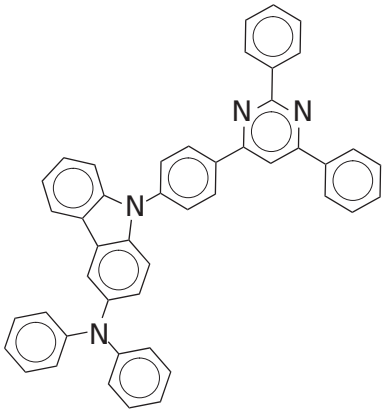
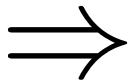
The Chain Rule Unchained: Modeling, Optimization and Inference with Autograd

Dougal Maclaurin

How do you find an organic LED emitter?



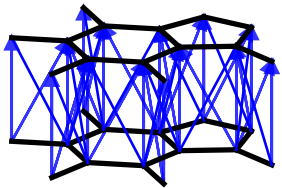


1. Gómez-Bombarelli, *et al.*, Design of Efficient Molecular Organic Light-Emitting Diodes ... Nature Materials, 2016
2. Dahl, *et al.*, Multi-Task Neural Networks for QSAR Predictions, arXiv, 2014
3. Unterthiner *et al.*, Deep Learning as an Opportunity in Virtual Screening, NIPS, 2014.

Molecular fingerprints: bag of substructures

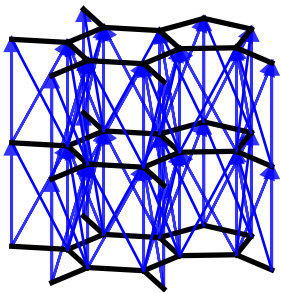
1. Morgan, The Generation of a Unique Machine Description for Chemical Structure. Journal of Chemical Documentation, 1965.
2. Glem, *et al.*, Circular Fingerprints: Flexible Molecular Descriptors with Applications from Physical Chemistry to ADME, IDrugs, 2006
3. Rogers and Hahn, Extended-Connectivity Fingerprints, Journal of Chemical Information and Modeling, 2010

Molecular fingerprints: bag of substructures



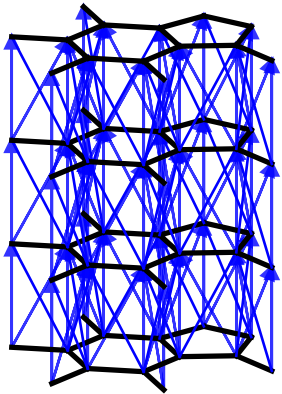
1. Morgan, The Generation of a Unique Machine Description for Chemical Structure. Journal of Chemical Documentation, 1965.
2. Glem, *et al.*, Circular Fingerprints: Flexible Molecular Descriptors with Applications from Physical Chemistry to ADME, IDrugs, 2006
3. Rogers and Hahn, Extended-Connectivity Fingerprints, Journal of Chemical Information and Modeling, 2010

Molecular fingerprints: bag of substructures



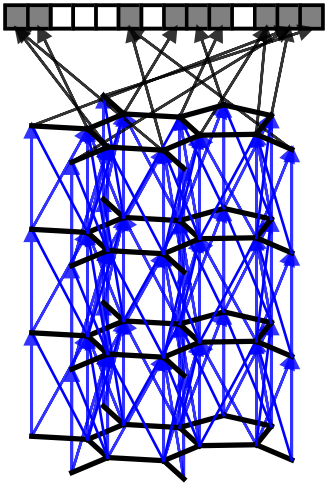
1. Morgan, The Generation of a Unique Machine Description for Chemical Structure. Journal of Chemical Documentation, 1965.
2. Glem, *et al.*, Circular Fingerprints: Flexible Molecular Descriptors with Applications from Physical Chemistry to ADME, IDrugs, 2006
3. Rogers and Hahn, Extended-Connectivity Fingerprints, Journal of Chemical Information and Modeling, 2010

Molecular fingerprints: bag of substructures



1. Morgan, The Generation of a Unique Machine Description for Chemical Structure. Journal of Chemical Documentation, 1965.
2. Glem, *et al.*, Circular Fingerprints: Flexible Molecular Descriptors with Applications from Physical Chemistry to ADME, IDrugs, 2006
3. Rogers and Hahn, Extended-Connectivity Fingerprints, Journal of Chemical Information and Modeling, 2010

Molecular fingerprints: bag of substructures



1. Morgan, The Generation of a Unique Machine Description for Chemical Structure. Journal of Chemical Documentation, 1965.
2. Glem, *et al.*, Circular Fingerprints: Flexible Molecular Descriptors with Applications from Physical Chemistry to ADME, IDrugs, 2006
3. Rogers and Hahn, Extended-Connectivity Fingerprints, Journal of Chemical Information and Modeling, 2010

Differentiable, parameterized fingerprints

Circular fingerprints

Input: molecule, radius R, fingerprint length S
Initialize: fingerprint vector $\mathbf{f} \leftarrow \mathbf{0}_S$
for each atom a in molecule **do**
    $\mathbf{r}_a \leftarrow g(a)$ ▷ look up atom features
for $L = 1$ to R **do** ▷ for each layer
    for each atom a in molecule **do**
        $\mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$
        $\mathbf{v} \leftarrow [\mathbf{r}_a, \mathbf{r}_1, \dots, \mathbf{r}_N]$ ▷ concatenate
        $\mathbf{r}_a \leftarrow \text{hash}(\mathbf{v})$ ▷ hash function
        $i \leftarrow \text{mod}(r_a, S)$ ▷ convert to index
        $\mathbf{f}_i \leftarrow 1$ ▷ Write 1 at index
Return: binary vector $\mathbf{f}$

Neural graph fingerprints

Input: molecule, radius R, $H_1^1 \dots H_R^5$, $W_1 \dots W_R$
Initialize: fingerprint vector $\mathbf{f} \leftarrow \mathbf{0}_S$
for each atom a in molecule **do**
    $\mathbf{r}_a \leftarrow g(a)$ ▷ look up atom features
for $L = 1$ to R **do** ▷ for each layer
    for each atom a in molecule **do**
        $\mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$
        $\mathbf{v} \leftarrow \mathbf{r}_a + \sum_{i=1}^{N} \mathbf{r}_i$ ▷ sum
        $\mathbf{r}_a \leftarrow \sigma(\mathbf{v} H_L^N)$ ▷ smooth function
        $\mathbf{i} \leftarrow \text{softmax}(\mathbf{r}_a W_L)$ ▷ sparsify
        $\mathbf{f} \leftarrow \mathbf{f} + \mathbf{i}$ ▷ add to fingerprint
Return: real-valued vector $\mathbf{f}$

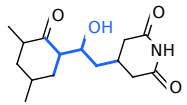
1. Duvenaud*, **Maclaurin*** *et al.*, Convolutional Networks on Graphs for Learning Molecular Fingerprints, NIPS, 2015
2. Li, Tarlow, Brockschmidt, and Zemel, Gated Graph Sequence Neural Networks, ICLR 2016
3. Kearnes *et al.*, Molecular Graph Convolutions: Moving Beyond Fingerprints, Journal of Computer-Aided Molecular Design, 2016
4. Schütt *et al.*, Quantum-Chemical Insights from Deep Tensor Neural Networks, Nature Communications, 2017
5. Gilmer *et al.* Neural Message Passing for Quantum Chemistry, ICML 2017

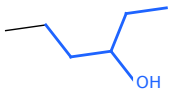
Predictive performance

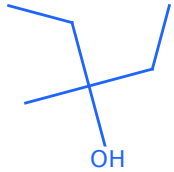
Target	Solubility	Drug efficacy	Photovoltaic efficiency
Units	log Mol/L	EC_{50} in nM	percentage points
Predict mean	4.29	1.47	6.40
Circular FPs + neural net	1.40	1.24	2.04
Neural FPs + neural net	**0.53**	**1.17**	**1.44**

Interpretability

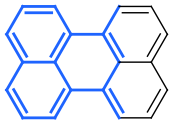
Fragments predictive of **solubility** in water

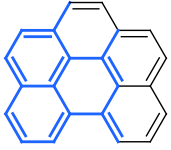


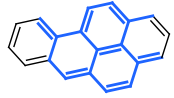




Fragments predictive of **insolubility** in water

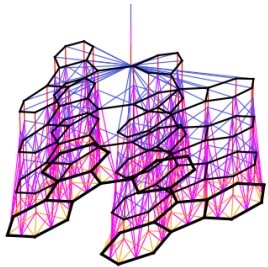






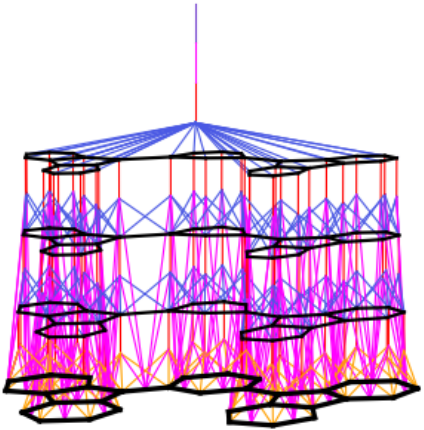
Summary: neural molecular fingerprints

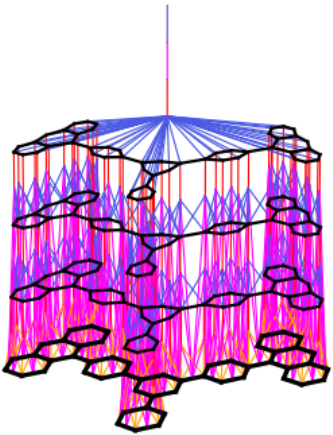
- Differentiable generalization of molecular fingerprints
- Learn a representation by training end-to-end
- Now the standard approach for supervised learning on molecules



D. Duvenaud*, **D. Maclaurin***, *et al. Convolutional Networks on Graphs for Learning Molecular Fingerprints.* NIPS 2015

R. Gómez-Bombarelli, *et al. Design of Efficient Molecular Organic Light-Emitting Diodes by a High-Throughput Virtual Screening and Experimental Approach.* Nature Materials 2016

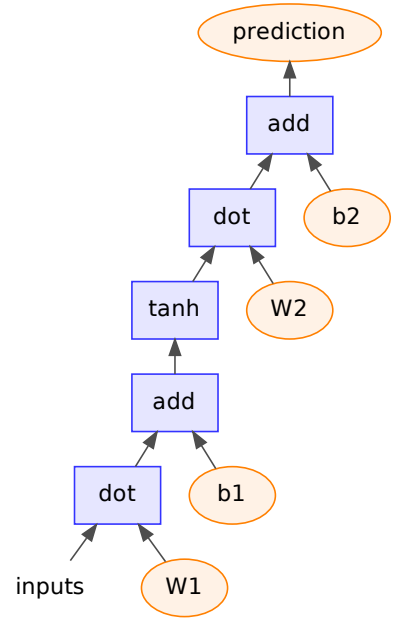


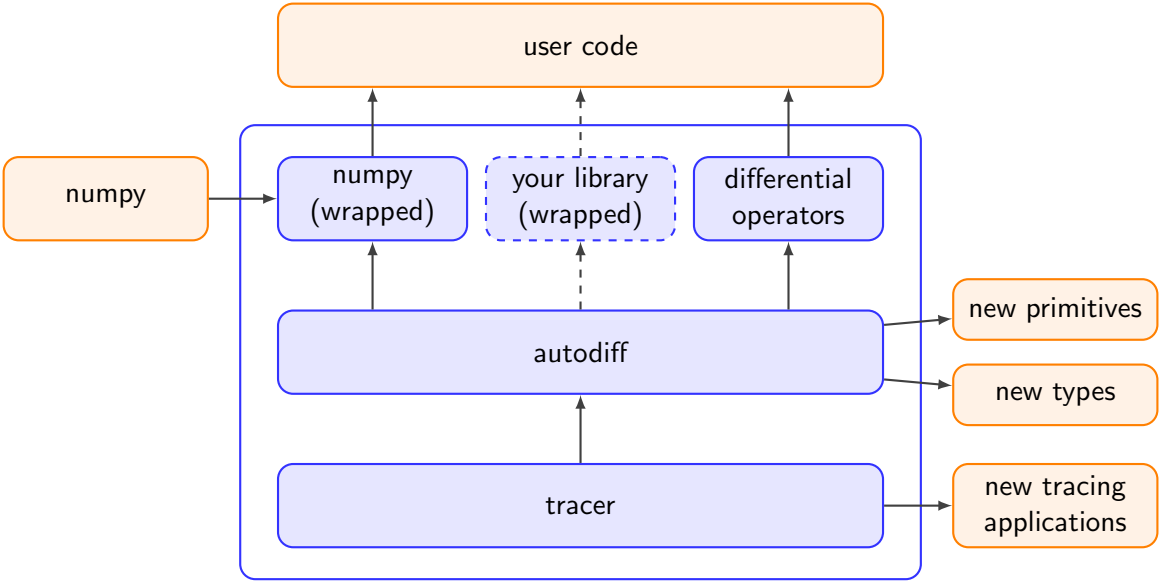


```
import numpy as np


def predict(weights, inputs):
    for W, b in weights:
        z = np.dot(inputs, W) + b
        inputs = np.tanh(z)
    return z

def loglik(weights, inputs, targets):
    preds = predict(weights, inputs)
    return np.sum((preds - targets)**2)
```

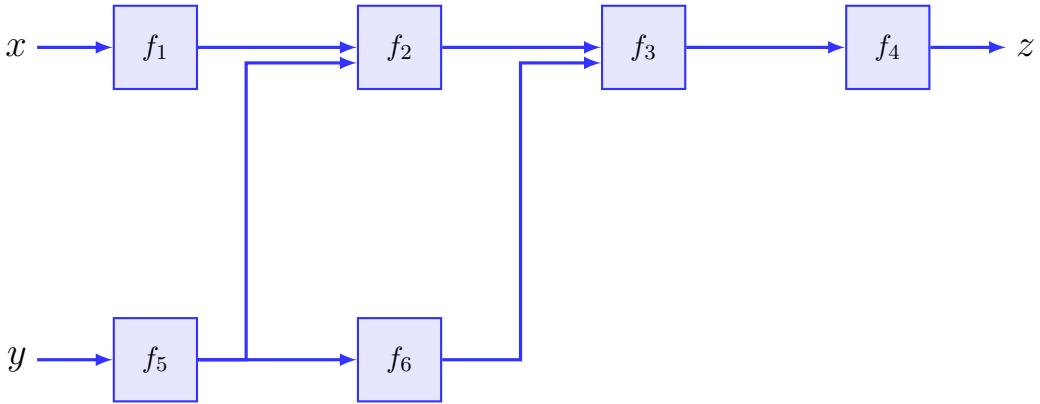
```
import autograd.numpy as np
from autograd import grad

def predict(weights, inputs):
    for W, b in weights:
        z = np.dot(inputs, W) + b
        inputs = np.tanh(z)
    return z

def loglik(weights, inputs, targets):
    preds = predict(weights, inputs)
    return np.sum((preds - targets)**2)

grad_function = grad(loglik)
```

```python
import autograd.numpy as np
from autograd import grad

def predict(weights, inputs):
    for W, b in weights:
        z = np.dot(inputs, W) + b
        inputs = np.tanh(z)
    return z

def loglik(weights, inputs, targets):
    preds = predict(weights, inputs)
    return np.sum((preds - targets)**2)

grad_function = grad(loglik)
```



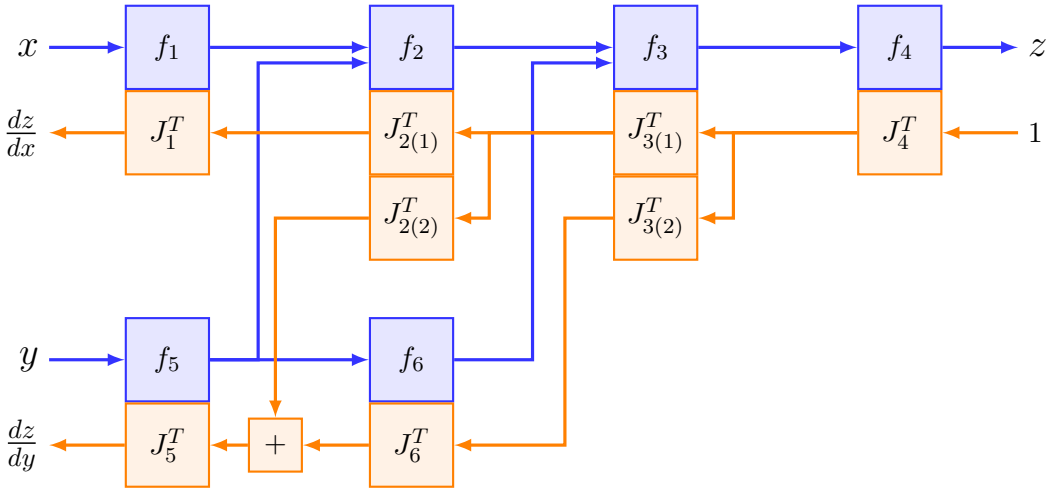

user code
numpy
numpy (wrapped)
your library (wrapped)
differential operators
autodiff
tracer
new primitives
new types
new tracing applications

Reverse-mode differentiation: cheap gradients



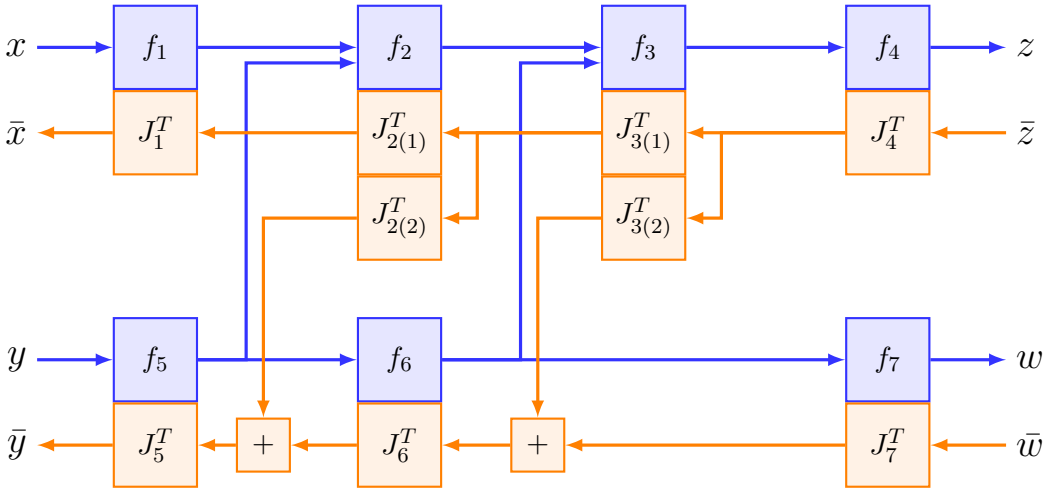
1. Griewank and Walther, Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, Second Edition, SIAM, 2008
2. Pearlmutter and Siskind, Reverse-Mode AD in a Functional Framework: Lambda the Ultimate Backpropagator, TOPLAS 2008

Reverse-mode differentiation: cheap gradients



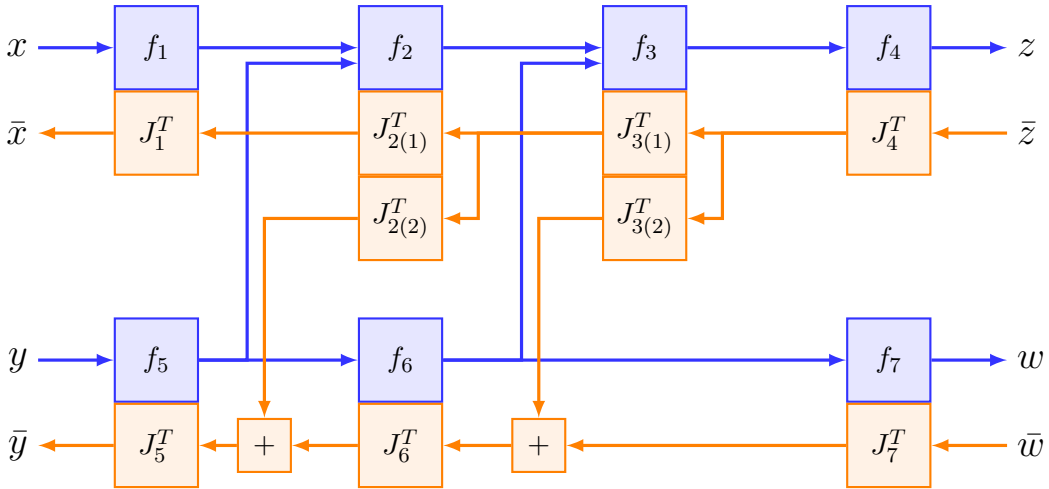
1. Griewank and Walther, Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, Second Edition, SIAM, 2008
2. Pearlmutter and Siskind, Reverse-Mode AD in a Functional Framework: Lambda the Ultimate Backpropagator, TOPLAS 2008

Reverse-mode differentiation: cheap $\mathbf{J^T v}$ products



1. Griewank and Walther, Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, Second Edition, SIAM, 2008
2. Pearlmutter and Siskind, Reverse-Mode AD in a Functional Framework: Lambda the Ultimate Backpropagator, TOPLAS 2008

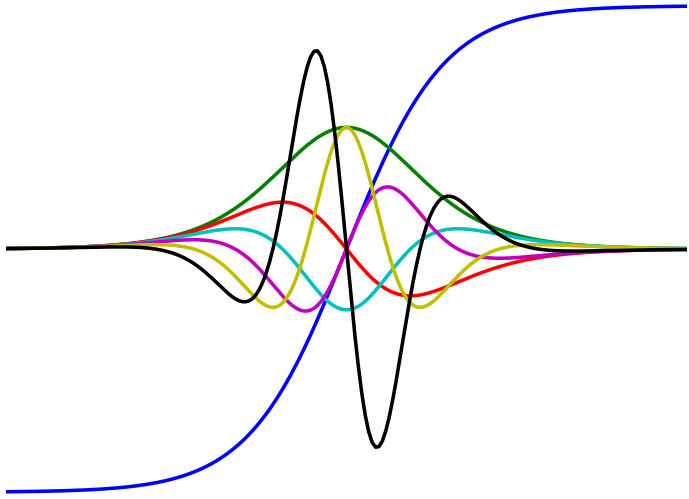
Reverse-mode differentiation: cheap $\mathbf{J^T v}$ products or $\mathbf{v^T J}$ products (vjp)



1. Griewank and Walther, Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, Second Edition, SIAM, 2008
2. Pearlmutter and Siskind, Reverse-Mode AD in a Functional Framework: Lambda the Ultimate Backpropagator, TOPLAS 2008

Higher order autodiff

```
from autograd.numpy import tanh
from autograd import grad

grad(tanh)
grad(grad(tanh))
grad(grad(grad(tanh)))
grad(grad(grad(grad(tanh))))
...
```



Higher order autodiff: efficient differential operators

Hessian-vector product

$$H.v, \quad H_{ij} := \partial_i \partial_j f$$

```
def hvp(f, x):
    return vjp(grad(f), x)
```

Generalized Gauss-Newton matrix vector product

$$G.v, \quad G := J_f^T H_g J_f$$

```
def ggnvp(f, g, x, v):
    f_vjp = vjp(f, x)
    f_jvp = jvp(f, x)
    g_hvp = hvp(g, f(x))
    return f_vjp(g_hvp(f_jvp(v)))
```

N[th] order derivative contraction

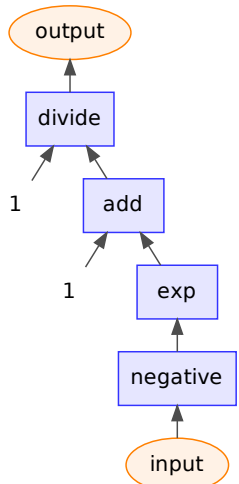
$$\left(\partial_i \partial_j \partial_k \dots f\right) u_i v_j w_k \dots$$

```
def contract(f, x, vs):
   if vs:
       v, vs = vs[0], vs[1:]
       return jvp(lambda z:
                  contract(f, z, vs), x)(v)
   else:
       return grad(f)(x)
```


1. Pearlmutter, Fast Exact Multiplication by the Hessian, Neural Computation, 1994
2. Martens, Deep learning via Hessian-free optimization, ICML 2010
3. Martens and Grosse, Optimizing Neural Networks with Kronecker-factored Approximate Curvature, ICML 2015

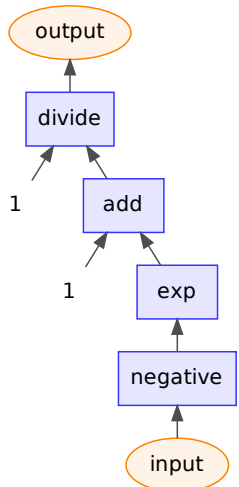
User-defined primitives

```
def logistic(x):
    return 1 / (1 + np.exp(-x))
```

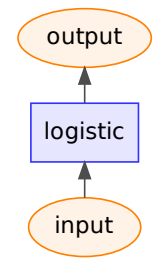


User-defined primitives

```
def logistic(x):
    return 1 / (1 + np.exp(-x))
```




```
@primitive
def logistic(x):
    return 1 / (1 + np.exp(-x))

defvjp(logistic,
       lambda ans, x: lambda g:
       g * ans * (1 - ans))
```



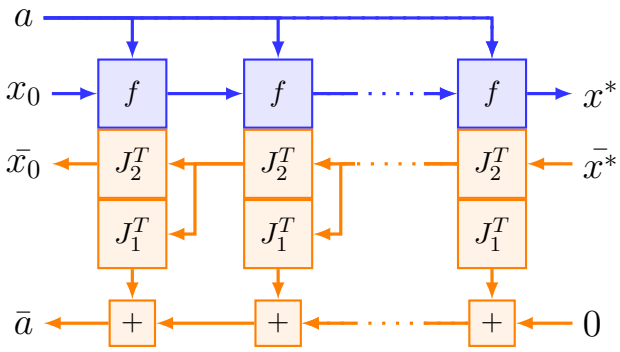
User-defined primitives: fixed point iteration

$$x = f(a, x) \quad \Rightarrow \quad x = x^*(a)$$

User-defined primitives: fixed point iteration

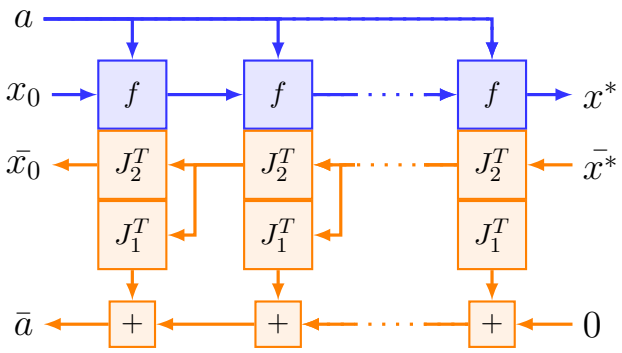
$$x = f(a, x) \quad \Rightarrow \quad x = x^*(a)$$

```
def fixed_point(f, a, x0=0):
    x, x_prev = f(a, x0), x0
    while np.abs(x - x_prev) > tol:
        x, x_prev = f(a, x), x
    return x
```



User-defined primitives: fixed point iteration

$$x = f(a, x) \quad \Rightarrow \quad x = x^*(a)$$



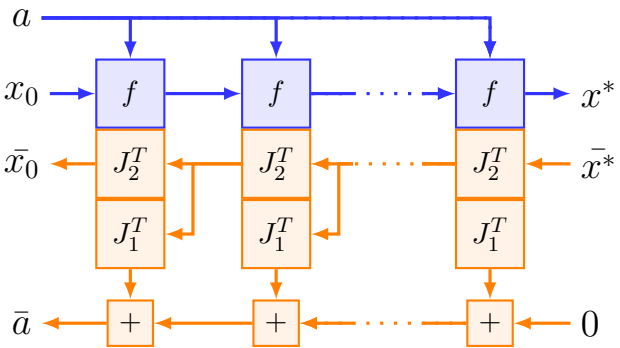

```
@primitive
def fixed_point(f, a, x0=0):
    x, x_prev = f(a, x0), x0
    while np.abs(x - x_prev) > tol:
        x, x_prev = f(a, x), x
    return x

def fixed_point_vjp(ans, f, a, x0):
    vjp_a = vjp(f)(a, ans)
    return (lambda g:
        vjp_a(fixed_point(
            partial(rev_iter, f),
            (a, ans, g))))

def rev_iter(f, a, x):
    a, x_star, x_star_bar = a
    vjp_x = vjp(f, 1)(a, x_star)
    return vjp_x(x) + x_star_bar
```

User-defined primitives: fixed point iteration

$$x = f(a, x) \quad \Rightarrow \quad x = x^*(a)$$

$$J_{x^*}(a) = \left[I - J_{f.2}\left(a, x^*\right)\right]^{-1} J_{f.1}(a, x)$$

$$= \left[\sum_{n=0}^{\infty} \left[J_{f.2}\left(a, x^*\right)\right]^n\right] J_{f.1}(a, x)$$




```
@primitive
def fixed_point(f, a, x0=0):
    x, x_prev = f(a, x0), x0
    while np.abs(x - x_prev) > tol:
        x, x_prev = f(a, x), x
    return x

def fixed_point_vjp(ans, f, a, x0):
    vjp_a = vjp(f)(a, ans)
    return (lambda g:
        vjp_a(fixed_point(
            partial(rev_iter, f),
            (a, ans, g))))

def rev_iter(f, a, x):
    a, x_star, x_star_bar = a
    vjp_x = vjp(f, 1)(a, x_star)
    return vjp_x(x) + x_star_bar
```


1. Johnson *et al.*, Composing Graphical Models with Neural Networks for Structured Representations and Fast Inference, NIPS 2016
2. Amos and Kolter, Optnet: Differentiable Optimization as a Layer in Neural Networks, ICML 2017

User-defined types

User-defined types

$$J(\mathbf{x}, \mathbf{v}) = \lim_{\alpha \to 0} \frac{f(\mathbf{x} + \alpha \mathbf{v}) - f(\mathbf{x})}{\alpha}$$

where

$f : V \to W$

$J : V \times V \to W$

$\alpha \in \mathbb{R}$

$\mathbf{x}, \mathbf{v} \in V$

$$\langle J^T(\mathbf{x}, \mathbf{w}), \mathbf{v} \rangle = \langle \mathbf{w}, J(\mathbf{x}, \mathbf{v}) \rangle$$

where

$J^T : V \times W \to V$

$\mathbf{w} \in W$

User-defined types

$$J(\mathbf{x}, \mathbf{v}) = \lim_{\alpha \to 0} \frac{f(\mathbf{x} + \alpha \mathbf{v}) - f(\mathbf{x})}{\alpha}$$

where

$f : V \to W$

$J : V \times V \to W$

$\alpha \in \mathbb{R}$

$\mathbf{x}, \mathbf{v} \in V$

$$\langle J^T(\mathbf{x}, \mathbf{w}), \mathbf{v} \rangle = \langle \mathbf{w}, J(\mathbf{x}, \mathbf{v}) \rangle$$

where

$J^T : V \times W \to V$

$\mathbf{w} \in W$

V, W are Hilbert spaces over $\mathbb{R}$ supporting:

scalar multiplication	$\alpha \mathbf{v}$
vector addition	$\mathbf{v}_1 + \mathbf{v}_2$
inner product	$\langle \mathbf{v}_1, \mathbf{v}_2 \rangle$

User-defined types: reproducing kernel Hilbert space

H, Hilbert space of functions $f : X \to \mathbb{R}$

scalar multiplication $(\alpha f)(x) = \alpha f(x)$

vector addition $(f_1 + f_2)(x) = f_1(x) + f_2(x)$

reproducing kernel $k : X \times X \to \mathbb{R}$

$\langle k(x, \cdot), f \rangle = f(x)$

map construction $f = \sum_{i=1}^{\infty} \alpha_i k(x_i, \cdot)$

User-defined types: reproducing kernel Hilbert space

H, Hilbert space of functions $f : X \to \mathbb{R}$

scalar multiplication	$(\alpha f)(x) = \alpha f(x)$
vector addition	$(f_1 + f_2)(x) = f_1(x) + f_2(x)$
reproducing kernel	$k : X \times X \to \mathbb{R}$
	$\langle k(x, \cdot), f \rangle = f(x)$

map construction $\quad f = \sum_{i=1}^{\infty} \alpha_i k(x_i, \cdot)$

Derivatives of function evaluation

$$\mathsf{eval}(f, x) = f(x)$$
$$J_{\mathsf{eval}}(f, x, g) = \qquad g \in H$$
$$J^T_{\mathsf{eval}}(f, x, a) = \qquad a \in \mathbb{R}$$

User-defined types: reproducing kernel Hilbert space

H, Hilbert space of functions $f : X \to \mathbb{R}$

scalar multiplication	$(\alpha f)(x) = \alpha f(x)$
vector addition	$(f_1 + f_2)(x) = f_1(x) + f_2(x)$
reproducing kernel	$k : X \times X \to \mathbb{R}$
	$\langle k(x, \cdot), f \rangle = f(x)$

map construction $\quad f = \sum_{i=1}^{\infty} \alpha_i k(x_i, \cdot)$

Derivatives of function evaluation

$$\begin{aligned} \mathsf{eval}(f, x) &= f(x) \\ J_{\mathsf{eval}}(f, x, g) &= g(x) && g \in H \\ J_{\mathsf{eval}}^T(f, x, a) &= && a \in \mathbb{R} \end{aligned}$$

User-defined types: reproducing kernel Hilbert space

H, Hilbert space of functions $f : X \to \mathbb{R}$

scalar multiplication	$(\alpha f)(x) = \alpha f(x)$
vector addition	$(f_1 + f_2)(x) = f_1(x) + f_2(x)$
reproducing kernel	$k : X \times X \to \mathbb{R}$
	$\langle k(x, \cdot), f \rangle = f(x)$

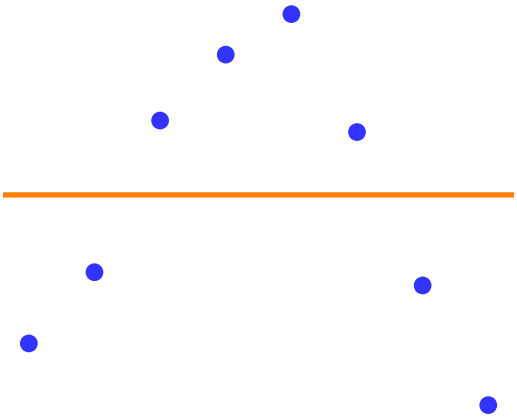
map construction $\quad f = \sum_{i=1}^{\infty} \alpha_i k(x_i, \cdot)$

Derivatives of function evaluation

$$
\begin{aligned}
\mathsf{eval}(f, x) &= f(x) \\
J_{\mathsf{eval}}(f, x, g) &= g(x) && g \in H \\
J^T_{\mathsf{eval}}(f, x, a) &= a\, k(x, \cdot) && a \in \mathbb{R}
\end{aligned}
$$

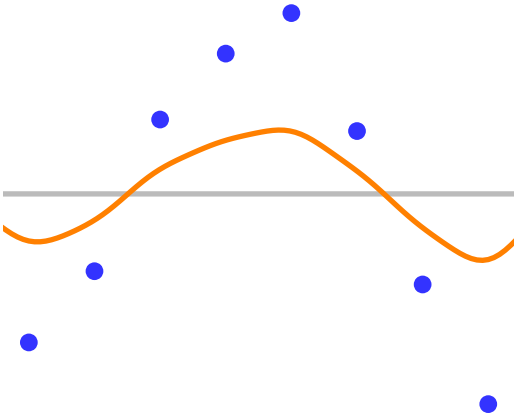
User-defined types: reproducing kernel Hilbert space

```
defvjp(RKHSFun.__call__,
    lambda ans, f, x: lambda g:
    RKHSFun(f.kernel, {x : g}))

def logprob(f, data):
    loglik   = -sum((f(x) - y)**2
                    for x, y in data)
    logprior = -norm_sq(f)
    return loglik + logprior

f = RKHSFun(sq_exp_kernel)
for i in range(30):
    f = f + 0.01 * grad(logprob)(f, data)
```



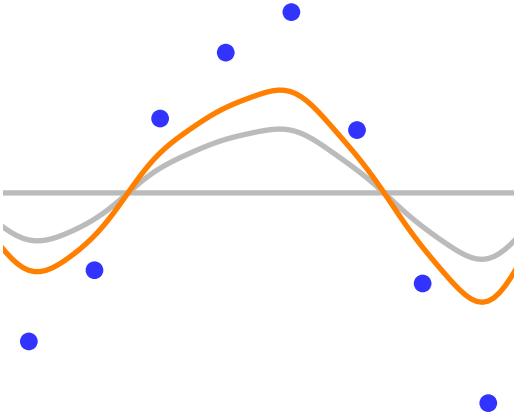
User-defined types: reproducing kernel Hilbert space

```
defvjp(RKHSFun.__call__,
    lambda ans, f, x: lambda g:
    RKHSFun(f.kernel, {x : g}))

def logprob(f, data):
    loglik   = -sum((f(x) - y)**2
                    for x, y in data)
    logprior = -norm_sq(f)
    return loglik + logprior

f = RKHSFun(sq_exp_kernel)
for i in range(30):
    f = f + 0.01 * grad(logprob)(f, data)
```



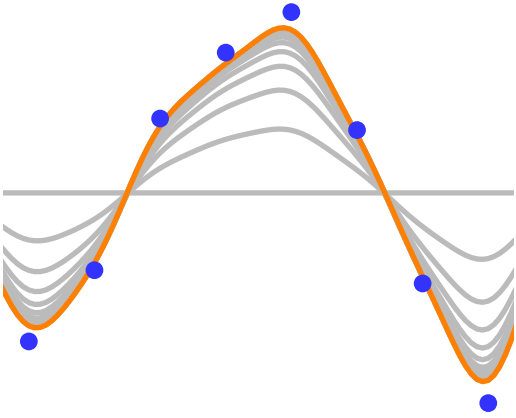
User-defined types: reproducing kernel Hilbert space

```
defvjp(RKHSFun.__call__,
    lambda ans, f, x: lambda g:
    RKHSFun(f.kernel, {x : g}))

def logprob(f, data):
    loglik   = -sum((f(x) - y)**2
                    for x, y in data)
    logprior = -norm_sq(f)
    return loglik + logprior

f = RKHSFun(sq_exp_kernel)
for i in range(30):
    f = f + 0.01 * grad(logprob)(f, data)
```



User-defined types: reproducing kernel Hilbert space

```
defvjp(RKHSFun.__call__,
    lambda ans, f, x: lambda g:
    RKHSFun(f.kernel, {x : g}))

def logprob(f, data):
    loglik   = -sum((f(x) - y)**2
                    for x, y in data)
    logprior = -norm_sq(f)
    return loglik + logprior

f = RKHSFun(sq_exp_kernel)
for i in range(30):
    f = f + 0.01 * grad(logprob)(f, data)
```



Implementation of black-box stochastic variational inference

Ryan Adams
@ryan_p_adams

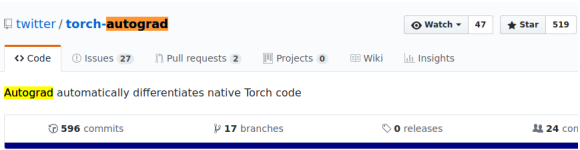
Replying to @DavidDuvenaud

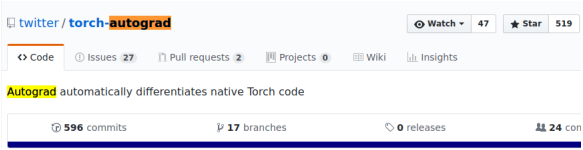
@DavidDuvenaud

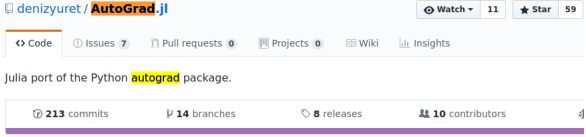
```
def elbo(p, lp, D, N):
 v=exp(p[D:])
 s=randn(N,D)*sqrt(v)+p[:D]
 return mvn.entropy(0,diag(v))+mean(lp(s))
gf = grad(elbo)
```

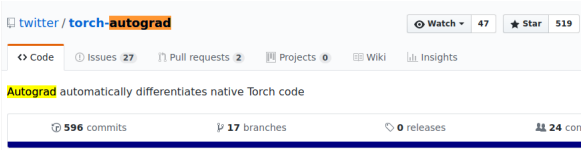
9:43 AM - 7 Nov 2015

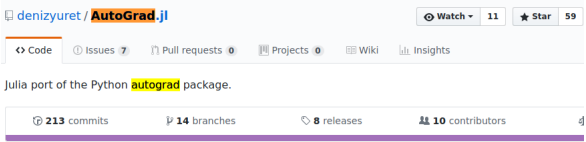
1. Kucukelbir *et al.*, Automatic Differentiation Variational Inference, JMLR 2017
2. Kingma and Welling, Auto-Encoding Variational Bayes, ICLR, 2014
3. Blundell *et al.*, Weight Uncertainty in Neural Networks, arXiv 2015

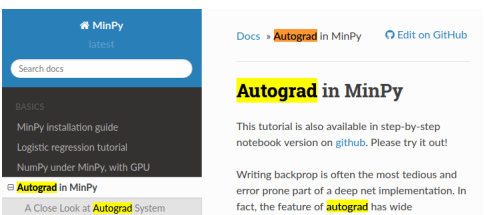

twitter / torch-autograd
Watch 47
Star 519
Code
Issues 27
Pull requests 2
Projects 0
Wiki
Insights
Autograd automatically differentiates native Torch code
596 commits
17 branches
0 releases
24 con

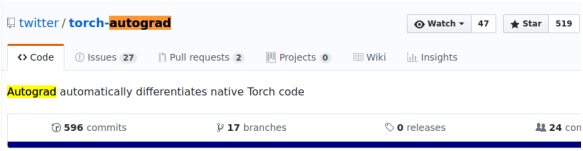

twitter / torch-autograd
Watch 47
Star 519
Code
Issues 27
Pull requests 2
Projects 0
Wiki
Insights
Autograd automatically differentiates native Torch code
596 commits
17 branches
0 releases
24 con

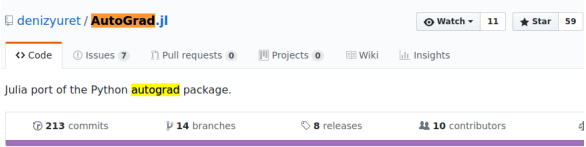

denizyuret / AutoGrad.jl
Watch 11
Star 59
Code
Issues 7
Pull requests 0
Projects 0
Wiki
Insights
Julia port of the Python autograd package.
213 commits
14 branches
8 releases
10 contributors

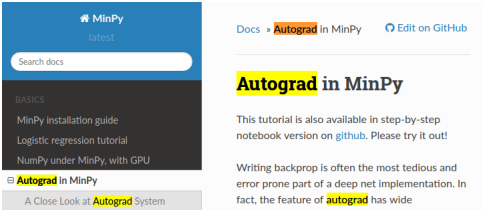

twitter / torch-autograd
Watch 47
Star 519
Code
Issues 27
Pull requests 2
Projects 0
Wiki
Insights
Autograd automatically differentiates native Torch code
596 commits
17 branches
0 releases
24 con


denizyuret / AutoGrad.jl
Watch 11
Star 59
Code
Issues 7
Pull requests 0
Projects 0
Wiki
Insights
Julia port of the Python autograd package.
213 commits
14 branches
8 releases
10 contributors


MinPy
latest
Search docs
BASICS
MinPy installation guide
Logistic regression tutorial
NumPy under MinPy, with GPU
Autograd in MinPy
A Close Look at Autograd System
Docs » Autograd in MinPy
Edit on GitHub
Autograd in MinPy
This tutorial is also available in step-by-step notebook version on github. Please try it out!
Writing backprop is often the most tedious and error prone part of a deep net implementation. In fact, the feature of autograd has wide

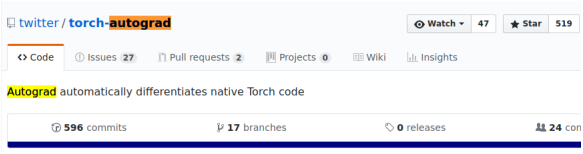


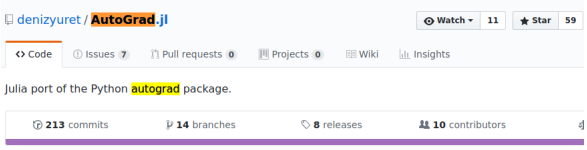


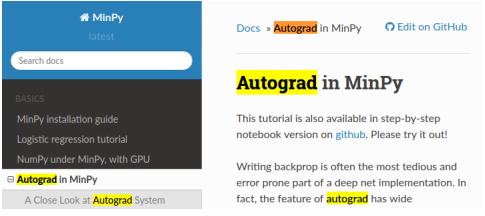


Flexible, Imperative Structure:

Prototype, build, and train neural networks in fully imperative manner using the MXNet autograd package and the Gluon trainer method:

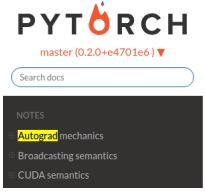


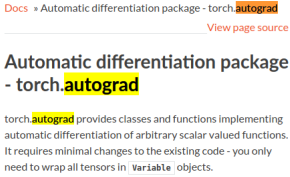




Flexible, Imperative Structure:

Prototype, build, and train neural networks in fully imperative manner using the MXNet autograd package and the Gluon trainer method:





ТАКСИ

(343) 272-72-72

г. Екатеринбург
ул. Белинского 222 оф. 2а

Для корпоративных клиентов

ВХОД

ГРУЗОПЕРЕВОЗКИ

(343) 311-08-08

На главную

О КОМПАНИИ

НОВОСТИ

УСЛУГИ: АВТОБУСЫ
ГРУЗОПЕРЕВОЗКИ
ГРУЗЧИКИ / ПЕРЕЕЗДЫ
ЭКСПЕДИРОВАНИЕ ГРУЗА

ТАРИФЫ ТАКСИ

ТАРИФЫ ГРУЗОПЕРЕВОЗКИ

ТАРИФЫ ПАССАЖИРСКИЕ
ПЕРЕВОЗКИ

ЛЕГКОВОЕ ТАКСИ

Такси Автоград

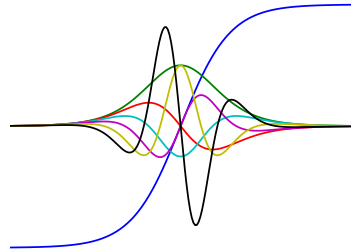
«Такси Автоград», одно из крупнейших такси в Екатеринбурге, которое осуществляет деятельность в полном соответствии с ФЗ- №69 о деятельности Такси с 2003 года.

Автопарк легковых такси подразделяется на две категории – автомобили комфорт и бизнес класса, на все случаи, будь то бюджетная доставка рядовых сотрудников, встреча/трансфер важных гостей в аэропорту или ЖД вокзале, или транспортное обслуживание корпоративного выезда.

- **Бесплатная подача** такси в любой район Екатеринбурга.
- **Автопарк** состоит из автомобилей иностранного производства, не старше трех лет.
- **Аккуратные и вежливые** водители, граждане РФ, со стажем не менее 5-ти лет.
- **Прием заказов** круглосуточно, как предварительно, так и в текущем режиме.
- **Вежливые** операторы

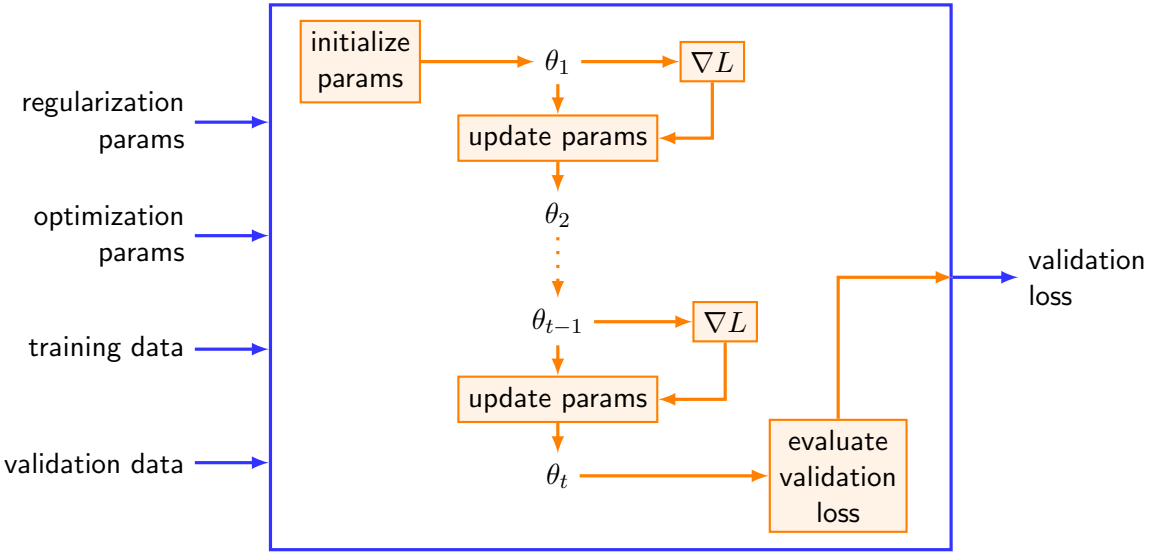
Summary: Autograd

- Traces Python programs to evaluate derivatives
- Simple core, easy to extend
- Inspired several other libraries

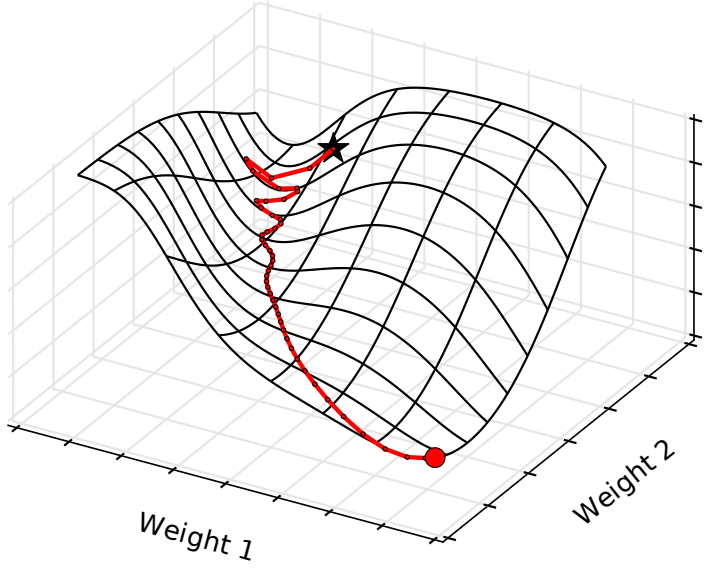


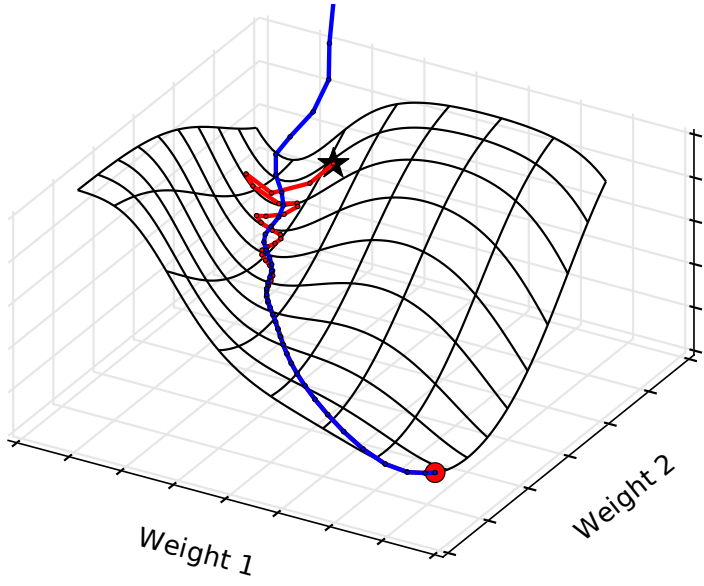
`https://github.com/HIPS/autograd`

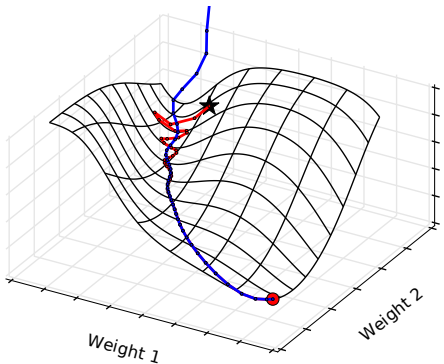
1. Snoek, Larochelle and Adams, Practical Bayesian Optimization of Machine Learning Algorithms, NIPS 2012
2. Golovin *et al.*, Google Vizier: A Service for Black-Box Optimization, SIGKDD 2017
3. Bengio, Gradient-Based Optimization of Hyperparameters, Neural Computation 2000
4. Domke, Generic Methods for Optimization-Based Modeling, AISTATS 2012



1. Snoek, Larochelle and Adams, Practical Bayesian Optimization of Machine Learning Algorithms, NIPS 2012
2. Golovin *et al.*, Google Vizier: A Service for Black-Box Optimization, SIGKDD 2017
3. Bengio, Gradient-Based Optimization of Hyperparameters, Neural Computation 2000
4. Domke, Generic Methods for Optimization-Based Modeling, AISTATS 2012


Weight 1
Weight 2


Weight 1
Weight 2

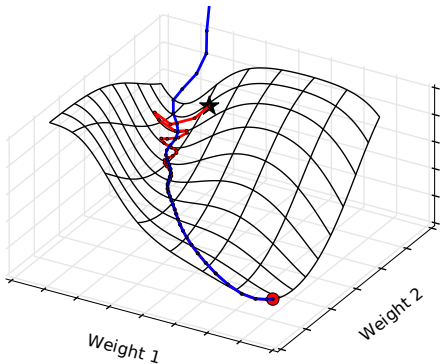


Forward update rule

$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + \alpha \mathbf{v}_t$$
$$\mathbf{v}_{t+1} \leftarrow \beta \mathbf{v}_t - \nabla L\left(\mathbf{x}_{t+1}\right)$$

Reverse update rule

$$\mathbf{v}_t \leftarrow \left(\mathbf{v}_{t+1} + \nabla L\left(\mathbf{x}_{t+1}\right)\right)/\beta$$
$$\mathbf{x}_t \leftarrow \mathbf{x}_{t+1} - \alpha \mathbf{v}_t$$



Forward update rule

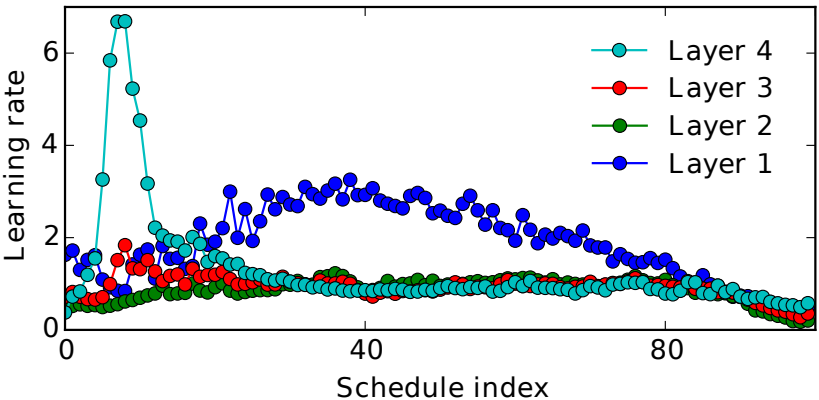
$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + \alpha \mathbf{v}_t$$
$$\mathbf{v}_{t+1} \leftarrow \beta \mathbf{v}_t - \nabla L\left(\mathbf{x}_{t+1}\right)$$

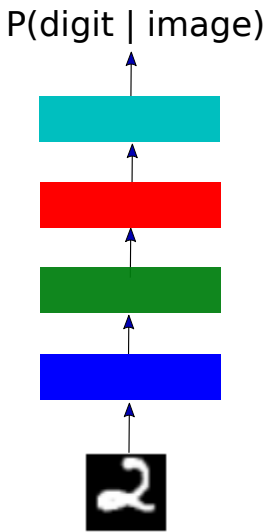
Reverse update rule

$$\mathbf{v}_t \leftarrow \left(\mathbf{v}_{t+1} + \nabla L\left(\mathbf{x}_{t+1}\right)\right)/\beta$$
$$\mathbf{x}_t \leftarrow \mathbf{x}_{t+1} - \alpha \mathbf{v}_t$$

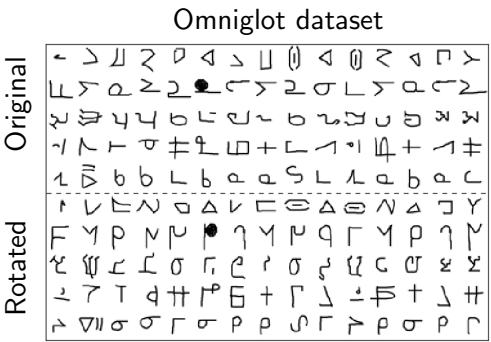
Optimization destroys information!

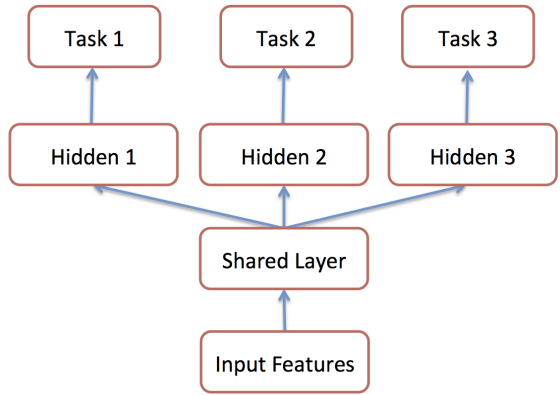
Optimized training schedules





Optimizing architecture





1. Lake, Salakhutdinov and Tenenbaum, Human-Level Concept Learning Through Probabilistic Program Induction, Science 2015

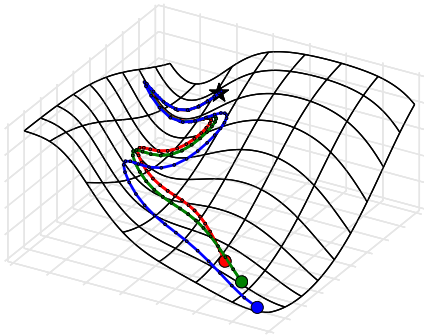
Optimizing architecture

	Input weights	Middle weights	Output weights	Train error	Test error
Separate networks				0.61	1.34
Tied weights				0.90	1.25
Learned sharing				0.60	**1.13**

(Matrices enforce weight-sharing between tasks)

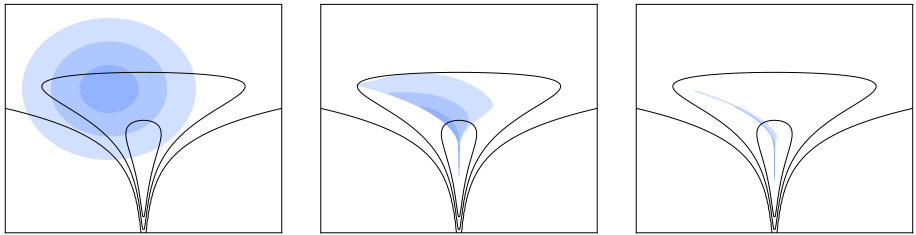
Summary: Hyperparameter optimization and reversible learning

- Exactly reverse learning by storing bits lost
- Optimize thousands of hyperparameters with gradients
- We can optimize
  - Per-parameter regularization
  - Weight initialization
  - Data preprocessing
  - Continuous architecture



D. Maclaurin*, D. Duvenaud* and R. P. Adams, *Gradient-based Hyperparameter Optimization through Reversible Learning.* ICML 2015

Early stopping and ensembling as variational inference

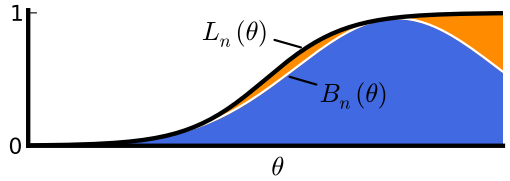


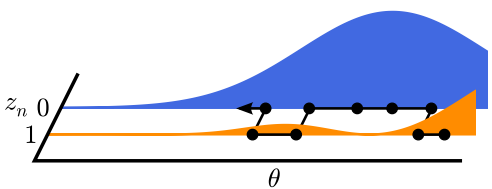
$$\hat{E}[q_t] = -\log p(\mathcal{D}, \hat{\theta}_t)$$

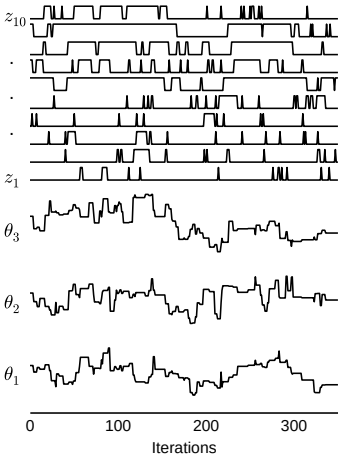
$$S[q_t] = S[q_0] + \sum_{t=1}^{T} \mathbb{E}_{q_t(\theta)} \left[\log \left| J(\theta) \right| \right]$$

D. Duvenaud*, **D. Maclaurin*** and R. P. Adams, *Early Stopping as Nonparametric Variational Inference.* AISTATS 2016

Firefly Monte Carlo

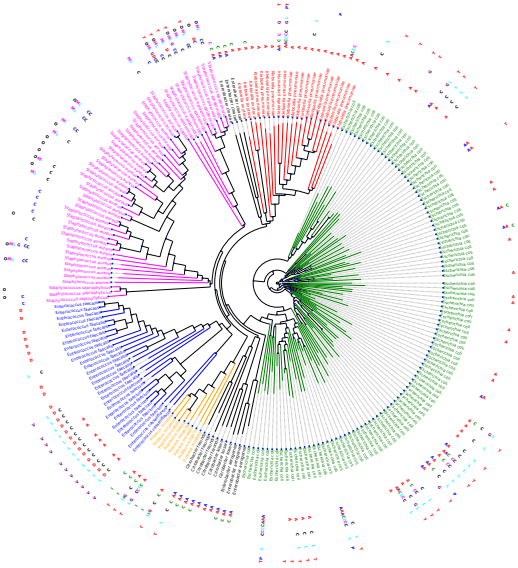




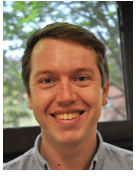


D. Maclaurin and R. P. Adams, *Firefly Monte Carlo: Exact MCMC with Subsets of Data.* UAI 2014 **(best paper award)**

DAYZERO DIAGNOSTICS



Ryan Adams
Professor
Princeton (prev Harvard, currently at Google Brain)



David Duvenaud
Assistant Professor
University of Toronto

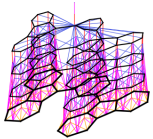
Matt Johnson
Senior Research Scientist
Google Brain

HIPS group and friends

Alex Wiltschko
Andrew Miller
David Reshef
Deborah Hanus
Diana Cai
Elaine Angelino
Finale Doshi-Velez
James Zou
Jasper Snoek
Jon Malmaud
Kevin Swersky
Miguel Hernández-Lobato
Mike Gelbart
Oren Rippel
Scott Linderman
Yakir Reshef

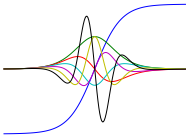
Aspuru-Guzik group

Alán Aspuru-Guzik
Jorge Aguilera-Iparraguirre
Martin Forsythe
Rafael Gómez-Bombarelli
Timothy Hirzel



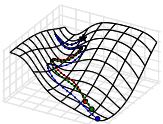
Neural molecular fingerprints

Duvenaud*, **Maclaurin***, Aguilera-Iparraguirre, Gómez-Bombarelli, Hirzel, Aspuru-Guzik & Adams, *NIPS 2015*



Autograd: effortless gradients in Python

Maclaurin, Duvenaud & Johnson, `github.com/HIPS/autograd`



Hyperparameter optimization with reversible learning

Maclaurin*, Duvenaud* & Adams, *ICML 2015*

Early stopping and ensembling as variational inference

$$\mathcal{L}[q] := \underbrace{-\mathbb{E}_q\left[-\log p(\theta, D)\right]}_{\text{Energy } E[q]} + \underbrace{\mathbb{E}_q\left[-\log q(\theta)\right]}_{\text{Entropy } S[q]} \quad \leq \quad \log p(\mathcal{D})$$

$$\hat{E}[q_t] = -\log p(\mathcal{D}, \hat{\theta}_t)$$

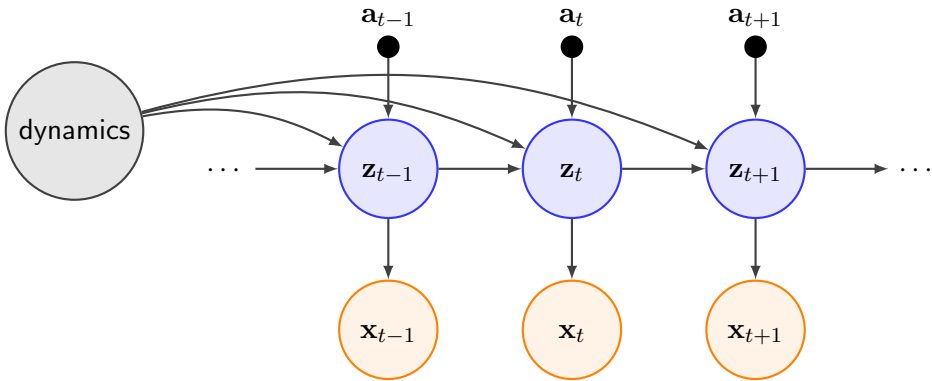
$$S[q_t] = S[q_0] + \sum_{t=1}^{T} \mathbb{E}_{q_t(\theta)}\left[\log \left| J(\theta) \right|\right]$$

Gradient descent, $\quad \theta_{t+1} = \theta_t - \alpha \nabla_\theta L(\theta_t)$

has Jacobian $\quad J(\theta_t) = I - \alpha \nabla_\theta \nabla_\theta^T L(\theta_t)$

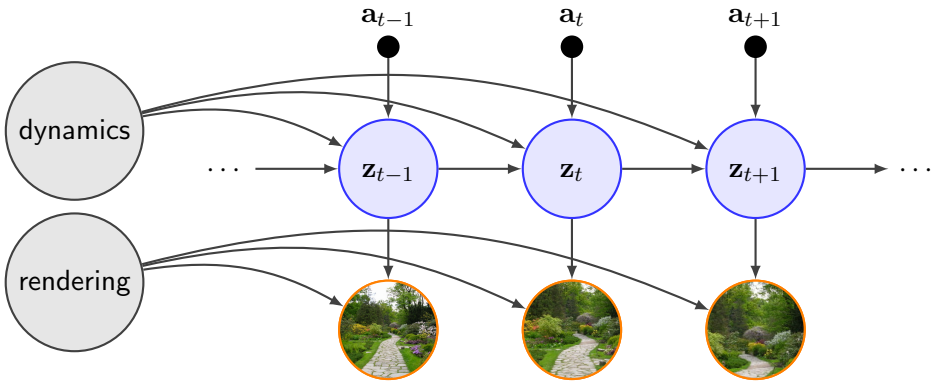
$$\hat{S}[q_t] = S[q_0] + \sum_{t=1}^{T} \log \left| I - \alpha \nabla_\theta \nabla_\theta^T p(\hat{\theta}_t, \mathcal{D}) \right|$$

Model-based RL with structured variational autoencoders



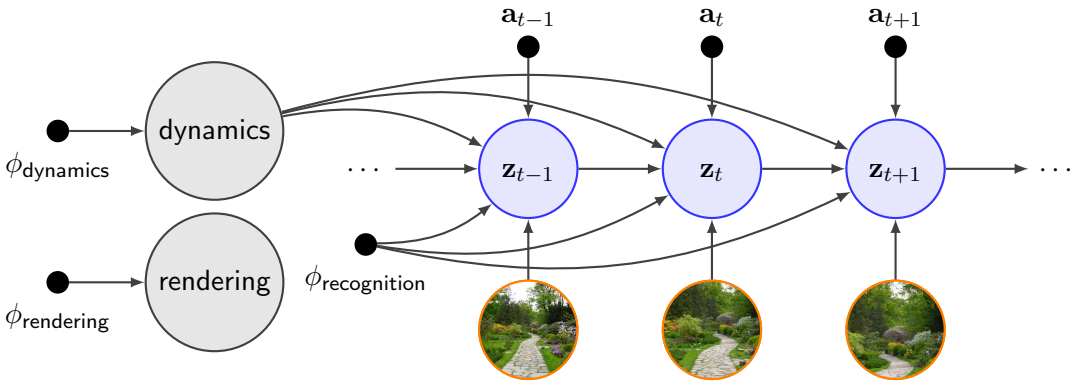
1. Kingma and Welling, Auto-Encoding Variational Bayes, ICLR, 2014
2. Johnson *et al.*, Composing Graphical Models with Neural Networks for Structured Representations and Fast Inference, NIPS 2016
3. Watter *et al.*, Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images, NIPS 2015

Model-based RL with structured variational autoencoders



1. Kingma and Welling, Auto-Encoding Variational Bayes, ICLR, 2014
2. Johnson *et al.*, Composing Graphical Models with Neural Networks for Structured Representations and Fast Inference, NIPS 2016
3. Watter *et al.*, Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images, NIPS 2015

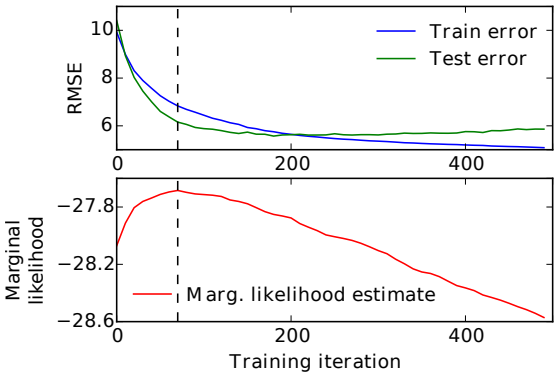
Model-based RL with structured variational autoencoders



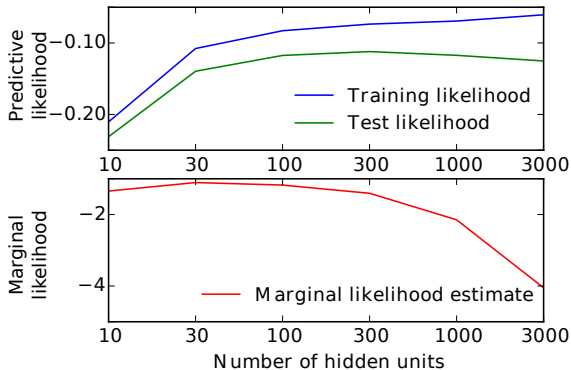
1. Kingma and Welling, Auto-Encoding Variational Bayes, ICLR, 2014
2. Johnson *et al.*, Composing Graphical Models with Neural Networks for Structured Representations and Fast Inference, NIPS 2016
3. Watter *et al.*, Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images, NIPS 2015

Using the marginal likelihood estimator for model selection

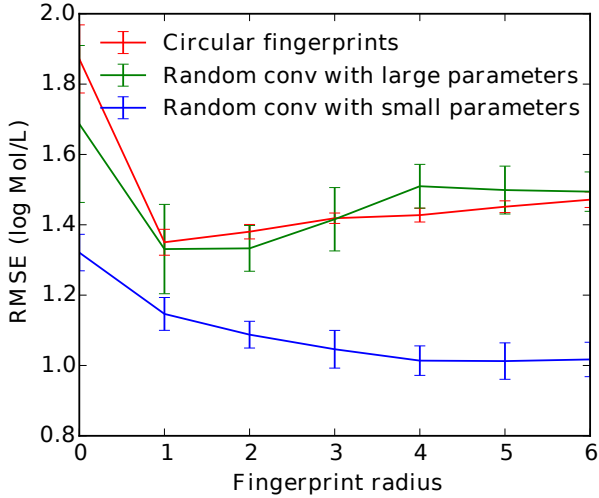
Choosing when to stop training



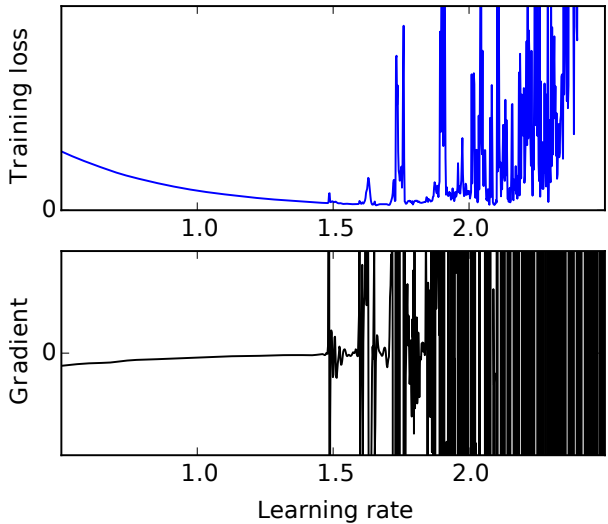
Choosing size of hidden layers



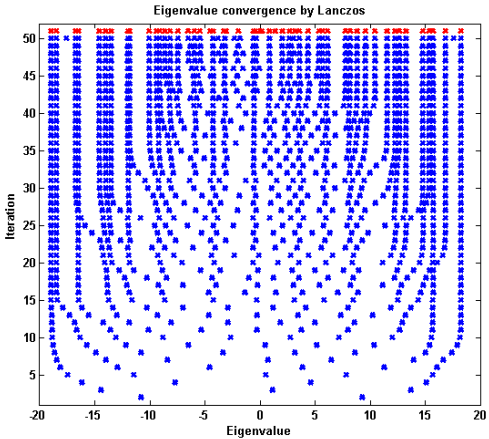
Neural fingerprints with large random weights are like circular fingerprints

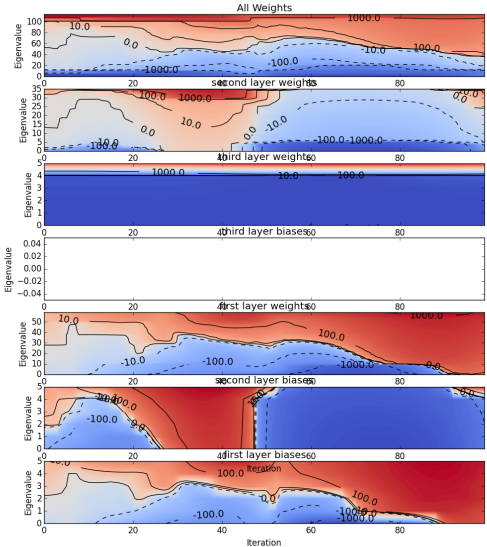


Limitations: unstable learning dynamics

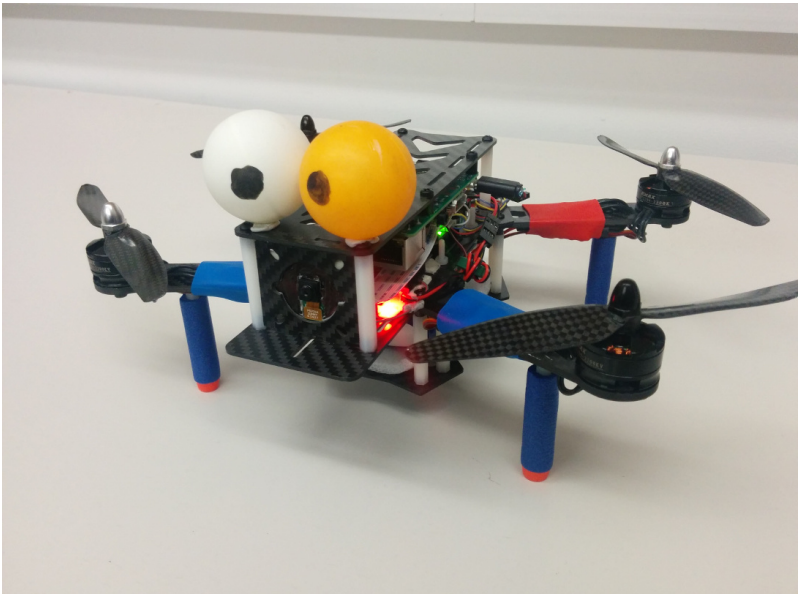


Eigenscapes

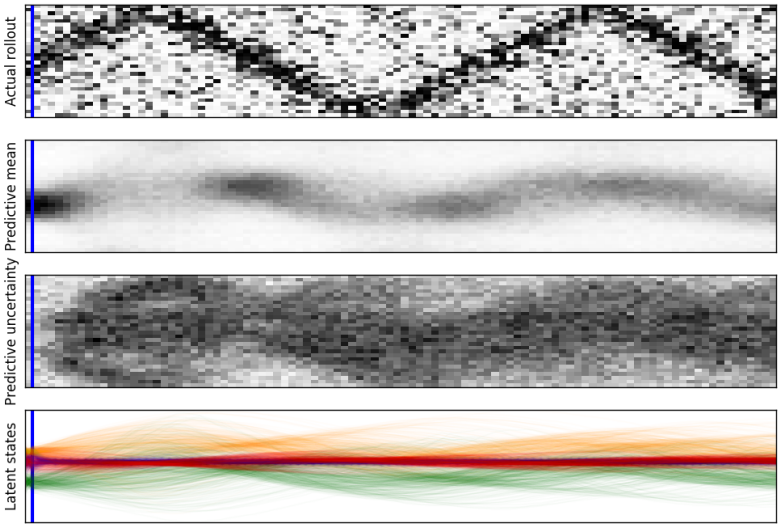




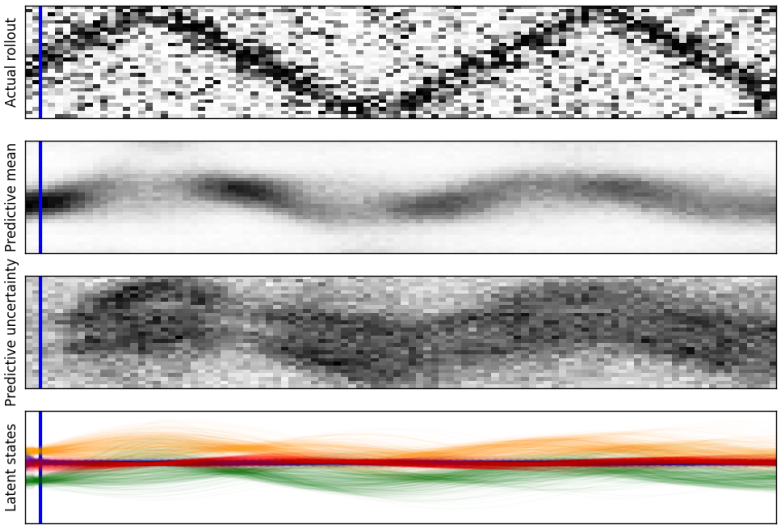
Drongo



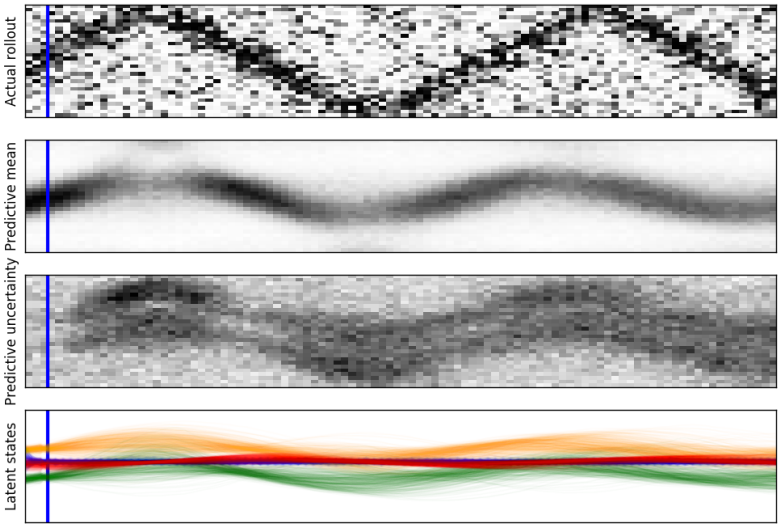
Bouncing ball: filtering and predicting



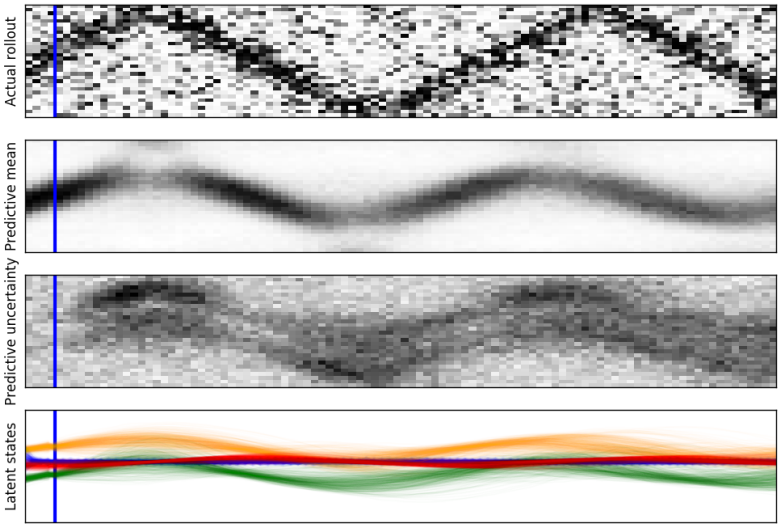
Bouncing ball: filtering and predicting



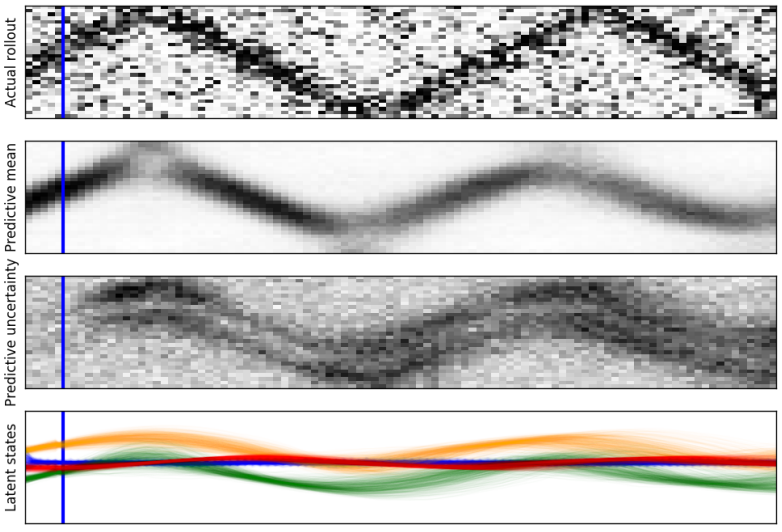
Bouncing ball: filtering and predicting



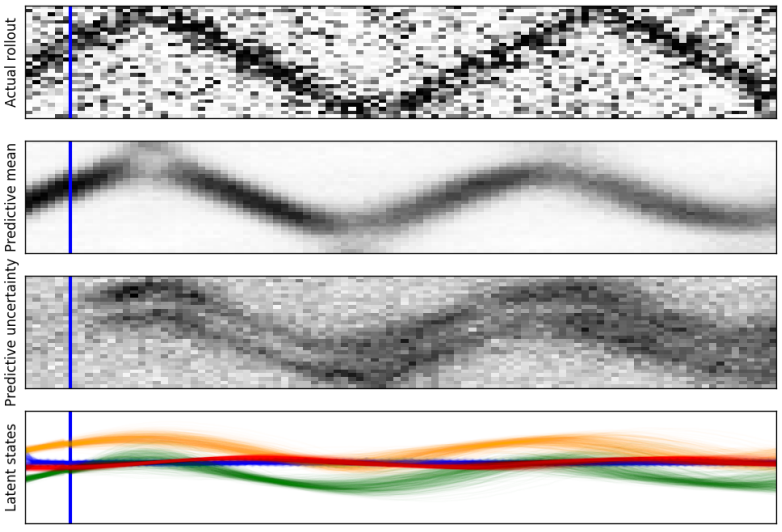
Bouncing ball: filtering and predicting



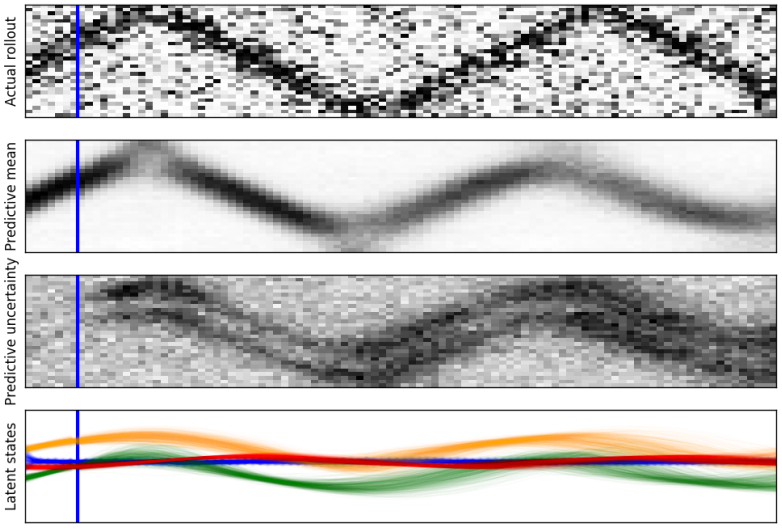
Bouncing ball: filtering and predicting



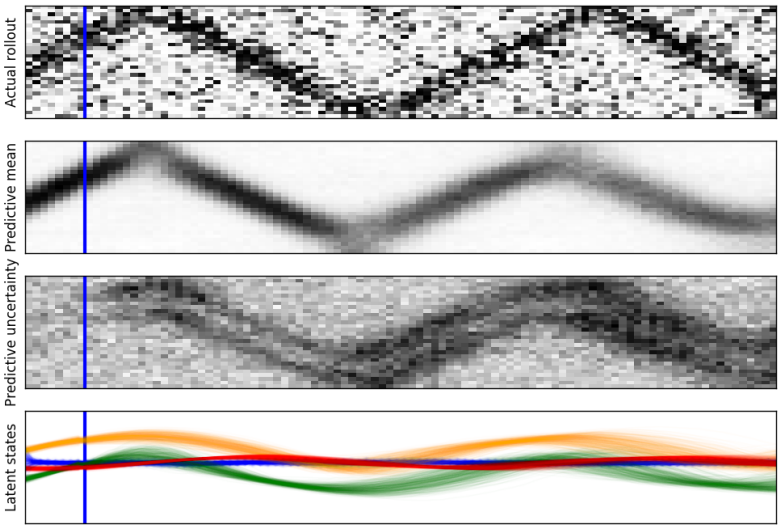
Bouncing ball: filtering and predicting



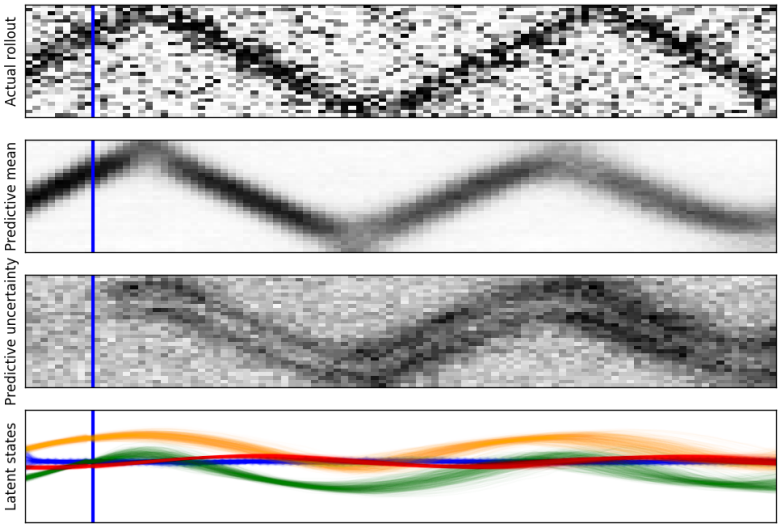
Bouncing ball: filtering and predicting



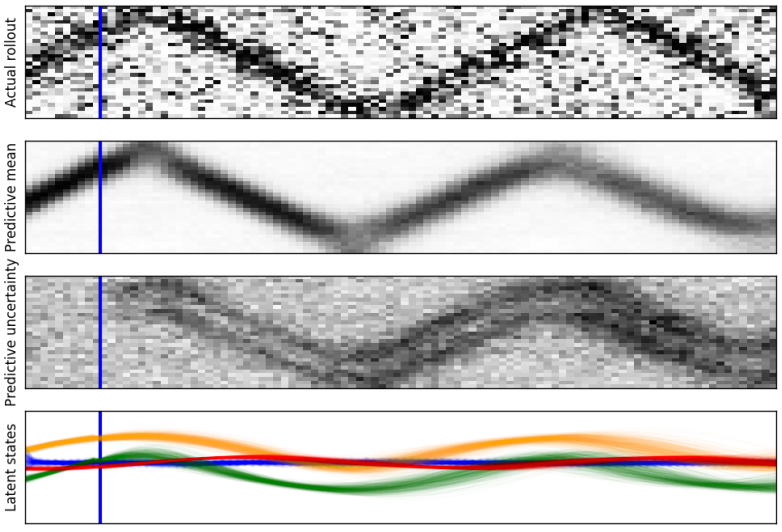
Bouncing ball: filtering and predicting



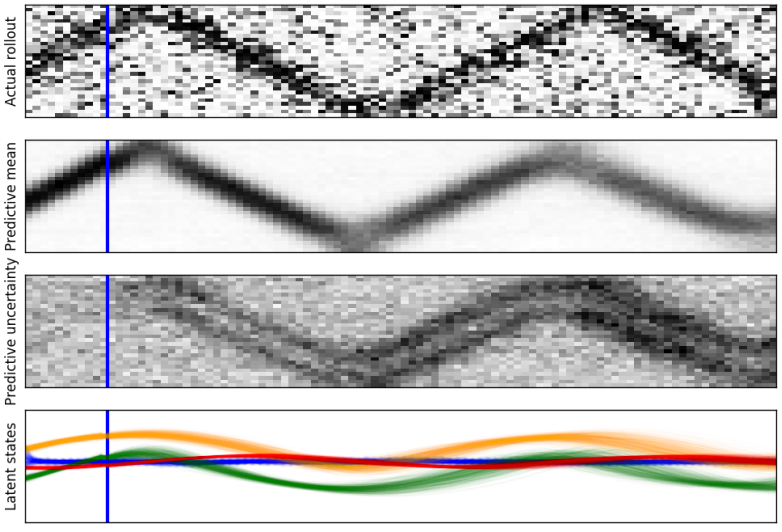
Bouncing ball: filtering and predicting



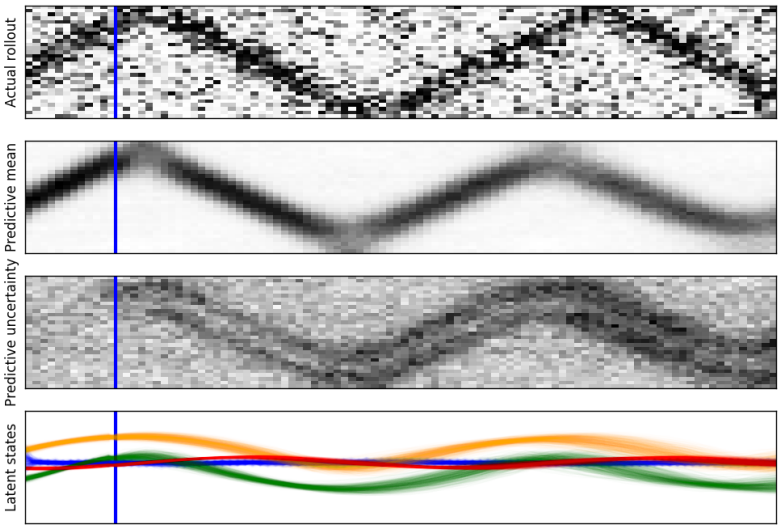
Bouncing ball: filtering and predicting



Bouncing ball: filtering and predicting



Bouncing ball: filtering and predicting



Bouncing ball: filtering and predicting