ and $\hat{\epsilon}$ may not be linear. Using a simple scaling factor lambda may not sufficiently capture the relation between the two variables.

- The inexactness characteristics of all tasks are different. It is possible that due to some inherent nature of the tasks, this network benefits more from inexactness. We encourage more research about gradient inexactness that could reveal the underlying relationship between tasks and their relationship with inexact gradients.

We have also tracked training losses during MGDA starting from after the compression priority training. The training loss trajectories demonstrated in Fig. 3.13 show that saliency performance increase without performance loss in the compression task. Compression task undergo small oscillations throughout training without any significant change. We believe this oscillation is caused by the use of the stochastic gradient descent, where the MGDA gradient update is done based on the loss incurred in each small batch, rather than the entire training set.

Tracking the trajectory of α in Fig. 3.14, we see that α starts out small (reducing the influence of the compression loss) because the initial solution is obtained by minimizing the compression loss. As the saliency performance gets improved, α gradually increase so that both losses are considered more equally.

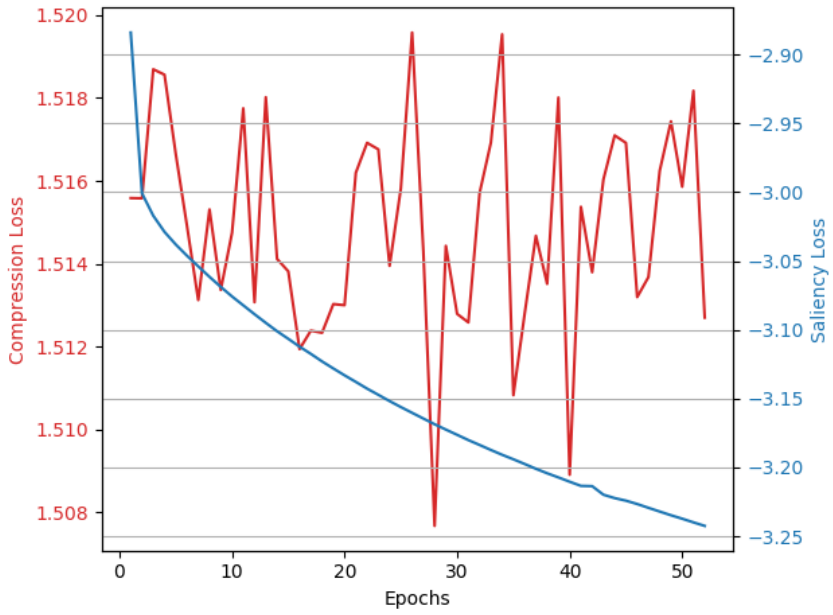


Figure 3.13: Saliency loss and Compression training loss variations during MGDA training starting from an optimum compression and sub-optimum saliency point. Losses are averaged over epochs.

We provide three generated test samples to show the visual results of the proposed framework. The left panel is the output saliency map produced by the proposed framework while the middle and right panels are the label saliency and original texture image respectively. It is clear that the proposed framework could generate convincing saliency map, while achieving good compression performance.

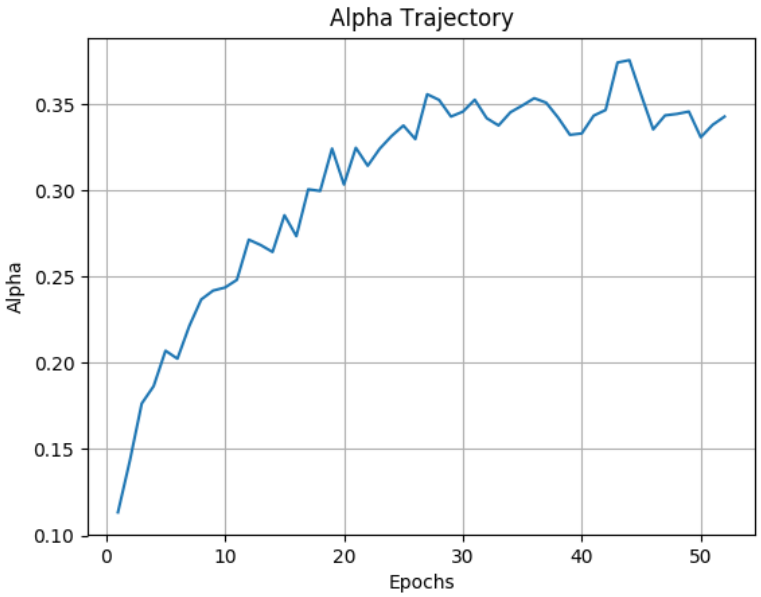


Figure 3.14: α trajectory during the proposed MGDA training initialized by the compression priority solution, averaged over epochs.

Figure 3.15: Left: the generated saliency map; middle: ground truth saliency map; right: original texture image.

(a) Original image.

(b) Reconstructed image.

Figure 3.16: Image reconstruction at 29.925 PSNR and 0.712 BPP using the inexact-MGDA $\lambda = 0.01$ trained model. Image from SALICON dataset.

(a) Original image.

(b) Reconstructed image.

Figure 3.17: Image reconstruction at 28.421 PSNR and 0.703 BPP using the inexact-MGDA $\lambda = 0.01$ trained model. Image from KODAK dataset [34].

Chapter 4

Conclusion and Future Work

We have proposed a novel adaptive inexactness algorithm extension to the existing MGDA algorithm and tested the performance in 3 network setups. In 2 of these setups we have observed significant gains by using the adaptive algorithm. Our inexactness estimation hypothesis seems to be holding true for these networks. However for the joint saliency and compression network the adaptive algorithm have performed very similarly to the non-adaptive case. We speculate that this is due to failure of adjusting the right scaling hyperparameter for this set of tasks. Unfortunately due to the sheer size of the compression and saliency network it was not feasible to run a full sweep on the hyperparameters.

4.1 Conclusions

- Using estimated inexactness have improved results for two of our experiments. We conclude that our hypothesis $\epsilon^2 = \lambda \frac{1}{N} \sum_p (\|\nabla \tilde{\mathcal{L}}^p - \mu_{\nabla \mathcal{L}}\|_2^2)$ is likely true and the mean of the minibatch gradient using stochastic gradient descent is a good estimate of the true epoch gradient.

- The inexact algorithm has yielded slightly better training loss perfor-

mance. We believe this is because inexact MGDA basically tries to make sure all minibatch gradients are similar to the epoch gradient. By doing so a better training performance is observed but the generalization benefits of using minibatches is lost.

- The optimum ϵ^* value that achieved the best performance was significantly smaller than the estimated ϵ value but by scaling the estimated value with a right small scaling factor we observed valuable gains. We believe by keeping the ϵ smaller than the actual error bound the model gains from the randomness introduced with the minibatch training as explained in [7].

- The pseudo-mean gradient introduced for tracking gradients over an epoch is similar to the momentum term in popular optimizers like Adam. By doing the SGD update over a running average of the gradient and scaling the gradient based on the running average of second moments, Adam essentially does minimal updates in the direction that shows more variation throughout training while maximizing updates in the consistent, less variational directions. We believe this is closely related to the gradient inexactness however the approach we are taking is significantly different. Inexact-MGDA algorithm operates in the multi-task scene trying to make sure all updates are Pareto-optimal for all the tasks whereas methods like Adam and momentum are analyzing inexactness for every element of the gradient vector.

4.2 Future Work

- A relationship between batchsize and inexactness estimation was not explored. It would be interesting to look at how inexactness amount and final performance change with different batchsizes.

- The pseudo-mean gradient introduced for tracking gradients over an

epoch is an approximation of the true epoch gradient. One could try different methods of estimating the true gradient. Using a running average instead of the pseudo-mean could yield promising results.

- Gradient inexactness is an interesting phenomenon studied widely in deep learning that is not unique to multi-task problems. We have approached the multi-task problem from the inexactness perspective as the multiple gradients provided with the tasks allowed for a straightforward analysis. Similar methods could be leveraged in a single-task scenario by replacing different task gradients with different minibatch gradients.

Bibliography

[1] Sebastian Peitz and Michael Dellnitz. Gradient-based multiobjective optimization with uncertainties. pages 159–182, 2018.

[2] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *arXiv preprint arXiv:1810.04650*, 2018.

[3] Yu Zhang and Qiang Yang. A survey on multi-task learning. *CoRR*, abs/1707.08114, 2017.

[4] I. Y. Kim and O. L. de Weck. Adaptive weighted sum method for multiobjective optimization: a new method for pareto front generation. *Structural and Multidisciplinary Optimization*, 31(2):105–116, Feb 2006.

[5] Jean-Antoine. Désidéri. Multiple-gradient descent algorithm (mgda). *RR-6953*, 2009.

[6] Wolfe P. Frank M. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, doi:10.1002/nav.3800030109, 1956.

[7] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016.

[8] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[10] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *CoRR*, abs/1612.01105, 2016.

[11] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[12] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. *CoRR*, abs/1705.07115, 2017.

[13] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*, 2016.

[14] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*, 2017.

[15] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. Full resolution image compression with recurrent neural networks. In *CVPR*, pages 5435–5443, 2017.

[16] Eirikur Agustsson, Michael Tschannen, Fabian Mentzer, Radu Timofte, and Luc Van Gool. Generative adversarial networks for extreme learned image compression. *arXiv preprint arXiv:1804.02958*, 2018.

[17] Oren Rippel and Lubomir Bourdev. Real-time adaptive image compression. *arXiv preprint arXiv:1705.05823*, 2017.

[18] Michael Tschannen, Eirikur Agustsson, and Mario Lucic. Deep generative models for distribution-preserving lossy compression. *arXiv preprint arXiv:1805.11057*, 2018.

[19] Lei Zhou, Chunlei Cai, Yue Gao, Sanbao Su, and Junmin Wu. Variational autoencoder for low bit-rate image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2617–2620, 2018.

[20] Mohammad Haris Baig, Vladlen Koltun, and Lorenzo Torresani. Learning to inpaint for image compression. In *Advances in Neural Information Processing Systems*, pages 1246–1255, 2017.

[21] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. Density modeling of images using a generalized normalization transformation. *arXiv preprint arXiv:1511.06281*, 2015.

[22] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. *arXiv preprint arXiv:1802.01436*, 2018.

[23] Alexander Alemi, Ben Poole, Ian Fischer, Joshua Dillon, Rif A Saurous, and Kevin Murphy. Fixing a broken elbo. In *International Conference on Machine Learning*, pages 159–168, 2018.

[24] Eleonora Vig, Michael Dorr, and David Cox. Large-scale optimization of hierarchical features for saliency prediction in natural images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2798–2805, 2014.

[25] Matthias Kümmerer, Lucas Theis, and Matthias Bethge. Deep gaze i: Boosting saliency prediction with feature maps trained on imagenet. *arXiv preprint arXiv:1411.1045*, 2014.

[26] 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.

[27] Matthias Kümmerer, Thomas SA Wallis, Leon A Gatys, and Matthias Bethge. Understanding low-and high-level contributions to fixation prediction. In *2017 IEEE International Conference on Computer Vision*, pages 4799–4808, 2017.

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[29] Nian Liu, Junwei Han, Dingwen Zhang, Shifeng Wen, and Tianming Liu. Predicting eye fixations using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 362–370, 2015.

[30] Ming Jiang, Shengsheng Huang, Juanyong Duan, and Qi Zhao. Salicon: Saliency in context. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1072–1080, 2015.

[31] Matthias Kümmerer and SA Thomas. Saliency benchmarking made easy: Separating models, maps and metrics. In *European Conference on Computer Vision*, pages 798–814. Springer, Cham, 2018.

[32] Marcella Cornia, Lorenzo Baraldi, Giuseppe Serra, and Rita Cucchiara. Predicting human eye fixations via an lstm-based saliency attentive model. *arXiv preprint arXiv:1611.09571*, 2016.

[33] Robert J Peters, Asha Iyer, Laurent Itti, and Christof Koch. Components of bottom-up gaze allocation in natural images. *Vision research*, 45(18):2397–2416, 2005.

[34] Kodak. http://r0k.us/graphics/kodak/, 1999.