Deep Learning

Workshop on Learning and Control
IIT Mandi

Pramod P. Khargonekar and Deepan Muthurayan

Department of Electrical Engineering and Computer Science
University of California, Irvine

July 2019

Outline

Why Learning and Data Science in Control?

Control Systems : Strong theoretical foundations

- Stability theory
- Algebraic, analytical, and topological structures
- Linear multivariable control
- Robust control
- Nonlinear control
- Adaptive Control
- Stochastic control
- Optimal control
- Distributed control

Control Systems: Diverse Application Domains

- Aerospace
- Automotive
- Transportation
- Communications
- Energy and power
- Water and agriculture
- Manufacturing
- Chemical processes
- Biomedical

Aspirational and Emerging Applications: Examples

- Smart-X
  1. Smart manufacturing - Industry 4.0
  2. Smart grid
  3. Smart cities
  4. Smart health
- Urban mobility
- Low carbon society
- Sustainable technologies
- Precision and population health and wellness
- Food-energy-water nexus
- Autonomous systems

Challenge: Design, operation, management and control of large, distributed, heterogeneous, complicated, interconnected socio-techno-economic systems

Vision: Control systems will play an important role but we will need to integrate with cyber-physical-human systems, data science, and machine learning

Why is Big Data Gaining Attention?

Source: Economist and www.sas.org

Data

- Cheap and ubiquitous sensors: cameras, microphones, GPS, touch, health and fitness, ...
- Internet-of-Things (IoT) into industrial and commercial arena: manufacturing, aerospace, chemical, electric grid, ...
- Scientific data: genomics, proteomics, brain imaging, telescopes, weather, satellites, ...
- User generated data
- Government data

Data Science Definition

" ... data science is a new interdisciplinary field that synthesizes and builds on statistics, informatics, computing, communication, management, and sociology to study data and its environments (including domains and other contextual aspects, such as organizational and social aspects) in order to *transform data to insights and decisions* by following a data-to-knowledge-to-wisdom thinking and methodology."

[Cao (2017)]

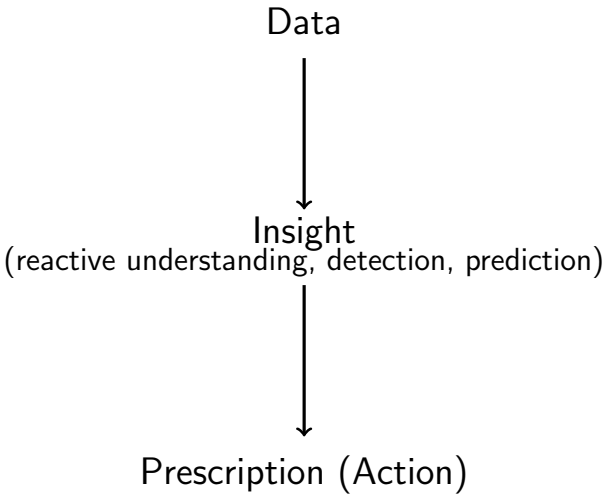
"Data Science is the science of learning from data; it studies the methods involved in the analysis and processing of data and proposes technology to improve methods in an evidence-based manner."

[Donoho (2015)]

Relevant Data Science Techniques and Resources

- Regression
- Classification — logistic regression and discriminant analysis
- Resampling methods
- Shrinkage — ridge regression, LASSO
- Dimensionality reduction — PCA, PLS, nonlinear techniques
- Streaming data
- Uncertainty quantification
- Programming and software tools
- Cloud infrastructure

Data to Decisions is a Form of Control



Computational Intelligence: Pattern Recognition or Model Building

Two fundamentally different perspectives on learning from data

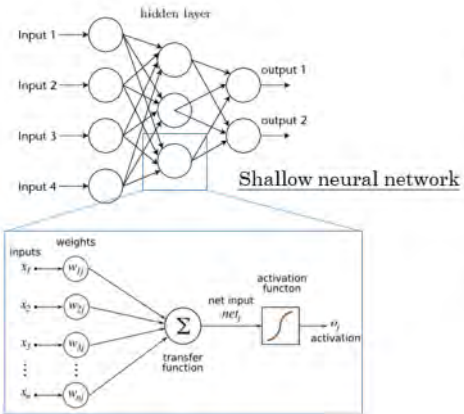
1. Statistical pattern recognition from data for prediction and control
2. Using data to build causal models to understand, predict and control

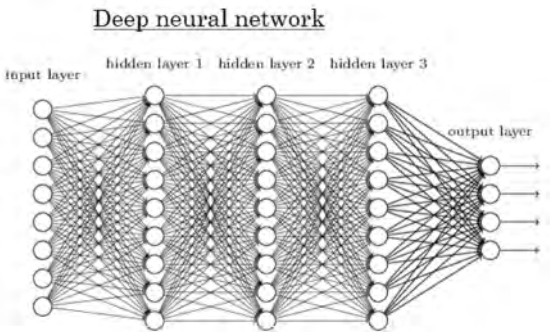
Curse of Dimensionality

- Introduced by R. E. Bellman to describe problems as dimension increases in the context of dynamic programming
- Relevant to data science, machine learning, and control problems
- High dimensional spaces have surprising and non-intuitive properties
  1. Ratio of volume of unit hypersphere to unit hypercube rapidly approaches zero
  2. Volume of multi-dimensional Gaussian inside radius 1.65 goes from 90% to zero very rapidly as dimension increases
  3. See the paper [Verleysen and Francois (2005)] for more details

Deep Learning

Deep Learning





Source: github

Deep Learning Historical Context: Perceptron by F. Rosenblatt

- 1957 report The Perceptron: A Perceiving and Recognizing Automaton
- 1958 paper by F. Rosenblatt The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain
- 1962 book Principles of Neurodynamics
- 1996 paper by M. Olazaran A Sociological Study of the Official History of the Perceptrons Controversy

Deep Learning Historical Context: Multilayer Neural Networks

- Rediscovery of backpropagation training methods
- 1970 thesis S. Linnainmaa: The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's thesis, Univ. Helsinki. [1976 paper]
- 1974 thesis by P. Werbos: Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. PhD thesis, Harvard University.
- 1985 report by D. B. Parker, Learning-Logic: Casting the Cortex of the Human Brain in Silicon. Technical Report Tr-47, Center for Computational Research in Economics and Management Science.
- 1985 paper by Y. LeCun, Une procedure d'apprentissage pour Reseau a seuil assymetrique in Cognitiva 85: a la Frontiere de l'Intelligence Artificielle, des Sciences de la Connaissance et des Neurosciences [in French] 599-604.
- 1986 paper by D. E. Rumelhart, G. E. Hinton, G. E., and R. J. Williams, Learning representations by back-propagating errors. Nature, 323, 533-536.

Building Block: A Single Artificial Neuron Unit

- Inputs: $x_1, x_2, \ldots x_n$
- Weights: $w_1, w_2, \ldots w_n$
- An activation function σ
- Examples of activation functions:

$$\sigma(z) = \frac{1}{1+e^{-z}}; \sigma(z) = \tanh(z); \sigma(z) = ReLU(z) = \max(0, z) \tag{1}$$

- Output given by

$$a = \sum_{j=1}^{n} w_j x_j \tag{2}$$

$$y = \sigma(a) \tag{3}$$

Typical Activation Functions

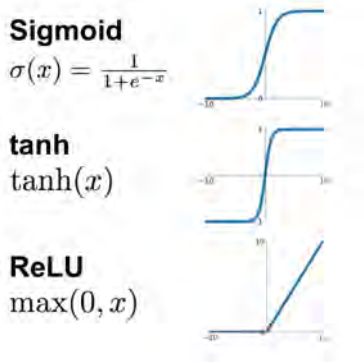


Image Source: medium

Output Node

- The output node activation function is designed for the task for the neural network
- For multiclass classification with K possible classes, a softmax function is used at the output layer. If we index classes by $1 \leq j \leq K$, then the softmax output layer is defined by

$$p(y = j|x) = \frac{e^{(w_j^T x + b_j)}}{\sum_{m=1}^{K} e^{(w_m^T x + b_m)}} \tag{4}$$

- If the output is a real variable, i.e., we have a regression or prediction problem, then the output node is an affine linear function.

Single Layer Neural Network: Universal Approximation Theorem

- Theorem: Consider a continuous function $y = f(x_1, x_2, \ldots, x_n)$ of n real variables with its domain $\mathbb{X}$ a compact subset of $\mathbb{R}^n$. Under some mild assumptions on σ, given $\epsilon > 0$, there exist $N > 0$, weights $w_{ij}, \alpha_i, 1 \leq i \leq N, 1 \leq j \leq n$ such that for all $x \in \mathbb{X}$

$$\left| f(x) - \sum_{i=1}^{N} \alpha_i \sigma(w_{ij} x_j) \right| < \epsilon \tag{5}$$

- This result goes back to papers by Cybenko(1989), Hornik et al.,(1989), Funahashi(1989), and Barron(1994).

Status in the Late 90's

"In the late 1990s, neural nets and backpropagation were largely forsaken by the machine-learning community and ignored by the computer-vision and speech-recognition communities. It was widely thought that learning useful, multistage, feature extractors with little prior knowledge was infeasible."

Deep Learning, LeCun, Bengio, and Hinton, Nature, 2015.

Deep Architectures

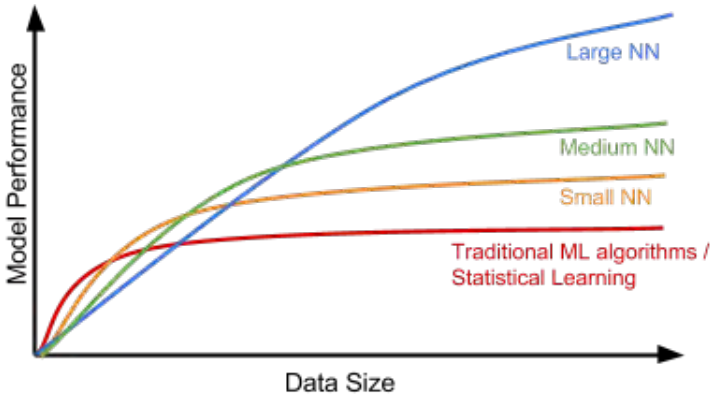
- Multiple layers of representation of input data
- Hidden layers automatically learn input features
- Compositionality built into the architecture
- Computationally efficient training, e.g., stochastic gradient descent
- Use of priors, e. g., number of hidden layers, number of units in each layer, to combat curse of dimensionality

Deep vs Shallow Classifiers

"This is why shallow classifiers require a good feature extractor that solves the selectivity-invariance dilemma — one that produces representations that are selective to the aspects of the image that are important for discrimination, but that are invariant to irrelevant aspects such as the pose of the animal. To make classifiers more powerful, one can use generic non-linear features, as with kernel methods, but generic features such as those arising with the Gaussian kernel do not allow the learner to generalize well far from the training examples. The conventional option is to hand design good feature extractors, which requires a considerable amount of engineering skill and domain expertise. But this can all be avoided if good features can be learned automatically using a general-purpose learning procedure. This is the key advantage of deep learning."

Deep Learning, LeCun, Bengio, and Hinton, Nature, 2015.

Scaling of Deep Neural Networks



Source: github

Deep Learning Breakthroughs: Key Elements

- Large datasets for training
- Multi-layer neural network models
- Leveraging computing power - graphics processors
- Computationally efficient training
- Able to combat curse of dimensionality, especially for composable functions ("functions of functions")

Major DL Innovations

- Convolutional neural networks
- Residual learning
- Long Short Term Memory (LSTM)
- Innovations in regularization: dropout
- Innovations in gradient descent: RMSprop, ADAM
- Generative adversarial framework (next lecture)

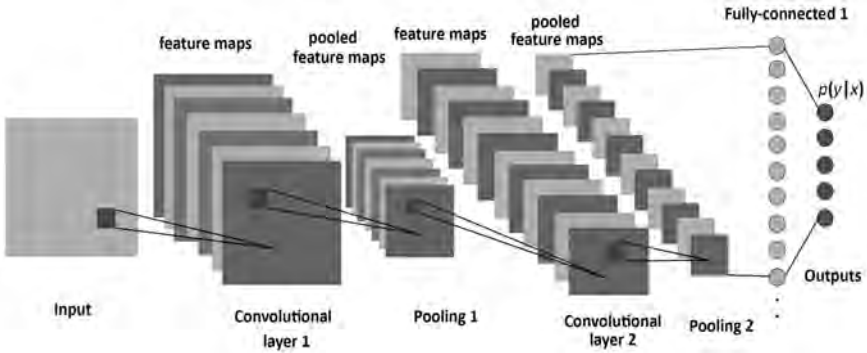
Major Applications

- Image recognition
- Object detection
- Segmentation
- Speech Recognition
- Machine Translation

Convolutional Neural Networks

- Special architectures to deal with data in the form of arrays, layers of convolutions and pooling
- LeNet, AlexNet, VGG, GoogLeNet, Residual Learning, ...
- Impressive performance gains in ImageNet competition with error rates below 5%
- Dominant method for computer vision
- ConvNet models being used in vision systems for autonomous cars

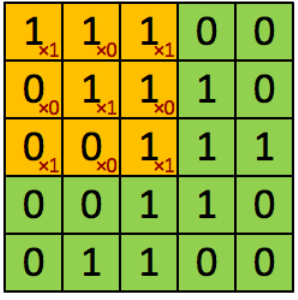
Example: Convolutional Neural Network



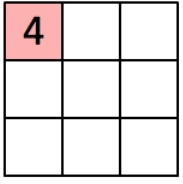
Source: MDPI

Convolutional Layer

Example:



Image

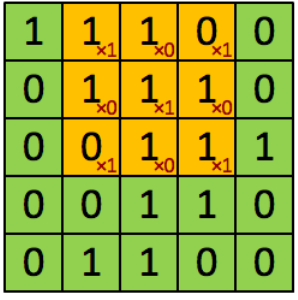


Convolved Feature

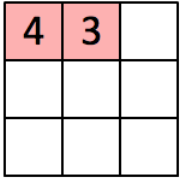
Source: comprehensive guide to neural networks

Convolutional Layer

Example:



Image

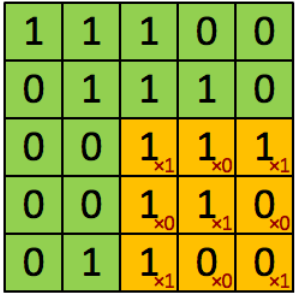


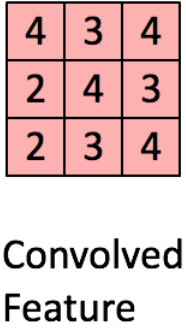
Convolved Feature

Source: comprehensive guide to neural networks

Convolutional Layer

Example:,

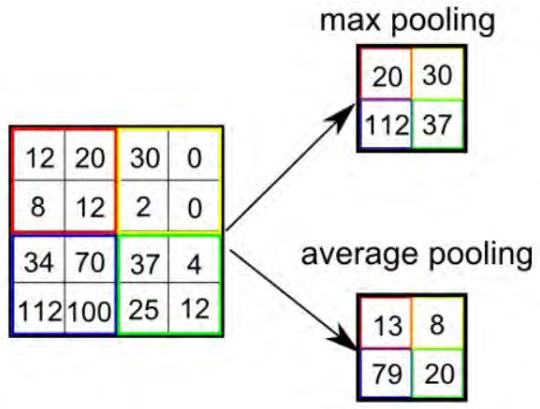




Source: comprehensive guide to neural networks

Pooling

Example:

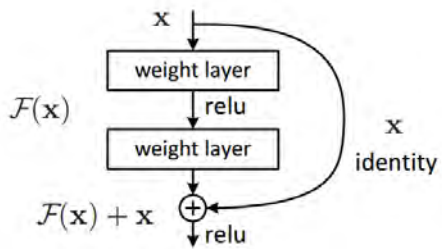


Source: comprehensive guide to neural networks

Residual Learning

Deep Residual Learning for Image Recognition, He et al. (2016)

- Addressed the "degradation problem" observed in training of deep neural networks: "with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly."
- Idea: Let $\mathcal{H}(x)$ denote the underlying function and let $\mathcal{F}(x) = \mathcal{H}(x) - x$. Introduce Residual learning building block



ResNeXt

"We present a simple, highly modularized network architecture for image classification. Our network is constructed by repeating a building block that aggregates a set of transformations with the same topology. Our simple design results in a homogeneous, multi-branch architecture that has only a few hyper-parameters to set. ... On the ImageNet-1K dataset, we empirically show that even under the restricted condition of maintaining complexity, increasing cardinality is able to improve classification accuracy."

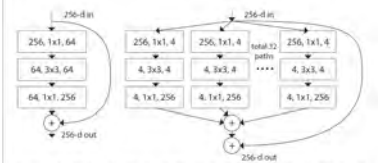


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

Aggregated Residual Transformations for Deep Neural Networks, Xie et al. (2017)

Dropout

Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al. (2014)

- Dropout: randomly dropping out units during training
- *"Units may change in a way that they fix up the mistakes of the other units. This may lead to complex co-adaptations. This in turn leads to overfitting because these co-adaptations do not generalize to unseen data."*
- Dropout breaks these complex co-adaptations
- Positive side-effect: dropout learns a sparse representation

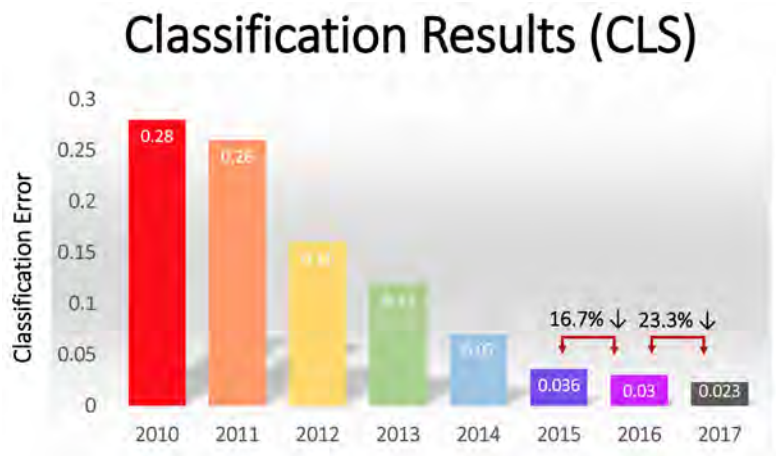
Dropout: Key Idea

"In the simplest case, each unit is retained with a fixed probability p independent of other units, where p can be chosen using a validation set or can simply be set at 0.5, which seems to be close to optimal for a wide range of networks and tasks. For the input units, however, the optimal probability of retention is usually closer to 1 than to 0.5." Srivastava et al. (2014)

"Dropout is more effective than other standard computationally inexpensive regularizers, such as weight decay, filter norm constraints and sparse activity regularization. Dropout may also be combined with other forms of regularization to yield a further improvement." - Deep Learning, 2016

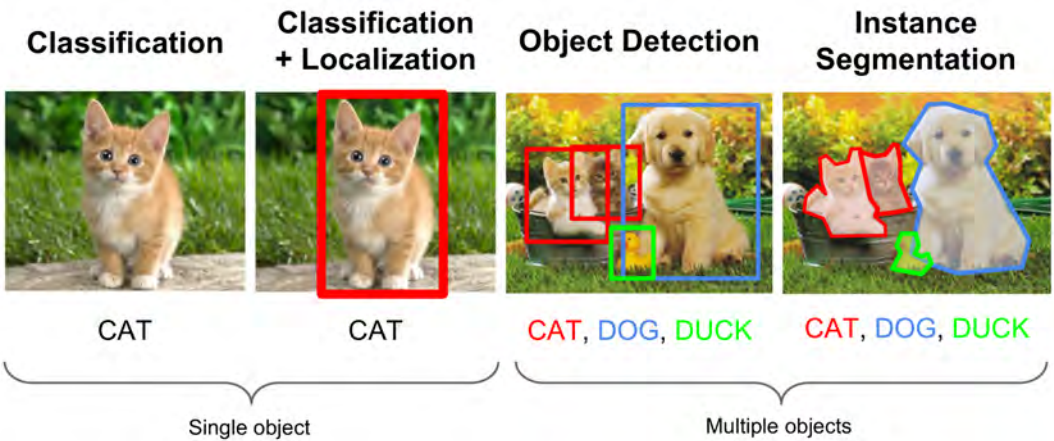
Breakthrough in Vision: ImageNet Competition

ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky, Sutskever, and Hinton, 2012



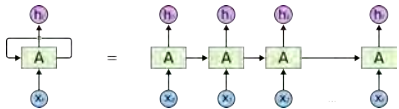
Source: image-net.org

Rapid Progress in Computer Vision



Source: techburst

Advances in Other Areas: End 2 End Sequence Learning



Source: Wiki

- Algorithm: Backpropagation Through Time (BPTT)
- Key Innovation: Long Short Term Memory (LSTM)
- Speech Recognition
- Google Voice Recognition reported a dramatic 49% increase in accuracy with CTC RNN (connectionist temporal classification RNN)
- Image to Text
- Speech Generation
- Machine Translation
- Language Models

Example: From Image to Text

Architecture: CNN + LSTM...

Figure 5. A selection of evaluation results, grouped by human rating.

[Show and Tell: A Neural Image Caption Generator, Vinyals et al. (2015)]

Example: From Image to Text

But lack of context can yield poor results

Figure 6. Perceiving scenes without intuitive physics, intuitive psychology, compositionality, and causality. Image captions are generated by a deep neural network (Karpathy & Fei-Fei 2017) using code from github.com/karpathy/neuraltalk2. Image credits: Gabriel Villena Fernández (left), TVBS Taiwan/Agence France-Presse (middle), and AP Photo/Dave Martin (right). Similar examples using images from Reuters news can be found at twitter.com/interesting_jpg.

Building machines that learn and think like people, Lake et al. (2017)

Deep Learning: Ideas and Algorithms

Deep Learning Setup

- Input vector x, also denoted by x_0 as the input to the first later
- n layers, each layer consisting of a certain number of neurons
- i-th layer accepts input vector x_{i-1}, uses weight matrix W_i and produces output vector x_i using

$$X_i = F_i(W_i, X_{i-1}) \tag{6}$$

  Functions F_i typically consist of affine linear operations followed by a nonlinear map.
- n-th layer produces X_n

$$X_n = F_n(W_n, F_{n-1}(W_{n-1}, F_{n-2}(\ldots))) \tag{7}$$

- A cost function C that computes error between the model output and actual output Y for a given training dataset. C is a function of Y, X and weights W. An example cost function is sum of squared errors.

Training of Deep Neural Networks

- Idea: adjust weights W_i of the neural network until the error between neural network output and the actual output is small
- Gradient: to do this, weights are adjusted based on the gradient of the error with respect to the weights
- In a multilayer network, calculation of these gradients is done using backpropagation
- Weights are adjusted using a stochastic gradient algorithm

Backpropagation

- We are going to apply chain rule for calculating partial derivatives of functions of functions
- With some abuse of notation, we can iteratively compute the partial derivative of the cost function wrt to intermediate variables X_i as i decreases from n to 1:

$$\frac{\partial C}{\partial X_{i-1}} = \frac{\partial C}{\partial X_i}\frac{\partial X_i}{\partial X_{i-1}} \tag{8}$$

$$= \frac{\partial C}{\partial X_i}\frac{\partial F_i(W_i, X_{i-1})}{\partial X_{i-1}} \tag{9}$$

- We can also compute the gradient of the cost function wrt to weights using

$$\frac{\partial C}{\partial W_i} = \frac{\partial C}{\partial X_i}\frac{\partial X_i}{\partial W_i} \tag{10}$$

$$= \frac{\partial C}{\partial X_i}\frac{\partial F_i(W_i, X_{i-1})}{\partial W_i} \tag{11}$$

Examples of Backpropagation Calculations

- Suppose F is a linear function, i. e., $X_i = W_i X_{i-1}$. Then

$$\frac{\partial C}{\partial X_{i-1}} = W_i^T \frac{\partial C}{\partial X_i} \tag{12}$$

$$\frac{\partial C}{\partial W_i} = \frac{\partial C}{\partial X_i} \left\{ \frac{\partial C}{\partial X_{i-1}} \right\}^T \tag{13}$$

- Suppose F is the max function, i. e., $x_0 = max(x_1, x_2)$. Then if $x_1 > x_2$,

$$\frac{\partial C}{\partial x_1} = \frac{\partial C}{\partial x_0} \tag{14}$$

- Suppose $z = ReLU(x)$.

$$\text{if } x < 0 \text{ then } \frac{\partial C}{\partial x} = 0 \text{ else } \frac{\partial C}{\partial x} = \frac{\partial C}{\partial z} \tag{15}$$

Generality of Backpropagation

- The idea is applicable to any directed acyclic graph (DAG). Even for networks with loops such as recurrent neural networks, one can apply the idea by "unfolding in time".
- The functions F can be quite general but need to be differentiable
- Deep learning frameworks include automatic differentiation and so no manual calculation of backpropagation formulae is needed

Learning: Optimization of Weights in Deep Neural Network Models

- Given training data, one can use optimization methods to find weights W_i to minimize the cost function.
- Two fundamental classes of methods
  1. Batch optimization
  2. Stochastic gradient descent
- Batch methods become challenging when data sets are large and cost and gradient computations require large amounts of memory and compute power
- Stochastic gradient methods have been found to be very efficient and effective in training of neural network weights
- A very nice review paper Optimization Methods for Large-Scale Machine Learning, Bottou et al. (2018).

Optimization Methods

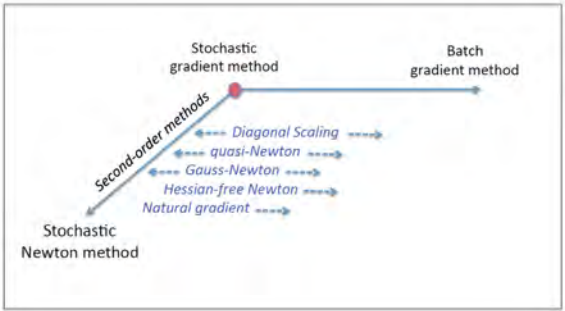


Fig. 6.1 *View of the schematic from Figure 3.3 with a focus on second-order methods. The dotted arrows indicate the effective regime of each method: the first three methods can employ minibatches of any size, whereas the last two methods are efficient only for moderate-to-large minibatch sizes.*

Optimization Methods for Large-Scale Machine Learning, Bottou et al. (2018)

Stochastic Gradient Descent: Basic Ideas

- Suppose we have a cost function $J(\theta)$ where J is a random variable that depends on parameters θ. We want to find θ to minimize the expected value of $J(\theta)$. For example, J could represent performance of the neural network on some training data and the randomness in J reflects inherent randomness in the training data.
- The basic gradient descent method updates parameters θ using the following formula:

$$\theta \leftarrow \theta - \alpha \nabla_\theta E(J(\theta)) \tag{16}$$

- Here α represents learning rate.
- In stochastic gradient descent, the idea is to get rid of the expectation and compute the gradient of J on a single element (or a small subset) of the training data. Thus

$$\theta_{t+1} \leftarrow \theta_t - \alpha \nabla_\theta J(\theta, x^t, y^t) \tag{17}$$

- In practice, the gradient of J is computed on a small batch of data (mini-batch) rather than a single element of the data.

Stochastic Gradient Descent: Learning Rate

- A key design parameter is the learning rate and learning schedule and how it is adjusted as training proceeds.
- Initially, the learning rate is kept fixed in one or more runs (epochs) through the training data set.
- After initial learning epochs, the learning rate is reduced using some form of annealing procedure, e.g., $a/(b+t)$.
- Another important practical issue to avoid bias in the presentation of training data is to shuffle the training samples.
- One can also add a regularization term to the cost function J with some norm on the weights.
- Connections to Robbins-Monro condition will be discussed in the RL lectures

Stochastic Gradient Descent: Momentum Methods

- The objective function may look like a flat ravine with steep sides.
- In such situation, SGD method becomes slow and oscillatory as the gradient descent takes it along the sides
- Momentum methods offer a very powerful approach. The idea is accumulate a velocity vector in the directions of persistent reduction in the objective function.

$$v_{t+1} = \beta_t v_t - \alpha_t \nabla_\theta J(\theta_t, x^t, y^t) \quad (18)$$

$$\theta_{t+1} = \theta_t + \gamma_{t+1} v_{t+1} \quad (19)$$

- If the cost function is a tilted plane, the "ball" reaches a terminal velocity.
- At the beginning of the learning process, the gradients may be large and so a small momentum (0.5) is used and is gradually increased to 0.9 or even higher when the weights are stuck in a ravine.
- This adjustment of momentum allows to learn without causing oscillations.

Stochastic Gradient Descent: Nesterov Accelerated Gradient Method

- The standard momentum method first computes the gradient and then adjusts the weights.
- In Nesterov Accelerated Gradient (NAG) method, the weights are adjusted first and then gradient is computed.
- Nesterov Accelerated Gradient (NAG) method has been reported to give very good performance:

$$\tilde{\theta}_t = \theta_t + \beta_t(\theta_t - \theta_{t-1}) \quad (20)$$

$$\theta_{t+1} = \tilde{\theta}_t - \alpha_t \nabla_\theta J(\tilde{\theta}_t, x^t, y^t) \quad (21)$$

Practical Tricks and Heuristics for Speeding Up Learning

- In practice, training of deep neural networks involves a lot of tricks and heuristics
- Initialization of weights: smaller incoming weights when the fan-in is big (proportional to square root of fan in)
- Scaling, shifting, decorrelating of inputs: zero mean, unit variance, PCA, ...
- Separate adaptive learning rate for each parameter
- RMSprop: Divide the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight

Hinton's RMSProp

- RMSProp is an unpublished gradient descent method by Hinton [Neural Networks for Machine Learning]
- RMSProp refers to root mean square propagation
- Gradient descent in RMSProp,

$$v_t = \beta v_{t-1} + (1-\beta)(\nabla_\theta J)^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t + \epsilon}}(\nabla_\theta J)$$

- In RMSProp,
  - The learning rate of a parameter is inversely proportional to square root of exponential average of the gradients w.r.t the parameter
  - Adaptive learning rates
  - Different learning rates for different parameters
  - RMSProp impedes search in direction of oscillations
- Typically, $\beta = 0.9$ and $\epsilon = 10^{-10}$

Initialization and Momentum

On the importance of initialization and momentum in deep learning

Ilya Sutskever[1] ILYASU@GOOGLE.COM
James Martens JMARTENS@CS.TORONTO.EDU
George Dahl GDAHL@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU

Sutskever et al. (2013)

ADAM Method

ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION

Diederik P. Kingma*
University of Amsterdam, OpenAI
dpkingma@openai.com

Jimmy Lei Ba*
University of Toronto
jimmy@psi.utoronto.ca

ABSTRACT

We introduce *Adam*, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning. Some connections to related algorithms, on which *Adam* was inspired, are discussed. We also analyze the theoretical convergence properties of the algorithm and provide a regret bound on the convergence rate that is comparable to the best known results under the online convex optimization framework. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods. Finally, we discuss *AdaMax*, a variant of *Adam* based on the infinity norm.

Kingma and Ba (2015)

ADAM Optimizer

Adam: A Method for Stochastic Optimization, Kingma and Ba, 2014

- Momentum accelerates convergence to minima
- RMSProp impedes search in direction of oscillations
- ADAM: adaptive moment optimization; combines best of both worlds
- Gradient descent in ADAM,

$$v_t = \beta_1 v_{t-1} + (1 - \beta_1)(\nabla_\theta J)^2$$

$$s_t = \beta_2 s_{t-1} + (1 - \beta_2)(\nabla_\theta J)^2$$

$$\theta_{t+1} = \theta_t - \frac{v_t}{\sqrt{s_t + \epsilon}}(\nabla_\theta J)$$

- The learning rate is,
  - proportional to the exponential average of the gradients
  - inversely proportional to square root of exponential average of the gradients
- Typically, $\beta_1 = 0.9$, $\beta_2 = 0.99$ and $\epsilon = 10^{-10}$

On the Insufficiency of Existing Momentum Schemes for Stochastic Optimization

Rahul Kidambi Praneeth Netrapalli Prateek Jain Sham Kakade

Kidambi et al. (2018)

Abstract from Kidambi et al. (2018)

"Momentum based stochastic gradient methods such as heavy ball (HB) and Nesterov's accelerated gradient descent (NAG) method are widely used in practice for training deep networks . . . they often provide significant improvements over stochastic gradient descent (SGD). In general, "fast gradient" methods have provable improvements over gradient descent only for the deterministic case, where the gradients are exact. In the stochastic case, the popular explanations for their wide applicability is that when these fast gradient methods are applied in the stochastic case, they partially mimic their exact gradient counterparts, resulting in some practical gain. This work provides a counterpoint to this belief by proving that there are simple problem instances where these methods cannot outperform SGD despite the best setting of its parameters. . . . These results suggest (along with empirical evidence) that HB or NAG's practical performance gains are a by-product of minibatching. Furthermore, this work provides a viable (and provable) alternative, . . . significantly improves over HB, NAG, and SGD's performance. This algorithm, denoted as ASGD, is a simple to implement stochastic algorithm, based on a relatively less popular version of Nesterov's AGD. Extensive empirical results in this paper show that ASGD has performance gains over HB, NAG, and SGD."

(Why) Does Deep Learning Paradigm Work?

Approximation Properties

- Classical result: Neural network with a single hidden layer can approximate any continuous function on a compact domain
- Current wisdom: deep networks have significant benefits
- Some theoretical results to show deep networks have exponential benefits over shallow networks
- Much work on expressive power of deep networks
- Minimization of training error is NP hard yet training can be done in practice
- Issues in training: saddle points in high-dimensional non-convex optimization

Generalization Properties

- Deep networks — too many parameters compared to training data
- (Why) do deep networks generalize well away from where they are trained?
- Insufficient and incomplete understanding of deep learning generalization performance
  1. Standard theoretical tools: VC dimension, Rademacher complexity, uniform stability
  2. Recent result: deep neural networks easily fit random labels [Understanding deep learning requires rethinking generalization, Zhang et al. (2016)]
  3. Regularization is neither necessary nor sufficient for generalization

Generalization Properties not Understood

Understanding Deep Learning Requires Rethinking Generalization, Zhang et al. (2016)

"Despite their massive size, successful deep artificial neural networks can exhibit a remarkably small difference between training and test performance. Conventional wisdom attributes small generalization error either to properties of the model family, or to the regularization techniques used during training. Through extensive systematic experiments, we show how these traditional approaches fail to explain why large neural networks generalize well in practice. Specifically, our experiments establish that state-of-the-art convolutional networks for image classification trained with stochastic gradient methods easily fit a random labeling of the training data. This phenomenon is qualitatively unaffected by explicit regularization, and occurs even if we replace the true images by completely unstructured random noise."

Gradient Descent Convergence

Gradient Descent Finds Global Minima of Deep Neural Networks

Simon S. Du [*1] Jason D. Lee [*2] Haochuan Li [*3 4] Liwei Wang [*5 4] Xiyu Zhai [*6]

Abstract

Gradient descent finds a global minimum in training deep neural networks despite the objective function being non-convex. The current paper proves gradient descent achieves zero training loss in polynomial time for a deep over-parameterized neural network with residual connections (ResNet). Our analysis relies on the particular structure of the Gram matrix induced by the neural network architecture. This structure allows us to show the Gram matrix is stable throughout the training process and this stability implies the global optimality of the gradient descent algorithm. We further extend our analysis to deep residual convolutional neural networks and obtain a similar convergence result.

The second mysterious phenomenon in training deep neural networks is "deeper networks are harder to train." To solve this problem, He et al. (2016) proposed the deep residual network (ResNet) architecture which enables randomly initialized first order method to train neural networks with an order of magnitude more layers. Theoretically, Hardt & Ma (2016) showed that residual links in linear networks prevent gradient vanishing in a large neighborhood of zero, but for neural networks with non-linear activations, the advantages of using residual connections are not well understood.

In this paper, we demystify these two mysterious phenomena. We consider the setting where there are n data points, and the neural network has H layers with width m. We focus on the least-squares loss and assume the activation function is Lipschitz and smooth. This assumption holds

Du et al. (2019)

Advantages of Depth in Approximation Power

On the Number of Linear Regions of Deep Neural Networks

Guido Montúfar
Max Planck Institute for Mathematics in the Sciences
montufar@mis.mpg.de

Razvan Pascanu
Université de Montréal
pascanur@iro.umontreal.ca

Kyunghyun Cho
Université de Montréal
kyunghyun.cho@umontreal.ca

Yoshua Bengio
Université de Montréal, CIFAR Fellow
yoshua.bengio@umontreal.ca

Montufar et al. (2014)

Exponential Growth in the Number of Linear Regions

Theorem: Consider a neural network with m inputs, L layers, and with each layer containing $n \geq m$ neurons with RelU activation function. Such a network can represent piecewise linear functions with $\Omega\left(\left(\frac{n}{m}\right)^{L-1} n^m\right)$ regions.

Pascanu et al. (2013), Montufar et al. (2014)

Advantages of Depth in Approximation Power

The Power of Depth for Feedforward Neural Networks

Ronen Eldan RONEN.ELDAN@WEIZMANN.AC.IL
Weizmann Institute of Science, Rehovot, Israel

Ohad Shamir OHAD.SHAMIR@WEIZMANN.AC.IL
Weizmann Institute of Science, Rehovot, Israel

Abstract

We show that there is a simple (approximately radial) function on $\mathbb{R}^d$, expressible by a small 3-layer feedforward neural networks, which cannot be approximated by any 2-layer network, to more than a certain constant accuracy, unless its width is exponential in the dimension. The result holds for virtually all known activation functions, including rectified linear units, sigmoids and thresholds, and formally demonstrates that depth – even if increased by 1 – can be exponentially more valuable than width for standard feedforward neural networks. Moreover, compared to related results in the context of Boolean functions, our result requires fewer assumptions, and the proof techniques and construction are very different.

Eldan and Shamir (2016)

Advantages of Depth in Approximation Power

International Journal of Automation and Computing

14(5), October 2017, 503-519
DOI: 10.1007/s11633-017-1054-2

Why and When Can Deep-but Not Shallow-networks Avoid the Curse of Dimensionality: A Review

Tomaso Poggio[1] Hrushikesh Mhaskar[2, 3] Lorenzo Rosasco[1] Brando Miranda[1] Qianli Liao[1]

[1]Center for Brains, Minds, and Machines, McGovern Institute for Brain Research, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
[2]Department of Mathematics, California Institute of Technology, Pasadena, CA 91125, USA
[3]Institute of Mathematical Sciences, Claremont Graduate University, Claremont, CA 91711, USA

Abstract: The paper reviews and extends an emerging body of theoretical results on deep learning including the conditions under which it can be exponentially better than shallow learning. A class of deep convolutional networks represent an important special case of these conditions, though weight sharing is not the main reason for their exponential advantage. Implications of a few key theorems are discussed, together with new results, open problems and conjectures.

Poggio et al. (2017)

Issue of Local Minima

- Given the large numbers of parameters in deep neural networks, one might be tempted to think there will be lots of bad local minima in the parameter space for the non-convex optimization problem.
- It turns out that there are the local minima generally tend to give similar, close to optimal, performance.
- But there are many saddle points.

Loss Surface of Multilayer Neural Networks

The Loss Surfaces of Multilayer Networks

Anna Choromanska achoroma@cims.nyu.edu
Mikael Henaff mbh305@nyu.edu
Michael Mathieu mathieu@cs.nyu.edu
Gérard Ben Arous benarous@cims.nyu.edu
Yann LeCun yann@cs.nyu.edu

Courant Institute of Mathematical Sciences
New York, NY, USA

Chromanska et al. (2015)

Results from Chromanska et al. (2015)

- Connection to spherical spin-glass models
- "Loss surfaces for large neural nets have many local minima that are essentially equivalent from the point of view of the test error, and these minima tend to be highly degenerate, with many eigenvalues of the Hessian near zero."
- (Empirical finding) "For large-size networks, most local minima are equivalent and yield similar performance on a test set."
- (Empirical finding) "The probability of finding a bad (high value) local minimum is non-zero for small-size networks and decreases quickly with network size."

DL for Sequences: RNN and LSTM

Recurrent Neural Networks

- RNN: Recurrent Neural Networks are a special type of neural networks
- They are ideal for modeling sequences because of they allow for dependence of the current output on past data. Also they can model non-linear dependence across time.

Recurrent Neural Networks

- Hopfield introduced recurrent neural networks in 1982.
- Recurrent neural networks (RNNs) are connectionist models with the ability to selectively pass information across sequence steps, while processing sequential data one element at a time.
- Hence they are suitable for modeling data that are
  - Sequential and dependent.
  - Of varying input lengths.
- And so RNNs are also a natural choice for time series and other sequential applications. But RNNs were inferior to DNNs till 2013 in speech recognition tasks, Hinton et al. (2013).
- The breakthrough for RNNs came in 2013 when Hinton et al. used a bidirectional deep recurrent neural network with LSTM to achieve the then state-of-the-art performance in TIMIT database.

History: Jordan Network

- Jordan introduced supervised learning on sequences in 1986 in a special type of network.
- Feedforward network with a single hidden layer that is extended with special units.
- Output node values are fed to the special units which in turn feed these values to the hidden nodes at the *next time step*.
- Special units are self-connected.

Jordan RNN

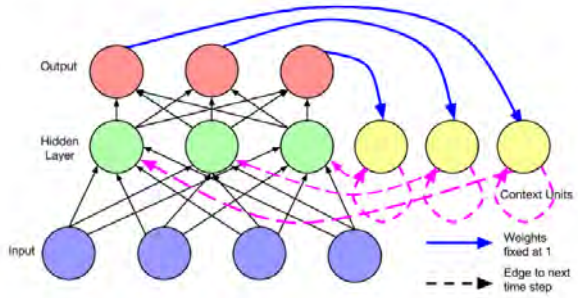


Figure 5: A recurrent neural network as proposed by Jordan [1986]. Output units are connected to special units that at the next time step feed into themselves and into hidden units.

History: Elman Network

- One context unit for each hidden node. Input to the context unit is the value of the hidden units from the previous time step. Output of the context unit is fed into the hidden unit at the current time step. This is very similar to the vanilla RNN as we know it today.

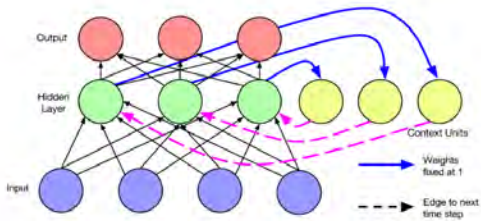


Figure 6: A recurrent neural network as described by Elman [1990]. Hidden units are connected to context units, which feed back into the hidden units at the next time step.

RNN and LSTM

- Two key papers:

  Z. C. Lipton, J. Berkovoitz and C. Elkan, "A Critical Review of Recurrent Neural Networks for Sequence Learning," arXiv:1506.00019 [cs.LG]
  [This paper contains a very readable and succinct account of back propagation and network training equations in Section 2.3.]

  K. Greff et al., LSTM: A Search Space Odyssey, IEEE Trans. on Neural Networks and Learning Systems, vol. 28, pp. 2222-2232, 2017.

Simple RNN Notation

- h^t: hidden layer vector at time t.
- x^t: input vector at time t.
- y^t: output vector at time t.
- Recurrent units at time t receive inputs from time t and also the hidden value from the previous time step h_{t-1}.

Simple RNN

- $h^t = \sigma(W_h x^t + R_h h^{t-1} + b_h)$
- $y^t = \text{soft-max}(W_y h^t + b_y)$

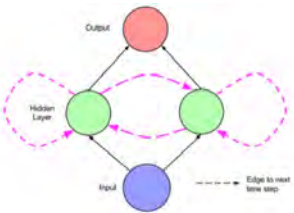


Figure 3: A simple recurrent network. At each time step t, activation is passed along solid edges as in a feedforward network. Dashed edges connect a source node at each time t to a target node at each following time $t + 1$.

In this figure, RNN is shown to have only two hidden layer neurons but the equations are completely general!

Unfolded RNN

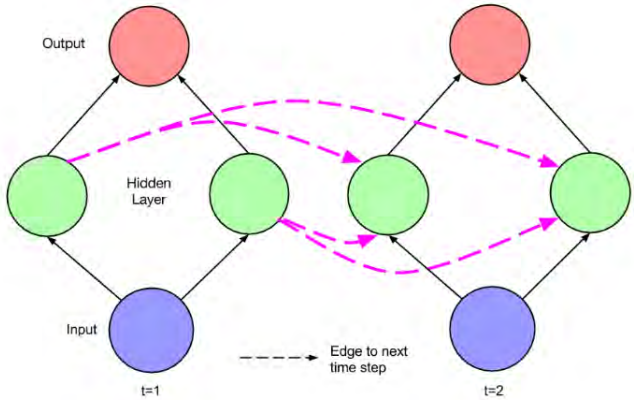


Figure 4: The recurrent network of Figure 3 unfolded across time steps.

Backpropagation Through Time (BPTT)

BPTT is is the standard backpropagation but applied to RNN and is used for training them

- Forward propagation,
  $h^t = \sigma(W_h x^t + R_h h^{t-1} + b_h)$
  $y^t = \text{soft-max}(W_y h^t + b_y)$.
- Error estimation, $E^t = (y^t - y_d^t)^2$. where y_t^d is the true output corresponding to input sequence x^t.
- W_h, R_h and b_h are updated by backpropagation of error through time (next slide).

By chain rule it follows that,

$$\frac{\partial E^t}{\partial W_h} = \frac{\partial E^t}{\partial y^t} \frac{\partial y^t}{\partial W_h} \tag{22}$$

$\Rightarrow$

$$\frac{\partial E^t}{\partial W_h} = \frac{\partial E^t}{\partial y^t} \frac{\partial y^t}{\partial h^t} \left(\frac{\partial h^t}{\partial W_h} + \frac{\partial h^t}{\partial h^{t-1}} (....) \right) \tag{23}$$

$\Rightarrow$

$$\frac{\partial E^t}{\partial W} = \frac{\partial E^t}{\partial h} \left(\frac{\partial h^t}{\partial W} + \frac{\partial h^t}{\partial h^{t-1}} \left(\frac{\partial h^{t-1}}{\partial W_h} + \frac{\partial h^{t-1}}{\partial h^{t-2}} (...) \right) \right) \tag{24}$$

and so on ...

This is error backpropagation through time (BPTT).

Gradient Descent Update

In practice, we truncate backpropagation up to a certain maximum N. This is an approximation to BPTT and so,

$$\frac{\partial E^t}{\partial W_h} \approx \sum_{k=0}^{N} \frac{\partial E^t}{\partial y^t} \frac{\partial y^t}{\partial h^t} \frac{\partial h^t}{\partial h^{t-1}} \dots \frac{\partial h^{t-k-1}}{\partial h^{t-k}} \frac{\partial h_{t-N}}{\partial W} \tag{25}$$

And update,

$$W_h(t+1) = W_h(t) + \alpha_t \frac{\partial E^t}{\partial W_h} = W(t) + \alpha_t \sum_{k=0}^{N} \frac{\partial E^t}{\partial h^t} \frac{\partial y^t}{\partial h^t} \frac{\partial h^t}{\partial h^{t-k}} \dots \frac{\partial h^{t-N-1}}{\partial h^{t-N}} \frac{\partial h^{t-N}}{\partial W_h} \tag{26}$$

Problem of Vanishing Gradients

BPTT suffers from the problem of vanishing gradients

Recall approximation to BPTT,

$$\frac{\partial E_t}{\partial W} \approx \sum_{k=0}^{N} \frac{\partial E_t}{\partial h_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_{t-k-1}}{\partial h_{t-k}} \frac{\partial h_{t-N}}{\partial W_h} \tag{27}$$

Suppose, $|\frac{\partial h_{t'}}{\partial h_{t'-1}}| < \delta$ for every t' and note that δ depends on weight vector W in our case.

Then $|\frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_{t-k-1}}{\partial h_{t-k}}| < \delta^k$. If $\delta < 1$, then δ^k approaches zero at an exponential rate with increase in k. This is the problem of vanishing gradient.

Reference for more reading: R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," ICML, pp. 1310-1318, 2013.

LSTM

LSTM

- The problem of vanishing gradients can be avoided by truncating BPTT within few time steps.
- But such a premature truncation affects learning of long range dependencies.
- LSTMs were introduced to resolve this issue.
- The first paper in LSTM was published by Hochreiter and Schmidhuber '97.
- The second paper by Schuster and Paliwal '97 introduced bidirectional RNNs.
- The third paper by Gers et al., 2000 introduced the forget gate.

First LSTM Cell

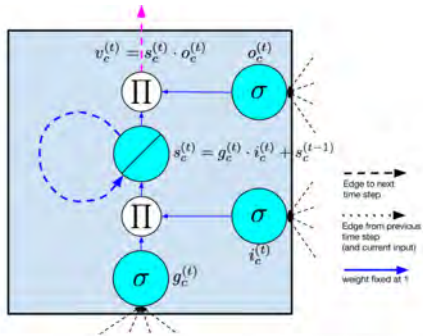


Figure 9: One LSTM memory cell as proposed by Hochreiter and Schmidhuber [1997]. The self-connected node is the internal state s. The diagonal line indicates that it is linear, i.e. the identity link function is applied. The blue dashed line is the recurrent edge, which has fixed unit weight. Nodes marked Π output the product of their inputs. All edges into and from Π nodes also have fixed unit weight.

LSTM with Forget Gate

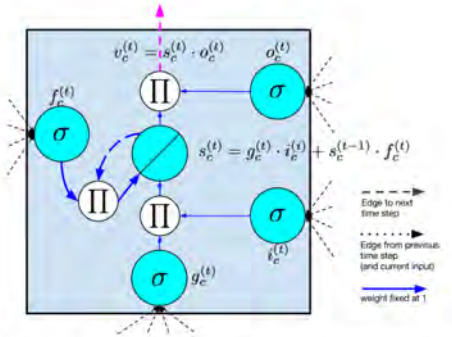


Figure 10: LSTM memory cell with a forget gate as described by Gers et al. [2000].

Forget gates have been effective and are a standard in modern applications of LSTM

Unfolded LSTM

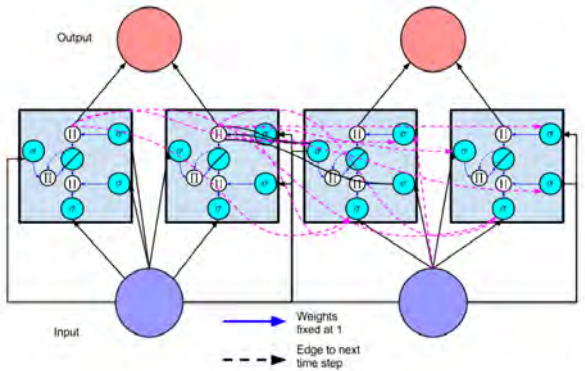


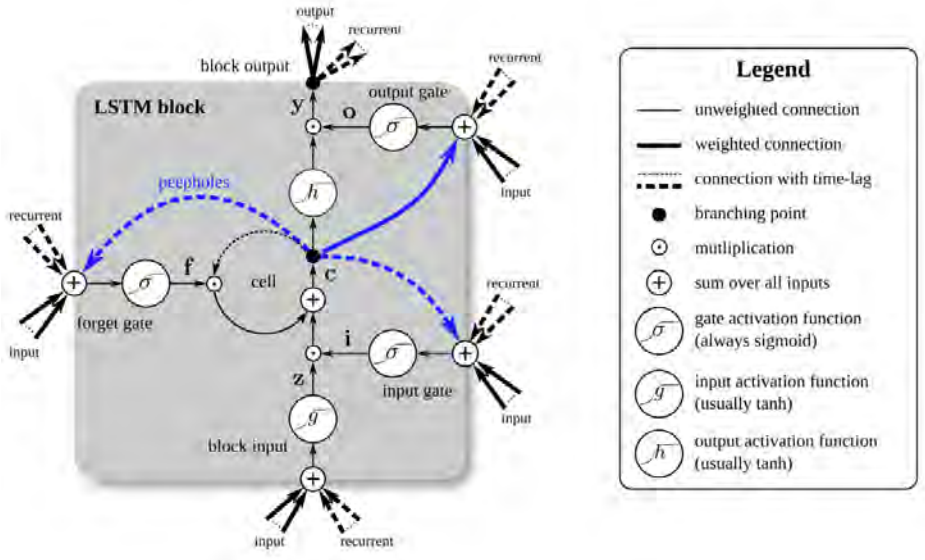
Figure 11: A recurrent neural network with a hidden layer consisting of two memory cells. The network is shown unfolded across two time steps.

Recent LSTM Paper

LSTM: A Search Space Odyssey

Klaus Greff, Rupesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber

Vanilla LSTM Cell



Key Elements of Vanilla LSTM

- State
- Gates
- Block input
- A single cell
- An output activation function
- Peephole connections

Gates

- A gate is a sigmoidal unit that, like the input node, takes activation from the current data point as well as from the hidden layer at the previous time step.
- A gate is so-called because its value is used to multiply the value of another node. It is a gate in the sense that if its value is zero, then flow from the other node is cut.
- There are three gates: input, output and forget.
- The gates along with the state of the cell act like a memory unit. The output, input, and forget simulate the memory operations read, write and reset, [Framewise phoneme classification with bidirectional LSTM and other neural network architectures, Graves and Schimdhuber (2005)].

Peephole Connections

- They were first introduced by Gers et al. (2000) to improve performance on tasks where timing is critical: "were added to the architecture in order to make precise timings easier to learn".

LSTM Notation

Let N be the number of LSTM blocks, and M the number of inputs. Then we get the following inputs for the LSTM layer,

Input Weights : $W_z, W_s, W_f, W_o \in \mathbb{R}^{N \times M}$

Recurrent Weights : $R_z, R_s, R_f, R_o \in \mathbb{R}^{N \times M}$

Peephole Weights : $p_s, p_f, p_o \in \mathbb{R}^N$

Bias Weights : $b_z, b_s, b_f, b_o \in \mathbb{R}^N$

LSTM Equations

Let x^t be the input vector at time t,

Block Input: $z^t = g(W_z x^t + R_z y^{t-1} + b_z)$

Input Gate: $i^t = \sigma(W_i x^t + R_i y^{t-1} + p_i \odot c^{t-1} + b_i)$

Forget Gate: $f^t = \sigma(W_f x^t + R_f y^{t-1} + p_f \odot c^{t-1} + b_f)$

Output Gate: $o^t = \sigma(W_o x^t + R_o y^{t-1} + p_o \odot c^{t-1} + b_o)$

Cell: $c^t = z^t \odot i^t + c^{t-1} \odot f^t$

Block Output: $y^t = h(c^t) \odot o^t$

Equations for Back Propagation Through Time (BPTT)

$$\delta y^t = \Delta^t + R_z^T \delta z^{t+1} + R_i^T \delta i^{t+1} + R_f^T \delta f^{t+1} + R_o^T \delta o^{t+1}$$

$$\delta c^t = \delta y^t \odot o^t \odot h^{'}(c^t) + p_o \odot \delta o^t + p_i \odot \delta i^{t+1} + p_f \odot \delta f^{t+1} + \delta c^{t+1} \odot f^{t+1}$$

$$\delta f^t = \delta c^t \odot c^{t-1} \odot \sigma^{'}(W_f x^t + R_f y^{t-1} + p_f \odot c^{t-1} + b_f)$$

$$\delta i^t = \delta c^t \odot z^t \odot \sigma^{'}(W_i x^t + R_i y^{t-1} + p_i \odot c^{t-1} + b_i)$$

$$\delta o^t = \delta y^t \odot h(c^t) \odot \sigma^{'}(W_o x^t + R_o y^{t-1} + p_o \odot c^{t-1} + b_o)$$

$$\delta z^t = \delta c^t \odot i^t \odot g^{'}(W_z x^t + R_z y^{t-1} + b_z)$$

Equations for Gradient of Weights

The gradients for the weights are calculated as follows, where $\star$ can be any of $\{z, i, f, o\}$, and $< \star_1, \star_2 >$ denotes the outer product of two vectors,

$$\delta W_\star = \sum_{t=0}^{T} < \delta\star^t, x^t > \qquad \delta p_i = \sum_{t=0}^{T} c^t \odot \delta i^{t+1}$$

$$\delta R_\star = \sum_{t=0}^{T} < \delta\star^{t+1}, y^t > \qquad \delta p_f = \sum_{t=0}^{T} c^t \odot \delta f^{t+1}$$

$$\delta b_\star = \sum_{t=0}^{T} \delta \star^t \qquad \delta p_o = \sum_{t=0}^{T} c^t \odot \delta o^t$$

Other Variants of LSTM

Discussed in Section 3-D of the paper by Greff et al.

- Training methods using extended Kalman filter and evolution based methods.
- Different LSTM block architecture by Bayer et al.
- Dynamic cortex memory by Otte et al.
- Gated Recurrent Units by Cho et al.

Empirical Evaluation of Different LSTM Methods

A major goal of the paper by Greff et al.

Three data sets

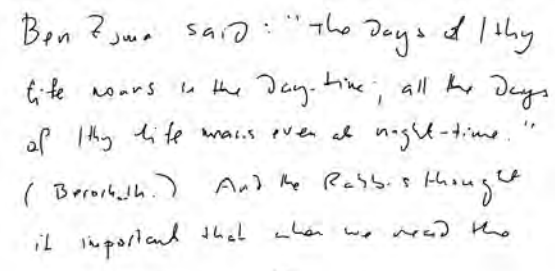
- TIMIT Speech corpus for speech recognition tasks
- IAM Online Handwriting Database
- JSB Chorales - 382 four part harmonized chorales by Bach

Network Architecture and Training Details

B. Network Architectures & Training

A network with a single LSTM hidden layer and a sigmoid output layer was used for the JSB Chorales task. Bidirectional LSTM [20] was used for TIMIT and IAM Online tasks, consisting of two hidden layers, one processing the input forward, and the other one backward in time, both connected to a single softmax output layer. As loss function, we employed Cross-Entropy Error for TIMIT and JSB Chorales, while for the IAM Online task, the CTC loss by Graves *et al.* [39] was used. The initial weights for all networks were drawn from a normal distribution with a standard deviation of 0.1. Training was done using stochastic gradient descent with Nesterov-style momentum [42] with updates after each sequence. The learning rate was rescaled by a factor of $(1 - \text{momentum})$. Gradients were computed using full BPTT for LSTMs [20]. Training stopped after 150 epochs or once there was no improvement on the validation set for more than 15 epochs.

IAM Online Handwriting Example



(a)

(b)

Fig. 2. (a) Example board (a08-551z, training set) from the IAM-OnDB dataset and (b) its transcription into character label sequences.

Summary of LSTM Variants

- No input gate (NIG): $i^t = 1$.
- No forget gate (NFG): $f^t = 1$.
- No output gate (NOG)): $o^t = 1$.
- No input activation function (NIAF): $f(x) = x$.
- No output activation function (NOAF): $h(x) = 1$.
- Coupled input and forget gate (CIFG): $f^t = 1 - i^t$.
- LSTM without peepholes (NP): $p_i = p_f = p_o = 0$.

LSTM Variant: Full Gate Recurrence

Full Gate Reurrence:

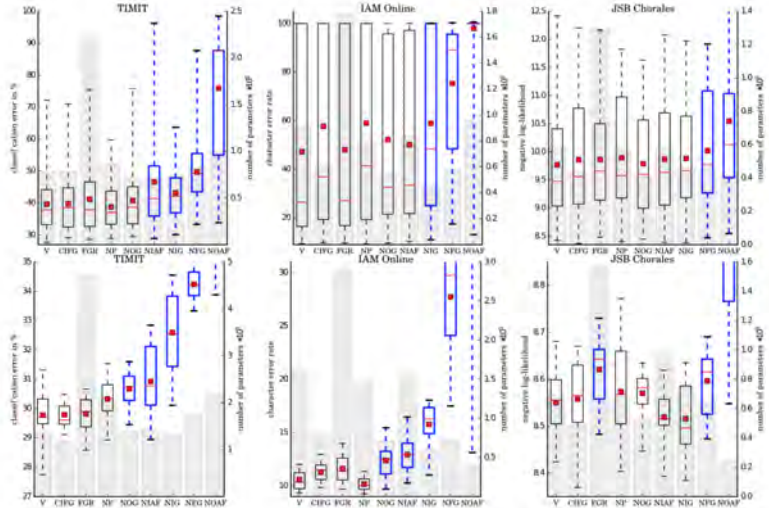
Input Gate: $i^t = \sigma(W_i x^t + R_i y^{t-1} + p_i \odot c^{t-1} + b_i + R_{ii} i^{t-1} + R_{fi} f^{t-1} + R_{oi} o^{t-1})$

Forget Gate: $f^t = \sigma(W_f x^t + R_f y^{t-1} + p_f \odot c^{t-1} + b_f + R_{if} i^{t-1} + R_{ff} f^{t-1} + R_{of} o^{t-1})$

Output Gate: $o^t = \sigma(W_o x^t + R_o y^{t-1} + p_o \odot c^{t-1} + b_o + R_{io} i^{t-1} + R_{fo} f^{t-1} + R_{oo} o^{t-1})$

"Another feature ... was the use of full gate recurrence (FGR), which means that all the gates received recurrent inputs from all the gates at the previous time step in addition to the recurrent inputs from the block outputs."

Performance Comparison



Observations

- Removing the output activation function (NOAF) or the forget gate (NFG) significantly hurt performance.
- Removing the input gate (NIG), the output gate (NOG), and the input activation function (NIAF) led to a significant reduction.
- Input and forget gate coupling (CIFG) did not significantly change the mean performance on any of the data sets.
- Removing peephole connections (NP) also did not lead to significant changes.
- Adding FGR did not significantly change performance on TIMIT or IAM Online, but led to worse results on the JSB Chorales data set.

Impact of Hyperparameters

- Learning Rate: "optimal value for the learning rate is dependent on the data set. For each data set, there is a large basin (up to two orders of magnitude) of good learning rates inside of which the performance does not vary much."
- Hidden Layer Size: "larger networks perform better, but with diminishing returns."
- Momentum: "One unexpected result of this paper is that momentum affects neither performance nor training time in any significant way."

Conclusions of Greff et al. paper

"Neural networks can be tricky to use for many practitioners compared with other methods whose properties are already well understood. This has remained a hurdle for newcomers to the field, since a lot of practical choices are based on the intuitions of experts, as well as experiences gained over time. With this paper, we have attempted to back some of these intuitions with experimental results. We have also presented new insights, both on architecture selection and hyperparameter tuning for LSTM networks ..."

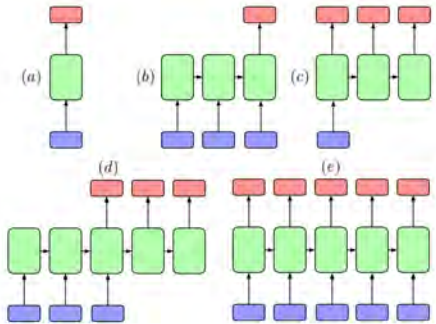
Applications

Applications

- Speech recognition.
- Handwriting recognition.
- Image captioning.
- Machine translation.
- Speech synthesis.
- Language models.

Implementation vary Across Applications

Lipton et al. (2015)



a) standard neural network b) text, audio and video classification c) image captioning d) sequence to sequence like machine translation e) language models.

Machine Translation

- Machine translation is challenging because the words have to be in an order.
- Output at each step is a soft-max output over the vocabulary.
- The inputs that represent texts are either binary representations or *meaning vectors*, learnt from supervisory learning. Current applications use meaning vectors as inputs.
- Sutskever et al. (2014) introduced a translation model using two multilayered LSTMs that demonstrated impressive performance translating from English to French.
- The first LSTM is used for encoding an input phrase from the source language and the
- Second LSTM for decoding the output phrase in the target language.
- Sutskever's Model was state-of-the-art in 2014 beating all previous models.

Sutskever's Machine Translator

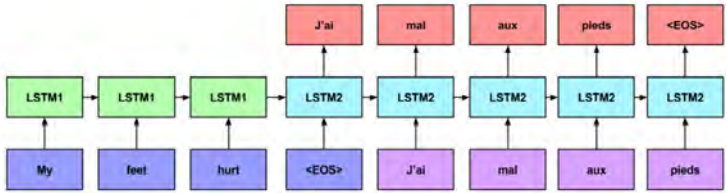


Figure 14: Sequence to sequence LSTM model of Sutskever et al. [2014]. The network consists of an encoding model (first LSTM) and a decoding model (second LSTM). The input blocks (blue and purple) correspond to word vectors, which are fully connected to the corresponding hidden state. Red nodes are softmax outputs. Weights are tied among all encoding steps and among all decoding time steps.

Google's Neural Machine Translator

Google's NMT is one of the current benchmarks.

Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi
yonghui,schuster,zhifengc,qvl,mnorouzi@google.com

Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, Jeffrey Dean

Abstract

Neural Machine Translation (NMT) is an end-to-end learning approach for automated translation, with the potential to overcome many of the weaknesses of conventional phrase-based translation systems. Unfortunately, NMT systems are known to be computationally expensive both in training and in translation inference – sometimes prohibitively so in the case of very large data sets and large models. Several authors have also charged that NMT systems lack robustness, particularly when input sentences contain rare words. These issues have hindered NMT's use in practical deployments and services, where both accuracy and speed are essential. In this work, we present GNMT, Google's Neural Machine Translation system, which attempts to address many of these issues. Our model consists of a deep LSTM network with 8 encoder and 8 decoder layers using residual connections as well as attention connections from the decoder network to the encoder. To improve parallelism and therefore decrease training time, our attention mechanism connects the bottom layer of the decoder to the top layer of the encoder. To accelerate the final translation speed, we employ low-precision arithmetic during inference computations. To improve handling of rare words, we divide words into a limited set of common sub-word units ("wordpieces") for both input and output. This method provides a good balance between the flexibility of "character"-delimited models and the efficiency of "word"-delimited models, naturally handles translation of rare words, and ultimately improves the overall accuracy of the system. Our beam search technique employs a length-normalization procedure and uses a coverage penalty, which encourages generation of an output sentence that is most likely to cover all the words in the source sentence. To directly optimize the translation BLEU scores, we consider refining the models by using reinforcement learning, but we found that the improvement in the BLEU scores did not reflect in the human evaluation. On the WMT'14 English-to-French and English-to-German benchmarks, GNMT achieves competitive results to state-of-the-art. Using a human side-by-side evaluation on a set of isolated simple sentences, it reduces translation errors by an average of 60% compared to Google's phrase-based production system.

1 Introduction

Neural Machine Translation (NMT) [41, 2] has recently been introduced as a promising approach with the potential of addressing many shortcomings of traditional machine translation systems. The strength of NMT lies in its ability to learn directly, in an end-to-end fashion, the mapping from input text to associated output text. Its architecture typically consists of two recurrent neural networks (RNNs), one to consume the input text sequence and one to generate translated output text. NMT is often accompanied by an attention mechanism [2] which helps it cope effectively with long input sequences.

An advantage of Neural Machine Translation is that it sidesteps many brittle design choices in traditional phrase-based machine translation [26]. In practice, however, NMT systems used to be worse in accuracy than

Google's NMT Uses Stacked LSTMs with Residual Connections

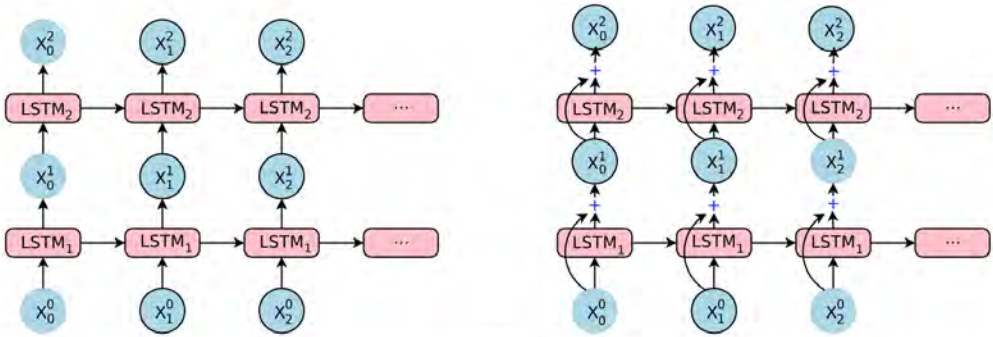
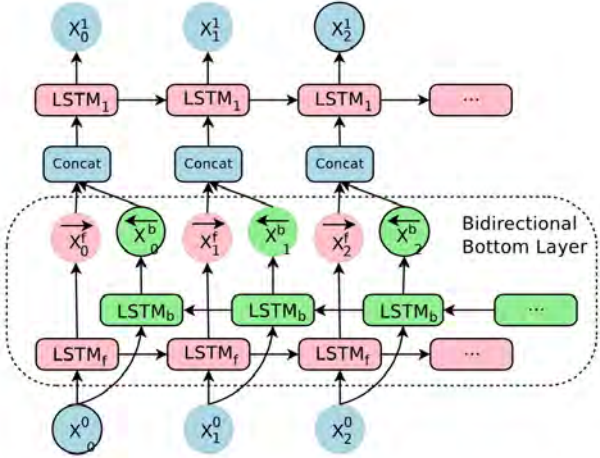


Figure 2: The difference between normal stacked LSTM and our stacked LSTM with residual connections. On the left: simple stacked LSTM layers [41]. On the right: our implementation of stacked LSTM layers with residual connections. With residual connections, input to the bottom LSTM layer ($\mathbf{x_i^0}$'s to LSTM_1) is element-wise added to the output from the bottom layer ($\mathbf{x_i^1}$'s). This sum is then fed to the top LSTM layer (LSTM_2) as the new input.

Google's NMT Encoder is a Bi-directional RNN

For translation systems, the information required to translate certain words on the output side can appear anywhere on the source side. In this sense, bi-directional RNNs offers a better context for a particular word output.



Metrics for Text Output

- Performance metric is not straightword because complete output is a text of varying length.
- A very standard metric to assess performance is BLEU, Lipton et al. (2015).
- BLEU measures accuracy by computing the geometric mean of n-gram precision, where n can vary from 1 to an upper limit N.
- BLEU scores are highly correlated with human judgement of what a good translation is on a large set of sentences but do not reflect accuracy on small set of translations.
- A more complex metric called METEOR was introduced by Banerjee and Lavie (2005) to overcome deficiencies in BLEU. METEOR agrees better with human judgement.

Results on Production Data

Table 10: Mean of side-by-side scores on production data

	PBMT	GNMT	Human	Relative Improvement
English → Spanish	4.885	5.428	5.504	87%
English → French	4.932	5.295	5.496	64%
English → Chinese	4.035	4.594	4.987	58%
Spanish → English	4.872	5.187	5.372	63%
French → English	5.046	5.343	5.404	83%
Chinese → English	3.694	4.263	4.636	60%

Image Captioning

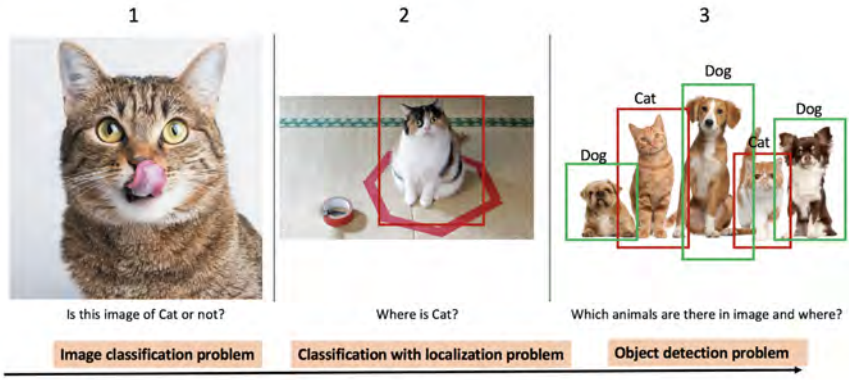
- Image captioning refers to generating text that describes the image.
- Inputs are images and outputs are captions.
- Vinyals et al. (2015) proposed a method similar to language translation following up on its success.
- An encoder that uses convolutional neural network for the encoding part and a LSTM decoder for the decoding part.
- Mao et al. (2014) independently developed a similar RNN image captioning network, and achieved then state-of-the-art results on the Pascal, Flickr30K, and COCO datasets.

Handwriting Recognition

- The main architecture is bi-directional LSTMs.
- Initial success in the late '00s, Graves et al., 2009.
- As of 2015, bi-directional LSTMs were the state-of-the-art achieving a word-level accuracy of 81.5% compared to HMMs 70.1%.

Deep Learning for Computer Vision

Detection



Source: evolution of object detection algorithms

Object Detection: Traditional RCNN

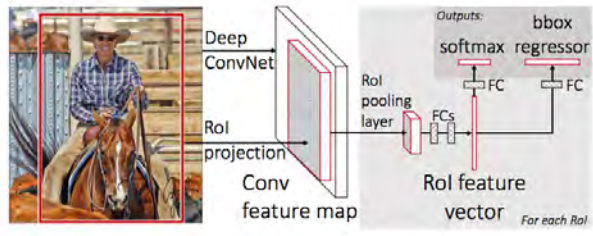
- RCNN: Regional Convolutional Neural Network
- Traditional RCNN: Regions are proposed as bounding boxes
- Each bounding box is run through a classifier. Finally the bounding boxes are refined through another regressor. Training happens in several stages:
  - Region Proposals
  - Learning the classifier
  - Learning the bounding box refinement
- Traditional RCNNs are slower in inference because the learned classifier runs a forward pass for each proposed region

Fast RCNN

- Fast RCNN simplifies this inference by reducing this multi-forward pass to a single forward pass. It also simplifies the multistep learning process.
- Key idea: ROI pooling
- The training is simplified and inference speed and accuracy are improved.
- One of the current leading frameworks.

Fast RCNN Architecture

In this architecture, the image is passed only once through the deep convolutional net



Faster RCNN

Faster RCNN: Towards Real-Time Object Detection with Region Proposal Networks, Ren et al., 2015

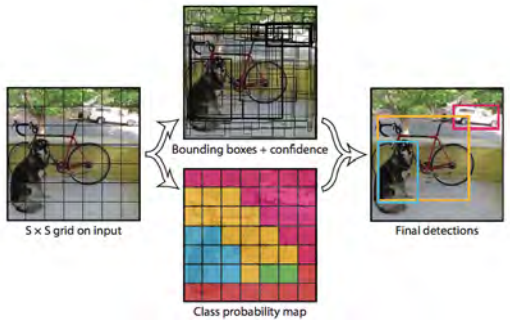
- Fast RCNN + Region Proposal Networks
- Region Proposal Networks: generates region proposals, which are used by Fast R-CNN for detection

YOLO: Proposal free method

- YOLO: You Only Look Once
- Proposal free method
- A single deep neural network is learned end to end by backpropagation training
- It looks only once at the image and detects the objects and its bounding boxes
- Competitive with Faster RCNNs

YOLO Framework

The input is divided in to grid of cells and the network predicts B bounding boxes and a confidence level for each cell that the bounding box contains the object that the grid cell is part of, which is then translated to the final detection output



YOLO 9000

YOLO 9000: YOLO9000: Better, Faster, Stronger, Redmon et al., 2016

YOLO 9000 can detect more than 9000 object categories

Segmentation

Segmentation $\equiv$ pixel wise prediction

Standard framework: Fully Convolutional Networks, Long et al., (2015)

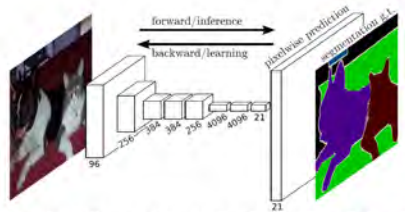


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

Mask RCNN

Mask R-CNN, He et al., 2018

- A framework for detection and segmentation
- Extends Faster RCNN by including a branch for prediction of segmentation mask
- Mask prediction branch: a small FCN applied to each ROI
- Key feature: ROI align instead of ROI pooling
- ROI align designed to avoid misalignments in predicted masks and real image

ROI Align

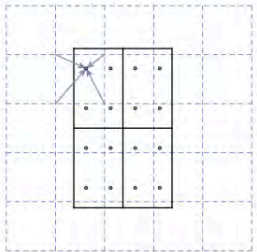


Figure 3. **RoIAlign:** The dashed grid represents a feature map, the solid lines an RoI (with 2×2 bins in this example), and the dots the 4 sampling points in each bin. RoIAlign computes the value of each sampling point by bilinear interpolation from the nearby grid points on the feature map. No quantization is performed on any coordinates involved in the RoI, its bins, or the sampling points.

Mask RCNN Framework

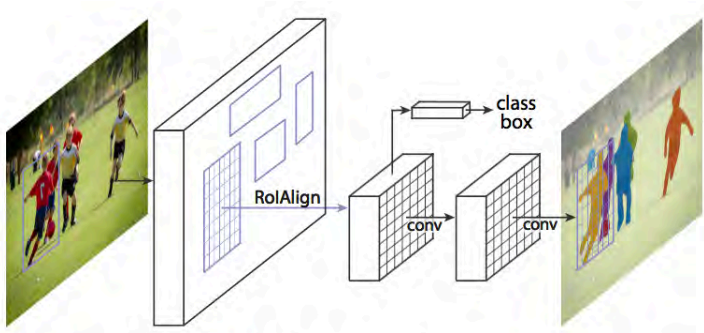


Figure 1. The **Mask R-CNN** framework for instance segmentation.

Mask RCNN on COCO

Mask RCNN is state-of-the-art in segmentation on COCO dataset

Figure 5. More results of **Mask R-CNN** on COCO test images, using ResNet-101-FPN and running at 5 fps, with 35.7 mask AP (Table 1).

Mask RCNN Segmentation Scores

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
MNC [10]	ResNet-101-C4	24.6	44.3	24.8	4.7	25.9	43.6
FCIS [26] +OHEM	ResNet-101-C5-dilated	29.2	49.5	-	7.1	31.3	50.0
FCIS+++ [26] +OHEM	ResNet-101-C5-dilated	33.6	54.5	-	-	-	-
Mask R-CNN	ResNet-101-C4	33.1	54.9	34.8	12.1	35.6	51.1
Mask R-CNN	ResNet-101-FPN	35.7	58.0	37.8	15.5	38.1	52.4
Mask R-CNN	ResNeXt-101-FPN	**37.1**	**60.0**	**39.4**	**16.9**	**39.9**	**53.5**

Table 1. **Instance segmentation** *mask* AP on COCO `test-dev`. MNC [10] and FCIS [26] are the winners of the COCO 2015 and 2016 segmentation challenges, respectively. Without bells and whistles, Mask R-CNN outperforms the more complex FCIS+++, which includes multi-scale train/test, horizontal flip test, and OHEM [38]. All entries are *single-model* results.

Mask RCNN Detection Scores

Mask RCNN is state-of-the-art in object detection on COCO dataset

	backbone	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{bb}_{S}	AP^{bb}_{M}	AP^{bb}_{L}
Faster R-CNN+++ [19]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [27]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [21]	Inception-ResNet-v2 [41]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [39]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
Faster R-CNN, RoIAlign	ResNet-101-FPN	37.3	59.6	40.3	19.8	40.2	48.8
Mask R-CNN	ResNet-101-FPN	38.2	60.3	41.7	20.1	41.1	50.2
Mask R-CNN	ResNeXt-101-FPN	**39.8**	**62.3**	**43.4**	**22.1**	**43.2**	51.2

Table 3. **Object detection** *single-model* results (bounding box AP), *vs.* state-of-the-art on `test-dev`. Mask R-CNN using ResNet-101-FPN outperforms the base variants of all previous state-of-the-art models (the mask output is ignored in these experiments). The gains of Mask R-CNN over [27] come from using RoIAlign (+1.1 AP^{bb}), multitask training (+0.9 AP^{bb}), and ResNeXt-101 (+1.6 AP^{bb}).

Note the effect of multitask training on the performance. Inclusion of segmentation task consistently improves the object detection scores

Key Point Detection using MASK RCNN

Mask RCNN is state-of-the-art in key-point detection on COCO dataset

Figure 7. Keypoint detection results on COCO `test` using Mask R-CNN (ResNet-50-FPN), with person segmentation masks predicted from the same model. This model has a keypoint AP of 63.1 and runs at 5 fps.

Mask RCNN Key Point Detection Scores

	AP^{kp}	AP^{kp}_{50}	AP^{kp}_{75}	AP^{kp}_{M}	AP^{kp}_{L}
CMU-Pose+++ [6]	61.8	84.9	67.5	57.1	68.2
G-RMI [32]†	62.4	84.0	68.5	**59.1**	68.1
Mask R-CNN, keypoint-only	62.7	87.0	68.4	57.4	71.1
Mask R-CNN, keypoint & mask	**63.1**	**87.3**	**68.7**	57.8	**71.4**

Table 4. **Keypoint detection** AP on COCO `test-dev`. Ours is a single model (ResNet-50-FPN) that runs at 5 fps. CMU-Pose+++ [6] is the 2016 competition winner that uses multi-scale testing, post-processing with CPM [44], and filtering with an object detector, adding a cumulative ~5 points (clarified in personal communication). †: G-RMI was trained on COCO *plus* MPII [1] (25k images), using two models (Inception-ResNet-v2 for bounding box detection and ResNet-101 for keypoints).

Note the effect of multitask training on the performance. Inclusion of segmentation task consistently improves the key point detection scores

Concluding Remarks

- Deep learning is the most successful of current machine learning technologies.
- It is in production use in several domains: face recognition, speech recognition, language translation, ...
- It is being used for scientific research: astronomy, genomics, neuroscience, high energy physics, ...
- It will spread to the network edge with increased computational power and communication speeds
- Enormous interest and investments from businesses and governments
- It is among the most important general purpose technologies currently under development

Ethical and Societal Issues

- Who owns the data?
- Privacy
- Security
- Human rights
- Bias and discrimination
- Jobs and work
- Power of platforms
- And many others ...

Contact Information for Further Information

- email: pramod.khargonekar@uci.edu; khargonekar@gmail.com
- Faculty website
- LinkedIn Page
- Google Scholar page