General Research Notes

Zachary Ross

April 11, 2022

Contents

Part I

Dated Findings

February 2, 2022

In the work presented to me, Dr. Lee primarily uses pruning for gradient masking rather than parameter masking, although most studies I've read seem to have done the opposite. I initially assumed that given the standard linear layer function with masking

$$\boldsymbol{z} = (\boldsymbol{M} \odot \boldsymbol{W})\boldsymbol{h} + \boldsymbol{b} \tag{1}$$

the effect of the mask would still result in non-zero values in the gradient of the matrix. This is falsified by the derivation

$$\frac{\partial L}{\partial W_{i,j}} = \frac{\partial L}{\partial z_i}\frac{\partial z_i}{\partial W_{ij}} = M_{ij}\frac{\partial L}{\partial z_i}h_j \tag{2}$$

or in vectorized form we have that

$$\frac{\partial L}{\partial \boldsymbol{W}} = \boldsymbol{M} \odot \left(\frac{\partial L}{\partial \boldsymbol{z}}(\boldsymbol{h})^\intercal\right) \tag{3}$$

which implies that the use of a forward mask guarantees that of a backward. The method used instead in Dr. Lee's examples was $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{h} + \boldsymbol{b}$ with masking only applied during the gradient (i.e. that of Equation 3). So what needs to be tested is whether or not we can apply masking solely during the forward phase i.e. implement Equation 1 with builds its gradient without the mask.

February 9, 2022

Thorough testing has disproved our previous hypothesis. Strangely enough, the gradient masking is much more effective than weight masking for some odd reason.

February 23, 2022

In the following discussion, I use beats per minute (BPM) to refer to heart rate. I only do it this way because I've already written it out and don't feel like changing it.

I've been pondering a bit recently on a hypothesis I've had, that being the perception of time is relative to the individual based on several factors. At first I believed that it's connected to BPM, but this was disproved by my addiction to coffee. I've noticed this relativity even in my day to day, some days seem faster than others while certain activities seem to be slower. For example, under the influence of coffee I feel that my day goes by much faster, which leads me to believe the effects of caffeine are misunderstood (which I will go into later). After or during an intense workout, my perception of time seems to slow way the hell down. I notice this because I'll be listening to music, and maybe it has to do with my percussionist background with intense focus on steady timing, but the songs pass by much slower. This is incredibly curious.

On other days where I don't quite feel myself, I notice instead that the music tends to be much quicker, often times leading me to feel like I'm missing alot.

For the coffee thing, maybe its affect on BPM does slow down perception of time, but caffeine could effectively and periodically cut out clips of time, making

us believe that time is moving faster. I believe this is affects productivity because it allows us to move from task to task seemingly "quicker". I don't believe it fully alters our productivity; rather, it seems like it allows us to reach the reward much quicker and thereby reinforce the thought process that we are "getting shit done". This activity/reward behavior most likely becomes a self-fulfilling cycle as the momentum continues, motivating us to continue doing tasks.

I've also considered how we would test these hypotheses because it's rather elusive. Say we tried to get a set of people to listen to a meteronome prior to an energy intensive activity and post said activity and asked them to compare the two for which one is faster. For one, how would we get reliable information? If we were just to pick random people from a population, they may not even be capable of discerning between two tempos. I suppose we could do a preliminary exam where we vet out those who could not tell the difference even between minor changes in tempo. We could pick out trained musicians, but then the study is limited to musicians techinically. I guess we would just have to do the vetting procedure and only select the upper echelon of people which can tell extremely minute differences in tempo after a select period of time.

Thoughts I had while working out, lol.

Part II

Algorithms and Theory

Chapter 1

Monte Carlo and Las Vegas Randomized Algorithms for Systems and Control*: An Introduction [9]

Abstract. *In this paper, we present an introduction to Monte Carlo and Las Vegas randomized algorithms for systems and control. Specific applications of these algorithms include stability analysis, Lyapunov functions, and distributed consensus problems.*

Chapter 2

Approximation Algorithms for k-Anonymity [1]

Abstract. *We consider the problem of releasing a table containing personal records, while ensuring individual privacy and maintaining data integrity to the extent possible. One of the techniques proposed in the literature is k-anonymization. A release is considered k-anonymous if the information corresponding to any individual in the release cannot be distinguished from that of at least k-1 other individuals whose information also appears in the release. In order to achieve k-anonymization, some of the entries of the table are either suppressed or generalized (e.g. an Age value of 23 could be changed to the Age range 20-25). The goal is to lose as little information as possible while ensuring that the release is k-anonymous. This optimization problem is referred to as the k-Anonymity problem. We show that the k-Anonymity problem is NP-hard even when the attribute values are ternary and we are allowed only to suppress entries. On the positive side, we provide an O(k)-approximation algorithm for the problem. We also give improved positive results for the interesting cases with specific values of k — in particular, we give a 1.5-approximation algorithm for the special case of 2-Anonymity, and a 2-approximation algorithm for 3-Anonymity.*

Chapter 3

Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming [3]

The initial problem attempted to be solved by this paper is creating an approximation algorithm for MAX-CUT to improve on the previous guarantee of $\frac{3}{4}$ for an undirected graph $G = (V, E)$ where $|V| = n$ with nonnegative weights $W_{ij} = W_{ji}$ i.e. $\boldsymbol{W}$ denotes a weighted adjacency matrix.

We construct a cut of G by assigning to each $i \in V$ a value $y_i \in \{-1, 1\}$ and construct $S \subset G$ by taking $S = \{i | y_i = 1\}$. Define the value of a cut as

$$w(S, \bar{S}) = \frac{1}{2} \sum_{i<j} W_{ij}(1 - y_i y_j) \tag{3.1}$$

The MAX-CUT integer quadratic program is then originally defined as

$$\text{MAX CUT}(G) = \max_{S \subset V} w(S, \bar{S}), \tag{3.2}$$

i.e. the goal is to find values y_i for all $i \in V$ that maximize this equation.

The proposed relaxation is to substitute each y_i in Equation 3.1 with some vector $\boldsymbol{v}_i \in \mathbb{R}^m$ $(m \leq n)$ such that $\|\boldsymbol{v}_i\|^2 = 1$ so that each $\boldsymbol{v}_i$ belongs to the m-dimensional unit sphere S_m. Under this relaxation, we instead construct $S \subset G$ by selecting some $\boldsymbol{r} \in S_m$ randomly and uniformly and take $S = \{i | \langle \boldsymbol{v}_i, \boldsymbol{r} \rangle \geq 0\}$ i.e. $\boldsymbol{r}$ determines a hyperplane which separates vertices of G. Then Equation 3.1 becomes

$$w(S, \bar{S}) = \frac{1}{2} \sum_{i<j} W_{ij}(1 - \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle). \tag{3.3}$$

Remark. For a real, symmetric matrix A, the following are equivalent:

- A is positive semidefinite;
- $\lambda(A) \subset \mathbb{R}^+ \cup \{0\}$ where $\lambda(X)$ is the set of X's eigenvalues; and
- $A = B^\intercal B$ for some $B \in \mathbb{R}^{m \times n}$ where $m \leq n$.

The goal is now to determine optimal values for the unit vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ and the dimensionality m for the vector space they reside. We can take $\boldsymbol{B} = \begin{bmatrix} \boldsymbol{v}_1 & \ldots & \boldsymbol{v}_n \end{bmatrix}$ so that $\boldsymbol{Y} = \boldsymbol{B}^\intercal \boldsymbol{B}$ is the Gram matrix of $\boldsymbol{B}$. Instead of solving for $\boldsymbol{B}$ right away, we can instead note that each $Y_{ij} = \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle$ and that our previous constraint is now just the constraint $Y_{ii} = 1$. We also have that Y is symmetric and PSD by construction. Semidefinite programming can be used to maximize the equation

$$\frac{1}{2} \sum_{i<j} W_{ij}(1 - Y_{ij}). \tag{3.4}$$

Rather than deal with each v_i directly, we can just use an approximate semidefinite programming algorithm to find an optimal Y in $O\left(\sqrt{n}\left(\log\left(\sum_{i<j} W_{ij}\right) + \log 1/\epsilon\right)\right)$ iterations (Alizadeh's adaptation of Ye's interior-point algorithm) with each iteration taking $O(n^3)$, where $\epsilon > 0$ denotes the acceptable distance from the optimal value Z_P^* (i.e. $Z_P^* - \epsilon$). Once an optimal $\boldsymbol{Y}$ has been found, an incomplete Cholesky decomposition can be used to obtain $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n \in S_m$ s.t.

$$\frac{1}{2} \sum_{i<j} W_{ij}(1 - \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle) \geq Z_P^* - \epsilon. \tag{3.5}$$

To find m, they note that any *extreme solution*, one which cannot be expressed as the strict convex combination of other feasible solutions, has at most rank l where

$$l \leq \frac{\sqrt{8n+1} - 1}{2} < \sqrt{2n}. \tag{3.6}$$

Any other solution has $m \leq n$.

3.1 Analysis

The analysis breaks apart into three cases:

1. the general case with nonnegative edge weights,
2. an tighter bound when the weight of the cut constitutes a large portion of the overall weight,
3. and extension to negative weights.

The general case constitutes defining the overall expected weight as in

$$\mathbb{E}[W] = \frac{1}{\pi} \sum_{i<j} W_{ij} \Pr\left[\operatorname{sgn}\left(\langle \boldsymbol{v}_i, \boldsymbol{r} \rangle\right) \neq \operatorname{sgn}\left(\langle \boldsymbol{v}_j, \boldsymbol{r} \rangle\right)\right]. \tag{3.7}$$

They then state the following lemma:

Lemma 3.1.

$$\Pr\left[\operatorname{sgn}\left(\langle \boldsymbol{v}_i, \boldsymbol{r}\rangle\right) \neq \operatorname{sgn}\left(\langle \boldsymbol{v}_j, \boldsymbol{r}\rangle\right)\right] = \frac{1}{\pi}\arccos(\langle \boldsymbol{v}_i, \boldsymbol{v}_j\rangle), \tag{3.8}$$

which follows from the fact that the probability that these two values have the same sign is equivalent to the dihedral angle between the vectors, i.e. the probability that a randomly selected hyperplane bisects the two divided by all the possible hyperplanes that bisect the unit sphere. This results in the theorem

Theorem 3.1.

$$\mathbb{E}\left[W\right] = \frac{1}{\pi}\sum_{i<j} W_{ij}\arccos(\langle \boldsymbol{v}_i, \boldsymbol{v}_j\rangle). \tag{3.9}$$

Let $\theta = \arccos(\langle \boldsymbol{v}_i, \boldsymbol{v}_j\rangle)$ and define

$$\alpha = \min_{0\leq\theta\leq\pi} \frac{2}{\pi}\frac{\theta}{1-\cos\theta}. \tag{3.10}$$

They then provide the following lemmas

Lemma 3.2. *For* $-1 \leq y \leq 1$, $\arccos(y)/\pi \geq \alpha \cdot \frac{1}{2}(1-y)$,

which follows from substituting $y = \cos\theta$, and

Lemma 3.3. $\alpha > 0.87856$

which follows from finding a lower bound for Equation 3.10. They use these to show that

Theorem 3.2.

$$\mathbb{E}\left[W\right] \geq \alpha\frac{1}{2}\sum_{i<j} W_{ij}(1-\langle \boldsymbol{v}_i, \boldsymbol{v}_j\rangle). \tag{3.11}$$

For the case where the cut constitutes a large portion of the overall weight, we define $W_{\text{tot}} = \sum_{i<j} W_{ij}$, $h(t) = \arccos(1-2t)/\pi$, and $\gamma = \arg\min_{0<t\leq 1} h(t)/t \approx 0.84458$. The analysis then provides the theorem:

Theorem 3.3. *Let*

$$A = \frac{1}{W_{tot}}\frac{1}{2}\sum_{i<j} W_{ij}(1-\langle \boldsymbol{v}_i, \boldsymbol{v}_j\rangle). \tag{3.12}$$

If $A \geq \gamma$, *then*

$$\mathbb{E}\left[W\right] \geq \frac{h(A)}{A}\frac{1}{2}\sum_{i<j} W_{ij}(1-\langle \boldsymbol{v}_i, \boldsymbol{v}_j\rangle), \tag{3.13}$$

i.e. as A varies between γ and 1, $h(A)/A$ varies between α and 1. This provides us with a performance guarantee of $h(A)/A - \epsilon$.

Part III

Deep Learning

Chapter 4

Efficient Per-Example Gradient Computations [4]

An important note about this paper is that it is *not* a derivation of the per-example gradient itself. Rather, it "describes an efficient technique for computing the *norm* of the gradient" with respect to the loss function.

Let each layer of a neural network be defined by the standard transformations

$$\begin{aligned} \boldsymbol{z}^{(i)} &= \boldsymbol{W}^{(i)}\boldsymbol{h}^{(i-1)} \\ \boldsymbol{h}^{(i)} &= \phi^{(i)}\left(\boldsymbol{z}^{(i)}\right) \end{aligned} \tag{4.1}$$

with the bias assumed to be an extra row/column of $\boldsymbol{W}$ and loss function $\mathcal{L}$. Let $\mathcal{B} \subset \mathcal{D}$ be a minibatch of the input dataset $\mathcal{D}$, and let $\mathcal{B}^{(j)}$ be the jth example in the minibatch with $\mathcal{L}^{(j)}$ corresponding to the loss of $\mathcal{B}^{(j)}$ and cost function $C = \sum_{j=1}^{n} \mathcal{L}^{(j)}$. The per example gradient norm is computed with respect both to the example and the layer, such that the gradient norm for layer i and example j is

$$\left\| \frac{d\mathcal{L}^{(j)}}{d\boldsymbol{W}^{(i)}} \right\|^2 = \sum_{k,l} \left(\frac{\partial \mathcal{L}^{(j)}}{\partial W_{k,l}^{(i)}} \right)^2, \tag{4.2}$$

which is just the frobenius norm. Note that this equation can be used to calculate the L_2 norm of the parameter gradient for an example or the L_2 norm of the gradient for an individual weight matrix by summing over j or i, respectively.

The proposed method makes use of $\boldsymbol{H}^{(i)}$ whose jth row contains the activation layer $\boldsymbol{h}^{(i)}$ corresponding to $\mathcal{B}^{(j)}$ i.e. $\boldsymbol{H}_j^{(i)} = \phi^{(i)}\left(\boldsymbol{W}^{(i)}\phi^{(i-1)}\left(\dots\left(\boldsymbol{W}^{(1)}\mathcal{B}^{(j)}\right)\dots\right)\right)$. Likewise, define $\boldsymbol{Z}^{(i)}$ for each $\boldsymbol{z}^{(i)}$ and compute $\bar{\boldsymbol{Z}} = \nabla_{\boldsymbol{Z}} C$, which can be computed in a single pass using standard backpropagation. This gives us

$$\left\| \frac{\partial \mathcal{L}^{(j)}}{\partial \boldsymbol{W}^{(i)}} \right\|^2 = \left(\sum_k \left(\bar{Z}_{j,k}^{(i)} \right)^2 \right) \left(\sum_k \left(H_{j,k}^{(i-1)} \right)^2 \right) = \left\| \bar{\boldsymbol{Z}}_j^{(i)} \right\|^2 \left\| \boldsymbol{H}_j^{(i-1)} \right\|^2. \tag{4.3}$$

4.1 Commentary

This method works well with gradient clipping due to the efficient computation of gradient norms. This calculation allows for simple re-scaling of the gradients used in backwards propagation.

Chapter 5

Understanding gradient clipping in incremental gradient methods [7]

This paper considers the class of problems dealing with the minimization of

$$\min_{x\in\mathbb{R}^n} f(x) \qquad \text{where} \qquad f(x) = \frac{1}{m}\sum_{i=1}^{m} f_i(x), \tag{5.1}$$

where $f_i : \mathbb{R}^n \to \mathbb{R}$ are differentiable, non-convex functions. As a side note, this differs from notation that I'm used to, in that x is considered to be the set of parameters while i, or more importantly f_i, is a function of the parameters using the i^{th} component of the dataset to measure the loss. That is, the function $f_i(x)$ determines the loss accumulated when using x as the set of parameters on input i. f is said to be the *objective function* while each f_i is called a *component function*.

The incremental method for SGD is

$$x_{k+1} = x_k - \alpha_k \nabla f_{i_k}(x_k) \quad k = 0, 1, \ldots \tag{5.2}$$

where x_k is the values of the parameters at ieration k, α_k is the learning rate, and f_{i_k} is a uniform randomly chosen component function $i_k \in \{1, 2, \ldots, m\}$ of the dataset. The incremental method for IGC is

$$x_{k,i} = x_{k,i-1} - \alpha_k \nabla f_i(x_{k,i-1}) \quad i = 1, \ldots, m \quad k = 0, 1, \ldots \tag{5.3}$$

where the recursion is defined by base case $x_{0,0}$ and iterative step $x_{k,0} = x_{k-1,m}$ and is more cyclical by nature.

The clipping function $\mathcal{C} : \mathbb{R}^n \to \mathbb{R}^n$ with clipping threshold parameter $\eta > 0$ is defined by

$$\mathcal{C}(g; \eta) = \min\left\{1, \frac{\eta}{\|g\|}\right\} \cdot g. \tag{5.4}$$

Our iterative methods can take advantage of clipping by clipping the gradient so that SGD with clipping is

$$x_{k+1} = x_k - \alpha_k \mathcal{C}(\nabla f_{i_k}(x_k); \eta), \tag{5.5}$$

and IGC with clipping is

$$x_{k,i} = x_{k,i-1} - \alpha_k \mathcal{C}(\nabla f_i(x_{k,i-1}); \eta), \tag{5.6}$$

and is written iteratively (using the notation $x_k = x_{k,0} = x_{k-1,m}$) as

$$x_{k+1} = x_k - \alpha_k \sum_{i=1}^{m} \mathcal{C}(\nabla f_i(x_{k,i-1}); \eta), \tag{5.7}$$

For the theoretical understanding of the clipping function's effects on the convergence of both SGD and IGC, we make the following assumptions:

Bounded below there exists an f^* such that for all $x \in \mathbb{R}^n$, $f(x) \geq f^*$;

Relaxed smoothness there exist constants $L_0, L_1 \in \mathbb{R}^+$ such that $\left\|\nabla^2 f(x)\right\| \leq L_0 + L_1 \left\|\nabla f(x)\right\|$.

The second condition becomes Lipschitz smoothness when $L_1 = 0$. Let $f_k = f(x_k)$ and assume for simplicity from here on out that SGD and IGC have a constant step size α and that IGC's notation for $x_{k,i}$ can be simplified for $x_{k,0}$ to x_k. The paper shows that convergence for the algorithm can be ensured if there exists a constant C such that

$$\langle \sum_{i=1}^{m} \mathcal{C}(\nabla f_i(x_k); \eta), \nabla f_k \rangle \geq C \left\|\nabla f_k\right\|^2 \tag{5.8}$$

i.e. if the magnitude of the clipped gradient projected in the direction of the regular gradient is bounded below by the magnitude of the regular gradient. This value is simplified further as follows.

Let $g_{ki} = \nabla f_i(x_k)$ and note the following properties of g_{ki}

$$g_{ki} = \beta_{ki} \nabla f_k + t_{ki} \quad \text{where} \quad \begin{array}{c} \beta_{ki} = \dfrac{\langle g_{ki}, \nabla f_k \rangle}{\left\|\nabla f_k\right\|^2} = \dfrac{\left\|g_{ki}\right\|}{\left\|\nabla f_k\right\|} \cos\theta_{ki} \\ t_{ki} \in \nabla f_k^{\perp} = \{z | \langle z, \nabla f_k \rangle = 0\} \end{array}. \tag{5.9}$$

Since $\frac{1}{m}\sum_i^m g_{ki} = \nabla f_k$, it is a necessary condition that $\sum_i^m \beta_{ki} = m$ and $\sum_i^m t_{ki} = 0$. Let $\gamma_{ki} = \min\{1, \eta / \left\|g_{ki}\right\|\}$ and note that $\mathcal{C}(g_{ki}; \eta) = \gamma_{ki} g_{ki}$. The paper then shows that

$$\langle \sum_{i=1}^{m} \mathcal{C}(\nabla f_i(x_k); \eta), \nabla f_k \rangle = \sum_{i=1}^{m} \gamma_{ki} \beta_{ki} \left\|\nabla f_k\right\|^2. \tag{5.10}$$

so that Equation 5.8 becomes

$$\sum_{i=1}^{m} \gamma_{ki} \beta_{ki} \geq C, \tag{5.11}$$

that is, *the necessary condition for convergence is that the sum in Equation 5.11 is greater than some positive scalar C.*

There are then essentially two conditions, either of which will ensure the convergence of our algorithm:

1. all g_{ki} can be partitioned into pairs so that if g_{kj} and g_{kl} are a pair then $\beta_{kj} + \beta_{kl} > 0$ and $\|t_{kj}\| = \|t_{kl}\|$;
2. all $\beta_{ki} > 0$ or equivalently all $\cos\theta_{ki} > 0$.

Under the second condition, the paper then shows that a lower bound can be derived making use of the minimum and maximum gradient magnitudes w.r.t. an example i. That is, if we let $m_g = \min_i \|g_{ki}\|$ and $M_g = \max_i \|g_{ki}\|$, and we let constants c_1 and c_2 be defined by the inequalities

$$0 < c_1 \leq \frac{1}{m}\sum_{i=1}^{m}\cos\theta_{ki} \quad \text{and} \quad 0 < c_2 \leq \frac{m_g}{M_g}, \tag{5.12}$$

then the lower bounds for Equation 5.8 are then divided into three scenarios:

1. if $\eta \geq M_g$, then

$$\langle\sum_{i=1}^{m}\mathcal{C}(\nabla f_i(x_k);\eta), \nabla f_k\rangle = m\|\nabla f_k\|^2; \tag{5.13}$$

2. if $\eta \leq m_g$, then

$$\langle\sum_{i=1}^{m}\mathcal{C}(\nabla f_i(x_k);\eta), \nabla f_k\rangle = \left(\eta\sum_{i=1}^{m}\cos\theta_{ki}\right)\|\nabla f_k\|; \tag{5.14}$$

3. and if $m_g < \eta < M_g$, then

$$\langle\sum_{i=1}^{m}\mathcal{C}(\nabla f_i(x_k);\eta), \nabla f_k\rangle \geq m\min\{\eta c_1, c_2\|\nabla f_k\|\}\|\nabla f_k\|. \tag{5.15}$$

The paper concludes by providing the following theorems on the behavior of SGD and IGC with gradient clipping.

Theorem 5.1. *Assuming that we have a lower bound and relaxed smoothness, if*

$$\alpha \leq \frac{c_1}{4\eta L_1} \tag{5.16}$$

then the iterates $\{x_k\}$ generated by Equation 5.5 satisfy

$$\frac{1}{T+1}\sum_{k=0}^{T}\min\{\eta c_1, c_2\|\nabla f_k\|\}\|\nabla f_k\| \leq \frac{2(f_0 - f^*)}{(T+1)\alpha} + 5\alpha\eta^2 L_0 + \frac{\alpha\eta^3 L_1 c_1}{c_2}. \tag{5.17}$$

Theorem 5.2. *Assuming that we have a lower bound, relaxed smoothness, and each component function also satisfies relaxed smoothness i.e.*

$$\left\|\nabla^2 f_i(x)\right\| \leq L_{0i} + L_{1i} \left\|\nabla f_i(x)\right\|, \quad i = 1, \ldots, m; \tag{5.18}$$

if

$$\alpha \leq \min\left\{\frac{c_1}{2m(2\eta L_1 + G)}, \frac{1}{5mL_{0i}}, \frac{1}{8m\eta L_1 i}\right\}, \quad i = 1, \ldots, m; \tag{5.19}$$

then the iterates $\{x_k\}$ *generated by Equation 5.7 satisfy*

$$\frac{1}{T+1}\sum_{k=0}^{T} \min\{\eta c_1, c_2 \left\|\nabla f_k\right\|\} \left\|\nabla f_k\right\| \leq \frac{2(f_0 - f^*)}{(T+1)\alpha m} + 5\alpha m\eta^2 L_0 + \frac{\alpha m \eta^2 c_1 (2\eta L_1 + G)}{2c_2}, \tag{5.20}$$

where

$$G = 5 \max_{1 \leq i \leq m} L_{0i} + 16\eta \max_{1 \leq i \leq m} L_{1i}. \tag{5.21}$$

Chapter 6

Pruning neural networks without any data by iteratively conserving synaptic flow [8]

This paper, while theoretically captivating, is lackluster with providing mathematical commentary on the concepts and proofs. It may be worthwhile to go back and either notate the more mathematical definitions of the paper, or derive my own definitions and reprove their claims using a more solid theoretical foundation. This may even set a better groundwork for future analysis.

6.1 Pruning Algorithms

Pruning algorithms are generally defined by

1. scoring parameters by some metric, and
2. masking parameters according to their scores.

The latter is generally done by removing the parameters (e.g. making the value zero via Hadamard product). This can either be done via global masking or layer-masking, with global-masking performing better but suffering from *layer-collapse*, which is when an entire layer is masked.

Let θ be a collection of network parameters and θ_{prune} be the parameters remaining after pruning. The *compression ratio* ρ of a pruning algorithm is

$$\rho = \frac{|\theta|}{|\theta_{\text{prune}}|} \tag{6.1}$$

. We define $\rho_{\max}$ as the maximal possible compression ratio for a *network* that doesn't lead to layer collapse and the ρ_{cr} as the maximal compression ratio a given

algorithm can achieve without inducing layer collapse. Note that the distinction in the two is $\rho_{\max}$ is maximal for a network while ρ_{cr} is maximal for an algorithm. These definitions motivate the following axiom:

Axiom 6.1 (Maximal Critical Compression)**.** *For any pruning algorithm and any network, we should always have* $\rho_{cr} = \rho_{\max}$.

6.2 Synaptic Saliency

Synaptic saliency is a class of score metrics defined by

$$\mathcal{S}(\theta) = \frac{\partial \mathcal{R}}{\partial \theta} \odot \theta \tag{6.2}$$

where $\mathcal{R}$ is a scalar loss function of the output y of a feed-forward network parameterized by θ. This metric satisfies two conservation laws:

Theorem 6.1 (Neuron-wise Conservation of Synaptic Saliency)**.** *For a feedforward neural network with continuous, homogeneous activation functions* $\phi(x) = \phi'(x)x$*, let* j *be the index of a hidden neuron in layer* i*. The sum of the synaptic saliency for the incoming parameters to a hidden neuron* $\left(\mathcal{S}_j^{(l)_{in}} = \langle \frac{\partial \mathcal{R}}{\partial \theta_j^{(l)}}, \theta_j^{(l)} \rangle\right)$ *is equal to the sum of the synaptic saliency for the outgoing parameters from the hidden neuron* $\left(\mathcal{S}_j^{(l)_{out}} = \langle \frac{\partial \mathcal{R}}{\partial \theta_{:,j}^{(l+1)}}, \theta_{:,j}^{(l+1)} \rangle\right)$.

Theorem 6.2 (Network-wise Conservation of Synaptic Saliency)**.** *The sum of the synaptic saliency across any set of parameters that exactly separates the input neurons* x *from the output neurons* y *of a feedforward neural network with homogeneous activation functions equals* $\langle \frac{\partial \mathcal{R}}{\partial x}, x \rangle = \langle \frac{\partial \mathcal{R}}{\partial y}, y \rangle$.

6.3 Algorithm

For the following theorem, we define the *prune size* to be the total score for the parameters pruned at any iteration and the *cut size* to be the total score for an entire layer.

Theorem 6.3. *If a pruning algorithm with global-masking assigns positive scores that respect layer-wise conservation, and if the prune size is strictly less than the cut size whenever possible, then the algorithm satisfies the Maximal Critical Compression axiom.*

The pruning algorithm defined in this paper uses a loss function

$$\mathcal{R}_{\text{SF}} = \mathbb{1}^\intercal \left(\prod_{l=1}^{L} |\theta^{(l)}| \right) \mathbb{1} \tag{6.3}$$

so that for a fully connected network $\left(f_\theta(x) = \theta^{(L)} \dots \theta^{(1)} \boldsymbol{x}\right)$ the Synaptic Flow score for a parameter $\theta_{i,j}^{(l)}$ is

$$\mathcal{S}_{\mathrm{SF}}(\theta_{i,j}^{(l)}) = \left[\mathbb{1}^\intercal \prod_{k=l+1}^{L} |\theta^{(k)}|\right]_i |\theta_{i,j}^{(l)}| \left[\prod_{k=1}^{l-1} |\theta^{(k)}| \mathbb{1}\right]_j \tag{6.4}$$

i.e. the portion of the l_1-path norm the network has through this parameter. This contributes to the development of Algorithm 1.

Algorithm 1 Iterative Synaptic Flow Pruning (SynFlow)

Require: network f_θ, compression ratio ρ, iteration steps n

$\mu = \mathbb{1}$ ▷ Initialize binary mask

for k in $[1, \dots, n]$ **do**

$\theta_\mu \leftarrow \mu \odot \theta_0$ ▷ Mask parameters

$\mathcal{R} \leftarrow \mathbb{1}^\intercal \left(\prod_{l=1}^{L} |\theta_\mu^{(l)}|\right) \mathbb{1}$ ▷ Evaluate SynFlow objective

$\mathcal{S} = \frac{\partial \mathcal{R}}{\partial \theta_\mu} \odot \theta_\mu$ ▷ Compute SynFlow score

$\tau \leftarrow \left(1 - \rho^{-k/n}\right)$ percentile of $\mathcal{S}$ ▷ Find threshold

$\mu \leftarrow (\tau < \mathcal{S})$ ▷ Update mask

end for

return $f_{\mu \odot \theta}$ ▷ Return masked network

6.4 Commentary

A thought I had: the loss function defined in Equation 6.3 seems to almost indicate a just-past-the-minimum metric for the predefined network. It effectively is just the sum of the entries in the product of the network space. Although I agree with the data agnostic approach, I feel there should be a more suitable loss function which should seek to preserve this score rather than minimize it, or at the very least, should compare this score against some optimum, i.e. the optimum score for that network under those parameters. I also feel that the prune size and cut size should have a greater impact over the outcome of the resulting network, such that the compression ratio is fairly equal for all layers.

Research Question 6.1. Does SynFlow, or other pruning methods for that matter, produce the same network given different initial parameterizations? Furthermore, might we be able to use a genetic algorithm to find the "lottery ticket"?

One thing that needs to be considered with this question is that it's entirely possible that isomorphic networks could be produced, so there may need to be some kind of check that occurs post pruning that confirms whether or not this network has been discovered already.

Chapter 7

Learning Low-rank Deep Neural Networks via Singular Vector Orthogonality Regularization and Singular Value Sparsification [10]

This paper focuses on an improvement on Deep Neural Networks (specifically ones involving convolutional layers) by utilizing the SVD of a convolution and sparsifying the underlying singular values. This is done by representing the kernel $\boldsymbol{K}$ of a convolution as a 4-D tensor with dimension $\boldsymbol{K} \in \mathbb{R}^{n \times c \times w \times h}$ where n is the number of filters (output channels), c is the number of input channels, and w, h are the width and height of the kernel respectively. They break the kernel down into the *channel-wise decomposition* and the *spatial-wise decomposition*.

Under channel-wise decomposition, $\boldsymbol{K}$ is reshaped into a 2-D matrix $\hat{\boldsymbol{K}} \in \mathbb{R}^{n \times cwh}$, which can be decomposed with SVD into $\boldsymbol{U} \in \mathbb{R}^{n \times r}$, $\boldsymbol{V} \in \mathbb{R}^{cwh \times r}$, and $\boldsymbol{s} \in \mathbb{R}^{r}$ where $\boldsymbol{U}, \boldsymbol{V}$ are unitary and $r = \min(n, cwh)$. The convolution then becomes the multiplication of convolutions $\boldsymbol{K}_1 \in \mathbb{R}^{r \times c \times w \times h}$ (reshaped from $\operatorname{diag}\left(\sqrt{\boldsymbol{s}}\right) \boldsymbol{V}^\intercal$) and $\boldsymbol{K}_2 \in \mathbb{R}^{c \times r \times 1 \times 1}$ (reshaped from $\boldsymbol{U} \operatorname{diag}\left(\sqrt{\boldsymbol{s}}\right)$), a simplification discovered by [12].

Under spatial-wise decomposition, $\boldsymbol{K}$ is reshaped into a 2-D matrix $\hat{\boldsymbol{K}} \in \mathbb{R}^{nw \times ch}$, which can be decomposed with SVD into $\boldsymbol{U} \in \mathbb{R}^{nw \times r}$, $\boldsymbol{V} \in \mathbb{R}^{ch \times r}$, and $\boldsymbol{s} \in \mathbb{R}^{r}$ where $\boldsymbol{U}, \boldsymbol{V}$ are unitary and $r = \min(n, cwh)$. The convolution then becomes the multiplication of convolutions $\boldsymbol{K}_1 \in \mathbb{R}^{r \times c \times 1 \times h}$ (reshaped from $\operatorname{diag}\left(\sqrt{\boldsymbol{s}}\right) \boldsymbol{V}^\intercal$) and $\boldsymbol{K}_2 \in \mathbb{R}^{c \times r \times 1 \times 1}$ (reshaped from $\boldsymbol{U} \operatorname{diag}\left(\sqrt{\boldsymbol{s}}\right)$).

In either case, the SVD decomposition of the matrix is stored in memory rather

Figure 7.1: Effect of applying different sparsity-inducing regularizers during SVD training. All models are achieved with Spatial-wise decomposition.

than the convolution, and the convolution is reconstructed from these matrices.o

Since the assurance that $\boldsymbol{U}, \boldsymbol{V}$ are orthogonal is a costly procedure, an orthogonality regularizer is added to the cost of the loss function. This orthogonality regularizer is given by

$$L_o(\boldsymbol{U}, \boldsymbol{V}) = \frac{1}{r^2}\left(\|\boldsymbol{U}^\intercal\boldsymbol{U} - \boldsymbol{I}\|_F^2 + \|\boldsymbol{V}^\intercal\boldsymbol{V} - \boldsymbol{I}\|_F^2\right). \tag{7.1}$$

This assures that the matrices will be approximately semi-unitary. Note that they can not be orthogonal since they are not square.

As part of the sparsification, another regularizer is used for sparsifying the singular values. This regularizer was tested using both the L^1 norm and a *Hoyer regularizer*, given by

$$L^H(\boldsymbol{s}) = \frac{\|\boldsymbol{s}\|_1}{\|\boldsymbol{s}\|_2}. \tag{7.2}$$

The results given by Figure 7 show that while the L^1 regularizer doesn't match the performance/accuracy tradeoff of the hoyer regularizer, allowing for a higher accuracy loss can result in better performance by the L^1 regularizer. I.e. extremely large compression rates which result in higher accuracy loss can provide a more generous reduction in #FLOPs under L^1, whereas having a more moderate compression and a lower accuracy loss has better reduction of #FLOPs under L^H.

The overall objective function is finally given by

$$L(\boldsymbol{U}, \boldsymbol{s}, \boldsymbol{V}) = L_T(\text{diag}\left(\sqrt{\boldsymbol{s}}\right)\boldsymbol{V}^\intercal, \boldsymbol{U}\text{diag}\left(\sqrt{\boldsymbol{s}}\right)) + \lambda_o \sum_{l=1}^{D} L_o\left(\boldsymbol{U}^{(l)}, \boldsymbol{V}^{(l)}\right) + \lambda_s \sum_{l=1}^{D} L_s\left(\boldsymbol{s}^{(l)}\right), \tag{7.3}$$

where L_T is the training loss computed with decomposed layers, L_o is the orthogonality loss from Equation 7.1, L_s is the sparsity-inducing regularization loss (either L^1 or L^H), and λ_o, λ_s are hyperparameters for controlling the effect of the objective functions on the overall loss. The paper demonstrates that the change in accuracy from changing $\lambda_o = 1.0$ to $\lambda_o = 0.0$ is about -2.0%, showing that the orthogonality causes a definite change in performance.

As for pruning, an energy threshold $e \in [0, 1]$ is a hyperparameter which for each layer, a set $\mathbb{K}$ is determined containing the largest number of singular values s.t.

$$\sum_{j \in \mathbb{K}} s_j^2 \leq e \sum_{i=1}^{r} s_i^2. \tag{7.4}$$

The values in $\mathbb{K}$ are then pruned by setting each $s_j = 0$.

Fill in some details on channel-wise and spatial-wise decomposition because I have no idea what either of these things are.

Chapter 8

Stabilizing Gradients for Deep Neural Networks via Efficient SVD Parameterization [11]

Abstract. *Vanishing and exploding gradients are two of the main obstacles in training deep neural networks, especially in capturing long range dependencies in recurrent neural networks (RNNs). In this paper, we present an efficient parametrization of the transition matrix of an RNN that allows us to stabilize the gradients that arise in its training. Specifically, we parameterize the transition matrix by its singular value decomposition (SVD), which allows us to explicitly track and control its singular values. We attain efficiency by using tools that are common in numerical linear algebra, namely Householder reflectors for representing the orthogonal matrices that arise in the SVD. By explicitly controlling the singular values, our proposed Spectral-RNN method allows us to easily solve the exploding gradient problem and we observe that it empirically solves the vanishing gradient issue to a large extent. We note that the SVD parameterization can be used for any rectangular weight matrix, hence it can be easily extended to any deep neural network, such as a multi-layer perceptron. Theoretically, we demonstrate that our parameterization does not lose any expressive power, and show how it potentially makes the optimization process easier. Our extensive experimental results also demonstrate that the proposed framework converges faster, and has good generalization, especially in capturing long range dependencies, as shown on the synthetic addition and copy tasks, as well as on MNIST and Penn Tree Bank data sets.*

The primary contributions of the paper is in

- paramterizing transition matrices through SVD and householder reflectors;
- applying SVD paramterization to RNNs to exert constraints on their transition matrices, and

- verifying their modification outperforms the original RNN networks.

They also theoretically validate their methods as well as show that it alleviates the gradient vanishing/exploding problem.

8.1 SVD and Orthogonality with Householder Reflectors

They define the SVD of a transition matrix $\boldsymbol{W} \in \mathbb{R}^{n\times n}$ as given by $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal$ where $\boldsymbol{\Sigma}$ is the diagonal matrix of singular values and $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{R}^{n\times n}$ are orthogonal.

Definition 8.1. Given a vector $\boldsymbol{u} \in \mathbb{R}^k$, $k \leq n$, the Householder Reflector $\mathcal{H}_k^n(\boldsymbol{u})$ is

$$\mathcal{H}_k^n(\boldsymbol{u}) = \begin{cases} \begin{bmatrix} \boldsymbol{I}_{n-k} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_k - 2\frac{\boldsymbol{u}\boldsymbol{u}^\intercal}{\|\boldsymbol{u}\|^2} \end{bmatrix}, & \boldsymbol{u} \neq \boldsymbol{0} \\ \boldsymbol{I}_n, & \text{otherwise.} \end{cases} \tag{8.1}$$

When n is well-defined, the previous function is shorthanded to $\mathcal{H}_k(\boldsymbol{u})$.

Let $\mathbb{O}(n)$ be the set of $n \times n$ orthogonal matrices. The following maps are established as precursors to the primary theorems of the paper:

$$\begin{aligned} \mathcal{M}_k : \mathbb{R}^k \times \cdots \times \mathbb{R}^n &\to \mathbb{R}^{n\times n} \\ \mathcal{M}_k(\boldsymbol{u}_k, \ldots, \boldsymbol{u}_n) &\mapsto \mathcal{H}_n(\boldsymbol{u}_n)\cdots\mathcal{H}_k(\boldsymbol{u}_k), \end{aligned} \tag{8.2}$$

and

$$\begin{aligned} \mathcal{M}_{k_1,k_2} : \mathbb{R}^{k_1} \times \cdots \times \mathbb{R}^n \times \mathbb{R}^{k_2} \times \cdots \times \mathbb{R}^n \times \mathbb{R}^n &\to \mathbb{R}^{n\times n} \\ \mathcal{M}_{k_1,k_2}(\boldsymbol{u}_{k_1}, \ldots, \boldsymbol{u}_n, \boldsymbol{v}_{k_1}, \ldots, \boldsymbol{v}_n, \boldsymbol{\sigma}) &\mapsto \\ \mathcal{H}_n(\boldsymbol{u}_n)\cdots\mathcal{H}_{k_1}(\boldsymbol{u}_{k_1})\mathrm{diag}(\boldsymbol{\sigma})\mathcal{H}_{k_2}(\boldsymbol{v}_{k_2})\cdots\mathcal{H}_n(\boldsymbol{v}_n). \end{aligned} \tag{8.3}$$

Theorem 8.1. $\mathrm{im}\,(\mathcal{M}_1)$ *is the set of all* $n \times n$ *orthogonal matrices.*

Theorem 8.2. $\mathrm{im}\,(\mathcal{M}_{1,1})$ *is the set of all* $n \times n$ *real matrices.*

Theorem 8.3. $\mathrm{im}\,(\mathcal{M}_{k_1,k_2})$ *includes the set of all orthogonal* $n \times n$ *matrices if* $k_1 + k_2 \leq n + 2$.

These definitions can be extended to the case of $\boldsymbol{W} \in \mathbb{R}^{m\times n}$ by taking

$$\boldsymbol{W} = \boldsymbol{U}(\boldsymbol{\Sigma}|\boldsymbol{0})(\boldsymbol{V}_L|\boldsymbol{V}_R)^\intercal = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}_L{}^\intercal, \tag{8.4}$$

where $\boldsymbol{U} \in \mathbb{R}^{m\times m}$, $\boldsymbol{\Sigma} \in \mathrm{diag}(\mathbb{R}^m)$, and $\boldsymbol{V}_L \in \mathbb{R}^{n\times m}$. Equation 8.3 can then be extended as

$$\begin{aligned} \mathcal{M}_{k_1,k_2}^{m,n} : \mathbb{R}^{k_1} \times \cdots \times \mathbb{R}^m \times \mathbb{R}^{k_2} \times \cdots \times \mathbb{R}^n \times \mathbb{R}^{\min(m,n)} &\to \mathbb{R}^{m\times n} \\ \mathcal{M}_{k_1,k_2}^{m,n}(\boldsymbol{u}_{k_1}, \ldots, \boldsymbol{u}_m, \boldsymbol{v}_{k_1}, \ldots, \boldsymbol{v}_n, \sigma) &\mapsto \\ \mathcal{H}_m^m(\boldsymbol{u}_m)\cdots\mathcal{H}_{k_1}^m(\boldsymbol{u}_{k_1})\hat{\boldsymbol{\Sigma}}\mathcal{H}_{k_2}^n(\boldsymbol{v}_{k_2})\cdots\mathcal{H}_n^n(\boldsymbol{v}_n). \end{aligned} \tag{8.5}$$

where $\hat{\boldsymbol{\Sigma}} = (\mathrm{diag}(\boldsymbol{\sigma})|\boldsymbol{0})$ if $m < n$ and $(\mathrm{diag}(\boldsymbol{\sigma})|\boldsymbol{0})^\intercal$ otherwise. This leads to the following lemma and consequently theorem:

Lemma 8.1. *Given* $\{v_i\}_{i=1}^n$*, define* $V^{(k)} = \mathcal{H}_n^n(\boldsymbol{v}_n)\cdots\mathcal{H}_k^n(\boldsymbol{v}_k)$ *for* $k \in [n]$*. We have:*

$$V_{*,i}^{(k_1)} = V_{*,i}^{(k_2)}, \forall k_1, k_2 \in [n], i \leq \min(n-k_1, n-k_2). \tag{8.6}$$

Theorem 8.4. *If* $m \leq n$*,* $\operatorname{im}\left(\mathcal{M}_{1,n-m+1}^{m,n}\right)$ *is the set of all* $m \times n$ *matrices; else the image of* $\operatorname{im}\left(\mathcal{M}_{m-n+1,1}^{m,n}\right)$ *is the set of all* $m \times n$ *matrices.*

Write up notes on the analysis of this paper.

Chapter 9

Federated Learning: Strategies for Improving Communication Efficiency [5]

This paper outlines a distributed learning method which operates on the basis that most the devices involved in the learning process are heterogenous and have unreliable internet connectivity. The synchronized federated learning algorithm generally follows these steps:

1. a subset of existing clients is selected, each of which downloads the current model;
2. each client in the subset computes an updated model based on their local data;
3. the model updates are sent from the selected clients to the server; and finally
4. the server aggregates these models (typically by averaging) to construct an improved global model.

The main contribution deals with deriving a method of federated learning which doesn't require full model updates being sent back and forth between the client and server, for which they derive two primary approaches:

Structured updates Updates are directly learned from a restricted space that can be parameterized using a smaller number of variables.

Sketched updates A full model is learned and then compressed prior to sending it to the server.

The general scheme is to learn a real matrix $\boldsymbol{W} \in \mathbb{R}^{d_1 \times d_2}$ from data stored across a large number of clients. This is done by sending a current model $\boldsymbol{W}_t$ to a subset

S_t of n_t clients. Each client updates their local model $\boldsymbol{W}_t^i$ (usually via SGD or some learning algorithm) so that its update is given by $\boldsymbol{H}_t^i \triangleq \boldsymbol{W}_i^t - \boldsymbol{W}_t$. The next global update is then given by

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t + \eta_t \boldsymbol{H}_t, \quad \boldsymbol{H}_t = \frac{1}{n_t} \sum_{i \in S_t} \boldsymbol{H}_t^i, \tag{9.1}$$

where η_t is the learning rate.

9.1 Structured Updates

Structured updates send back and forth full-sized models based on a predefined structure. By doing so, there is little loss of information over the communication channel but potential for higher data throughput.

9.1.1 Low Rank

We enforce each $\boldsymbol{H}_t^i \in \mathbb{R}^{d_1 \times d_2}$ to be a low rank matrix of rank at most k for a fixed k by letting $\boldsymbol{H}_t^i = \boldsymbol{A}_t^i \boldsymbol{B}_t^i$ where $\boldsymbol{A}_t^i \in \mathbb{R}^{d_1 \times k}$ and $\boldsymbol{B}_t^i \in \mathbb{R}^{k \times d_2}$. This matrix is further compressed by representing $\boldsymbol{A}_t^i$ as a random matrix defined by a seed and only optimizing $\boldsymbol{B}_t^i$, saving a factor of d_1/k parameters in communication. $\boldsymbol{A}_t^i$ is generated afresh each round and for each client independently.

9.1.2 Random Mask

We restrict each $\boldsymbol{H}_t^i$ to be sparse based on a predefined sparsity pattern with respect to a random seed (similar to low rank). The sparsity pattern is generated afresh each round and for each client independently.

9.2 Sketched Updates

Sketched updates place higher emphasis on communication cost rather than fully structured models.

9.2.1 Subsampling

We only communicate a matrix $\hat{\boldsymbol{H}}_t^i = \boldsymbol{H}_t^i \odot \boldsymbol{M}$ where $\boldsymbol{M}$ is a sparse mask and is generated afresh each round and for each client independently. This can be done s.t. $\mathbb{E}\left[\hat{\boldsymbol{H}}_t\right] = \boldsymbol{H}_t$.

9.2.2 Probabilistic Quantization

Let $\boldsymbol{h} = (h_1, \ldots, h_{d_1 \times d_2}) = \operatorname{vec}\left(\boldsymbol{H}_t^i\right)$, $h_{\max} = \max_j h_j$, and $h_{\min} = \min_j h_j$. We can apply b-bit quantization by dividing $[h_{\min}, h_{\max}]$ into 2^b intervals. Then if h_i

falls in the interval $[h^i_{\min}, h^i_{\max}]$, then the compressed update for index i is defined by

$$\tilde{h}_i = \begin{cases} h^i_{\max}, & \text{with probability } \frac{h_i - h^i_{\min}}{h^i_{\max} - h^i_{\min}} \\ h^i_{\min}, & \text{with probability } \frac{h^i_{\max} - h_i}{h^i_{\max} - h^i_{\min}}. \end{cases} \tag{9.2}$$

Using this, we have that $\mathbb{E}\left[\tilde{\boldsymbol{h}}\right] = \boldsymbol{h}$.

Error bounds on this method can be improved by a factor of $\mathcal{O}(d/\log d)$ by first performing a random rotation on $\boldsymbol{h}$, quantizing, sending the information to the server, then performing the inverse rotation. The paper states that a structured rotation matrix, the product of a Walsh-Hadamard and binary diagonal entry matrix, can have a computational complexity of $\mathcal{O}(d)$ for generating the matrix and $\mathcal{O}(d \log d)$ for applying the matrix.

Part IV

Finance

Chapter 10

An Introduction to Quantitative Finance [2]

It is important to note that I only present the conclusions to propositions presented in this book rather than deriving the proofs themselves. This is especially the case whenever I list out the value of any derivative: these values are derived in the book using various proofs and are *not* defined to have these values in the first place. For the most part, any derivative with maturity T has a definitive value at maturity, but is a random variable up until that point. The book spends a good amount of time providing deterministic values for these r.v.s, and all that is generally listed here are these deterministic values.

10.1 Interest Rates

Definition 10.1. A *notional* or *principal* is an initial deposit or value N.

Definition 10.2. An *interest rate* r is the rate at which a value is increased according to a specified frequency of time.

Suppose we have an account with interest rate r and notional N. The value of the account at time T is

$$N_T = Ne^{rT} \tag{10.1}$$

Throughout finance, there exist discrete and continuous analogs for most equations due the discrete case being more realistic while the continuous case is far more mathematically efficient. That being said, discrete analogs will be used sparsely and we will primarily depend on continuous equations. The discrete versions can be derived from the continuous versions by noting that

$$e^{rT} = \left(1 + \frac{r_m}{m}\right)^{mT} \tag{10.2}$$

where T is commonly denoted as the time in years, m is the frequency the interest is compounded, and r_m is the equivalent interest rate with the discrete frequency.

Table 10.1: Daycount conventions and accrual factors

Daycount convention	α **for 16 December 2011 - 16 March 2012**
act/365	91/365
act/act	15/365 + 76/366
act/360	91/360
30/365	1/4

r_m can then be recovered from r using

$$r_m = m\left(e^{\left(\frac{r}{m}\right)} - 1\right). \tag{10.3}$$

Definition 10.3. A *money market account* is an account compounded continuously at rate r with notional 1. I.e. $M_0 = 1$ and $M_t = e^{rt}$.

Definition 10.4. A *zero coupon bond* (ZCB) with maturity T is an asset that pays 1 at time T (and nothing else). The value of a ZCB at time t for a continuously compounded rate r is denoted as

$$Z(t,T) = e^{-r(T-t)} \tag{10.4}$$

where $Z(T,T) = 1$ by definition. The values $Z(t,T)$ with $0 \leq t \leq T$ are known as *discount factors* or *present values*.

The intuition behind the present values for a ZCB derives from the fact that if we have two portfolios, one a ZCB with maturity T and another that contains cash which will accumulate interest to be worth 1 at time T, then both will be worth the same value at each t leading up to T and thereafter.

Definition 10.5. An *annuity* a series of fixed cashflows at specified times $T_i = 1, \ldots, n$. The value at time t is given by

$$V = C\sum_{i=1}^{n} Z(t,T_i). \tag{10.5}$$

When we substitute in Equation 10.4, we notice that this is just a geometric series. Because of this, if an annuity is payed once a year for M years with rate r, this equates to

$$V = \sum_{i=1}^{n} \frac{1}{(1+r)^i} = \frac{1}{r}\left(1 - \frac{1}{(1+r)^M}\right) \tag{10.6}$$

Definition 10.6. The *accrual factor* is (under the discrete analogy) the frequency which interest is compounded, and is denoted as α.

In Equations 10.2 and 10.3, this is the same as taking $\alpha = 1/m$. Due to the year being uneasily divided by simple values of α, several *daycount conventions* have been made for different settings. We list a sample of these in Table 10.1. The daycount conventions are seemingly deceptive in that they hide various details

behind terminology. For example, two cases are 30/x and x/360, which respectively denote that each month in the period should be counted as only having 30 days and each month in a year should be counted as only having 30 days. I want to emphasize the weird convention because when we encounter a fraction in the work such as 30/act, this means to say that if we specify a period of three months, then this fraction actually denotes

$$\frac{30 * 3}{m_1 + m_2 + m_3} \tag{10.7}$$

where each m_i is the number of actual days in the specified month. Like wise, if we have a daycount act/act, this denotes the number specifies the number of actual days in the period divided by the actual days in the year. If we have a leap year, then the denominator contains 366, whereas if February is included in the daycount, then we may have 28 or 29 days dependent on the year. I hope this helps to explain it better, if not then *oof*.

A few specific cases arise but are only defined in the exercises. I will list those here.

Definition 10.7. *Semi-bond* is the US dollar interest rate which is semi-annual compounding with 30/360 daycount and is denoted as γ_{SB}.

Definition 10.8. *Annual money* is the US dollar interest rate which is annual compounding with act/360 daycount and is denoted as γ_{AM}.

If we know the rate r_α and accrual factor α of a quoted interest rate over a specified period of time, and we are interested in the equivalent rate r_β under a different accrual factor β over the same period of time, then this information can be retrieved by making note of the equivalency

$$r_\alpha \alpha = r_\beta \beta. \tag{10.8}$$

This makes conversions simple as demonstrated by the following example.

Example 10.1. If on 16 December 2011 the act/365 rate for three months is 5%, the act/360 rate is

$$r_\beta = r_\alpha \frac{\alpha}{\beta} = 5\% \frac{\text{act}/365}{\text{act}/360} = 5\% \frac{360}{365} = 4.9315\%. \tag{10.9}$$

Example 10.2. The conversion between semi-bonds and annual money is given by

$$\gamma_{SB} \frac{30}{360} = \gamma_{AM} \frac{\text{act}}{360}, \tag{10.10}$$

so that

$$\gamma_{SB} = \gamma_{AM} \frac{\text{act}}{30}, \tag{10.11}$$

and likewise

$$\gamma_{AM} = \gamma_{SB} \frac{30}{\text{act}}. \tag{10.12}$$

Definition 10.9. A *stock* or *share* is an asset giving ownership in a fraction of a company. The price at time t of a stock is denoted by S_t. The current known price is called the *spot*.

Definition 10.10. A *fixed rate bond* with coupon c and notional N is an asset that pays a coupon cN at a specified frequency (usually a year) and N at its maturity date T. A *floating rate bond* is defined similarly with a variable interest rate.

10.2 Forward Contracts and Forward Prices

Definition 10.11. A *derivative contract* (*derivative*) is a financial contract between two parties whose value derives from the value of another variable.

Remark. Derivatives are often defined by the payout at a specified time. E.g. if S_T indicates the total snowfall in inches at time T then the weather derivative $g(S_T)$ can be defined by

$$g(S_T) = I\{S_T > 50\} = \begin{cases} \$1 & \text{if } S_T > 50 \\ 0 & \text{otherwise,} \end{cases} \tag{10.13}$$

where both S_T and $g(S_T)$ are random variables with unknown values until $T = 1$.

Definition 10.12. A *forward contract* (*forward*) is an agreement between two counterparties to trade a specific asset at maturity T with delivery price K. At a time $t \leq T$, the buyer is *long* the contract while the seller is *short* the contract.

The value at time $t \leq T$ of being long a forward contract is given by $V_K(t, T)$, where by construction we have that $V_K(T, T) = S_T - K$.

Definition 10.13. The *forward price* $F(t, T)$ at current time $t \leq T$ is the delivery price K s.t. $V_K(t, T) = 0$.

Remark. The forward price can be seen as denoting the spot price at maturity, so in essence you will be agreeing at time $t \leq T$ to trade the asset at time T for its actual price. If you had $K < F(t, T)$ then the person long the contract has a better deal since they would pay less than the appreciated value (i.e. you could just sell the asset for profit), whereas $K > F(t, T)$ provides the better deal for the short since they would receive more than the appreciated value.

Table 10.2: Forward prices for various assets

Underlying	**Forward price**
Asset paying no income	$S_t e^{r(T-t)}$
Asset paying known income I	$(S_t - I)e^{r(T-t)}$
Asset paying dividends at rate q	$S_t e^{(r-q)(T-t)}$
Foreign exchange	$X_t e^{(r_\$ - r_f)(T-t)}$

The forward price is decided under various conditions, those being listed in Table 10.2. Given the forward price, we derive the value of being long a forward contract as

$$V_K(t,T) = (F(t,T) - K)e^{-r(T-t)}. \tag{10.14}$$

As a final note, this chapter begins to discuss proof methods for deriving the values of derivatives. It's fairly instructional to review these as they form the basis for evaluating many assets, so we shall include the definitions and some proof examples below.

Definition 10.14. A *replication proof* is one where we prove the value of an asset by demonstrating that if two portfolios always have the same value at time T and if neither add/subtract anything of non-zero value between t and T, then they must have the same value at $t \leq T$.

Definition 10.15. *Arbitrage* is a situation where, starting with an empty portfolio, a sequence of simple market transactions will end up with a positive portfolio value at time T.

Definition 10.16. A *no-arbitrage proof* uses the assumption that no arbitrage situations exist, i.e. all market transactions have an equivalent exchange.

Remark. Most no-arbitrage proofs are constructed similarly to a proof by contradiction.

It's often informative constructing no-arbitrage proofs since they force you to think about situations in which you can exploit an arbitrage if detected. We demonstrate no-arbitrage proofs and replication proofs in the following examples.

Example 10.3. Let S_t be the current stock of a price paying no income. Let r_{BID} be the interest rate at which one can invest/lend money, and r_{OFF} be the interest rate at which one can borrow money, $r_{BID} \leq r_{OFF}$. Both rates are continuously compounded. Using arbitrage arguments, find upper and lower bounds for the forward price of the stock for a forward contract with maturity $T > t$.

Solution. Assume that $F(t,T) > S_t e^{r_{OFF}(T-t)}$. At time t, we could short a forward contract with forward price $F(t,T)$ (which is done at no upfront cost since $V_{F(t,T)}(t,T) = 0$), take out a loan for S_t at rate r_{OFF} for time $T-t$, and purchase the asset for S_t. Then at time T, we would execute the forward contract, selling the stock for $F(t,T)$ and paying back the loan with appreciated value $S_t e^{r_{OFF}(T-t)}$, leaving us with $F(t,T) - S_t e^{r_{OFF}(T-t)} > 0$. This provides an upper bound under the assumption of no-arbitrage.

Assume that $F(t,T) < S_t e^{r_{BID}(T-t)}$. At time t, we could long a forward contract with forward price $F(t,T)$, borrow the asset S_t, sell the asset and invest/lend the remains at the rate r_{BID} for time $T-t$. Then at time T, we receive $S_t e^{r_{BID}(T-t)}$ at the maturity of the deposit, which we can can use $F(t,T)$ of to execute the forward contract and return the underlying asset. This leaves us with $S_t e^{r_{BID}(T-t)} - F(t,T) > 0$. This provides a lower bound under the assumption of no-arbitrage.

Our bounds then come out to

$$S_t e^{r_{BID}(T-t)} \leq F(t,T) \leq S_t e^{r_{OFF}(T-t)}. \tag{10.15}$$

Chapter 11

A Random Walk Down Wall Street [6]

Part V

Differential Privacy

Bibliography

[1] Gagan Aggarwal et al. "Approximation Algorithms for k-Anonymity". In: *Proceedings of the International Conference on Database Theory (ICDT 2005)*. Nov. 2005. URL: http://ilpubs.stanford.edu:8090/645/.

[2] S. Blyth. *An Introduction to Quantitative Finance*. OUP Oxford, 2013. ISBN: 9780199666591. URL: https://books.google.com/books?id=SXbcAAAAQBAJ.

[3] Michel X. Goemans and David P. Williamson. "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming". In: *J. ACM* 42.6 (Nov. 1995), pp. 1115–1145. ISSN: 0004-5411. DOI: 10.1145/227683.227684. URL: https://doi.org/10.1145/227683.227684.

[4] Ian Goodfellow. *Efficient Per-Example Gradient Computations*. 2015.

[5] Jakub Konečný et al. "Federated Learning: Strategies for Improving Communication Efficiency". In: *CoRR* abs/1610.05492 (2016). arXiv: 1610.05492. URL: http://arxiv.org/abs/1610.05492.

[6] Burton. G. Malkiel. *A Random Walk Down Wall Street*. Norton, New York, 1973.

[7] Jiang Qian et al. "Understanding gradient clipping in incremental gradient methods". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2021, pp. 1504–1512.

[8] Hidenori Tanaka et al. "Pruning neural networks without any data by iteratively conserving synaptic flow". In: *CoRR* abs/2006.05467 (2020). arXiv: 2006.05467. URL: https://arxiv.org/abs/2006.05467.

[9] Roberto Tempo and Hideaki Ishii. "Monte Carlo and Las Vegas Randomized Algorithms for Systems and Control*: An Introduction". In: *European Journal of Control* 13.2 (2007), pp. 189–203. ISSN: 0947-3580. DOI: https://doi.org/10.3166/ejc.13.189-203. URL: https://www.sciencedirect.com/science/article/pii/S0947358007708191.

[10] Huanrui Yang et al. "Learning Low-rank Deep Neural Networks via Singular Vector Orthogonality Regularization and Singular Value Sparsification". In: *CoRR* abs/2004.09031 (2020). arXiv: 2004.09031. URL: https://arxiv.org/abs/2004.09031.

[11] Jiong Zhang, Qi Lei, and Inderjit Dhillon. "Stabilizing Gradients for Deep Neural Networks via Efficient SVD Parameterization". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 5806–5814. URL: https://proceedings.mlr.press/v80/zhang18g.html.

[12] Xiangyu Zhang et al. "Accelerating Very Deep Convolutional Networks for Classification and Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.10 (2016), pp. 1943–1955. DOI: 10.1109/TPAMI.2015.2502579.