Tutorial: Towards Transfer Learning (w/ keras)

Cameron Beebe

MCMP/GSN

June 7th, 2018
PyData Munich!

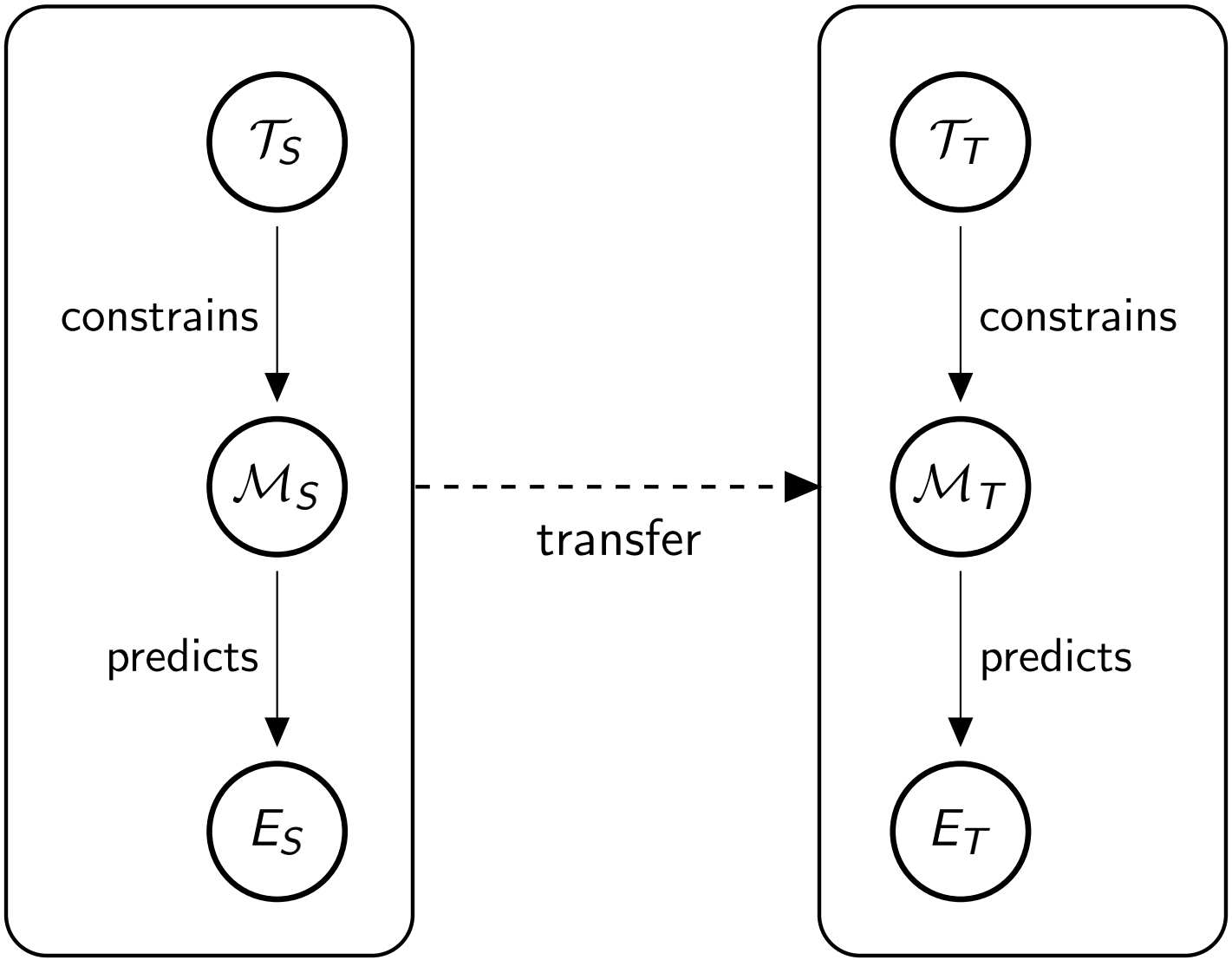
Outline

1. Theoretical Motivation
2. Practical Motivation
3. Cool Example
4. Using Keras (and what to look out for)
5. Conclusion

My Background

- Logic and Philosophy of Science
- Theoretical CS^2 and AI (at MCMP/GSN)
  - Model-based & inter-domain reasoning (e.g. Analogy)
- Connectionist Transfer Learning as worked out formal method
- Initial impressions: attempting to do some experiments

Models: Source to Target[1]



[1] Graph adapted from Roland Poellinger

Direct Transfer Pratt et al. (1991)

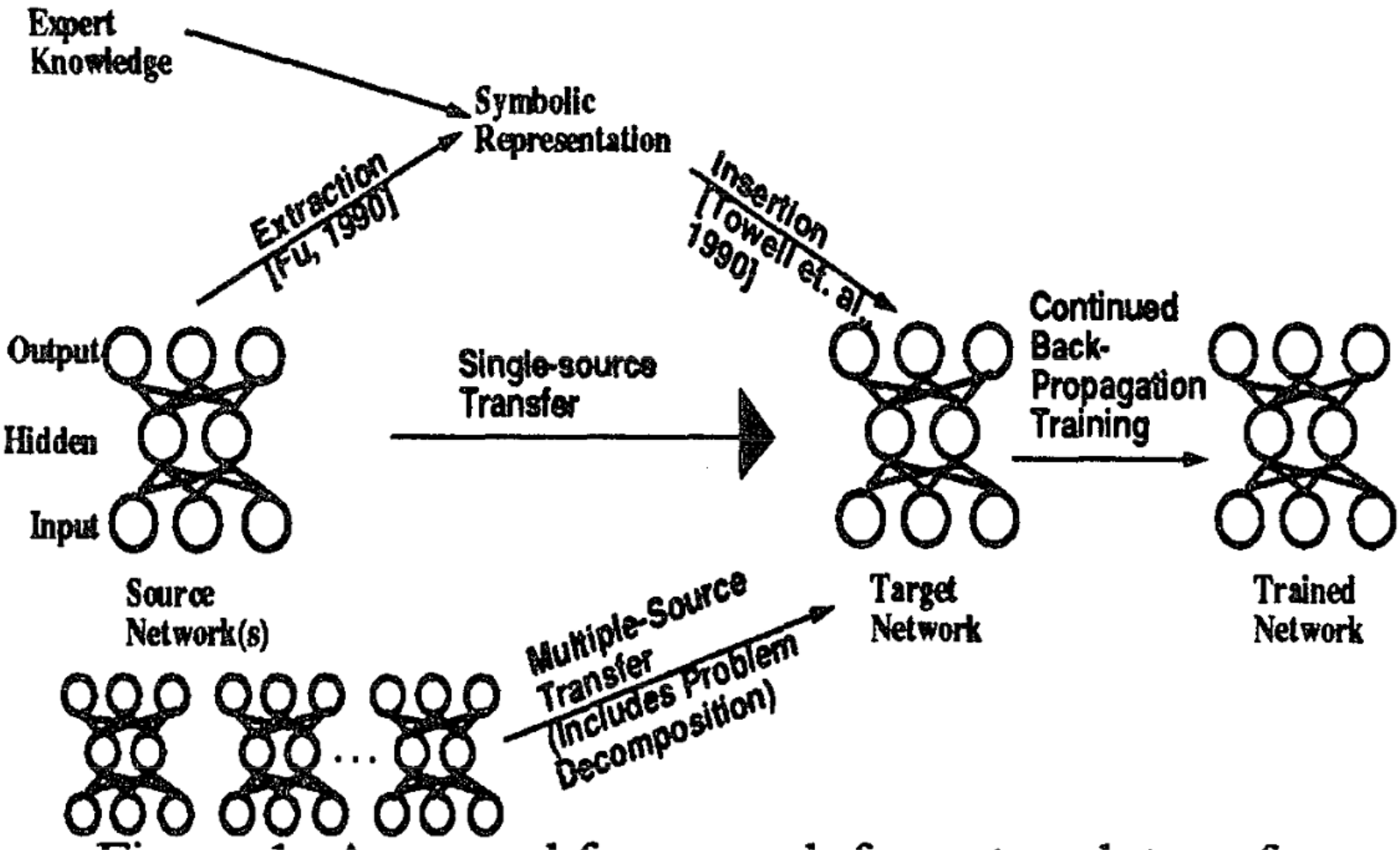


Figure 1: A general framework for network transfer

Theoretical Motivation

- Biology: Perception
- If Bio. Perceptual Sensors are reused, why not a **Perceptron**?
- Sensors are functions of previous inputs.

Theoretical Motivation

(Sharkey and Sharkey, 1993, p. 314):

> *(i), [. . .] it makes little sense from a cognitive science perspective to have neural nets that are trained from scratch on every new task, and that are unable to draw on any prior knowledge;*
> *(ii), so far as neural nets are used for psychological modelling, training nets that are in effect tabula rasa flies in the face of the mass of evidence of the important role of innate structure in the brain.;*
> *(iii), it is simply not practical to have to create and train an entirely new net as each fresh problem is approached.*

Theoretical Motivation

(Sharkey and Sharkey, 1993, p. 314):

> *(i), [. . .] it makes little sense from a cognitive science perspective to have neural nets that are trained from scratch on every new task, and that are unable to draw on any prior knowledge;*
> *(ii), so far as neural nets are used for psychological modelling, training nets that are in effect tabula rasa flies in the face of the mass of evidence of the important role of innate structure in the brain.;*
> *(iii), it is simply not* **practical** *to have to create and train an entirely new net as each fresh problem is approached.*

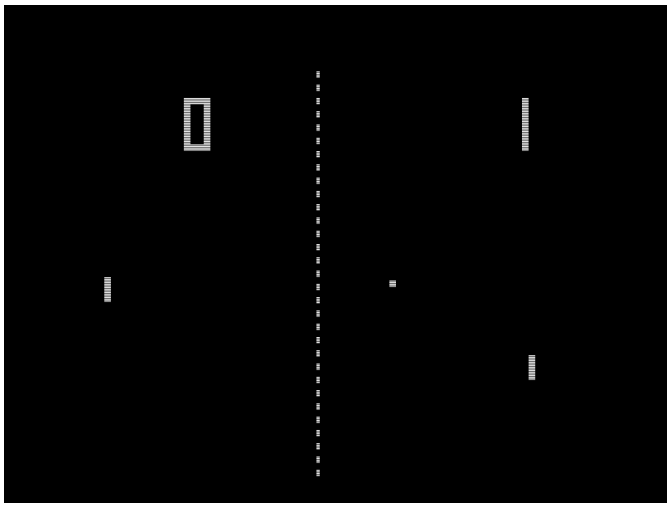
Practical Motivation

- Fitting (i.e. learning, training) takes *time*.
- Computations create *heat*.
- Training ANNs is resource intensive.
- How can we be more efficient?
- Three R's: Reduce, Reuse, Recycle

Lesson

Don't throw away your trained network!

Cool Example: Learning Pong → Learning Alien



- We have two classic Atari games: Pong, and Alien.
- Time steps in two separate NN to train Pong, Alien: t_P, t_A
- With *positive* transfer using **one** NN, consecutive training time $t_{PA} < (t_P + t_A)$

PathNet Fernando et al. (2017)

"The parameters contained in the optimal path evolved on the first task are fixed, and all other parameters are reset to their random initial values." Fernando et al. (2017)

PathNet Fernando et al. (2017)

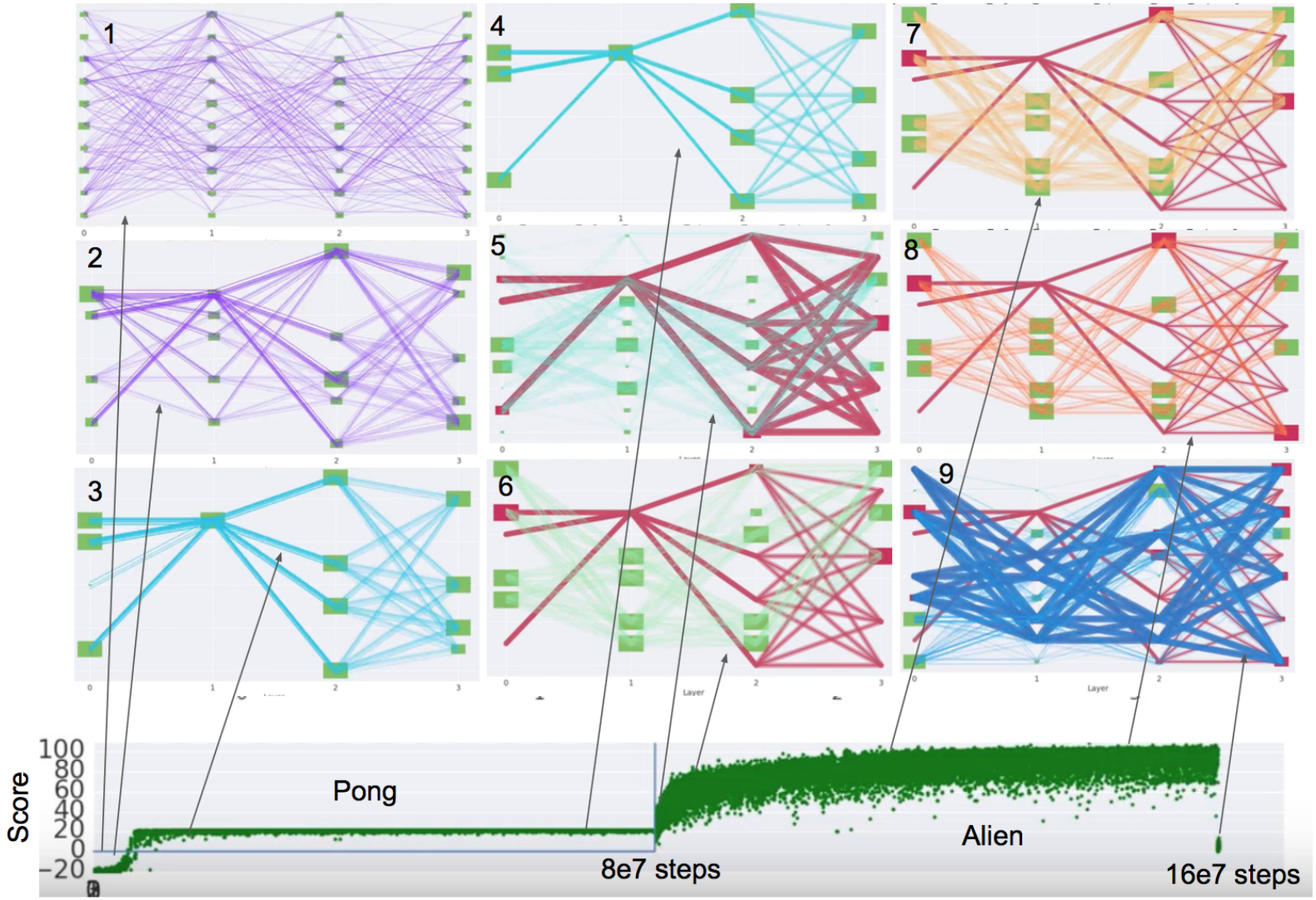


Figure 1: A population of randomly initialized pathways (purple lines in Box 1) are evolved whilst learning task A, Pong. At the end of training, the best pathway is fixed (dark red lines in Box 5) and a new population of paths are generated (light

Thinking about Transfer

- Freezing/fixing parameters (weights)
  - Many choices.
- What do do with the rest?
- Positive vs. Negative
- Target problem reasonably considered to be outside source domain (intra- vs. inter-domain)[2]

[2] Most cool examples I am aware of are handpicked source/target domains where we expect positive transfer. Procedure not automated (yet?).

Ways to measure transfer

- Time to some performance threshold
- Rate of accuracy $\uparrow$, loss $\downarrow$
- # of trainable parameters
  - Even "neutral" results w/ less trainable parameters is still more efficient

What tool to use?

- scikit-learn
- neupy
- tensorflow
- keras
- pytorch

Transfer Learning in keras

- High-level library
  - Tensorflow back end
- Pre-trained models and weights built in
  - Example: `MobileNet(include_top = False, weights='imagenet')`
- Easy to save models and weights after `model.fit()`

include_top

- =True: Keep last dense layers of imported keras models for classification. Number of classes must be equal to original.
- =False: Remove last dense layers to customize classification.
- Also relevant using incompatible input_shape, which we will do as an exercise.

`input_shape`

- Be careful whether using `input_shape=tuple` or `input_dim=integer`
- Documentation can be misleading (e.g. `MobileNet` throws error with `input_shape=(32,32,3)`)
- Be prepared to get dimension and shape errors.

Prediction

- `model.predict()` allows us to see how the model performs on test data.
- We can then do some analysis, e.g. Brier score, with the result of `model.predict_proba()`.

Notes

- Be careful with `batch_size`. (`MobileNet` has `BatchNorm` layers)
- Tensorflow 1.2 dropped Mac GPU support.
- CUDA drivers for NVIDIA GPUs: which versions, compatibility, etc... not so straight forward.

Take Home Messages

- Transfer learning recycles and enables experimentation with pre-trained models.
- Be mindful of shapes and dimensions.
- Have an Nvidia GPU.
- Don't panic.

Next Steps

- keras.io API Models allow greater flexibility and more complex models
- CUDA up and running for GPU (for either pytorch or keras/tensorflow)
- Experiment with GPU

Links

- Keras Apps: (models with pre-trained weights)
- Saving Models
- PyTorch Transfer Learning Tutorial

Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., Pritzel, A., and Wierstra, D. (2017). PathNet: Evolution Channels Gradient Descent in Super Neural Networks. *CoRR*, abs/1701.08734.

Pratt, L., Mostow, J., and Kamm, C. (1991). Direct Transfer of Learned Information Among Neural Networks. In *AAAI-91 Proceedings*, pages 584–589.

Sharkey, N. E. and Sharkey, A. J. C. (1993). Adaptive generalisation. *Artificial Intelligence Review*, 7(5):313–328.