Neural ODEs and its applications

Piotr Borycki Piotr Kubacki

Jagiellonian University
Faculty of Mathematics and Computer Science
Institute of Computer Science and Computational Mathematics

Thursday 16th May, 2024

Agenda

1. Intro
2. Introduction to flow models
   - Likelihood function
3. Continuous normalizing flows
   - About optimizing with gradient of loss
   - About adjoint state
4. FFJORD
   - Complexity improvement
5. Meta-learning, FSL
   - Meta Learning
   - Few-shot learning
6. MAML
   - MAML
   - Hypernetworks
   - HyperMAML
7. Bayesian methods in ML
8. Bayesian HyperMAML

Contents

1. Intro
2. Introduction to flow models
   - Likelihood function
3. Continuous normalizing flows
   - About optimizing with gradient of loss
   - About adjoint state
4. FFJORD
   - Complexity improvement
5. Meta-learning, FSL
   - Meta Learning
   - Few-shot learning
6. MAML
   - MAML
   - Hypernetworks
   - HyperMAML
7. Bayesian methods in ML
8. Bayesian HyperMAML

Universal approximation theorem (width)

Theorem (Universal approximation theorem)

Let $C(X, \mathbb{R}^m)$ denote the set of continuous functions from a subset X of a Euclidean $\mathbb{R}^n$ space to a Euclidean space $\mathbb{R}^m$. Let $\sigma \in C(\mathbb{R}, \mathbb{R})$. Note that $(\sigma \circ x)_i = \sigma(x_i)$, so $\sigma \circ x$ denotes σ applied to each component of x.

Then σ is not polynomial if and only if for every $n \in \mathbb{N}$, $m \in \mathbb{N}$, compact $K \subseteq \mathbb{R}^n$, $f \in C(K, \mathbb{R}^m), \varepsilon > 0$ there exist $k \in \mathbb{N}$, $A \in \mathbb{R}^{k \times n}$, $b \in \mathbb{R}^k$, $C \in \mathbb{R}^{m \times k}$ such that

$$\sup_{x \in K} \|f(x) - g(x)\| < \varepsilon$$

where $g(x) = C \cdot (\sigma \circ (A \cdot x + b))$

Taking depth to the limit

Many machine learning models (e.g. residual networks, recurrent neural network decoders and normalizing flows) build on transforming hidden state by below formula

$$h_{t+1} = h_t + f(h_t, \theta_t)$$

where $h_t \in \mathbb{R}^D$ and $t \in \{0, \ldots, T\}$.
This iterative formula is very similiar to Euler's method of some continuous transformation. What if instead of discrete number of layers, we allowed the transformation to be continuous. In this case we would have to parametrize the dynamics of hidden units that is:

$$\frac{dh(t)}{dt} = f(t, h(t), \theta),$$

where f is represented by a neural network with parameters θ

ResNet vs ODENet

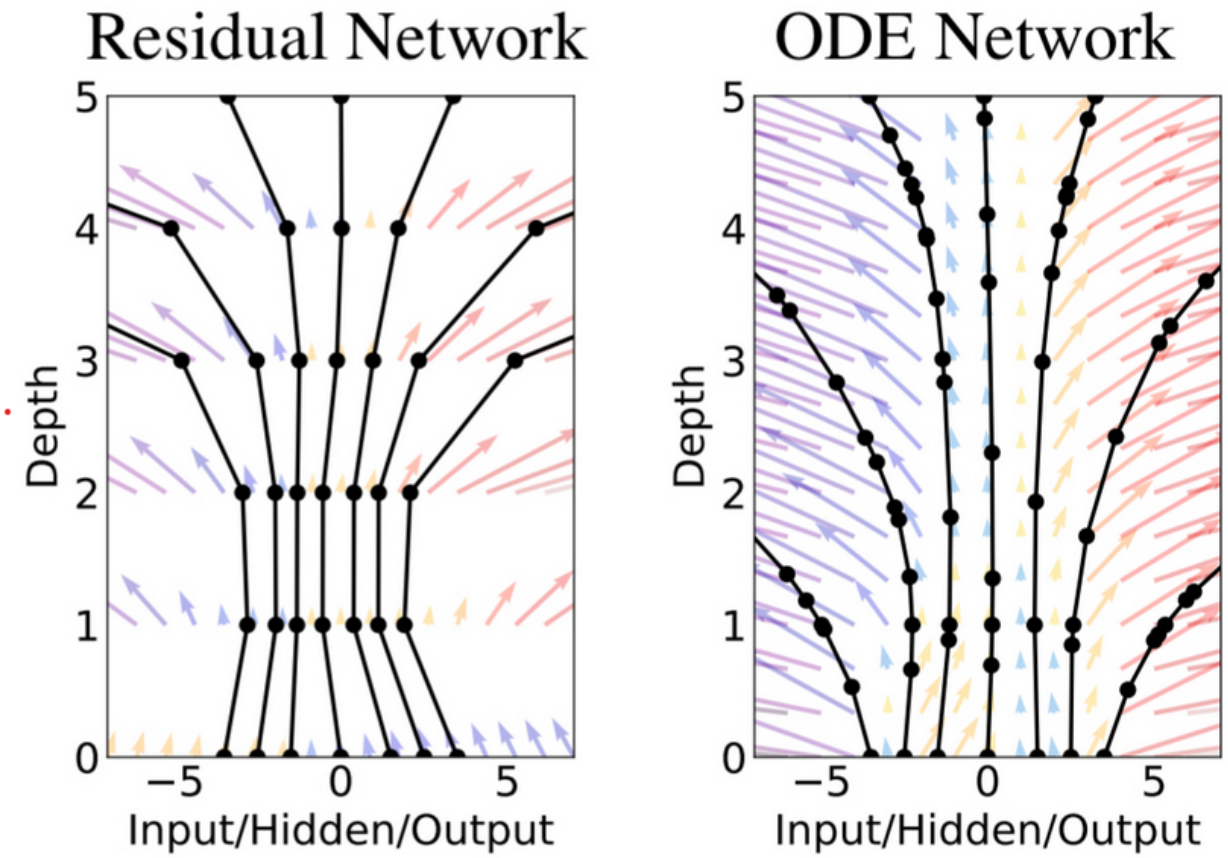


Figure:
Left: A Residual network defines a discrete sequence of finite transformations. Right: A ODE network defines a vector field, which continuously transforms the state. Both: Circles represent evaluation locations

NODE (Neural ODE)

For an arbitrary neural network $f_\theta(z(t), t)$ and a transformation of a hidden state given by:

$$z(t+1) = z(t) + f_\theta(z(t), t)$$

We define an Initial Value Problem as follows:

$$\begin{cases} \frac{dz(t)}{dt} = f_\theta(z(t), t), \ \ t \in [0, T] \\ z(0) = z_0 \end{cases}$$

NODE (Neural ODE)

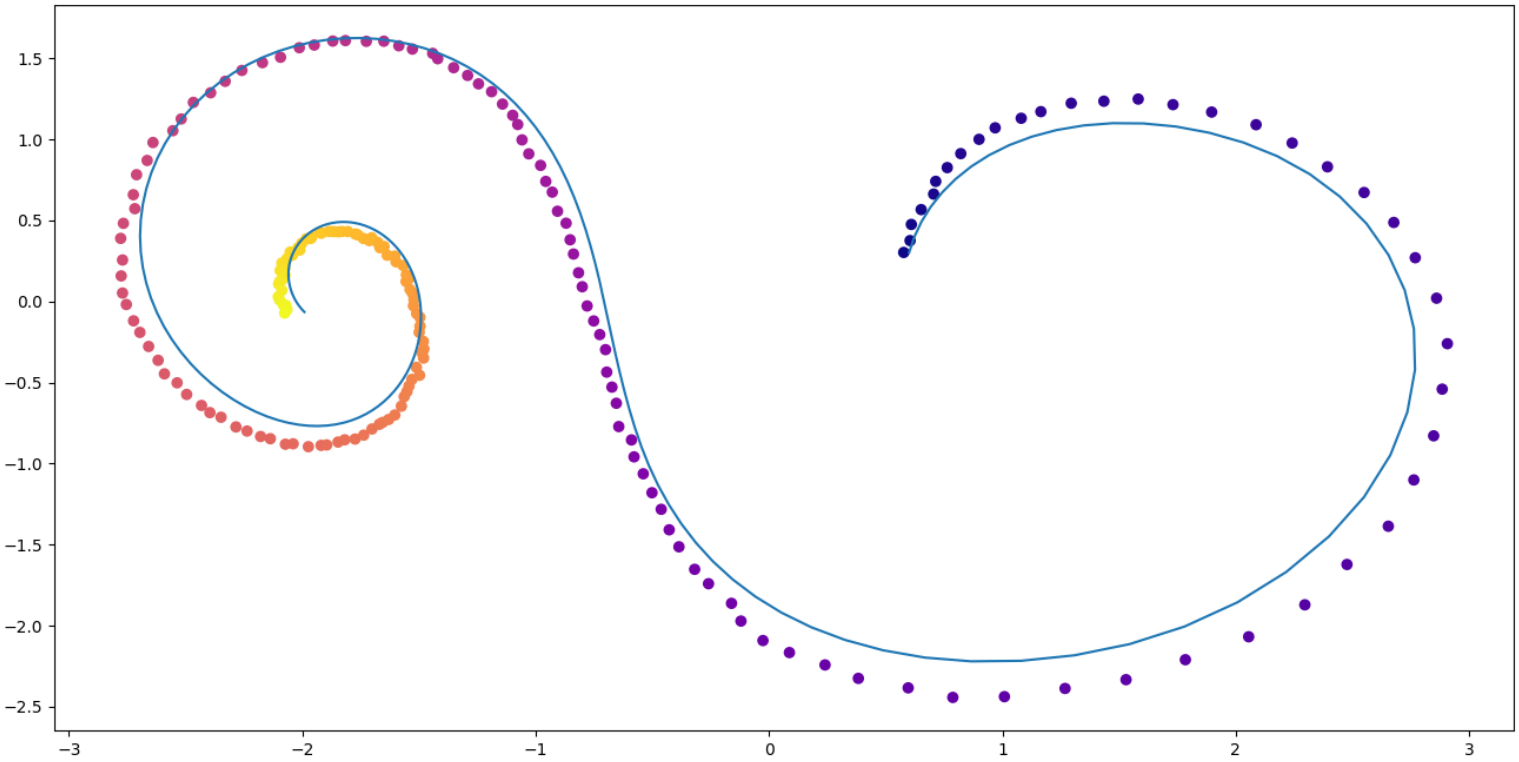


Figure:
Neural-ODE network learns dynamics of state transformation given the transformation stages dataset and ODE-Solver. [see animation]

Contents

1. Intro
2. Introduction to flow models
   - Likelihood function
3. Continuous normalizing flows
   - About optimizing with gradient of loss
   - About adjoint state
4. FFJORD
   - Complexity improvement
5. Meta-learning, FSL
   - Meta Learning
   - Few-shot learning
6. MAML
   - MAML
   - Hypernetworks
   - HyperMAML
7. Bayesian methods in ML
8. Bayesian HyperMAML

Likelihood function

Let $(g_\theta)_\theta$ be a density family parametrized by θ. Let $X = (x_i)_{i=1,\dots n} \subset \mathbb{R}^n$ be the dataset. We are looking for the best density g_θ. To do so, we are searching for the θ such that

$$\arg\max_\theta \ \mathrm{logl}(X, g_\theta) = \arg\max \frac{1}{n} \sum_i \log g_\theta(x_i)$$

How do we model these densities?

First approach

We could try to choose a specific distribution like $\mathcal{N}(\mu, \Sigma)$

Task: Solve $\arg\max_{(\mu,\Sigma)} -\frac{n}{2} \log\det\Sigma - \frac{1}{2} \sum_i^n (y_i - \mu)^T \Sigma^{-1} (y_i - \mu)$

Likelihood function

- Pros: We have the formula for the likelihood function
- Cons: We are narrowing the solution to the normal (or any other fixed) distribution

Likelihood function

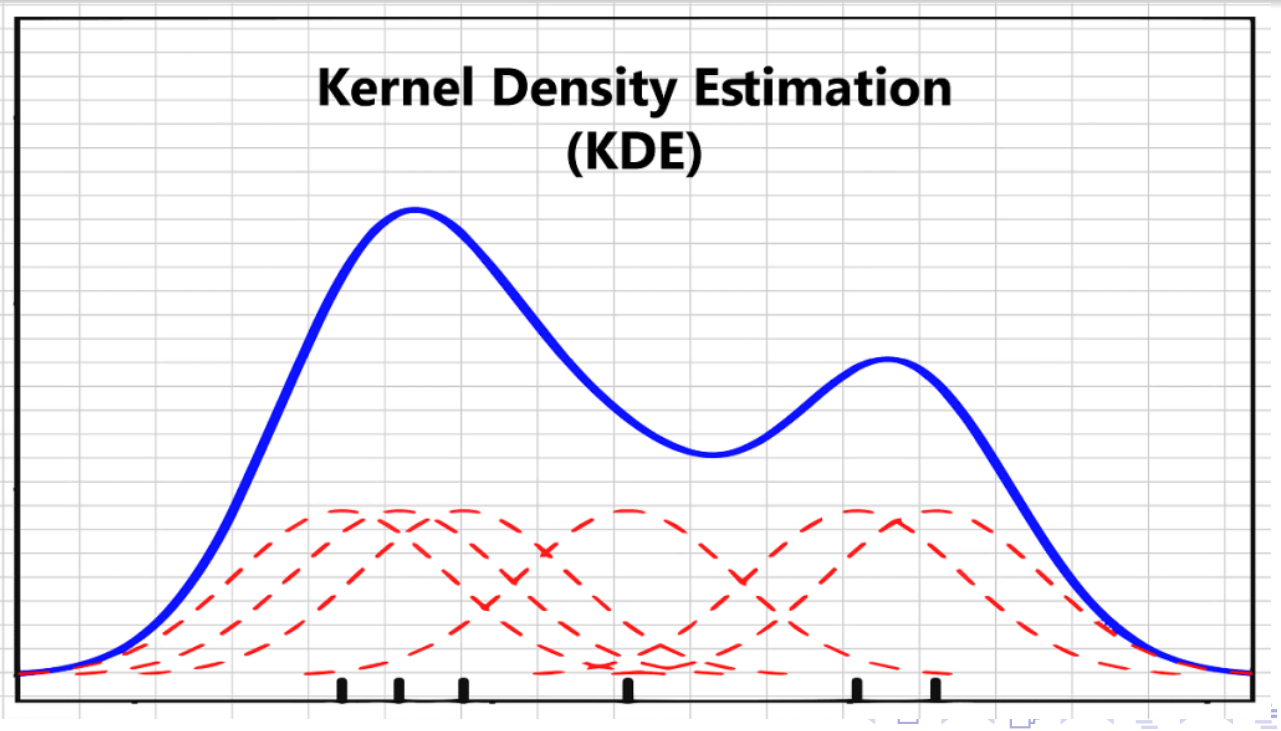
Second approach

Kernel density estimation

- Estimate assuming kernel function $K : \mathbb{R}^n \longrightarrow [0, \infty)$

Estimate density function $\hat{f} : \mathbb{R}^n \longrightarrow [0, \infty)$

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K(\frac{x - x_i}{h})$$



Likelihood function

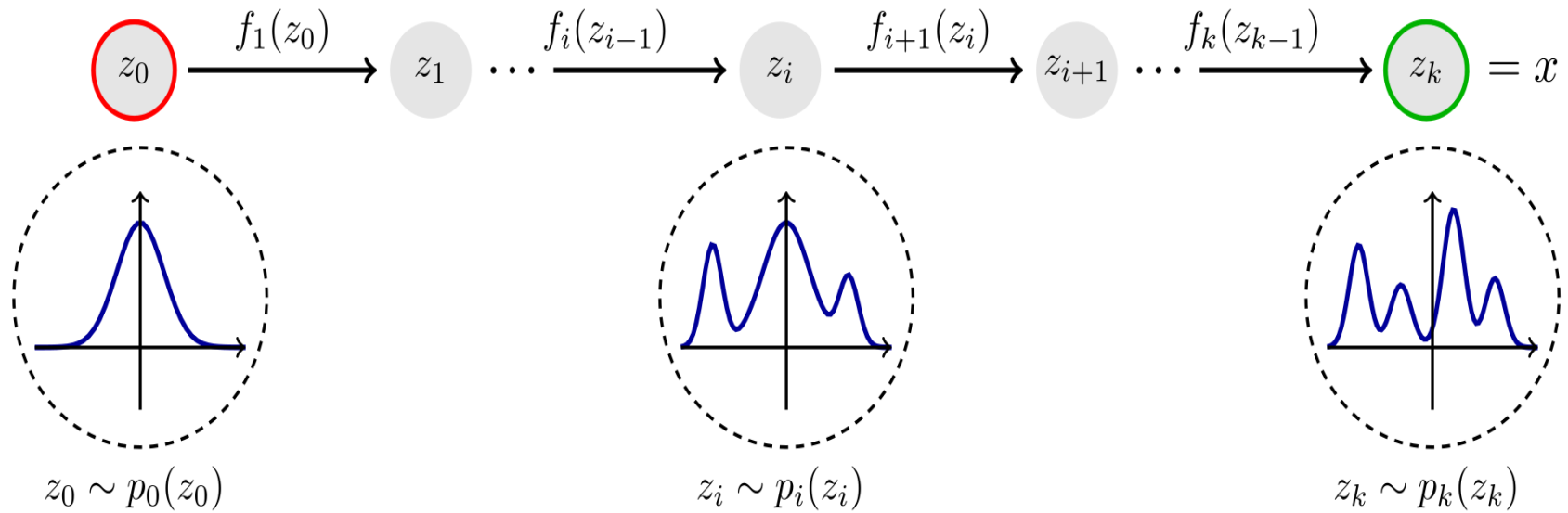
- Pros: We can choose the kernel
- Pros: Much stronger flexity of the result density function
- Cons: We are still imposing the specific kernel to the data

Likelihood function

We would like to create conditions such that we don't really impose any specific distribution.

Idea [Flow]

- We want to get a transformation f from some base distribution to the distribution we want to model. We want to twist and bend some base distribution without fixed restrictions to our dataset X in such a way that it maximizes likelihood function



Flow models construction

Let $z_0 \sim p_0 = \mathcal{N}(0, \mathrm{I})$.

Let $\{f_i\}_{i=1}^n$ be invertible and diffrentiable.

Let $\{z_i\}_{i=1}^n$ be such that $z_{i+1} = f_i(z_i)$ for $i = 0, 1, \cdots, n-1$.

Let $z_k \sim p_k$ for $k = 1, 2, \cdots, n$.

Flow models construction

Theorem

With the assumptions above we get:

$$-\log p_k(z_k) = -\log p_0(z_0) + \sum_{i=1}^{n} \log \left| \det \frac{d\ f_{i-1}(z_{i-1})}{d\ z_{i-1}} \right| \tag{1}$$

where: $\frac{d\ f_{i-\mathbf{1}}(z_{i-\mathbf{1}})}{d\ z_{i-\mathbf{1}}}$ *is a jacobian of* f_{i-1} *at* z_{i-1}.

Proof.

From the change of variable rule and the inverse function theorem we receive:

$$p_1(z_1) = p_0(z_0) \left| \det \frac{d\ f_1^{-1}(z_1)}{d\ z_1} \right| = p_0(z_0) \left| \det \left(\frac{d\ f_1(z_1)}{d\ z_1} \right)^{-1} \right| \tag{2}$$

Proof.

Jacobian matrix of f is invertible because f_i and f_i^{-1} are diffrentiable, so we get:

$$p_0(z_0)\,|\det(\frac{d\ f_1(z_1)}{d\ z_1})^{-1}| = p_0(z_0)\,|\det(\frac{d\ f_1(z_1)}{d\ z_1})|^{-1} \tag{3}$$

Inductively:

$$p_k(z_k) = p_0(z_0)\ \prod_{i=1}^{n} |\det \frac{d\ f_{i-1}(z_{i-1})}{d\ z_{i-1}}|^{-1} \tag{4}$$

$$\log p_k(z_k) = \log p_0(z_0) - \sum_{i=1}^{n} \log |\det \frac{d\ f_{i-1}(z_{i-1})}{d\ z_{i-1}}| \tag{5}$$

Flow models construction

Proof.

We receive a function that we can minimize:

$$-\log p_k(z_k) = -\log p_0(z_0) + \sum_{i=1}^{n} \log \left| \det \frac{d\ f_{i-1}(z_{i-1})}{d\ z_{i-1}} \right| \tag{6}$$

□

Architecture

Example of generative architecture that applies normalizing flow is the **Real NVP** model. Transformations $\{f_j\} : \mathbb{R}^d \longrightarrow \mathbb{R}^D$ are such that:

$$\begin{cases} f = (f_1, ..., f_D) \\ f_i(x) = x \ \text{ dla } \ i \leq d \\ f_{d+1:D}(x) = x_{d+1:D} \odot \exp(s(x_{1:d})) + t(x_{1:d}) \end{cases} \tag{7}$$

where: $\odot$ is *element- wise*, maps $s, t : \mathbb{R}^d \longrightarrow \mathbb{R}^{D-d}$ are components to be tought by the network.

It turns out that the maps above are easily invertible:

$$\begin{cases} f_i^{-1}(y) = y \ \text{ dla } \ i \leq d \\ f_{d+1:D}^{-1}(y) = (f_{d+1:D}(y) - t(f_{1:d}(y)) \odot \exp(-s(f_{1:d}(y))) \end{cases} \tag{8}$$

Furthermore, the jacobian matrix of f is always lower- triangular, so its determinant is easy to calculate.

Planar flows

Planar flows are based on transformations f_{planar} from the class:

$$\mathcal{F}(g) := \{f_{\text{planar}} : \mathbb{R}^d \to \mathbb{R}^d \mid \mathrm{x} \mapsto \mathrm{x} + \mathrm{u}g(\mathrm{w}^T\mathrm{x} + b), u \in \mathbb{R}^d, w \in \mathbb{R}^d, b \in \mathbb{R}\}$$

, where g in general is a smooth non-linear function.
Additionaly the Jacobian of a function $f \in \mathcal{F}(g)$ is given as:

$$\frac{\partial f}{\partial x} = I + uw^T g'(w^T x + b)$$

And the determinant is then given ast:

$$\det \frac{\partial f}{\partial x} = 1 + u^T w g'(w^T x + b)$$

Planar flows

In general the elements of $\mathcal{F}(g)$ are not necessarily invertible. However, for a function $g(x) = \tanh(x)$, the sufficient condition for invertibility is $w^T u \geq -1$. This condition can be enforced by substituting u after every weight update with:

$$\hat{u}(u, w) = u + [m(w^T u) - w^T u] \frac{w}{\|w\|^2}$$

where $m(x) = -1 + \log(1 + \exp(x))$

Contents

1. Intro
2. Introduction to flow models
   - Likelihood function
3. Continuous normalizing flows
   - About optimizing with gradient of loss
   - About adjoint state
4. FFJORD
   - Complexity improvement
5. Meta-learning, FSL
   - Meta Learning
   - Few-shot learning
6. MAML
   - MAML
   - Hypernetworks
   - HyperMAML
7. Bayesian methods in ML
8. Bayesian HyperMAML

The big picture

1. Without any additional assumptions (like NVP) calculation of $\frac{df}{dz}$ has cubic computational cost in either $dim(z)$ or the number of hidden layers.
2. We can introduce some trade-off between expressiveness of the model and its computational cost
3. Another approach is to move from the discrete set of layers domain to continuous transformation

About optimizing with gradient of loss

Let's consider a scalar-valued loss function L, whose input is the output of a ODE-Solver.

$$L(z(t_1)) = L(\text{ODESOLVER}(z(t), t_0, t_1)) = L(z_0 + \int_{t_0}^{t_1} f(z(t), t, \theta)dt)$$

1. Like in any common deep learning model, we need $\frac{\partial L}{\partial \theta}$.
2. To determine how the loss changes with θ, first we need to determine how the gradient changes depending on hidden state $z(t)$
3. To do that, we introduce $a(t) = \frac{\partial L}{\partial z(t)}$ - adjoint state
4. We will use dynamics of $a(t)$ to obtain dynamics of L on θ

adjoint state - intuition

The adjoint state is the gradient with respect to the hidden state at a specified time. In standard neural networks, the gradient of a hidden layer h_t depends on the gradient from the next layer t_{t+1} by chain rule:

$$\frac{d\ L}{d\ h_t} = \frac{d\ L}{d\ h_{t+1}} \frac{d\ h_{t+1}}{d\ h_t}$$

With a continuous hidden state, we can write the transformation after an ϵ change of time and chain rule can also be applied:

$$\frac{\partial L}{\partial z(t)} = \frac{\partial L}{\partial T_\epsilon(z(t), t)} \frac{\partial T_\epsilon(z(t), t)}{\partial z(t)} \quad \Longleftrightarrow \quad a(t) = a(t+\epsilon) \frac{d\ z(t+\epsilon)}{d\ z(t)}$$

where: $z(t)$ recalls a variable of L

$T_\epsilon(z(t), t) = z(t+\epsilon)$ is a function equal to the state of z in time $t+\epsilon$

About adjoint state

- Let $\frac{d\ z(t)}{d\ t} = f(z(t), t, \theta)$
- Let $a(t) = \frac{d\ L}{d\ z(t)}$
- We will prove that:

$$\frac{d\ a(t)}{d\ t} = -a(t)\frac{\partial f(z(t), t, \theta)}{\partial z(t)}$$

- We will generalize the above to obtain gradients for $\frac{\partial L}{\partial \theta}$

About adjoint state

Theorem

$$\frac{d\ a(t)}{d\ t} = -a(t)\frac{\partial f(z(t), t, \theta)}{\partial z(t)} \tag{9}$$

Proof.

About adjoint state

$$\underbrace{a(t_N) = \frac{\partial L}{\partial z(t_N)}}_{\text{initial condition}}$$

$$\underbrace{a(t_0) = a(t_N) + \int_{t_N}^{t_0} a(t)^T \frac{d\ a(t)}{d\ t} = a(t_N) - \int_{t_N}^{t_0} a(t)^T \frac{\partial f(z(t), t, \theta)}{\partial z(t)}}_{\text{gradient with respect to initial condition}}$$

About adjoint state

Theorem

$$\frac{\partial L}{\partial \theta} = -\int_{t_N}^{t_0} \frac{\partial f(z(t), t, \theta)}{\partial \theta} \tag{10}$$

About adjoint state

$$f_{aug}(\begin{bmatrix} z & \theta & t \end{bmatrix}) := \frac{d}{d\ t} \begin{bmatrix} z \\ \theta \\ t \end{bmatrix} = \begin{bmatrix} f(\begin{bmatrix} z & \theta & t \end{bmatrix}) \\ \mathbf{0} \\ 1 \end{bmatrix}$$

Let's consider θ, t as states with constant differential equation:

$$\frac{d\ \theta(t)}{d\ t} = \mathbf{0} \quad \text{and} \quad \frac{d\ t(t)}{d\ t} = 1$$

$$a_{aug} := \begin{bmatrix} a \\ a_\theta \\ a_t \end{bmatrix} \quad a_\theta(t) := \frac{\partial L}{\partial \theta(t)} \quad a_t(t) := \frac{\partial L}{\partial t(t)}$$

About adjoint state

Proof.

About adjoint state

If we set $a_\theta(t_N) = \mathbf{0}$:

$$\frac{\partial L}{\partial \theta} = \mathbf{0} - \int_{t_N}^{t_0} a(t) \frac{\partial f(z(t), t, \theta)}{\partial \theta} dt$$

$$\frac{\partial L}{\partial t_N} = a(t_N) f(z(t_N), t_N, \theta)$$

$$\frac{\partial L}{\partial t_0} = a_t(t_0) = a_t(t_N) - \int_{t_N}^{t_0} a(t) \frac{\partial f(z(t), t, \theta)}{\partial t} dt$$

Final algorithm

1. Training:
   1. Forward pass:

$$z(t_N) = z_0 + \int_{t_N}^{t_0} f(z(t), t, \theta) dt$$

   where

$$\int_{t_N}^{t_0} f(z(t), t, \theta) dt = (\cdots)\textbf{ODE Solver}$$

   2. Backward pass:

$$\frac{\partial L}{\partial \theta} = -\int_{t_N}^{t_0} a(t) \frac{\partial f(z(t), t, \theta)}{\partial \theta} dt$$

   with

$$\frac{\partial L}{\partial z(t)} = \frac{\partial L}{\partial z(t_N)} - \int_{t_N}^{t_0} a(t) \frac{\partial f(z(t), t, \theta)}{\partial \theta} dt$$

2. Evaluation

$$z(t) = z(t_N) + \int_{t_N}^{t} f(z(t), t, \theta) dt$$

Bottlenecks of discrete normalizing flows

As previously stated discrete normalizing flows utilize the change of variables theorem to compute the exact changes in probability if samples are transformed by a bijective function f:

$$z_1 = f(z_0) \Rightarrow \log p(z_1) = \log p(z_0) - \log \left|\det \frac{\partial f}{\partial z_0}\right|$$

The main bottleneck of using the change of variable formula is computing the determinant of the Jacobian $\frac{\partial f}{\partial z}$, which in general has a cubic time in terms of the dimension of the Jacobian.

How about continuous normalizing flows

Surprisingly, moving from a discrete set of layers to a continuous transformation simplifies the computation

Theorem (Instantaneous Change of Variables)

Let $z(t)$ be a finite continuous random variable with probability $p(z(t))$ dependent on time. Let $\frac{dz}{dt} = f(z(t), t)$ be a differential equation describing a continuous in time transformation of $z(t)$. Assuming that f is uniformly Lipschitz continuous in z and continuous in t, then the change in log probability also follows a differential equation:

$$\frac{d \log p(z(t))}{dt} = -\mathrm{tr}\left(\frac{\partial f}{\partial z}(z(t), t)\right)$$

Jacobi's formula

Theorem (Jacobi's formula)

If A is a differentiable map from real numbers to $n \times n$ matrices, then:

$$\frac{d}{dt} \det A(t) = (\det A(t)) \cdot \operatorname{tr}\left(A(t)^{-1} \cdot \frac{dA(t)}{dt}\right)$$

Comparison of NFs and CNFs

Normalizing flow

$$z_0 \sim p_0(z_0)$$
$$z_1 = f(z_0)$$
f invertible and smooth
$$\log p_1(z_1) = \log p_0(z_0) - \log \left| \det \frac{\partial f}{\partial z_0} \right|$$

Continuous normalizing flow

$$z(t_0) = z_0 \sim p_0(z_0)$$
$$\frac{dz}{dt} = f(z(t), t)$$
f uniformly Lipschitz continuous in z and continuous in t
$$\frac{d \log p(z(t))}{dt} = -\mathrm{tr}\left(\frac{\partial f}{\partial z}(z(t), t)\right)$$

Advantages

- instead of the log determinant, one just needs to compute the trace of the Jacobian which is generally computationally cheaper
- the dynamics f of the differential equation do not need to be bijective

Contents

1. Intro
2. Introduction to flow models
   - Likelihood function
3. Continuous normalizing flows
   - About optimizing with gradient of loss
   - About adjoint state
4. FFJORD
   - Complexity improvement
5. Meta-learning, FSL
   - Meta Learning
   - Few-shot learning
6. MAML
   - MAML
   - Hypernetworks
   - HyperMAML
7. Bayesian methods in ML
8. Bayesian HyperMAML

Complexity improvement

Let us assume that:

- D - dimentionality of the data
- H - size of the largest hidden dimention in f
- $f : \mathbb{R}^D \rightarrow \mathbb{R}^D$ (neural network) has the evaluating cost $\mathcal{O}(DH)$

Then the cost of computing the likelihood is:

1. Normalizing flow (general case): $\mathcal{O}((DH + D^3)L)$ where L is the number of transformations used
2. CNF: $\mathcal{O}((DH + D + D^2)\hat{L})$ where $\hat{L}$ is the number of evaluations of f used by ODE-Solver
3. FFJORD: $\mathcal{O}((DH + D)\hat{L})$

Automatic-differentiation

Some well-known from literature of automatic-differentiation ("Evaluating Derivatives: principles and techniques of algorithmic differentiation" by Griewank and Walther, chapter 3.2):

Reverse mode differentiation can

- calculate the gradient $\frac{\partial g}{\partial \theta}$ of a scalar function g with a time complexity equal to a constant multiple of the time to calculate g
- calculate the vector jacobian product $v^T \frac{\partial f}{\partial z}$ for some constant vector v and vector valued function f with a time complexity equal to a constant multiple of the time to calculate f

Hutchinson's trace estimator

- We can get an unbiased estimate of the trace of a matrix by taking a double product of that matrix with a noise a vector (Hutchinson, 1989).
- Let $A \in \mathbb{R}^{D \times D}$,
  $p(\epsilon)$ - distribution over D-dimensional vectors such that
  $\mathbb{E}(\epsilon) = 0$ and $Cov(\epsilon) = I$

$$Tr(A) = \mathbb{E}_{p(\epsilon)}[\epsilon^T A \epsilon]$$

- The Monte Carlo estimator derived from the equation above is known as the Hutchinson's trace estimator

Hutchinson's trace estimator

$$
\begin{aligned}
& log p(z(t_1)) \\
&= log(p(z(t_0)) - \int_{t_0}^{t_1} Tr(\frac{\partial f}{\partial z(t)} dt \\
&= log(p(z(t_0)) - \int_{t_0}^{t_1} \mathbb{E}_{p(\epsilon)}[\epsilon^T \frac{\partial f}{\partial z(t)} \epsilon] dt \\
&= log(p(z(t_0)) - \mathbb{E}_{p(\epsilon)}[\int_{t_0}^{t_1} \epsilon^T \frac{\partial f}{\partial z(t)} \epsilon dt]
\end{aligned}
$$

Unbiased estimation of trace

1. Sample ϵ

$$\mathcal{O}(1)$$

2. Evaluate dynamics

$$f_t \leftarrow f_\theta(z(t), t))$$

$$\mathcal{O}(DH)$$

3. Compute vector-Jacobian product with automatic differentiation

$$g \leftarrow \epsilon^T \frac{\partial f}{\partial z}|_{z(t)}$$

$$\mathcal{O}(DH)$$

4. Calculate unbiased estimate of trace

$$\hat{Tr} = multiply(g, \epsilon)$$

$$\mathcal{O}(D)$$

Contents

1. Intro
2. Introduction to flow models
   - Likelihood function
3. Continuous normalizing flows
   - About optimizing with gradient of loss
   - About adjoint state
4. FFJORD
   - Complexity improvement
5. Meta-learning, FSL
   - Meta Learning
   - Few-shot learning
6. MAML
   - MAML
   - Hypernetworks
   - HyperMAML
7. Bayesian methods in ML
8. Bayesian HyperMAML

Meta Learning

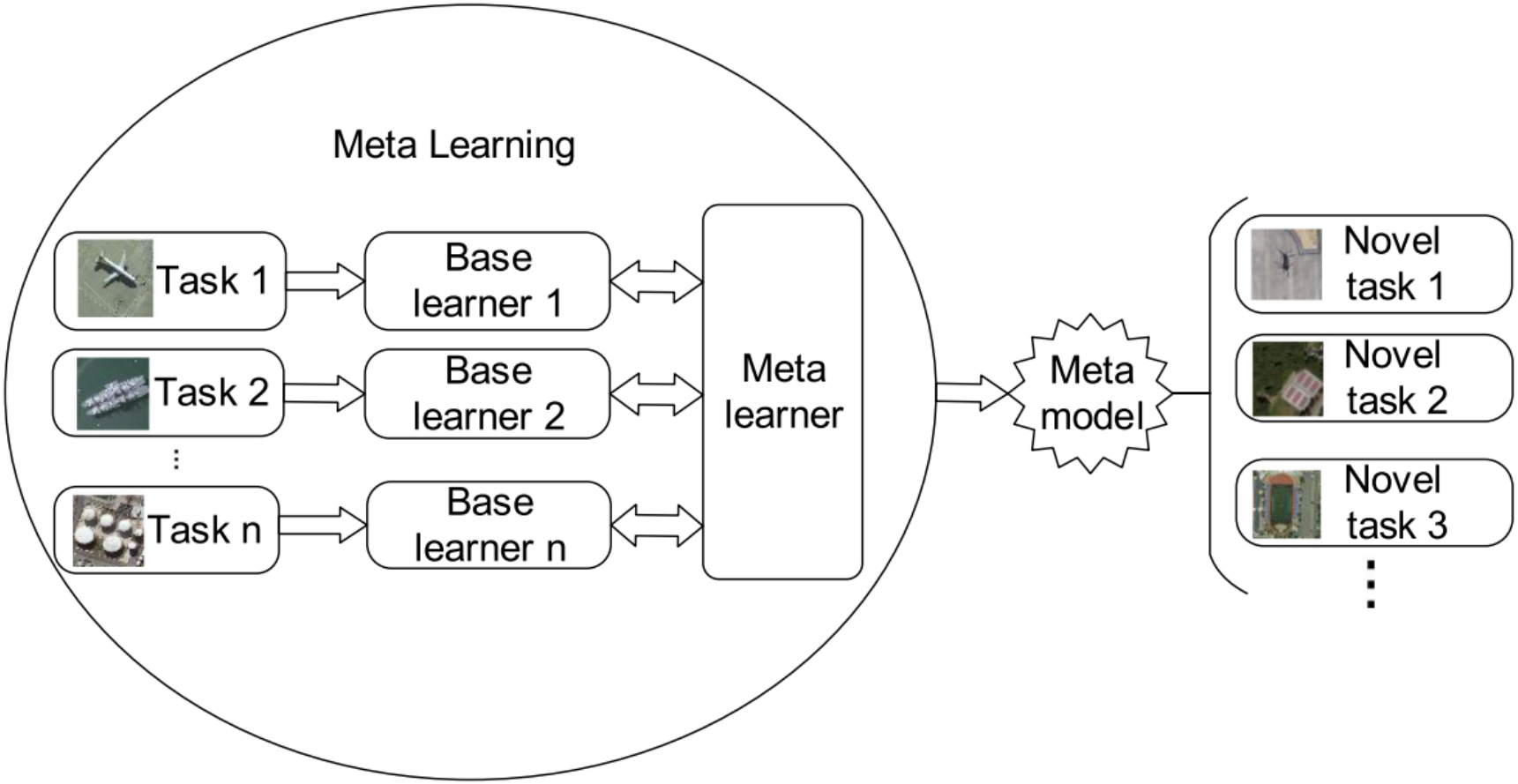


Figure: Illustration of the meta-learning framework

Few-shot learning

- Let $\mathcal{S} = \{(x_l, y_l)\}_{l=1}^{L}$ be called support set where every domain class is represented equally.
- Let $L = k \cdot n$ where: k - amount of classes, n - number of examples per class $\mathcal{S}$.
- We call such a problem a n-way k-shot problem.
- Let $\mathcal{Q} = \{(x_m, y_m)\}_{m=1}^{M}$ obe the query set, where $M \geq k$.
- Let us assume that the label set of $\mathcal{S}$ is equal to the label set of $\mathcal{Q}$ (but number of examples per class may vary)

Formally, we consider a task: $\mathcal{T} = \{\mathcal{S}, \mathcal{Q}\}$.

Few-shot learning - training and inference

- During training, both query and support set have labels
  1. We calculate adaptation by evaluating support set with global parameters θ
  2. We shift the global parameters by evaluating query set with adapted parameters $(\theta + \Delta\theta)$
- Training is expected to shift θ to a place that makes a model flexible and prepared for further adaptations
- During inference only support set have labels
  - We evaluate to the query-set domain after adaptation from support-set

Contents

1. Intro
2. Introduction to flow models
   - Likelihood function
3. Continuous normalizing flows
   - About optimizing with gradient of loss
   - About adjoint state
4. FFJORD
   - Complexity improvement
5. Meta-learning, FSL
   - Meta Learning
   - Few-shot learning
6. MAML
   - MAML
   - Hypernetworks
   - HyperMAML
7. Bayesian methods in ML
8. Bayesian HyperMAML

MAML

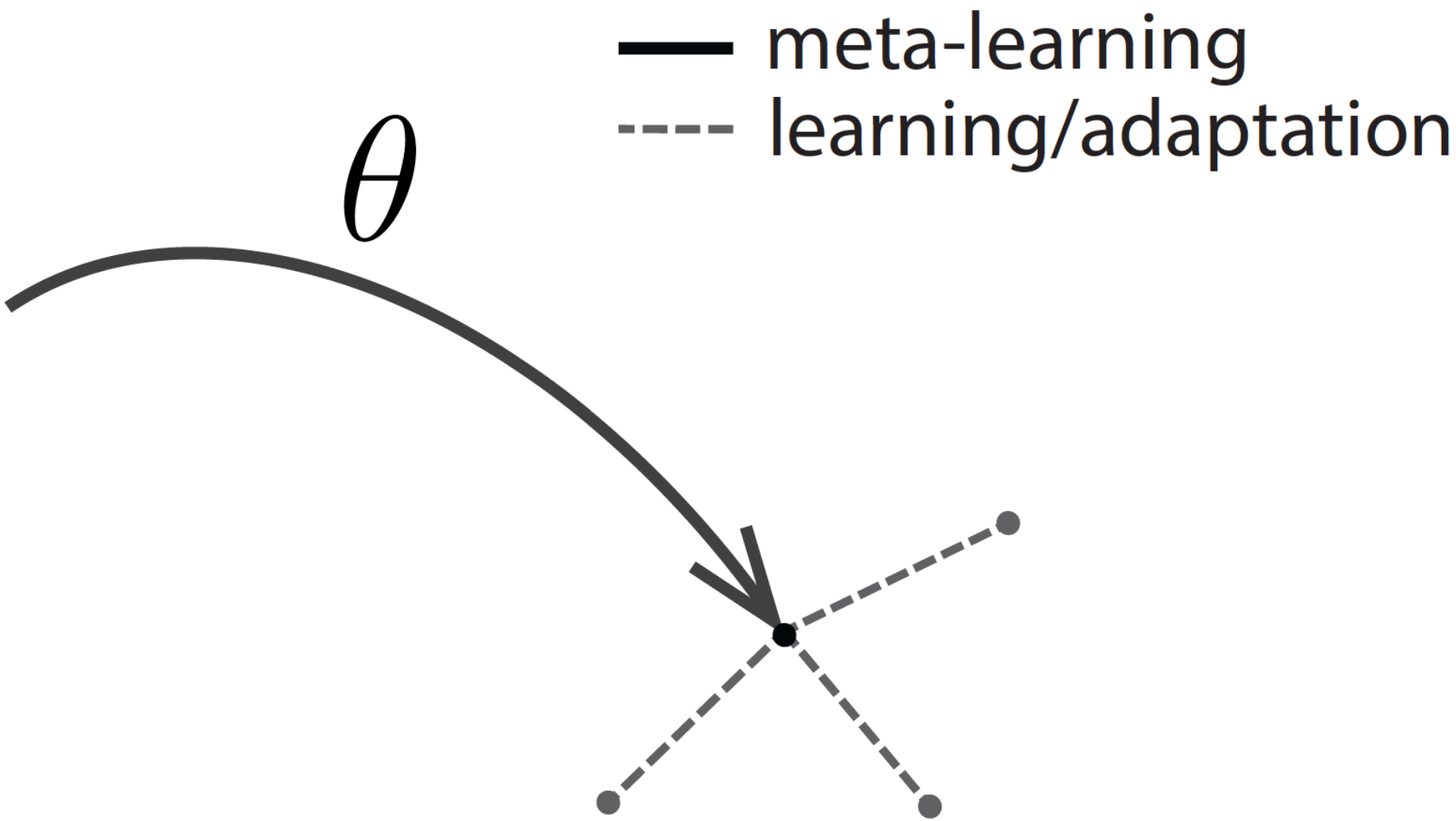


Figure: Model-Agnostic Meta-Learning

MAML

Listing 1 MAML

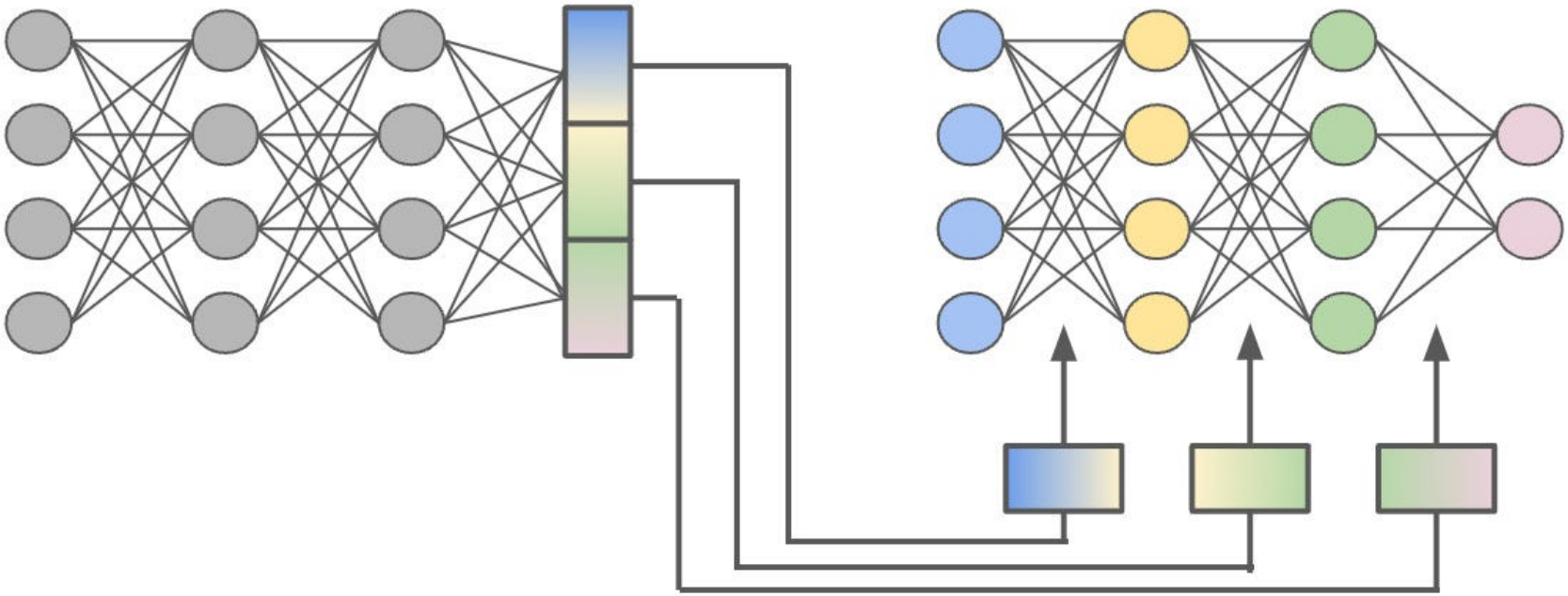
Require: $p(\mathcal{T})$: distribution over task space
Require: α, β: additional, fixed parameter

1: Sample θ
2: **while** not done **do**
3: Wylosuj batch $\mathcal{B} = \{\mathcal{T}_i\}_{i=1}$
4: **for each** $\mathcal{T}_i \in \mathcal{B}$ **do**
5: Compute $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ wrt$\mathcal{S}_i$
6: Compute adaptation (gradient descent) $\theta_i = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
7: $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i})$ (gradient descent) $\triangleright\ \theta = \{\theta^E, \theta^H\}$

Hypernetworks

Hypernetwork

Main network



HyperMAML

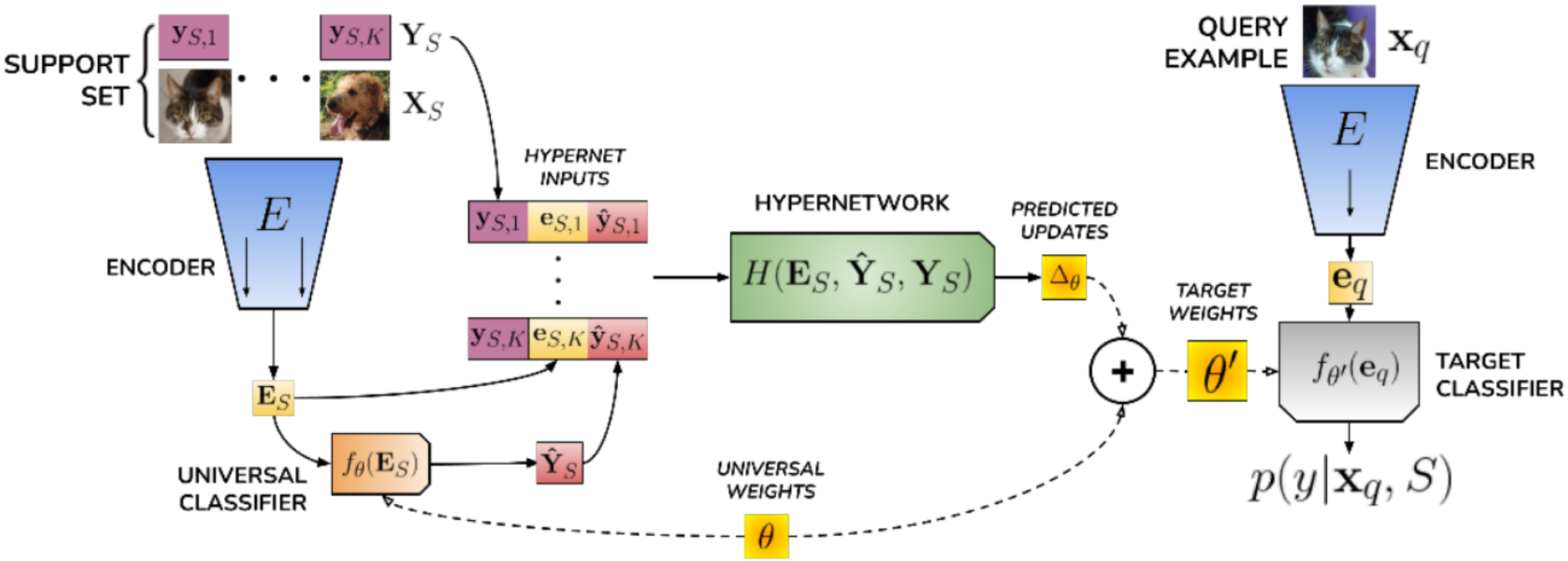


Figure: HyperMAML model architecture

HyperMAML

Listing 2 HyperMAML

Require: $p(\mathcal{T})$: probability distribution over task space
Require: β: additional parameter

Sample θ, η, γ
while not done **do**
    Sample batch $\mathcal{B} = \{\mathcal{T}_i\}_{i=1}$
    for each $\mathcal{T}_i \in \mathcal{B}$ **do**
        Compute adaptation $\theta_i = \theta - \mathbf{H}(\mathcal{S}_i) = \theta - \Delta\theta$
    $\mathcal{L}_{\text{HyperMAML}} := \sum_{T_i} \mathcal{L}_{Q_i}(f_{\theta_i})$
    $\theta \leftarrow \theta - \beta \nabla_\theta \mathcal{L}_{\text{HyperMAML}}(f_\theta)$ ▷ Global parameters θ
    $\eta \leftarrow \eta - \beta \nabla_\eta \mathcal{L}_{\text{HyperMAML}}(f_\theta)$ ▷ Hypernetwork H_η
    $\gamma \leftarrow \gamma - \beta \nabla_\gamma \mathcal{L}_{\text{HyperMAML}}(f_\theta)$ ▷ Extractor E_γ

Contents

1. Intro
2. Introduction to flow models
   - Likelihood function
3. Continuous normalizing flows
   - About optimizing with gradient of loss
   - About adjoint state
4. FFJORD
   - Complexity improvement
5. Meta-learning, FSL
   - Meta Learning
   - Few-shot learning
6. MAML
   - MAML
   - Hypernetworks
   - HyperMAML
7. Bayesian methods in ML
8. Bayesian HyperMAML

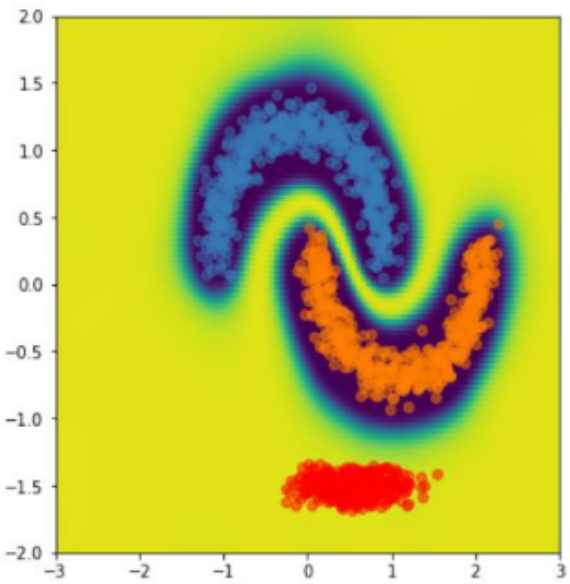
Uncertainty in neural networks

Typically the optimization process of neural networks is based on Maximum a posteriori (MAP) estimation

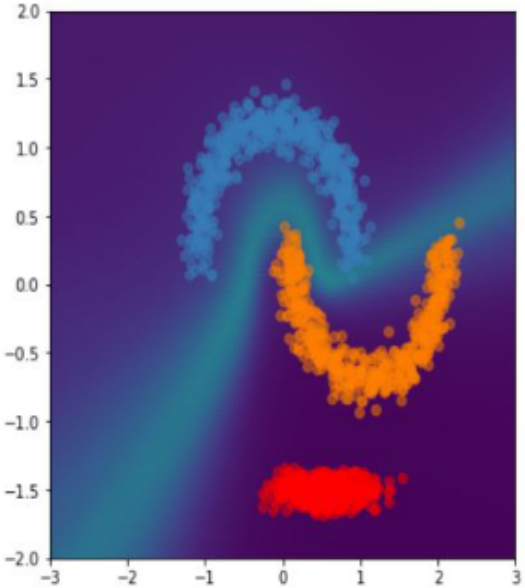
$$\theta^* = \arg\max_{\theta} p(\theta|x, y)$$

The problem with this approach is that we are given just one prediction per example.

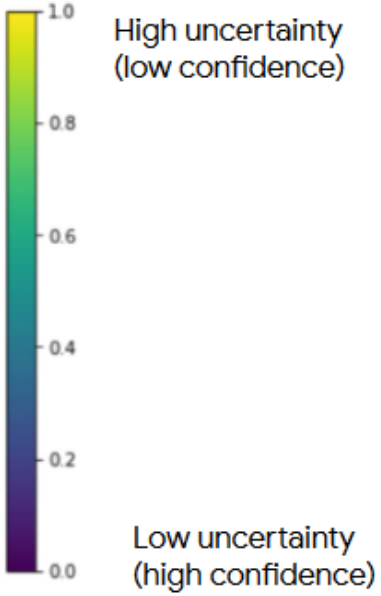
Problems with standard neural networks



Ideal behavior



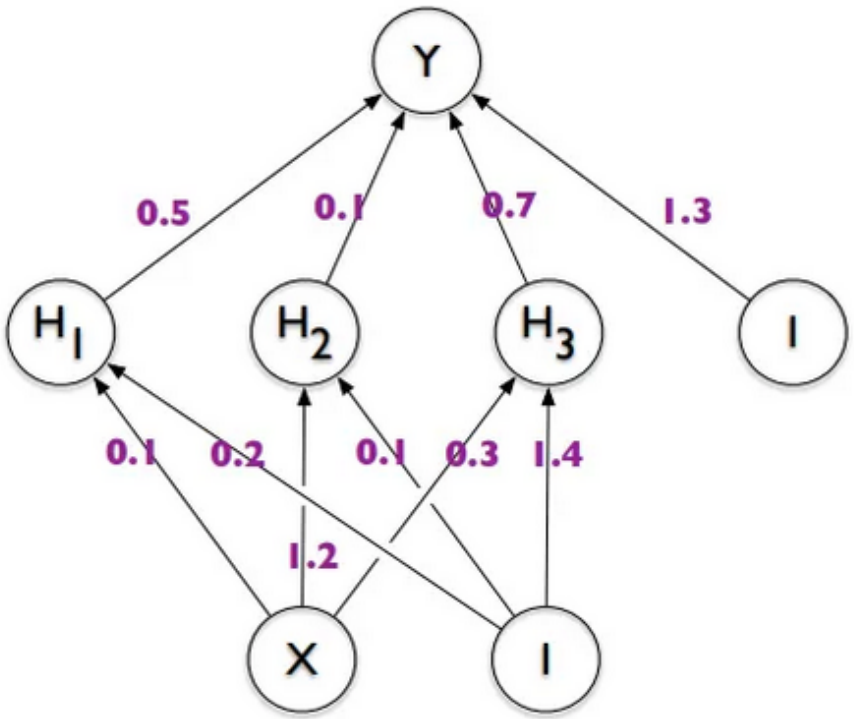
Deep neural networks

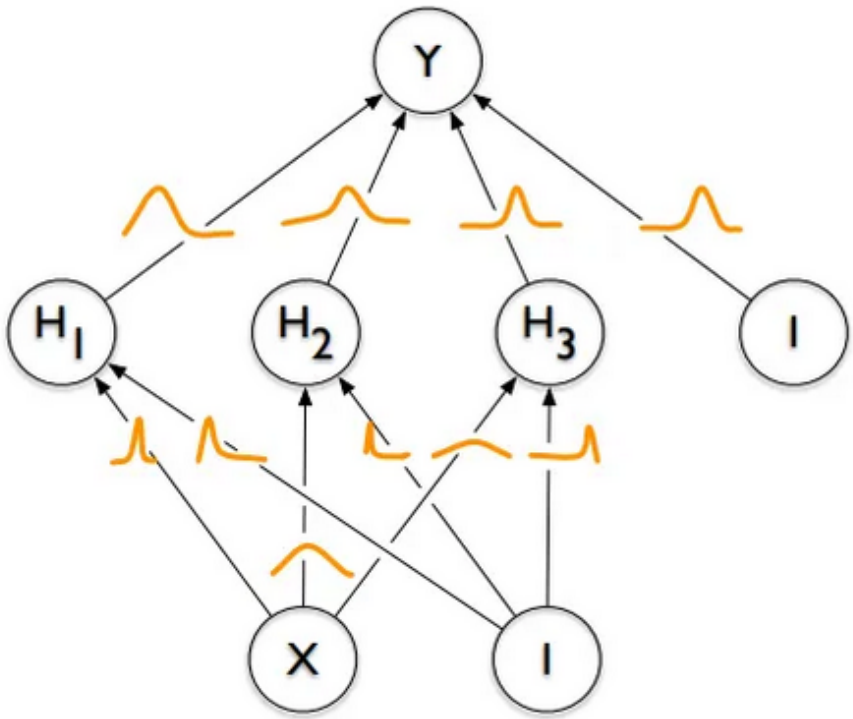


How to get uncertainty?

1. Ensembling - get many models and aggregate the answer from them.
2. Probabilistic approach - get full distribution of $p(\theta|D)$

Bayesian Neural Networks idea





Probabilistic approach

We are interested in calculating a posterior distribution of weight $p(\theta|D)$.
Bayes theorem tells us that:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} = \frac{p(D|\theta)p(\theta)}{\int_\Theta p(D|\theta')p(\theta')\,d\theta'}$$

, where $p(\theta)$ is a specified prior distribution
This approach allows us to make a probabilistic prediction:

$$p(\hat{y}(x)|D) = \int_\Theta p(\hat{y}(x)|\theta)p(\theta|D)\,d\theta = \mathbb{E}_{p(\theta|D)}[p(\hat{y}(x)|\theta)]$$

Problems

The integrals that come up in those equations are intractable in most cases. Consequently, we have to approximate the posterior distribution. There are two main approaches to deal with this:

1. Sampling methods - which approximately compute $p(\hat{y}(x)|D)$ by generating a finite set of network weights with the hope to approximate the real distribution
2. Variational inference - which instead model the posterior $p(\theta|D)$ using a parameterized distribution $q_\phi(\theta)$, called the approximate posterior. This distribution is iteratively improved by solving a suitable optimization problems. This optimization can be done by using standard optimization techniques, however the problem is specifying $q_\phi(\theta)$.

Sampling

- Monte carlo integration - in this approach we simply sample weights $\{\theta_i\}_{i=1}^N$ from prior distribution

$$p(D) = \mathbb{E}_{p(\theta)}[p(D|\theta)] \approx \frac{1}{N}\sum_{i=1}^{N} p(D|\theta_i)$$

Then given a good approximation for $p(D)$ we could use Monte Carlo again:

$$p(\hat{y}(x)|D) = \mathbb{E}_{p(\theta|D)}[p(\hat{y}(x)|\theta)] = \mathbb{E}_{p(\theta)}\left[p(\hat{y}(x)|\theta)\frac{p(D|\theta)}{p(D)}\right]$$

$$\approx \frac{1}{N}\sum_{i=1}^{N} p(\hat{y}(x)|\theta_i)\frac{p(D|\theta_i)}{p(D)}$$

Variational inference

In this approach we first define a parametrized and tractable stand-in distribution, called approximate distribution $q_\phi(\theta)$ and we try to tune the parameters ϕ, so that it better approximates the intractable distribution. We need to also formalize the notion of similarity between two probability distributions and writing down an optimization problem that corresponds to maximizing this similarity.

Kullback-Leibler divergence

For calculating the similiarity between two probability distributions we can use Kullback-Leibler divergence defined as:

$$KL(q_\phi(w)\|p(w|D)) := \int q_\phi(w) \log\left(\frac{q_\phi(w)}{p(w|D)}\right) dw$$
$$= \mathbb{E}_{q_\phi(w)}\left[\log q_\phi(w) - \log p(w|D)\right]$$

Evidence lower bound (ELBO)

$$
\begin{aligned}
KL(q_\phi(w)\|p(w|D)) &= \mathbb{E}_{q_\phi(w)}\left[\log q_\phi(w) - \log p(w|D)\right] \\
&= \mathbb{E}_{q_\phi(w)}\left[\log q_\phi(w)\right] - \mathbb{E}_{q_\phi(w)}\left[\log p(w|D)\right] \\
&= \mathbb{E}_{q_\phi(w)}\left[\log q_\phi(w)\right] - \mathbb{E}_{q_\phi(w)}\left[\log p(w, D) - \log p(D)\right] \\
&= \mathbb{E}_{q_\phi(w)}\left[\log q_\phi(w) - \log p(w, D)\right] + \underbrace{\log p(D)}_{\text{untractable}}
\end{aligned}
$$

We can't compute $p(D)$, however we can do something about it. Let's first rearrange:

$$
\mathbb{E}_{q_\phi(w)}\left[\log p(w, D) - \log q_\phi(w)\right] = \log p(D) - KL(q_\phi(w)\|p(w|D))
$$

By maximizing the quantity on the left we will be both maximizing $p(D)$ and minimizing the divergence between $q_\phi(w)$ and $p(w, D)$, which we wanted to do.

ELBO

We can define:

$$ELBO(q_\phi) := \mathbb{E}_{q_\phi(w)}\left[\log p(w, D) - \log q_\phi(w)\right]$$

We want to get a decomposition that is easy to manage:

$$\begin{aligned} ELBO(q_\phi) &= \mathbb{E}_{q_\phi(w)}\left[\log p(w, D)\right] - \mathbb{E}_{q_\phi(w)}\left[\log q_\phi(w)\right] \\ &= \mathbb{E}_{q_\phi(w)}\left[\log p(D|w)p(w)\right] - \mathbb{E}_{q_\phi(w)}\left[\log q_\phi(w)\right] \\ &= \mathbb{E}_{q_\phi(w)}\left[\log p(D|w)\right] + \mathbb{E}_{q_\phi(w)}\left[p(w) - q_\phi(w)\right] \\ &= \mathbb{E}_{q_\phi(w)}\left[\log p(D|w)\right] - KL(q_\phi(w)\|p(D)) \end{aligned}$$

We want to maximize this quantity. However it isn't that obvious how to optimize ϕ since the distribution depends on it.

Unbiased Monte Carlo gradients

Theorem

Let ϵ be a random variable having a probability density given by $q(\epsilon)$ and let $\mathrm{w} = t(\theta, \epsilon)$ where $t(\theta, \epsilon)$ is a deterministic function. Suppose further that the marginal probability density of w, $q(\mathrm{w}|\theta)$, is such that $q(\epsilon)\,d\epsilon = q(\mathrm{w}|\theta)\,d\mathrm{w}$. Then for a function f with derivatives in w:

$$\frac{\partial}{\partial\theta}\mathbb{E}_{q(\mathrm{w}|\theta)}[f(\mathrm{w},\theta)] = \mathbb{E}_{q(\epsilon)}\left[\frac{\partial f(\mathrm{w},\theta)}{\partial \mathrm{w}}\frac{\partial \mathrm{w}}{\partial\theta} + \frac{\partial f(\mathrm{w},\theta)}{\partial\theta}\right].$$

Proof

Proof.

$$\begin{aligned}
\frac{\partial}{\partial\theta}\mathbb{E}_{q(\mathrm{w}|\theta)}[f(\mathrm{w},\theta)] &= \frac{\partial}{\partial\theta}\int f(\mathrm{w},\theta)q(\mathrm{w}|\theta)\mathrm{d}\mathrm{w} \\
&= \frac{\partial}{\partial\theta}\int f(\mathrm{w},\theta)q(\epsilon)\mathrm{d}\epsilon \\
&= \int \frac{\partial f(\mathrm{w},\theta)}{\partial\theta}q(\epsilon)\mathrm{d}\epsilon \\
&= \mathbb{E}_{q(\epsilon)}\left[\frac{\partial f(\mathrm{w},\theta)}{\partial\mathrm{w}}\frac{\partial\mathrm{w}}{\partial\theta} + \frac{\partial f(\mathrm{w},\theta)}{\partial\theta}\right]
\end{aligned}$$

□

Reparametrization example $\mathcal{N}(\mu, \Sigma)$

1. Sample $\epsilon \sim \mathcal{N}(0, I)$.
2. Let $\mathrm{w} = \mu + \log(1 + \exp(\rho)) \circ \epsilon$.
3. Let $\theta = (\mu, \rho)$.
4. Calculate the gradient with respect to the mean

$$\Delta_\mu = \frac{\partial f(\mathrm{w}, \theta)}{\partial \mathrm{w}} + \frac{\partial f(\mathrm{w}, \theta)}{\partial \mu}.$$

5. Calculate the gradient with respect to the standard deviation parameter ρ

$$\Delta_\rho = \frac{\partial f(\mathrm{w}, \theta)}{\partial \mathrm{w}} \frac{\epsilon}{1 + \exp(-\rho)} + \frac{\partial f(\mathrm{w}, \theta)}{\partial \rho}.$$

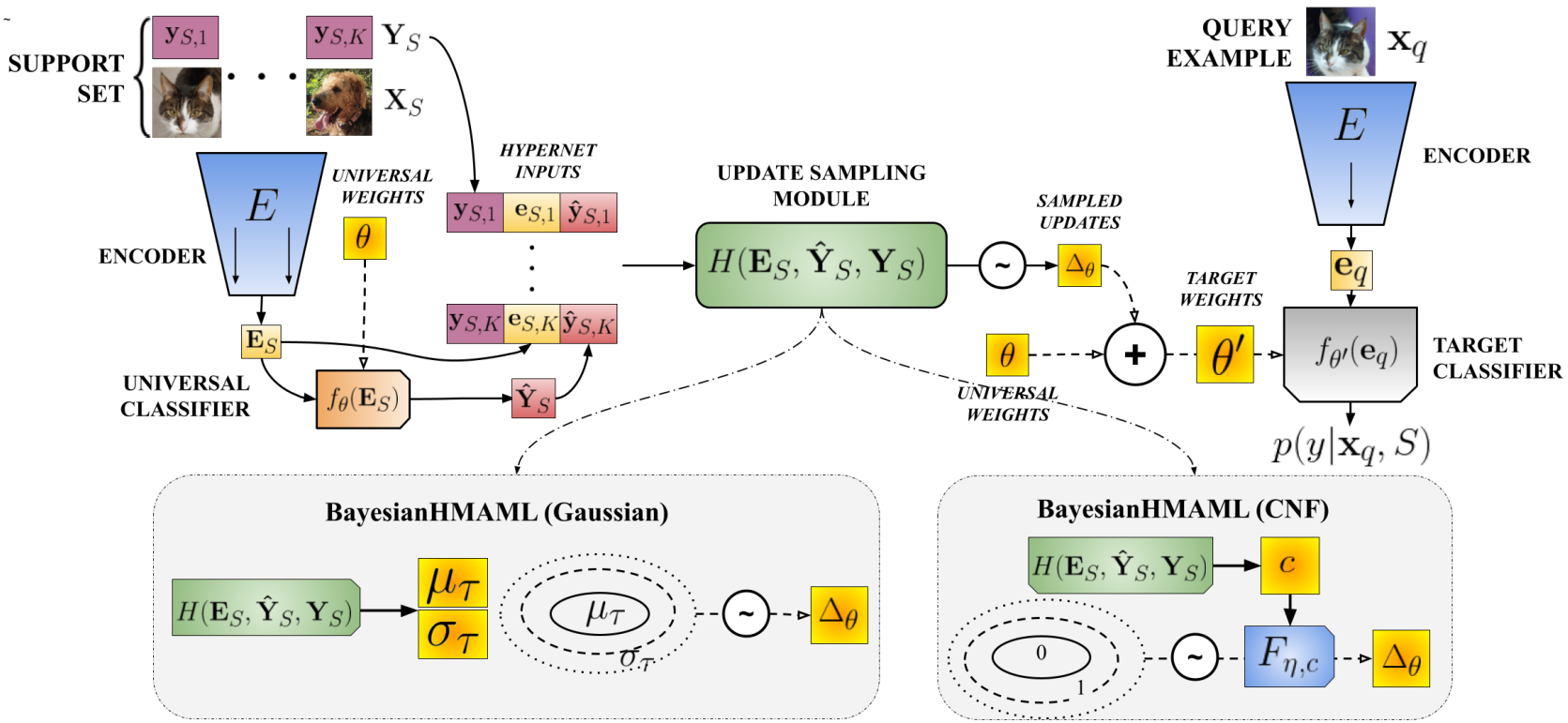
6. Update the variational parameters:

$$\mu \leftarrow \mu - \alpha \Delta_\mu$$
$$\rho \leftarrow \rho - \alpha \Delta_\rho.$$

Contents

1. Intro
2. Introduction to flow models
   - Likelihood function
3. Continuous normalizing flows
   - About optimizing with gradient of loss
   - About adjoint state
4. FFJORD
   - Complexity improvement
5. Meta-learning, FSL
   - Meta Learning
   - Few-shot learning
6. MAML
   - MAML
   - Hypernetworks
   - HyperMAML
7. Bayesian methods in ML
8. Bayesian HyperMAML

BHyperMAML


SUPPORT SET
$\mathbf{y}_{S,1}$
$\mathbf{y}_{S,K}$
$\mathbf{Y}_S$
$\mathbf{X}_S$
ENCODER
E
UNIVERSAL WEIGHTS
θ
$\mathbf{E}_S$
UNIVERSAL CLASSIFIER
$f_\theta(\mathbf{E}_S)$
$\hat{\mathbf{Y}}_S$
HYPERNET INPUTS
$\mathbf{y}_{S,1}$ $\mathbf{e}_{S,1}$ $\hat{\mathbf{y}}_{S,1}$
$\mathbf{y}_{S,K}$ $\mathbf{e}_{S,K}$ $\hat{\mathbf{y}}_{S,K}$
UPDATE SAMPLING MODULE
$H(\mathbf{E}_S, \hat{\mathbf{Y}}_S, \mathbf{Y}_S)$
SAMPLED UPDATES
Δ_θ
UNIVERSAL WEIGHTS
θ
TARGET WEIGHTS
θ'
QUERY EXAMPLE
$\mathbf{x}_q$
ENCODER
E
$\mathbf{e}_q$
$f_{\theta'}(\mathbf{e}_q)$
TARGET CLASSIFIER
$p(y|\mathbf{x}_q, S)$
BayesianHMAML (Gaussian)
$H(\mathbf{E}_S, \hat{\mathbf{Y}}_S, \mathbf{Y}_S)$
μ_τ
σ_τ
Δ_θ
BayesianHMAML (CNF)
$H(\mathbf{E}_S, \hat{\mathbf{Y}}_S, \mathbf{Y}_S)$
c
0
1
$F_{\eta,c}$
Δ_θ

BayesianMAML

In BayesianMAML the objective is given as:

$$\mathcal{L}_{\mathcal{D}} = \mathbb{E}_{q(\theta|\psi)}\underbrace{\left[\sum_{i}^{N} \mathbb{E}_{q(\theta_i'|\lambda_i)}\left[\log p(\mathcal{T}_i|\theta_i') - KL\left(q(\theta_i'|\lambda_i)\|p(\theta_i'|\theta)\right)\right]\right]}_{\mathcal{L}_{\mathcal{T}_i}} - KL(q(\theta|\psi)\|p(\theta))$$

where $q(\theta_i'|\lambda_i)$ and $q(\theta|\psi)$ are respectively per-task posterior approximation and approximate posterior for the universal weights. They are tied together by the prior $p(\theta_i'|\theta)$.

BHyperMAML

The learning objective takes the following form:

$$\mathcal{L}_D^{our} = \sum_i^N \mathbb{E}_{q(\theta_i'|\lambda_i(\theta,\mathcal{S}_i))} \left[\log p(\mathcal{T}_i|\theta_i') - \gamma \cdot KL\left(q(\theta_i'|\theta, \lambda_i(\theta, \mathcal{S}_i))\|p(\theta_i')\right)\right],$$

where we used the standard normal priors for the weights of the neural network f, i.e., $p(\theta_i') = \mathcal{N}(\theta_i'|0, \mathbb{I})$
In this case the adapted parameters follow the distribution:

$$\theta_i' \sim q(\theta|\lambda_i(\theta, S_i)).$$

Moreover the parameters $\lambda_i(\theta, S_i)$ will be the output of the hypernetwork

$$\lambda_i(\theta, S_i) \coloneqq H_\phi(\theta, S_i)$$

BHyperMAML(Gaussian)

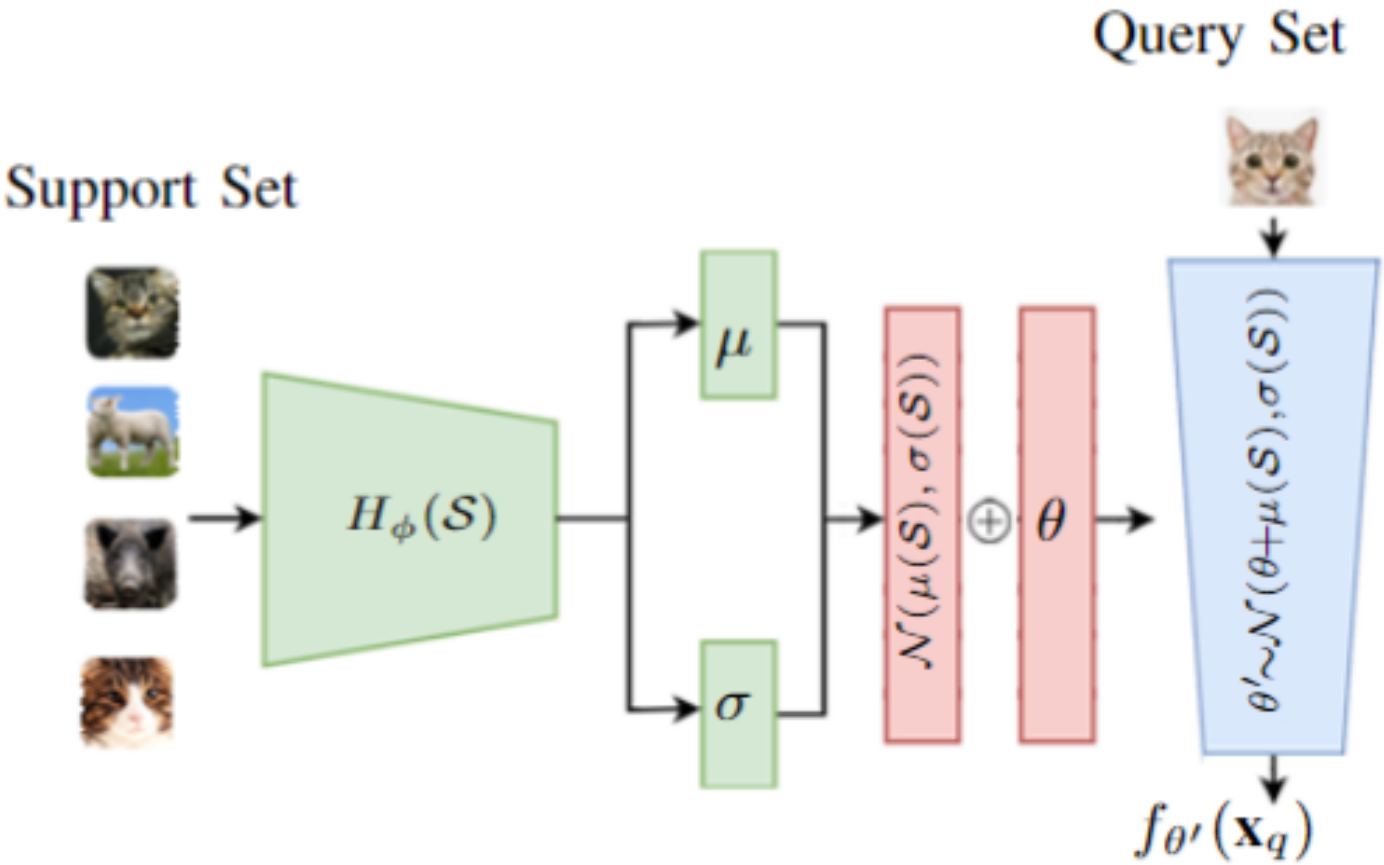
This apporach is a very simple realization of the algorithm. In this case the output of the hypernetwork are simply the parameters of the distribution

$$(\mu_\theta(\mathcal{S}_i), \sigma_\theta(\mathcal{S}_i)) := H_\phi(\mathcal{S}_i, \theta)$$

And the weights are sampled in the following way

$$\theta_i' \sim \mathcal{N}(\theta + \mu_\theta(\mathcal{S}_i), \sigma_\theta(\mathcal{S}_i))$$

BHyperMAML(Gaussian)



BHyperMAML(CNF)

In this version the weights distribution is not described by any parametric distribution. In this case the output of the hypernetwork is instead responsible for conditioning the Continuous Normalizing Flow $F_{\eta, C(\theta, \mathcal{S}_i)}(\cdot)$: $C(\theta, \mathcal{S}_i) := H_\phi(\mathcal{S}_i, \theta)$.
The posterior distribution can be then obtained by a two stage process $\Delta\theta'_i \sim F_{\eta, C(\theta, \mathcal{S}_i)}$ and $\theta'_i = \theta + \Delta\theta'_i$,
And the sampling process is also a two stage process
$\Delta\theta'_i := F_{\eta, C(\theta, \mathcal{S}_i)}(z)$, where $z \sim \mathrm{N}(0, t \cdot \mathbb{I})$,

DKL

The Kulback-Leibler divergence can be written in closed form in both cases. For the gaussian version:

$$KL(\mu, \Sigma) = \frac{1}{2}\left[\mu^T\mu + \mathrm{tr}\Sigma - k - \log\det\Sigma\right]$$

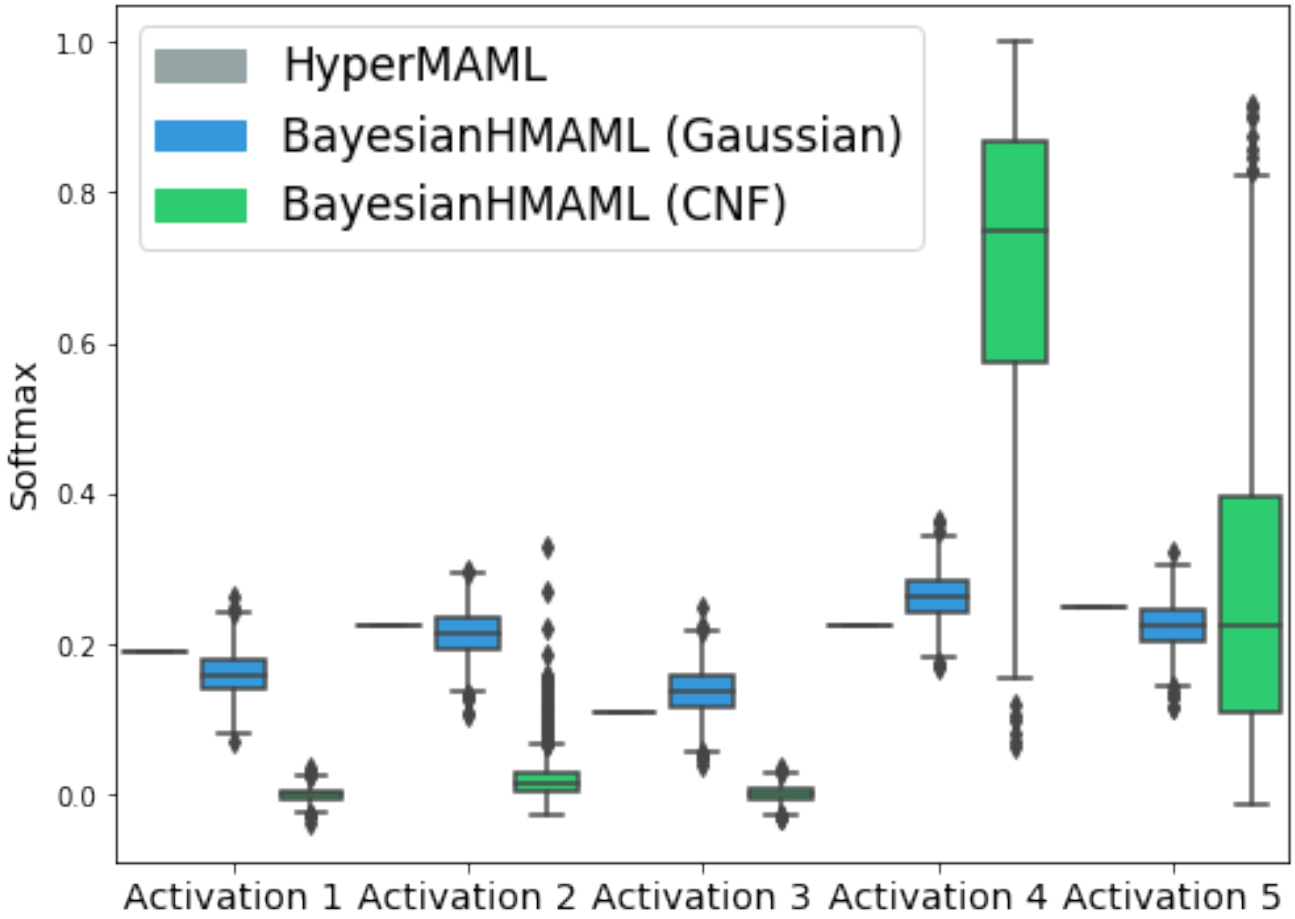
And in the case of CNF

$$KL(\cdot) = \frac{1}{P}\sum_{z\sim\mathcal{N}(0,t\cdot\mathbb{I})}\left(\log\mathcal{N}\left(F^{-1}_{\eta,C}(\Delta\theta'_i|0, t\cdot\mathbb{I}\right) + \log\det|J| - \log\mathcal{N}(\theta'_i|0,\mathbb{I})\right)$$

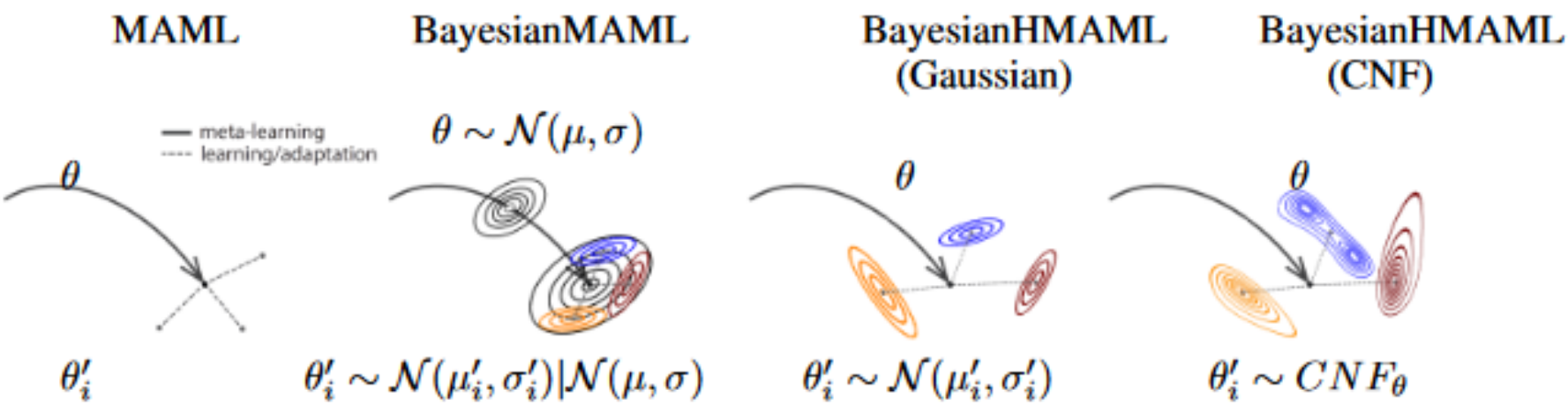
Results

Method	Omniglot→EMNIST 1-shot	Omniglot→EMNIST 5-shot	mini-ImageNet→CUB 1-shot	mini-ImageNet→CUB 5-shot
Feature Transfer	64.22 ± 1.24	86.10 ± 0.84	32.77 ± 0.35	50.34 ± 0.27
ProtoNet	72.04 ± 0.82	87.22 ± 1.01	33.27 ± 1.09	52.16 ± 0.17
MAML	74.81 ± 0.25	83.54 ± 1.79	34.01 ± 1.25	48.83 ± 0.62
DKT	75.40 ± 1.10	90.30 ± 0.49	40.14 ± 0.18	*56.40 ± 1.34*
Bayesian MAML	63.94 ± 0.47	65.26 ± 0.30	33.52 ± 0.36	51.35 ± 0.16
HyperMAML	79.07 ± 1.09	89.22 ± 0.78	36.32 ± 0.61	49.43 ± 0.14
BHyperMAML (G)	*80.95 ± 0.46*	89.21 ± 0.27	36.90 ± 0.34	49.24 ± 0.38
BHyperMAML (G)+ada.	81.05 ± 0.47	*89.76 ± 0.26*	37.23 ± 0.44	50.79 ± 0.59
BHyperMAML (CNF)	72.02 ± 0.56	82.36 ± 0.12	33.77 ± 0.30	44.09 ± 0.32
BHyperMAML (CNF)+ada.	72.54 ± 0.36	82.63 ± 0.37	34.67 ± 0.35	45.14 ± 0.27

Uncertainty



Comparison


MAML
BayesianMAML
BayesianHMAML (Gaussian)
BayesianHMAML (CNF)
meta-learning
learning/adaptation
θ
$\theta \sim \mathcal{N}(\mu, \sigma)$
θ
θ
θ'_i
$\theta'_i \sim \mathcal{N}(\mu'_i, \sigma'_i)|\mathcal{N}(\mu, \sigma)$
$\theta'_i \sim \mathcal{N}(\mu'_i, \sigma'_i)$
$\theta'_i \sim CNF_\theta$