Deep Linear Networks

Particia Suriani Sumantri

Session 1: Layers are automatically balanced

References

- **Algorithmic Regularization in Learning Deep Homogeneous Models: Layers are Automatically Balanced**, Gunasekar, S., Lee, J., Soudry, D., Srebro, N. (2018).
- **The geometry of the deep linear network**, Govindarajan (2021).

The Learning Problem

Given a dataset

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\} \subseteq \mathbb{R}^{d_x} \times \mathbb{R}^{d_y}$$

the goal is to learn a function $f : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$ from a dataset of input-output pairs $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$. We do a deep linear network, parameterized by weights $\mathbf{W}$ via our function approximator $f_{\mathbf{W}}(\mathbf{W})$.

Loss Function

We quantify prediction error using a loss function:

$$\ell(\mathbf{W}) = \sum_{i=1}^{N} \frac{1}{2} \|f_{\mathbf{W}}(x_i) - y_i\|_2^2$$

Optimization

We find the best weights by minimizing the loss using the continuous-time dynamics of **Gradient Flow**:

$$\frac{d}{dt}\mathbf{W}(t) = -\nabla_{\mathbf{W}}\ell(\mathbf{W}(t))$$

Gradient Flow: Key Questions

- Convergence guarantee?
- Convergence rate?
- Architecture-rate: Characterize the minimizer reached?
- Effect of loss-loss is floating point errors, etc?

Our Focus: Deep Homogeneous Models

A feed-forward neural network composition function:

$$f_{\mathbf{W}}(\mathbf{W}) = W^{(L)} W^{(L-1)} \cdots W^{(1)} x$$

- We focus on networks where the activation function ϕ is **homogeneous**.

A Key Property: Homogeneity

An activation function ϕ is homogeneous of degree τ if for any scalar $\alpha \geq 0$:

$$\phi(\alpha z) = \alpha^{\tau} \phi(z)$$

- The property holds for many popular activations:
  - Linear: $\phi(z) = \alpha z$
  - ReLU: $\phi(z) = \max(0, z)$
  - Leaky ReLU: $\phi(z) = \max(\alpha z, z)$ for $0 < \alpha < 1$
- A bias will not hinder the homogeneity of a layer.

The Consequence: Scale Invariance

Homogeneity in activations leads to a scale invariance in the entire network. We can scale one layer's weights and scale down the next layer, and the overall function remains unchanged:

$$f\left(W_1, \ldots, \alpha W^{(\ell)}, \ldots\right) = f\left(W_1, \ldots, \alpha^{-1} W^{(\ell+1)}, \ldots\right)$$

This is because the scalar α passes through the homogeneous activation ϕ and cancels out.

The Optimization Puzzle

The scale invariance creates a fundamental problem for optimization:

- **Non-compactness**: A flat loss plateau of all equivalent, there are infinitely many other ways that achieve the same loss.
- **Non-convexity**: Even if we fix a group of equivalents, there are infinitely many other ways that do not.

$$\left\{W^{(1)}, W^{(2)}\right\} \text{ is equivalent to } \left\{\alpha W^{(1)}, \frac{1}{\alpha} W^{(2)}\right\}$$

- **Unbounded Minima**: We can make the norms of the weights arbitrarily large (by choosing a large α) without changing the loss:

$$\|\alpha W^{(1)}\|_2 \to \infty \text{ as } \alpha \to \infty$$

Why Classical Theory Fails

Many classical theorems for guaranteed convergence of optimization algorithms rely on assumptions that are violated here.

Classical Assumption: The loss function is coercive (i.e., $\|\mathbf{W}\|_2 \to \infty \implies \mathcal{L}(\mathbf{W}) \to \infty$) so that the iterates are constrained to a compact set.

- In deep homogeneous models, the loss landscape has "valleys" extending to infinity where the loss is zero.
- Gradient descent on these valleys can be guaranteed to stay in a bounded region, making convergence analysis possible.

The "Obvious" Fix: Regularization for General Models

For a deep network, the most common form of explicit regularization is L2 regularization, also known as weight decay:

$$\mathcal{L}_{\text{reg}}(\mathbf{W}) = \mathcal{L}(\mathbf{W}) + \frac{\lambda}{2} \|\mathbf{W}^{(\ell)}\|_2^2$$

- This regularizer penalizes large weights in any layer, discouraging norms from becoming excessively large.
- This makes the objective coercive (the loss grows as weights increase), ensuring the parameters remain in a bounded set where analysis is more straightforward.

Example: Matrix Factorization

The objective is to find factors $U \in \mathbb{R}^{m \times r}$ and $V \in \mathbb{R}^{n \times r}$ that approximate a target matrix $M^* \in \mathbb{R}^{m \times n}$ where the rank is at most r.

$$\min_{U \in \mathbb{R}^{m \times r}, V \in \mathbb{R}^{n \times r}} \|UV^\top - M^*\|_F^2$$

- This objective is homogeneous: for any $\alpha \neq 0$, the function is unchanged if we replace (U, V) with $(\alpha U, \frac{1}{\alpha} V)$.
- The regularizer also becomes unbalanced and grow infinitely large.

The "Obvious" Fix: Regularization for Matrix Factorization

A common theoretical workaround is to add a penalty term to the loss that forces the norms of the factors to be balanced:

$$\min_{U,V} \left\|UV^\top - M^*\right\|_F^2 + \frac{\lambda}{2}\left(\|U\|_F^2 - \|V\|_F^2\right)^2$$

- This regularizer removes the homogeneity and forces $\|U\|_F \approx \|V\|_F$.
- The resulting objective is now "well-conditioned" (e.g., coercive), and standard theory can be applied to prove convergence.

But... Is It Necessary? The Empirical Surprise

Let's compare Gradient Descent on the original vs. the regularized problem for matrix factorization. (Fig 1 from Liu et al. 2016).

(a) Balanced Initialization (b) Unbalanced Initialization

The unregularized problem converges just as well, and surprisingly, the ratio of the factor norms remains unbalanced. The layers are balanced on their own!

A similar observation holds for Deep Linear Networks (DLN). (Fig 2 from Du et al. 2018).

(a) Balanced Initialization (b) Unbalanced Initialization

The Central Question

Why does Gradient Descent automatically balance the layers and converge?

The Answer: Implicit Regularization

- The combination of the over-parameterized, homogeneous model structure and first-order optimization methods leads to a hidden property: **Implicit Regularization**.
- The optimization dynamics on these models implicitly enforce a constraint that keeps the layers balanced.
- This prevents the parameters from drifting to infinity or an unbalanced state.

Formalizing the Dynamics: Gradient Flow

To analyze this, we study the dynamics of Gradient Descent in the continuous-time limit. The simple **GF** models over time t according to the differential equation:

$$\frac{d}{dt}\mathbf{W}(t) = -\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W}(t))$$

This capture the behavior of GD with an infinitesimally small learning rate.

Result 1: Exact Autobalancing

This system is the original dynamics of Gradient Descent, but with a continuous-time component added. It ensures **single** balance.

Theorem 2.1 (Du, Hu, Lee 2018). Under gradient flow, for any record ϕ and any hidden layer j, the difference between the squared norms of the incoming and outgoing weights is conserved:

$$\left\|W_j(t)\right\|_F^2 - \left\|W_{j-1}(t)\right\|_F^2 = 0$$

Interpretation

The squared norm of the incoming weights to a neuron, $\left\|W^{(j)}(t)\right\|_F^2$, is the vector of weights leading into neuron j.

The squared norm of the outgoing weights, $\left\|W^{(j+1)}(t)\right\|_F^2$, is the vector of weights coming from neuron j.

The theorem states that the difference between these two quantities is an **invariant** of **motion**. It does not change during training.

This explains the balanced phenomenon we saw in the empirical plots. It means that the initial imbalance cannot be fixed, or learning rebalancing, we precisely this. They add the initial norms to the roughly equal.

A Stronger Invariance for Linear Networks

For the special case of linear activations ($\phi(z) = z$), we can strengthen matrix-level quantity is conserved:

Theorem 2.2 (Du, Hu, Lee 2018). If activation between layers j & and $j+1$ is linear, then certain quantity is conserved:

$$W_j(t)W_j(t)^\top - W_{j-1}(t)^\top W_{j-1}(t) = 0$$

This shows that not just the difference of squared norms, but a more structured matrix difference, remains constant throughout training.

Connecting to Matrix Factorization

The matrix factorization problem $\min_U \|UU^\top - M\|_F^2$ is equivalent to a deep linear network with two layers ($W_1 = U$, $W_2 = U^\top$). In this case, the stronger invariance from Theorem 2.2 ensures that $W_1W_1^\top = W_2^\top W_2$ or $UU^\top = U^\top U$.

$$W^{(1)}W^{(1)\top} - W^{(2)\top}W^{(2)} = U^\top U - UU^\top = 0$$

A notoriously crude gradient flow. This implies that if we initialize with $\|U\|_F^2 \approx \|V\|_F$, the balance will be maintained by the dynamics.

From Theory to Practice: Open Questions

Gradient flow with perfectly balanced initialization is an idealization. In practice, we use Gradient Descent with a finite learning rate:

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \eta \nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W}_k)$$

- What happens if the initialization isn't perfectly balanced?
- How does the discrete nature of Gradient Descent affect the autobalancing property?
- Can we prove **convergence** to a global minimum for these unregularized problems?

Answers for Matrix Factorization

Lawrence 2018 (Du, Hu, Lee 2018): For square matrix factorization, assume the entries of M_{ij} are $N(0, \sigma)$ distributed. Let λ_1 be the largest eigenvalue of M and λ_r be the smallest non-zero eigenvalue. Assume $\lambda_1 \geq 1$ and $\lambda_r \leq \frac{1}{\sqrt{r}}$. Then with high probability, the iterative dynamics converge to a global minimum.

$$\|UU^\top - M\|_F^2 \to 0$$

For the rank-1 case, even stronger results hold, guaranteeing not just balance but also their convergence.

Theorem 3.2 (Du, Hu, Lee 2018). For rank-1 matrix factorization, assuming $\alpha = \|U_0\|_2^2 \|V_0\|_2^2$ and $\alpha \leq \frac{\sigma_1^2(M)}{2}$ is sufficiently small, if we use a sufficiently small learning rating, then with probabilities at least $1 - \delta$, $\|U_t\|_2^2 \to \|V_t\|_2^2$ and $\|U_t V_t^\top - M\|_F^2 \to \sigma$.

Proof Sketch: Unrolling the Mechanism of Autobalancing

To ease the core idea, we'll prove the autobalancing result for the simplest case: a two-layer linear network with a scalar output, $f_{\mathbf{W}} = W_2 W_1 x$. The goal is to show:

$$\frac{d}{dt}\|W_1\|_2^2 = \frac{d}{dt}\|W_2\|_2^2$$

Task: Backpropagation (Chain Rule)

Let $\mathcal{L} = \sum_{i=1}^{N} \ell_i$ be the gradient with respect to the output for a single data point, and let $\delta_i = \frac{\partial \ell_i}{\partial f}$.

- Gradient for the second layer: $\nabla_{W_2}\mathcal{L} = \delta \cdot x^\top$
- Gradient for the first layer: $\nabla_{W_1}\mathcal{L} = (W_2^\top \delta) \cdot x^\top$

Calculations

First, we compute the time derivative of $\|W_2\|^2$:

$$\frac{d}{dt}\|W_2\|^2 = \left\langle \frac{\partial}{\partial t} W_2, W_2 \right\rangle + \left\langle W_2, \frac{\partial}{\partial t} W_2 \right\rangle$$
$$= -\langle \nabla_{W_2}\mathcal{L}(\mathbf{W}), W_2 \rangle - \langle W_2, \nabla_{W_2}\mathcal{L}(\mathbf{W}) \rangle$$
$$= -\langle \delta \cdot x^\top, W_2 \rangle - \langle W_2, \delta \cdot x^\top \rangle$$
$$= -\langle W_2^\top \delta, x \rangle - \langle x, W_2^\top \delta \rangle$$

Next, we compute the time derivative of $\|W_1\|^2$:

$$\frac{d}{dt}\|W_1\|^2 = \left\langle \frac{\partial}{\partial t} W_1, W_1 \right\rangle + \left\langle W_1, \frac{\partial}{\partial t} W_1 \right\rangle$$
$$= -\langle \nabla_{W_1}\mathcal{L}(\mathbf{W}), W_1 \rangle - \langle W_1, \nabla_{W_1}\mathcal{L}(\mathbf{W}) \rangle$$
$$= -\langle (W_2^\top \delta) \cdot x^\top, W_1 \rangle - \langle W_1, (W_2^\top \delta) \cdot x^\top \rangle$$
$$= -\langle W_2^\top \delta, W_1 x \rangle - \langle x, W_1^\top W_2^\top \delta \rangle$$

The Punchline: Perfect Cancellation

The two calculations are exactly the same. Therefore, their difference is zero, and the quantity $\|W_1\|_2^2 - \|W_2\|_2^2$ is conserved.

Summary of Findings

- Deep homogeneous models have a scale invariance that makes classical optimization analysis difficult.
- However, first-order optimization on these over-parameterized models exhibits a powerful implicit regularization.
- It automatically balances the norms of the layers, preventing the weights from becoming pathologically unbalanced.
- This has been rigorously proven for gradient flow (exact balancing) and then extended to practical gradient descent (approx balancing).
- This core mechanism lies in the symmetric structure of backpropagation.

Prerequisites for Next Session

Riemannian Geometry

A smooth **Manifold** M is a space that is locally diffeomorphic to a Euclidean space $\mathbb{R}^n$.

At each point $p \in M$, the **Tangent Space** T_pM is the vector space of all velocity vectors of smooth curves on M passing through p.

Example: For the manifold $M = \mathbb{R}_{m,n}(\mathbb{R})$, the space of matrices, the tangent space at any $W \in M$ is just a copy of the space of matrices itself: $T_W M \cong M_{m,n}(\mathbb{R})$.

The Riemannian Metric & Gradient

A **Riemannian Metric** g provides an inner product $\langle \cdot, \cdot \rangle_p$ on each tangent space T_pM. This allows each tangent space with a Hilbert space structure.

Definition: The Riemannian Gradient. Given a smooth function $f : M \to \mathbb{R}$ on a manifold (M, g), the gradient $\nabla_g f$ is a vector field on M that represents the exterior derivative df_p in the sense:

$$g_p(v, \nabla_g f(p)) = df_p(v) \quad \text{for all } v \in T_pM$$

The existence and uniqueness of $\nabla_g f$ are guaranteed by the Riesz Representation Theorem on the Hilbert space (T_pM, g_p).

Gradient Example 1: The Space of Matrices with Frobenius Metric

The manifold $\mathbb{R}_{m,n}(\mathbb{R})$ with the standard Frobenius metric $g_W(Z_1, Z_2) = \mathrm{Tr}(Z_1^\top Z_2)$ is a flat space. The gradient of a function $f(W)$ is simply its matrix of partial derivatives, which is the standard matrix calculus gradient. The Euclidean gradient of the loss function ℓ analyzed in our notes:

$$\mathrm{grad}_g f = \nabla f(W).$$

Gradient Example 2: A Preconditioned Metric

Let $M = \mathbb{R}_{m,n}(\mathbb{R})$ and $P(W)$ be a smooth, symmetric positive definite (SPD) operator. Define a new metric at W by $g_W(Z_1, Z_2) = \mathrm{Tr}(Z_1^\top P(W) Z_2)$. To find the Riemannian gradient $\mathrm{grad}_g f$ of some loss function, we solve the linear system:

$$\mathrm{d}f_W(Z) = \mathrm{Tr}(\nabla f(W)^\top Z)$$

Substituting the definitions for the metric and the directional derivative, solving $\mathrm{Tr}(\mathrm{grad}_g f \cdot Z)$ for every tangent direction Z:

$$\mathrm{Tr}(\mathrm{grad}_g f \cdot P^\top Z) = \mathrm{Tr}(\nabla f(W)^\top Z)$$

Since this must hold for all tangent vectors Z, we can equate the matrix terms, which yields:

$$\mathrm{grad}_g f = P^{-1} \nabla f(W).$$

The Takeaway: The gradient of a function is not absolute; it depends on the chosen geometry (metric). Changing the metric from the standard one to a preconditioned one transforms the gradient itself.

Matrix Decompositions

We will also need two fundamental tools from linear algebra: the Singular Value Decomposition (SVD) and the Polar Decomposition.

Singular Value Decomposition (SVD)

Theorem. For any real matrix $M \in \mathbb{R}^{m \times n}$, there exists a decomposition $M = U\Sigma V^\top$ where:

- $U \in \mathbb{R}^{m \times m}$ is an **orthogonal** matrix (left singular vectors).
- $V \in \mathbb{R}^{n \times n}$ is an **orthogonal** matrix (right singular vectors).
- $\Sigma \in \mathbb{R}^{m \times n}$ is a rectangular diagonal matrix of non-negative singular values.

Polar Decomposition

Any square matrix $M \in \mathbb{R}_n^n$ can be decomposed into a "stretch" (a symmetric positive semi-definite matrix) and a "rotation" (an orthogonal matrix).

- **Left Polar Decomposition**: $M = QP$, where $Q \in \mathbb{R}^{n \times n}$ is orthogonal and $P \in \mathbb{R}^{n \times n}$ is symmetric PSD.
- **Right Polar Decomposition**: $M = SQ$, where $S \in \mathbb{R}^{n \times n}$ is symmetric PSD and $Q \in \mathbb{R}^{n \times n}$ is orthogonal.

Connection between SVD & Polar Decomposition

The two decompositions are intimately linked. Given the SVD $M = U\Sigma V^\top$, we can construct the polar factors:

- Stretch matrix: $P = V\Sigma V^\top$ (or $S = U\Sigma U^\top$)
- Orthogonal matrix: $Q = UV^\top$

Conversely, given the polar factors $M = QP$, we can find the SVD by taking the eigenvalue/eigenvector of the stretch matrix $P = V\Sigma V^\top$ and substituting it back in: $M = QV\Sigma V^\top$. Let $U = QV$. Then we have $M = U\Sigma V^\top$.

Thank You!

Any questions?