

Transfer Learning: how to learn from little data

The biggest constraint in training a model is obtaining a sufficient amount of training data.

The deeper (greater number of layers) your model

- the more weights/parameters need to be estimated
- increases the quantity of training data

Recall our lecture on Interpreting the layers of a Neural Network

- layers close to the input seem to learn simple features
- layer l creates new features that are combinations of features of layer $(l - 1)$

Is it possible that we can "re-use" feature transformations ?

- Use the layers closest to input for a NN trained on a "source" Task
- But apply these layers (and their transformations on input) to a new "target" Task ?

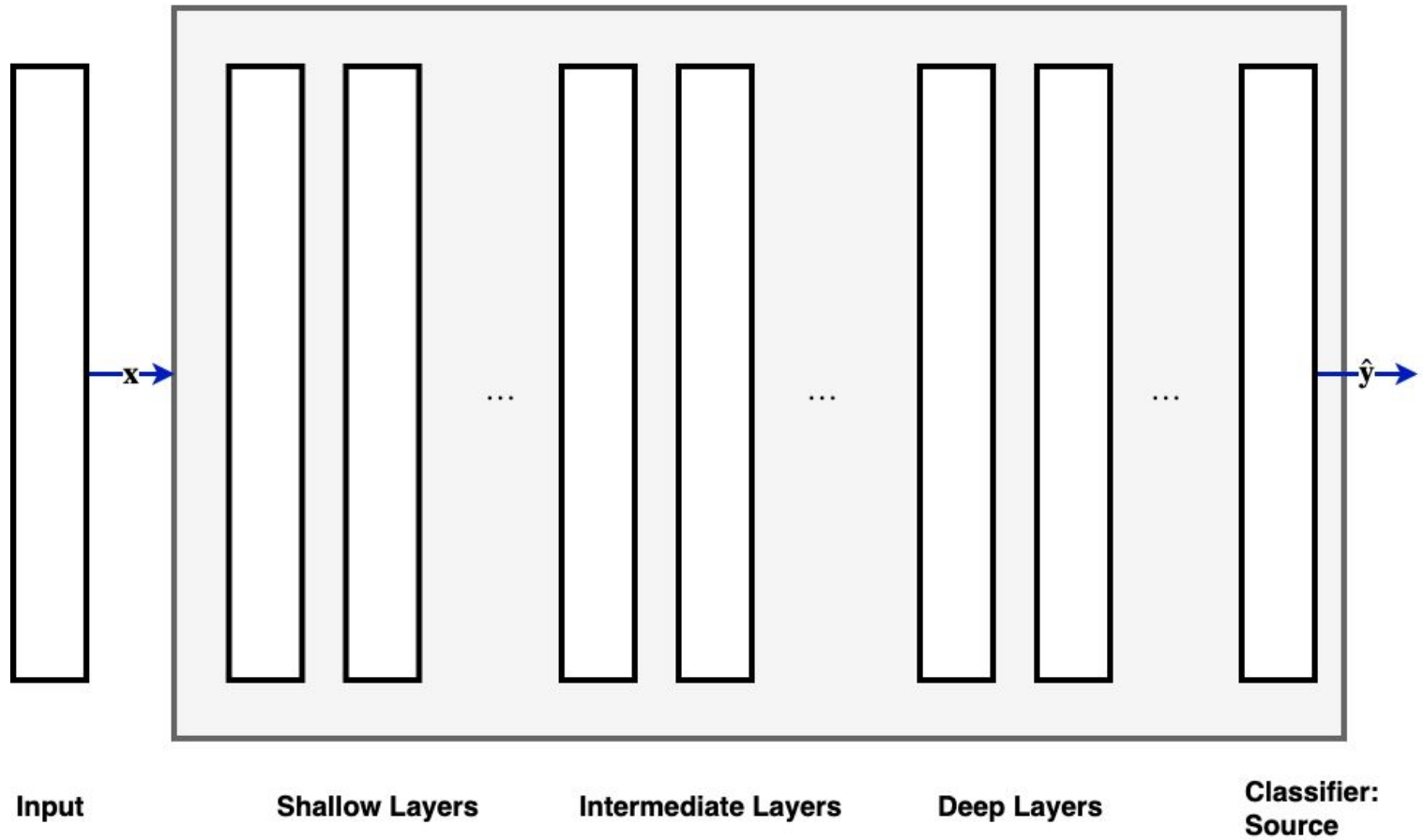
Yes !

This is called *Transfer Learning*

Create a NN for the new Task by

- using some number of layers (closest to input) of a *trained* model for some source task
- appending new *untrained* layers for the target task
 - final "head": regression, classification

Pre-trained model: Source Task



Quite often, the Source task's model

- has been trained on lots of data
- has been trained for large amounts of time
 - 2-3 weeks for image models
- has a very large number of parameters

The Transfer Learning approach imports the Source task's layer (with weights) at no cost !

The new layers added for the Target task might be able to benefit from the feature transformations created by the Source Task.

This means

- the Target task training modifies *only* the parameters of the new layers
 - *freeze* the weights of the imported Source layers
- By using a small number of parameters
 - Target task can be trained on a small amount of training data

How to choose the prefix of the Source task

Where do we truncate the Source task's model ?

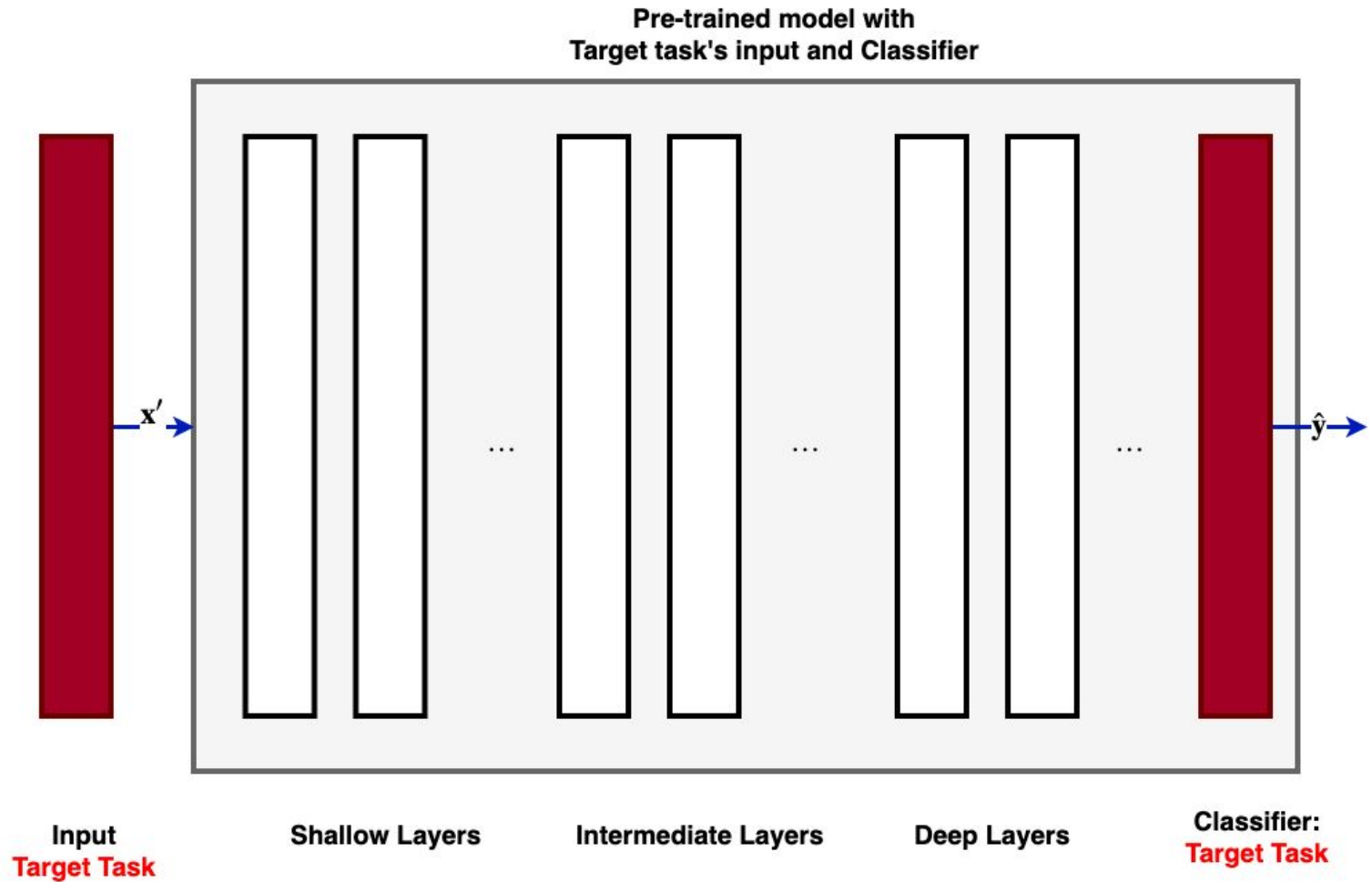
In other words: how deep should the prefix of the Source model's NN be ?

Consider the features created at the final layer of the prefix:

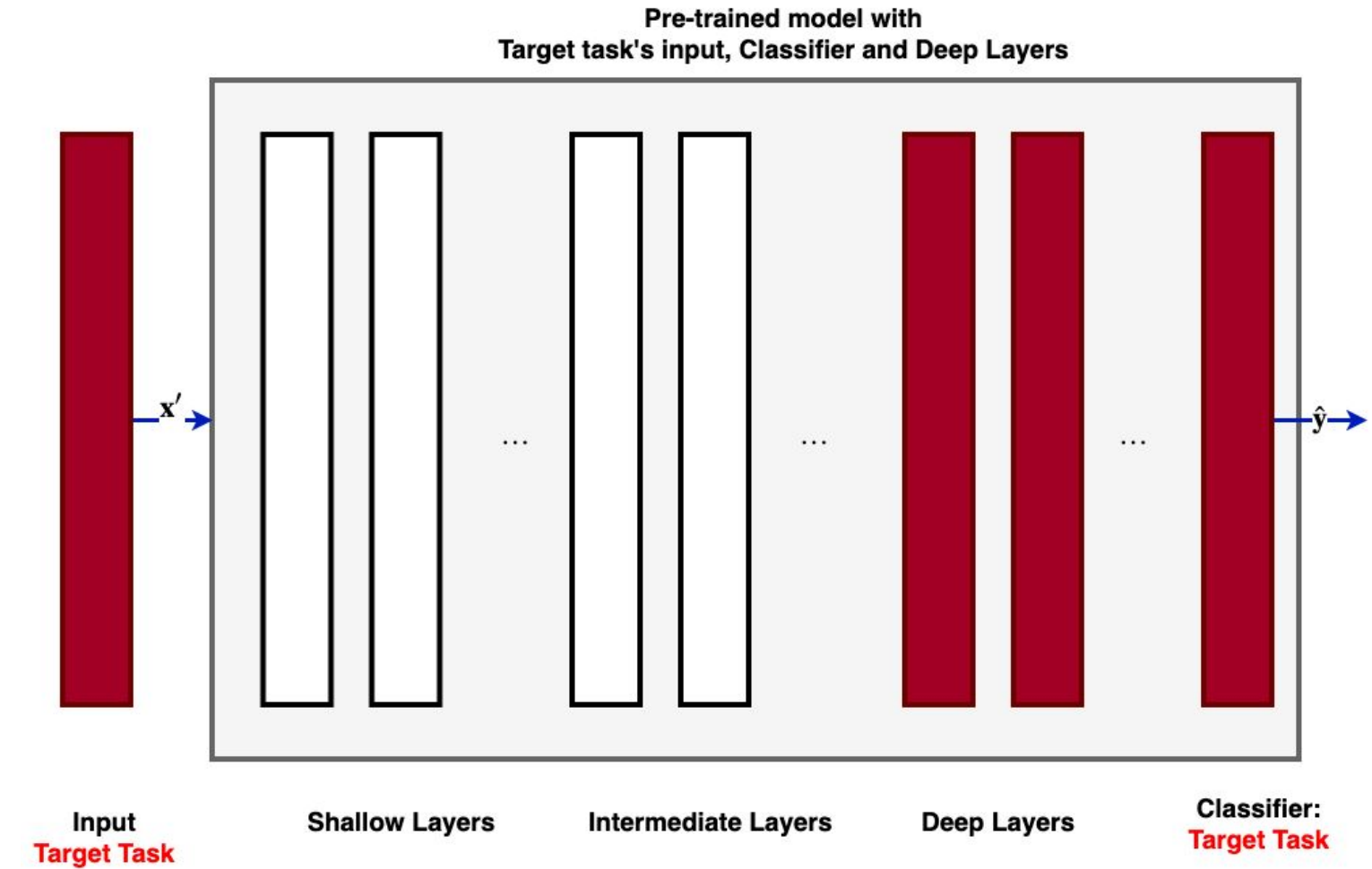
- Very shallow
 - Features learned may be too simple
 - Target may be able to benefit from deeper prefix
- Too deep
 - Features learned may be *too specialized* to the Source task

In other words: experiment !

Transfer Learning: replace the head of the pre-trained model



Transfer Learning: replace the head, deep layers of the pre-trained model



Transfer

Limitations of Transfer Learning

There is no guarantee that the features learned by the Source task will be useful for the Target task

- greater chance if tasks/domains are similar

Training the Target task

Why do we freeze the weights of the imported Source prefix ?

- the weights of the Target task's suffix are uninitialized
 - large gradients to start
- so early in training: we don't to destroy the weights in the prefix

After the suffix is trained, we sometimes

- unfreeze the latter layers of the prefix
- train with a *much lower* learning rate than the suffix

In other words: we try to "fine-tune" the prefix.

The key is fine-tuning

- wait until Suffix has been trained enough to generate small gradients
- differential learning rates per layer
 - the Prefix has been trained on lots of examples
 - don't want to alter these weights based on the small number of Target training examples

Transfer learning in Keras

```
target_model = Sequential() # Import the prefix of source_model # - Import the architecture # - and the
weights # Freeze the imported weights for layer in source_model.layer[:num_prefix_layers]:
target_model.add( layer, trainable=False ) # Add the Suffix of the target task to the model
target_model.add( ... )
```

Pre-trained Models in Keras

Image

ImageNet pre-trained models (<https://keras.io/applications/>)

NLP

Pre-trained word embeddings
(https://keras.io/examples/pretrained_word_embeddings/)

Model zoo

[Open source, pre-trained models \(https://modelzoo.co\)](https://modelzoo.co)

- Could probably have used above to split autotencoder into encoder section
- Image recognition
 - load pre-trained
 - cut off head
 - where:
 - too late: highly specialized
 - better to do earlier
 - earlier layers learn "generic features:
 - replace head
 - train on *small* number of specialized examples

In [4]: `print("Done")`

Done