# CSC 2541: Neural Net Training Dynamics Lecture 3 - Metrics

Roger Grosse

University of Toronto, Winter 2021

# Today

- 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 ouptuts 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.

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

• 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.



- 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

• Suppose we want to minimize  $\mathcal{J}(\mathbf{w})$ . The proximal point method trades off the cost and the distance from the current iterate:

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

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

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



• Note: this is an idealized update rule, not something we can efficiently implement. E.g., if  $\lambda = 0$ , it simply optimizes  $\mathcal{J}$  directly.

NNTD (UofT)

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

• Setting the gradient equal to **0** and rearranging:

$$\operatorname{prox}_{\mathcal{J},\lambda}(\mathbf{w}^{(k)}) = \mathbf{u}_{\star} = \mathbf{w}^{(k)} - \lambda^{-1} \nabla \mathcal{J}(\mathbf{u}_{\star})$$

- This is implicit gradient descent: it's almost a gradient descent update, except that the gradient is evaluated at  $\mathbf{u}_{\star}$ .
  - Not a practical algorithm, but often convenient to analyze.

• Approximation 1: linearize the cost around the current weights

$$\begin{aligned} \operatorname{prox}_{\mathcal{J},\lambda}(\mathbf{w}^{(k)}) &= \arg\min_{\mathbf{u}} \left[ \mathcal{J}(\mathbf{w}^{(k)}) + \nabla \mathcal{J}(\mathbf{w}^{(k)})^{\top}(\mathbf{u} - \mathbf{w}^{(k)}) + \lambda \rho(\mathbf{u}, \mathbf{w}^{(k)}) \right] \\ &= \arg\min_{\mathbf{u}} \left[ \nabla \mathcal{J}(\mathbf{w}^{(k)})^{\top} \mathbf{u} + \lambda \rho(\mathbf{u}, \mathbf{w}^{(k)}) \right] \end{aligned}$$

• Setting the gradient to 0:

$$\nabla_{\mathbf{u}}\rho(\mathbf{u}_{\star},\mathbf{w}^{(k)}) = -\lambda^{-1}\nabla\mathcal{J}(\mathbf{w}^{(k)}).$$

• 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  $\mathcal{J}$  and second-order Taylor approximation to  $\rho$ :

$$\begin{split} \rho(\mathbf{u}, \mathbf{w}^{(k)}) &= \frac{1}{2} (\mathbf{u} - \mathbf{w}^{(k)})^\top \mathbf{G}(\mathbf{u} - \mathbf{w}^{(k)}) + \mathcal{O}(\|\mathbf{u} - \mathbf{w}^{(k)}\|^3) \\ \mathbf{G} &= \nabla_{\mathbf{u}}^2 \rho(\mathbf{u}, \mathbf{w}^{(k)}) \big|_{\mathbf{u} = \mathbf{w}^{(k)}} \end{split}$$

• G is the metric matrix

• Plugging in both approximations:

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

• Optimal solution:

$$\mathbf{u}_{\star} = \mathbf{w}^{(k)} - \lambda^{-1} \mathbf{G}^{-1} \nabla \mathcal{J}(\mathbf{w}^{(k)})$$

- For certain  $\rho$ ,  $\mathbf{G}^{-1} \nabla \mathcal{J}(\mathbf{w}^{(k)})$  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:

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

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

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

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

• 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:

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

- 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  $\theta$

NNTD (UofT)

• The metric matrix is the Fisher information matrix (derivation in the readings):

$$\begin{aligned} \nabla_{\mathbf{u}}^{2} \mathrm{D}_{\mathrm{KL}}(p_{\mathbf{u}} \| p_{\boldsymbol{\theta}}) \Big|_{\mathbf{u}=\boldsymbol{\theta}} &= \mathbf{F}_{\boldsymbol{\theta}} \\ &= \mathrm{Cov}_{\mathbf{x} \sim p_{\boldsymbol{\theta}}}(\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x})) \\ &= \mathbb{E}_{\mathbf{x} \sim p_{\boldsymbol{\theta}}} \left[ (\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x})) (\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}))^{\top} \right] \end{aligned}$$

- The vectors  $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x})$  are called Fisher scores
- Intuitively,  $\Delta \theta^{\top} \mathbf{F}_{\theta} \Delta \theta$  tells you how much the distribution changes if you adjust the parameters by  $\Delta \theta$

Fisher metric for univariate Gaussians:



• Proximal operator:

$$\operatorname{prox}_{\mathcal{J},\lambda}(\boldsymbol{\theta}) = \operatorname*{arg\,min}_{\mathbf{u}} \left[ \mathcal{J}(\mathbf{u}) + \lambda \mathrm{D}_{\mathrm{KL}}(p_{\mathbf{u}} \| p_{\boldsymbol{\theta}}) \right]$$

• Infinitesimal limit:

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \alpha \mathbf{F}_{\boldsymbol{\theta}}^{-1} \nabla \mathcal{J}(\boldsymbol{\theta}^{(k)})$$
$$= \boldsymbol{\theta}^{(k)} - \alpha \tilde{\nabla} \mathcal{J}(\boldsymbol{\theta}^{(k)}),$$

where  $\alpha = \lambda^{-1}$ 

- The vector  $\tilde{\nabla} \mathcal{J}(\boldsymbol{\theta}) = \mathbf{F}_{\boldsymbol{\theta}}^{-1} \nabla \mathcal{J}(\boldsymbol{\theta})$  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)

- 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:

$$f^*g(\mathbf{x}_1,\ldots,\mathbf{x}_K) = g(f(\mathbf{x}_1),\ldots,f(\mathbf{x}_K))$$

- 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

$$\begin{aligned} \mathbf{G}_{\mathbf{x}} &= \nabla_{\mathbf{x}}^{2} f^{*} \rho(\mathbf{x}, \mathbf{x}_{0}) \big|_{\mathbf{x} = \mathbf{x}_{0}} \\ &= \mathbf{J}_{\mathbf{z}\mathbf{x}}^{\top} \left[ \nabla_{\mathbf{z}}^{2} \rho(\mathbf{z}, \mathbf{z}_{0}) \big|_{\mathbf{z} = \mathbf{z}_{0}} \right] \mathbf{J}_{\mathbf{z}\mathbf{x}} + \underbrace{\sum_{a} \frac{\partial \rho}{\partial z_{a}} \nabla_{\mathbf{x}}^{2} [f(\mathbf{x})]_{a}}_{=\mathbf{0}} \\ &= \mathbf{J}_{\mathbf{z}\mathbf{x}}^{\top} \mathbf{G}_{\mathbf{z}} \mathbf{J}_{\mathbf{z}\mathbf{x}} \end{aligned}$$

• 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.

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

•  $\mathbf{G}_{\mathbf{z}} = \mathbf{I}$ , so

$$\mathbf{G}_{\mathbf{x}} = \mathbf{J}_{\mathbf{z}\mathbf{x}}^\top \mathbf{G}_{\mathbf{z}} \mathbf{J}_{\mathbf{z}\mathbf{x}} = \mathbf{J}_{\mathbf{z}\mathbf{x}}^\top \mathbf{J}_{\mathbf{z}\mathbf{x}}$$

• Approximating the proximal point update with this metric:



• 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:



• Second-order Taylor approximation:

$$\begin{aligned} \mathbf{G}_{\mathbf{w}} &= \nabla_{\mathbf{w}}^{2} \rho_{\text{pull}}(\mathbf{w}, \mathbf{w}') \big|_{\mathbf{w} = \mathbf{w}'} \\ &= \mathbb{E}_{\mathbf{x}} \left[ \nabla_{\mathbf{w}}^{2} \rho(f(\mathbf{w}, \mathbf{x}), f(\mathbf{w}', \mathbf{x})) \right] \\ &= \mathbb{E}_{\mathbf{x}} [\mathbf{J}_{\mathbf{zw}}^{\top} \mathbf{G}_{\mathbf{z}} \mathbf{J}_{\mathbf{zw}}]. \end{aligned}$$

- 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:

• Compare with Gauss-Newton Hessian-vector products:

### Connection to Gauss-Newton Hessian

#### • Recall:

- Gauss-Newton Hessian:  $\mathbf{G} = \mathbb{E}_{\mathbf{x}}[\mathbf{J}_{\mathbf{zw}}^{\top}\mathbf{H}_{\mathbf{z}}\mathbf{J}_{\mathbf{zw}}]$
- Pullback metric:  $\mathbf{G} = \mathbb{E}_{\mathbf{x}}[\mathbf{J}_{\mathbf{zw}}^{\top}\mathbf{G}_{\mathbf{z}}\mathbf{J}_{\mathbf{zw}}]$
- These are equal if  $\mathbf{G}_{\mathbf{z}} = \mathbf{H}_{\mathbf{z}}$ . When does this happen?
- Special case: squared error loss, with a Euclidean metric on the outputs. Then  $\mathbf{H}_{\mathbf{z}} = \mathbf{G}_{\mathbf{z}} = \mathbf{I}$ , and  $\mathbf{G} = \mathbb{E}_{\mathbf{x}}[\mathbf{J}_{\mathbf{zw}}^{\top}\mathbf{J}_{\mathbf{zw}}]$  is the Gauss-Newton matrix

## Bregman Divergence

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

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



#### • Examples:

• (squared) Euclidean distance:

$$\phi(\mathbf{z}) = \frac{1}{2} \|\mathbf{z}\|^2 \implies D_{\phi}(\mathbf{z}, \mathbf{z}') = \frac{1}{2} \|\mathbf{z} - \mathbf{z}'\|^2$$

• **z** are natural parameters of an exponential family,  $\phi$  is the log partition function  $\Rightarrow D_{\phi}$  is KL divergence

NNTD (UofT)

## Bregman Divergence

• The Hessian of a Bregman divergence  $D_{\phi}$  is just the Hessian of  $\phi$ :

$$\begin{aligned} \nabla_{\mathbf{z}}^{2} \mathbf{D}_{\phi}(\mathbf{z}, \mathbf{z}') \Big|_{\mathbf{z}=\mathbf{z}'} &= \nabla_{\mathbf{z}}^{2} \left[ \phi(\mathbf{z}) - \phi(\mathbf{z}') - \nabla \phi(\mathbf{z}')^{\top} (\mathbf{z} - \mathbf{z}') \right] \Big|_{\mathbf{z}=\mathbf{z}'} \\ &= \nabla_{\mathbf{z}}^{2} [\phi(\mathbf{z})] \Big|_{\mathbf{z}=\mathbf{z}'} \\ &= \nabla^{2} \phi(\mathbf{z}') \end{aligned}$$

- If the output loss  $\mathcal{L}$  is convex, we can choose  $\phi = \mathcal{L}$
- Then  $\mathbf{G}_{\mathbf{z}} = \mathbf{H}_{\mathbf{z}}$ , and therefore the GN Hessian equals the metric matrix.

- An important special case of pullback metrics is when the outputs parameterize a probability distribution and  $\rho$  is KL divergence
  - Then  $\mathbf{G}_{\mathbf{z}} = \nabla^2 \rho$  is the Fisher information matrix  $\mathbf{F}_{\mathbf{z}}$
  - Pullback metric is  $\mathbb{E}_{\mathbf{x}}[\mathbf{J}_{\mathbf{zw}}^{\top}\mathbf{F}_{\mathbf{z}}\mathbf{J}_{\mathbf{zw}}]$  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.

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

$$\begin{split} \mathbf{F}_{\mathbf{w}} &= \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[ \mathbf{J}_{\mathbf{zw}}^{\top} \mathbf{F}_{\mathbf{z}} \mathbf{J}_{\mathbf{zw}} \right] \qquad (\text{def'n}) \\ &= \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[ \mathbf{J}_{\mathbf{zw}}^{\top} \mathbb{E}_{\mathbf{t} \sim r(\cdot \mid \mathbf{x})} [\mathcal{D} \mathbf{z} \mathcal{D} \mathbf{z}^{\top}] \mathbf{J}_{\mathbf{zw}} \right] \qquad (\text{def'n}) \\ &= \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[ \mathbf{J}_{\mathbf{zw}}^{\top} \mathcal{D} \mathbf{z} \mathcal{D} \mathbf{z}^{\top} \mathbf{J}_{\mathbf{zw}} \right] \\ &= \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[ \mathbf{J}_{\mathbf{zw}}^{\top} \mathcal{D} \mathbf{z} \mathcal{D} \mathbf{z}^{\top} \mathbf{J}_{\mathbf{zw}} \right] \qquad (\text{Chain Rule}) \end{split}$$

• Since the final formula closely resembles that of the Fisher information matrix, we call this the Fisher information matrix for the network.



$$\mathbf{F} = \mathbb{E}_{\substack{\mathbf{x} \sim p_{ ext{data}} \ \mathbf{t} \sim r(\cdot \mid \mathbf{x})}} [\mathcal{D} \mathbf{w} \mathcal{D} \mathbf{w}^{ op}]$$

• **Caution:** don't confuse the Fisher matrix with the empirical Fisher matrix:

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

- 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!)

- 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:



• Trying to attach units to the gradient descent update:



• No consistent assignment of dimensions!

NNTD (UofT)

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

$$\begin{array}{ccc} \text{GD with} \\ \text{LR = 0.01} \\ \text{in minutes} \end{array} & w_1 \leftarrow w_1 - 0.02 \end{array} & \overbrace{\scriptstyle \text{LR = 0.01}}^{\text{GD with}} \frac{\mathrm{d}h}{\mathrm{d}w_1} = 2 \qquad w_1 \leftarrow w_1 - 72 \\ \hline \\ \hline \\ \frac{\mathrm{d}h}{\mathrm{d}\tilde{w}_1} = 60 \frac{\mathrm{d}h}{\mathrm{d}w_1} - \overbrace{\scriptstyle \text{loc}}^{\text{d}h} \frac{\mathrm{d}h}{\mathrm{d}w_1} = 60 \tilde{w}_1 - \overbrace{\scriptstyle \text{loc}}^{\text{d}h} \frac{\mathrm{d}h}{\mathrm{d}\tilde{w}_1} = 120 \\ \hline \\ \frac{\mathrm{d}h}{\mathrm{d}\tilde{w}_1} = 120 \\ \hline \\ \frac{\mathrm{GD with}}{\mathrm{LR = 0.01}} & \tilde{w}_1 \leftarrow \tilde{w}_1 - 1.2 \end{array}$$

• 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.

NNTD (UofT)

• 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:

#### parameter space



#### 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.

#### output space



- 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:

$$\tilde{\nabla}\mathcal{J}(\mathbf{w}) = \mathbf{G}^{-1}\nabla\mathcal{J}(\mathbf{w})$$

- 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)