Hamiltonian Monte-Carlo for Orthogonal Matrices

Victor Yanush[1] Dmitry Kropotov [2]

[1]Samsung-HSE Laboratory [2]Moscow State University

Motivation

Unitary Recurrent neural network[1] (uRNN):

$$h_t = \sigma(Wh_{t-1} + Vx_t + b)$$
$$y_t = Uh_t + c$$
$$W \in \{X \in \mathbb{C}^{n \times n} \mid X^*X = I_n\} \text{ i.e. unitary matrix}$$

Properties:

- solves vanishing/exploding gradients
- beats LSTM on sequence copy and pixel-by-pixel MNIST classification problems
- in theory can process arbitrary length dependencies

We can replace unitarity for orthogonality for simplicity

[1] Wisdom, Scott, et al. "Full-capacity unitary recurrent neural networks."

Proof of norm preservation

By chain rule:

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_T}\frac{\partial h_T}{\partial h_t} = \frac{\partial L}{\partial h_T}\prod_{k=t}^{T-1}\frac{\partial h_{k+1}}{\partial h_k} = \frac{\partial L}{\partial h_T}\prod_{k=t}^{T-1} D_{k+1}W^T,$$

where $D_{k+1} = \mathrm{diag}(\sigma'(z_{k+1}))$.
Gradient norm:

$$\left\|\frac{\partial L}{\partial h_t}\right\| = \left\|\frac{\partial L}{\partial h_T}\prod_{k=t}^{T-1} D_{k+1}W^T\right\| \leq \left\|\frac{\partial L}{\partial h_T}\right\|\prod_{k=t}^{T-1}\|D_{k+1}W^T\|$$

$$= \left\|\frac{\partial L}{\partial h_T}\right\|\prod_{k=t}^{T-1}\|D_{k+1}\|$$

$$\text{where } \|D_k\| = \max_{j=1,\ldots,n}|\sigma'(z_k^{(j)})|,$$

Proof of norm preservation

By chain rule:

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_T}\frac{\partial h_T}{\partial h_t} = \frac{\partial L}{\partial h_T}\prod_{k=t}^{T-1}\frac{\partial h_{k+1}}{\partial h_k} = \frac{\partial L}{\partial h_T}\prod_{k=t}^{T-1} D_{k+1}W^T,$$

where $D_{k+1} = \mathrm{diag}(\sigma'(z_{k+1}))$.
Gradient norm:

$$\left\|\frac{\partial L}{\partial h_t}\right\| = \left\|\frac{\partial L}{\partial h_T}\prod_{k=t}^{T-1} D_{k+1}W^T\right\| \leq \left\|\frac{\partial L}{\partial h_T}\right\|\prod_{k=t}^{T-1}\|D_{k+1}W^T\|$$

$$= \left\|\frac{\partial L}{\partial h_T}\right\|\prod_{k=t}^{T-1}\|D_{k+1}\| = \left\|\frac{\partial L}{\partial h_T}\right\|$$

where $\|D_k\| = \max_{j=1,\ldots,n}|\sigma'(z_k^{(j)})|,$

Orthogonal matrices in deep learning

Orthogonality — one of the **most useful** constraints:

- Orthogonal matrices in RNNs fix vanishing/exploding gradients
- GANs with orthogonal regularization[2] are more stable
- Additional regularization for usual networks[3]
- Faster learning

Also we can reparameterize other kinds of parameters:

- Low-rank matrices using SVD
- Tensor-train format tensors
- Others

---

[2] Brock, Andrew, et al. "Neural photo editing with introspective adversarial networks."

[3] Huang, Lei, et al. "Orthogonal weight normalization: Solution to optimization over multiple ..."

Motivation

How to apply Bayesian inference to the uRNN?

- Variational inference:
  - Either complicated inference with (semi-)implicit distributions
  - Or very simple distributions with high bias
- Markov Chain Monte-Carlo (MCMC):
  - Slow, but no bias
  - We can construct true Bayesian ensemble with it

We choose MCMC!

Problem statement

Problem: sample from the distribution $\pi(\theta)$ which is known up to the normalizing constant. Usually $\pi(\theta)$ is the posterior:

$$\pi(\theta) = p(\theta \mid X) \sim p(X \mid \theta)p(\theta)$$

Samples can be used for inference:

$$p(x_{\text{test}} \mid X_{\text{train}}) = \int p(x_{\text{test}} \mid \theta)p(\theta \mid X_{\text{train}})d\theta \approx \frac{1}{n}\sum_{i=1}^{n} p(x_{\text{test}} \mid \theta_i)$$

where $\theta_i \sim p(\theta \mid X_{\text{train}})$

Markov Chain Monte-Carlo

Markov Chain Monte-Carlo (MCMC) algorithms consist of:

- proposal distribution $q(\theta_{\text{new}} \mid \theta_{\text{old}})$ from which we acquire new samples
- Metropolis-Hastings correction which accepts samples with probability

$$\min\left(1, \frac{\pi(\theta_{\text{new}})q(\theta_{\text{old}} \mid \theta_{\text{new}})}{\pi(\theta_{\text{old}})q(\theta_{\text{new}} \mid \theta_{\text{old}})}\right)$$

Proposal should be efficient enough to reject not too many samples

Hamiltonian Monte-Carlo

In Hamiltonian Monte-Carlo (HMC) we

- augment parameter space with auxiliary variables: $\hat{\theta} = (\theta, r)$
- work with joint distribution $\hat{\pi}(\theta, r) = \pi(\theta)\hat{\pi}(r \mid \theta)$

Joint distribution is usually written in terms of *Hamiltonian*:

$$\hat{\pi}(\theta, r) = \frac{1}{Z} \exp(-H(\theta, r))$$

Most often

$$\pi(r \mid \theta) = \mathcal{N}(r \mid 0, I)$$

$$H(\theta, r) = -\log \pi(\theta) + \frac{1}{2} r^T r$$

Hamiltonian dynamics

We can explore the distribution following the Hamiltonian dynamics:

$$\frac{d\theta}{dt} = \frac{\partial H}{\partial r}$$
$$\frac{dr}{dt} = -\frac{\partial H}{\partial \theta}$$

Energy is conserved in Hamiltonian dynamics:

$$\frac{dH}{dt} = \frac{\partial H}{\partial \theta}\frac{d\theta}{dt} + \frac{\partial H}{\partial r}\frac{dr}{dt} = \frac{\partial H}{\partial \theta}\frac{\partial H}{\partial r} - \frac{\partial H}{\partial r}\frac{\partial H}{\partial \theta} = 0$$

Maintaining the same energy we quickly explore various regions of θ space with high probability because:

$$\hat{\pi}(\theta, r) \propto \exp(-H(\theta, r))$$

Hamiltonian Monte-Carlo

Sample proposal:

- Given θ_{old}, sample $r_{\text{old}} \sim \pi(r_{\text{old}} \mid \theta_{\text{old}})$
- Integrate Hamiltonian equations for some time to get $(\theta_{\text{new}}, r_{\text{new}})$
- Take θ_{new}

Notice that acceptance probability is equal to:

$$\min\left(1, \frac{\exp(-H(\theta_{\text{new}}, r_{\text{new}}))}{\exp(-H(\theta_{\text{old}}, r_{\text{old}}))}\right)$$
$$= \min\left(1, \exp(\underbrace{H(\theta_{\text{old}}, r_{\text{old}}) - H(\theta_{\text{new}}, r_{\text{new}})}_{=0 \text{ as energy is conserved}})\right) = 1$$

We don't even need rejection step, as we sample from correct distribution!

Hamiltonian Monte-Carlo

Sample proposal:

- Given θ_{old}, sample $r_{\text{old}} \sim \pi(r_{\text{old}} \mid \theta_{\text{old}})$
- Integrate Hamiltonian equations for some time to get $(\theta_{\text{new}}, r_{\text{new}})$
- Take θ_{new}

Notice that acceptance probability is equal to:

$$\min\left(1, \frac{\exp(-H(\theta_{\text{new}}, r_{\text{new}}))}{\exp(-H(\theta_{\text{old}}, r_{\text{old}}))}\right)$$
$$= \min\left(1, \exp(\underbrace{H(\theta_{\text{old}}, r_{\text{old}}) - H(\theta_{\text{new}}, r_{\text{new}})}_{=0 \text{ as energy is conserved}})\right) = 1$$

We don't even need rejection step, as we sample from correct distribution!

Almost always we cannot solve Hamiltonian equations *exactly*

Leapfrog integration

The only option we have is to integrate numerically.
Leapfrog integration:

$$
\begin{aligned}
r^{n+1/2} &= r^n + \frac{\varepsilon}{2}\nabla \log \pi(\theta^n) \\
\theta^{n+1} &= \theta^n + \varepsilon r^{n+1/2} \\
r^{n+1} &= r^{n+1/2} + \frac{\varepsilon}{2}\nabla \log \pi(\theta^{n+1})
\end{aligned}
$$

Discretization error $\Rightarrow$ need MH correction step: accept sample with probability

$$
\min\left(1, \exp(H(\theta_{\text{old}}, r_{\text{old}}) - H(\theta_{\text{new}}, r_{\text{new}}))\right)
$$

Integration properties

Good integration scheme should inherit properties of solutions of Hamiltonian equations:

- Reversibility: $\psi(\theta_{\text{new}}, -r_{\text{new}}) = (\theta_{\text{old}}, -r_{\text{old}})$
- Energy conservation $H(\theta_{\text{new}}, r_{\text{new}}) = H(\theta_{\text{old}}, r_{\text{old}})$

It's easy to see that Leapfrog is reversible.
What about energy conservation?

Symplecticity

Is Leapfrog a good choice for integration?

1. We want $H(\theta_{\text{old}}, r_{\text{old}}) \approx H(\theta_{\text{new}}, r_{\text{new}})$ to make rejection rate not too high
2. Sufficient condition for approximate energy conservation[4] is *symplecticity*

Symplecticity by definition means that for

$$J = \frac{\partial(\theta_{\text{new}}, r_{\text{new}})}{\partial(\theta_{\text{old}}, r_{\text{old}})}, \; A = \begin{bmatrix} 0 & -I \\ I & 0 \end{bmatrix}$$

we have

$$J^T A J = A$$

That condition holds for Leapfrog also.

[4] Hairer, Ernst. "Long-time energy conservation of numerical integrators."

Why HMC fails?

Leapfrog integration:

$$
\begin{aligned}
r^{n+1/2} &= r^n + \frac{\varepsilon}{2}\nabla \log \pi(\theta^n) \\
\theta^{n+1} &= \theta^n + \varepsilon r^{n+1/2} \\
r^{n+1} &= r^{n+1/2} + \frac{\varepsilon}{2}\nabla \log \pi(\theta^{n+1})
\end{aligned}
$$

Assume that we have constraints on θ (i.e. $g(\theta) = 0$).
Which equations fail?

Why HMC fails?

Leapfrog integration:

$$r^{n+1/2} = r^n + \frac{\varepsilon}{2}\nabla \log \pi(\theta^n)$$
$$\theta^{n+1} = \theta^n + \varepsilon r^{n+1/2}$$
$$r^{n+1} = r^{n+1/2} + \frac{\varepsilon}{2}\nabla \log \pi(\theta^{n+1})$$

Assume that we have constraints on θ (i.e. $g(\theta) = 0$).
Which equations fail?

We need the concept of manifold to fix them.

About manifolds

Manifold — a set which looks like Euclidean space around every point.

For example, ellipse:

- looks like straight line near every point
- has two parameterizations:
  1. $t \in [0, 2\pi)$ — this is an example of intrinsic coordinates
  2. $(x, y) \in \mathbb{R}^2$ with constraints

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

This is an example of extrinsic coordinates

Map between two parameterizations:

$$t \mapsto (a \cos t, b \sin t)$$

About manifolds

Every manifold can be parameterized in two ways:

1. Using *intrinsic* coordinates: $t \in \mathbb{R}^m$ where inner product is defined as

$$\langle u, v \rangle_t = u^T G(t) v.$$

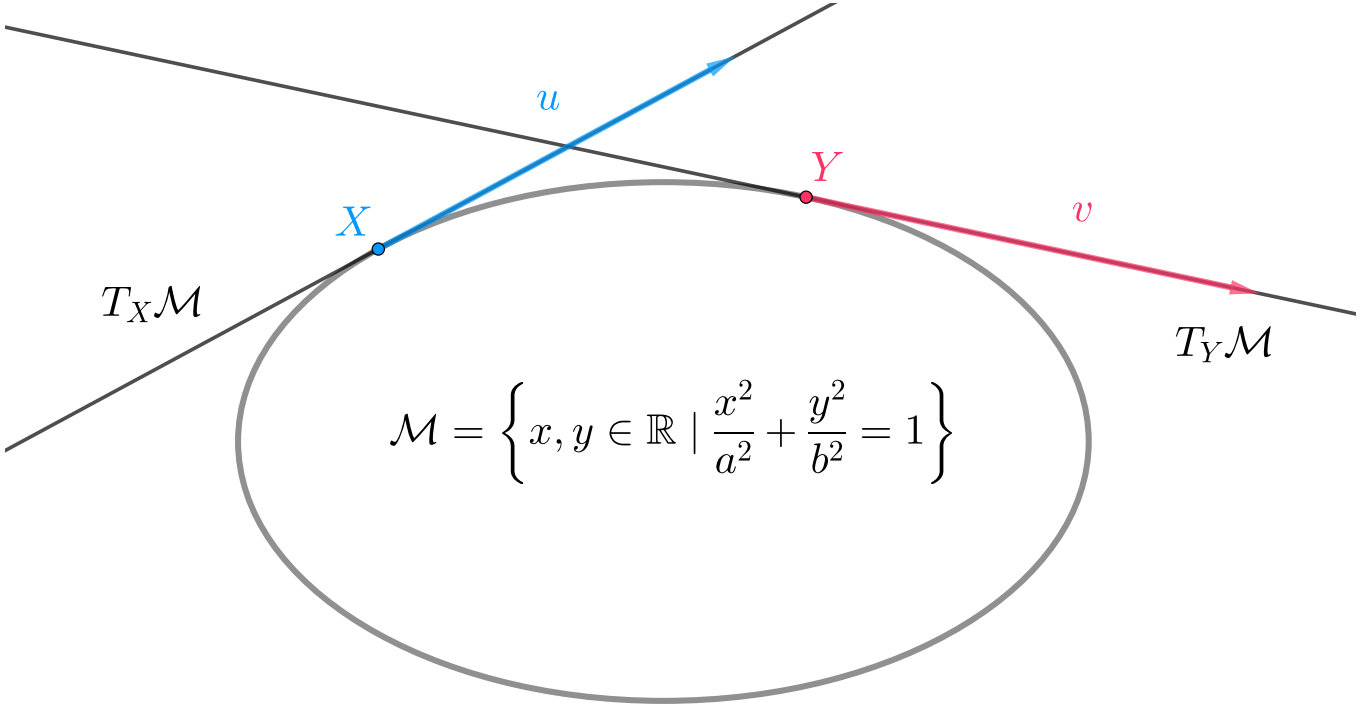
   Here we can use conventional HMC.

2. Using *extrinsic* coordinates: $x = \phi(t) \in \mathcal{M} \subset \mathbb{R}^k$ with some constraints $g(x) = 0$.
   - Inner product is defined as usual;
   - Manifold $\mathcal{M}$ is said to be *embedded* in $\mathbb{R}^k$

   Here HMC fails because of the constraints.

Tangent spaces

For smooth manifold $\mathcal{M}$ there is a tangent space $T_x\mathcal{M}$ at each point x:



Riemannian gradient

Riemannian gradient — tangent vector along which function increases the most.

Consider $f : \mathcal{M} \to \mathbb{R}$. From Taylor expansion:

$$f(x + dx) = f(x) + \langle \nabla f(x), dx \rangle + O(\|dx\|^2)$$

If we restrict dx to the tangent space we get

$$\langle \nabla f(x), dx \rangle = \langle \nabla f(x)_{||}, dx \rangle + \underbrace{\langle \nabla f(x)_{\perp}, dx \rangle}_{=0} = \langle \nabla f(x)_{||}, dx \rangle$$

Riemannian gradient $\hat{\nabla} f(x)$ can be computed as:

$$\hat{\nabla} f(x) = \nabla f(x)_{||} = \text{proj}_{T_x}(\nabla f(x))$$

Back to HMC

What is the domain of θ and r?

- θ lies in some **manifold** $\mathcal{M}$ induced by the constraints
- r is velocity, so r should lie in the **tangent space** $T_\theta \mathcal{M}$

Leapfrog revisited

Leapfrog integration:

$$r^{n+1/2} = \underbrace{r^n}_{\in T_{\theta^n}} + \underbrace{\frac{\varepsilon}{2}\nabla \log \pi(\theta^n)}_{\notin T_{\theta^n}}$$

$$\theta^{n+1} = \theta^n + \varepsilon r^{n+1/2}$$

$$r^{n+1} = r^{n+1/2} + \frac{\varepsilon}{2}\nabla \log \pi(\theta^{n+1})$$

We can replace $\nabla \log \pi(\theta)$ with **Riemannian gradient** $\hat{\nabla} \log \pi(\theta)$

Leapfrog revisited

Leapfrog integration:

$$r^{n+1/2} = r^n + \frac{\varepsilon}{2}\hat{\nabla}\log\pi(\theta^n)$$

$$\underbrace{\theta^{n+1}}_{\notin\mathbb{O}^{n\times p}} = \underbrace{\theta^n}_{\in\mathbb{O}^{n\times p}} + \underbrace{\varepsilon r^{n+1/2}}_{\in T_{\theta^n}}$$

$$r^{n+1} = r^{n+1/2} + \frac{\varepsilon}{2}\hat{\nabla}\log\pi(\theta^{n+1})$$

We need to move θ^n in the direction of $r^{n+1/2}$ but **along the curve** on the manifold

Leapfrog revisited

Leapfrog integration:

$$
\begin{aligned}
r^{n+1/2} &= r^n + \frac{\varepsilon}{2}\hat{\nabla}\log\pi(\theta^n) \\
\theta^{n+1} &= \theta^n + \varepsilon r^{n+1/2} \\
\underbrace{r^{n+1}}_{\in T_{\theta^{n+1}}} &= \underbrace{r^{n+1/2}}_{\in T_{\theta^n}} + \underbrace{\frac{\varepsilon}{2}\hat{\nabla}\log\pi(\theta^{n+1})}_{\in T_{\theta^{n+1}}}
\end{aligned}
$$

We need to **transport** $r^{n+1/2}$ from T_{θ^n} to $T_{\theta^{n+1}}$

Leapfrog revisited

In the usual HMC:

$$r_0 \sim \mathcal{N}(0, I)$$

Here we should make sure $r_0 \in T_{\theta^0}$.
Solution:

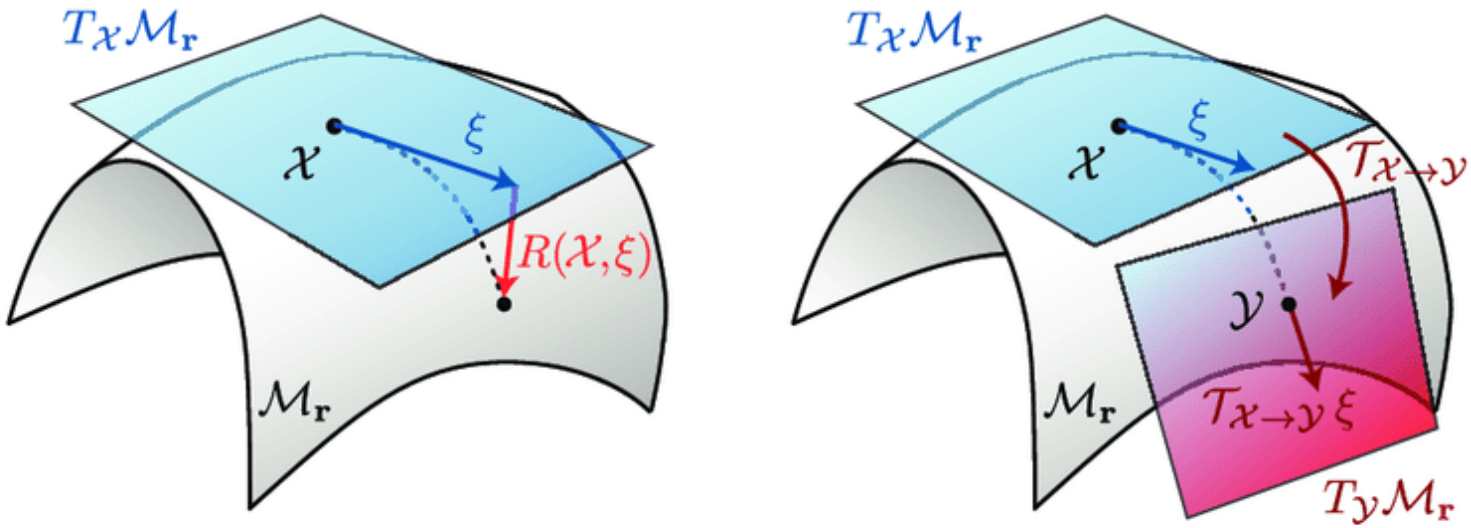
$$\xi \sim \mathcal{N}(0, I)$$
$$r_0 = \text{proj}_{T_{\theta_0}}(\xi)$$

This gives the same result as sampling from standard normal in the tangent space

Retraction and vector transport

Basically we need to be able to do two things:

1. move θ *along* r($\Rightarrow$ **retraction**)
2. move r from $T_{\theta_0}\mathcal{M}$ to $T_{\theta_1}\mathcal{M}$ ($\Rightarrow$ **vector transport**)



(a) Retraction (b) Vector transport

Orthogonal matrices

Useful definitions and properties[5]:

- Manifold of orthogonal matrices:

$$\mathbb{O}^{n\times p} = \{X \in \mathbb{R}^{n\times p} \mid X^T X = I\}$$

- Tangent space at the point X:

$$T_X\mathbb{O}^{n\times p} = \{Z \in \mathbb{R}^{n\times p} \mid Z^T X + X^T Z = 0\}$$

- Projection onto the tangent space:

$$\text{proj}_{T_X}(U) = U - XU^T X$$

[5] Tagare, Hemant D. Notes on optimization on Stiefel manifolds

Cayley transform

For any skew-symmetric matrix $A \in \mathbb{R}^{n \times n}$ we can define *Cayley transform*:

$$Q = \text{Cayley}(A) = (I + A)^{-1} (I - A)$$

We can show that

$$Q \in \mathbb{O}^{n \times n}$$

$$Q^{-1} = \text{Cayley}(-A)$$

Retraction for orthogonal matrices

For $X \in \mathbb{O}^{n\times p}$ We can define curve $Y(\varepsilon)$ in $\mathbb{O}^{n\times p}$ such that:

$$Y(\varepsilon) = \text{Cayley}\left(-\frac{\varepsilon}{2}A\right)X = \left(I - \frac{\varepsilon}{2}A\right)^{-1}\left(I + \frac{\varepsilon}{2}A\right)X$$

It has the following properties:

- $\forall \varepsilon \in \mathbb{R} : Y(\varepsilon) \in \mathbb{O}^{n\times p}$
- $Y(0) = X$
- $\frac{d}{d\varepsilon}Y(\varepsilon)\Big|_{\varepsilon=0} = AX$

If $A = UX^T - XU^T$, then tangent vector at X is

$$U - XU^TX = \text{proj}_{T_X}(U)$$

Retraction and vector transport for orthogonal matrices

For orthogonal matrices we define:

- Retraction (update for θ) is given by Cayley transform:

$$\mathcal{R}(X, U) = \left(I - \frac{\varepsilon}{2}A\right)^{-1}\left(I + \frac{\varepsilon}{2}A\right)X$$

- vector transport of Z from T_X to $T_{\mathcal{R}(X,U)}$:

$$\tau(X, U, Z) = \left(I - \frac{\varepsilon}{2}A\right)^{-1}\left(I + \frac{\varepsilon}{2}A\right)Z$$

where $A = UX^T - XU^T$

oHMC

Final algorithm:

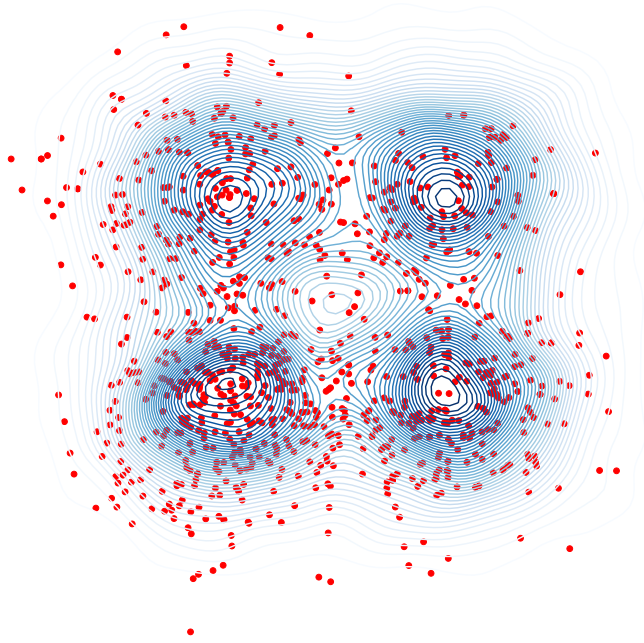
$$
\begin{aligned}
r^{n+1/2} &= r^n + \frac{\varepsilon}{2}\hat{\nabla}\log\pi(\theta^n) \\
\theta^{n+1} &= \mathcal{R}(\theta^n, \varepsilon r^{n+1/2}) \\
r^{n+1} &= \tau(\theta^n, \varepsilon r^{n+1/2}, r^{n+1/2}) + \frac{\varepsilon}{2}\hat{\nabla}\log\pi(\theta^{n+1})
\end{aligned}
$$

A few remarks:

- Time complexity is only $O(np^2)$ per iteration
- This scheme is reversible
- In stochastic case HMC is fixed in the same way

Most importantly: this scheme is symplectic

Toy example



Much better effective sample size than with standard parameterization $W = V\left(V^T V\right)^{-1/2}$:

	min	**median**
HMC	20.3	63.8
oHMC	103.8	139.7

Orthogonal and low-rank networks

Orthogonal network = network with orthogonal weight matrices

- For fully-connected layers weight matrix is orthogonal as is
- For convolutional kernel $W \in \mathbb{R}^{o \times i \times w \times h}$ we require reshaped matrix $\hat{W} \in \mathbb{R}^{o \times iwh}$ to be orthogonal

For low-rank networks we require the same weight matrices to be low-rank and we parameterize them using SVD decomposition:

$$W = U \Sigma V^T,$$
$$U, V \text{ — orthogonal}$$
$$\Sigma = \text{diag}\{\sigma_1, \ldots, \sigma_r\} \geq 0$$

Orthogonal networks

Results:

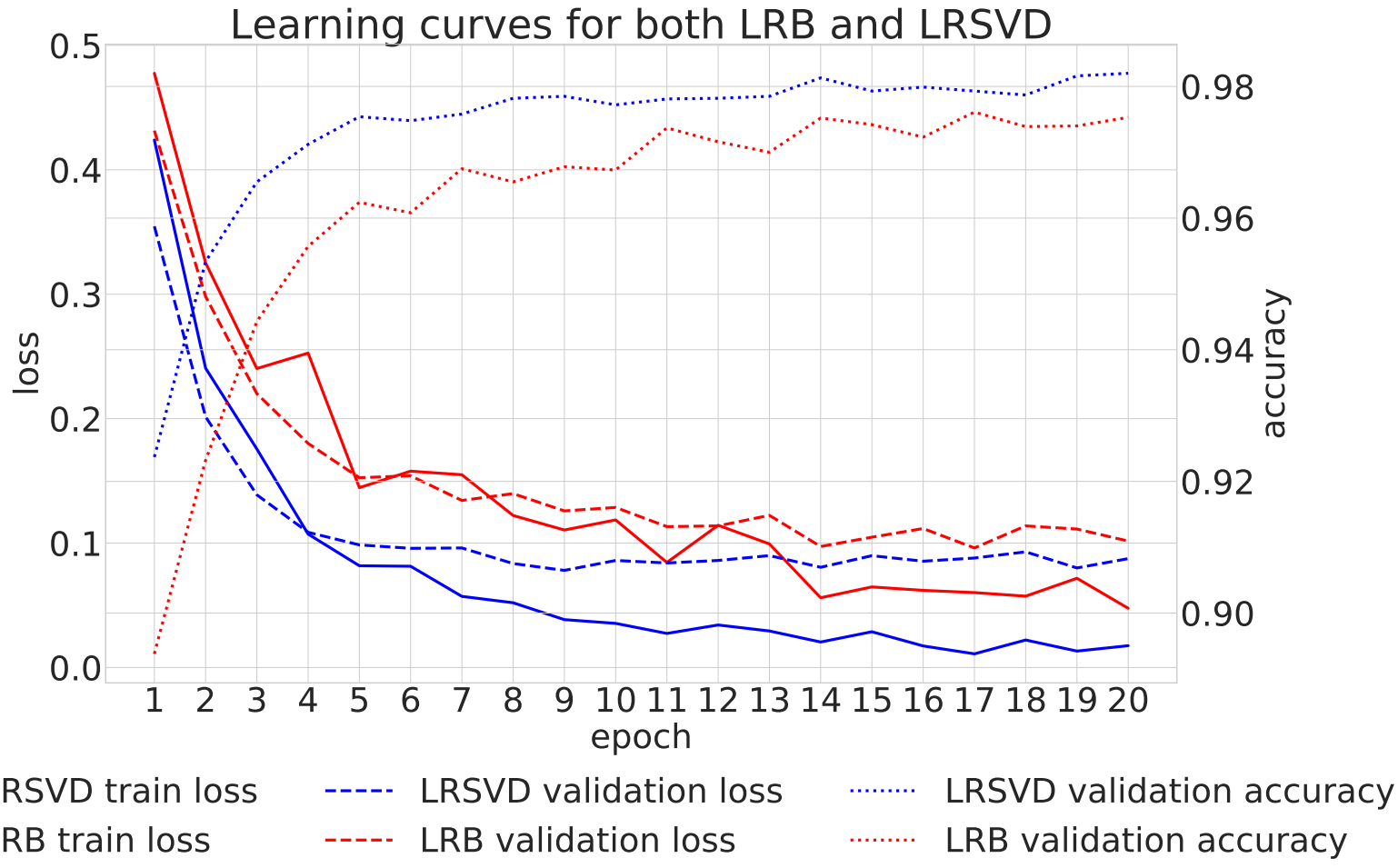
- Learning proceeds faster with orthogonal weights
- Regularization helps with generalization
- Ensembling of neural networks also works

	1 network	**Ensemble**
MNIST	$97.34 \pm 0.08\%$	$97.87 \pm 0.08\%$
CIFAR-10	$92.70 \pm 0.05\%$	$92.85 \pm 0.03\%$

Much smaller orthogonal networks usually can achieve the same accuracy as bigger non-orthogonal!

Low-rank perceptron

Multilayer perceptron on MNIST with low-rank weight matrices in SVD parameterization(blue) vs $W = AB^T$, where $A \in \mathbb{R}^{n \times r}, B \in \mathbb{R}^{m \times r}$ (i.e. bottleneck (red)):



Conclusion

We proposed an MCMC algorithm which has the following properties:

- It works with orthogonal matrices
- Better sample efficiency than for standard orthogonal (ambiguous) reparameterization $W = V\left(V^T V\right)^{-1/2}$
- Applicable not only to neural networks, but also for any probabilistic models
- Can be applied reparameterize other kinds of variables
- Possibly, it can be generalized to other manifolds