Uniwersytet Jagielloński
INSTYTUT INFORMATYKI I MATEMATYKI KOMPUTEROWEJ
Studia dzienne
Kraków, 2023

# Few shot learning i algorytmy regularyzacyjne na przykładzie algorytmu HyperMAML i metodach BayesHMAML oraz FHMAML

**Piotr Kubacki**[⋆†] **Piotr Borycki**[†]
1173028 1172662

praca powstała przy współpracy z Katedrą Nauczania Maszynowego (GMUM)

[†]praca powstała w trakcie uczestnictwa w programie tutoringowym w ramach programu Inicjatywa Doskonałości w Uniwersytecie Jagiellońskim

[⋆]praca powstała w trakcie uczestnictwa w programie stypendialnym w ramach projektu TEAM-NET Fundacji na rzecz Nauki Polskiej POIR.04.04.00-00-14DE/18-00: Sztuczne sieci neuronowe inspirowane biologicznie

Opiekun pracy licencjackiej:
dr hab. Przemysław Spurek, UJ

# 1 Abstrakt

Większość aktualnych technik stosowanych w nauczaniu maszynowym nie potrafi się dobrze generalizować z małej ilości danych. W szczególności trening głębokich sieci neuronowych wymaga dużej ilości informacji, których nie zawsze jesteśmy w stanie uzyskać. Potrafią natomiast rozwiązywać niezwykle skomplikowane problemy, jeśli mają do nich dostęp. Przykładem takiej sytuacji jest chociażby chatbot ChatGPT. W odróżnieniu od wspomnianych głębokich sieci neuronowych, człowiek jest w stanie dobrze uczyć się z małej ilości danych. Na przykład, kiedy człowiek zobaczy przedmiot, którego nigdy wcześniej nie widział, będzie w stanie rozpoznać go w przyszłości. Próbą rozwiązania wspomnianego problemu w przypadku głębokich sieci neuronowych jest few-shot learning, gdzie staramy się uczyć sieć neuronową radzić sobie w sytuacji, kiedy otrzyma ona jedynie kilka przykładów. W pracy zostaną przedstawione autorskie algorytmy, rozwiązujące problem few-shot learningu, sieci bayesowskie oraz strukturę hypernetworków.

# 2 Preliminaria

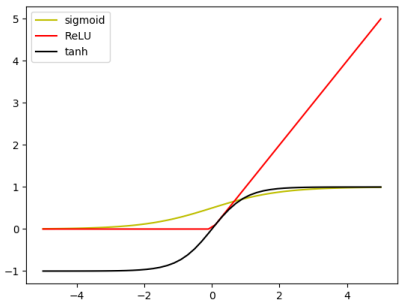
W tej pracy będziemy używać zamiennie nazw pojęć i ich odpowiedników w języku angielskim. Umożliwi to większą naturalność wypowiedzi oraz zapozna czytelnika ze stosowanym żargonem. Na początku wprowadzimy kilka pojęć wykorzystywanych w dalszej części pracy, zaczynając od sztucznej sieci neuronowej.

**Sieć neuronowa** Sztuczna sieć neuronowa jest nałożeniem na siebie dużej ilości prostych odwzorowań, których obliczenie zajmuje niewiele czasu. Jednym z najbardziej naturalnych wyborów są odwzorowania afiniczne. Odwzorowania tego typu są problematyczne, ze względu na to, że złożenie odwzorowań afinicznych jest afiniczne. W celu rozwiązania tego problemu wprowadza się tzw. **funkcję aktywacji**. Są to nieliniowe funkcje postaci $g : \mathbb{R} \to \mathbb{R}$.

Najczęstszymi funkcjami aktywacji są:

- $\sigma \colon \mathbb{R} \ni x \mapsto \frac{1}{1+e^{-x}} \in \mathbb{R}$
- $\tanh \colon \mathbb{R} \ni x \mapsto \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \in \mathbb{R}$
- $\mathrm{ReLU} \colon \mathbb{R} \ni x \mapsto \max(0, x) \in \mathbb{R}$

Na rysunku przedstawione są wykresy wyżej wymienionych funkcji.




Za pomocą odwzorowania afinicznego oraz funkcji aktywacji jesteśmy w stanie zdefiniować najbardziej podstawowy komponent sieci neuronowej, a dokładniej sztuczny neuron.
**Neuronem** nazywamy funkcję:

$$f \colon \mathbb{R}^d \ni x \mapsto g(\langle w, x\rangle + b) \in \mathbb{R}, \text{ gdzie: } w \in \mathbb{R}^d,\ b \in \mathbb{R},\ g - \text{funkcja aktywacji}$$

Zestawiając ze sobą $n$ neuronów jesteśmy w stanie zdefiniować funkcję:

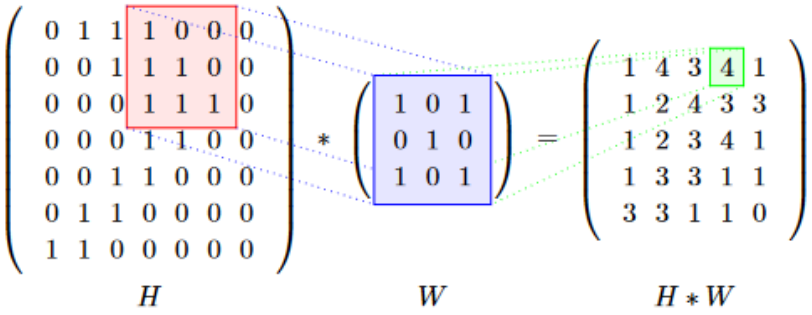
$$F_i \colon \mathbb{R}^d \ni x_{i-1} \mapsto \hat{g}(W_i^{\mathbf{T}} x_{i-1} + b_i) \in \mathbb{R}^n$$

gdzie: $W_i \in \mathbb{R}^{d\times n}$, $b_i \in \mathbb{R}^n$, $\hat{g}$- funkcja aktywacji $g$ aplikowana na każdej współrzędnej.
Funkcję $F_i$ nazywamy **warstwą fully-connected**. Nazywana jest ona w ten sposób, ponieważ na jej wejściu znajduje się wyjście z poprzedniej warstwy, więc wszystkie neurony danej warstwy wpływają na wszystkie neurony następnej. Ostatecznie **sieć neuronową** definiujemy jako:

$$\Phi \colon \mathbb{R}^k \ni x \mapsto F_o \circ F_i \circ \cdots \circ F_1(x) \in \mathbb{R}^m$$

gdzie: $F_o(x) = W_o^{\mathbf{T}} x + b_o \in \mathbb{R}^m$ jako ostatnio warstwa sieci jest pozbawiona funkcji aktywacji
Oprócz sieci *fully-connected* istnieją także **sieci konwolucyjne** służace głównie do przetwarzania obrazów. Są to sieci, których warstwy nazywane są konwolucyjnymi, definiują zestawy filtrów modyfikujących obraz poprzez dokonanie operacji konwolucji. Każdy z filtrów jest funkcją działającą na otoczeniu danego piksela i zwraca pojedyńczą wartość rzeczywistą. W przypadku sieci

konwolucyjnych parametrami są właśnie wartości konkretnych filtrów. Na rysunku 1 przedstawione jest przykładowe działanie warstwy konwolucyjnej.




Rys. 1: Do macierzy reprezentującej obraz $H$ w każdym pikselu przykładany jest filtr $W$ (zaznaczony niebieskim kolorem). Następnie wykonujemy operację konwolucji pomiędzy filtrem i częścią obrazka pod filtrem (zaznaczoną na czerwono). Wynik konwolucji jest wpisywany do obrazka wynikowego w miejsce środkowego elementu. [TSS+22]

Za pomocą takiej sieci możemy znacząco zmniejszyć rozmiar obrazka i ostatecznie, spłaszczając go, przetwarzać dalej, korzystając z warstw *fully-connected*. Musimy jednak znaleźć metodę znajdowania optymalnych parametrów sieci. W tym celu zdefiniujemy funkcję kosztu, za pomocą której będziemy w stanie ocenić jakość działania naszego modelu.

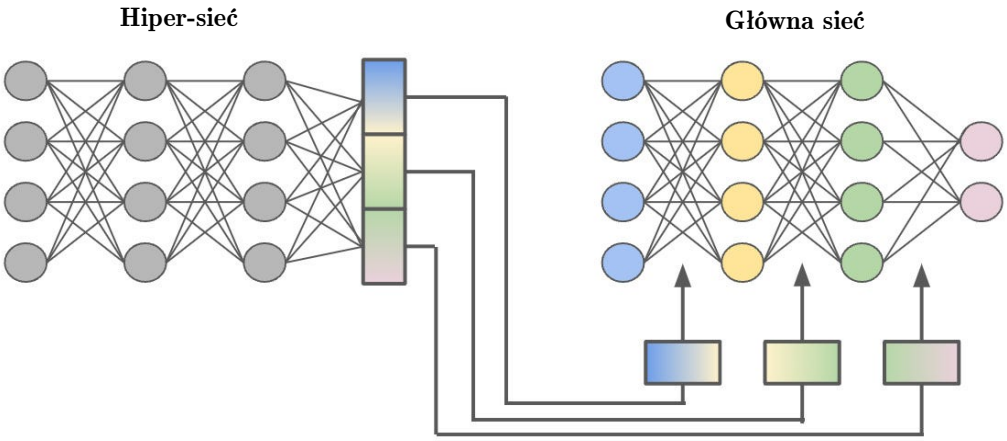
**Funkcja kosztu** Klasycznie parametry można dobrać, maksymalizując funkcję wiarygodności (dodatkowo biorąc logarytm, co ułatwia obliczenia nie zmieniając asymptotyki). Za pomocą tej metody otrzymujemy funkcję $l$ nazywaną *log likelihood*. Funkcję kosztu możemy wtedy zapisać jako $L = -l$. Możemy więc równoważnie, zamiast maksymalizować funkcję wiarygodności, minimalizować funkcję kosztu, do czego wykorzystać możemy metodę spadku gradientu. Oznaczając przez $\theta$ parametry modelu, które chcemy estymować, rozważmy oczekiwaną funkcję kosztu:

$$L(\theta) = -\mathbb{E}_{(X,Y)\sim P}(\log(p_\theta(Y|X))$$

W przypadku klasyfikacji $k$ klas, wzór ten można przedstawić jako funkcję nazywaną *cross-entropią*:

$$L(\theta) = -\mathbb{E}_{(X,Y)\sim P}\left(\sum_{i=1}^{k} \mathbf{1}_{Y=i} \log(p_\theta(Y = i|X))\right)$$

**Hiper-sieci** Szczególnym przykładem sieci neuronowych są tzw. '*hiper-sieci* (ang. hypernetwork). Ich zadanie jest odmienne od wcześniej wspomnianych sieci neuronowych, ponieważ ich celem jest generowanie parametrów dla innej sieci. Na rysunku 2 przedstawione jest działanie takiej sieci. Hiper-sieci definiują abstrakcję, która przypomina tę, którą możemy obserwować w świecie biologicznym. Autorzy wskazują na analogię w relacji genotyp-fenotyp oraz hipersieć-sieć główna, do której dostarczane są wagi [HDL16]. Zazwyczaj hiper-sieci są znacznie mniejszymi sieciami, niż te dla których wagi generują. Mogą być one także wykorzystywane na inne sposoby, np. w *few-shot learningu*. Przykład takiego zastosowania będzie przedstawiony w dalszej części pracy. Będą nam do tego potrzebne także bayesowskie sieci neuronowe.




Rys. 2: Hiper-sieć (po lewej) generuje wagi dla docelowej sieci (po prawej) [Wol]

**Bayesowskie sieci neuronowe** W bayesowskich sieciach neuronowych zamiast wag punktowych, wagi modelowane są za pomocą rozkładów prawdopodobieństwa. Wykorzystać do tego możemy twierdzenie Bayesa:

$$P(w|D) = \frac{P(w)P(D|w)}{\int_w P(D|w)P(w)}$$

gdzie: $w$ - wagi modelu, $D$ - zbiór danych

Z tym klasycznym podejściem wiąże się niestety problem obliczeniowy spowodowany dużą ilością wag sieci neuronowych. W związku z tym, zamiast obliczenia $P(w|D)$, staramy się wyestymować go wykorzystując tzn. *variational inference*. W metodzie tej dodajemy drugi posterior $q(w|\theta)$. Jest on rozkładem prawdopodobieństwa parametryzowanym za pomocą $\theta$. Do znalezienia $\theta$ możemy wykorzystać miarę **KL**. Jest to miara stosowana do badania rozbieżności między dwoma rozkładami prawdopodobieństwa, dana wzorem [Bis06]:

$$\mathbf{KL}(p,q) := \sum_i p(i) \log \frac{p(i)}{q(i)} \quad \text{dla rozkładów dyskretnych} \tag{1}$$

$$\mathbf{KL}(p,q) := \int p(i) \log \frac{p(i)}{q(i)}\, dx \quad \text{dla rozkładów ciągłych} \tag{2}$$

gdzie: $p$, $q$ - porównywane rozkłady

Minimalizując **KL** pomiędzy $q(w|\theta)$ oraz $P(w|D)$, jesteśmy w stanie otrzymać $\theta^*$, co z kolei daje nam $q(w|\theta^*)$, jako najlepszą estymację $P(w|D)$:

$$\begin{aligned}
\theta^* &= \arg\min_\theta \mathbf{KL}[q(w|\theta)\|P(w|D)] \\
&= \arg\min_\theta \int q(w|\theta) \log \frac{q(w|\theta}{P(w)P(D|w)}\, dw \\
&= \arg\min_\theta \mathbf{KL}[q(w|\theta)\|P(w)] - \mathbb{E}_{q(w|\theta)}[\log P(D|w)]
\end{aligned}$$

Ostatnia równość jest znana jako *variational free energy* lub *evidence-based lower bound* (ELBO). Otrzymujemy funkcję kosztu:

$$L(D,\theta) = \mathbf{KL}[q(w|\theta)\|P(w)] - \mathbb{E}_{q(w|\theta)}[\log P(D|w)]$$

Funkcja ta równa jest **KL** + **CE**, gdzie **CE** to entropia krzyżowa.

# 3 Wprowadzenie

## 3.1 Meta- learning

Meta- learning to dziedzina nauczania maszynowego dotycząca metod, które w celu wyuczenia modelu stosują pewne algorytmy uczące do metadanych, wynikających z eksperymentów nauczania maszynowego prowadzonych na wyjściowym modelu. Głównym wykorzystaniem metadanych

jest uelastycznienie sieci neuronowej na rozwiązywanie problemów uczenia, w celu usprawnienia obecnych algorytmów lub zbudowania na ich bazie nowych algorytmów. Powszechnie mówi się, że meta-learning to zbiór metod, które służą do uczenia się procesu uczenia (ang. learning to learn). [SS10]
Inna intuicja przydatna dla zrozumienia tego konceptu jest następująca: algorytmy Meta- learningu to takie, które przeprowadzają uczenie na metadanych. Optymalizujemy funkcję, która służy do przewidywania, jak najlepiej stosować algorytmy uczące na danych treningowych.

## 3.2 Few- shot learning

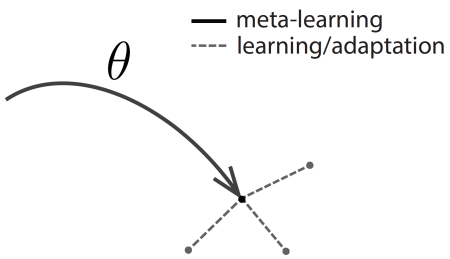
Jednym z alternatywnych rozwiązań problemu braku danych jest **Few- shot learning** (FSL). Jest to zbiór algorytmów nauczania maszynowego, w których proces treningu jest przeprowadzony na ograniczonej ilości danych. FSL jest postrzegany jako problem z dziedziny *meta- learningu*. Głównym zadaniem takich algorytmów jest wyuczenie modelu, który będzie w stanie dostosować się do pewnej domeny pracy, jedynie za pomocą kilku etykietowanych przykładów z tej domeny. Opisywanie zagadnienia *meta* i *few-shot learningu* może różnić się w zależności od kontekstu. Zdefiniujemy teraz kilka najważniejszych pojęć, które są poprawne w kontekście poruszanych w tej pracy problemów i niezbędne do ich dalszego zrozumienia.
Niech $\mathcal{S} = \{(\mathbf{x}_l, \mathbf{y}_l)\}_{l=1}^{L}$ będzie zbiorem supportu, który składa się z par wejście- wyjście, gdzie L oznacza moc zbioru, w którym każda klasa ma równy udział. Niech $L = k \cdot n$ oraz $k$ - liczba klas, $n$ - liczba przykładów na klasę w zbiorze $\mathcal{S}$. Mówimy w takim wypadku o zagadnieniu $n$-way $k$-shot. Niech $\mathcal{Q} = \{(\mathbf{x}_m, \mathbf{y}_m)\}_{m=1}^{M}$ oznacza zbiór query, gdzie $M \geq k$. Dla naszych potrzeb zakładamy, że zbiór etykiet elementów zbioru $\mathcal{S}$ jest równy zbiorowi etykiet elementów zbioru $\mathcal{Q}$ (tzn. zakres klas zbioru $\mathcal{S}$ jest równy zakresowi klas zbioru $\mathcal{Q}$, ale ilość etykiet danych klas przypisywana do poszczególnych elementów zbiorów może być różna). Formalnie rozważamy zadanie (ang. task) $\mathcal{T} = \{\mathcal{S}, \mathcal{Q}\}$.
Rozróżniamy dwa etapy: treningu i inferencji. W czasie treningu, modele FSL otrzymują na wejście losowo dobrane $\mathcal{D} = \{\mathcal{T}_n\}_{n=1}^{N}$. Zarówno zbiór supportu jak query w czasie treningu posiadają etykiety.
W czasie inferencji, rozważamy $\mathcal{T}_* = \{\mathcal{S}_*, \mathcal{X}_*\}$, taki że:

- $\mathcal{S}_*$ to zbiór supportu, w którym podane są etykiety klas (które stanowią domenę tasku $\mathcal{T}_*$).
- $\mathcal{X}_*$ to zbiór query taki, że elementy tego zbioru nie posiadają jawnej etykiety.

Celem modelu jest przewidzieć etykiety elementów zbioru query po uprzednim przetworzeniu (zakładając obecność w inferencji) zbioru supportu $\mathcal{S}_*$ oraz używając architektury wytrenowanej na $\mathcal{D}$.

## 3.3 MAML

Wśród algorytmów FSL, jednym z najpopularniejszych jest **Model- Agnostic Meta- Learning** (MAML). Jednym z założeń algorytmu jest agnostyczność modelu (ang. model- agnostic), co oznacza, że jest on kompatybilny z każdą metodą, która stosuje metodę spadku gradientu (ang. gradient descent), niezależnie od jej przeznaczenia oraz zakresu zadań. W tej pracy zajmujemy się problemem klasyfikacji, chociaż stosowalność modelu MAML jest znacznie szersza i obejmuje zagadnienia takie jak regresja, czy uczenie przez wzmacnianie (ang. reinforcement learning). [FAL17]




Rys. 3: Wizualizacja algorytmu MAML, który pod wpływem procesu meta- learningu optymalizuje parametry globalne sieci $\theta$ w celu łatwego dostosowania do nowych tasków [BKP+22]

Wprowadzimy teraz niezbędne założenia metody MAML. Niech $E$ będzie ekstraktorem cech (ang. feature extractor), który ma parametry $\theta^E$ oraz niech $H$ będzie klasyfikatorem o parametrach $\theta^H$. Rozważmy model sieci neuronowej z parametrami $\theta = (\theta^E, \theta^H)$, który reprezentuje odwzorowanie $f_\theta$. Będziemy rozważać zadanie przetwarzania i klasyfikacji obrazków. Podstawowym elementem większości architektur poświęconych przetwarzaniu obrazów jest ekstraktor cech. Jest to moduł, który pozwala na redukcję informacji potrzebnych do reprezentacji danych. Na potrzeby naszych rozważań możemy przyjąć, że $E$ to sieć neuronowa, która przekształca wysokowymiarowy obiekt w pewną zredukowaną reprezentację wektorową w $\mathbb{R}^n$. Dla naszego przypadku będzie to sieć konwolucyjna (np. ResNet [HZRS15]). Taką reprezentację bedziemy nazywać **embeddingiem**. Mówimy, że embedding to zanurzenie elementu z przestrzeni cech (ang. feature space) na przestrzeń ukrytą (ang. latent space). Klasyfikator potraktujmy jako sieć neuronową, która mapuje embedding na zbiór $K = \{0, 1, ..., k\}$, który nazwiemy zbiorem etykiet. Dla naszych potrzeb to oznacza, że $H$ bierze pewną reprezentację obrazka i zamienia ją na pewną etykietę, np. reprezentację obrazka, który przedstawia słonia na etykietę *słoń*. Omówimy teraz dokładniej algorytm MAML. Jego treść

**Listing 1** MAML

[FAL17]

**Require:** $p(\mathcal{T})$: rozkład prawdopodobieństwa nad przestrzenią tasków
**Require:** $\alpha$, $\beta$: dodatkowe, stałe parametry uczenia
1: Ustaw $\theta$ na losową wartość
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: Oblicz $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ ze względu na elementy w $\mathcal{S}_i$
6: Oblicz adaptację (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) ▷ $\theta = \{\theta^E, \theta^H\}$

można znaleźć w listingu 1. Algorytm zaczyna pracę, przyjmując losowe parametry $\theta$. Następnie zbiór danych (ang. dataset) dzieli się na losowo dobrane partie danych (ang. batch). Operując na wagach $\theta$ obliczamy tzn. adaptację. Adaptację uzyskujemy poprzez obliczenie gradientu funkcji kosztu, która dana jest wzorem:

$$\mathcal{L}_{\mathcal{T}_i}(f_\theta) = \sum_{j=1}^{k} \sum_{l=1}^{L} y_l^{(j)} \log(f_\theta(x_l^{(j)})) + (1 - y_l^{(j)}) \log(1 - f_\theta(x_l^{(j)})) \tag{3}$$

gdzie: $(x_l^{(j)}, y_l^{(j)})$ to para input-label, $L$ to ilość klas. Wzór ten wyraża zwykłą entropię krzyżową dla klasyfikacji o skończonej liczbie klas. Wagi $\theta$ (przed adaptacją) nazywamy globalnymi. Wagi $\theta_i$ po adapacji nazywamy szybkimi.

Obliczony gradient oraz parametr $\alpha$ przykładamy do algorytmu gradient descent. Po wyjściu z pętli, dostosowujemy parametry globalne za pomocą parametru $\beta$ i sumie funkcji kosztu wywołanego przez błąd klasyfikacji wag szybkich.
Na rysunku 3 przedstawiony jest proces adaptacji do wag szybkich pod wpływem supportu $\mathcal{S}$.

# 4 Metody

## 4.1 HyperMAML

Algorytm MAML posiada pewne ograniczenia wynikające z optymalizacji opartej na wyliczaniu gradientu. Okazuje się, że model często nie jest w stanie dokonać wystarczającej adaptacji za pomocą jednego, a nawet kilku kroków gradientowych. Co więcej, w przypadku przetwarzania danych wysokiego wymiaru, zwiększanie ilości kroków gradientowych, w celu polepszenia jakości działania, stanowi spory obliczeniowy narzut. [PPT+22]. Przykładem zaproponowanego rozwiązania tego problemu jest HyperMAML, modyfikacja algorytmu MAML, wprowadzająca paradygmat hiper-sieci, który zastępuje adaptację poprzez gradient. [PPT+22] Treść algorytmu HyperMAML można przeczytać w listingu 2. Modyfikacja względem algorytmu bazowego jest następująca: do architektury wprowadzamy nową sieć neuronową parametryzowaną przez $\eta$. Adaptacja gradientowa jest zastąpiona przez hypernetwork **H**, który odwzorowuje *embedding*, predykcję na wagach

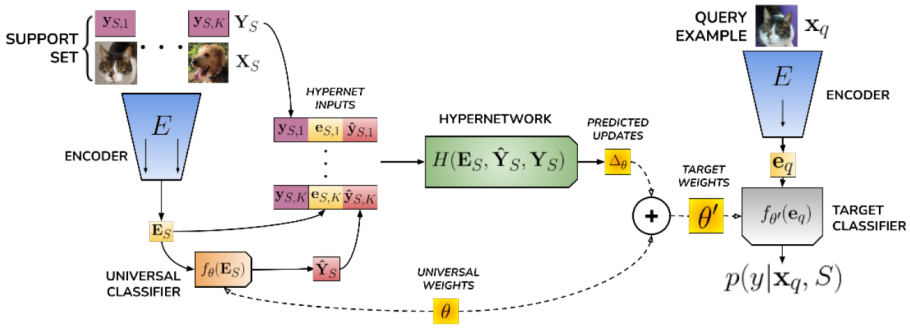
globalnych $\theta$ oraz etykiety supportu na tzn. deltę wag $\Delta\theta$. Następnie procedura postępuje analogicznie do algorytmu MAML, z zastrzeżeniem, że po wyjściu z pętli dostosowywane są również wagi $\eta$. Poglądową architekturę HyperMAML można zobaczyć na rysunku 4.

**Listing 2** HyperMAML

[PPT+22]

**Require:** $p(\mathcal{T})$: rozkład prawdopodobieństwa nad przestrzenią tasków
**Require:** $\beta$: dodatkowy parametr uczenia
1: Ustaw $\theta$, $\eta$, $\gamma$ na losową wartość
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: Oblicz adaptację $\theta_i = \theta - \mathbf{H}(\mathcal{S}_i) = \theta - \Delta\theta$
6: $\mathcal{L}_{\text{HyperMAML}} := \sum_{T_i} \mathcal{L}_{Q_i}(f_{\theta_i})$
7: $\theta \leftarrow \theta - \beta\nabla_\theta \mathcal{L}_{\text{HyperMAML}}(f_\theta)$ ▷ Wagi globalne $\theta$
8: $\eta \leftarrow \eta - \beta\nabla_\eta \mathcal{L}_{\text{HyperMAML}}(f_\theta)$ ▷ Hiper- sieć $H_\eta$
9: $\gamma \leftarrow \gamma - \beta\nabla_\gamma \mathcal{L}_{\text{HyperMAML}}(f_\theta)$ ▷ Ekstraktor $E_\gamma$

Powyższa konstrukcja wnosi kilka istotnych usprawnień. Po pierwsze, uniknięcie obliczeń związanych z propagacją gradientu funkcji kosztu w pętli adaptacji poprzez zastosowanie hiper-sieci jest bardziej obliczeniowo efektywne. Po drugie, zastosowanie hiper-sieci zwiększa efektywność adaptacji modelu. [PPT+22]




Rys. 4: Architektura modelu HyperMAML [PPT+22]

## 4.2 Regularyzacja

Sieci neuronowe zawierają bardzo dużą ilość parametrów. Z tego powodu istotnym problemem w ich uczeniu jest zjawisko nadmiernego dopasowania modelu (ang. overfitting). O modelu, w którym występuje overfitting mówimy, że słabo generalizuje dane, na których jest uczony. Oznacza to, że nawet przy wysokiej sprawności modelu wykazywanej w metrykach liczonych na zbiorze treningowym, model znacznie gorzej będzie radzić sobie z pracą z danymi na zbiorze testowym. W celu naprawienia problemu niskiej generalizacji, stosuje się różne techniki regularyzacyjne. Wyróżniamy między innymi:

1. penalizacja normy wag. Przykłady:
   (a) regularyzacja lasso
   (b) regularyzacja ridge
   (c) elastic-net

   Więcej o regularyzacji przez penalizację normy wag można przeczytać w [HTF01]

2. dropout. Jest to technika stosowana na losowo wybranych wagach sieci. Regularyzacja jest dokonywana poprzez losowe części wyzerowanie parametrów w trakcie procesu propagacji wprzód (ang. forward propagation). Więcej o technice dropout można przeczytać w [SHK+14]

3. early-stopping. Metoda opierająca się na różnych metrykach zwracanych przez model. Jej zadaniem jest zatrzymianie procesu uczenia w momencie, gdy przestanie on przynosić wystarczające rezultaty. W praktyce, metoda zatrzymuje proces uczenia zanim dojdzie do zjawiska overfittingu. Więcej o tej metodzie można przeczytać w [YRC07] .

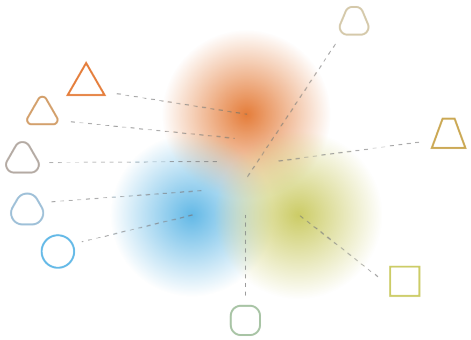
### 4.3 Problem jakości reprezentacji

Wspominaliśmy już o przestrzeni ukrytej, która w problemach głębokiego uczenia stanowi zanurzenie umożliwiające redukcję wymiarowości i ekstrakcję cech. Od idealnej przestrzeni ukrytej $P$ oczekuje się, że:

- Niech $x, y \in P$ są blisko siebie. Ich odpowiedniki $x', y'$ w faktycznej przestrzeni danych są do siebie podobne.

- Wybieramy losowy punkt $x \in P$. Chcemy, żeby posiadał on sensowny odpowiednik w przestrzeni danych. Ustalmy rozkład prawdopodobieństwa $R$. Niech $s \sim R$. Przez sensowny rozumiemy taki, ze wylosowanie $s$ z rozkładu $R$ jest tak samo prawdopodobne jak wylosowanie $s'$ z rozkładu przestrzeni danych. Zauważmy, że w sytuacji odwrotnej do idealnej, łatwo uda nam się wylosować próbki z przestrzeni ukrytej takie, że ich odpowiedniki w przestrzeni danych nie mają sensu lub nierealnie różnią się od dostępnych danych.

Intuicje na temat powyższych warunków można zobaczyć na rysunku 5.



Rys. 5: Reprezentacja danych w "idealnej" przestrzeni ukrytej [Roc]. Figury geometryczne o podobnym kształcie leżą blisko siebie, w ramach wspólnej rozmaitości kształtów

Problem reprezentacji możemy przenieść na grunt problemu regularyzacji. Jak już opisywaliśmy, overfitting w sieciach neuronowych występuje, gdy sieć nie generalizuje się na dane z przestrzeni, na której uczymy. Przykładowo oznacza to, że dla dla dwóch podobnych obrazków, sieć zwróci dwa zupełnie niepodobne do siebie wyniki. Wspominaliśmy również, że sieci neuronowe pracują na wagach, które propagują dane. Jeśli założymy sposób reprezentacji wag taki jak w algorytmach MAML i HyperMAML, to można powiedzieć, że sieci neuronowe propagują dane deterministyczne. Naturalnym wydaje się pytanie, czy taki sposób reprezentacji jest wystarczający. W celach regularyzacyjnych możemy bowiem szukać bardziej wyrafinowanych sposób na reprezentacje parametrów modelu.
W następnych podrozdziałach wprowadzimy dwie modyfikacje regularyzacyjne algorytmu HyperMAML. Pierwsza z nich inicjuje tzn. bayesowską sieć neuronową. Jest to zmiana sposobu reprezentacji wag poprzez utożsamienie każdej wagi sieci z rozkładem normalnym $\mathcal{N}(\mu, \sigma)$. Druga z nich generalizuje pierwszą, utożsamiając parametry sieci z bardziej skomplikowanymi rozkładami, modelowanymi przez zewnętrzny algorytm generatywny.

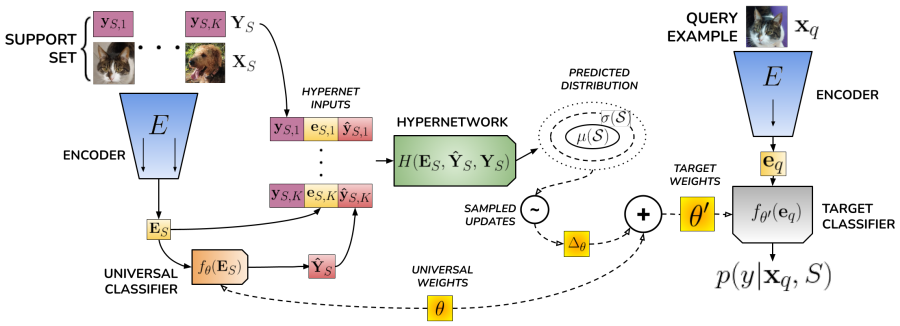
## 4.4 Bayesowskie podejście do algorytmu MAML

Najprostszym bayesowskim podejściem do algorytmu MAML jest zadanie prioru dla parametrów modelu, a uczenie się ich posterioru. W takim algorytmie model musi nauczyć się rozkładów zarówno dla parametrów $\theta$ jak i $\theta'$. Prowadzi to do hierarchicznego modelu bayesowskiego: $\theta \rightarrow \theta'_i \rightarrow \mathcal{T}_i$ (zaproponowane np. w pracy [RB18]). W podejściu tym stosuje się wcześniej wspomniany *variational inference* wraz z reparametryzacją gradientu([BCKW15]) oraz ELBO maksymalizwane ze względu na parametry $\lambda_i$ oraz $\psi$:

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

gdzie: $q(\theta'_i|\lambda_i)$ oraz $q(\theta|\psi)$ to kolejno przybliżony posterior dla tasku i przybliżony posterior dla uniwersalnych wag. Są one ze sobą związane poprzez prior $p(\theta'_i|\theta)$.

Powyższe rozwiązanie wiąże się z kilkoma problemami. Jednym z nich jest odpowiednie dobranie priora $p(\theta'_i|\theta)$, tak aby $q(\theta'_i|\lambda_i)$ oraz $q(\theta|\psi)$ były w stanie uczyć się równocześnie poprzez $p(\theta'_i|\theta)$. Pojawiło się kilka prób rozwiązania tego problemu, jedyną z nich jest podejście przedstawione w pracy [RB18]. W podejściu proponowanym przez Ravi'ego oraz Beatson'a przyjmują oni $q(\theta'_i|\lambda_i) = \text{SGD}(\mathcal{T}_i, \theta)$, gdzie SGD oznacza wykonanie kilku kroków gradientowych $\nabla_{\lambda_i}\mathcal{L}_{\mathcal{T}_i}$.




Rys. 6: Architektura modelu BayesHMAML. [BKP+22].

## 4.5 BayesHMAML

W tej pracy zaprezentowane jest podejście, próbujące ominąć problemy wcześniejszych metod. Nasze rozwiązanie polega na uczeniu się uniwersalnych wag $\theta$ w sposób punktowy. Nie pojawia się więc wariancja w przypadku parametrów $\theta$, a zamiast tego przechodzi ona całkowicie na parametry $\theta'_i$. Podejście to pozwala na uniknięcie powtarzającego się priora pomiędzy $\theta$ i $\theta'_i$. W jego miejsce pojawia się niehierarchiczny prior $p(\theta'_i)$. Dzięki tym modyfikacjom oraz użyciu hiper-sieci, jesteśmy w stanie osiągnąć lepsze optimum dla funkcji celu:

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

W praktyce wykorzystujemy standardowy rozkład normalny jako prior dla wag sieci, czyli $p(\theta'_i) = \mathcal{N}(\theta'_i|0,\mathbb{I})$ oraz $\gamma$ jest hiperparametrem modelu

Najważniejszym elementem algorytmu BayesHMAML jest sposób zadania posterioru jako $q(\theta'_i|\lambda_i(\theta,\mathcal{S}_i))$. W szczególności, w naszym modelu ustalamy $q(\theta'_i|\lambda_i(\theta,\mathcal{S}_i)) = \mathcal{N}(\theta'_i|\theta + \mu_i(\mathcal{S}_i)\sigma_i(\mathcal{S}_i))$, gdzie $\mu_i(\mathcal{S}_i)$ oraz $\sigma_i(\mathcal{S}_i))$ to wyjście hiper-sieci $H_\phi$. Optymalizacja funkcji kosztu $\mathcal{L}_{\mathcal{D}}$ jest wykonywana względem wag globalnych $\theta$ oraz wag hiper-sieci $\phi$. Działanie algorytmu przedstawione jest w listingu 3, natomiast poglądową architekturę można znaleźć na rysunku 6. Uczenie wykonywane jest na mini-batchach, a hiperparametr $\gamma$ zmienia się podczas uczenia. Dla każdego tasku $\mathcal{T}_i$ zbierane są informacje z $\mathcal{S}_i$, dzięki czemu jesteśmy w stanie wygenerować rozkłady prawdopodobieństwa

odpowiednie dla każdego $\mathcal{T}_i$ z osobna: $(\mu(\mathcal{S}_i), \sigma(\mathcal{S}_i)) = H_\phi(\mathcal{S}_i, f_\theta(\mathcal{S}_i)$. Następnie losowane są szybkie $\theta'_i \sim \mathcal{N}(\theta + \mu(\mathcal{S}_i), \sigma(\mathcal{S}_i))$ na podstawie których dokonujemy ostatecznej predykcji.

Ostateczna optymalizacja parametrów modelu pomiędzy taskami dokonywana jest na podstawie funkcji kosztu składającej sie z cross-entropii oraz dywergencji Kullbacka-Leiblera:

$$\mathcal{L}_\mathcal{T} = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \left[ \mathbf{CE}(f_{\theta'_i}) - \gamma \mathbf{KL}[\mathcal{N}(\theta + \mu(\mathcal{S}_i), \sigma(\mathcal{S}_i)) \| \mathcal{N}(0, \mathbb{I})] \right]$$

gdzie: hiperparametr $\gamma$ zmeinia się od 0 do pewnej ustalonej wartości w trakcie uczenia

**Listing 3** BayesHMAML

[BKP+22]

**Require:** $p(\mathcal{T})$: rozkład prawdopodobieństwa nad przestrzenią tasków
**Require:** $\beta$, $\gamma$: dodatkowe parametry uczenia

**while** not done **do**
  Wylosuj batch $\mathcal{B} = \{\mathcal{T}_i\}_{i=1}$
  **for each** $\mathcal{T}_i \in \mathcal{B}$ **do**
    Oblicz prawdopodobieństwo rozkładów adaptowanych parametrów sieci:
    $(\mu(\mathcal{S}_i), \sigma(\mathcal{S}_i)) = H_\eta(S_i, f_\theta(S_i))$
    Losuj $\theta_i \sim \mathcal{N}(\theta + \mu(S_i), \sigma(S_i))$
  $\mathcal{L}_{\text{BayesHMAML}} = \sum_{\mathcal{T}_i} [\mathcal{L}_{\mathcal{T}_i}(f_{\theta_i}) - \gamma KLD[\mathcal{N}(\theta + \mu(\mathcal{S}_i), \sigma(\mathcal{S}_i)), \mathcal{N}(0, \mathbb{I})]]$
  $\theta \leftarrow \theta - \beta \nabla_\theta \mathcal{L}_{\text{BayesHMAML}}(f_\theta)$ ▷ Wagi globalne $\theta$
  $\eta \leftarrow \eta - \beta \nabla_\eta \mathcal{L}_{\text{BayesHMAML}}(f_\theta)$ ▷ Hiper- sieć $H_\eta$
  $\gamma \leftarrow \gamma - \beta \nabla_\gamma \mathcal{L}_{\text{BayesHMAML}}(f_\theta)$ ▷ Ekstraktor $E_\gamma$

## 4.6 Modele generatywne flow (ang. flow- based generative models)

Modele generatywne flow (dalej będziemy nazywać modelami *flowowymi*) to rodzina modeli generatywnych, które wykorzystują metodę **normalizing flow** w celu transformacji prostych rozkładów na złożone estymacje rozkładów przestrzeni wysokoparametrowych danych [RM15]. Metoda normalizing flow wykorzystuje regułę zamiany zmiennych do konstrukcji rozkładów za pomocą ciągu pewnych łatwo odwracalnych, różniczkowalnych odwzorowań. Poglądową ilustrację metody normalizing flow można zobaczyć na rysunku 7. W tym podrozdziale opowiemy o ogólnej konstrukcji metody, następnie przejdziemy do opisania wariantu **Continual Normalizing Flow** (CNF).

### 4.6.1 Konstrukcja

Niech $z_0$ pochodzi z rozkładu startowego $p_0$. W praktyce będziemy ustalać z góry rozkład startowy, np. $\mathcal{N}(0, \mathbb{I})$. Niech $\{f_i\}_{i=1}^n$ będzie ciągiem funkcji odwracalnych i różniczkowalnych. Niech $\{z_i\}_{i=1}^n$ będzie ciągiem zmiennych losowych przekształcanych z $z_0$, czyli $z_i = f_i(z_i)$ dla $i = 1, 2, ..., n$. Oznaczymy też $p_k$ jako rozkład przejściowy, z którego pochodzi zmienna losowa $z_k$.
Wyprowadzimy teraz wzór na log likelihood dla rozkładu docelowego $p_k$.

**Twierdzenie 1.** *Przy powyższych założeniach i oznaczeniach, zachodzi wzór:*

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

*gdzie:* $\det \frac{d\, f_{i-1}(z_{i-1})}{d\, z_{i-1}}$ *oznacza wyznacznik jakobianu funkcji* $f_{i-1}$ *ewaluowanego dla* $z_{i-1}$.

*Dowód.* Z reguły zamiany zmiennych oraz z twierdzenia o funkcji odwrotnej mamy:

$$p_1(z_1) = p_0(z_0)\, |\det \frac{d\, f_1^{-1}(z_1)}{d\, z_1}| = p_0\, |\det (\frac{d\, f_1(z_1)}{d\, z_1})^{-1}| \tag{5}$$

Jakobian funkcji odwrotnej jest odwracalny, ponieważ $f_i$ oraz $f_i^{-1}$ są różniczkowalne, zatem zachodzi:

$$p_0(z_0)\, |\det (\frac{d\, f_1(z_1)}{d\, z_1})^{-1}| = p_0(z_0)\, |\det (\frac{d\, f_1(z_1)}{d\, z_1})|^{-1} \tag{6}$$

Indukcyjnie, dostaniemy:

$$p_k(z_k) = p_0(z_0) \prod_{i=1}^{n} |\det \frac{d\, f_{i-1}(z_{i-1})}{d\, z_{i-1}}|^{-1} \tag{7}$$

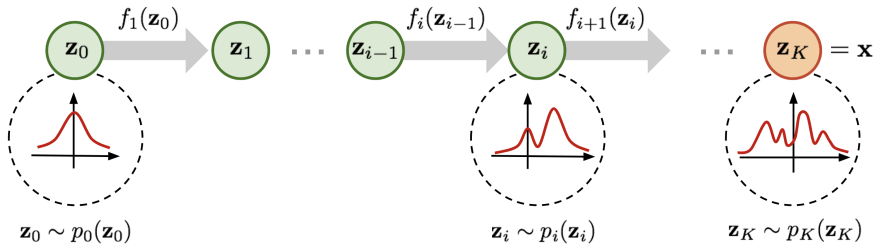
$$\log p_k(z_k) = \log p_0(z_0) - \sum_{i=1}^{n} \log |\det \frac{d\, f_{i-1}(z_{i-1})}{d\, z_{i-1}}| \tag{8}$$

Otrzymujemy funkcję, którą można minimalizować:

$$-\log p_k(z_k) = -\log p_0(z_0) + \sum_{i=1}^{n} \log |\det \frac{d\, f_{i-1}(z_{i-1})}{d\, z_{i-1}}| \tag{9}$$

□

W metodach flowowych będziemy modelować funkcje $f_i$ za pomocą sieci neuronowej. Ścieżkę zbudowaną ze złożeń funkcji $f_i$, którą pokonuje zmienna z rozkładu początkowego do rozkładu końcowego nazywamy **flowem**. O procesie treningu dla ogólnego przypadku można przeczytać w [PNR+21]. Poglądową transformację zmiennej z rozkładu początkowego na rozkład końcowy ilustruje rys. 7.




Rys. 7: Uproszczona ilustracja procesu normalizing flow, czyli transformacja prostego rozkładu w skomplikowaną aproksymację rozkładu dla danych w $K$ krokach [Wen]

W praktyce, z powodów numerycznych i optymalizacyjnych, funkcje flowowe ustala się w taki sposób, żeby spełniały dwa następujące warunki:

1. są łatwe do odwrócenia
2. ich jakobian jest łatwo wyliczalny

W następnej podsekcji krótko opiszemy architekturę, która spełnia powyższe warunki.

### 4.6.2 Real NVP

Przykład architektury generatywnej stosującej normalizing flow jest model **Real NVP** . Ciąg funkcji transformujących $\{f_j\}$ stanowią tutaj odwzorowania $\mathbb{R}^d \longrightarrow \mathbb{R}^D$ postaci [DSDB17]:

$$\begin{cases} f = (f_1, ..., f_D) \\ f_i(x) = x \;\text{ dla }\; i \leq d \\ f_{d+1:D}(x) = x_{d+1:D} \odot \exp(s(x_{1:d})) + t(x_{1:d}) \end{cases} . \tag{10}$$

gdzie: $\odot$ to iloczyn *element- wise*, odwzorowania $s, t : \mathbb{R}^d \longrightarrow \mathbb{R}^{D-d}$ są komponentami do wyuczenia.

Okazuje się, że tak sformuowane przekształcenia dają się łatwo odwrócić:

$$\begin{cases} f_i^{-1}(y) = y \;\text{ dla }\; i \leq d \\ f_{d+1:D}^{-1}(y) = (f_{d+1:D}(y) - t(f_{1:d}(y)) \odot \exp(-s(f_{1:d}(y))) \end{cases} \tag{11}$$

Co więcej, macierz Jakobiego funkcji $f$ jest zawsze dolnotrójkątna, więc jej wyznacznik jest łatwo obliczalny.

## 4.7 Continual Normalizing Flow

W metodach wykorzystujących własność (8), stosuje się pewną ilość przekształceń na skończonej ilości warstw. Największą trudnością w tym wypadku jest obliczanie wyznacznika jakobianu $\frac{df}{dz}$, które ma złożoność sześcienną ze względu na wymiarowość z lub ilość warstw architektury sieci [CRBD18]. Okazuje się, że przechodząc z takiego dyskretnego podejścia na ciągłą transformację zmiennej losowej, obliczenia stają się dużo prostsze do zrealizowania.

**Twierdzenie 2.** *Niech $z(t)$ będzie zmienną losową skończonego wymiaru o prawdopodobieństwie $p(z(t))$, które jest zależne od czasu $t$. Niech $\frac{dz}{dt} = f(z(t), t)$ będzie równaniem różniczkowym opisującym ciągłą względem czasu transformację $z(t)$. Załóżmy, że $f$ jest Lipschitzowska na $z$ i ciągła na $t$. Wtedy zachodzi:*

$$\frac{d \log p(z(t))}{dt} = -\mathbf{Tr}(\frac{df}{dz(t)}) \tag{12}$$

*gdzie:* **Tr** *oznacza ślad macierzy.*

*Dowód.* Pełny dowód twierdzenia można zobaczyć w [CRBD18] □

Stosując analogię do twierdzenia 1 oraz zakładając ciągłość transformacji w czasie, otrzymujemy log likelihood dla rozkładu docelowego architektury CNF:

$$\log p_1(z(t_1) = \log p_0(z(t_0)) - \int_{t_0}^{t_1} \mathbf{Tr}(\frac{df}{dz(t)})\, dt \tag{13}$$

W metodach CNF zatem stosuje się ciągłą funkcję dynamiki $f(z(t), t, \theta)$, reprezentowaną przez sieć neuronową i parametryzowaną za pomocą zwyczajnych równań różniczkowych.

### 4.7.1 Propagacja wsteczna w modelu CNF

Rozważmy dowolną funkcję kosztu $L : z(t_1) \longrightarrow \mathbb{R}$:

$$L(z(t_1)) = L(\int_{t_0}^{t_1} f(z(t), t, \theta)\, dt) \tag{14}$$

Można pokazać [CRBD18, Pon87], że pochodna $L$ po parametrach $\theta$ ma postać:

$$\frac{d\, L}{d\, \theta} = -\int_{t_1}^{t_0} (\frac{d\, L}{d\, z(t)})^T\, \frac{d\, f(z(t), t, \theta)}{d\, \theta}\, dt \tag{15}$$

Optymalizacja funkcji $L$ wygląda następująco:

1. Do optymalizacji funkcji kosztu potrzebujemy wielkości, określającej wielkość gradientu w zależności od czasu. Funkcję $a(t) = -\frac{d\, L}{d\, z(t)}$ nazywa się stanem pomocniczym. Dynamika $\frac{d\, a(t)}{d\, t}$ jest wyliczalna za pomocą *metody adjoint*. Więcej o niej można przeczytać w [CRBD18].
2. Obliczamy $z(t_1)$ na podstawie dynamiki $f_\theta(z(t), t) = \frac{d\, z(t)}{d\, t}$ oraz warunku początkowego $z(t_0)$
3. Obliczamy wstecznie (15), rozwiązując $\frac{d\, L}{d\, z(t)}$ z warunkiem początkowym $\frac{d\, L}{d\, z(t_1)}$. Do obliczenia (15) musimy dodatkowo znać całą trajektorię $z(t)$, co można osiągnąć, obliczając ją wstecznie, zaczynając od $z(t_1)$

Równania różniczkowe są rozwiązywane za pomocą metod numerycznych wynikających z konkretnej implementacji. Na nasze potrzeby takie solvery będziemy traktować jako czarną skrzynkę (ang. black box).

### 4.7.2 Free-Form Continous Dynamics For Scalable Reversible Generative Models (FFJORD)

Jednym z najbardziej powszechnych modeli implementujących CNF jest FFJORD [GCB+18]. Głównym wkładem algorytmu jest wprowadzenie usprawnień dotyczących obliczeń, opierając się na tzn. estymacji Hutchinsona, która efektywnie estymuje wartość śladu macierzy Jakobiego dzięki jej zredukowaniu do skalara za pomocą wektora szumu z odpowiedniego rozkładu.

## 4.8 Zastosowanie CNF w hipersieciach

Omówimy teraz kolejną modyfikację algorytmu HyperMAML. Dotychczas opisywaliśmy hiper- sieci w kontekście generowania przesunięcia wag klasyfikatora. W naszym zastosowaniu do regularyzacji modelu HyperMAML, hypernetwork jest stosowany do generowania warunkowania dla modułu CNF. Stosujemy hypernetwork:

$$H_\eta \colon \mathcal{S}_i \longrightarrow \theta_i \tag{16}$$

który odwzorowuje support na parametryzację warunkową modułu CNF. Funkcja dynamiki transformacji zmiennej losowej jest tworzona na podstawie wewnętrznych parametrów modułu i warunkowania podanego z zewnątrz. Funkcja ta buduje rozkład przesunięcia wag wyjściowego klasyfikatora $p(x \,|\, \mathcal{S}_i; \psi) = p_\psi(x\,;\, \theta_i)$. Z tak zbudowanego rozkładu, losujemy próbkę $\Delta\theta$ (za pomocą ciągłej transformacji próbki z ustalonego rozkładu początkowego na rozkład $p_\theta$). Następnie wielkość $\Delta\theta$ przykładamy do globalnych wag $\theta$, analogicznie do algorytmu HyperMAML. Tak scharakteryzowany algorytm nazywamy FHMAML. Poglądowy fragment architektury dotyczący dołączenia modułu CNF do HyperMAML można zobaczyć na rys. 8. Schemat algorytmu FHMAML można zobaczyć w listingu 5. Więcej na temat warunkowania modułu flowowego, jak i na temat szczegółów implementacyjnych, w tym implementacji warunkowania, modelu flowowego wykorzystanego w pracach przy modelu FHMAML można przeczytać w [ZP20].

### 4.8.1 Funkcja kosztu

Podobnie jak w metodzie BHMAML, do określenie funkcji kosztu wykorzystamy ELBO oraz funkcję KLD. Rozkłady flowowe nie są normalne. Dla rozkładów niegaussowskich stosuje się następujące przybliżenie:
Niech $q$ - przybliżany rozkład niegaussowski, $\theta$ - parametryzacja rozkładu flowa.

$$KL(q(w) \,|\, \mathcal{N}(0, \mathbb{I})) = \int q(w) \log\left(\frac{q(w)}{\mathcal{N}(0,\mathbb{I})(w)}\right) dw \tag{17}$$

$$\approx \frac{1}{S} \sum_{w \sim q(w)} \log\left(\frac{q(w)}{\mathcal{N}(0,\mathbb{I})(w)}\right) \tag{18}$$

$$= \frac{1}{S} \sum_{w \sim q(w|\theta)} \left(\log\left(q(w|\theta)\right) - \log \mathcal{N}(0,\mathbb{I})(w)\right) \tag{19}$$

$$= \frac{1}{S} \sum_{i=1}^{S} \left(\log \mathcal{N}(0,\mathbb{I})\left(F_\theta^{-1}(F_\theta(z_i))\right) + J - \log \mathcal{N}(0,\mathbb{I})(F_\theta(z_i))\right) \tag{20}$$

gdzie: $J$ oznacza jakobian zamiany zmiennych , $S$ oznacza rozmiar batcha.
Sposób obliczania powyższych komponentów funkcji wynika z implementacji [ZP20]. Poglądowy przebieg obliczania $\Delta\theta$ oraz funkcji kosztu w module CNF można zobaczyć w listingu 4

**Listing 4** Obliczanie $\Delta\theta$ oraz funkcji kosztu w module CNF

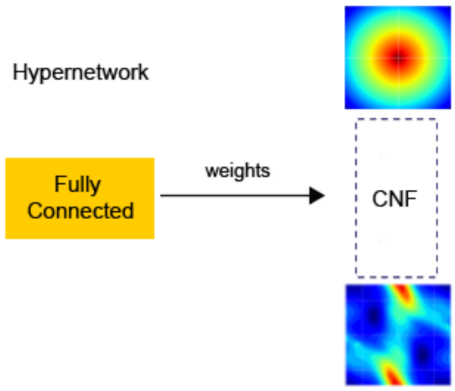
**Require:** rozkład początkowy - w naszym przypadku zakładamy $\mathcal{N}(0, \mathbb{I})$
**Require:** $\theta$: warunkowanie z Hypernetworka
1: Wylosuj $z_1, \ldots, z_k \sim \mathcal{N}(0, \mathbb{I})$
2: Oblicz $\Delta\theta = (w_1, \ldots, w_k) = (F_\theta(z_1), \ldots, F_\theta(z_k))$
3: Oblicz gęstość flowa względem rozkładu bazowego $\log \mathcal{N}(0,\mathbb{I})(F_\theta^{-1}(w_i))$
4: Oblicz gęstość priora $\log \mathcal{N}(0,\mathbb{I})(w_i)$
5: **return** $\sum_{i=1}^{k} \log \mathcal{N}(0,\mathbb{I})\left(F_\theta^{-1}(w_i)\right) + J$ , $\sum_{i=1}^{k} \log \mathcal{N}(0,\mathbb{I})(F_\theta(z_i)))$ , $\Delta\theta$

## 4.9 Zagadnienie niepewności

Niepewność to miara braku pewności co do poprawności wyjścia sieci neuronowej. W praktyce, rzadko mamy dostęp do danych, które umożliwią trening dający idealne wyniki. Z tego powodu, jednym z podstawowych zagrożeń dla sieci neuronowych jest nadmiarowa pewność działania. Zbytnia pewność siebie stanowi w oczywisty sposób szczególne zagrożenie w takich dziedzinach jak ochrona zdrowia, czy systemy autonomicznej jazdy. Z tego powodu, obecnie dużą wartość stanowią

**Listing 5** FHMAML

**Require:** $p(\mathcal{T})$: rozkład prawdopodobieństwa nad przestrzenią tasków
**Require:** $\beta$: dodatkowy parametr uczenia
Ustaw $\theta$, $\eta$, $\psi$, $\gamma$ na losową wartość
**while** not done **do**
  Wylosuj batch $\mathcal{B} = \{\mathcal{T}_i\}_{i=1}$
  **for each** $\mathcal{T}_i \in \mathcal{B}$ **do**
    Oblicz warunkowość $\theta_i = \mathbf{H}(\mathcal{S}_i)$
    Oblicz adaptację $\theta_i' = \theta - \mathbf{CNF}_\psi(\theta_i; x) = \theta - \Delta\theta$
  $\mathcal{L}_{\text{FHMAML}} := \sum_{T_i} \mathcal{L}_{Q_i}(f_{\theta_i})$
  $\theta \leftarrow \theta - \beta\nabla_\theta \mathcal{L}_{\text{FHMAML}}(f_\theta)$ ▷ Wagi globalne $\theta$
  $\eta \leftarrow \eta - \beta\nabla_\eta \mathcal{L}_{\text{FHMAML}}(f_\theta)$ ▷ Hiper- sieć $H_\eta$
  $\psi \leftarrow \psi - \beta\nabla_\psi \mathcal{L}_{\text{FHMAML}}(f_\theta)$ ▷ Warunkowy moduł CNF $CNF_\psi$
  $\gamma \leftarrow \gamma - \beta\nabla_\gamma \mathcal{L}_{\text{FHMAML}}(f_\theta)$ ▷ Ekstraktor $E_\gamma$

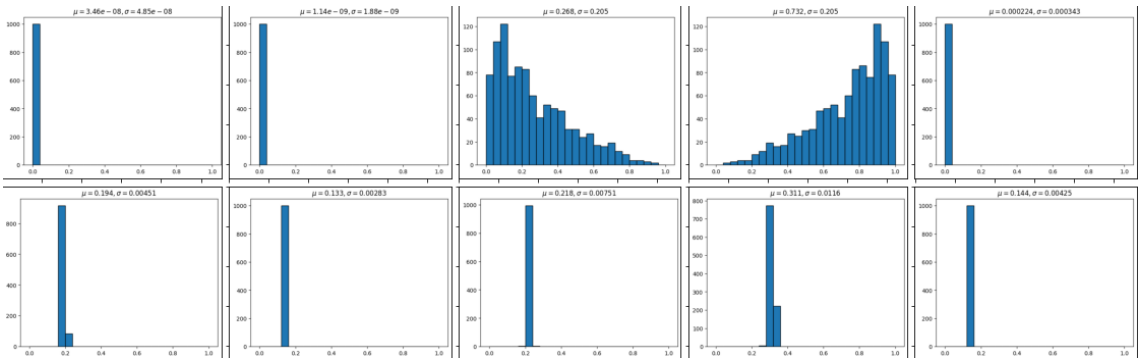



Rys. 8: Poglądowa architektura dołączanego modułu CNF [ZP20]

systemy, które prócz wysokich wyników dokładności, potrafią dostarczać informacji o poziomie przekonania do predykcji. Więcej o niepewności w Nauczaniu Maszynowym można przeczytać w [GTA+21]
Jedną z zalet architektur takich jak BayesHMAML lub FHMAML jest możliwość klarownej estymacji niepewności. Na obu modelach przeprowadzono następujący eksperyment:

1. Ustalmy support $\mathcal{S}$. Ustalmy też dwa zestawy query: $\mathcal{Q}_1$, $\mathcal{Q}_2$. Pierwszy pochodzi z domeny $\mathcal{S}$, drugi pochodzi z innej domeny ze względu na każdy element $\mathcal{S}$.
2. Przeprowadzamy adaptację na wagach globalnych, otrzymując $p(x|\mathcal{S})$
3. Losujemy 1000 zestawów wag z przestrzeni wag po adaptacji. Ewaluujemy $\mathcal{Q}_1$, $\mathcal{Q}_2$ na każdym zestawie.

Okazuje się, że oba modele zachowują dużą pewność na zbiorze $\mathcal{Q}_1$, jako że modele były wcześniej dostosowane do jego domeny. W przypadku $\mathcal{Q}_2$ modele podejmowały różne decyzje w zależności od wylosowanego zestawu wag. Wśród obu modeli, większą niepewnością dla nieadaptowanej domeny wykazał się model FHMAML. Na rysunku 9 można zobaczyć przykładowy wynik powyższego eksperymentu.




Rys. 9: Porównanie eksperymentu niepewności dla FHMAML (1. rząd) oraz BayesHMAML (2. rząd). Ten przykład pokazuje niepewność klasyfikacji dla zadania klasyfikacji 5- klasowej wykorzystując próbkę 1000 zestawów wag pobranych z przestrzeni wag adaptowanych do domeny obcej dla $\mathcal{Q}$. Nad histogramami można dostrzec dwie statystyki: średnią predykcję (0.0 oznacza klasę pierwszą, 0.2 oznacza klasę drugą,. . . , 1.0 oznacza klasę piątą) oraz odchylenie standardowe predykcji.

# Bibliografia


[BCKW15] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International conference on machine learning*, pages 1613–1622. PMLR, 2015.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[BKP+22] Piotr Borycki, Piotr Kubacki, Marcin Przewięźlikowski, Tomasz Kuśmierczyk, Jacek Tabor, and Przemysław Spurek. Hypernetwork approach to bayesian maml, 2022.

[CRBD18] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[DSDB17] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp, 2017.

[FAL17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[GCB+18] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models, 2018.

[GTA+21] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna M. Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, Muhammad Shahzad, Wen Yang, Richard Bamler, and Xiao Xiang Zhu. A survey of uncertainty in deep neural networks. *CoRR*, abs/2107.03342, 2021.

[HDL16] David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks, 2016.

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[PNR+21] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021.

[Pon87] L.S. Pontryagin. *Mathematical Theory of Optimal Processes*. Classics of Soviet Mathematics. Taylor & Francis, 1987.

[PPT+22] M Przewięźlikowski, P Przybysz, J Tabor, M Zięba, and P Spurek. Hypermaml: Few-shot adaptation of deep models with hypernetworks. *arXiv preprint arXiv:2205.15745*, 2022.

[RB18] Sachin Ravi and Alex Beatson. Amortized bayesian meta-learning. In *International Conference on Learning Representations*, 2018.

[RM15] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows, 2015.

[Roc] Joseph Rocca. Understanding variational autoencoders (vaes). `https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73`. Accesed: 2023 - 03 - 22.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[SS10] T. Schaul and J. Schmidhuber. Metalearning. *Scholarpedia*, 5(6):4650, 2010. revision #91489.

[TSS+22] Jacek Tabor, Marek Smieja, Lukasz Struski, Przemysław Spurek, and Maciej Wołczyk. *Głębokie uczenie. Wprowadzenie.* Helion, 2022.

[Wen] Lilian Weng. Flow-based deep generative models. `https://lilianweng.github.io/posts/2018-10-13-flow-models`. Accesed: 2023 - 03 - 26.

[Wol] Cameron R. Wolfe. Scene representation networks. `https://towardsdatascience.com/scene-representation-networks-bae6186d00d9`.

[YRC07] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, August 2007.

[ZP20] Maciej Zięba, Marcin Przewięźlikowski, Marek Śmieja, Jacek Tabor, Tomasz Trzcinski, and Przemysław Spurek. Regflow: Probabilistic flow-based regression for future prediction, 2020.

# 5 Dodatek

## 5.1 Szczegóły implementacyjne

Do tej pracy dołączony jest kod implementacyjny autorstwa Piotra Kubackiego i Piotra Boryckiego. Kod można znaleźć w repozytorium w serwisie github: https://github.com/piotr310100/few-shot-hypernets-public. Interesują nas branche z prefiksem *flow*. Główna implementacja znajduje się w branchu *flow_conditional_AtomOption*. W branchu o nazwie *flow_experiment_uncertainty* można zobaczyć kod, za pomocą którego wykonano eksperyment, który jest wizualizowany na rys. 9. W pliku opisującym repozytorium można znaleźć instrukcje dotyczące uruchomienia programu. Implementacja została przeprowadzona w języku Python przy udziale frameworka Pytorch. Dodatkowo, implementacja modułu CNF jest autorstwa [ZP20]. Repozytorium z modułem CNF można znaleźć pod adresem: https://github.com/maciejzieba/regressionFlow. Aby móc uruchomić model FHMAML z repozytorium few-shot-hypernets-public, należy dokonać instalacji niezbędnych podprogramów wymienionych w opisie repozytorium few-shot-hypernets-public oraz regressionFlow