Net2Net: Rapidly Transferring Knowledge between Large Networks

Tianqi Chen                  Ian Goodfellow        Jon Shlens
University of Washington    OpenAI                 Google

Work was done when all the authors were at Google Brain
Outline

- The Problem
- Proposed Methods
- Experimental Results
Outline

- The Problem
- Proposed Methods
- Experimental Results
Neural nets are getting larger ...

but large model = long training time
Deep Learning: Ideal vs Reality

**Ideal World**

- **Step 1: Inspiration, Find a Perfect Architecture**
  - Randomly initialized net with a perfect architecture

- **Step 2: Muscle work, Feed data Training**
  - Perfect net with perfect weights.

- Problem solved

**Reality: the Loop of Experiments**

- **Initial Design**
- **Training**
  - Try another one
- **Redesign the Model**
- More experiments
Motivation

- We usually make a **wider / deeper** net
  - As we get more data.
  - As we explore new models.

- Happens in general machine learning as well
  - Best model complexity need to match the dataset size.
  - Model selection problem.

- Ultimate goal: model evolution and continuous learning.

- Can we **reuse** the old model?
Outline

- The Problem
- Proposed Methods
- Experimental Results
Possible ways to Deal with an Old Net (🧠)
Possible ways to Deal with an Old Net

- To Eat (dump the model)
- Break into proteins..., and rebuild from scratch
Possible ways to Deal with an Old Net (🧠)

- To Eat (dump the model)
  - Break into proteins..., and rebuild from scratch

- To Learn from
  - Ask old net to “teach” the new one

![Diagram showing the process of learning from an old model](image)
Initial Attempt: Learning from Old Model

Ask new model to predict the activations of each layers of teacher model

-> **Intuition:** The new model should be as smart as old ones in each layer

-> This should let us learn lower layers quicker

-> **It did not work,** possibly due to too many random initialized components (next slide)

-> **It takes time to train a new kid, even with a great teacher...**
Possible ways to Deal with an Old Net (🧠)

- To Eat (dump the model)
  - Break into proteins..., and rebuild from scratch

- To Learn from
  - Ask old net to “teach” the new one

- **Net2Net**
  - Use old model to initialize new model.
  - In another word, transform old net to new one.
Net2Net Workflow

Traditional Workflow

Initial Design

Training

Rebuild the Model

Net2Net Workflow

Initial Design

Training

Reuse the Model

Net2Net Operator

Training
The Obstacle: (Partial) Random Initialized Components in the Net

Idealized Experiment on ImageNet (Inception-BN) Setup

- Take a trained model
- Copy the First $k$ layers over, randomly initialize the rest layers
- Training

More uninitialized components in the model, -> Less gain we get in initial bootstrap phase
Motivated Solution to the Problem

We want to be *at least as good as* the old model to start with.

Avoid Adding Randomly Initialized Components
- Transform to bigger net using **Function-preserving Transformations**
- Definition of Function-Preserving Transformation:
  -> For any inputs, the two nets produce identical outputs

```
Take a trained model → A Transformed Equivalent Bigger Net → Random Weights

Function Preserving Transformation

+ 0.01

Continue Training
```
Two Ways to Expand Model Capacity

Net2WiderNet

Problem
Find Function Preserving Transformations in both cases
Function-Preserving Transformation for Wider Nets

Original Model → Randomly Remap the Nodes to Wider Model → Connects the weights, divide by duplication factor of input node.

Ready to Apply for ConvNets (Inception)
- Each node represent a depth channel in feature map
Function-Preserving Transformations for Deeper Nets (General Idea)

Existing Layer

Find Factorization

Approximate Factorization of Original Layer

Original Model

A Deeper Model That approximates original model
Function-Preserving Transformations for Deeper Nets: Add Identity Layer

Function-Preserving Initializations for Common modules in ConvNets

- Convolution: Identity Filter
- Batch Norm: $\gamma = \text{stdvar}$, $\beta = \text{mean}$
- Batch Norm without rescale
  - $\beta = \frac{\text{mean}}{\text{stdvar}}$
  - rescale connection in later layer with $\text{stdvar}$

Original Model + Layers that Initialized as Identity Mapping → A Deeper Model Contains Identity Mapping Initialized Layers
Outline

- The Problem

- Proposed Methods

- Experimental Results
Experimental Setup

- All the experiments are conducted on Inception model on ImageNet dataset.

- Use smaller learning rate to match end of schedule of source model.

- Terminology:
  - *Source model*  the trained smaller model
  - *Target model*  the new model we want to train
Experiment Results for Net2WiderNet

**Source** An inception with 0.54 number of channels in inception towers as original inception.

**Target** Standard Inception.

**Baseline**

Copy part of nets, randomly initialize rest

![Graph showing accuracy on validation set over number of mini-batches passed.](image-url)
Experiment Results for Net2DeeperNet

**Source** Standard Inception

**Target**
Add four layers of conv to each inception tower

**Baseline**
Training from random initialization
Exploring New Design Space

**Source** Standard Inception Model

**Targets**

- Wider Inception: increase channels by 2 times
- Deeper Inception: add 8 identity conv to each inception tower

---

![Graph showing accuracy over mini-batches](image-url)
Take-aways

- It is possible to reuse the existing model help training bigger models.

- Avoid adding random components.

- Use Function-preserving transformation

- Use smaller learning rate when continue training.

- LearningModel Evolution
Thank You