Introduction to deep transfer learning with Xfer

Andreas Damianou

Amazon, Cambridge UK

Talk at the University of Leeds, 8 March 2019

Outline

- Deep neural networks quick reminder
- Transfer learning intro
- Xfer
  - Meta-learning
- Considerations

Further Resources

- Notebook: adamian.github.io/talks/Damianou_DL_tutorial_19.ipynb
- Xfer: github.com/amzn/xfer/
- Blog: link.medium.com/De5BXPJ9TT
- A more complete tutorial on deep learning: adamian.github.io/talks/Damianou_deep_learning_rss_2018.pdf

Deep Neural Networks intro

Deep neural networks: hierarchical function definitions

A neural network is a composition of functions (layers), each parameterized with a *weight vector* $\mathbf{w}_l$. E.g. for 2 layers:

$$f_{\text{net}} = h_2(h_1(\mathbf{x}; \mathbf{w}_1); \mathbf{w}_2).$$

Deep neural networks: hierarchical function definitions

A neural network is a composition of functions (layers), each parameterized with a *weight vector* $\mathbf{w}_l$. E.g. for 2 layers:

$$f_{\text{net}} = h_2(h_1(\mathbf{x}; \mathbf{w}_1); \mathbf{w}_2).$$

Generally $f_{\text{net}} : \mathbf{x} \mapsto \mathbf{y}$ with:

$$\begin{aligned}
\mathbf{h}_1 &= \varphi(\mathbf{x}\mathbf{w}_1 + b_1) \\
\mathbf{h}_2 &= \varphi(\mathbf{h}_1\mathbf{w}_2 + b_2) \\
&\dots \\
\hat{\mathbf{y}} &= \varphi(\mathbf{h}_{L-1}\mathbf{w}_L + b_L)
\end{aligned}$$

ϕ is the (non-linear) activation function.

Defining the Loss

- We have our function approximator $f_{\text{net}}(x) = \hat{y}$
- We have to define our loss (objective function) to relate this function outputs to the observed data.
- E.g. squared difference $\sum_n (y_n - \hat{y}_n)^2$ or cross-entropy

Probabilistic re-formulation

- Training minimizing loss:

$$\arg\min_{\mathbf{w}} \underbrace{\frac{1}{2}\sum_{i=1}^{N}(f_{\mathsf{net}}(\mathbf{w}, x_i) - y_i)^2}_{\text{fit}} + \lambda \underbrace{\sum_i \| \mathbf{w}_i \|}_{\text{regularizer}}$$

Equivalent probabilistic view for regression, maximizing posterior probability:

$$\arg\max_{\mathbf{w}} \underbrace{\log p(\mathbf{y}|\mathbf{x}, \mathbf{w})}_{\text{fit}} + \underbrace{\log p(\mathbf{w})}_{\text{regularizer}}$$

where $p(\mathbf{y}|\mathbf{x}, \mathbf{w}) \sim \mathcal{N}$ and $p(\mathbf{w}) \sim$ Laplace

Optimization still done with back-prop (i.e. gradient descent).

Probabilistic re-formulation

- Training minimizing loss:

$$\arg\min_{\mathbf{w}} \underbrace{\frac{1}{2}\sum_{i=1}^{N}(f_{\textsf{net}}(\mathbf{w}, x_i) - y_i)^2}_{\textsf{fit}} + \lambda \underbrace{\sum_i \| \mathbf{w}_i \|}_{\textsf{regularizer}}$$

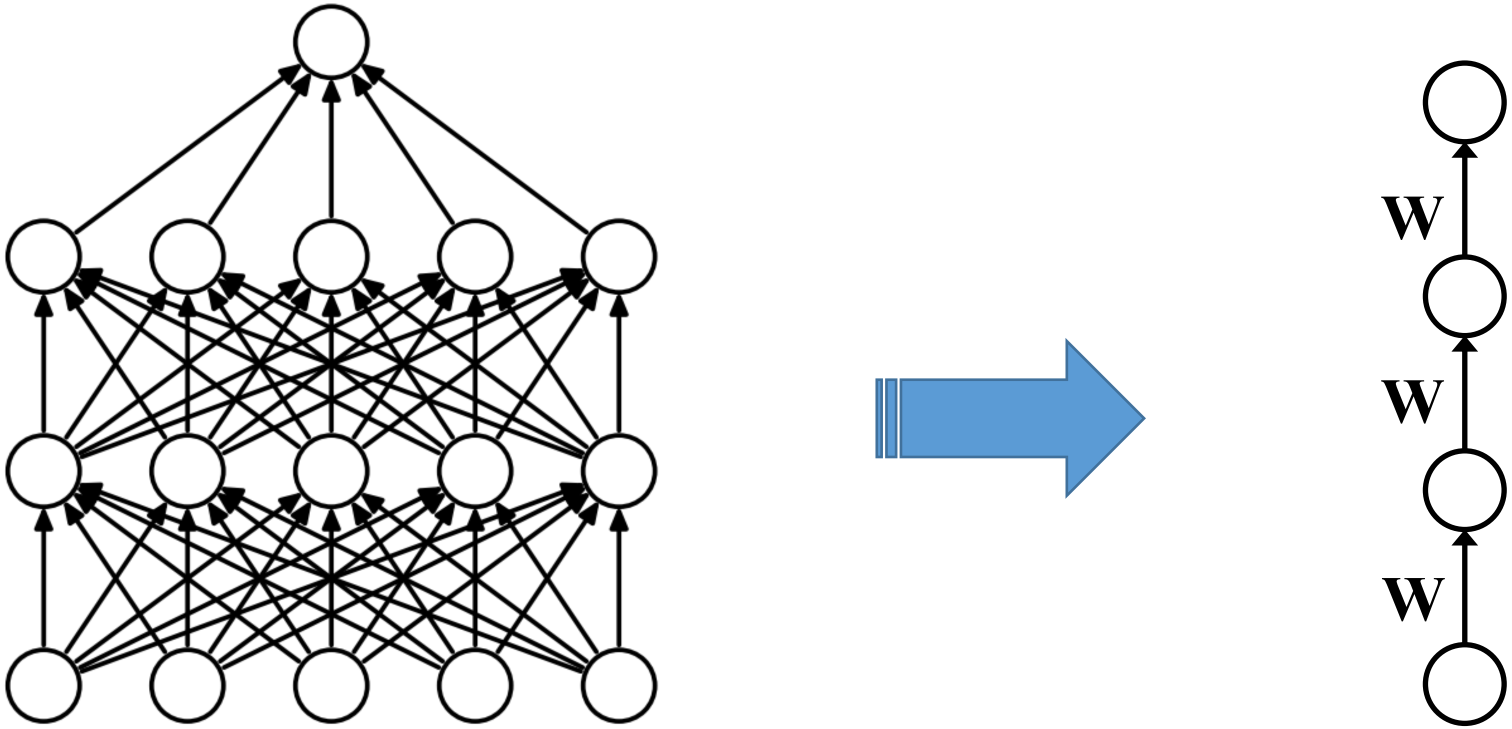
- Equivalent probabilistic view for regression, maximizing posterior probability:

$$\arg\max_{\mathbf{w}} \underbrace{\log p(\mathbf{y}|\mathbf{x}, \mathbf{w})}_{\textsf{fit}} + \underbrace{\log p(\mathbf{w})}_{\textsf{regularizer}}$$

where $p(\mathbf{y}|\mathbf{x}, \mathbf{w}) \sim \mathcal{N}$ and $p(\mathbf{w}) \sim$ Laplace

- Optimization still done with back-prop (i.e. gradient descent).

Graphical depiction



Derivative wrt w_0

$$
\begin{aligned}
\frac{\vartheta(\mathbf{h}_2 - \mathbf{y})^2}{\vartheta \mathbf{w}_0} &= -2\frac{1}{2}(\mathbf{h}_2 - \mathbf{y})\frac{\vartheta \mathbf{h}_2}{\vartheta \mathbf{w}_0} = \\
&= (\mathbf{y} - \mathbf{h}_2)\frac{\vartheta \phi(\mathbf{h}_1\mathbf{w}_1)}{\vartheta \mathbf{h}_1\mathbf{w}_1}\frac{\vartheta \mathbf{h}_1\mathbf{w}_1}{\vartheta \mathbf{h}_1}\frac{\vartheta \mathbf{h}_1}{\vartheta \mathbf{w}_0} = \\
&= \epsilon_2 \; g_1 \; \mathbf{w}_1^T \; \frac{\vartheta \phi(\mathbf{x}\mathbf{w}_0)}{\mathbf{x}\mathbf{w}_0}\frac{\vartheta \mathbf{x}\mathbf{w}_0}{\vartheta \mathbf{w}_0} = \\
&= \epsilon_2 \; g_1 \; \mathbf{w}_1^T \underbrace{\frac{\vartheta \phi(\mathbf{x}\mathbf{w}_0)}{\vartheta \mathbf{x}\mathbf{w}_0}}_{g_0} \; \mathbf{x}^T
\end{aligned}
$$

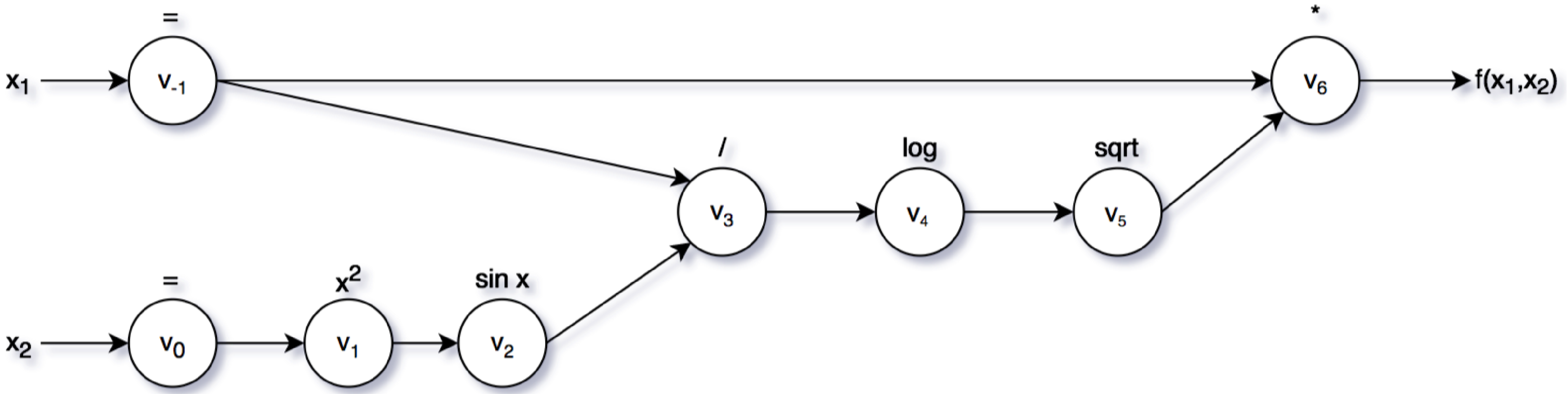
Propagation of error is just the chain rule.

Optimization & Implementation

GOTO notebook!!

Automatic differentiation

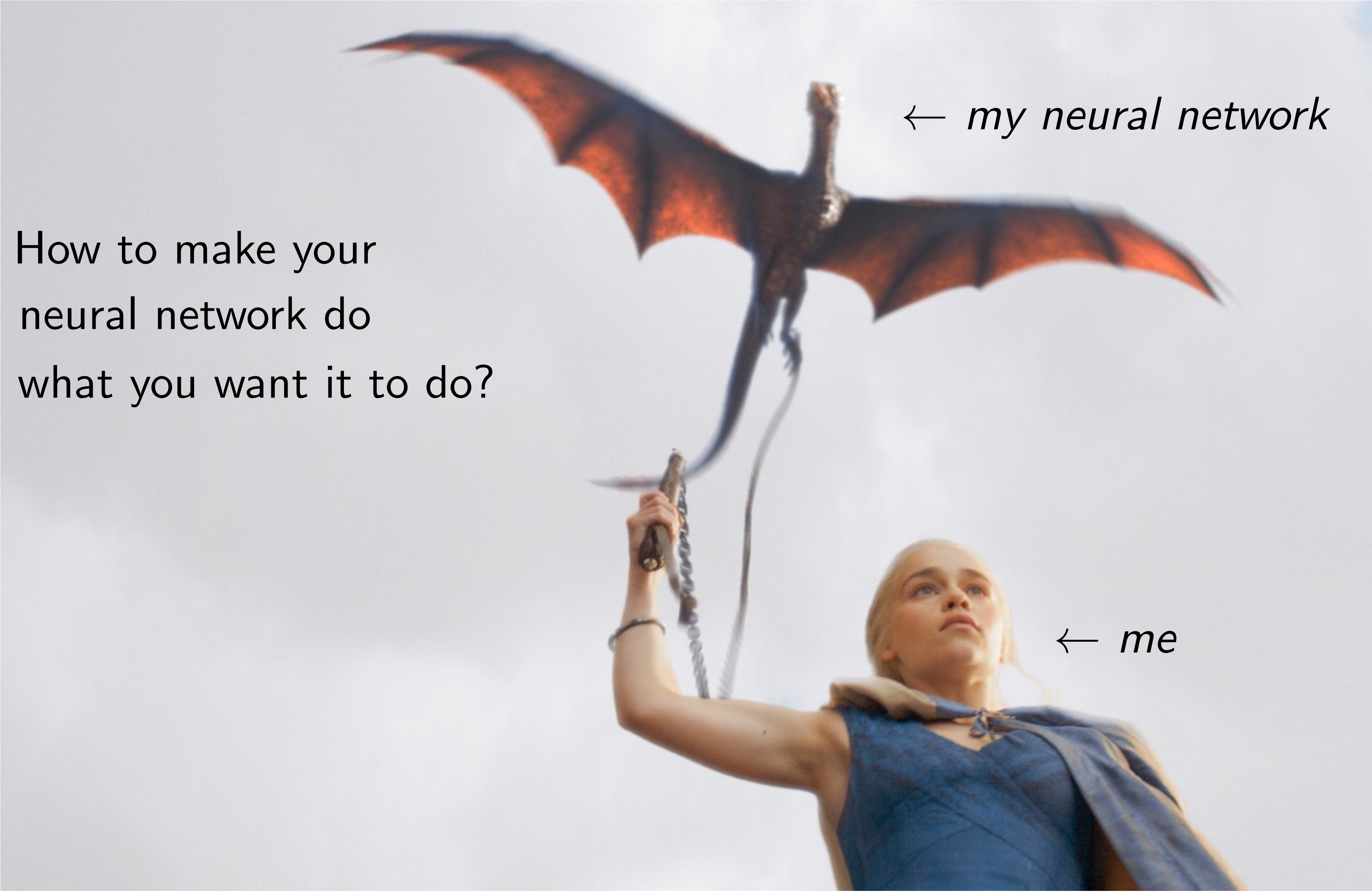
Example: $f(x_1, x_2) = x_1\sqrt{\log \frac{x_1}{sin(x_2^2)}}$ has *symbolic graph*:



(image: sanyamkapoor.com)

Back to notebook!

Taming the dragon



Probabilistic re-formulation

- Training minimizing loss:

$$\arg\min_{\mathbf{w}} \underbrace{\frac{1}{2}\sum_{i=1}^{N}(f_{\mathsf{net}}(\mathbf{w}, x_i) - y_i)^2}_{\text{fit}} + \lambda \underbrace{\sum_{i} \| \mathbf{w}_i \|}_{\text{regularizer}}$$

- Equivalent probabilistic view for regression, maximizing posterior probability:

$$\arg\max_{\mathbf{w}} \underbrace{\log p(\mathbf{y}|\mathbf{x}, \mathbf{w})}_{\text{fit}} + \underbrace{\log p(\mathbf{w})}_{\text{regularizer}}$$

where $p(\mathbf{y}|\mathbf{x}, \mathbf{w}) \sim \mathcal{N}$ and $p(\mathbf{w}) \sim$ Laplace

- Optimization still done with back-prop (i.e. gradient descent).

Bayesian deep learning

We saw that optimizing the parameters is a challenge.
Why not marginalize them out completely?

Integrating out weights

$D := (\mathbf{x}, \mathbf{y})$

$$p(w|D) = \frac{p(D|w)p(w)}{p(D) = \int p(D|w)p(w)\mathsf{d}w}$$

Inference

- $p(D)$ (and hence $p(w|D)$) is difficult to compute because of the nonlinear way in which w appears through g.

- Attempt at *variational inference*:

$$\underbrace{\mathrm{KL}\left(q(w;\theta)\,\|\,p(w|D)\right)}_{\text{minimize}} = \log(p(D)) - \underbrace{\mathcal{L}(q(w;\theta))}_{\text{maximize}}$$

where

$$\mathcal{L}(q(w;\theta)) = \underbrace{\mathbb{E}_{q(w;\theta)}\left[\log p(D,w)\right]}_{\mathcal{F}} + \mathbb{H}\left[q(w;\theta)\right]$$

- Term in red is still problematic. Solution: MC.

- Such approaches can be formulated as *black-box* inferences.

Inference

- $p(D)$ (and hence $p(w|D)$) is difficult to compute because of the nonlinear way in which w appears through g.

- Attempt at *variational inference*:

$$\underbrace{\mathsf{KL}\left(q(w;\theta)\,\|\,p(w|D)\right)}_{\text{minimize}} = \log(p(D)) - \underbrace{\mathcal{L}(q(w;\theta))}_{\text{maximize}}$$

where

$$\mathcal{L}(q(w;\theta)) = \underbrace{\mathbb{E}_{q(w;\theta)}\left[\log p(D,w)\right]}_{\mathcal{F}} + \mathbb{H}\left[q(w;\theta)\right]$$

- Term in red is still problematic. Solution: MC.

- Such approaches can be formulated as *black-box* inferences.

Inference

- $p(D)$ (and hence $p(w|D)$) is difficult to compute because of the nonlinear way in which w appears through g.

- Attempt at *variational inference*:

$$\underbrace{\mathsf{KL}\left(q(w;\theta) \,\|\, p(w|D)\right)}_{\text{minimize}} = \log(p(D)) - \underbrace{\mathcal{L}(q(w;\theta))}_{\text{maximize}}$$

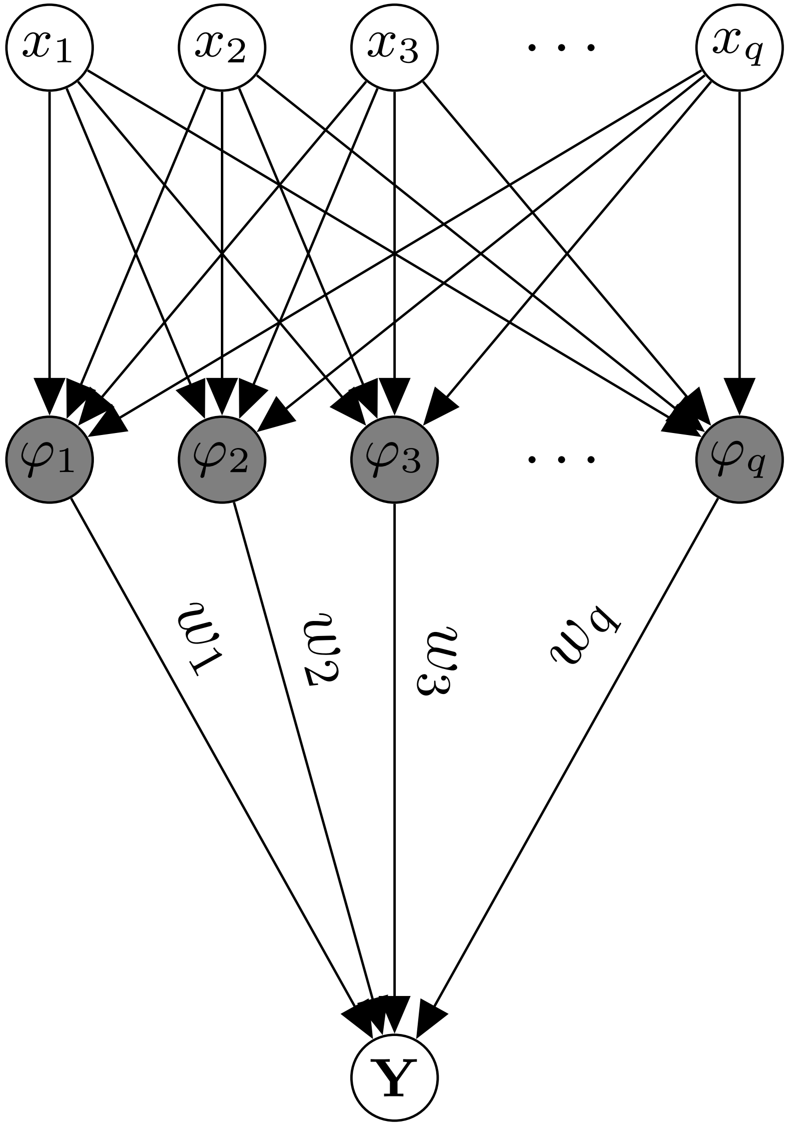
where

$$\mathcal{L}(q(w;\theta)) = \underbrace{\mathbb{E}_{q(w;\theta)}[\log p(D,w)]}_{\mathcal{F}} + \mathbb{H}\left[q(w;\theta)\right]$$

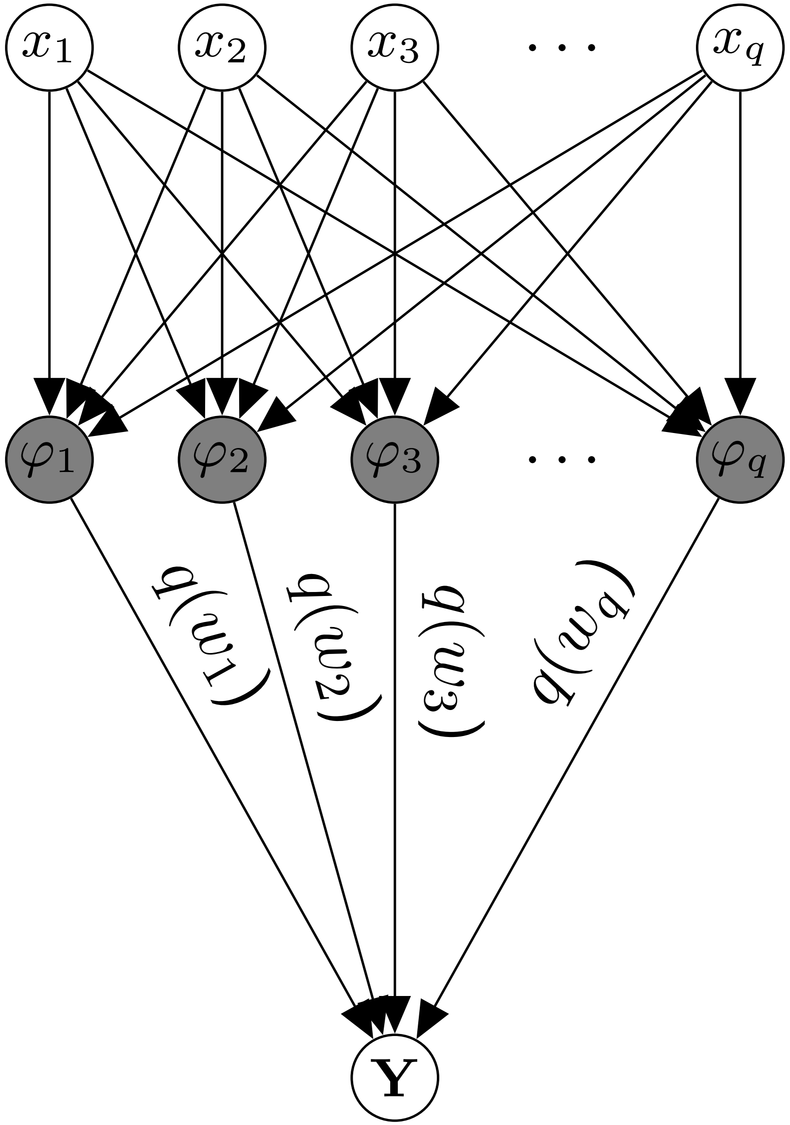
- Term in red is still problematic. Solution: MC.

- Such approaches can be formulated as *black-box* inferences.

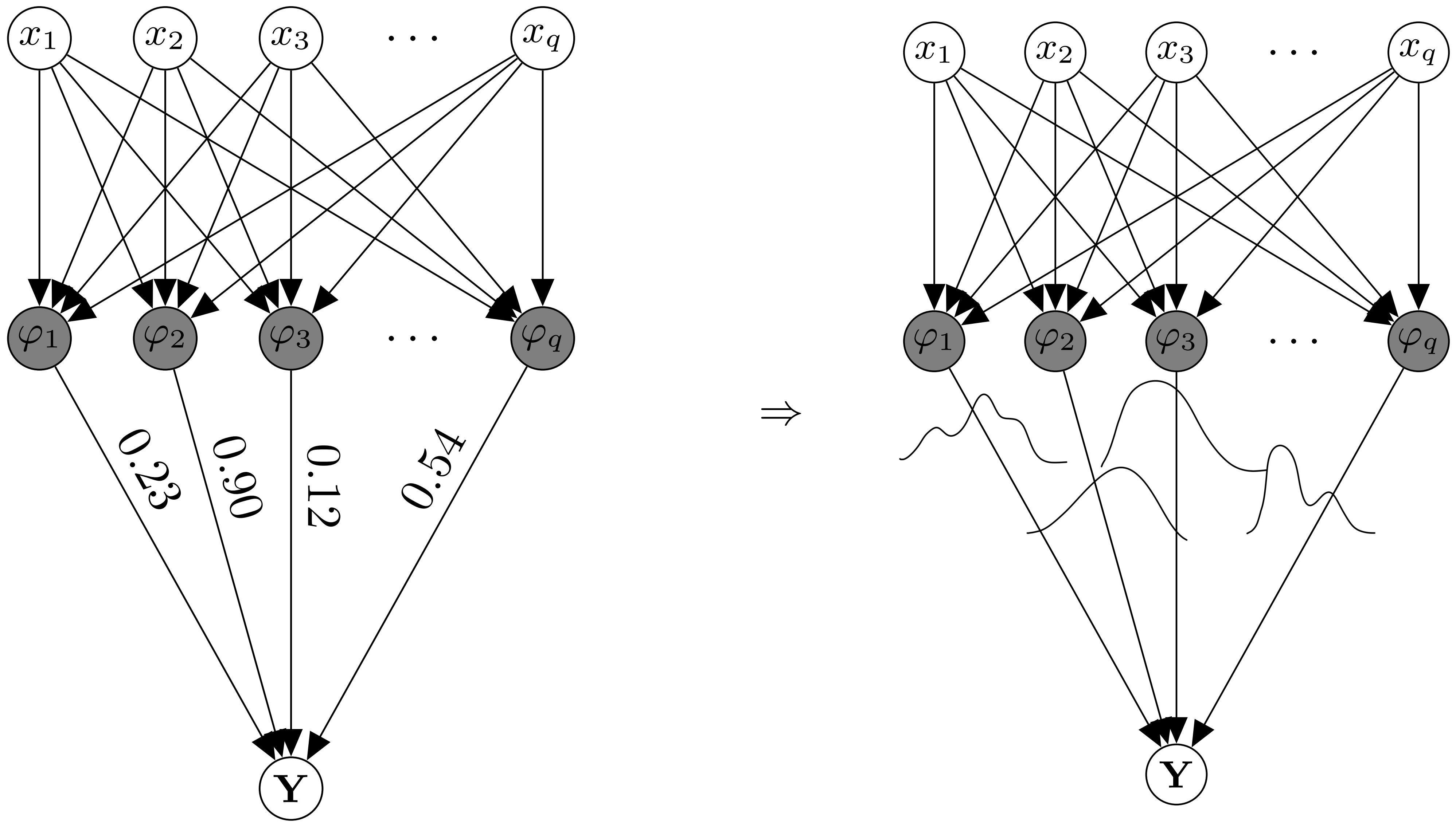
BNN with priors on its weights



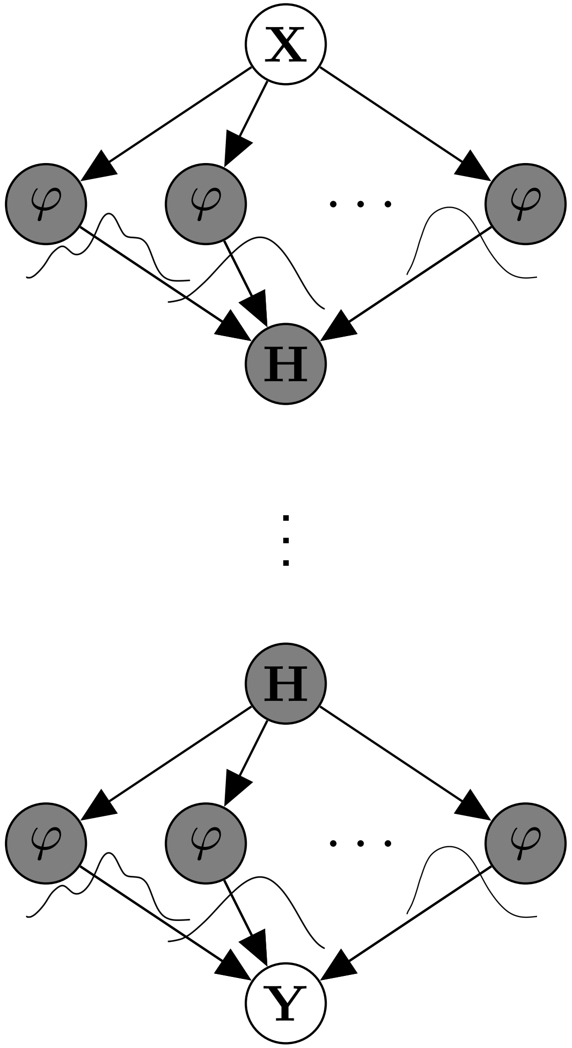
$\Rightarrow$



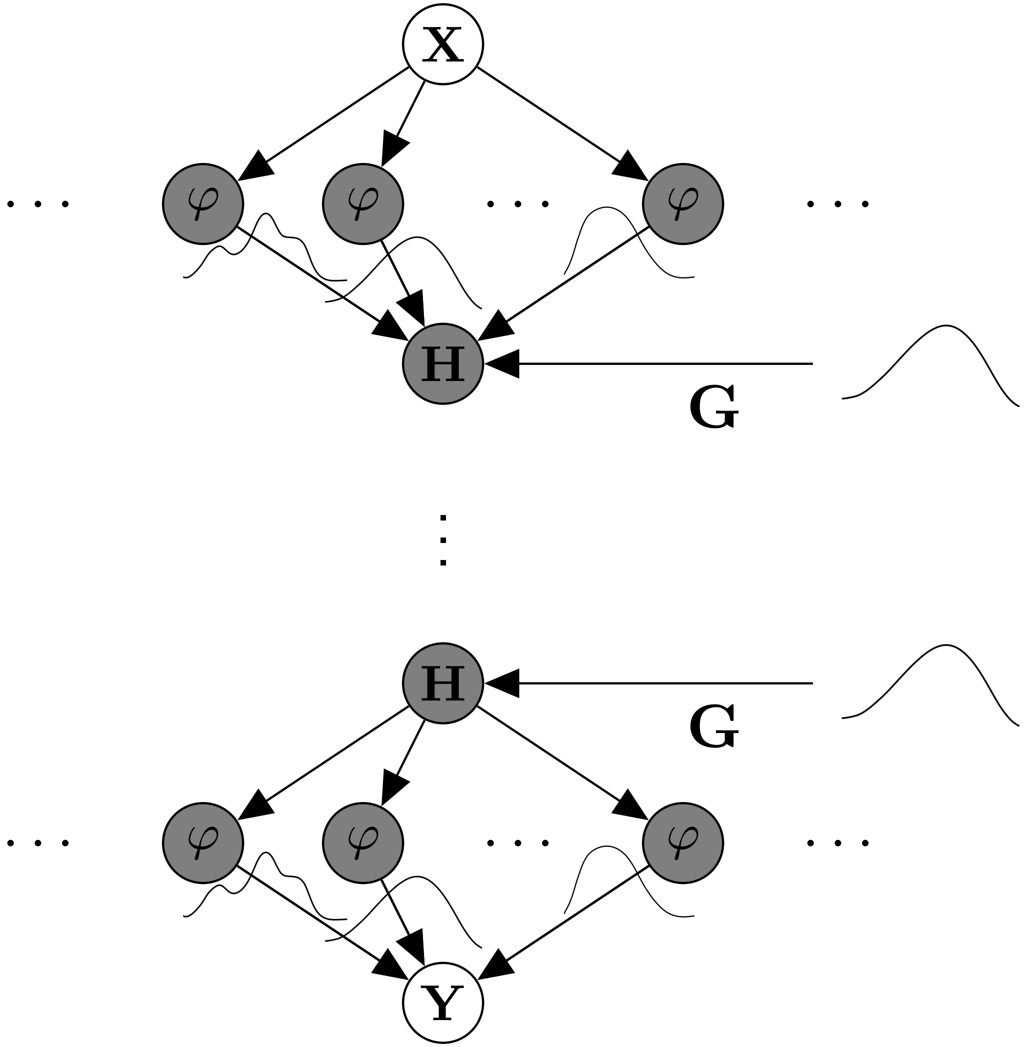
BNN with priors on its weights



Bayesian neural network *(what we saw before)*



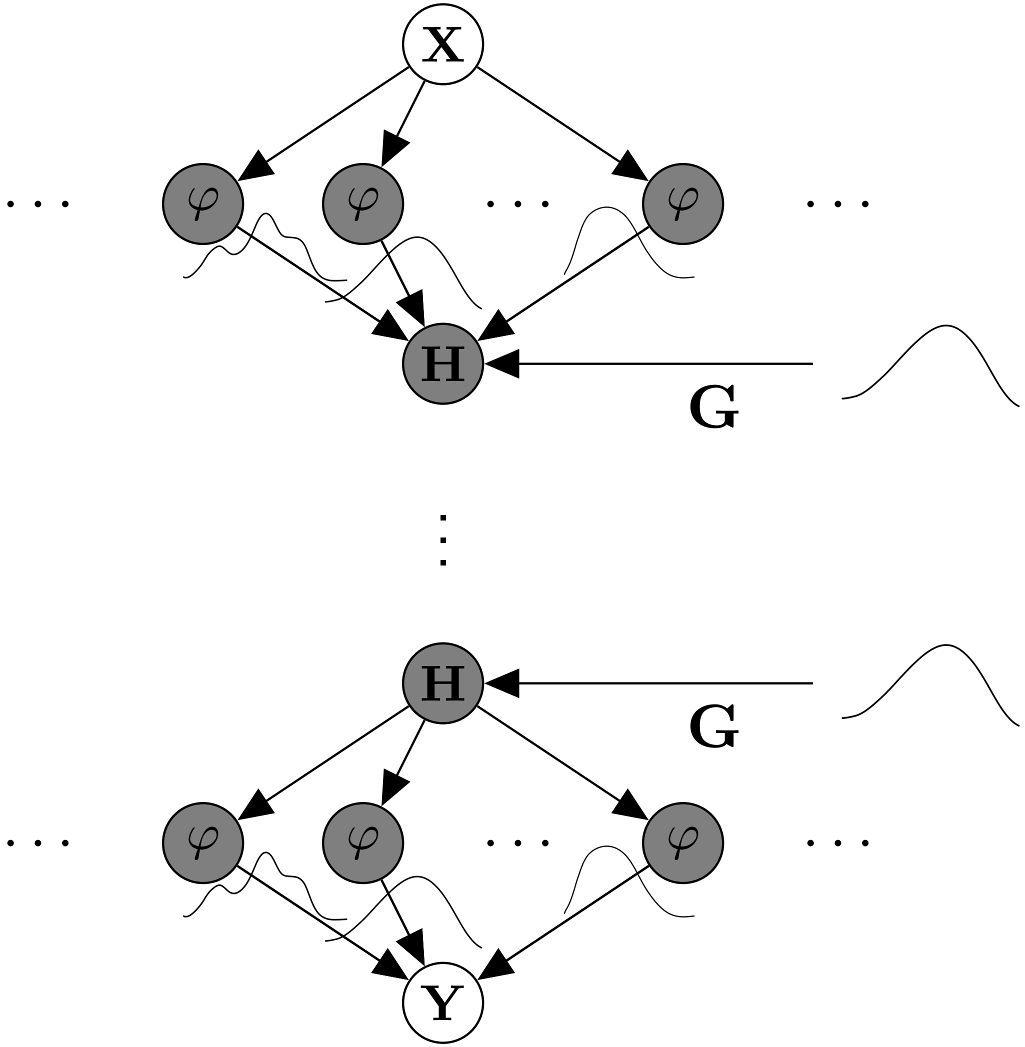
From NN to GP



- NN: $\mathbf{H}_2 = \mathbf{W}_2 \phi(\mathbf{H}_1)$
- GP: ϕ is $\infty-$dimensional so:
  $$\mathbf{H}_2 = f_2(\mathbf{H}_1; \theta_2) + \epsilon$$
- NN: $p(\mathbf{W})$
- GP: $p(f(\cdot))$

Deep Gaussian processes. A. Damianou, N. Lawrence, 2013

From NN to GP



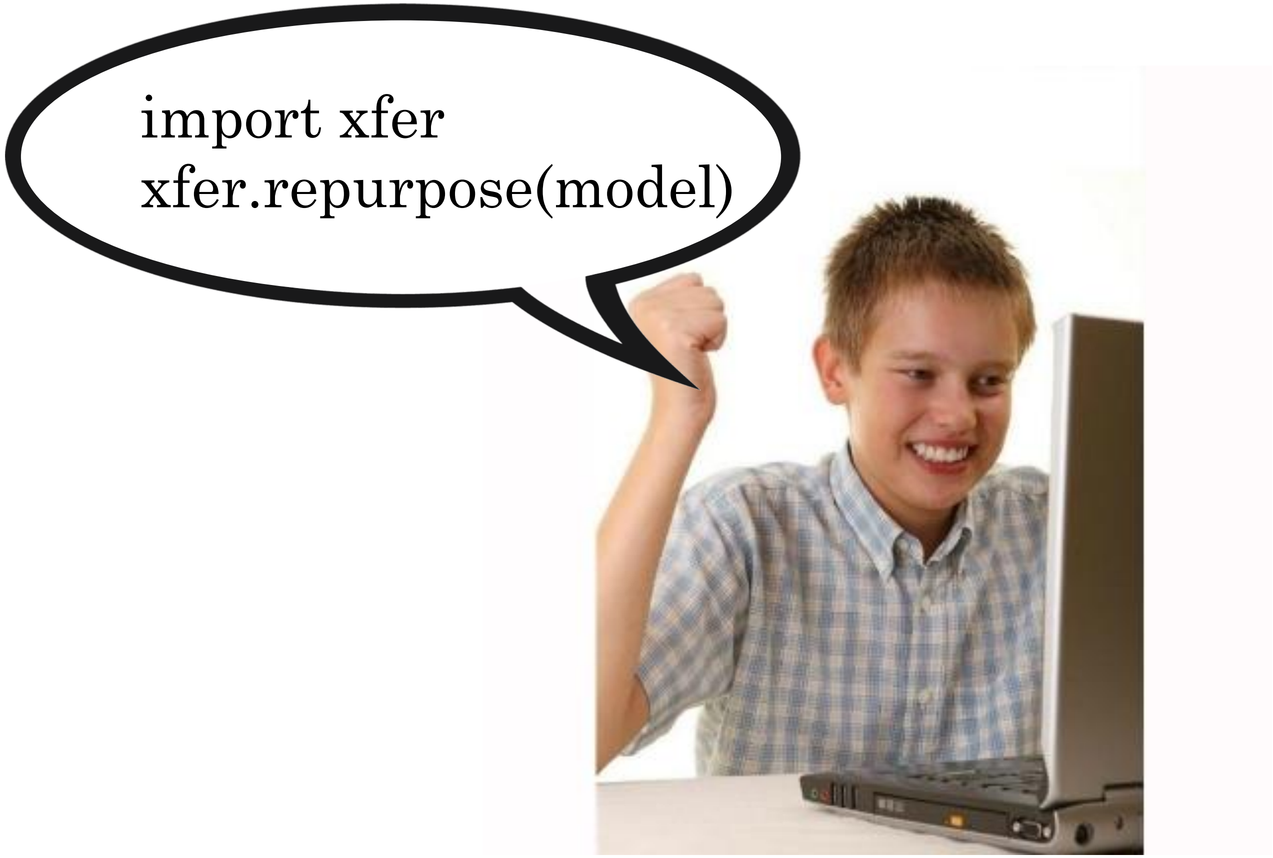
- NN: $\mathbf{H}_2 = \mathbf{W}_2\phi(\mathbf{H}_1)$
- GP: ϕ is $\infty-$dimensional so:
  $$\mathbf{H}_2 = f_2(\mathbf{H}_1; \theta_2) + \epsilon$$
- NN: $p(\mathbf{W})$
- GP: $p(f(\cdot))$

Deep Gaussian processes. A. Damianou, N. Lawrence, 2013

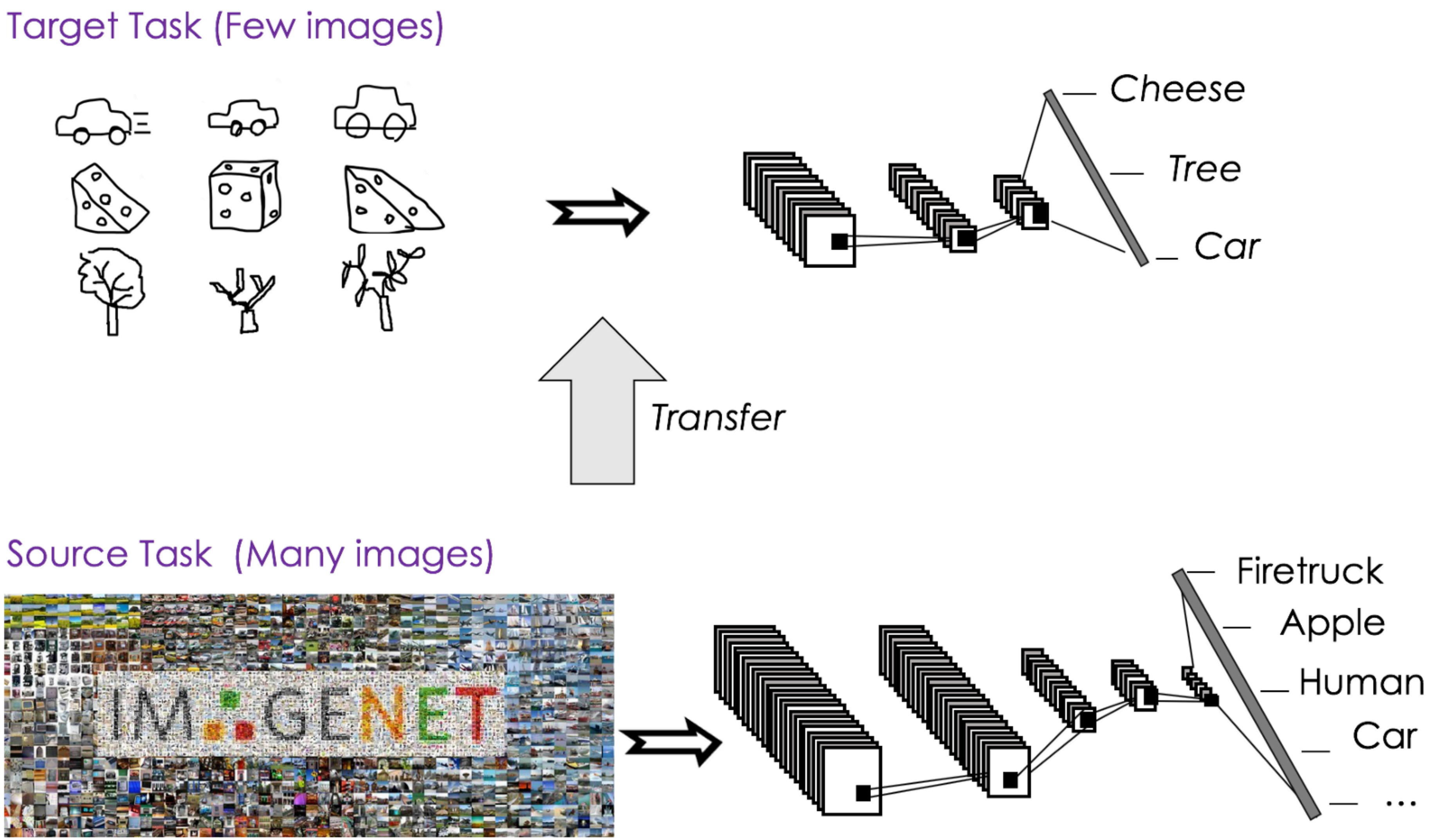
Transfer Learning

Motivations for TL: DNN training requires expertise

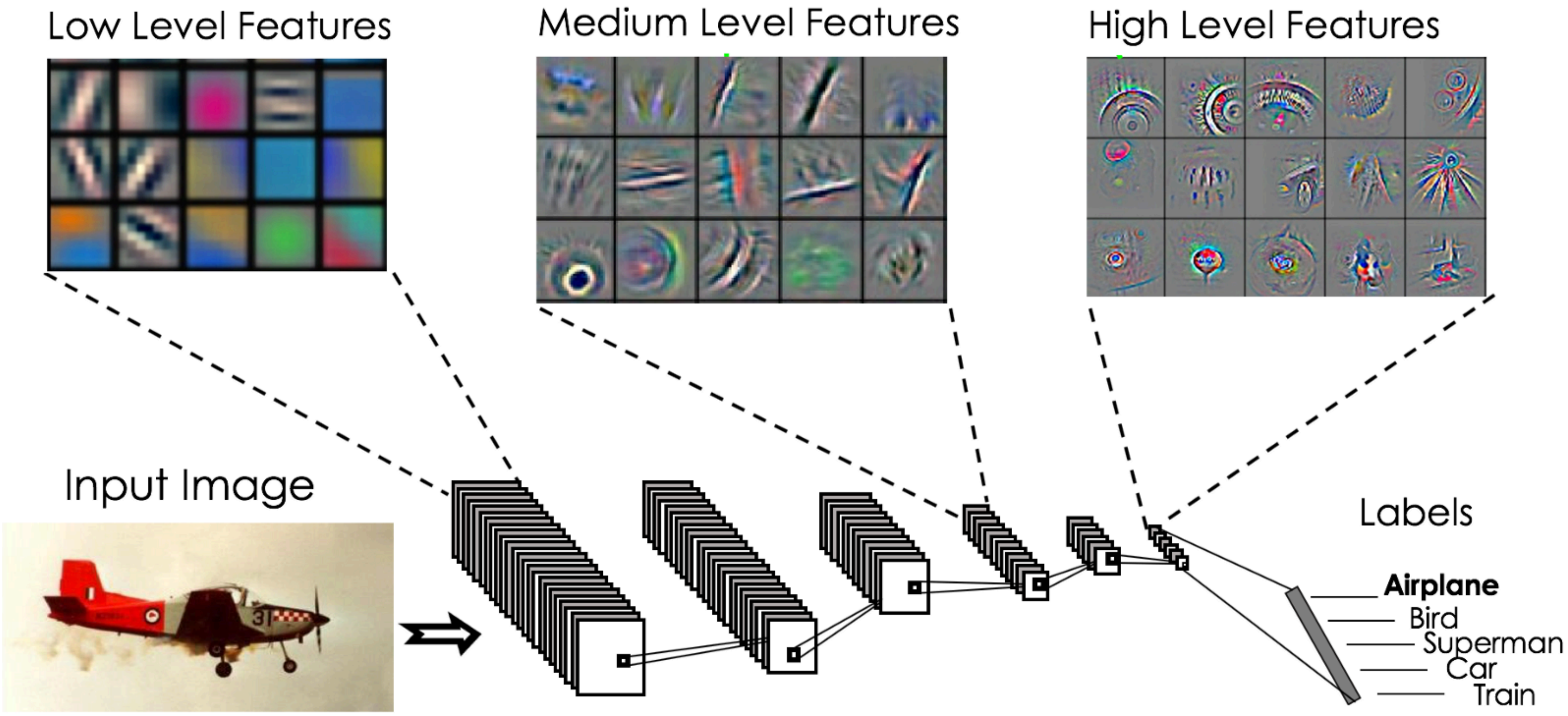
- Leveraging the power of DNNs even without too much expertise



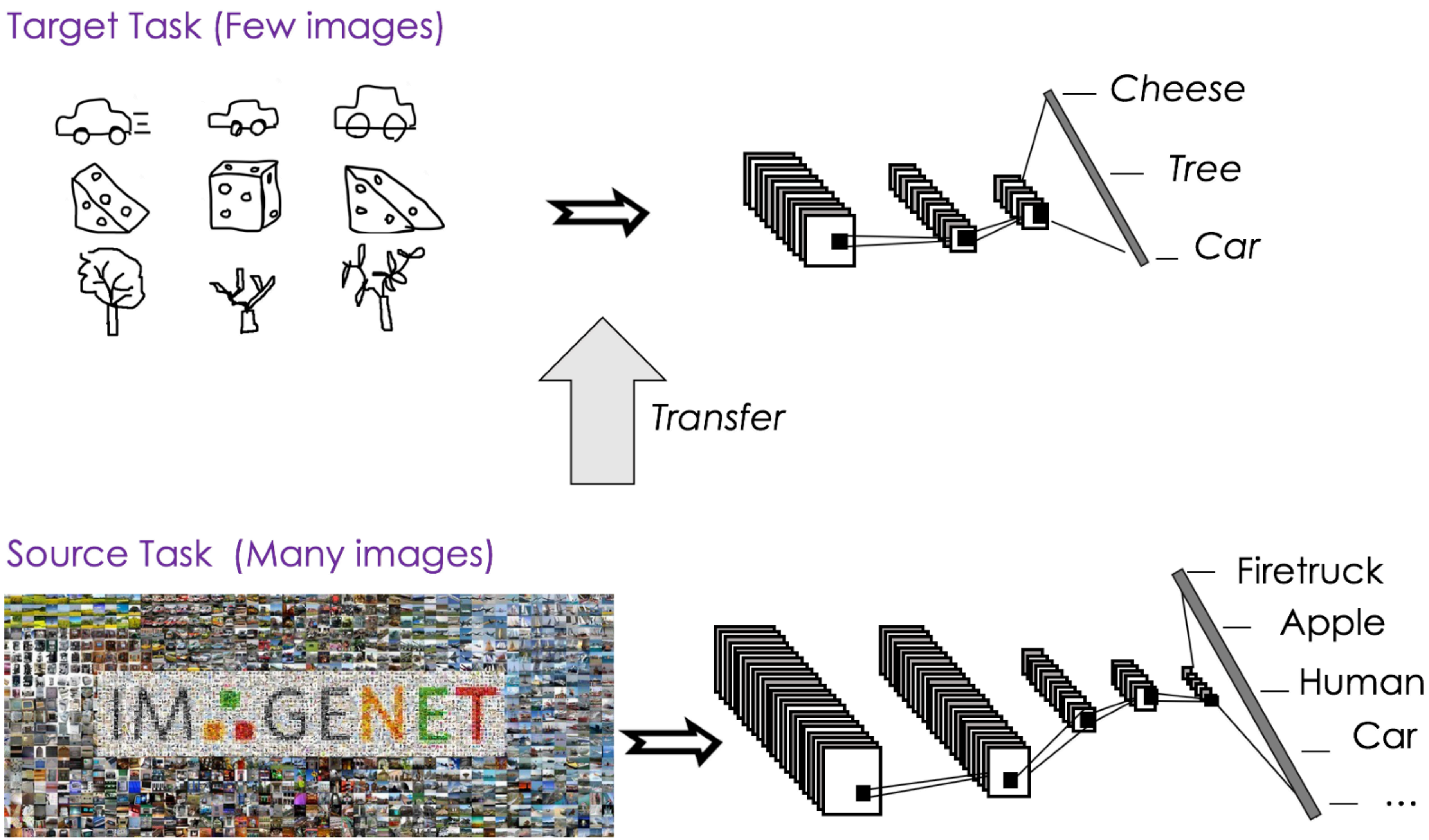
Motivations for TL: Leverage commonalities in data



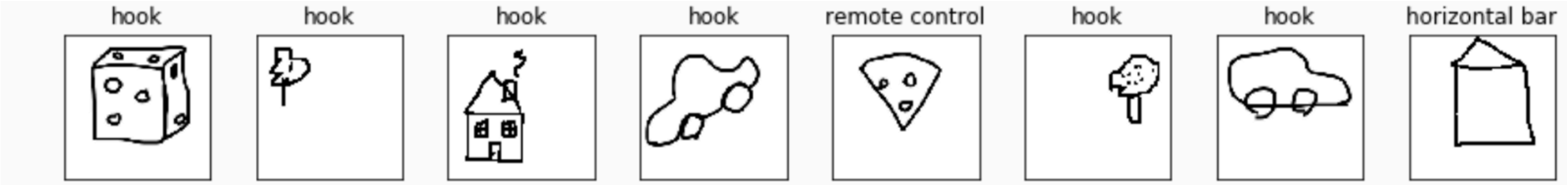
Why does Transfer Learning work?



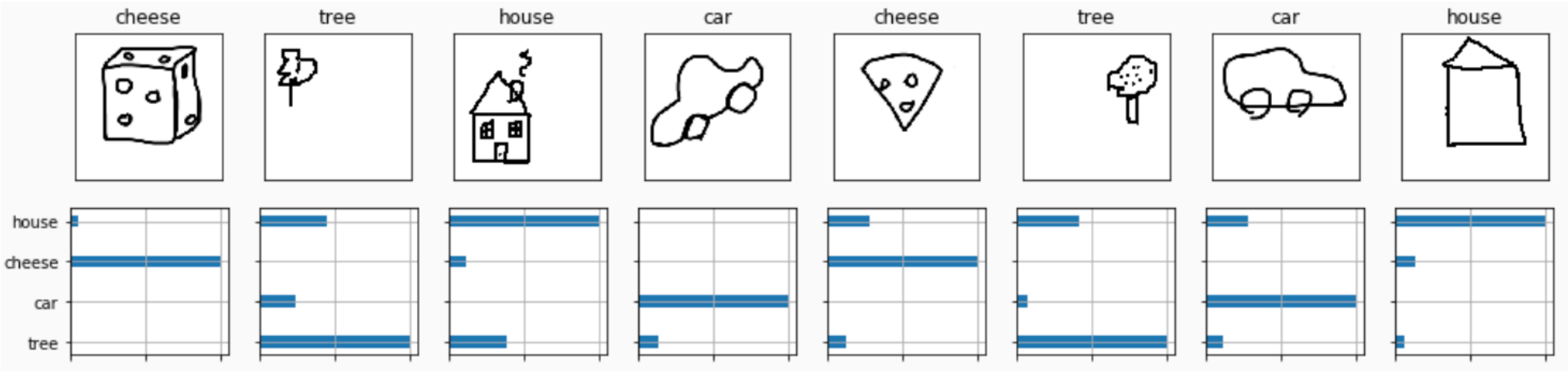
Back to our transfer example



Predictions using a pre-trained model (no transfer)



Predictions using Xfer



github.com/amzn/xfer

Deep Transfer Learning for MXNet

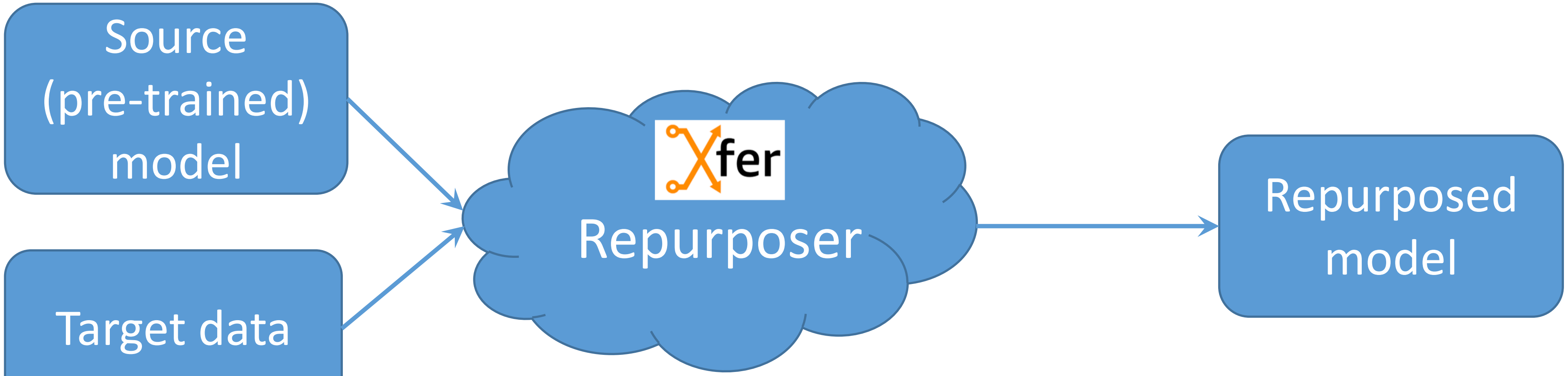
build passing | docs passing | codecov 96% | pypi v1.0.0 | license Apache-2.0

Website | Documentation | Contribution Guide

What is Xfer?

Xfer is a library that allows quick and easy transfer of knowledge[1,2,3] stored in deep neural networks implemented in MXNet. Xfer can be used with data of arbitrary numeric format, and can be applied to the common cases of image or text data.

Xfer Repurposers

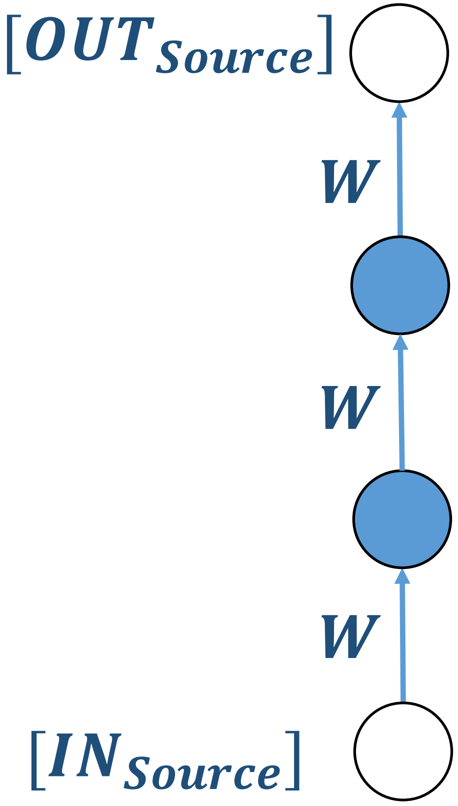


Three kinds of repurposers:

- Meta-model based
- Fine-tuning based
- Multi-task and meta-learning based (learning to learn)

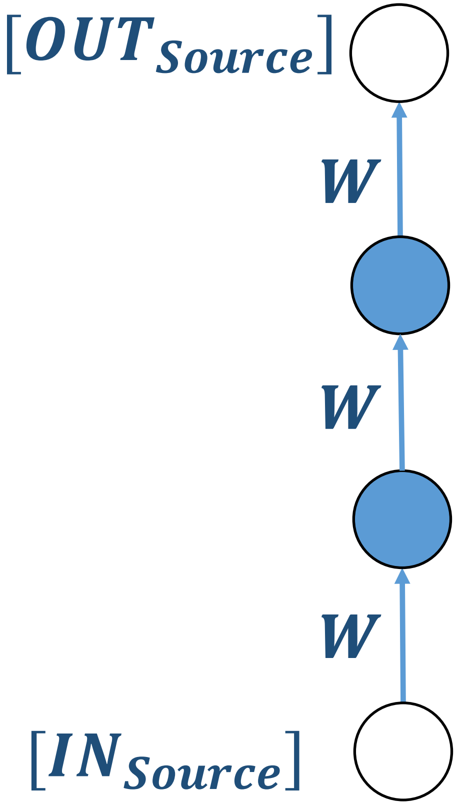
Meta-model based repurposing

Given:
(source task)

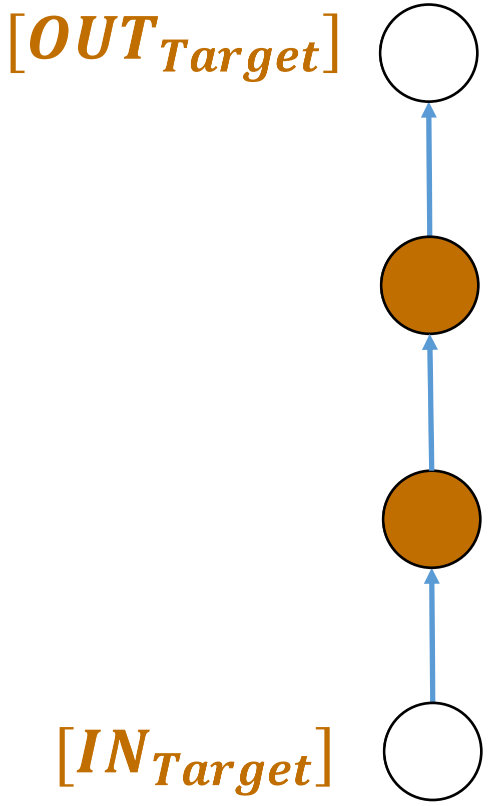


Meta-model based repurposing

Given:
(source task)

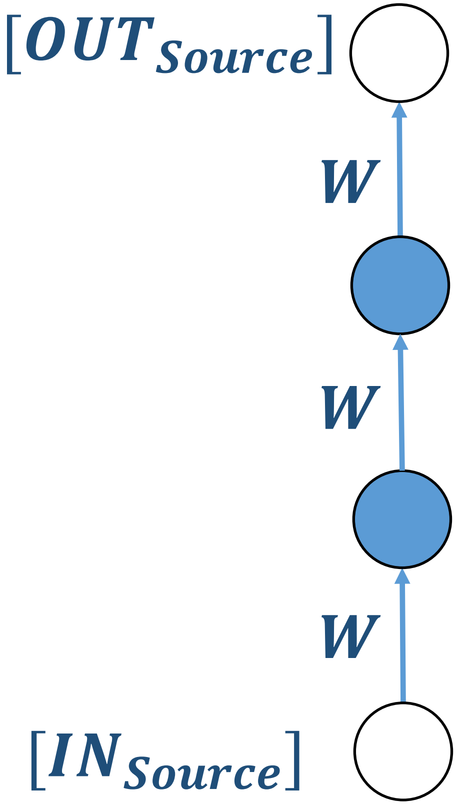


Step 1:
(target task)

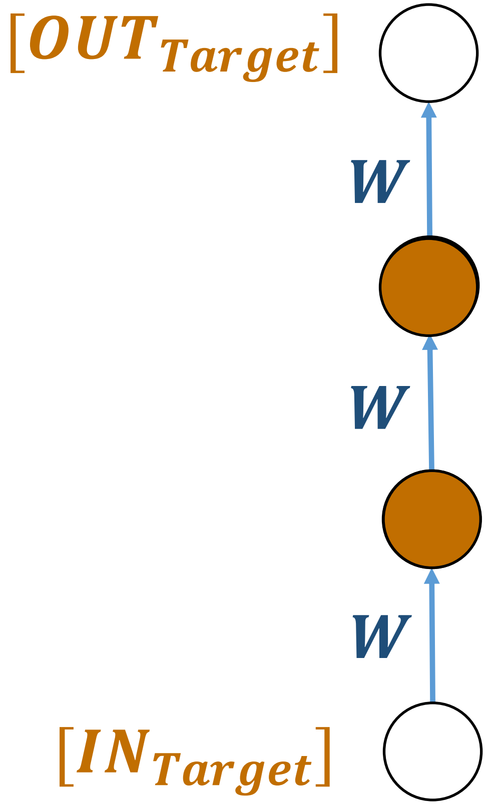


Meta-model based repurposing

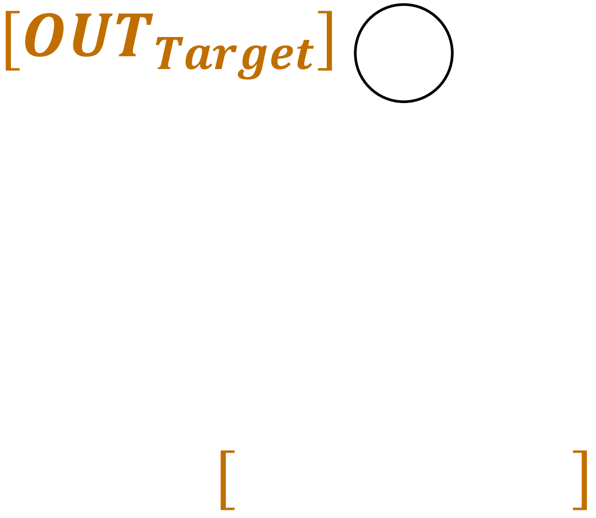
Given:
(**source task**)



Step 1:
(**target task**)

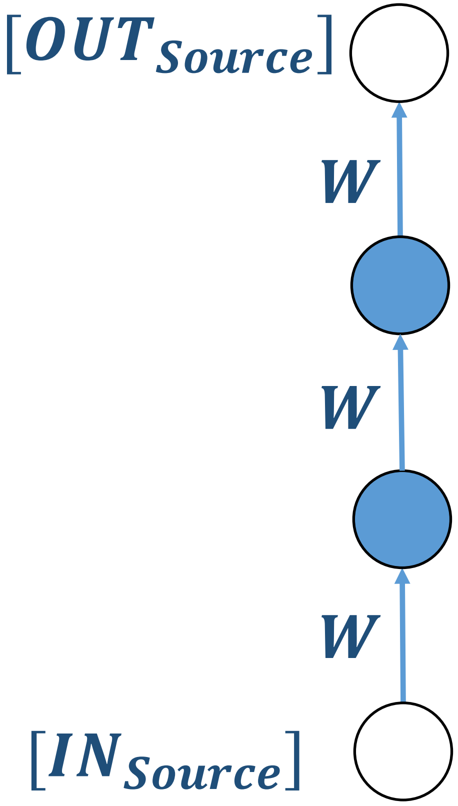


Step 2:
Meta-model

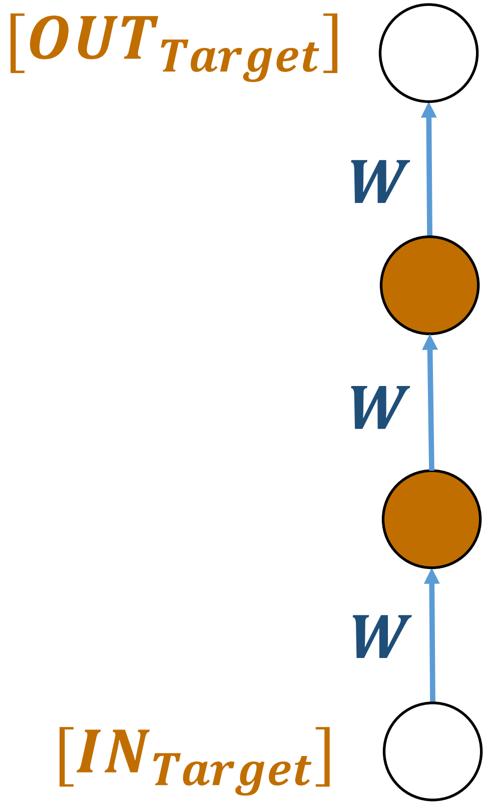


Meta-model based repurposing

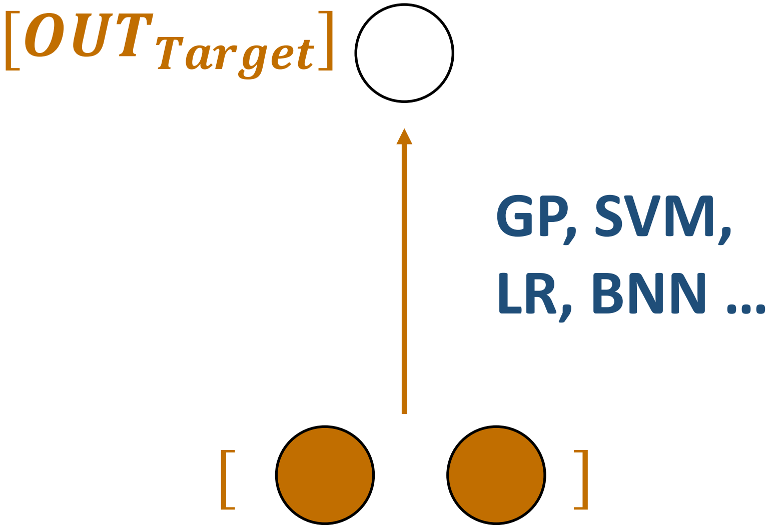
Given:
(**source task**)



Step 1:
(**target task**)



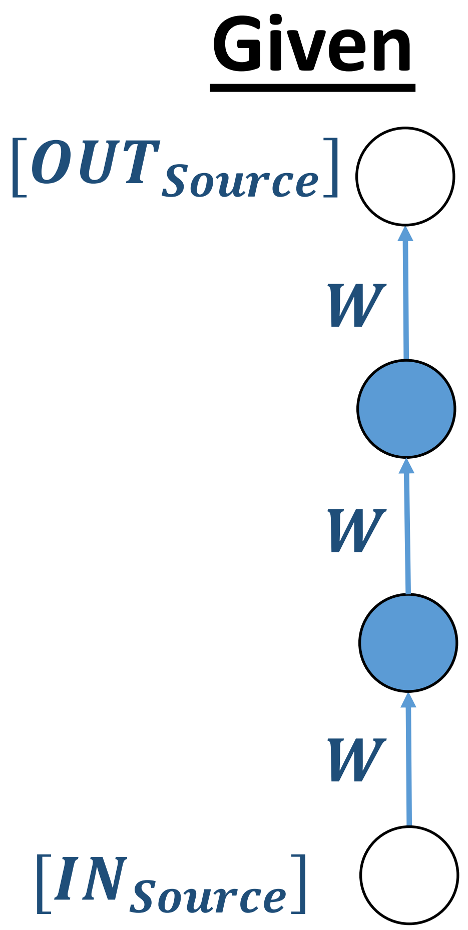
Step 2:
Meta-model



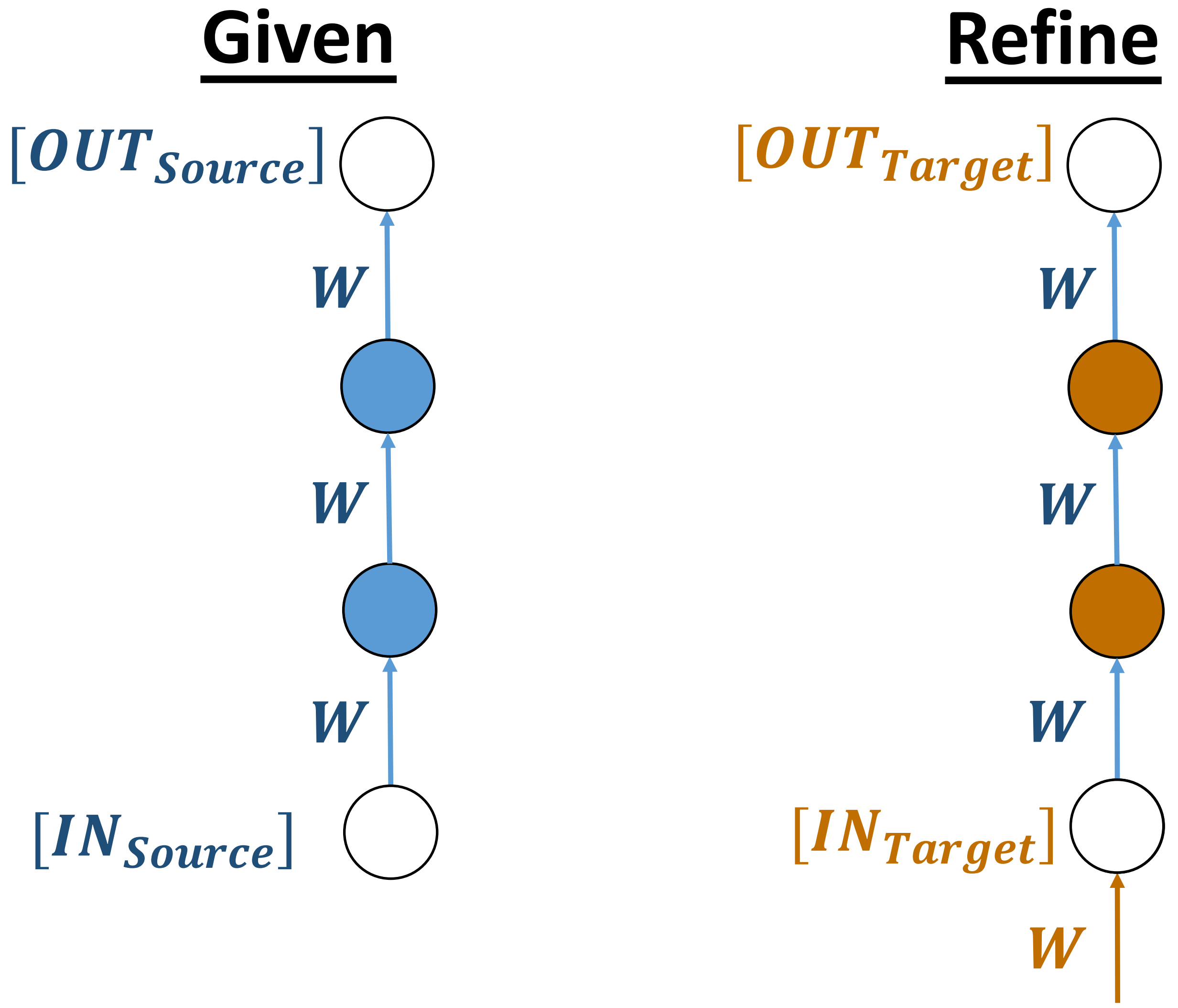
Meta-model based repurposing

```
repurposer = xfer.LrRepurposer(source_model, feature_layer_names=['fc2','fc3’])

repurposer.repurpose(train_iterator)

predictions = repurposer.predict_label(test_iterator)
```

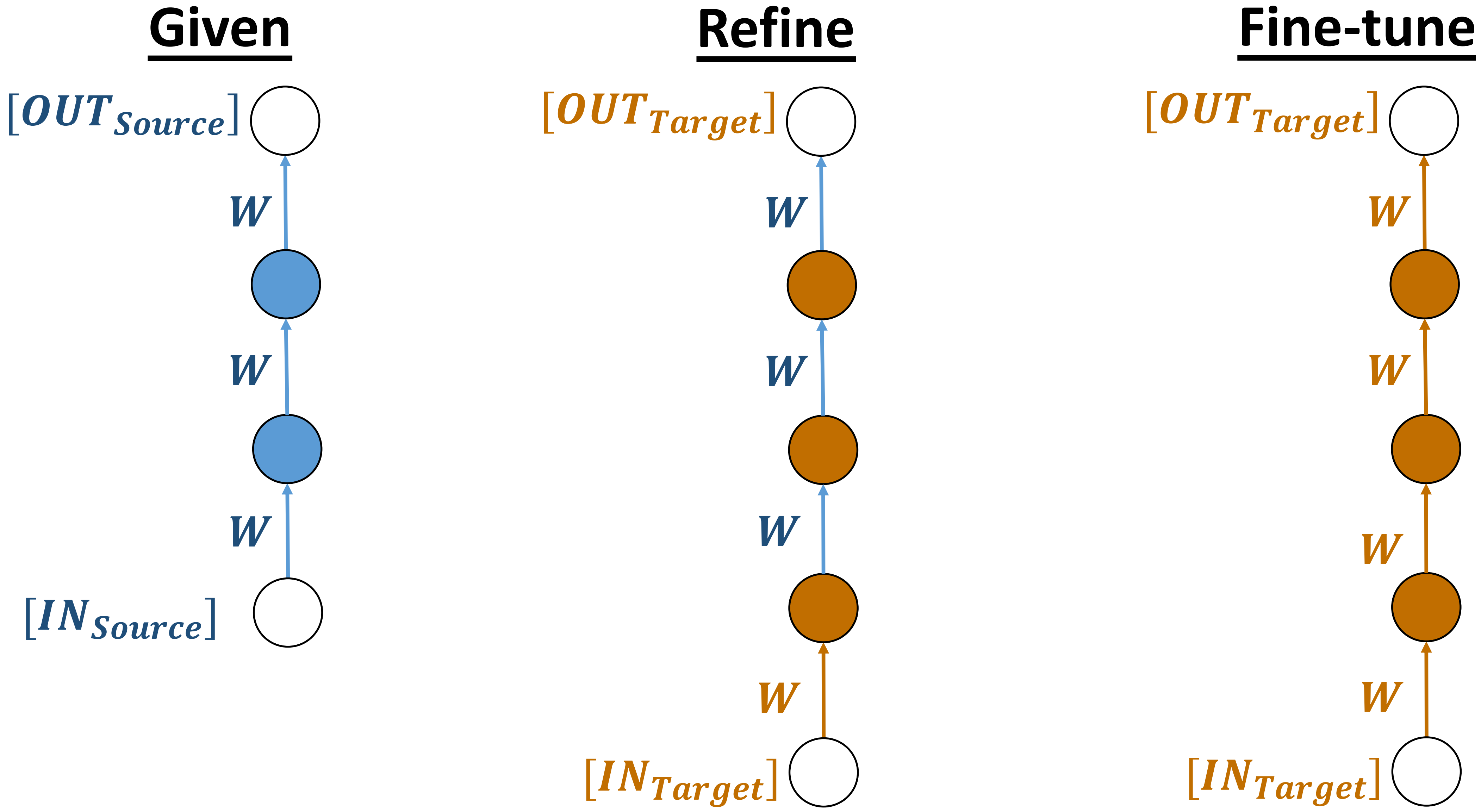
Fine-tuning based repurposing



Fine-tuning based repurposing



Fine-tuning based repurposing



Fine-tuning based repurposing

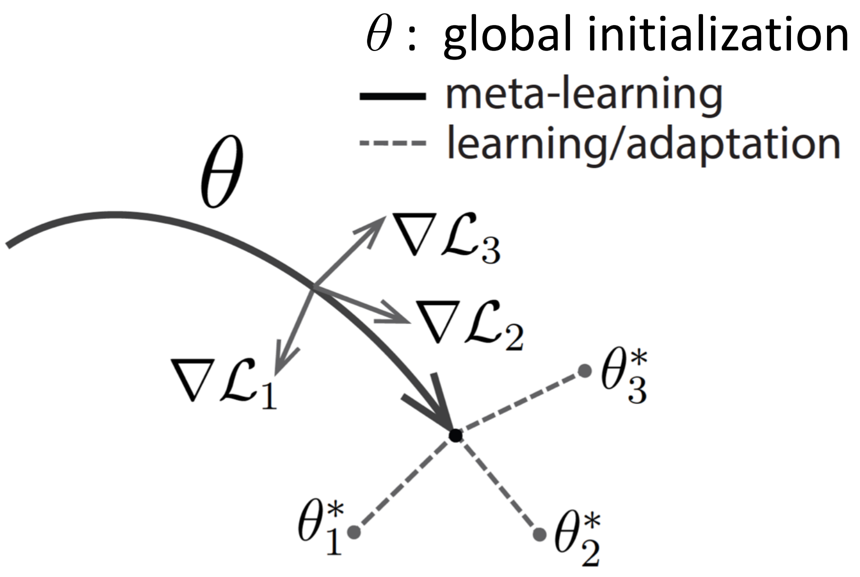
```
mh = xfer.model_handler.ModelHandler(source_model)

conv1 = mxnet.sym.Convolution(name='convolution1', kernel=(20,20), num_filter=64)

mh.add_layer_bottom([conv1])

mod = mh.get_module(iterator, fixed_layer_parameters=mh.get_layer_parameters(['conv1_1']),
        random_layer_parameters=mh.get_layer_parameters(['fc6', 'fc7’]))

mod.fit(iterator, num_epoch=5)
```

Transfer through meta-learning

- Learning to learn
- Related to multi-task learning
- Our approach: transfer knowledge across learning *processes*
  - Transfer learning in a higher level of abstraction
  - Transfer learning among typically many tasks
  - All task sub-models act as source and target models

Meta-learning or multi-task learning

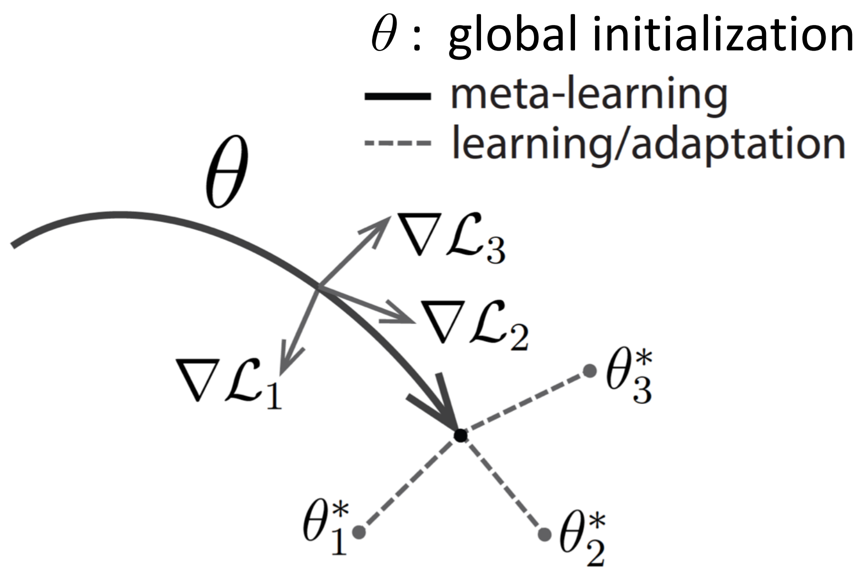
- Optimize θ such that on average θ_i^* are as best as possible.



MAML approach by Chelsea Finn et al. 2017

Meta-learning or multi-task learning

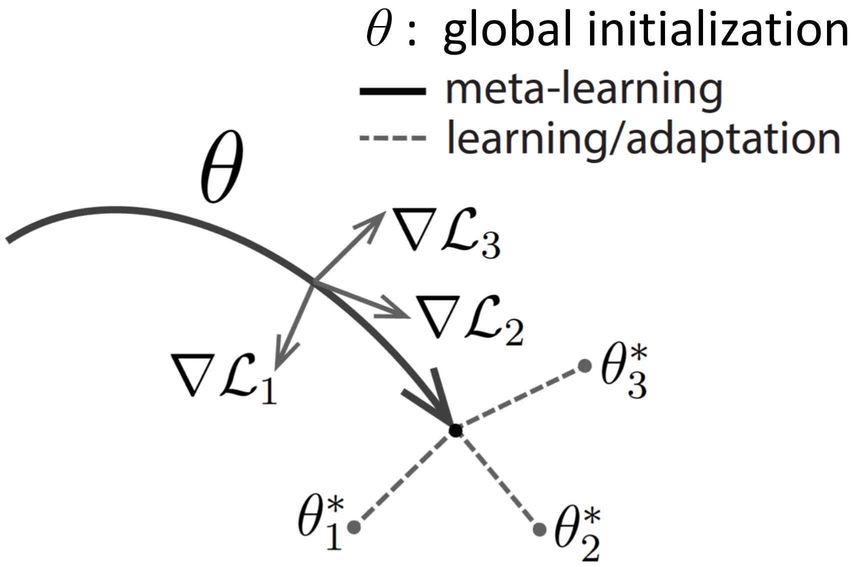
- Optimize θ such that on average θ_i^* are as best as possible.
- θ and θ_i^* are in the same space. So we can backprop.



MAML approach by Chelsea Finn et al. 2017

Meta-learning or multi-task learning

- Optimize θ such that on average θ_i^* are as best as possible.
- θ and θ_i^* are in the same space. So we can backprop.



MAML approach by Chelsea Finn et al. 2017

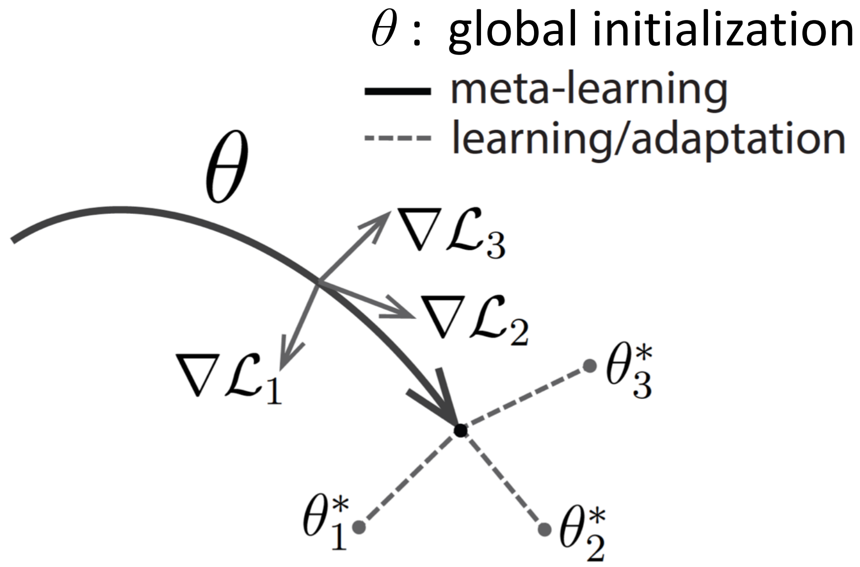
$$\min_{\theta} \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}\left(f_{\theta-\alpha\nabla_\theta \mathcal{L}_{\tau_i}(f_\theta)}\right)$$

Meta-learning optimization loop

- Start with initial θ
- for $meta\_steps = 1, 2....$:
  - Take a batch of instances per task
  - Update $\theta_1 , \theta_2 , ... \theta_\tau$ using each task's loss function individually
  - Update θ such that the average of all tasks' losses is minimized

Meta-learning or multi-task learning

- Optimize θ such that on average θ_i^* are as best as possible.
- θ and θ_i^* are in the same space. So we can backprop.

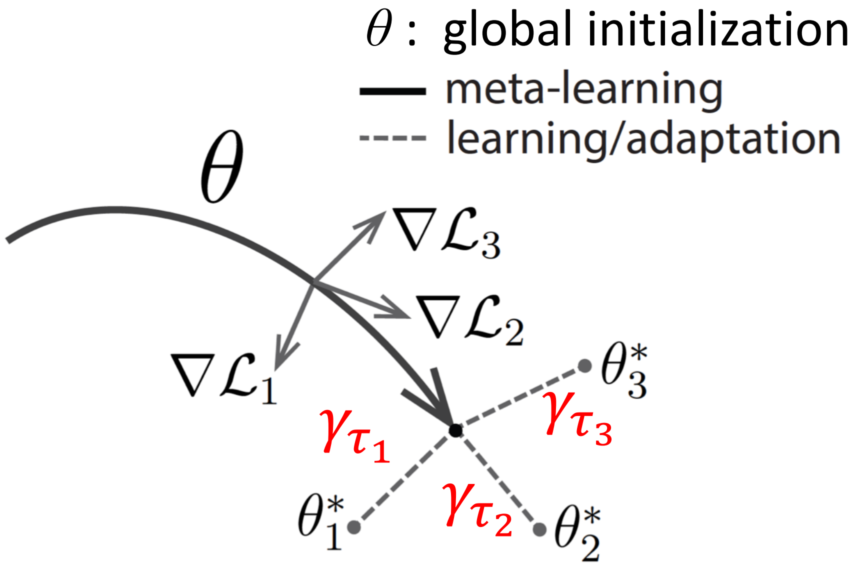


MAML approach by Chelsea Finn et al. 2017

$$\min_{\theta} \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}\left(f_{\theta-\alpha\nabla_\theta \mathcal{L}_{\tau_i}(f_\theta)}\right)$$

Meta-learning or multi-task learning

- Optimize θ such that on average θ_i^* is as best as possible and $\theta \rightarrow \theta_i^*$ is as short as possible.
- θ and θ_i^* are in the same space. So we can backprop.



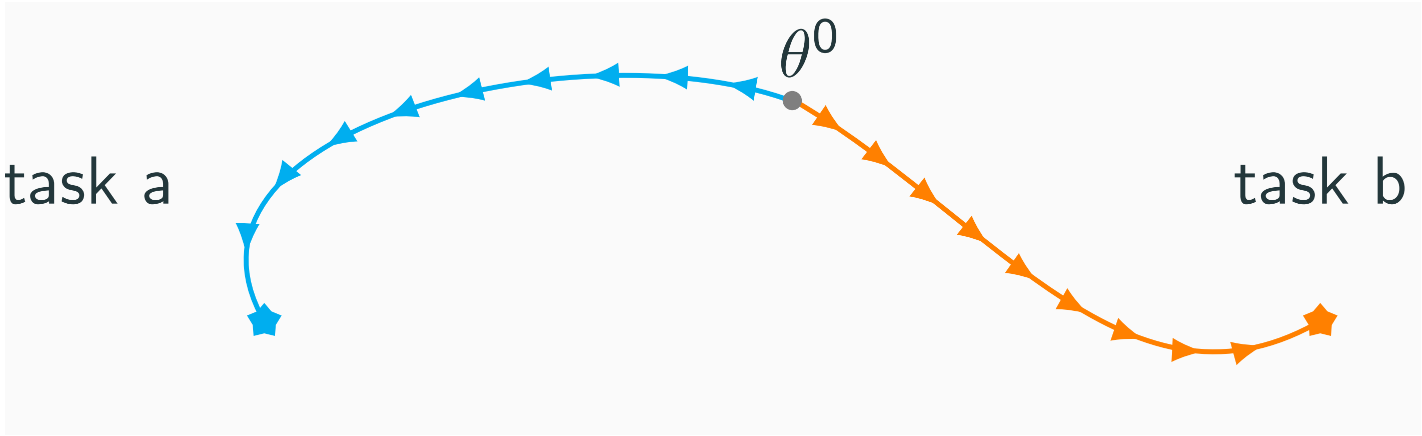
Leap approach by Flennerhag et al. 2019
(in ***Xfer*** *soon!)*

$$\min_{\theta} \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}\left(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\tau_i}(f_\theta)}\right) + \gamma_{\tau_i}(\theta)$$

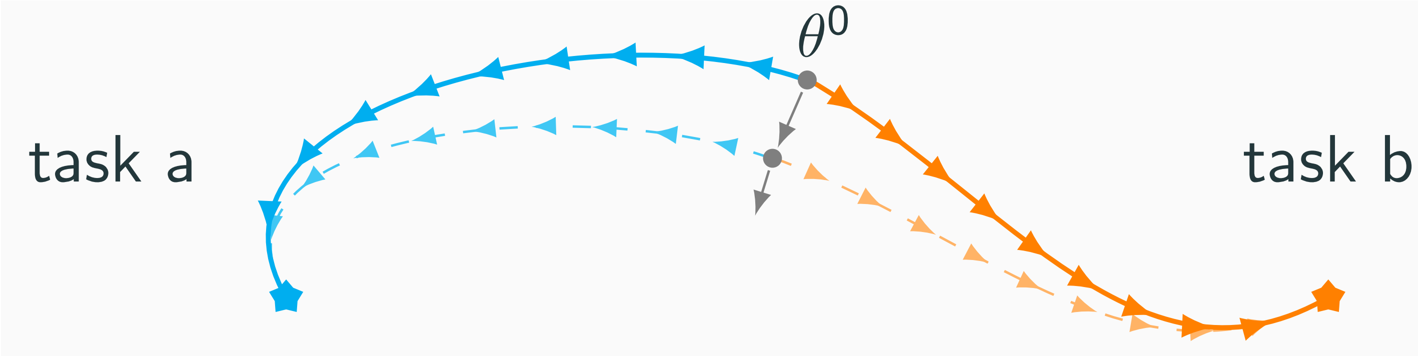
Leap balances gradient paths from all tasks…

… to minimize the expected gradient path.

Meta-step 1



Meta-step 2



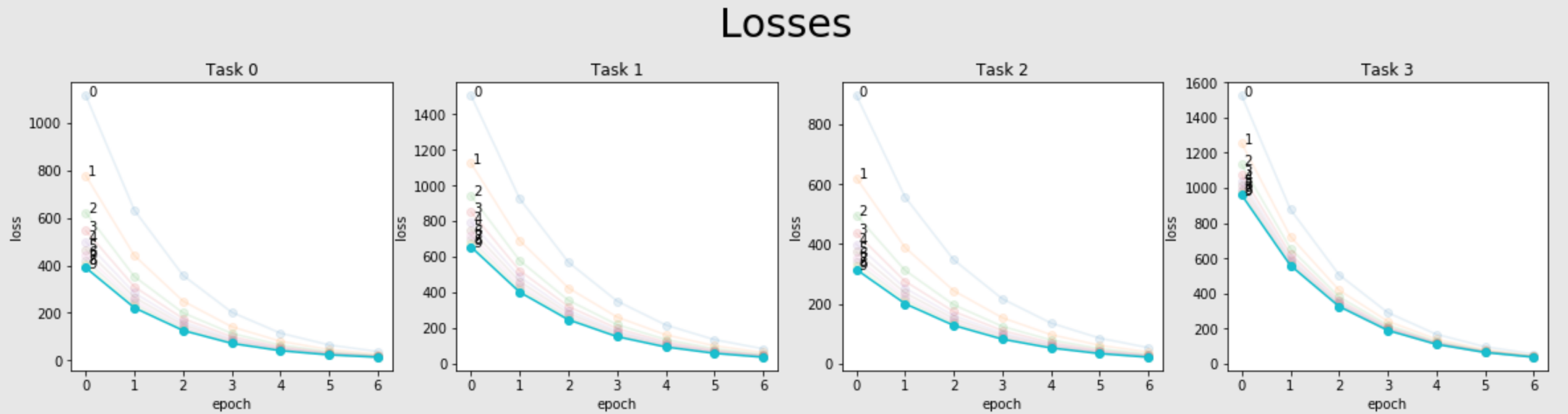
Xfer meta-learning *(available soon!)*

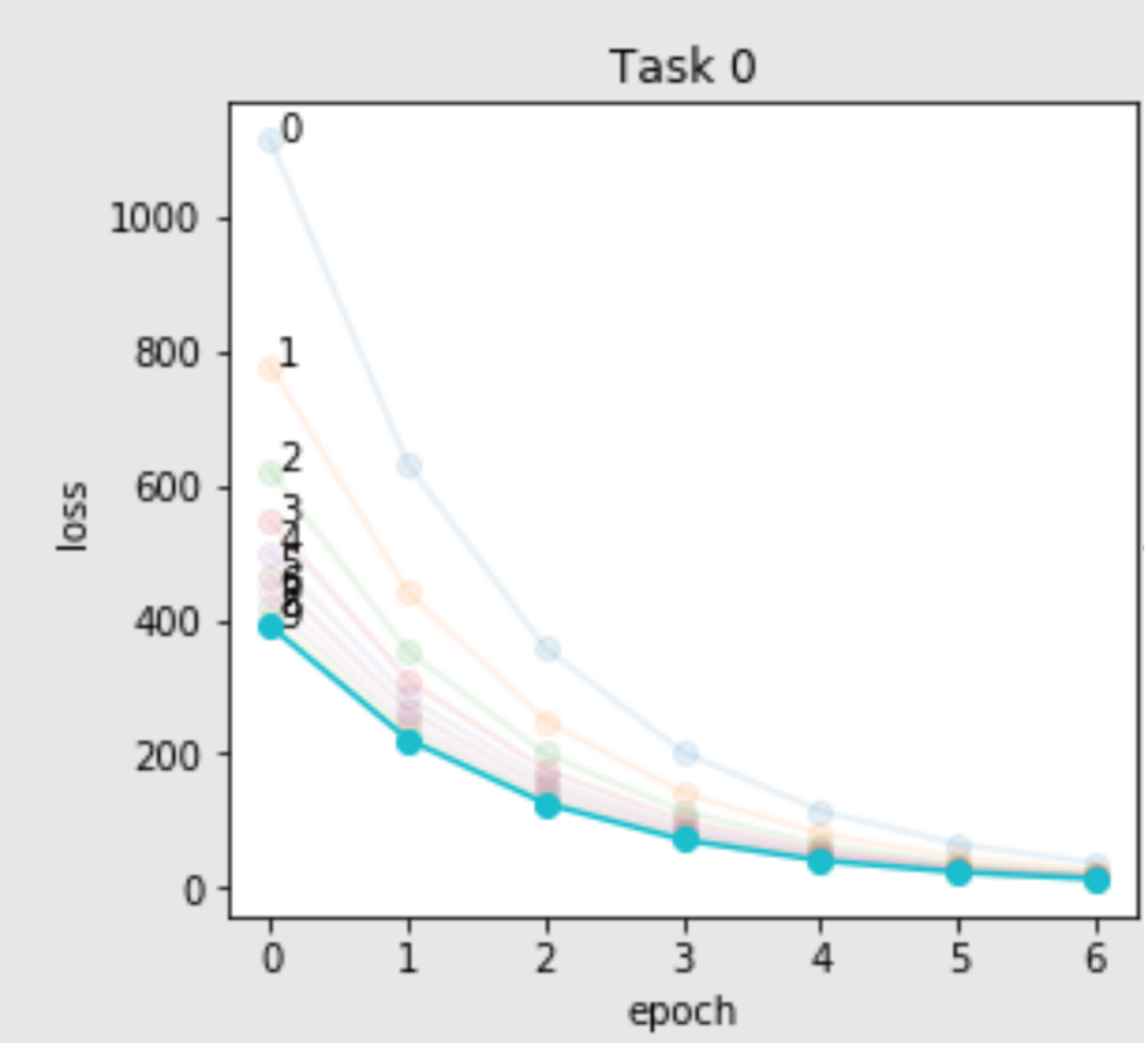
```
import xfer.contrib.xfer_leap as leap

lmr = leap.leap_meta_repurposer.LeapMetaRepurposer(model, num_meta_steps, num_epochs)

lmr.repurpose(train_data_all)
```

```
Metastep: 0, Num tasks: 4, Mean Loss: 57.061
        Metastep: 1, Task: 0, Initial Loss: 778.318, Final Loss: 25.655, Loss delta: -752.663
        Metastep: 1, Task: 1, Initial Loss: 1123.906, Final Loss: 60.993, Loss delta: -1062.913
        Metastep: 1, Task: 2, Initial Loss: 620.399, Final Loss: 38.558, Loss delta: -581.841
        Metastep: 1, Task: 3, Initial Loss: 1251.979, Final Loss: 46.972, Loss delta: -1205.006
```

```
Metastep: 8, Num tasks: 4, Mean Loss: 27.376
        Metastep: 9, Task: 0, Initial Loss: 389.985, Final Loss: 13.036, Loss delta: -376.949
        Metastep: 9, Task: 1, Initial Loss: 654.023, Final Loss: 34.885, Loss delta: -619.138
        Metastep: 9, Task: 2, Initial Loss: 314.407, Final Loss: 21.424, Loss delta: -292.983
        Metastep: 9, Task: 3, Initial Loss: 958.127, Final Loss: 37.829, Loss delta: -920.299
```

```
lmr.meta_logger.plot_losses()
```




Task 0
loss
epoch
1000
800
600
400
200
0
0
1
2
3
4
5
6

Data properties considerations

Source task: $\boldsymbol{X}_S \xrightarrow{Model_S} \boldsymbol{Y}_S$

Target task: $\boldsymbol{X}_T \xrightarrow{Model_T} \boldsymbol{Y}_T$

Transfer learning: Use $Model_S$ to improve $Model_T$

Setting	**Description**	**Considerations**
$\mathcal{X}_S \neq \mathcal{X}_T$	Different input domains	Domain adaptation
$\mathcal{Y}_S \neq \mathcal{Y}_T$	Different label spaces	Multi-task learning might be preferable
$p(\mathbf{Y}_S) \neq p(\mathbf{Y}_T)$	Dissimilar output distribution	Transferring lower layers preferable
$p(\mathbf{X}_S) \neq p(\mathbf{X}_T)$	Dissimilar input distribution	Transferring higher layers preferable
$\|\mathbf{Y}_T\| \ll \|\mathbf{Y}_S\|$	Much fewer labelled data in T	Data efficient TL required
$\|\mathbf{Y}_T\| \gg \|\mathbf{Y}_S\|$	Much fewer labelled data in S	Take care of catastrophic forgetting or train T from scratch

Conclusions

- NNs are mathematically simple; challenge is how to optimize them.
- Data efficiency? Uncertainty Calibration? Interpretability? Safety?
- Bayesian NNs solve *some* of the above.
- Repurposing neural networks is more practical.
- Xfer: library for automatic repurposing

Acknowledgements

- Jordan Massiah
- Keerthana Elango
- Pablo Garcia Moreno
- Nikos Aletras
- Sebastian Flennerhag

Further Resources

- Notebook: adamian.github.io/talks/Damianou_DL_tutorial_19.ipynb
- Xfer: github.com/amzn/xfer/
- Blog: link.medium.com/De5BXPJ9TT
- A more complete tutorial on deep learning: adamian.github.io/talks/Damianou_deep_learning_rss_2018.pdf