Support Vector Machines
Support Vector Machines

(just kidding, sort of)
What this talk is about

---

DL will solve all of our problems!  

DL is all hype!
What this talk is about

DL will solve all of our problems!

DL is all hype!
What this talk is about

DL will solve all of our problems!

DL is all hype!
Deep Learning: Modular in Theory, Inflexible in Practice

diogo@enlitic.com
Deep Learning
Some Basics
Some Basics

- DL = composing optimizable subcomponents
Some Basics

- DL = composing optimizable subcomponents
- optimizable ~ differentiable
Some Basics

- DL = composing optimizable subcomponents
- optimizable ~ differentiable
- differentiable = can do backprop
Some Basics

- DL = composing optimizable subcomponents
- optimizable ~ differentiable
- differentiable = can do backprop
- backprop = chain rule + dynamic programming
Some Basics

● DL = composing optimizable subcomponents
● optimizable ~ differentiable
● differentiable = can do backprop
● backprop = chain rule + dynamic programming
● practical DL = DL + gradient descent + software + data
Some Basics

- DL = composing optimizable subcomponents
- optimizable ~ differentiable
- differentiable = can do backprop
- backprop = chain rule + dynamic programming
- practical DL = DL + gradient descent + software + data
- gradient descent = local learning
Some Basics

- DL = composing optimizable subcomponents
- optimizable ~ differentiable
- differentiable = can do backprop
- backprop = chain rule + dynamic programming
- practical DL = DL + gradient descent + software + data
- gradient descent = local learning
- software in {theano, tensorflow, torch, caffe, mxnet, ...}
Modular in Theory
DL solves real problems
DL solves real problems
DL solves real problems
DL solves real problems
DL solves real problems
DL creates
DL creates
DL creates
DL creates
DL creates
Deep Residual Learning for Image Recognition

Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
(Submitted on 2 Dec 2015)

Deep residual networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers—far deeper than VGG nets—but still having lower complexity. An ensemble of these residual nets achieves 3.5% error on the ImageNet test set. This result won the 1st place on the ImageNet 2015 classification task. We also present analysis on CIFAR-10 with 160 and 1600 layers.

The depth of representations is of central importance for many visual recognition tasks. Solving it with our extremely-deep representations, we obtain a 2% relative improvement on the COCO object detection dataset. Deep residual nets are foundations of our submissions to LSUN RC and COCO 2015 competitions, where we also won the 1st places on the tasks of imageNet detection, ImageNet localization, COCO detection, and COCO segmentation.

Semi-Supervised Learning with Ladder Networks

Antti Raatikainen, Harri Valpola, Mikko Honkela, Mathias Berglund, Tapio Raiko
(Submitted on 3 Jul 2013 (v1), last revised 24 Nov 2013 (v2))

We combine semi-supervised learning with unsupervised learning in deep neural networks. The proposed model is trained to simultaneously minimize the sum of supervised and unsupervised cost functions by backpropagation, avoiding the need for layer-wise pre-training. Our work builds on the Ladder network proposed by Valpola (2013), which we extend by combining the model with supervision. We show that the resulting model reaches state-of-the-art performance in semi-supervised MNIST and CIFAR-10 classification, in addition to permutation-invariant MNIST classification with all labels.

Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks

Alec Radford, Luke Metz, Smugmug Chintala
(Submitted on 21 Nov 2015 (v1), last revised 7 Jan 2016 (v3))

In recent years, unsupervised learning with convolutional networks (CNNs) has seen huge adoption in computer vision applications. Comparatively, unsupervised learning with CNNs has received less attention. In this work, we hope to help bridge the gap between the success of CNNs for supervised learning and unsupervised learning. We introduce a class of CNNs called deep convolutional generative adversarial networks (DCGANs) that have certain advantages over standard CNNs.

Training on various image datasets, we show convincing evidence that our deep convolutional adversarial pair learns a hierarchy of representations from object parts to scenes in both the generator and discriminator. Additionally, we use the learned features for novel tasks—demonstrating their applicability to general image representations.

Benchmarking Deep Reinforcement Learning for Continuous Control

Yan Duann, Xi Chen, Rein Houthooft, John Schulman, Pieter Abbeel
(Submitted on 2 Apr 2016 (v1), last revised 27 May 2016 (v2))

Recently, researchers have made significant progress combining the advances in deep learning for learning feature representations with reinforcement learning. Some notable examples include learning agents to play Atari games based on raw pixel data and to acquire advanced manipulation skills using raw sensory inputs. However, it has been difficult to quantify progress in the domain of continuous control due to the lack of a commonly adopted benchmark. In this work, we present a benchmark suite of continuous control tasks, including classic tasks like cart-pole swing-up, tasks with very high state and action dimensionality such as 3D humanoid locomotion, tasks with partial observations, and tasks with hierarchical structure. We report novel findings based on the systematic evaluation of a range of implemented reinforcement learning algorithms. Both the benchmark and reference implementations are released at this URL in order to facilitate experimental reproducibility and to encourage adoption by other researchers.
DL takes in * data

Sequence to Sequence Learning with Neural Networks
Ilya Sutskever, Oriol Vinyals, Quoc V. Le
(Submitted on 10 Sep 2014 v1, last revised 14 Dec 2014 (this version, v3))

Deep Neural Networks (DNNs) are powerful models that have achieved excellent performance on difficult learning tasks. Although DNNs work well whenever large labeled training sets are available, they cannot be used to map sequences to sequences. In this paper, we present a general end-to-end approach to sequence learning that makes minimal assumptions on the sequence structure. Our method uses a multilayered Long Short-Term Memory (LSTM) to map the input to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector. Our main result is that on an English to French translation task from the WMT14 dataset, the translations produced by the LSTM achieve a BLEU score of 38.8 on the entire test set, where the LSTMs BLEU score was penalized on out-of-vocabulary words. Additionally, the LSTM did not have difficulty on long sentences. For comparison, a phrase-based SMT system achieves a BLEU score of 33.3 on the same dataset. When we used the LSTM to rerank the 1000 hypotheses produced by the aforementioned phrase-based SMT system, its BLEU score increases to 36.5, which is close to the previous best result on this task. The LSTM also learned sensible phrase and sentence representations that are sensitive to word order and are relatively invariant to the active and the passive voice. Finally, we found that reversing the order of the words in all source sentences (but not target sentences) improved the LSTM's performance markedly, because doing so introduced many short-term dependencies between the source and the target sentence which made the optimization problem easier.

Convolutional Networks on Graphs for Learning Molecular Fingerprints
David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, Ryan P. Adams
(Submitted on 30 Sep 2015 v1, last revised 7 Nov 2015 (this version, v2))

We introduce a convolutional neural network that operates directly on graphs. These networks allow end-to-end learning of prediction pipelines whose inputs are graphs of arbitrary size and shape. The architecture we present generalizes standard molecular feature extraction methods based on circular fingerprints. We show that these data-driven features are more interpretable and have better predictive performance on a variety of tasks.

Pointer Networks
Oriol Vinyals, Meire Fortunato, Navdeep Jaitly
(Submitted on 9 Jun 2015)

We introduce a new neural architecture to learn the conditional probability of an output sequence with elements that are discrete tokens corresponding to positions in an input sequence. Such problems cannot be trivially addressed by existing approaches such as sequence-to-sequence and Neural Turing Machines, because the number of target classes in each step of the output depends on the length of the input, which is variable. Problems such as sorting variable sized sequences, and various combinatorial optimization problems belong to this class. Our model solves the problem of variable size output dictionaries using a recently proposed mechanism of neural attention. Instead of using attention to blend hidden units of an encoder to a context vector at each decoder step, it uses attention as a pointer to select a member of the input sequence as the output. We call this architecture a Pointer Net (Ptr-Net). We show Ptr-Nets can be used to learn approximate solutions to three challenging geometric problems – finding planar convex hulls, computing Delaunay triangulations, and the planar Travelling Salesman Problem – using training examples alone. Ptr-Nets not only improve over sequence-to-sequence with input attention, but also allow us to generalize to variable size output dictionaries. We show that the learnt models generalize beyond the maximum lengths they were trained on. We hope our results on these tasks will encourage a broader exploration of neural learning for discrete problems.
DL is modular

Neural Module Networks
Jacob Andreas, Marcus Rohrbach, Trevor Darrell, Dan Klein
(Submitted on 9 Nov 2015 v1), last revised 1 Jun 2016 (this version, v3)

Visual question answering is fundamentally compositional in nature—a question like “where is the dog?” shares substructure with questions like “what color is the dog?” and “where is the cat?” This paper seeks to simultaneously exploit the representational capacity of deep networks and the compositional linguistic structure of questions. We describe a procedure for constructing and learning “neural module networks”, which compose collections of jointly-trained neural “modules” into deep networks for question answering. Our approach decomposes questions into their linguistic substructures, and uses these structures to dynamically instantiate modular networks (with reusable components for recognizing dogs, classifying colors, etc.). The resulting compound networks are jointly trained. We evaluate our approach on two challenging datasets for visual question answering, achieving state-of-the-art results on both the VQA natural image dataset and a new dataset of complex questions about abstract shapes.

Learning to Compose Neural Networks for Question Answering
Jacob Andreas, Marcus Rohrbach, Trevor Darrell, Dan Klein
(Submitted on 7 Jan 2016 v1), last revised 7 Jun 2016 (this version, v4)

We describe a question answering model that applies to both images and structured knowledge bases. The model uses natural language strings to automatically assemble neural networks from a collection of composable modules. Parameters for these modules are learned jointly with network-assembly parameters via reinforcement learning, with only (world, question, answer) triples as supervision. Our approach, which we term a dynamic neural model network, achieves state-of-the-art results on benchmark datasets in both visual and structured domains.

Learning Modular Neural Network Policies for Multi–Task and Multi–Robot Transfer
Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, Sergey Levine
(Submitted on 22 Sep 2016)

Reinforcement learning (RL) can automate a wide variety of robotic skills, but learning each new skill requires considerable real-world data collection and manual representation engineering to design policy classes or features. Using deep reinforcement learning to train general purpose neural network policies alleviates some of the burden of manual representation engineering by using expressive policy classes, but exacerbates the challenge of data collection, since such methods tend to be less efficient than RL with low-dimensional, hand–designed representations. Transfer learning can mitigate this problem by enabling us to transfer information from one skill to another and even from one robot to another. We show that neural network policies can be decomposed into “task–specific” and “robot–specific” modules, where the task–specific modules are shared across robots, and the robot–specific modules are shared across all tasks on that robot. This allows for sharing task information, such as perception, between robots and sharing robot information, such as dynamics and kinematics, between tasks. We exploit this decomposition to train mix–and–match modules that can solve new robot–task combinations that were not seen during training. Using a novel neural network architecture, we demonstrate the effectiveness of our transfer method for enabling zero–shot generalization with a variety of robots and tasks in simulation for both visual and non–visual tasks.
DL is super cool

Inferring Algorithmic Patterns with Stack-Augmented Recurrent Nets
Armand Joulin, Tomas Mikolov
(Submitted on 3 Mar 2015 (v1), last revised 1 Jun 2015 (this version, v4))

Despite the recent achievements in machine learning, we are still very far from achieving real artificial intelligence. In this paper, we discuss the limitations of standard deep learning approaches and show that some of these limitations can be overcome by learning how to grow the complexity of a model in a structured way. Specifically, we study the simplest sequence prediction problems that are beyond the scope of what is learnable with standard recurrent networks, algorithmically generated sequences which can only be learned by models which have the capacity to count and to memorize sequences. We show that some basic algorithms can be learned from sequential data using a recurrent network associated with a trainable memory.

Learning to Transduce with Unbounded Memory
Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, Phil Blunsom
(Submitted on 8 Jun 2015 (v1), last revised 3 Nov 2015 (this version, v3))

Recently, strong results have been demonstrated by Deep Recurrent Neural Networks on natural language transduction problems. In this paper we explore the representational power of these models using synthetic grammars designed to exhibit phenomena similar to those found in real transduction problems such as machine translation. These experiments lead us to propose new memory-based recurrent networks that implement continuously differentiable analogues of traditional data structures such as Stacks, Queues, and DeQues. We show that these architectures exhibit superior generalisation performance to Deep RNNs and are often able to learn the underlying generating algorithms in our transduction experiments.
Hierarchical Multiscale Recurrent Neural Networks

Junyoung Chung, Sungjin Ahn, Yoshua Bengio

(Submitted on 6 Sep 2016 (v1), last revised 7 Sep 2016 (this version, v2))

Learning both hierarchical and temporal representation has been among the long-standing challenges of recurrent neural networks. Multiscale recurrent neural networks have been considered as a promising approach to resolve this issue, yet there has been a lack of empirical evidence showing that this type of models can actually capture the temporal dependencies by discovering the latent hierarchical structure of the sequence. In this paper, we propose a novel multiscale approach, called the hierarchical multiscale recurrent neural networks, which can capture the latent hierarchical structure in the sequence by encoding the temporal dependencies with different timescales using a novel update mechanism. We show some evidence that our proposed multiscale architecture can discover underlying hierarchical structure in the sequences without using explicit boundary information. We evaluate our proposed model on character-level language modelling and handwriting sequence modelling.
A bold claim

---

Today’s DL modules can solve all our problems if you ignore the practical aspects (data, software, optimization).
Inflexible in Practice
Inflexible in Practice: Data
It all starts with data efficiency

- Neural networks are extremely data inefficient
- Flexible models + lack of priors = data inefficiency
- For some domains, ALL the data may not be enough
- "We can do better with less data" - Yoshua Bengio
The data we have and use

Question: Why do people use ImageNet as a baseline?
A. We think it's representative of all computer vision problems
B. It's a competition
C. It isn't terrible
Question: Why do people use ImageNet as a baseline?
A. We think it's representative of all computer vision problems
B. It's a competition
C. It isn't terrible
Answer: B and C
The data we have and don’t use

- Things we’re bad at:
  - Leveraging unsupervised data
  - Multi-task learning
  - Transfer learning
  - Using implicit data
    - e.g. the weights and trajectories of every single model we’ve trained so far
The data we don’t have

- There are many interesting domains that we have little/no data on
  - Especially not publically available data
- There are many interesting challenges that we don’t measure
  - What gets measured gets managed
  - For example:
    - Long-term memory
    - RNNs in general
    - Visual attention
    - Hierarchical learning
Inflexible in Practice: Software
Computation Graphs

- All of DL right now amounts to graph construction
- Right now, we only use the simplest: adding one node at a time
- This means the amount of work is linear in the graph size
- There are many more interesting ways to create graphs
  ○ Work could be logarithmic in graph size
Composability

- We don’t want to reimplement the same thing over and over and over...
- Most tools are extremely un-composable

```python
import rnn_cell_mulint_layernorm_modern

rnn_cell = rnn_cell_mulint_layernorm_modern.BasicLSTMCell_MulInt_LayerNorm(size)
#OR
rnn_cell = rnn_cell_mulint_layernorm_modern.GRUCell_MulInt_LayerNorm(size)
#OR
rnn_cell = rnn_cell_mulint_layernorm_modern.HighwayRNNCell_MulInt_LayerNorm(size)
```
Good vs Bad Software

- - -

● Good
  ○ Allows you to try things you didn’t even think of
  ○ Encouraged to try more things
  ○ Easy to use previously coded tricks with new ideas

● Bad
  ○ Easier to explain in words than in software
  ○ Give up on ideas because they’re “unlikely to be worth the effort”
  ○ Everything’s one-off
Some things are hard
Some things are hard: Nodes with costs

- L2-SVM
- Learnable Activation Units

Deep Learning using Linear Support Vector Machines

Yichuan Tang

Submitted on 2 Jun 2013 (v1), last revised 21 Feb 2015 (this version, v4)

Recently, fully-connected and convolutional neural networks have been trained to achieve state-of-the-art performance on a wide variety of tasks such as speech recognition, image classification, natural language processing, and bioinformatics. For classification tasks, most of these 'deep learning' models employ the softmax activation function for prediction and minimize cross-entropy loss. In this paper, we demonstrate a small but consistent advantage of replacing the softmax layer with a linear support vector machine. Learning minimizes a margin-based loss instead of the cross-entropy loss. While there have been various combinations of neural nets and SVMs in prior art, our results using L2-SVMs show that by simply replacing softmax with linear SVMs gives significant gains on popular deep learning datasets MNIST, CIFAR-10, and the ICML 2013 Representation Learning Workshop's face expression recognition challenge.

Learning Activation Functions to Improve Deep Neural Networks

Forest Agostinelli, Matthew Hoffman, Peter Sadowski, Pierre Baldi

Submitted on 21 Dec 2014 (v1), last revised 21 Apr 2015 (this version, v3)

Artificial neural networks typically have a fixed, non-linear activation function at each neuron. We have designed a novel form of piecewise linear activation function that is learned independently for each neuron using gradient descent. With this adaptive activation function, we are able to improve upon deep neural network architectures composed of static rectified linear units, achieving state-of-the-art performance on CIFAR-10 (7.51%), CIFAR-100 (30.83%), and a benchmark from high-energy physics involving Higgs boson decay modes.
Some things are hard: Nodes with updates

- Batch Normalization

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
Sergey Ioffe, Christian Szegedy
(Submitted on 11 Feb 2015 (v1), last revised 2 Mar 2015 (this version, v3))

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as internal covariate shift, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization for each training mini-batch. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout. Applied to a state-of-the-art Image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin. Using an ensemble of batch-normalized networks, we improve upon the best published result on ImageNet classification: reaching 4.9% top-5 validation error (and 4.8% test error), exceeding the accuracy of human raters.
Some things are hard: Gradient Transformations

- Feedback Alignment
- Gradient Noise
- Batch Manhattan

Adding Gradient Noise Improves Learning for Very Deep Networks
(Submitted on 21 Nov 2015)

Deep feedforward and recurrent networks have achieved impressive results in many perception and language processing applications. This success is partially attributed to architectural innovations such as convolutional and long short-term memory networks. The main motivation for these architectural innovations is that they capture better domain knowledge, and importantly are easier to optimize than more basic architectures. Recently, more complex architectures such as Neural Turing Machines and Memory Networks have been proposed for tasks including question answering and general computation, creating a new set of optimization challenges. In this paper, we discuss a low-overhead and easy-to-implement technique of adding gradient noise which we find to be surprisingly effective when training these very deep architectures. The technique not only helps to avoid overfitting, but also can result in lower training loss. This method alone allows a fully-connected 20-layer deep network to be trained with standard gradient descent, even starting from a poor initialization. We see consistent improvements for many complex models, including a 72% relative reduction in error rate over a carefully-tuned baseline on a challenging question-answering task, and a doubling of the number of accurate binary multiplication models learned across 7,000 random restarts. We encourage further application of this technique to additional complex modern architectures.

How Important is Weight Symmetry in Backpropagation?
Qianli Liao, Joel Z. Lebo, Tomaso Poggio
(Submitted on 17 Oct 2015 (v1), last revised 2 Dec 2015 (this version, v3))

Gradient backpropagation (BP) requires symmetric feedforward and feedback connections -- the same weights must be used for forward and backward passes. This "weight transport problem" [1] is thought to be one of the main reasons that BP's biological implausibility. Using 15 different classification datasets, we systematically study to what extent BP really depends on weight symmetry in a study that turned out to be surprisingly similar in spirit to Little's demonstration [2] but orthogonal in its results. Our experiments indicate that (1) the magnitudes of feedforward weights do not matter to performance (2) the signs of feedback weights do matter -- the more concordant signs between feedforward and their corresponding feedback connections, the better (3) with feedback weights having random magnitudes and 100% concordant signs, we were able to achieve the same or even better performance than SGD. (4) some normalizations/stabilizations are indispensable for such asymmetric BP to work, namely Batch Normalization (BN) [3] and/or a "Batch Manhattan" (BM) update rule.
Some things are hard: Replacing parameters

- DropConnect
- Weight Tying
- Making a weight a function of the input

Regularization of Neural Networks using DropConnect

Li Wan
Matthew Zeiler
Sixin Zhang
Yann LeCun
Rob Fergus

Dept. of Computer Science, Courant Institute of Mathematical Science, New York University
Some things are hard: Modified update rules

- Scaled ADAM / SGD

<table>
<thead>
<tr>
<th>Optimization</th>
<th>No learning rate scaling</th>
<th>Learning rate scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGD</td>
<td>15.65%</td>
<td>11.45%</td>
</tr>
<tr>
<td>Nesterov momentum</td>
<td>12.81%</td>
<td>10.47%</td>
</tr>
<tr>
<td>ADAM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Some things are hard: Multi-stage pipelines

- Faster R-CNN
- Unsupervised pre-training

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks
Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun
(Submitted on 4 Jun 2015 (v1), last revised 6 Jan 2016 (this version, v3))

State-of-the-art object detection networks depend on region proposal algorithms to hypothesize object locations. Advances like SPPnet and Fast R-CNN have reduced the running time of these detection networks, exposing region proposal computation as a bottleneck. In this work, we introduce a Region Proposal Network (RPN) that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals. An RPN is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position. The RPN is trained end-to-end to generate high-quality region proposals, which are used by Fast R-CNN for detection. We further merge RPN and Fast R-CNN into a single network by sharing their convolutional features—using the recently popular terminology of neural networks with 'attention' mechanisms, the RPN component tells the unified network where to look. For the very deep VGG-16 model, our detection system has a frame rate of 5fps (including all steps) on a GPU, while achieving state-of-the-art object detection accuracy on PASCAL VOC 2007, 2012, and MS COCO datasets with only 300 proposals per image. In ILSVRC and COCO 2015 competitions, Faster R-CNN and RPN are the foundations of the 1st-place winning entries in several tracks. Code has been made publicly available.
Today’s Software

● My principle: The tools you use shouldn’t limit the way you think
● Bad news: Today’s tools are limiting
● We are making progress though...
Inflexible in Practice: Optimization
Local Learning

- How do we learn complicated things?
- How do we explore the parameter space?
- Differentiability be computationally expensive
  - E.g. for networks that use memory

Andrej Karpathy, Machine Learning PhD student at Stanford
Written Sep 8 · Upvoted by Tao Xu, Built ML systems at Airbnb, Quora, Facebook and
Microsoft. and Xavier Amatriain, Former researcher, now leading ML and engineering teams

I haven’t found a way to properly articulate this yet but somehow everything we do in deep
learning is memorization (interpolation, pattern recognition, etc) instead of thinking
(extrapolation, induction, etc). I haven’t seen a single compelling example of a neural
network that I would say “thinks”, in a very abstract and hard-to-define feeling of what
properties that would have and what that would look like.
Exploration

- Not just a problem for reinforcement learning
- Optimization can get networks stuck!
- Trajectories matter
- Recurring theme: Just because it can work, doesn't mean it will work
Exploration

- Not just a problem for reinforcement learning
- Optimization can get networks stuck!
- Trajectories matter
- Recurring theme: Just because it can work, doesn't mean it will work

GradNets: Dynamic Interpolation Between Neural Architectures

Diogo Almeida, Nate Sauder
(Submitted on 21 Nov 2015)

In machine learning, there is a fundamental trade-off between ease of optimization and expressive power. Neural Networks, in particular, have enormous expressive power and yet are notoriously challenging to train. The nature of that optimization challenge changes over the course of learning. Traditionally in deep learning, one makes a static trade-off between the needs of early and late optimization. In this paper, we investigate a novel framework, GradNets, for dynamically adapting architectures during training to get the benefits of both. For example, we can gradually transition from linear to non-linear networks, deterministic to stochastic computation, shallow to deep architectures, or even simple downsampling to fully differentiable attention mechanisms. Benefits include increased accuracy, easier convergence with more complex architectures, solutions to test-time execution of batch normalization, and the ability to train networks of up to 200 layers.
What about non-local learning?

- It doesn’t scale to more parameters
- And performs much worse than local learning

Reinforcement Learning Neural Turing Machines – Revised

Wojciech Zaremba, Ilya Sutskever

(Submitted on 4 May 2015 (v1), last revised 12 Jan 2016 (this version, v3))

The Neural Turing Machine (NTM) is more expressive than all previously considered models because of its external memory. It can be viewed as a broader effort to use abstract external Interfaces and to learn a parametric model that interacts with them.

The capabilities of a model can be extended by providing it with proper Interfaces that interact with the world. These external Interfaces include memory, a database, a search engine, or a piece of software such as a theorem verifier. Some of these Interfaces are provided by the developers of the model. However, many important existing Interfaces, such as databases and search engines, are discrete.

We examine feasibility of learning models to interact with discrete Interfaces. We investigate the following discrete Interfaces: a memory Tape, an input Tape, and an output Tape. We use a Reinforcement Learning algorithm to train a neural network that interacts with such Interfaces to solve simple algorithmic tasks. Our Interfaces are expressive enough to make our model Turing complete.
Inflexible in Practice: Understanding
Theory

- List of practically useful DL theory:
Generalization

- We want generalizable techniques
- To know something is generalizable, we need to understand it and the things it interacts with
  - To at least know when to apply it
- Since we can't do that, resort to trying it on hopefully several tasks and see what sticks
- Result: we never KNOW when something WILL work
We don’t know how anything REALLY works

- For example:
  - Batch normalization
  - Additive gradient noise
  - Hierarchical learning
We can’t answer the big questions

- How to know which problems are solvable
- How to know how much data is needed
- How to construct architectures and set hyperparameters
  - “Don’t be a hero” -Andrej Karpathy
  - Translation: stick to what works
How architectures are created

- Tons of trial and error
- Forking of existing architectures
  - Which are overfit for any competitive task
  - And we don’t understand how
- Hyperparameter optimization doesn’t give us understanding
- Something we worked on that might help: Genetic Architect
My Rule of Thumb

---

If someone is sure about something, then they are wrong.

-Me
My Rule of Thumb

---

*If someone is sure about something, then they are wrong, except if it is about not knowing something.*

- *Me*
My Rule of Thumb

If someone is sure about something, then they are wrong, except if it is about not knowing something or that something did work that one time.

-Me
Good News: All of this is solvable-ish
Questions?