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
- Which increases the need for a larger quantity of training data

Recall our module on Interpreting the layers of a Neural Network (NN)

- 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 Neural Network 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 Neural Network 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

Hopefully a picture will clarify.

Here is the multi-layer network for the Source Task

- The Classifier "head" is specialized to the Source Task (e.g., the classes of the Source task)
- For example: recognizing images from 1000 possible classes

Transfer Learning: pre-trained model

And here

- We change the inputs from $\mathbf{x}$ (Source Task) to $\mathbf{x}'$ (Target Task)
- Replacing the Classifier for the Source Task
- With a Classifier specialized to the Target Task (e.g., the classes of the Target task)
- For example: recognizing images from 2 new classes distinct from the Source task's

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

- We **import** all the weights from the Source Task layers
- We **retrain** the new Target Task Classifier Head using the training data for the Target Task

Transfer Learning

- Leverages all the effort in training the Source Task
- To benefit training the Target Task

This technique is most valuable when the model for the Source task has been trained

- With a **large** quantity of data
- For a **long** time

and when The Target task

- Has a **small** quantity of training data
- Limited computational resources

Why might this work ?

- The weights of the Source Task
- Have been laboriously trained
- To recognize "concepts"
- That transfer to other Tasks

Some examples may help

Image Classification

- Source Task: Label images to one of a 1000 possible classes (ImageNet)
- Target Task: Label images with classes **distinct** from the Source classes

Target task may have a very limited training data

Self driving cars

- Source Task: Frames from a video game driving simulator
- Target Task: Driving a real car

Real training data (actual driving) hard to obtain

- Simulation is easy

Training the Target task

When we import the Source Task's layers

- We retain (**freeze**) their weights.
- While we train/learn weights for the newly appended suffix of Target Task layers (e.g., the new head)

Why is this a good idea ?

The newly appended layers for the Target Task

- Have **uninitialized** weights
- Which means that the gradients for the Loss function will likely be large at first

If we **did not** freeze the imported weights

- The imported layers would likely "forget" the concepts that were learned

Once the suffix has been sufficiently trained so that it's weights are no longer random

- We *may* unfreeze weights of the imported layers
- That are *closest to the new head*
- Using a *much lower* learning rate

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

Fine tuning may allow the imported layers to adapt to the Target task.

The key is to remember that

- The imported layers have been trained on **many** examples
- So their weights are less likely to need to adapt
- Than the weights of the suffix, which have been trained on **few** examples

Sometimes the learning rate is varied by layer

- Imported layers have low learning rates
- Suffix layers have higher learning rates
- Degenerate case: learning rate for imported layers is zero

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

The chances may improve if the domains of the Source and Target tasks are more similar.

Pre-trained Models in Keras

Image

[ImageNet pre-trained models (https://keras.io/applications/)](https://keras.io/applications/)

NLP

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

Model zoo

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

- YOLO: object detection and classification (https://modelzoo.co/model/yolo-tensorflow)
- OpenPose: Human body, hand and facial keypoints (https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/.github/media/pose_face_hands.gif)

Transfer learning in Keras

From a coding perspective, Transfer Learning looks something like this:

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(...)

The Keras Sequential model is similar to an array of layers

- We can index into the array to obtain slices (the prefix)
- We can append to the array to add the new suffix

We will show some real code shortly.

How to choose the prefix of the Source task

It is not necessary (nor advisable) to keep **all** the layers of the Source Task's model

A smaller prefix might be better.

But where should we truncate the Source task's model ?

If the prefix is too small

- That is: we retain only the shallow layers of the Source Task's model
- We may not benefit from the fullness of the concepts learned by the Source Task
    - Similar to underfitting

On the other hand, if the prefix is too large

- That is: we retain very deep layers of the Source Task's model
- We may have concepts that are so *specialized* for the Source Task
- That they fail to benefit the Target Task

The best advice: experiment !

We will do exactly that in the example notebook for this module.

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

Conclusion

Transfer learning is a method to make you highly productive

- Leverage an existing model that may have been very expensive to train
    - Revolutionized Image Processing and Natural Language Processing
- "Cut off the head" and retrain *new head* on smaller number of examples

But there is still an element of art in knowing how much of the head to cut off

- Deeper layers may have over-specialized; best to cut them off
- Shallower layers may only recognize generic features; best to keep more of them

In [4]:

```
print("Done")
```

```
Done
```