CSC 2541: Neural Net Training Dynamics
Lecture 3 - Metrics

Roger Grosse

University of Toronto, Winter 2021
Today’s question: how to measure the “distance” between neural networks?

- Euclidean distance between weights isn’t very meaningful, since neural nets are complicated nonlinear functions
- Instead, we’ll look at distance in the space of functions represented by the networks
- Taking the second-order Taylor approximation of function space distance gives a metric, which we can build interesting algorithms out of
- If the outputs represent probabilities, this yields the Fisher-Rao metric

Main motivating example: optimization (natural gradient descent)
Motivating Example

- Here’s the Rosenbrock function, a famous toy problem from optimization:

\[ h(x_1, x_2) = (a - x_1)^2 + b(x_2 - x_1^2)^2 \]

- Gradient descent bounces across the valley and gets stuck:
Motivating Example

- We can understand the Rosenbrock function as a composition of two functions, analogously to cost functions in ML. This is known as *composite optimization*.

\[
\begin{align*}
\mathcal{J}(x_1, x_2) &= \mathcal{L}(f(x_1, x_2)) \\
f(x_1, x_2) &= (a - x_1, \sqrt{b(x_2 - x_1^2)}) \\
\mathcal{L}(z_1, z_2) &= z_1^2 + z_2^2.
\end{align*}
\]

- Here’s the gradient descent trajectory in **parameter space** and **output space**:
Motivating Example

- If we could do gradient descent on the outputs, then it would converge instantly. Of course, this is cheating.

![Diagram](image)

- This solution doesn’t directly carry over to neural nets
  - Mapping from parameters to weights may not be invertible, or not easily invertible
  - Will the solution generalize?

- But we can achieve a similar effect using proximal optimization
Proximal Optimization
Suppose we want to minimize $J(w)$. The proximal point method trades off the cost and the distance from the current iterate:

$$w^{(k+1)} = \operatorname{prox}_{J,\lambda}(w^{(k)}) = \arg \min_u \left[ J(u) + \lambda \rho(u, w^{(k)}) \right]$$

- $\rho$ is a dissimilarity function (like a distance, but doesn’t need to satisfy all the axioms)
- $\lambda \rho(u, w^{(k)})$ is the proximity term, and $\operatorname{prox}_{f,\lambda}$ is the proximal operator
Proximal Optimization

- **Example:** \( \rho(w, w') = \frac{1}{2} \| w - w' \|^2 \)

- **Note:** this is an idealized update rule, not something we can efficiently implement. E.g., if \( \lambda = 0 \), it simply optimizes \( J \) directly.
Proximal Optimization

\[ w^{(k+1)} = \text{prox}_{\mathcal{J},\lambda}(w^{(k)}) = \arg \min_u \left[ \mathcal{J}(u) + \lambda \rho(u, w^{(k)}) \right] \]

- Setting the gradient equal to 0 and rearranging:

\[ \text{prox}_{\mathcal{J},\lambda}(w^{(k)}) = u_\star = w^{(k)} - \lambda^{-1} \nabla \mathcal{J}(u_\star) \]

- This is implicit gradient descent: it’s almost a gradient descent update, except that the gradient is evaluated at \( u_\star \).
  - Not a practical algorithm, but often convenient to analyze.
Proximal Optimization

- **Approximation 1**: linearize the cost around the current weights

  \[
  \text{prox}_{\mathcal{J}, \lambda}(w^{(k)}) = \arg \min_u \left[ \mathcal{J}(w^{(k)}) + \nabla \mathcal{J}(w^{(k)})^\top (u - w^{(k)}) + \lambda \rho(u, w^{(k)}) \right]
  \]

  \[
  = \arg \min_u \left[ \nabla \mathcal{J}(w^{(k)})^\top u + \lambda \rho(u, w^{(k)}) \right]
  \]

  Setting the gradient to 0:

  \[
  \nabla_u \rho(u^*, w^{(k)}) = -\lambda^{-1} \nabla \mathcal{J}(w^{(k)}).
  \]

- This is called **mirror descent**, for reasons explained in the readings
  - A foundational technique in online learning
  - Closed form solutions for many \(\rho\) of interest (e.g. KL divergence)
  - The case of squared error reduces to ordinary gradient descent
Approximation 2: Take the infinitesimal limit, $\lambda \to \infty$.

Then we take the first-order Taylor approximation to $J$ and second-order Taylor approximation to $\rho$:

$$\rho(u, w^{(k)}) = \frac{1}{2} (u - w^{(k)})^\top G (u - w^{(k)}) + O(\|u - w^{(k)}\|^3)$$

$$G = \nabla^2_{u} \rho(u, w^{(k)}) \bigg|_{u=w^{(k)}}$$

$G$ is the metric matrix
Proximal Optimization

- Plugging in both approximations:

\[
\text{prox}_{\mathcal{J}, \lambda}(w^{(k)}) = \arg\min_u \left[ \nabla \mathcal{J}(w^{(k)})^\top u + \frac{\lambda}{2} (u - w^{(k)})^\top G(u - w^{(k)}) \right]
\]

- Optimal solution:

\[
u_\star = w^{(k)} - \lambda^{-1} G^{-1} \nabla \mathcal{J}(w^{(k)})
\]

- For certain \( \rho \), \( G^{-1} \nabla \mathcal{J}(w^{(k)}) \) is called the natural gradient

- If \( \rho \) is squared Euclidean distance, this is just ordinary gradient descent
Approximation 3: Second-order Taylor approximations to both $\mathcal{J}$ and $\rho$

Update rule:

$$w^{(k+1)} = w^{(k)} - (H + \lambda G)^{-1} \nabla \mathcal{J}(w^{(k)})$$

If $\rho$ is squared error, then $G = I$, and this gives a damped Newton update:

$$w^{(k+1)} = w^{(k)} - (H + \lambda I)^{-1} \nabla \mathcal{J}(w^{(k)})$$

Relationship to trust-region methods (see readings for details)
Fisher Information
Fisher Information

- So far we’ve only considered squared error proximity terms. If we’re optimizing over probability distributions, a more meaningful dissimilarity function is KL divergence:

\[ D_{KL}(q \parallel p) = \mathbb{E}_{x \sim q} \left[ \log q(x) - \log p(x) \right] \]

- Doesn’t satisfy some of the axioms of distance metrics (symmetry, triangle inequality), but that’s OK

- Nonnegative, equals 0 iff \( p = q \)

- Information theoretic interpretation as relative entropy: the number of bits wasted if you encode data from \( q \) using a code designed for \( p \)

- This formula doesn’t mention the parameters, so it’s an intrinsic dissimilarity function
  - But we’ll typically optimize over a parametric family \( \{p_\theta\} \) parameterized by \( \theta \)
The metric matrix is the **Fisher information matrix** (derivation in the readings):

\[
\nabla^2_u D_{KL}(p_u \parallel p_\theta) \bigg|_{u=\theta} = F_\theta \\
= \text{Cov}_{x \sim p_\theta}(\nabla_\theta \log p_\theta(x)) \\
= \mathbb{E}_{x \sim p_\theta} \left[ (\nabla_\theta \log p_\theta(x))(\nabla_\theta \log p_\theta(x))^\top \right]
\]

The vectors \( \nabla_\theta \log p_\theta(x) \) are called **Fisher scores**

Intuitively, \( \Delta \theta^\top F_\theta \Delta \theta \) tells you how much the distribution changes if you adjust the parameters by \( \Delta \theta \)
Fisher Information

Fisher metric for univariate Gaussians:

\[ \sigma \]
\[ \mu \]
Fisher Information

- Proximal operator:

  \[ \text{prox}_{\mathcal{J},\lambda}(\theta) = \arg \min_u [\mathcal{J}(u) + \lambda D_{KL}(p_u \parallel p_\theta)] \]

- Infinitesimal limit:

  \[
  \theta^{(k+1)} = \theta^{(k)} - \alpha F_\theta^{-1} \nabla \mathcal{J}(\theta^{(k)}) \\
  = \theta^{(k)} - \alpha \tilde{\nabla} \mathcal{J}(\theta^{(k)}),
  \]

  where \( \alpha = \lambda^{-1} \)

- The vector \( \tilde{\nabla} \mathcal{J}(\theta) = F_\theta^{-1} \nabla \mathcal{J}(\theta) \) is the natural gradient, and the update rule is natural gradient descent

- The term natural indicates that the update direction is equivalent in any coordinate system. This works because KL divergence is an intrinsic dissimilarity function.
Exponential Families

(omitted due to time constraints; see the NNTD readings)
Function Space Distance
Motivations

- “Output space gradient descent” can be more efficient
- Natural gradient descent is invariant to parameterization

Can we do the same for neural nets?

**Idea:** define a metric on the outputs of a network and pull it back to weight space

**Pullback** of a function:

\[ f^*g(x_1, \ldots, x_K) = g(f(x_1), \ldots, f(x_K)) \]
Function Space Distance

- We compute the metric matrix by taking a second-order Taylor approximation to $f^*\rho$ around the current point.
- The decomposition is similar to the Gauss-Newton Hessian (Lecture 2), except this time it’s exact.

\[
G_x = \nabla_x^2 f^*\rho(x, x_0) \bigg|_{x=x_0} \nabla_z^2 \rho(z, z_0) \bigg|_{z=z_0} \nabla_x^2 [f(x)]_a = 0
\]

\[
= J_{zx}^\top G_z J_{zx}
\]
Let’s revisit the Rosenbrock function. Instead of doing output space gradient descent, put a Euclidean metric on the outputs and pull it back to parameter space.

\[ \rho_{\text{euc}}(z, z') = \frac{1}{2} \| z - z' \|^2 \]

\[ f^*\rho_{\text{euc}}(x, x') = \frac{1}{2} \| f(x) - f(x') \|^2 \]

- \( G_z = I \), so

\[ G_x = J_{zx}^\top G_z J_{zx} = J_{zx}^\top J_{zx} \]

- Approximating the proximal point update with this metric:
Function Space Distance

- For neural nets, **function space distance** measures how dissimilar the outputs are *in expectation*:

\[
\rho_{\text{pull}}(\mathbf{w}, \mathbf{w}') = \mathbb{E}_{\mathbf{x}}[f_{\mathbf{x}}^*\rho(\mathbf{w}, \mathbf{w}')] = \mathbb{E}_{\mathbf{x}}[\rho(f(\mathbf{w}, \mathbf{x}), f(\mathbf{w}', \mathbf{x}))]
\]

or the finite sample version:

\[
\rho_{\text{pull}}(\mathbf{w}, \mathbf{w}') = \frac{1}{N} \sum_{i=1}^{N} \rho(f(\mathbf{w}, \mathbf{x}^{(i)}), f(\mathbf{w}', \mathbf{x}^{(i)}))
\]
Function Space Distance

- Second-order Taylor approximation:

\[ G_w = \nabla^2_w \rho_{\text{pull}}(w, w') \bigg|_{w=w'} \]
\[ = \mathbb{E}_x \left[ \nabla^2_w \rho(f(w, x), f(w', x)) \right] \]
\[ = \mathbb{E}_x [J_{zw}^\top G_z J_{zw}] . \]

- I’ll call this the pullback metric (there isn’t a standard term in our field)

- We’ll overload notation by using \( G \) for both the pullback metric and the Gauss-Newton Hessian. This is OK since the two matrices often coincide.
Matrix-Vector Products

- Computing matrix-vector products:

```python
def pullback_mvp(f, rho, w, v):
    z0, R_z = jvp(f, (w,), (v,))
    rho_z0 = lambda z: rho(z, z0)
    R_gz = hvp(rho_z0, z0, R_z)
    _, f_vjp = vjp(f, w)
    return f_vjp(R_gz)[0]
```

- Compare with Gauss-Newton Hessian-vector products:

```python
def gnhvp(f, L, w, v):
    z, R_z = jvp(f, (w,), (v,))
    R_gz = hvp(L, z, R_z)
    _, f_vjp = vjp(f, w)
    return f_vjp(R_gz)[0]
```
Connection to Gauss-Newton Hessian
Recall:
- Gauss-Newton Hessian: \( G = \mathbb{E}_x [ J_{zw}^\top H_z J_{zw} ] \)
- Pullback metric: \( G = \mathbb{E}_x [ J_{zw}^\top G_z J_{zw} ] \)

These are equal if \( G_z = H_z \). When does this happen?

**Special case:** squared error loss, with a Euclidean metric on the outputs. Then \( H_z = G_z = I \), and \( G = \mathbb{E}_x [ J_{zw}^\top J_{zw} ] \) is the Gauss-Newton matrix
Bregman Divergence

- Let $\phi$ be a strictly convex function of $z$
- Bregman divergence:

$$D_\phi(z, z') = \phi(z) - \phi(z') - \nabla \phi(z')^\top (z - z')$$

**Examples:**
- (squared) Euclidean distance:
  $$\phi(z) = \frac{1}{2} \|z\|^2 \implies D_\phi(z, z') = \frac{1}{2} \|z - z'\|^2$$
- $z$ are natural parameters of an exponential family, $\phi$ is the log partition function $\implies D_\phi$ is KL divergence
Bregman Divergence

- The Hessian of a Bregman divergence $D_\phi$ is just the Hessian of $\phi$:

$$
\nabla_z^2 D_\phi(z, z')|_{z=z'} = \nabla_z^2 \left[ \phi(z) - \phi(z') - \nabla \phi(z')^\top (z - z') \right] |_{z=z'} \\
= \nabla_z^2 [\phi(z)]|_{z=z'} \\
= \nabla^2 \phi(z')
$$

- If the output loss $\mathcal{L}$ is convex, we can choose $\phi = \mathcal{L}$

- Then $G_z = H_z$, and therefore the GN Hessian equals the metric matrix.
Fisher Information Matrix
An important special case of pullback metrics is when the outputs parameterize a probability distribution and $\rho$ is KL divergence.

- Then $G_z = \nabla^2 \rho$ is the Fisher information matrix $F_z$.
- Pullback metric is $\mathbb{E}_x [J_{zw}^T F_z J_{zw}]$ as usual.

While the Fisher metric has many convenient properties, there’s nothing *that* special about it when it comes to neural nets. For most algorithms, you’re free to choose another output space metric.
Fisher Information Matrix

- Let $\mathcal{D}z = \nabla_z \log p(t \mid z)$
- We can simplify the pullback metric:

\[
F_w = \mathbb{E}_{x \sim p_{\text{data}}} \left[ J_{zw}^\top F_z J_{zw} \right] \quad \text{(def'n)}
\]
\[
= \mathbb{E}_{x \sim p_{\text{data}}} \left[ J_{zw}^\top \mathbb{E}_{t \sim r(\cdot \mid x)} [\mathcal{D}z \mathcal{D}z^\top] J_{zw} \right] \quad \text{(def'n)}
\]
\[
= \mathbb{E}_{x \sim p_{\text{data}}} \left[ J_{zw}^\top \mathcal{D}z \mathcal{D}z^\top J_{zw} \right]
\]
\[
= \mathbb{E}_{x \sim p_{\text{data}}} \left[ \mathcal{D}w \mathcal{D}w^\top \right] \quad \text{(Chain Rule)}
\]

- Since the final formula closely resembles that of the Fisher information matrix, we call this the **Fisher information matrix** for the network.
Fisher Information Matrix

\[ G = \mathbb{E}[J_{zw}^T H_z J_{zw}] \]

Gauss-Newton Hessian

equal for squared error loss

equal for Bregman divergence

equal for Euclidean metric

equal for exponential family NLL

equal for KL divergence

Hessian

\[ H = \nabla^2 J(w) \]

linearized network; equal at optimum if outputs minimize loss

equal for maximum likelihood in exponential families

Fisher information matrix

\[ F = \mathbb{E}_{x \sim \mathcal{P}, t \sim r_w} \left[ Dw D w^T \right] \]
Fisher Information Matrix

\[ \mathbf{F} = \mathbb{E}_{x \sim p_{\text{data}}} [\mathcal{D}w \mathcal{D}w^\top] \]

\[ \mathbf{t} \sim r(\cdot | x) \]

- **Caution:** don’t confuse the Fisher matrix with the empirical Fisher matrix:

\[ \mathbf{F}_{\text{emp}} = \mathbb{E}_{x, t \sim p_{\text{data}}} [\mathcal{D}w \mathcal{D}w^\top] \]

- The difference is that the true Fisher matrix samples the targets from the model’s predictions, while the empirical Fisher uses the training targets.

- Only the true Fisher is related to the Hessian. Using the empirical Fisher to approximate the Hessian is a common mistake (even in published papers!)
Invariance
Invariance

- Various fields have bookkeeping devices to prevent us from performing nonsensical operations, e.g.
  - dimensional analysis
  - types in programming
- Recall from Lecture 1: gradient descent applied to linear regression is not invariant to affine transformations of the inputs
- What if we try to assign units to a linear regression problem:

\[ y = w_1 x_1 + w_2 x_2 + b \]

- output has unit $\$$
- input has unit $\min$
- input has unit $\text{ft}$

- weight must have unit $\$/\min$
- weight must have unit $\$/\text{ft}$
- bias must have unit $\$$
Invariance

- Trying to attach units to the gradient descent update:

\[ w_1 \leftarrow w_1 - \alpha \frac{dh}{dw_1} \]

So the learning rate must have unit \( \text{min}^2/\text{min} \)

\[ w_2 \leftarrow w_2 - \alpha \frac{dh}{dw_2} \]

So the learning rate must have unit \( \text{ft}^2/\text{ft} \)

- No consistent assignment of dimensions!
Invariance

- Look what happens to the gradient descent update when you change from minutes to seconds:

  \[ w_1 \leftarrow w_1 - 0.02 \]

  \[ \frac{dh}{dw_1} = 2 \]

  \[ w_1 \leftarrow w_1 - 72 \]

  \[ \frac{dh}{d\tilde{w}_1} = 60 \frac{dh}{dw_1} \]

  \[ \tilde{w}_1 = 60 \tilde{w}_1 \]

  \[ \frac{dh}{d\tilde{w}_1} = 120 \]

  \[ \tilde{w}_1 \leftarrow \tilde{w}_1 - 1.2 \]

- The gradient gets 3600 times larger!
Mathematicians have developed lots of abstractions to ensure that things are invariant to the choice of coordinate system.

Vectors vs. covectors

- Two vectors in the tangent space to $M$.
- A vector in the tangent space, and a covector, visualized as a linear functional. When we apply this covector to this vector, it evaluates to 2 (count the contours).
- A Riemmanian metric gives an inner product on the tangent space at every point. This induces a norm.

Important distinction: vectors can be pushed forward, and covectors can be pulled back.
Invariance

- Pulling back a covector:

  parameter space

  Pulling back the differential to parameter space and evaluating it on a vector...

  ...is equivalent to...

  ... pushing forward the vector to output space and then evaluating the differential

  output space
Pulling back a metric:

- Pushing forward two vectors to output space and then evaluating the inner product using the output space metric...

- ...is equivalent to...

- ...pulling back the metric to parameter space and evaluating the inner product.
Invariance

- The thing we call the “gradient” is really a covector called the differential. We know it’s a covector because you can pull it back (i.e. backprop).
- In order to update the parameters, you need a vector.
- Using a Riemannian metric, you can convert between covectors and vectors (this is the musical isomorphism)
- If the metric is defined in a coordinate-independent way (e.g. pullback from output space), then it is natural.
- When computed in a coordinate system, this coincides with our formula for the natural gradient:

\[ \tilde{\nabla}J(w) = G^{-1} \nabla J(w) \]
Invariance

• To the first order, the natural gradient update is invariant to differentiable reparameterizations of the model.
  • I.e., the natural gradient updates in two coordinate systems will be equivalent (to first order) in terms of the functions represented by the network
  • If the coordinate systems are related by an affine transformation, then the invariance is exact

• In our field, “natural gradient” is often assumed to mean the Fisher metric, but there are lots of other natural metrics (and Amari considered other examples in his paper)
  • E.g., the pullback of a Euclidean metric on outputs is independent of how the network is parameterized (except for the output layer)