Deep transfer learning with Xfer

Andreas Damianou

Amazon, Cambridge UK

Outline

- Deep neural networks quick reminder
- Transfer learning intro
- Xfer
  - Transfer learning via meta-learning
- Considerations

Resources

- Notebook:
  adamian.github.io/talks/Damianou_DL_Xfer.ipynb

- A more complete tutorial on deep learning:
  adamian.github.io/talks/Damianou_deep_learning_rss_2018.pdf

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:

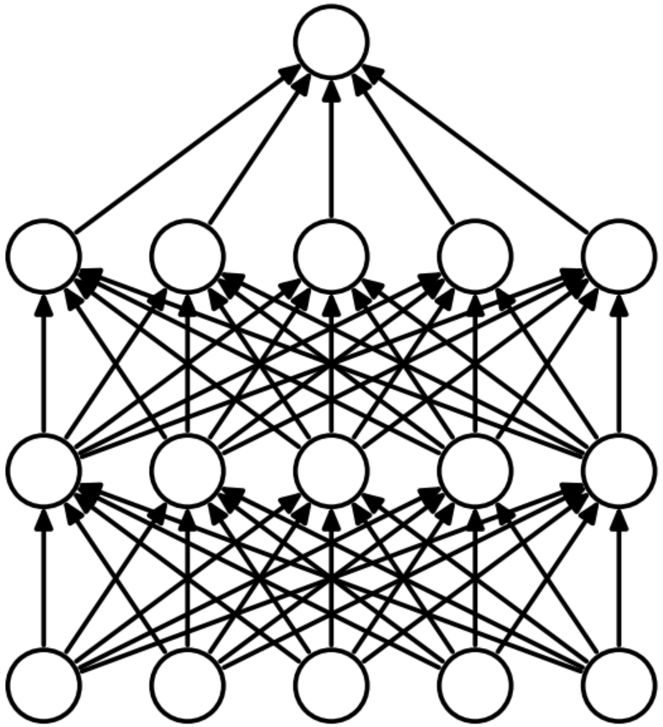
$$\begin{aligned}
\mathbf{h}_1 &= \varphi(\mathbf{x}\mathbf{w}_1 + b_1) \\
\mathbf{h}_2 &= \varphi(\mathbf{h}_1\mathbf{w}_2 + b_2) \\
&\dots \\
\hat{\mathbf{y}} &= \varphi(\mathbf{h}_{L-1}\mathbf{w}_L + b_L)
\end{aligned}$$

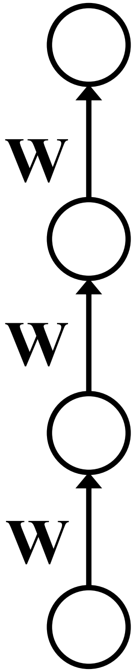
ϕ is the (non-linear) activation function.

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

Graphical depiction



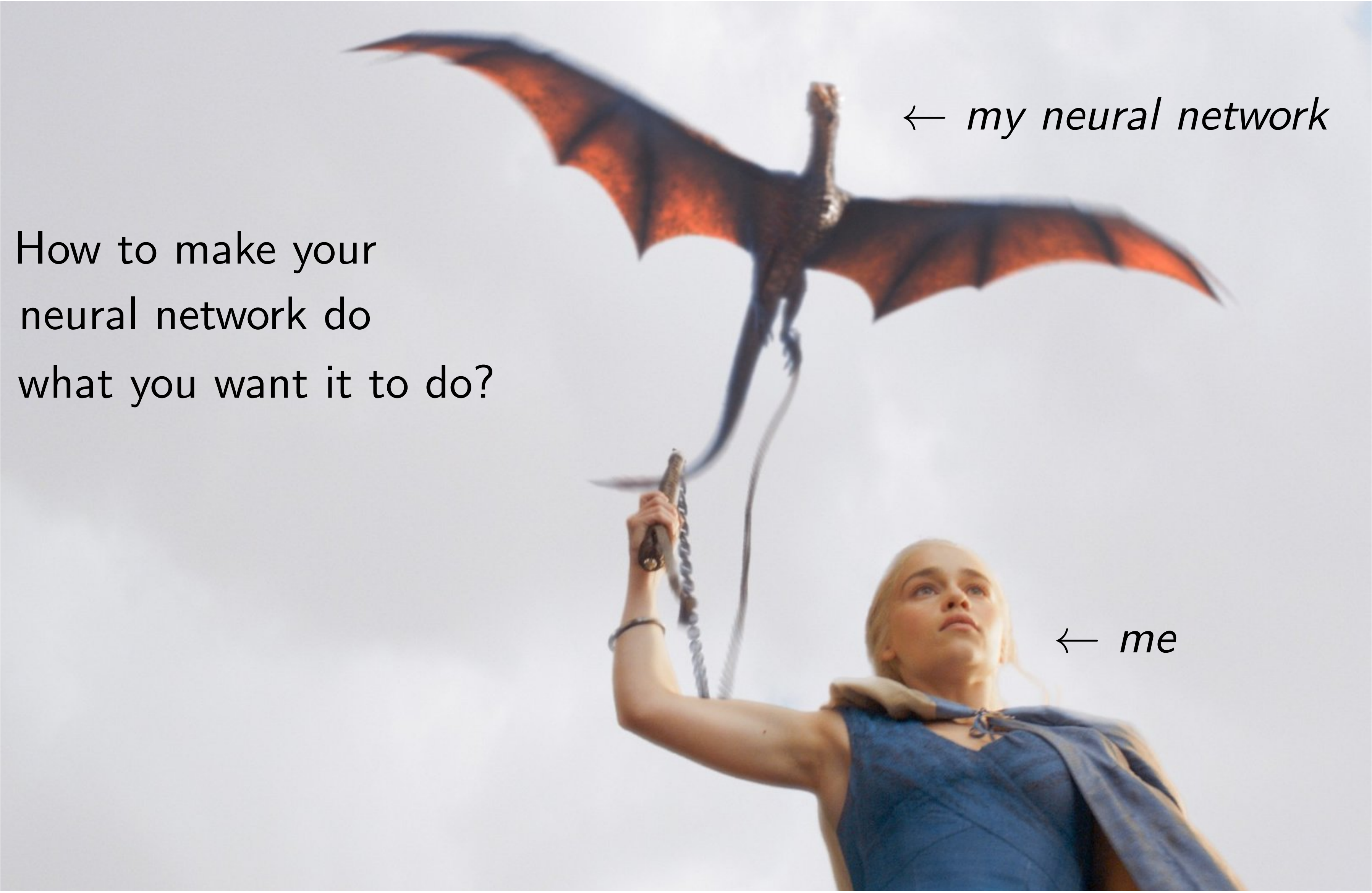


Optimization and implementation

- Optimization done with back-propagation, based on the chain rule

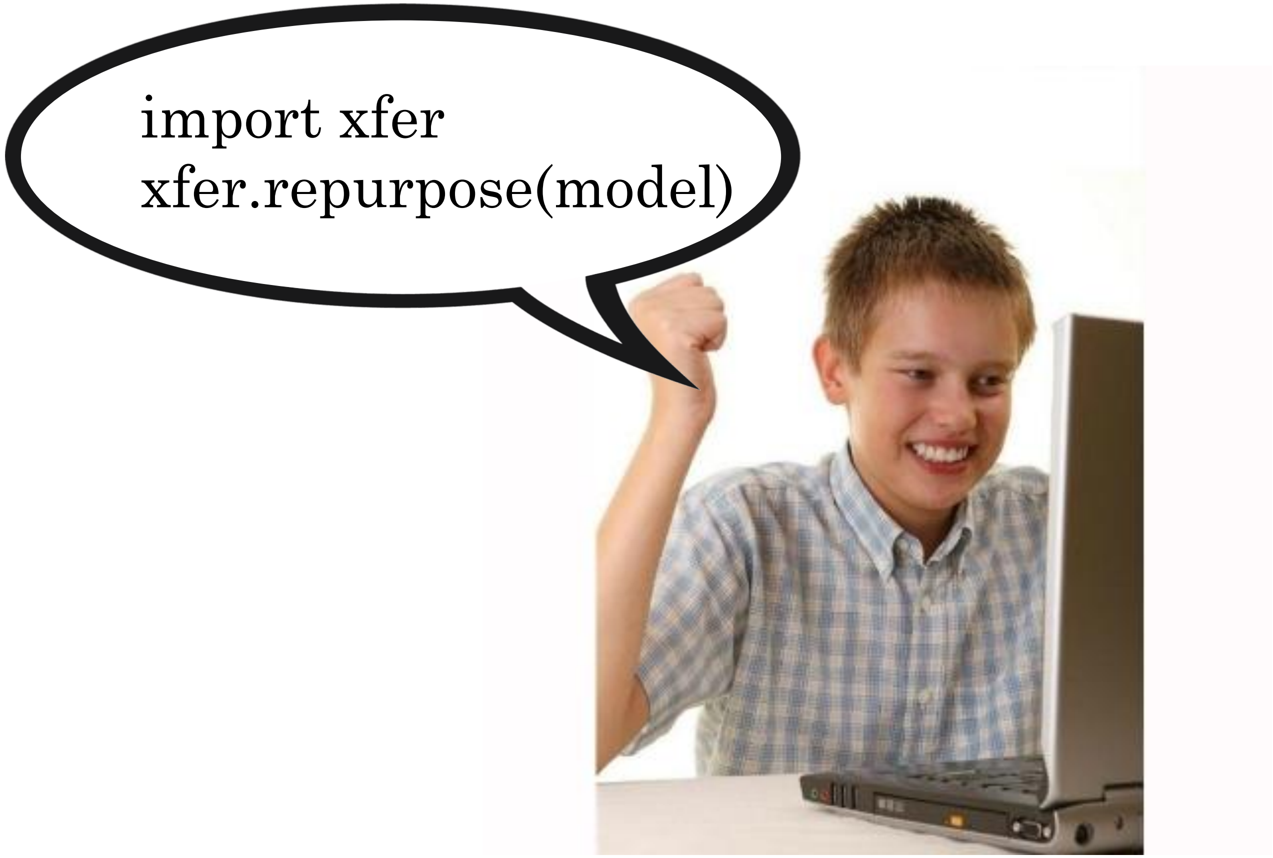
GOTO notebook!!

Taming the dragon

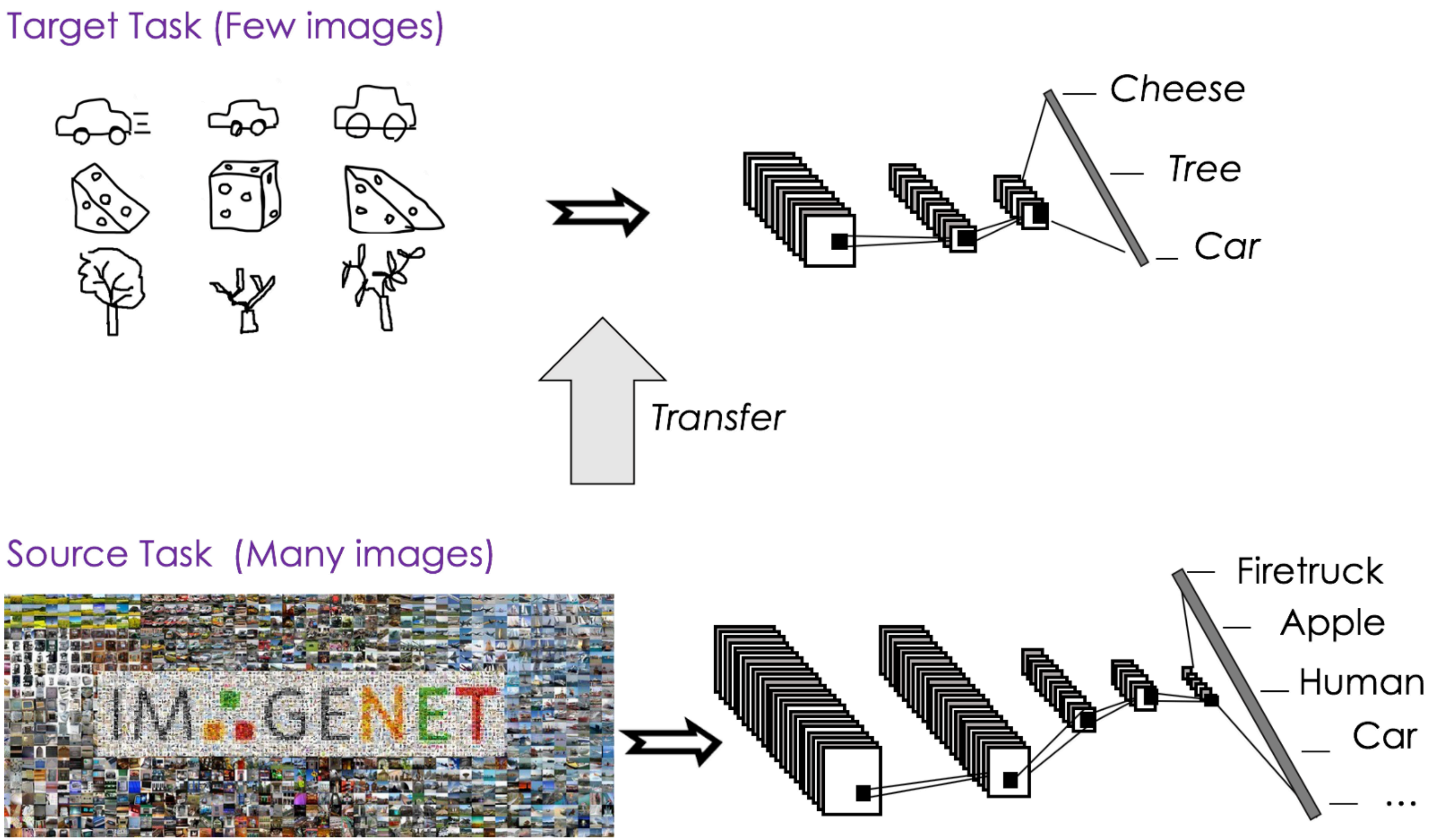


Motivations for TL: DNN training requires expertise

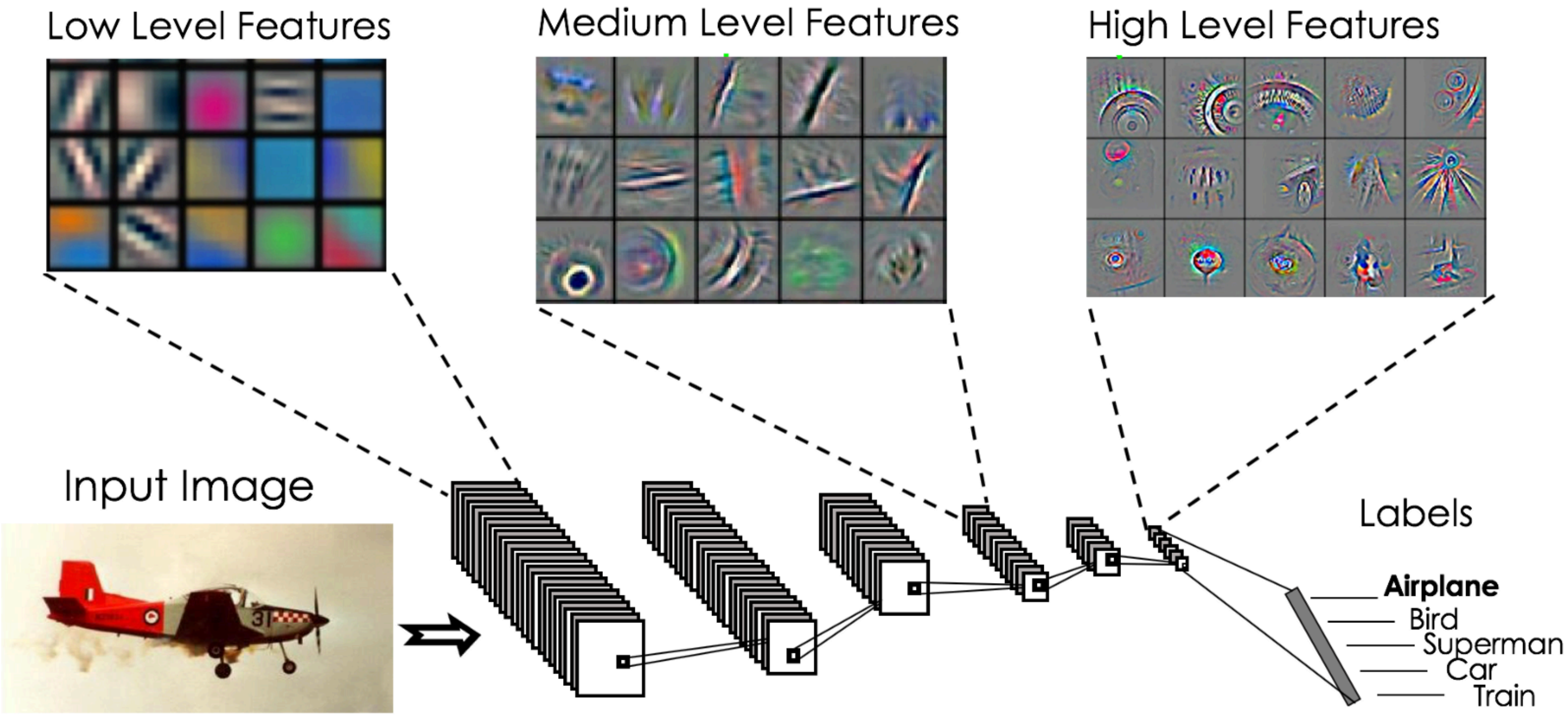
- Leveraging the power of DNNs even without too much expertise



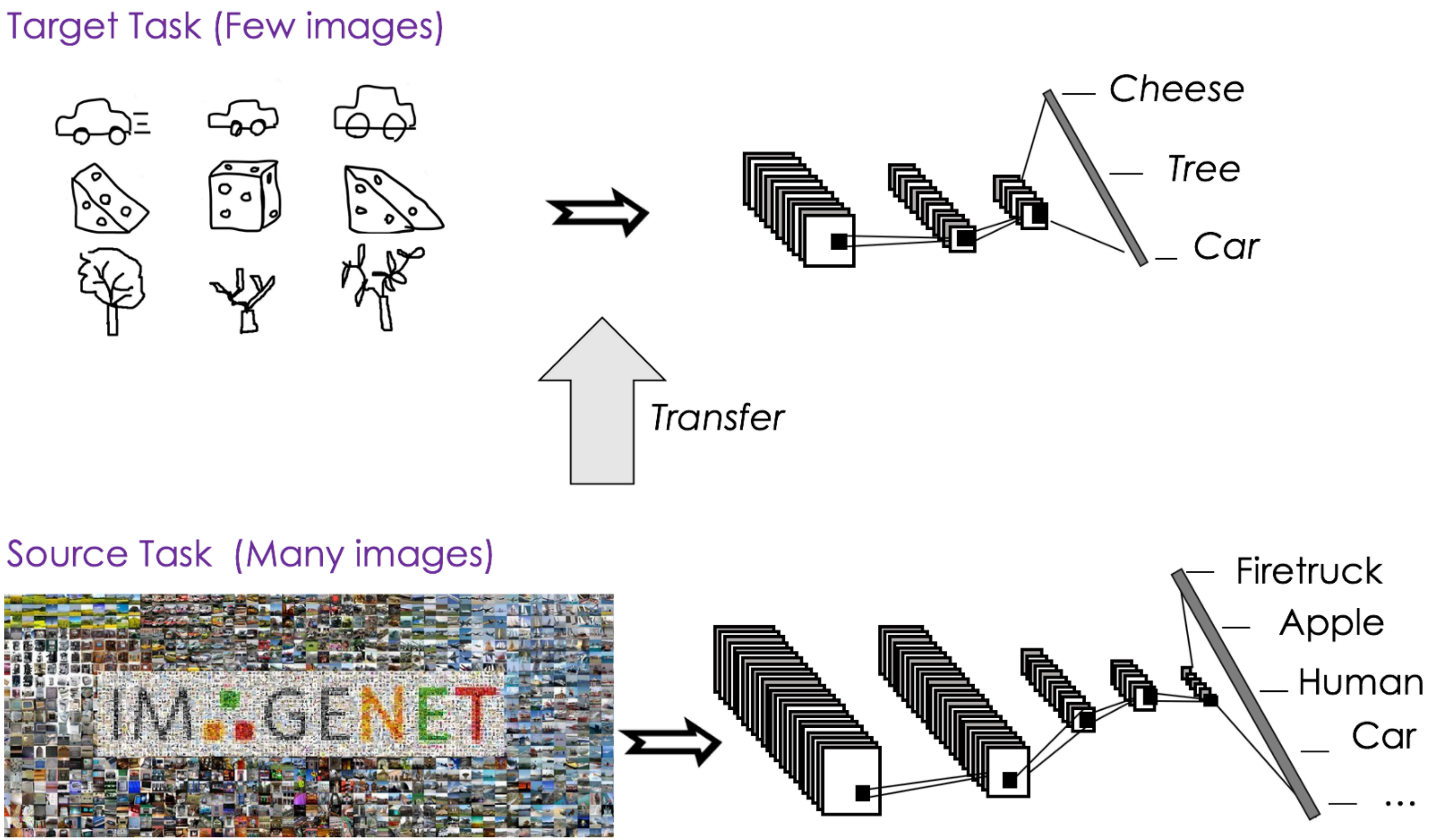
Motivations for TL: Leverage commonalities in data



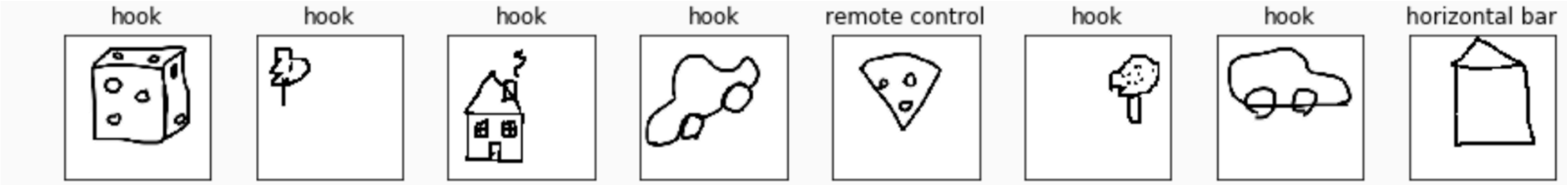
Why does Transfer Learning work?



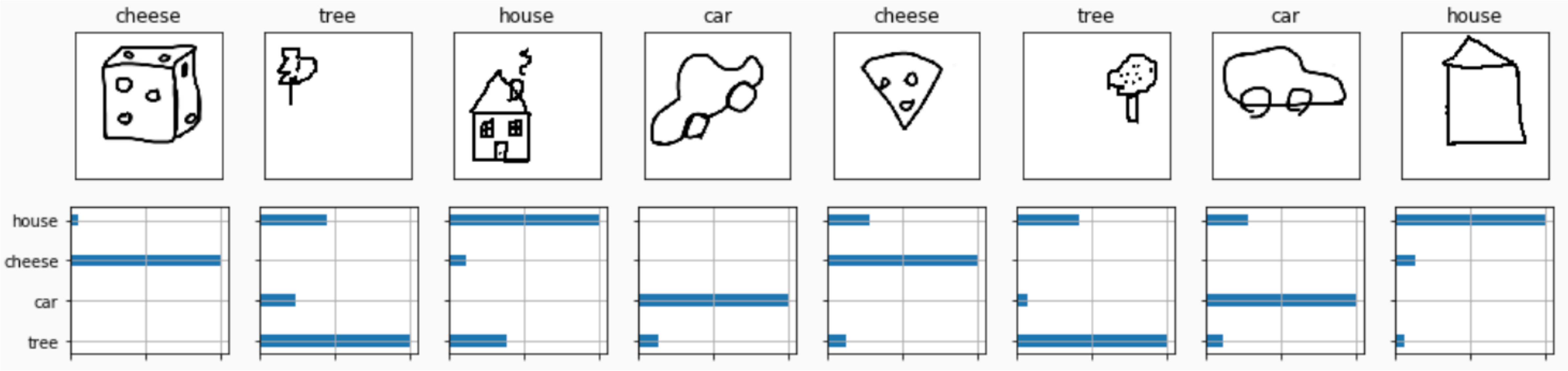
Back to our transfer example



Predictions using a pre-trained model (no transfer)



Predictions using Xfer



github.com/amzn/xfer

Xfer

Deep Transfer Learning for MXNet

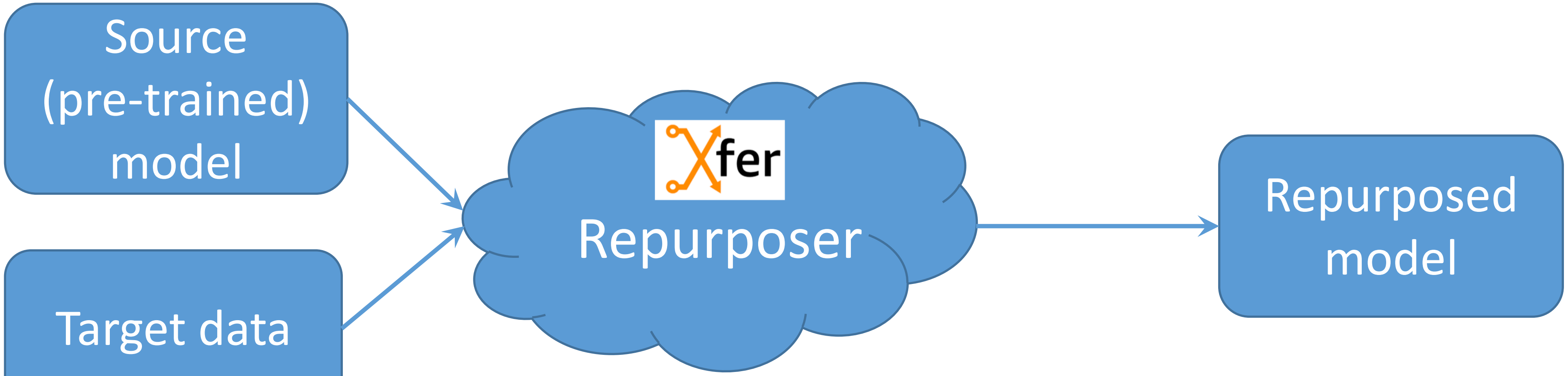
build passing | docs passing | codecov 96% | pypi v1.0.0 | license Apache-2.0

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

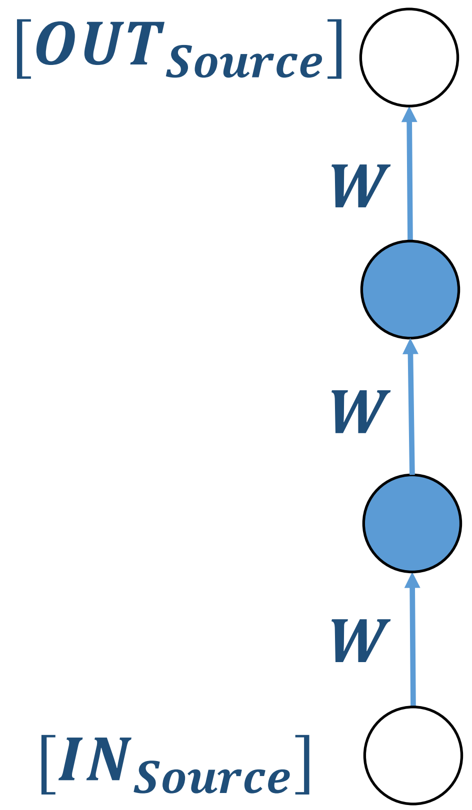


Two kinds of repurposers:

- Meta-model based
- Fine-tuning based

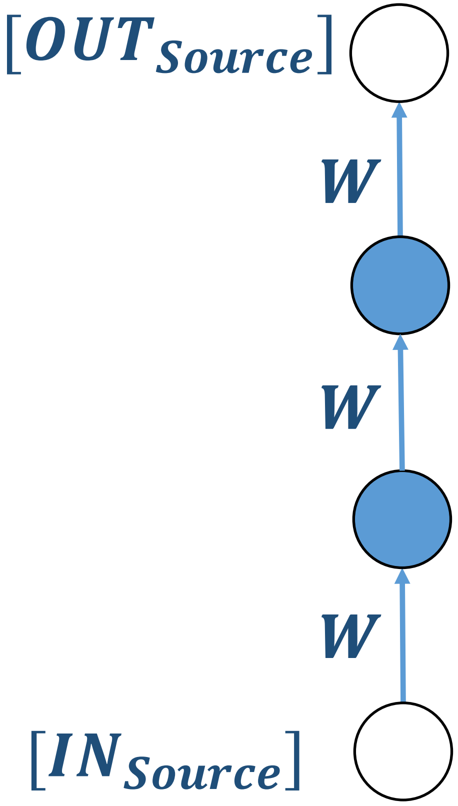
Meta-model based repurposing

Given:
(source task)

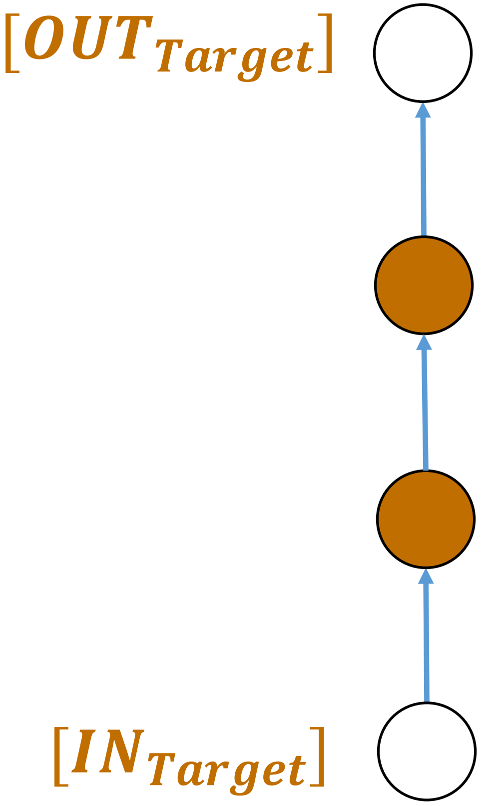


Meta-model based repurposing

Given:
(source task)



Step 1:
(target task)

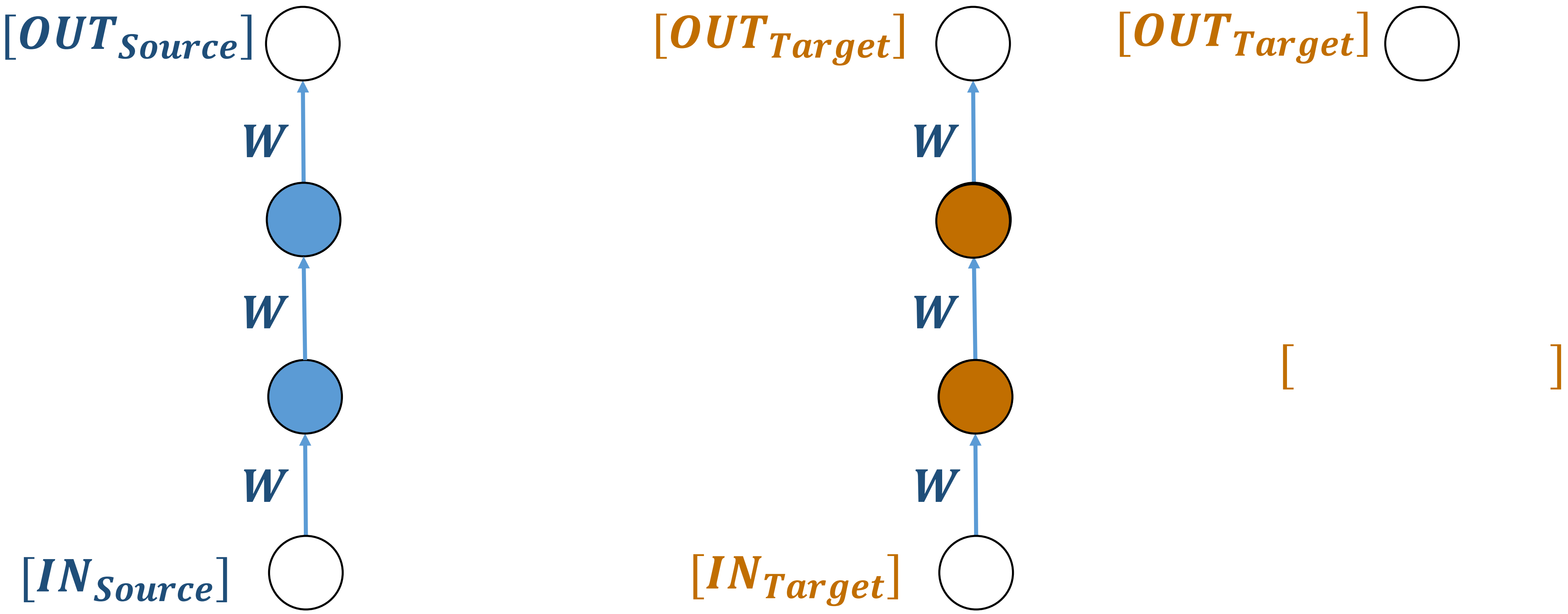


Meta-model based repurposing

Given:
(**source task**)

Step 1:
(**target task**)

Step 2:
Meta-model

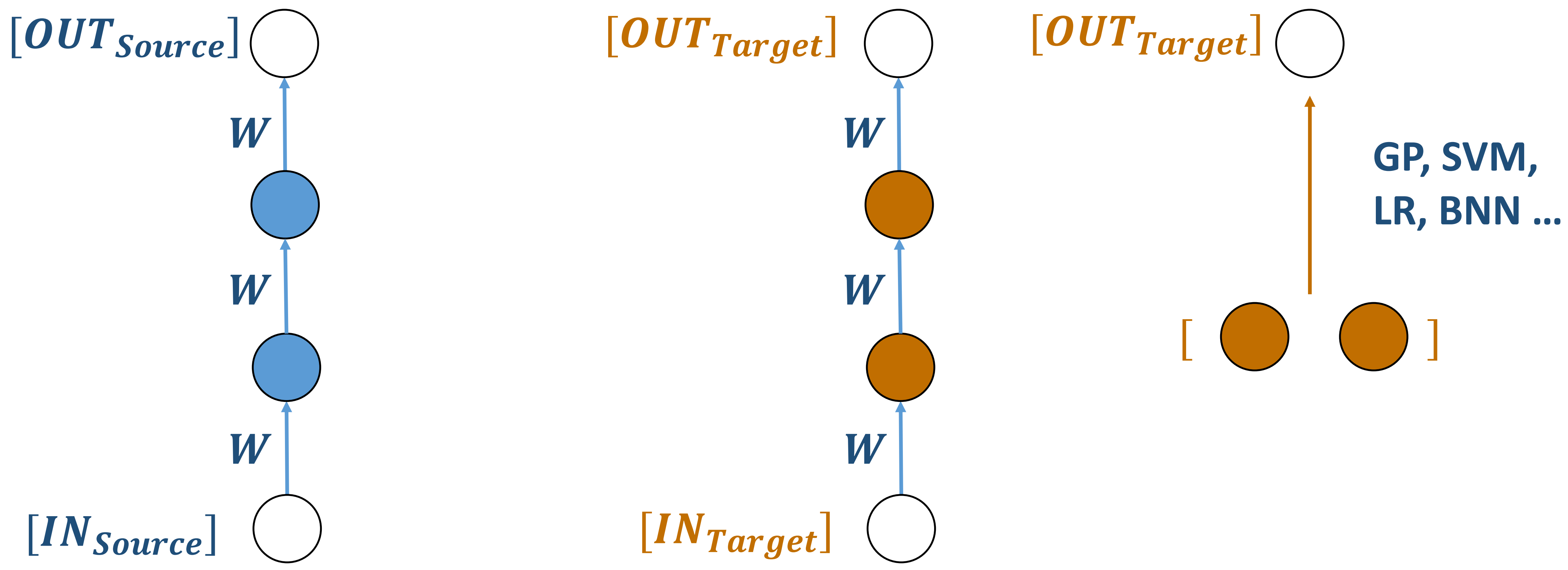


Meta-model based repurposing

Given:
(**source task**)

Step 1:
(**target task**)

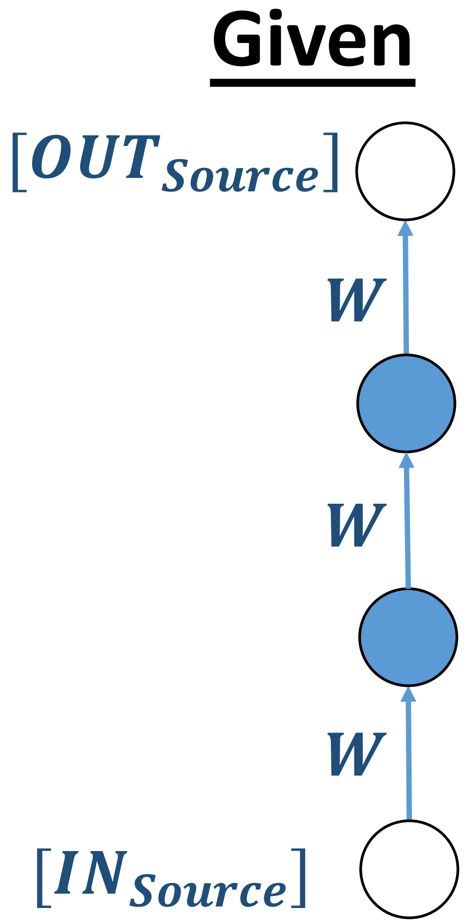
Step 2:
Meta-model



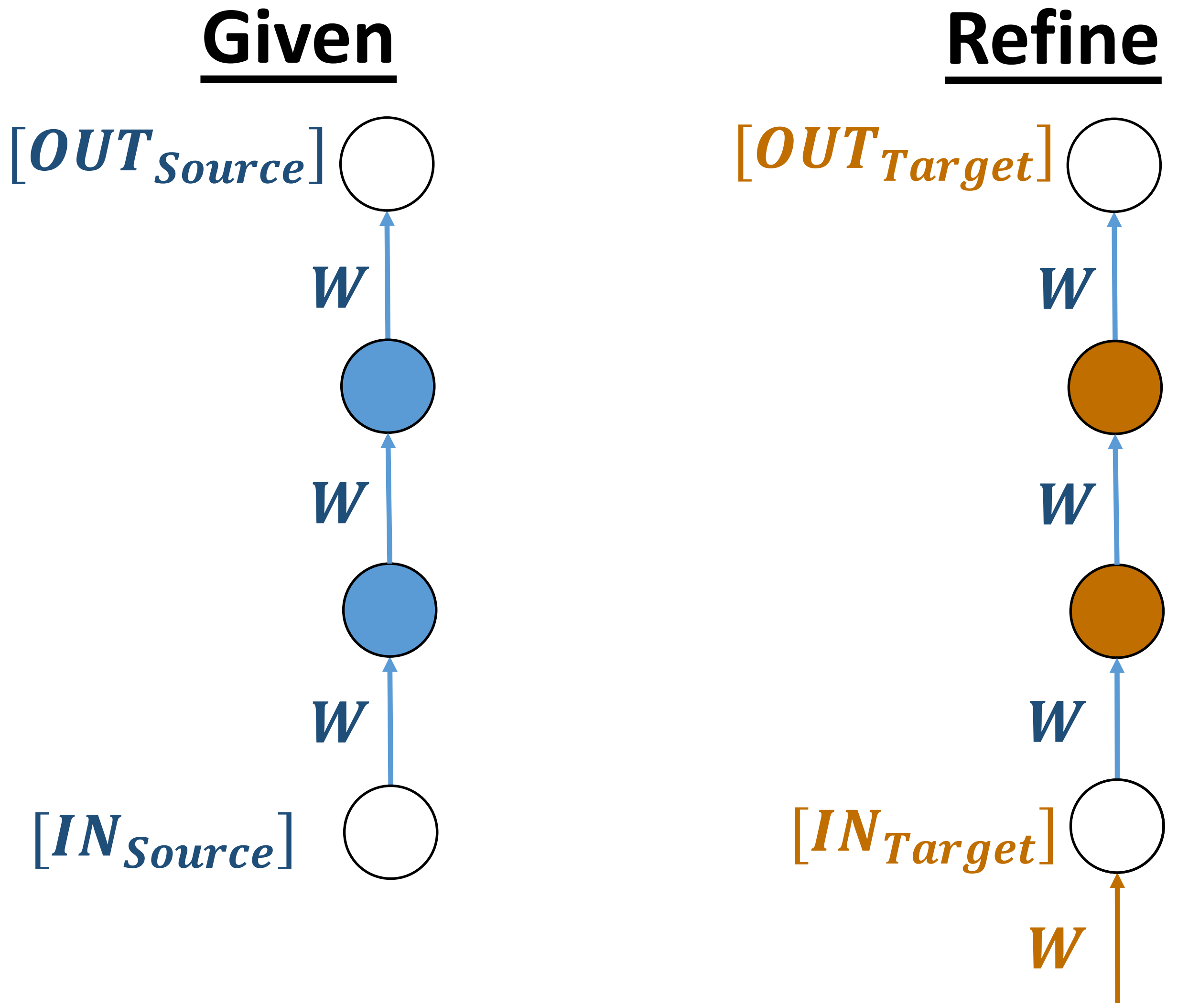
Meta-model based repurposing

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

repurposer.repurpose(train_iterator)

predictions = repurposer.predict_label(test_iterator)
```

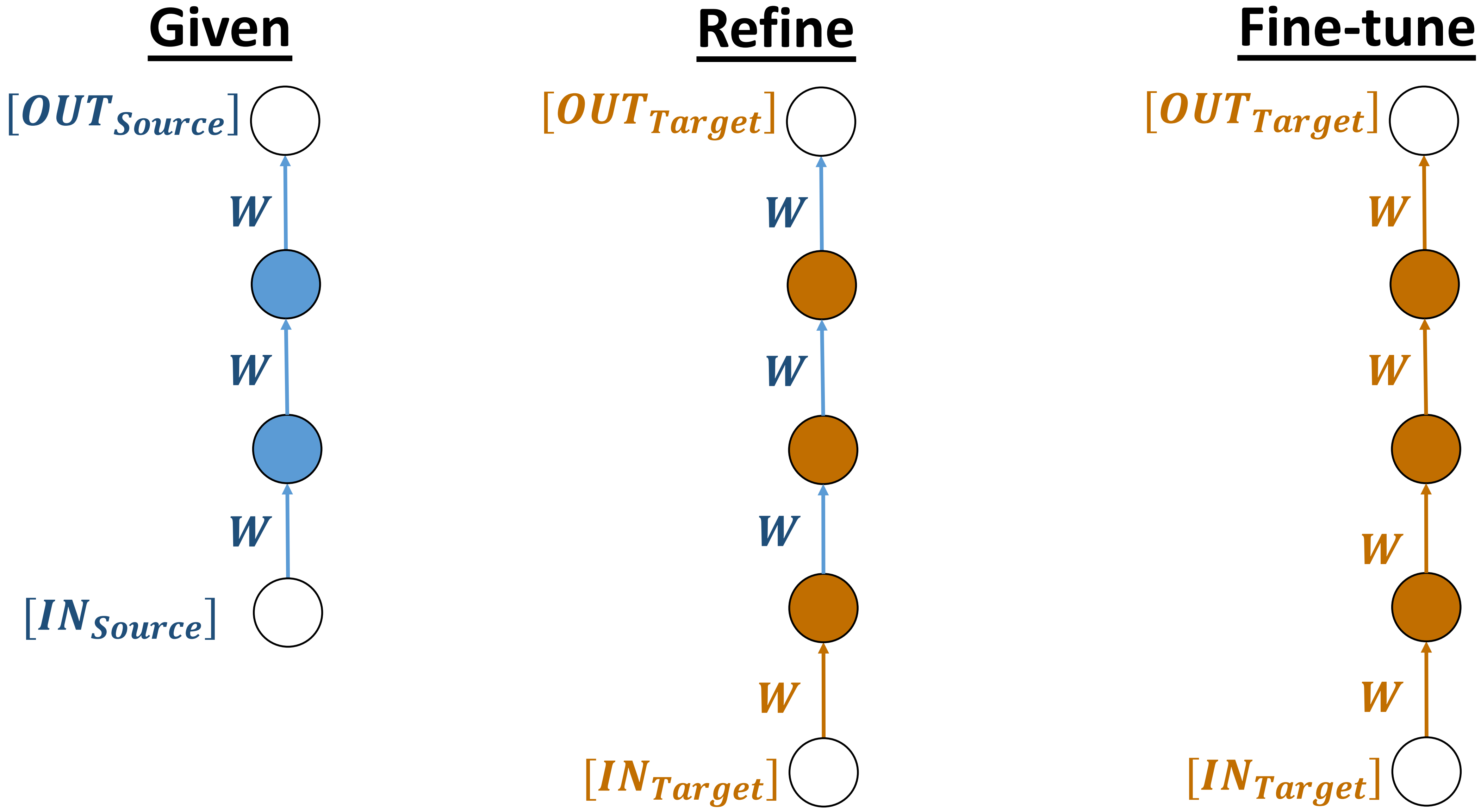
Fine-tuning based repurposing



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)
```

Closer look at ModelHandler: **inspection**

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

print(mh.layer_names)

print(mh.get_layer_type('relu5_2’))

print(mh.get_layer_names_matching_type('Convolution'))

mh.visualize_net()
```

Closer look at ModelHandler: **feature extraction**

```
features, labels = mh.get_layer_output(data_iterator= iterator, layer_names= ['fc6', 'fc8'])
```

Closer look at ModelHandler: **model manipulation**

```
mh.drop_layer_top(4)

mh.drop_layer_bottom(1)

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

fc = mx.sym.FullyConnected(name= 'fullyconntected1', num_hidden= 4)

softmax = mx.sym.SoftmaxOutput(name = 'softmax’)

mh.add_layer_bottom([conv1])

mh.add_layer_top([fc, softmax])
```

Custom repurposers

```
class KNNRepurposer(xfer.MetaModelRepurposer):
    def __init__(...):
        super(KNNRepurposer, self).__init__(...)

    def _train_model_from_features(...):
        lin_model = KNeighborsClassifier(n_neighbors=self.n_neighbors,...)
        ...

    def _predict_probability_from_features(): ...

    def _predict_label_from_features(): ...

    def get_params(self): ...

    def serialize(self, file_prefix): ...
```

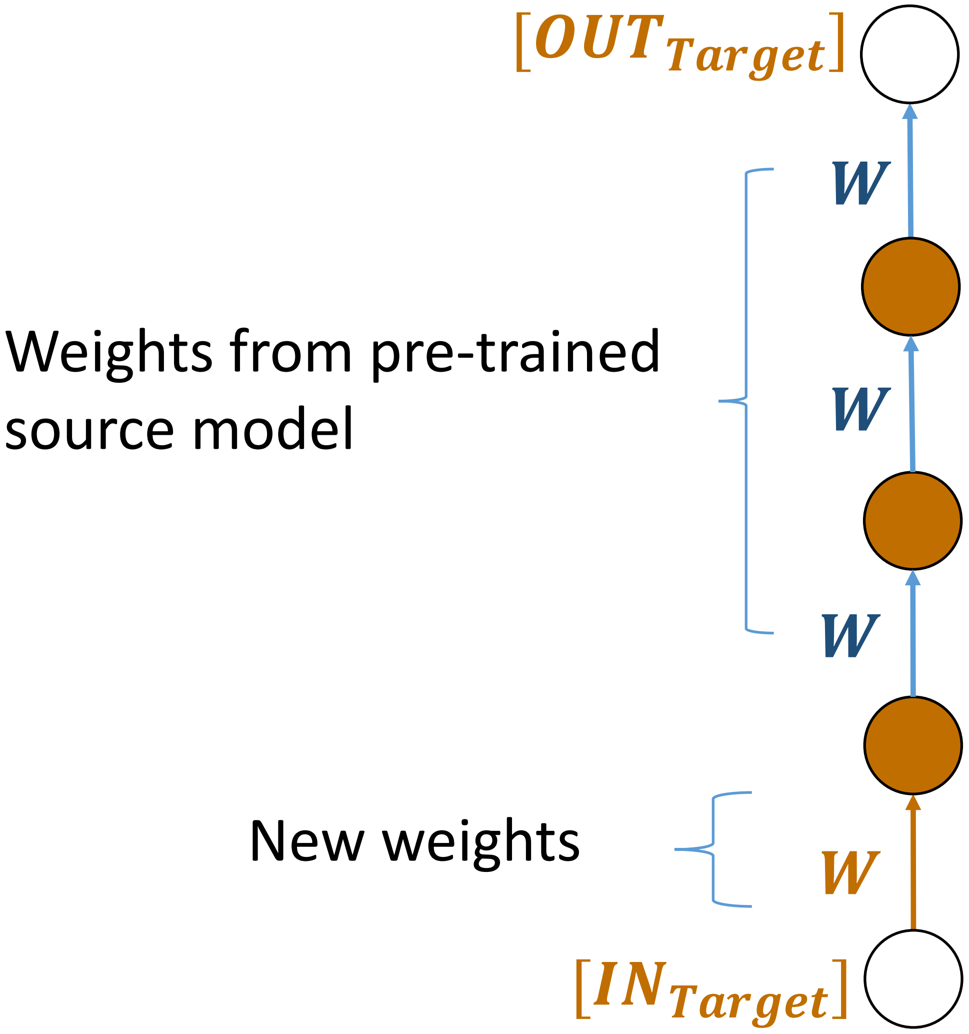
https://xfer.readthedocs.io/en/master/demos/xfer-custom-repurposers.html

Custom repurposers

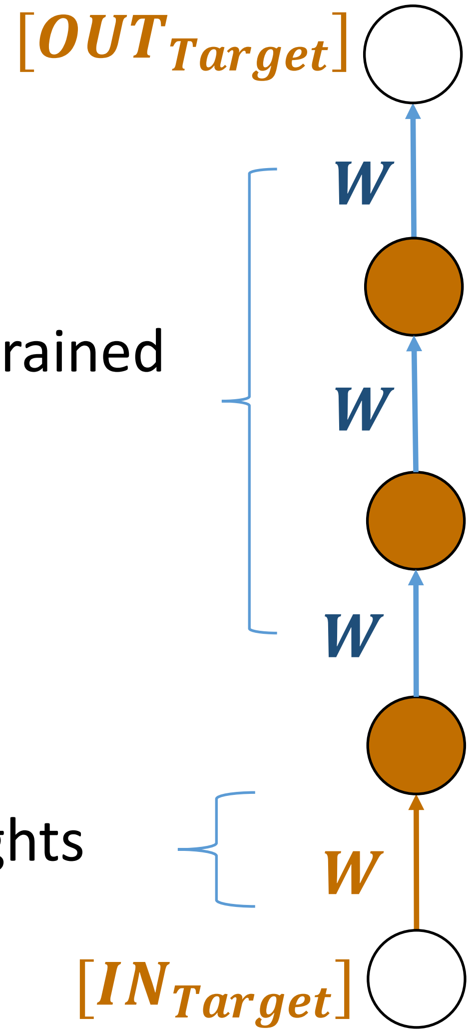
```
class Add2FullyConnectedRepurposer(xfer.NeuralNetworkRepurposer):

    ...

    def _create_target_module(self, train_iterator: mx.io.DataIter):
        model_handler = xfer.model_handler.ModelHandler(self.source_model, ...)

        # ModelHandler functionality goes here...

        return model_handler.get_module(train_iterator, fixed_layer_parameters= conv_layer_params)
```

https://xfer.readthedocs.io/en/master/demos/xfer-custom-repurposers.html

Reminder: fine-tuning based repurposing



Reminder: fine-tuning based repurposing



Weights from pre-trained source model

New weights

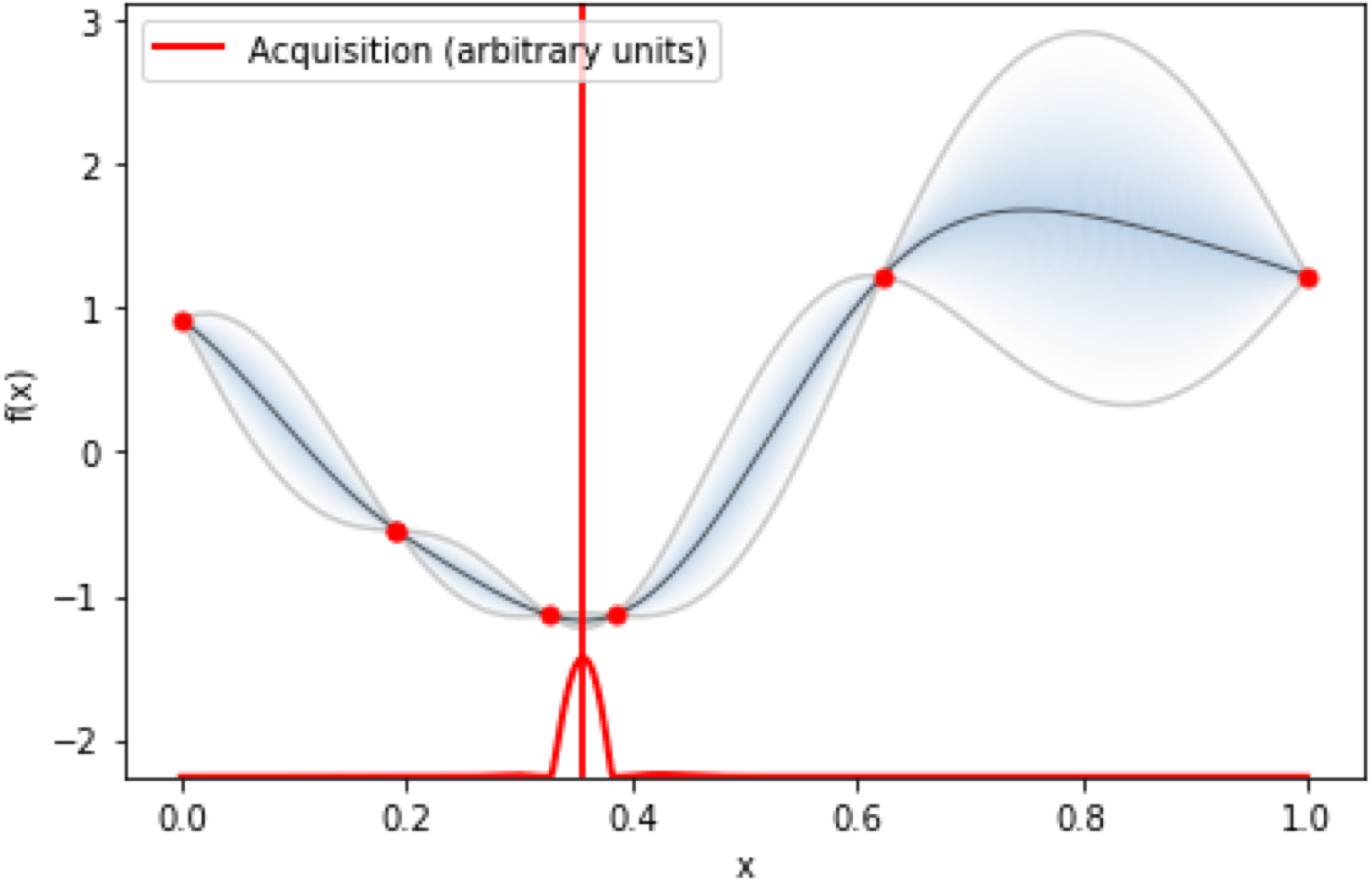
- What learning rate to use for pre-trained vs new weights?
- How many epochs?
- What optimizer to use?

HPO for hyperparameter tuning

```
optimizer_id_to_name = {1: 'sgd', 2:'adam'}

domain_with_2_hyperparams =
        [{'name': 'learning_rate', 'type': 'continuous', 'domain': (0,1)},
         {'name': 'optimizer', 'type': 'discrete', 'domain': (1,2)}]

hyperparameter_optimizer2 = GPyOpt.methods.BayesianOptimization(
        f = hpo_objective_function,
        domain = domain_with_2_hyperparams))

hyperparameter_optimizer2.run_optimization()
```

https://xfer.readthedocs.io/en/master/demos/xfer-hpo.html

```
hyperparameter_optimizer.plot_acquisition()
```



Xfer with Gluon

- Gluon models can be used with Xfer provided they use HybridBlocks so that the symbol can be extracted.

```
net = gluon.nn.HybridSequential()
…
net.hybridize()
```

Xfer with Gluon

- Gluon models can be used with Xfer provided they use HybridBlocks so that the symbol can be extracted.

```
net = gluon.nn.HybridSequential()
…
net.hybridize()
```

- The Gluon model (block) is then converted into a model (symbol)

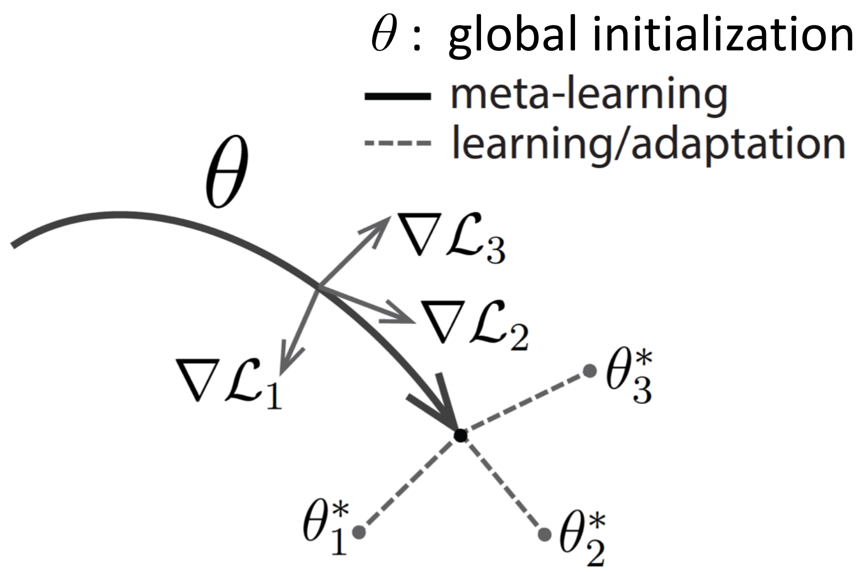
```
sym = block(data)
args, auxs = block2symbol(block.collect_params())
model = symbol2model(sym, data)
model.set_params(args, auxs)
```

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.

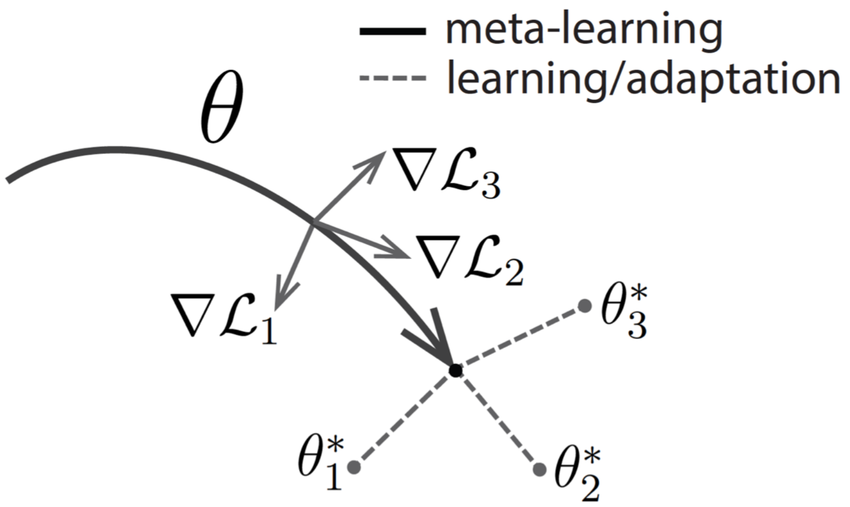


MAML approach by Chelsea Finn et al. 2017

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.

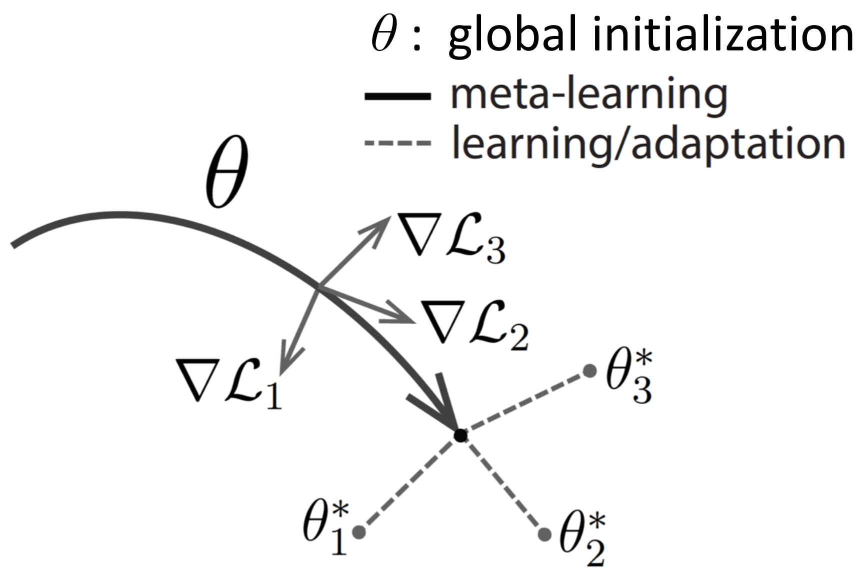
θ : global initialization



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

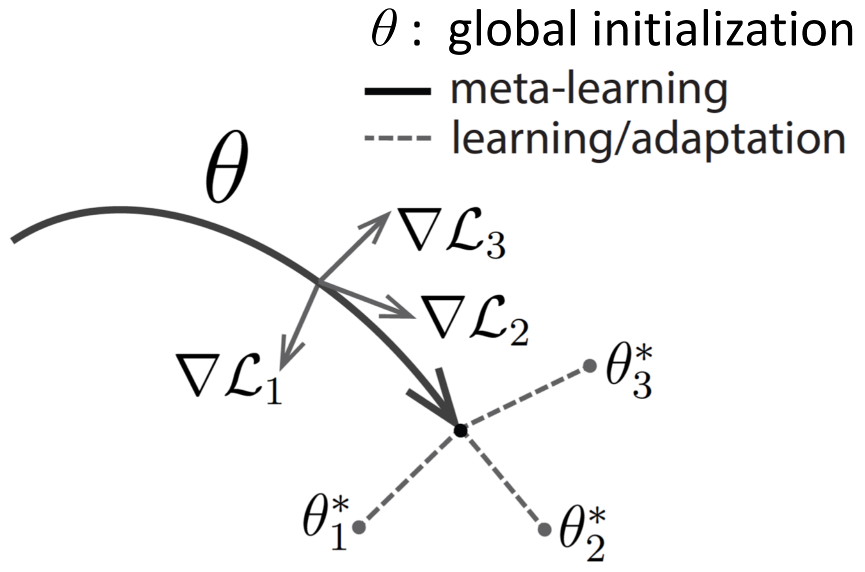
$$\min_{\theta} \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}\left(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\tau_i}(f_\theta)}\right)$$

- Start with initial θ
- for $meta\_steps = 1, 2....$:
  - Take a batch of instances per task
  - Update θ_1 , θ_2 , ... θ_τ 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.

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

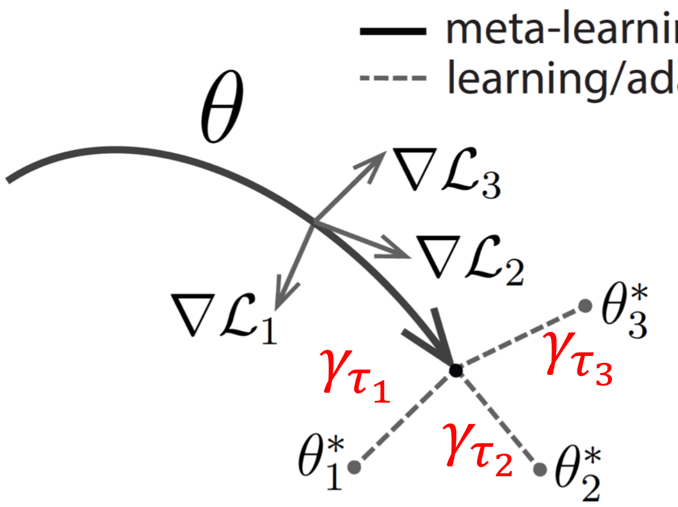


MAML approach by Chelsea Finn et al. 2017

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.

θ : global initialization



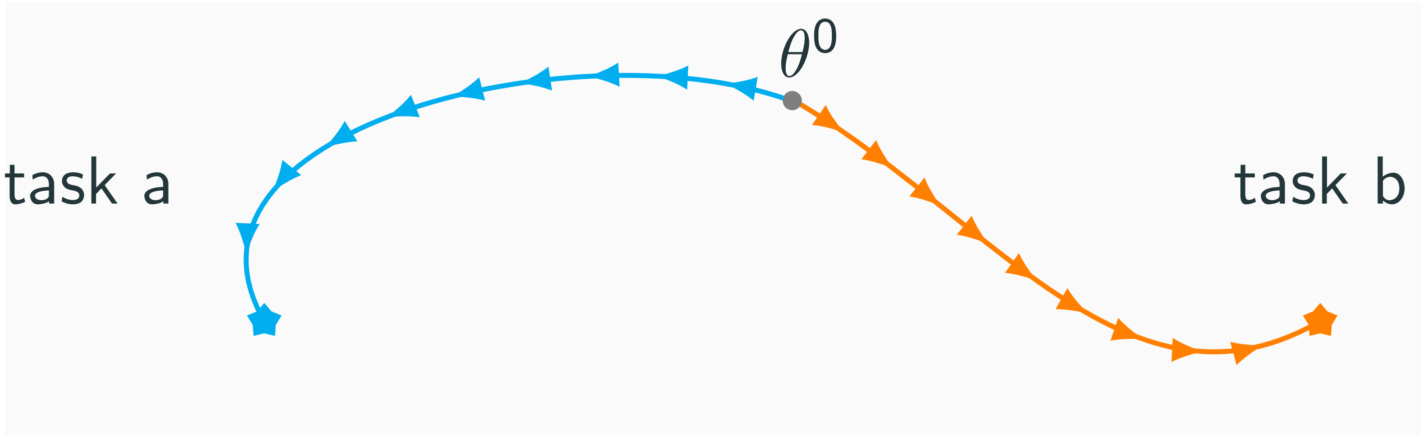
Leap approach by Flennerhag et al. 2019
*(in **Xfer** soon!)*

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

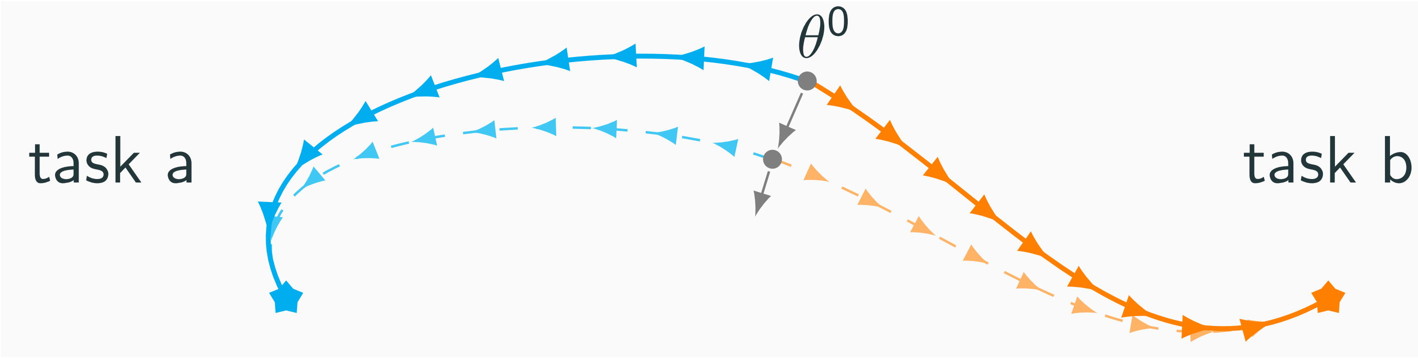
Leap balances gradient paths from all tasks...

... to minimize the expected gradient path.

Meta-step 1



Meta-step 2



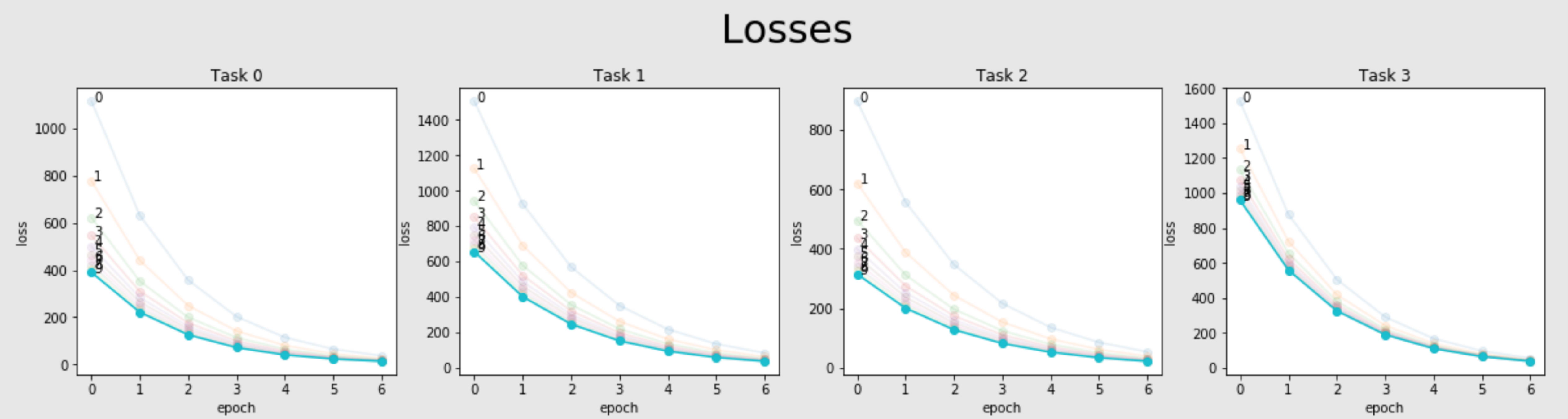
Xfer meta-learning *(available soon!)*

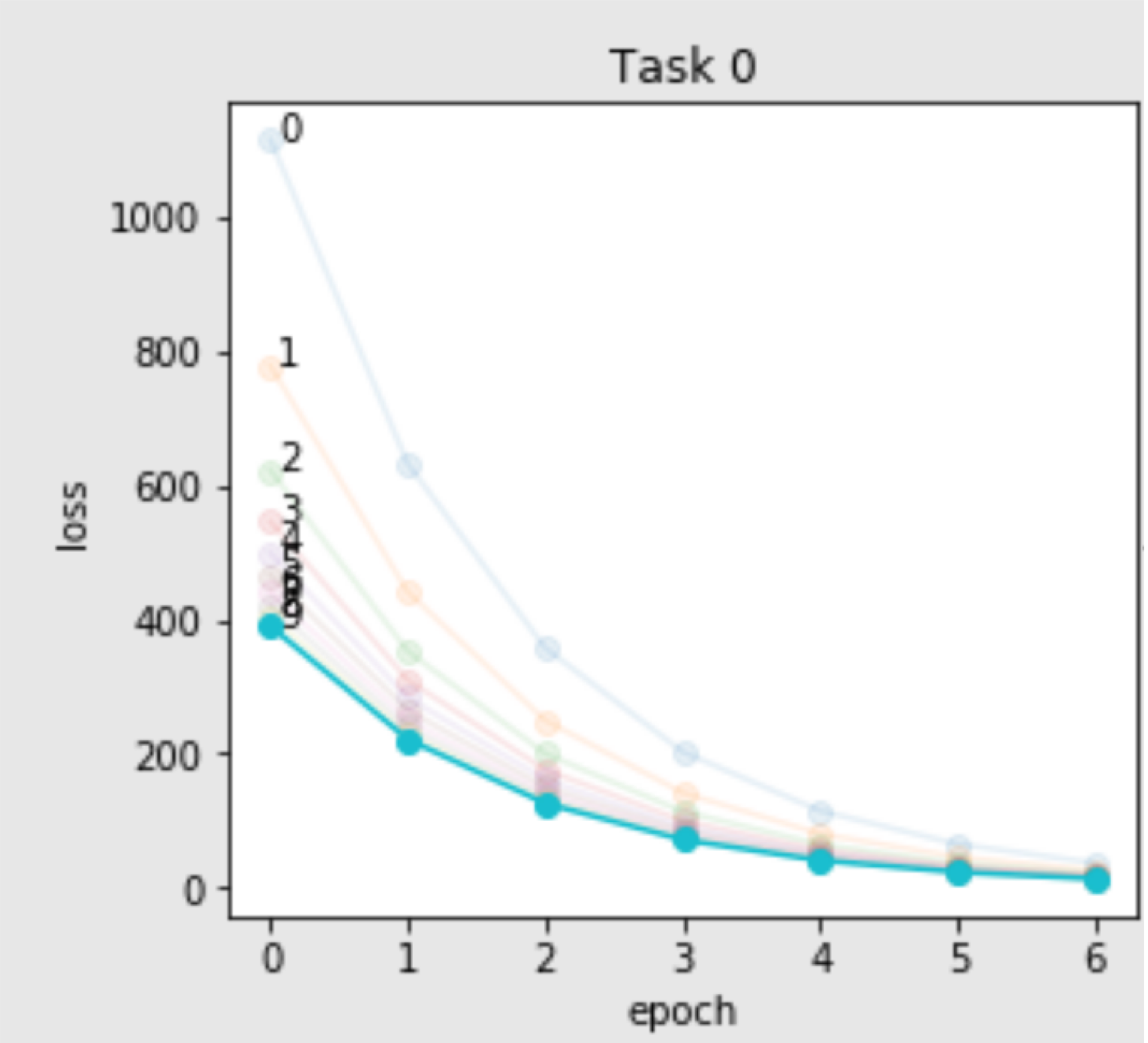
```
import xfer.contrib.xfer_leap as leap

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
0
1
2
3
4
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
$\lvert\mathbf{Y}_T\rvert \ll \lvert\mathbf{Y}_S\rvert$	Much fewer labelled data in T	Data efficient TL required
$\lvert\mathbf{Y}_T\rvert \gg \lvert\mathbf{Y}_S\rvert$	Much fewer labelled data in S	Take care of catastrophic forgetting or train T from scratch

Acknowledgements

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

Thanks!

- Notebook: adamian.github.io/talks/Damianou_DL_Xfer.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

APPENDIX

Leap

Transferability