Making Deep Learning Practical

Dr. Fayyaz ul Amir Afsar Minhas
PIEAS Biomedical Informatics Research Lab
Department of Computer and Information Sciences
Pakistan Institute of Engineering & Applied Sciences
PO Nilore, Islamabad, Pakistan
http://faculty.pieas.edu.pk/fayyaz/

Making deeper neural networks practical

- Optimization
- Handling vanishing (or exploding) gradients
  - Pre-training (old!)
  - Drop-out
  - Batch Normalization
- Computational challenges
  - Use of computational graphs for automatic differentiation of neural networks
    - Allows for different types of architectures
  - Using GPU parallelization

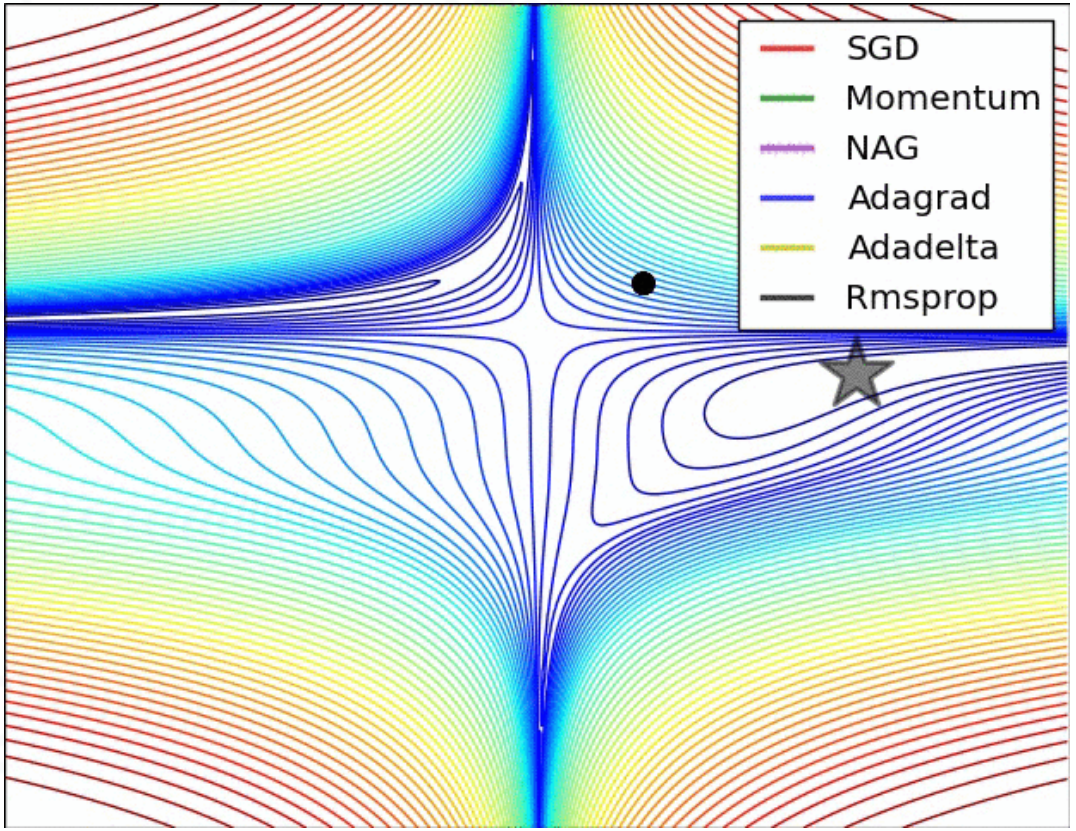
Optimization Methods

- Gradient Descent: Go down!
- Stochastic Gradient Descent
- Mini-batch Gradient Descent
- SGD with momentum: accelerate if going downhill for a long time
- Nesterov momentum: accelerate but not indefinitely
- Adagrad: Adaptive Learning Rate by accumulating past gradients
- AdaDelta/RMSProp: Adaptive Learning rate but does not accumulate all past gradients
- Adam: Adaptive learning rate with momentum

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$



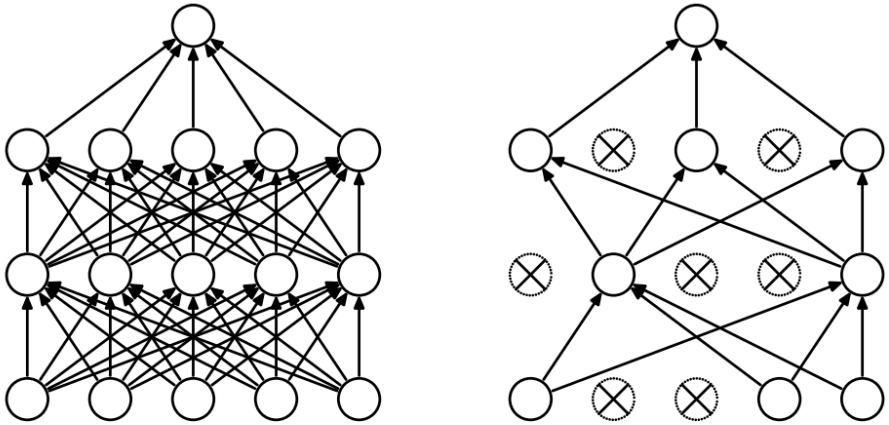
An overview of gradient descent optimization algorithms by Sebastian Ruder, 20-16
http://sebastianruder.com/optimizing-gradient-descent/ , https://arxiv.org/abs/1609.04747

Optimization Methods

- Sparse data
  - Adaptive Learning rate
- RMSProp, AdaDelta and Adam are very similar
  - Do well in general
  - Adam slightly outperforms RMSProp and is a good choice
- Parallelizing and distributing SGD
  - TensorFlow uses computational graph distribution
  - Other parallel schemes include: HogWild! Delayed SGD, etc.

Understanding Drop-out

- "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" by Srivastava et al., 2014.
  - Randomly drop units (along with their connections) from the neural network during training
  - Average weights across all "thinned" networks
  - Replaces explicit regularization and produces faster learning
  - Drop-out layer in keras!



(a) Standard Neural Net (b) After applying dropout.

Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Effect of Dropout

6.1.1 MNIST

Method	Unit Type	Architecture	Error %
Standard Neural Net (Simard et al., 2003)	Logistic	2 layers, 800 units	1.60
SVM Gaussian kernel	NA	NA	1.40
Dropout NN	Logistic	3 layers, 1024 units	1.35
Dropout NN	ReLU	3 layers, 1024 units	1.25
Dropout NN + max-norm constraint	ReLU	3 layers, 1024 units	1.06
Dropout NN + max-norm constraint	ReLU	3 layers, 2048 units	1.04
Dropout NN + max-norm constraint	ReLU	2 layers, 4096 units	1.01
Dropout NN + max-norm constraint	ReLU	2 layers, 8192 units	0.95
Dropout NN + max-norm constraint (Goodfellow et al., 2013)	Maxout	2 layers, (5 × 240) units	0.94
DBN + finetuning (Hinton and Salakhutdinov, 2006)	Logistic	500-500-2000	1.18
DBM + finetuning (Salakhutdinov and Hinton, 2009)	Logistic	500-500-2000	0.96
DBN + dropout finetuning	Logistic	500-500-2000	0.92
DBM + dropout finetuning	Logistic	500-500-2000	**0.79**

Understanding Batch-Normalization

- Re-normalization of weight parameters after every mini-batch to zero-norm and unit-variance with backpropagation
  - Note: This is not re-initialization to random values, rather the current weights are updated
- Reduces the effects of weight normalization and enforces regularization leading to faster learning
  - More effective than drop-out
  - No need for “pre-training”
- Available in Keras!

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv preprint arXiv:1502.03167v3*, 2015.

Computational Graphs

- Making a generic package for multi-layer neural networks requires
  - An abstract way of representing various computational operations involved in the network
  - Distributed Evaluations
  - Calculation of gradients
- Computational Graphs allow us to do all this!

Computational Graphs

- CGs are an easy way to think about mathematical expressions
- Formalizes the idea of neural networks and generalizes backpropagation and makes it computationally efficient
- The “compile” in keras builds a Computational Graph for the Network (can take time!)

Example

- Differentiation via the chain rule can be represented as the computational graph
- Symbolic derivatives
- Parallelization
  - Compute independent components in parallel
- Avoiding re-computation
  - Re-use symbolic derivatives
  - Store previous values

$$z = f\left(f\left(f(w)\right)\right) \qquad \frac{dz}{dw} = \frac{dz}{dy}\frac{dy}{dx}\frac{dx}{dw}$$

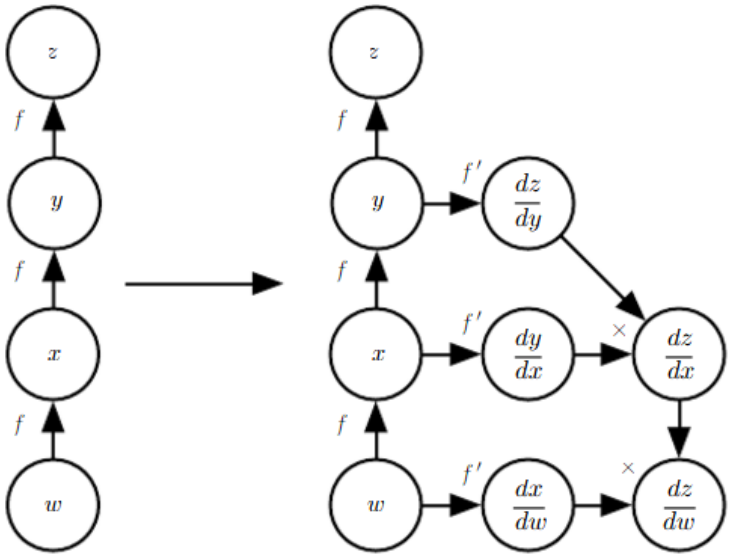
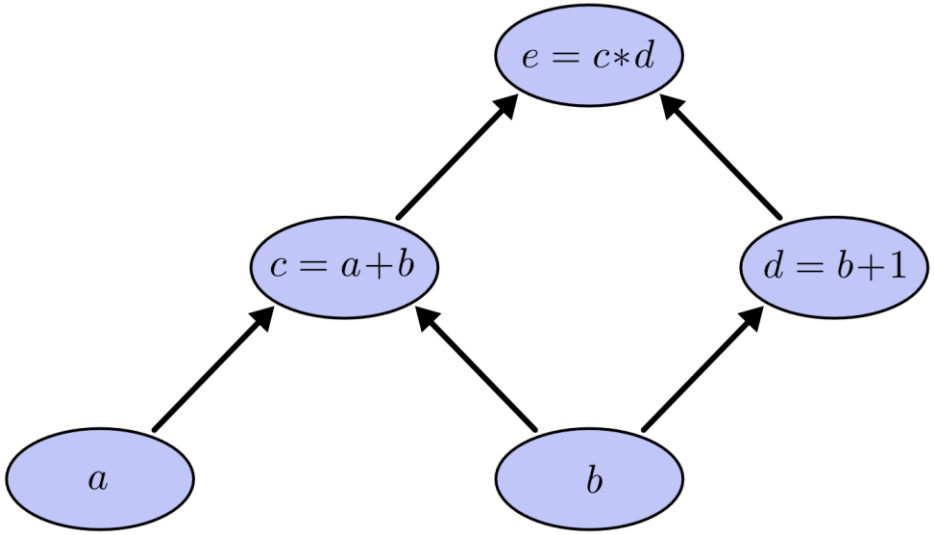
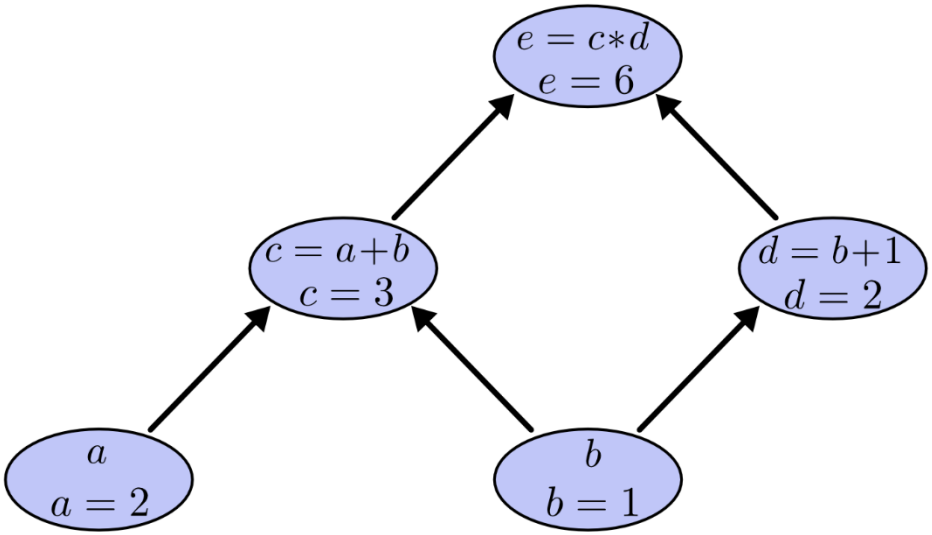
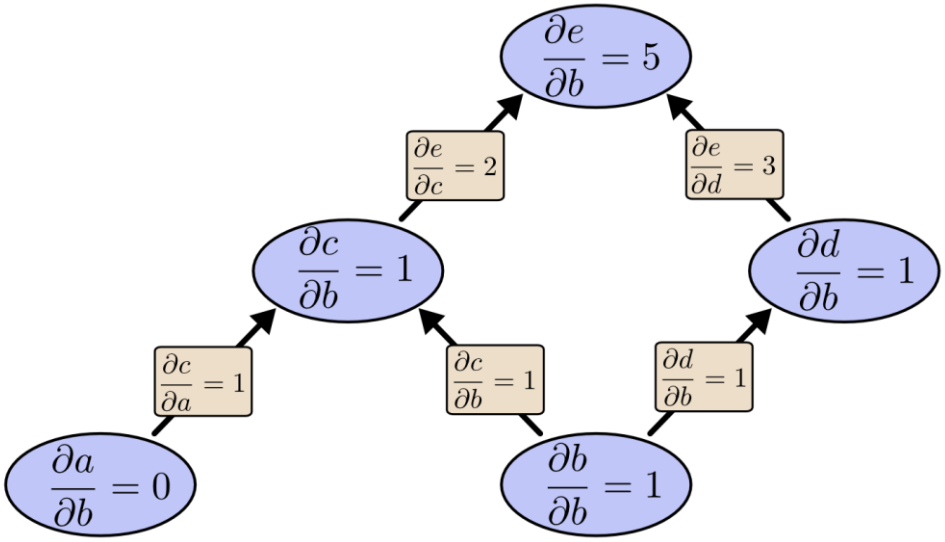


Figure 6.10: An example of the symbol-to-symbol approach to computing derivatives. In this approach, the back-propagation algorithm does not need to ever access any actual specific numeric values. Instead, it adds nodes to a computational graph describing how to compute these derivatives. A generic graph evaluation engine can later compute the derivatives for any specific numeric values. *(Left)*In this example, we begin with a graph representing $z = f(f(f(w)))$. *(Right)*We run the back-propagation algorithm, instructing it to construct the graph for the expression corresponding to $\frac{dz}{dw}$. In this example, we do not explain how the back-propagation algorithm works. The purpose is only to illustrate what the desired result is: a computational graph with a symbolic description of the derivative.

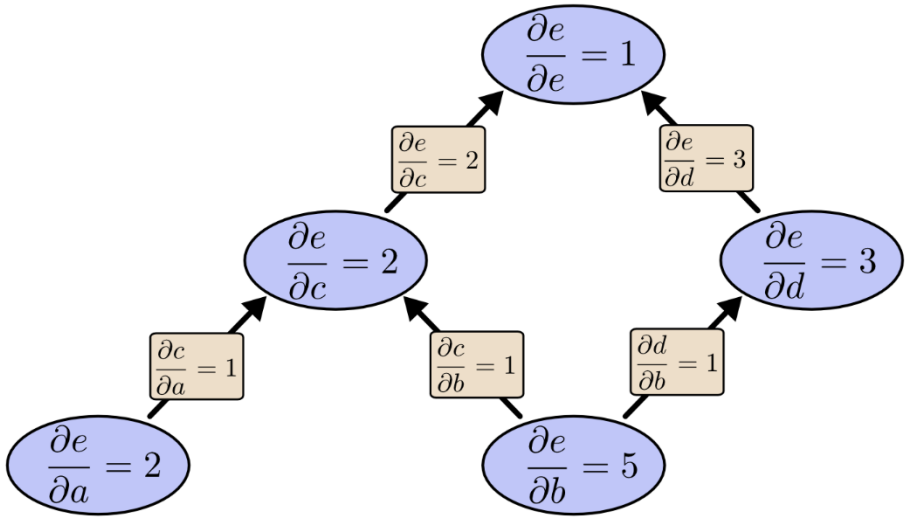
The magic of computational graphs







Forward-mode differentiation



reverse-mode differentiation

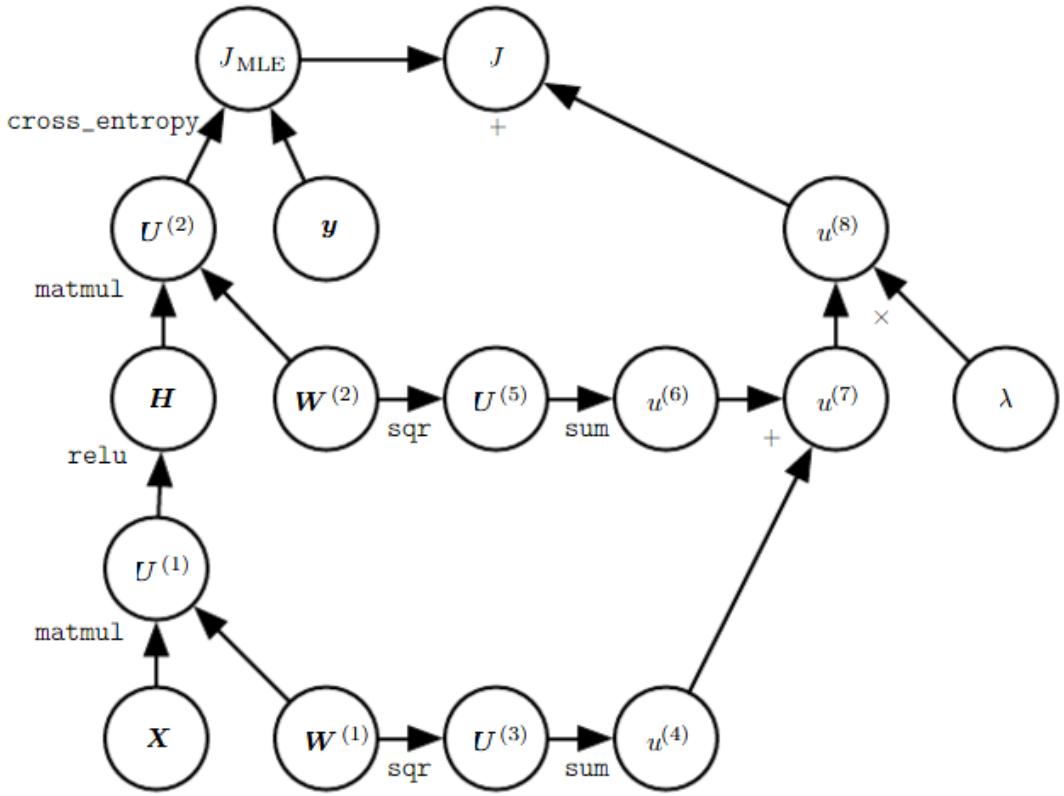
http://colah.github.io/posts/2015-08-Backprop/

The magic of computational graphs

- Forward-mode differentiation gave us the derivative of our output with respect to a single input, but reverse-mode differentiation gives us all of them.
- For this graph, that's only a factor of two speed up, but imagine a function with a million inputs and one output.
- Forward-mode differentiation would require us to go through the graph a million times to get the derivatives.
- Reverse-mode differentiation can get them all in one fell swoop!
- A speed up of a factor of a million is pretty nice!

Practical Example

$$J = J_{\text{MLE}} + \lambda \left(\sum_{i,j} \left(W^{(1)}_{i,j} \right)^2 + \sum_{i,j} \left(W^{(2)}_{i,j} \right)^2 \right)$$



Origins of Deep Learning

Year	Contributer	Contribution
300 BC	Aristotle	introduced Associationism, started the history of human's attempt to understand brain.
1873	Alexander Bain	introduced Neural Groupings as the earliest models of neural network, inspired Hebbian Learning Rule.
1943	McCulloch & Pitts	introduced MCP Model, which is considered as the ancestor of Artificial Neural Model.
1949	Donald Hebb	considered as the father of neural networks, introduced Hebbian Learning Rule, which lays the foundation of modern neural network.
1958	Frank Rosenblatt	introduced the first perceptron, which highly resembles modern perceptron.
1974	Paul Werbos	introduced Backpropagation
1980	Teuvo Kohonen	introduced Self Organizing Map
	Kunihiko Fukushima	introduced Neocogitron, which inspired Convolutional Neural Network
1982	John Hopfield	introduced Hopfield Network
1985	Hilton & Sejnowski	introduced Boltzmann Machine
1986	Paul Smolensky	introduced Harmonium, which is later known as Restricted Boltzmann Machine
	Michael I. Jordan	defined and introduced Recurrent Neural Network
1990	Yann LeCun	introduced LeNet, showed the possibility of deep neural networks in practice
1997	Schuster & Paliwal	introduced Bidirectional Recurrent Neural Network
	Hochreiter & Schmidhuber	introduced LSTM, solved the problem of vanishing gradient in recurrent neural networks

Origins of Deep Learning

2006	Geoffrey Hinton	introduced Deep Belief Networks, also introduced layer-wise pretraining technique, opened current deep learning era.
2009	Salakhutdinov & Hinton	introduced Deep Boltzmann Machines
2012	Geoffrey Hinton	introduced Dropout, an efficient way of training neural networks
2013	Kingma & Welling	introduced Variational Autoencoder (VAE), which may bridge the field of deep learning and the field of Bayesian probabilistic graphic models.
2014	Ian J. Goodfellow	introduced Generative Adversarial Network.
2015	Ioffe & Szegedy	introduced Batch Normalization

- Wang, Haohan, and Bhiksha Raj. "On the Origin of Deep Learning." *arXiv:1702.07800 [Cs, Stat]*, February 24, 2017. http://arxiv.org/abs/1702.07800.

Modern Practices

- Deep Convolutional Neural Networks
- Residual Networks
- Generative Models
  - Auto-encoders: VAE, NAE
  - Generative Adversarial Networks
  - Recurrent Neural Networks
- Recurrent Models
  - RNN
  - LSTM
- Transfer Learning
- Zero and One-shot learning

End of Lecture-1

We want to make a machine that will be proud of us.

- Danny Hillis