

# Introduction to Deep Learning

An Overview of Principles, Challenges, Architectures, and Tools

Chandrabose Aravindan  
<AravindanC@ssn.edu.in>

Machine Learning Research Group  
SSN College of Engineering, Chennai

Presented at:  
Sri Venkateswara College of Engineering, Chennai



## 1 Introduction

# Outline

- 1 Introduction
- 2 Neural Networks



# Outline

- 1 Introduction
- 2 Neural Networks
- 3 Deep Neural Networks



- 1 Introduction
- 2 Neural Networks
- 3 Deep Neural Networks
- 4 Architectures for Deep Learning
  - Convolutional Neural Networks
  - Recurrent Neural Networks

- 1 Introduction
- 2 Neural Networks
- 3 Deep Neural Networks
- 4 Architectures for Deep Learning
  - Convolutional Neural Networks
  - Recurrent Neural Networks
- 5 Tools for Deep Learning

- 1 Introduction
- 2 Neural Networks
- 3 Deep Neural Networks
- 4 Architectures for Deep Learning
  - Convolutional Neural Networks
  - Recurrent Neural Networks
- 5 Tools for Deep Learning
- 6 Summary

- 1 Introduction
- 2 Neural Networks
- 3 Deep Neural Networks
- 4 Architectures for Deep Learning
  - Convolutional Neural Networks
  - Recurrent Neural Networks
- 5 Tools for Deep Learning
- 6 Summary



## Credits

This talk is based on published papers and materials available on the Internet (especially the figures and images!). Since I have a big list, I am giving a general acknowledgment. I have a not-so-comprehensive list of references at the end though.

# Intelligence and Learning

- “Artificial Intelligence” is a field of study in Computer Science that strives to make computers perform tasks that require “intelligence” (whatever that means!)

# Intelligence and Learning

- “Artificial Intelligence” is a field of study in Computer Science that strives to make computers perform tasks that require “intelligence” (whatever that means!)
- Consider for example, classifying an email as spam or not

# Intelligence and Learning

- “Artificial Intelligence” is a field of study in Computer Science that strives to make computers perform tasks that require “intelligence” (whatever that means!)
- Consider for example, classifying an email as spam or not
- Given that a buyer is buying a book at online store, suggest some related products for that buyer
- Given an ultrasound image of abdomen scan of a pregnant lady, predict the weight of the baby
- Given a sentence in English, generate the translated sentence in Tamil
- Given a video clip, generate a description of the action performed

# Intelligence and Learning

- “Artificial Intelligence” is a field of study in Computer Science that strives to make computers perform tasks that require “intelligence” (whatever that means!)
- Consider for example, classifying an email as spam or not
- Given that a buyer is buying a book at online store, suggest some related products for that buyer
- Given an ultrasound image of abdomen scan of a pregnant lady, predict the weight of the baby
- Given a sentence in English, generate the translated sentence in Tamil
- Given a video clip, generate a description of the action performed
- And the list can go on and on and on . . .

# Intelligence and Learning

- “Artificial Intelligence” is a field of study in Computer Science that strives to make computers perform tasks that require “intelligence” (whatever that means!)
- Consider for example, classifying an email as spam or not
- Given that a buyer is buying a book at online store, suggest some related products for that buyer
- Given an ultrasound image of abdomen scan of a pregnant lady, predict the weight of the baby
- Given a sentence in English, generate the translated sentence in Tamil
- Given a video clip, generate a description of the action performed
- And the list can go on and on and on . . .
- Is there any commonality to such diverse applications?
- Are there any well understood concepts and algorithms that can be used to build such applications?



# Intelligent Tasks

- These are some examples of “Intelligent tasks” — tasks that are “easy” for humans but “extremely difficult” to build a software
- Artificial Intelligence is about building systems that can efficiently perform such “intelligent tasks”
- One of the important aspects that enable humans to perform such Intelligent tasks is their ability to [learn from experiences](#) (either [supervised](#) or [unsupervised](#))
- Is it possible for us to build systems that learn from data?

# Machine Learning

- Traditionally, we know the functions that we wish to compute and we discuss about algorithms and complexities
- But, what if we need to compute a function without knowing the function?!



# Machine Learning

- Traditionally, we know the functions that we wish to compute and we discuss about algorithms and complexities
- But, what if we need to compute a function without knowing the function?!
- For example, what is the function that maps an email to a spam? Audio waves to emotions? Item that a customer is buying now to what other products she might be interested in? Marks to the grades? (lack of omniscience)

# Machine Learning

- Traditionally, we know the functions that we wish to compute and we discuss about algorithms and complexities
- But, what if we need to compute a function without knowing the function?!
- For example, what is the function that maps an email to a spam? Audio waves to emotions? Item that a customer is buying now to what other products she might be interested in? Marks to the grades? (lack of omniscience)
- But we do have examples of such functional mappings!
- Is it possible to learn the function from the examples?

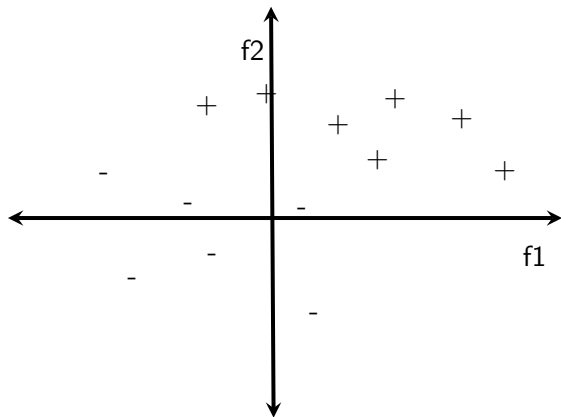
# Machine Learning

- Traditionally, we know the functions that we wish to compute and we discuss about algorithms and complexities
- But, what if we need to compute a function without knowing the function?!
- For example, what is the function that maps an email to a spam? Audio waves to emotions? Item that a customer is buying now to what other products she might be interested in? Marks to the grades? (lack of omniscience)
- But we do have examples of such functional mappings!
- Is it possible to learn the function from the examples?
- It may not be possible for us to find the exact function underlying the examples, but a close enough “hypothesis” could be learned from the examples

# What could “examples” and “hypothesis” mean?

- Consider a task where we need to decide whether an instance belongs to a class or not — for example, decide if skin cancer from a given dermoscopic image
- This is a **binary classification** task
- Assume that we have identified the features (that can be represented by real numbers) and have collected positive and negative instances — for skin cancer identification, we may simply take moments
- Each instance (dermoscopic image) is a vector in the feature space
- Machine learning problem here is to find a geometric model using which we can predict the class of new instances
- In this talk, we focus on building discriminating neural network models from instances (inductive learning) using supervised learning algorithms

# Binary Classification



# Traditional Machine Learning

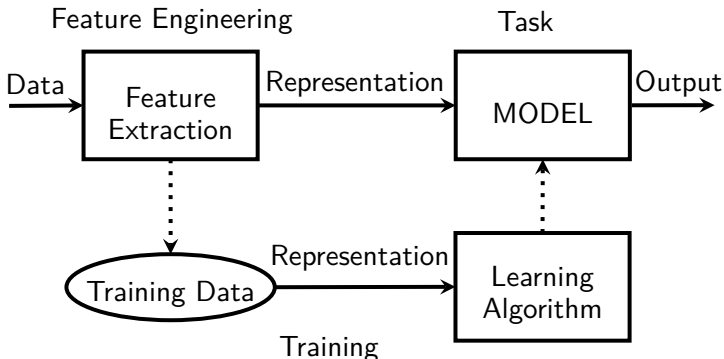


Figure: Traditional Machine Learning

- Inductive Learning
  - Learning from examples
  - Supervised — labelled data
  - Unsupervised — unlabelled data
  - Semi-supervised
- Reinforcement Learning
  - Learning from rewards / punishments
  - Learning Policies
- Explanation-based Learning
  - Background theory and a few examples are given
  - Learn by trying to explain the examples using the theory
  - Theory is refined

# Types of Models

- Geometric Models
  - Neural Networks
  - Support Vector Machines
  - Distance-based Models
- Probabilistic Models
  - Naive Bayes
  - Belief Networks
  - Hidden Markov Models
  - Conditional Random Fields
  - Stochastic Neural Networks
- Logical Models
  - Decision Trees
  - Random Forest
  - Association Rules



- Binary Classification
- Multi-class Classification
- Multi-label Classification
- Regression
- Ranking
- Structure Prediction
- Sequence Generation
- Image/Video/Speech/Text processing and analysis

- 1 Introduction
- 2 Neural Networks**
- 3 Deep Neural Networks
- 4 Architectures for Deep Learning
  - Convolutional Neural Networks
  - Recurrent Neural Networks
- 5 Tools for Deep Learning
- 6 Summary

- Artificial Neural Network (ANN) is a computational model that is capable of representing any continuous or discrete function
- Interesting point about ANN is that a model can be “learnt” from input-output example pairs
- Extremely useful when the underlying functional mapping is not clear and all we have is set of examples
- Neural networks have gone through waves of development (Perceptron, Back-Propagation, SVM) and this talk presents an overview of its latest wave — Deep Learning

# Model of a Neuron

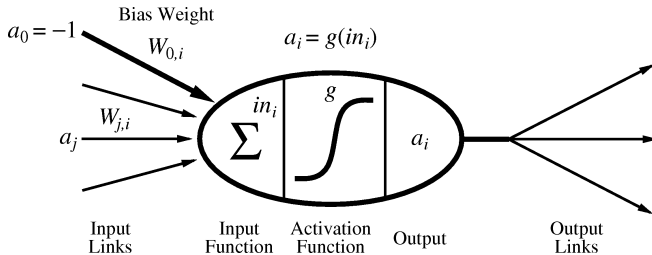


Figure: Model of a Neuron (Adopted from [Russel and Norvig, 2009])

## What does a neuron do?

A neuron is a computational unit that performs weighted sum of its inputs and applies a non-linear activation to generate its output

# Activation Functions

- Binary and Bipolar step functions

$$f(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

$$f(x) = \begin{cases} 1 & x > 0 \\ -1 & x \leq 0 \end{cases}$$

- Binary and bipolar sigmoid functions

$$s(x) = \frac{1}{1 + e^{-\sigma x}}$$

$$S(x) = 2s(x) - 1 = \frac{1 - e^{-\sigma x}}{1 + e^{-\sigma x}}$$

- Hyperbolic tan (tanh)

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

# Activation Functions

- Feature values are scaled either in the range of (0,1) or (-1,1)
- Suitable for both Regression and Classification tasks

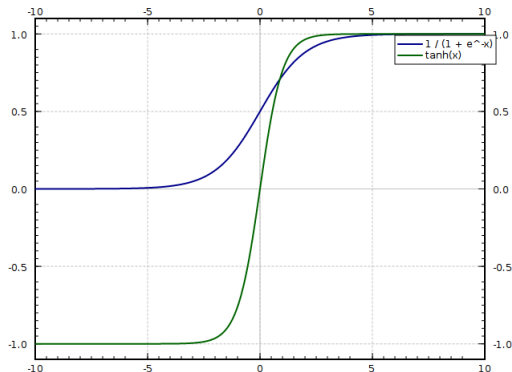


Figure: Sigmoid and tanh activation functions

# Activation Functions

- Desirable properties of an activation function include that it is differentiable and the differentiation is expressible in terms of the function itself

- Binary sigmoid:

$$s'(x) = \sigma s(x) [1 - s(x)]$$

- Bipolar sigmoid:

$$S'(x) = \frac{\sigma}{2} [1 + S(x)] [1 - S(x)]$$

- Hyperbolic tan (tanh)

$$h'(x) = [1 + h(x)] [1 - h(x)] = [1 - h^2(x)]$$

- Feed Forward Networks
  - Perceptron
  - Multi-Layer Perceptron
- Recurrent Networks
  - Hopfield Network
  - Echo State Network
  - Long Short-Term Memory Network (LSTM)
- Stochastic Networks
  - Boltzmann Machines



# Feed Forward Neural Network

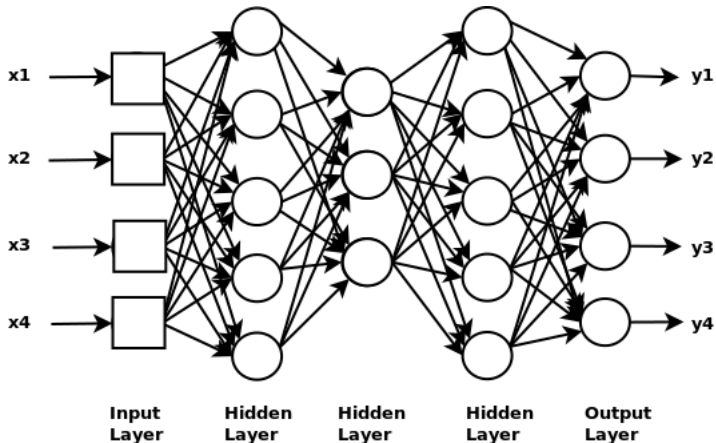


Figure: Feed Forward Network

# Recurrent Neural Network

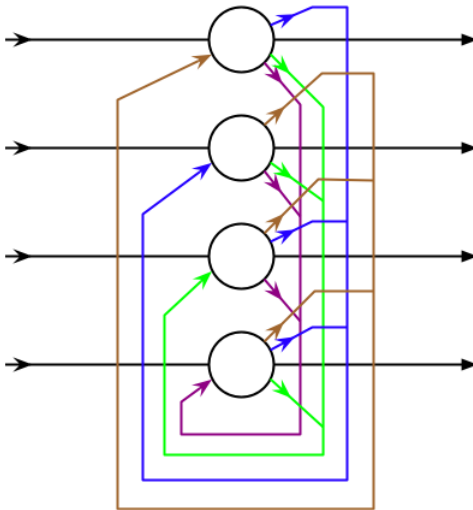


Figure: Hopfield Network (Figure adopted from Wikipedia)

# Single neuron Classification

- Let us consider a neuron with a step function for activation — output is “positive” when weighted sum of input is greater than 0 — output is “negative” when weighted sum of input is less than 0

# Single neuron Classification

- Let us consider a neuron with a step function for activation — output is “positive” when weighted sum of input is greater than 0 — output is “negative” when weighted sum of input is less than 0
- What is the boundary separating the two classes?

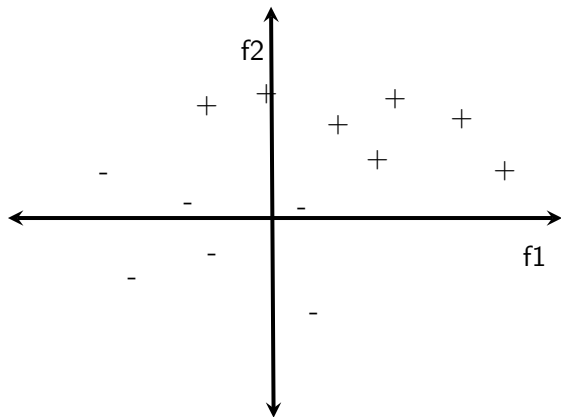
# Single neuron Classification

- Let us consider a neuron with a step function for activation — output is “positive” when weighted sum of input is greater than 0 — output is “negative” when weighted sum of input is less than 0
- What is the boundary separating the two classes?

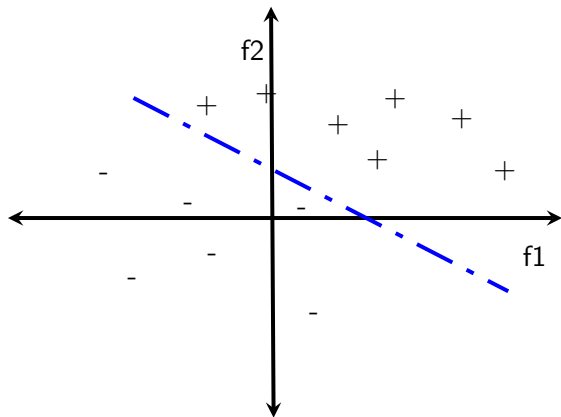
$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = 0$$

- In case of two dimensions this is a line; for three dimensions this is a plane; and in general it is a hyperplane
- Thus, we are looking for a geometric model (hyperplane) defined by weights as model parameters

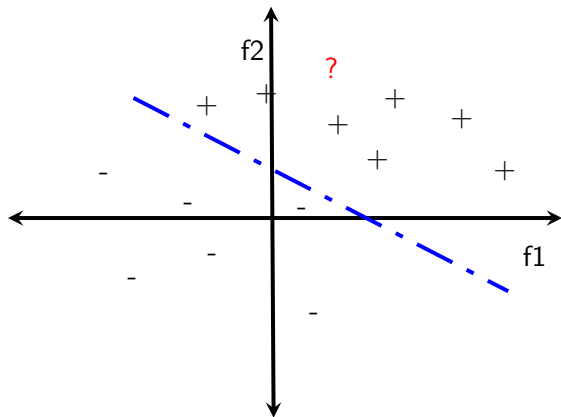
# Single Neuron — Linear Separation



# Single Neuron — Linear Separation

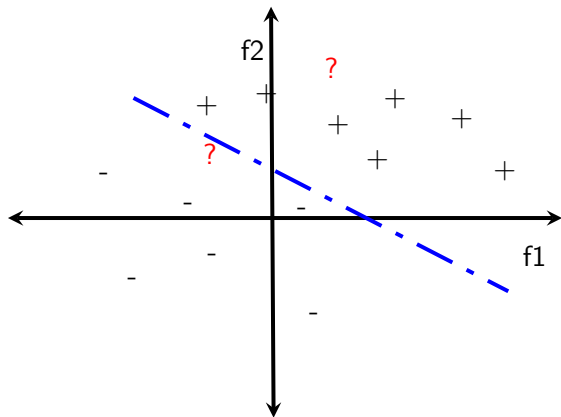


# Single Neuron — Linear Separation

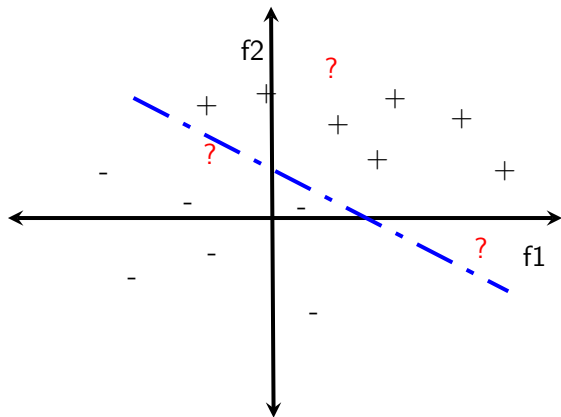




# Single Neuron — Linear Separation



# Single Neuron — Linear Separation



# Feed-Forward Neural Network

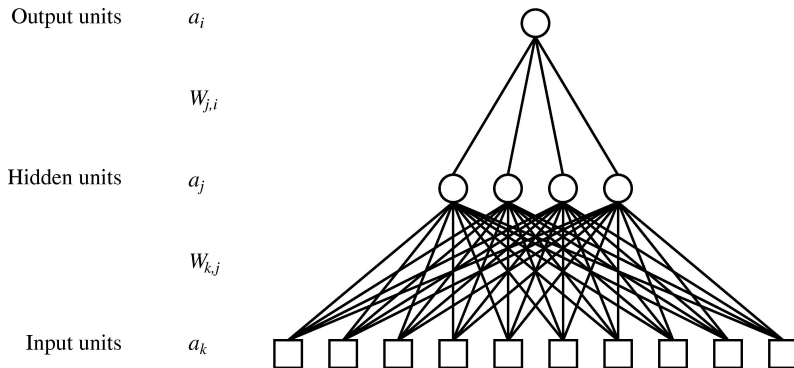
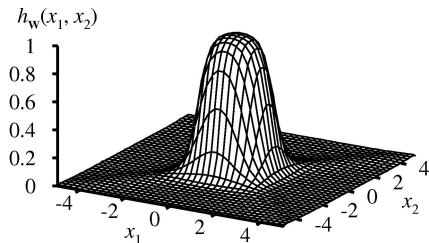
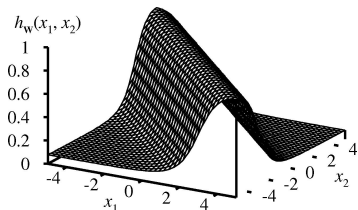


Figure: Multi-Layer Perceptron (Adopted from [Russel and Norvig, 2009])

# Non-linear separation



- Any continuous function can be represented with two layers and any function with three layers [Hornik et al., 1989]
- Combine two opposite facing threshold functions to make a ridge
- Combine two perpendicular ridges to make a bump
- Add bumps of various sizes and locations to fit any surface

- Machine learning problem is to find a model (described using some model parameters) from the given examples, to perform this classification task

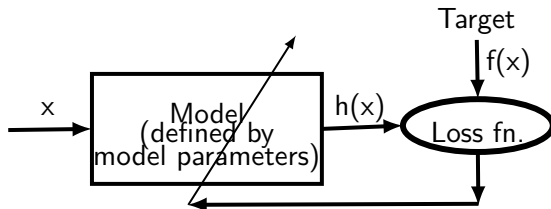
# Learning the weights

- Machine learning problem is to find a model (described using some model parameters) from the given examples, to perform this classification task
- In this case, the model parameters are the weights and thus we need to **learn the weights** (including bias) from the examples

# Learning the weights

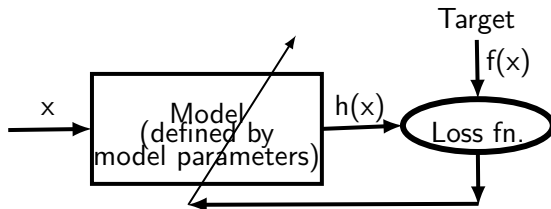
- Machine learning problem is to find a model (described using some model parameters) from the given examples, to perform this classification task
- In this case, the model parameters are the weights and thus we need to **learn the weights** (including bias) from the examples
- **gradient descent** based **optimization algorithms** may be used to minimize a **loss function** such as mean squared error

# Supervised Learning Feedback Loop





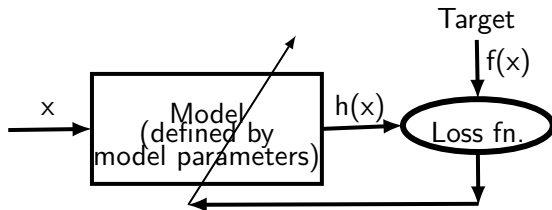
# Supervised Learning Feedback Loop



## Major Issue

Will this feedback loop converge?

# Supervised Learning Feedback Loop



## Major Issue

Will this feedback loop converge?

## Major Issue

Will the model generalize beyond the training samples?

# Loss function is available only for the last layer?

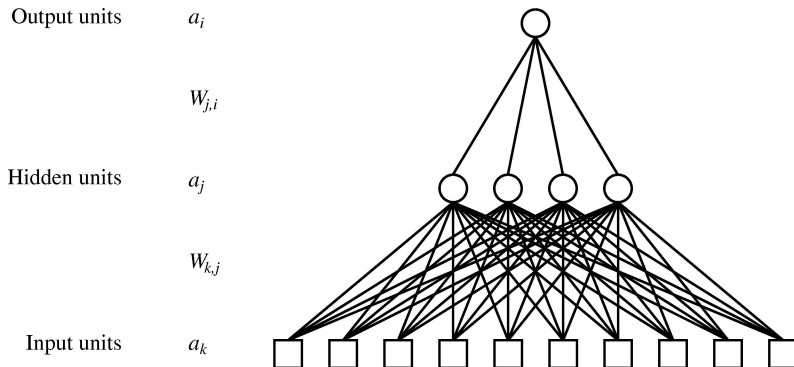


Figure: Multi-Layer Perceptron (Adopted from [Russel and Norvig, 2009])

# Back-propagation algorithm

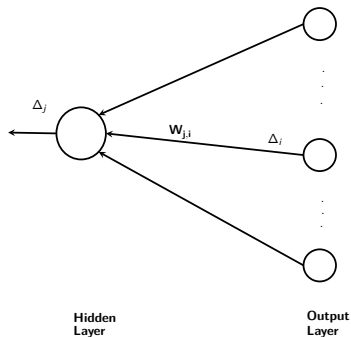
- Gradient based error minimization techniques do not work because we do not know the target for hidden units!

# Back-propagation algorithm

- Gradient based error minimization techniques do not work because we do not know the target for hidden units!
- Solution is to “back propagate” error from all the output units to set a target for a hidden unit [Rumelhart et al., 1986]

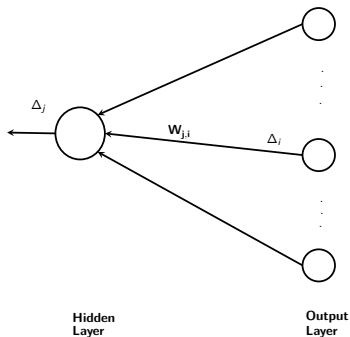
# Back-propagation algorithm

- Gradient based error minimization techniques do not work because we do not know the target for hidden units!
- Solution is to “back propagate” error from all the output units to set a target for a hidden unit [Rumelhart et al., 1986]



# Back-propagation algorithm

- Gradient based error minimization techniques do not work because we do not know the target for hidden units!
- Solution is to “back propagate” error from all the output units to set a target for a hidden unit [Rumelhart et al., 1986]



$$\Delta_j = g'(inp_j) \sum_i W_{j,i} \Delta_i, \text{ where } \Delta_i = Err_i \times g'(inp_i)$$

# Weight Update Rules

- Output Layer: Weight update is similar to that of a Perceptron

$$W_{j,i} \leftarrow W_{j,i} + \eta \times a_j \times \Delta_i$$

where

$$\Delta_i = Err_i \times g'(inp_i)$$



# Weight Update Rules

- Output Layer: Weight update is similar to that of a Perceptron

$$W_{j,i} \leftarrow W_{j,i} + \eta \times a_j \times \Delta_i$$

where

$$\Delta_i = Err_i \times g'(inp_i)$$

- Hidden Layer: Back-propagate error from output layer and use that for updating weights

$$\Delta_j = g'(inp_j) \sum_i W_{j,i} \Delta_i$$

$$W_{k,j} \leftarrow W_{k,j} + \eta \times a_k \times \Delta_j$$

# Back-propagation algorithm

- Initialize the weights — many methods have been proposed
- Perform the following iteration for each example:
  - Feed forward: Compute the output of network using current weights
  - Back-propagation: Compute and propagate the error back (starting from the output layer)
  - Weight-updation: Update all the weights using the weight update rules ([Incremental Learning](#))
  - Weight-updation: Add up the weight updates for all the examples and apply at the end of an iteration ([Batch Learning](#))
- Continue to perform these iterations until loss falls below a pre-determined value

# Issues in Back-propagation algorithm

- Back-propagation algorithm has generated a lot of interest and several variations have been proposed.

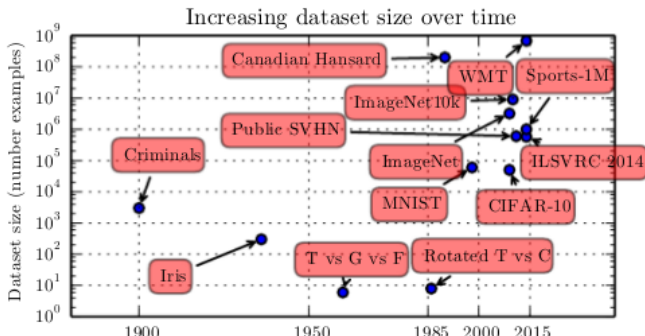
# Issues in Back-propagation algorithm

- Back-propagation algorithm has generated a lot of interest and several variations have been proposed.
- However, there are several issues in employing this algorithm
  - It has been shown that finding weights to minimize error is NP-complete [Blum and Rivest, 1989]
  - Algorithm is not guaranteed to converge
  - Over-fitting to the training samples
  - Error minimization process can get trapped in a local minima
  - Extremely slow convergence
  - Selection of initial weights may be critical
  - Selection of optimal parameters
  - Extremely difficult to explain the model and its predictions

- 1 Introduction
- 2 Neural Networks
- 3 Deep Neural Networks**
- 4 Architectures for Deep Learning
  - Convolutional Neural Networks
  - Recurrent Neural Networks
- 5 Tools for Deep Learning
- 6 Summary

# Large datasets demand higher model capacity

- Today, we have very large datasets available — especially for images, videos, speech, and text
- However, traditional feature engineering and classifiers based on given-representations have saturated — “shallow” networks have relatively small model parameters to capture information content of large data sets



# Feature Engineering

- For the past couple of decades, major focus of machine learning has been selecting and extracting appropriate features
- Images: color, texture, edges, connected components, shapes, moments, SIFT, SURF, HOG, Wavelets-based etc.
- Text: n-grams, character n-grams, POS tags, Named Entities, tags from shallow parsing, word sense disambiguation, etc.
- Speech: Mel Cepstral coefficients (MFCC), energy, zero crossing rates, pitch, timbre, etc.

# Feature Engineering

- For the past couple of decades, major focus of machine learning has been selecting and extracting appropriate features
- Images: color, texture, edges, connected components, shapes, moments, SIFT, SURF, HOG, Wavelets-based etc.
- Text: n-grams, character n-grams, POS tags, Named Entities, tags from shallow parsing, word sense disambiguation, etc.
- Speech: Mel Cepstral coefficients (MFCC), energy, zero crossing rates, pitch, timbre, etc.
- Which features are suitable for a given task? — leads to feature engineering
- Which classifier is better? Ensemble of classifiers?
- Is a machine learning algorithm really “intelligent”?



# Scope of Learning

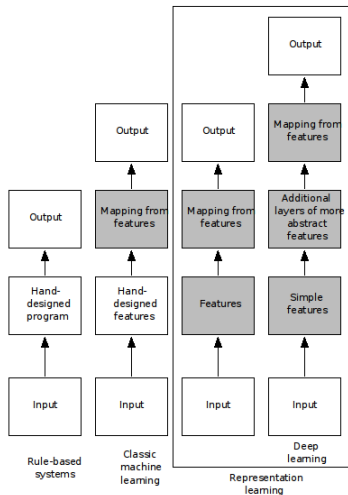


Figure: Scope of learning [Goodfellow et al., 2016]

# Depth enables simple representations

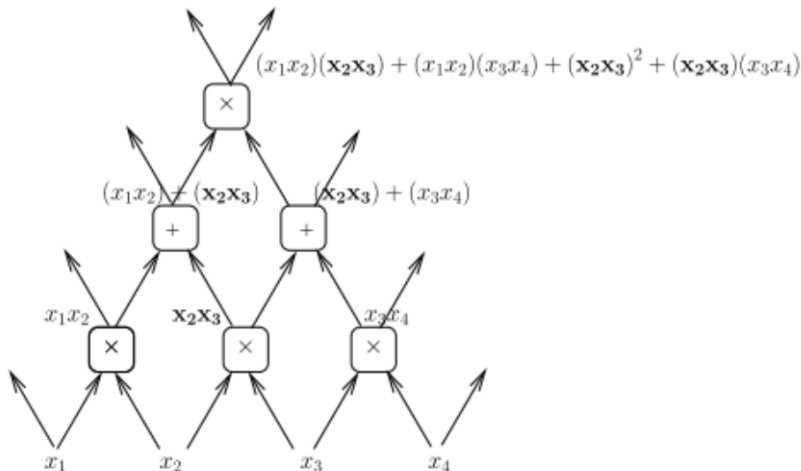


Figure: Polynomial Circuit [Goodfellow et al., 2016]

# Deep Representation Learning

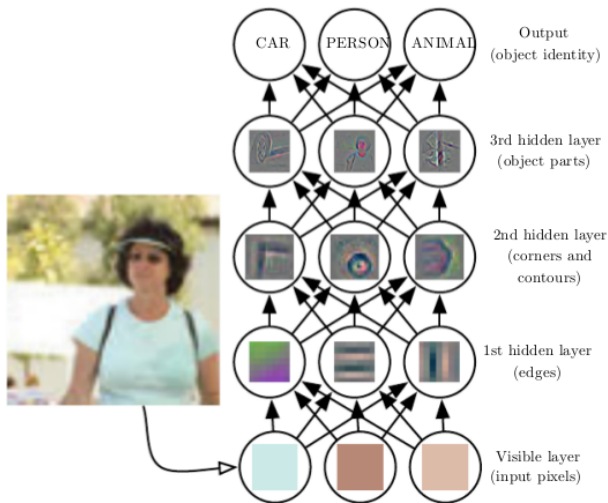


Figure: Deep Representation Learning [Goodfellow et al., 2016]

# Challenges in increasing the layers

- Diminishing/exploding Gradients
- Extremely slow convergence
- Lack of generalization
- Lack of huge data — number of model parameters increases with depth
- Lack of labeled data — supervised learning requires examples with corresponding targets
- Computational inefficiency

# New Deep Learning Ideas

- New Activation Functions
- Initialization Methods
- Optimization Techniques
- Regularization Methods
- Data Augmentation
- GPU-based and distributed algorithms

# Activation Function: Rectifier

- Rectifier is a simple activation function (see for example [Jarrett et al., 2009, Nair and Hinton, 2010]) defined as

$$f(x) = \max(0, x)$$

- Neuron using rectifier is popularly known as ReLU (Rectified Linear Unit)
- One variation of this is “softplus” function

$$f(x) = \ln(1 + e^x)$$

$$f'(x) = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}}$$

# Rectifier and Softplus Functions

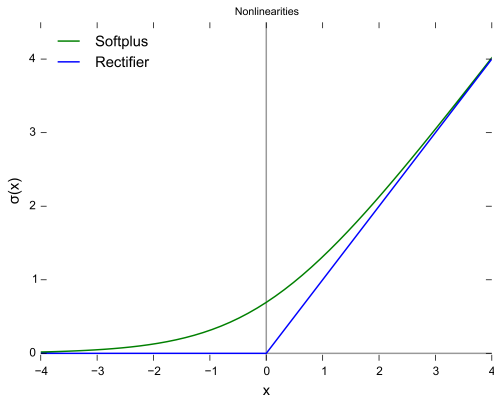


Figure: Rectifier and Softplus (Figure adopted from Wikipedia)

# ReLU is faster!

- It has been shown that ReLU converges faster than networks using Traditional 'tanh' [Krizhevsky et al., 2012]

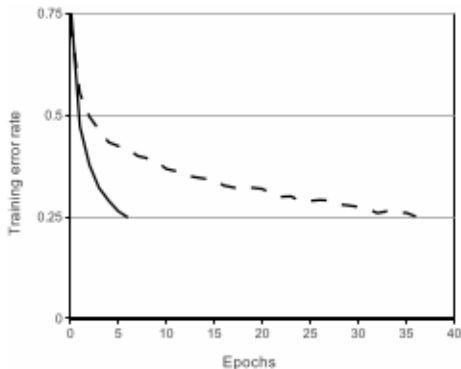


Figure: ReLU is faster than tanh



# Activation Function: Variations of Rectifier

- One major problem with rectifiers: Several neurons may “die” — never get activated with any of the training data

# Activation Function: Variations of Rectifier

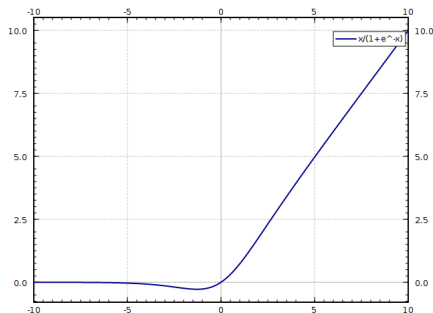
- One major problem with rectifiers: Several neurons may “die” — never get activated with any of the training data
- One variation of rectifier to avoid this is **Leaky ReLU**:  $f(x) = \alpha x$  when  $x < 0$  and  $f(x) = x$  otherwise. Here  $\alpha$  is a small constant
- **Parameterised ReLU (PReLU)**: Instead of a hyper-parameter  $\alpha$ , each neuron may have its own parameter  $a_i$ , where  $f(x_i) = a_i x_i$  when  $x_i < 0$
- **Randomized Leaky ReLU**: A random Gaussian noise may be added to the output when  $x < 0$
- Recently, a **maxout activation function** has been proposed which generalizes all the above ideas:  $f(x) = \max(a_1 x_1 + b_1, a_2 x_2 + b_2)$

# Activation Function: Swish

- “Swish” is similar to ReLU, but smooth and non-monotonic

$$s(x) = x \cdot \sigma(x) = \frac{x}{1 + e^{-x}}$$

- Recent studies have shown that Swish performs better than ReLU on many problems [Ramachandran et al., 2017]
- Yet to be studied in depth



# Activation Function: Softmax

- Squashes a  $k$ -dimension vector of real values to  $k$ -dimension vector of real values in the range  $(0,1)$  that add up to 1
- Converts a  $k$ -dimension vector to probability distribution over  $K$  different possible outcomes
- Useful in multi-class classification
- It is a winner-takes-all strategy where the maximum value in the original vector gets an exponentially high probability
- For example, softmax of  $[1,2,3,4,1,2,3]$  is  $[0.024, 0.064, 0.175, 0.475, 0.024, 0.064, 0.175]$

$$SM(X)_i = \frac{e^{x_i}}{\sum_{j=1}^k e^{x_j}} \quad \forall i : 1..k$$

# Regularization Methods: Dropout

- One of the important concepts that has evolved recently is that of “Dropout” (see for example [Srivastava et al., 2014])
- Set the output of a hidden neuron to 0 with a probability of 0.5 (effectively “drop” those neurons out)
- Dropped out neurons do not take part in forward pass and back-propagation
- Hence, different architecture is sampled every time, but these networks share weights
- This technique reduces the complex co-adaption of neurons and reduces the model parameters to be learned
- However, dropout almost doubles the number of iterations required

# Data Augmentation

- We may not have enough data for learning a large number of model parameters (especially labeled data)
- One way out is to augment the existing data with some “label-preserving” operations
- For example, to augment labeled image dataset, we may do the following:
  - Image translations and horizontal reflections. Given, 256x256 images, extract random 224x224 patches and their horizontal reflections — object labels do not change!
  - Alter the intensities of RGB channels — one idea may be to perform PCA to find principal components and add a magnitude of this to the intensities:  $I_{xy} = I_{xy} + [p_1, p_2, p_3][\alpha_1 \lambda_1 \alpha_2 \lambda_2 \alpha_3 \lambda_3]^T$

# Optimization Techniques

- Standard Gradient Descent is basically a batch update mechanism where gradient is calculated using all the training instances
- However, loading a large set of examples to memory at once may not be possible — intractable
- Extremely slow convergence
- Can not be done in “online” mode

# Stochastic Gradient Descent

- Stochastic Gradient Descent (SGD) incrementally updates the weights for each training example
- This may result in wide swings the weights (which may be good sometimes!)
- To overcome this problem, learning rate is slowly decreased as the learning progresses
- In other words, fluctuations are permitted initially, but controlled with the progress of the iterations — simulated annealing idea
- It is possible to take best of both worlds and perform mini-batch updates — gradients are calculated for a batch of  $n$  training examples
- Mini-Batch updates provide better exploitation of GPU and distributed computing



# Challenges in Mini-Batch Stochastic Gradient Descent

- How to choose appropriate learning rate? A high value may lead to undesirable fluctuations
- What may a better schedule for reducing the learning rate? Extremely slow schedule may lead us to a better minima, but then convergence will also be extremely slow
- Should all the parameters have the same learning rate?

# Emerging Optimization Ideas

- Momentum: add a fraction of previous weight vector while updating the weights
- Nesterov Accelerated Gradient (NAG)
- Adagrad — adaptive learning rates
- Adadelta
- Adaptive Moment Estimation (Adam)
- Adamax
- Nesterov-accelerated Adaptive Moment Estimation (Nadam)
- Parallelizing and distributing optimization

- Computational inefficiency can be addressed through effective parallelism using GPUs
- Libraries such as CUDA and cuDNN are available
- Tensor libraries, such as TensorFlow and Torch, make effective use of GPUs
- CNN architecture can “branch-off” and “merge” to make use of multiple GPU cards in the system (see for example, AlexNet [Krizhevsky et al., 2012])

- 1 Introduction
- 2 Neural Networks
- 3 Deep Neural Networks
- 4 Architectures for Deep Learning**
  - Convolutional Neural Networks
  - Recurrent Neural Networks
- 5 Tools for Deep Learning
- 6 Summary

- 1 Introduction
- 2 Neural Networks
- 3 Deep Neural Networks
- 4 Architectures for Deep Learning**
  - Convolutional Neural Networks
  - Recurrent Neural Networks
- 5 Tools for Deep Learning
- 6 Summary

# 1D Convolution

255.45	273.90	286.20	283.55	277.10	268.85	257.05
--------	--------	--------	--------	--------	--------	--------

0.20	0.30	0.50
------	------	------

276.36	282.42	280.86	274.27	264.60
--------	--------	--------	--------	--------

# 1D Convolution

255.45	273.90	286.20	283.55	277.10	268.85	257.05
--------	--------	--------	--------	--------	--------	--------

0.20	0.30	0.50
------	------	------

276.36	282.42	280.86	274.27	264.60
--------	--------	--------	--------	--------

- Kernel (also known as filter)
- Receptive Field
- Feature Map

# 2D Convolution

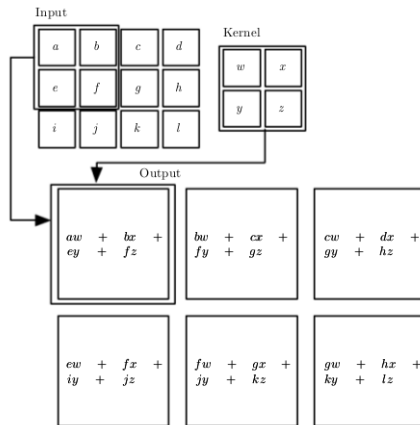


Figure: Adopted from [Goodfellow et al., 2016]



# 3D Convolution

- A 3D input volume ( $W \times H \times D$ ) may be convolved with a 3D kernel ( $F \times F \times D$ )
- Note that 3D convolution results in a 2D volume!

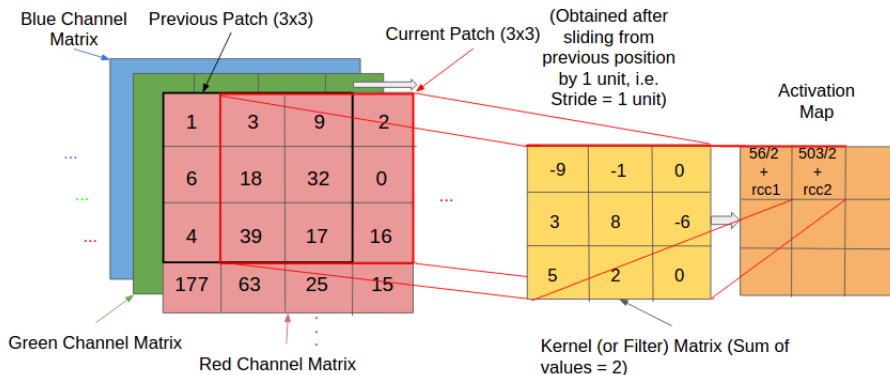
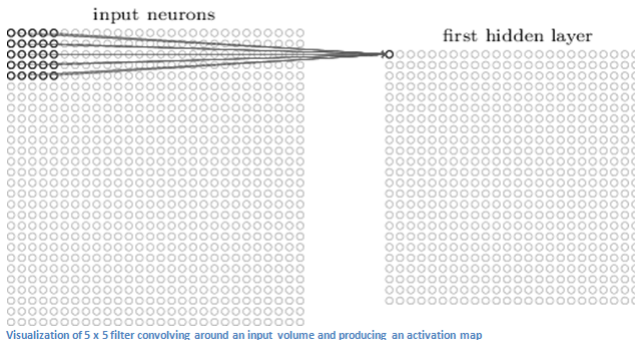


Figure: Adopted from ACM XRDS Blog

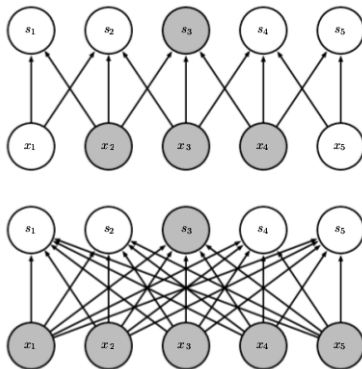
# Convolution Neural Network



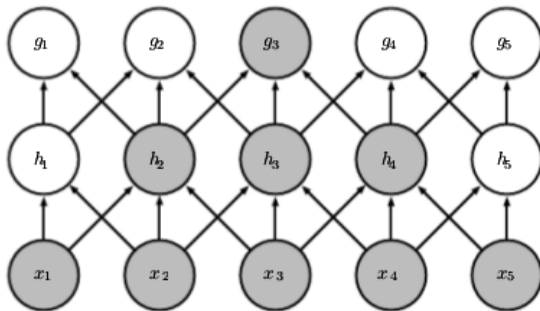
- 2-Dimensional Topological view of a layer of neurons
- Sparse connections — reduces model parameters and improves generalization

# Sparse Connections?

- Greatly reduces the number of model parameters — and thereby helps to increase generalization
- However, only “local” information is captured



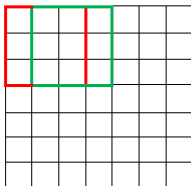
# Sparse Connections?



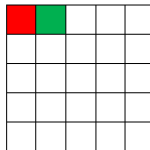
**Figure:** Having sparse connections may not be a drawback  
[Goodfellow et al., 2016]

# Convolution Stride

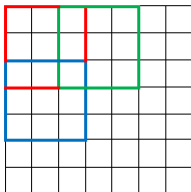
7 x 7 Input Volume



5 x 5 Output Volume



7 x 7 Input Volume



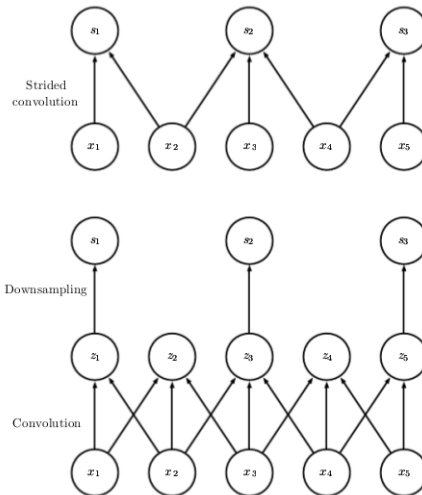
3 x 3 Output Volume



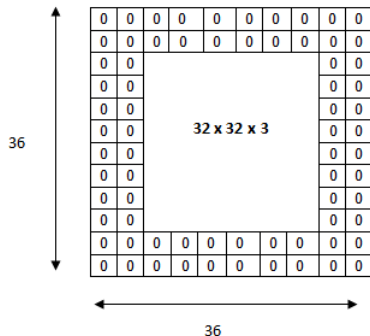
- Shared / Tied weights — improves generalization

# Why to Stride with $S > 1$ ?

- Striding with  $S > 1$  is equivalent to stride with  $S = 1$  and then downsampling [Goodfellow et al., 2016]



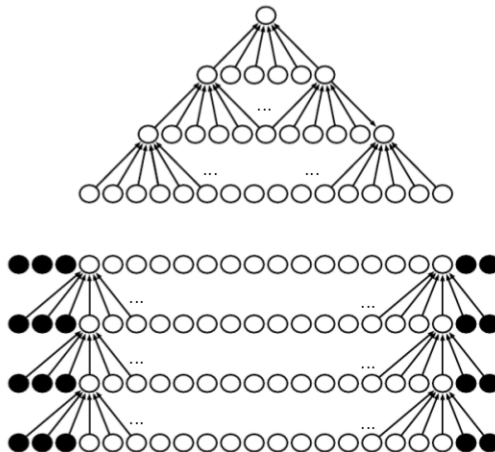
# Convolution Padding



The input volume is 32 x 32 x 3. If we imagine two borders of zeros around the volume, this gives us a 36 x 36 x 3 volume. Then, when we apply our conv layer with our three 5 x 5 x 3 filters and a stride of 1, then we will also get a 32 x 32 x 3 output volume.

# Why to Pad?

- It may not be possible to increase the depth of convolutions without padding [Goodfellow et al., 2016]





# Pooling

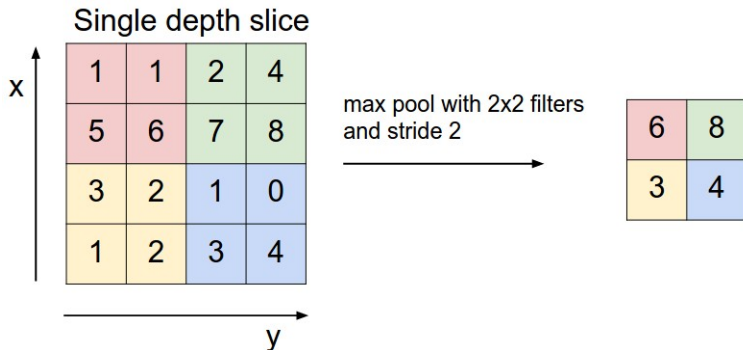
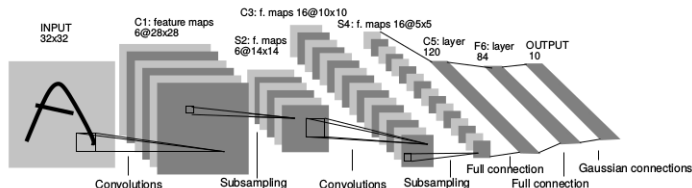


Figure: Max-Pooling

- Sub-sampling / down-sampling — improves generalization
- Average pooling is also possible
- Filter size and stride are usually same — no overlapping

# Image Classification: LeNet5



**Figure:** Convolution Network for Character Identification [LeCun et al., 1998]

- Trained using  $32 \times 32$  pixel size images from MNIST (at most  $20 \times 20$  characters centered in  $28 \times 28$ )
- $C_x$  are convolutional layers (C1, C3, C5)
- $S_x$  are subsampling (pooling) layers (S1, S4)
- One fully connected hidden layer (F6) with 84 neurons and an output layer with 10 neurons

- About 15 million images belonging to about 22000 categories
- All collected from Web and labeled by human (Amazon's Mechanical Turk crowd sourcing)
- RGB Images with varying resolutions
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
  - 1000 categories
  - 1.2 million training samples (roughly 1000 per category)
  - 50,000 validation images
  - 150,000 test images
  - Performance Measures: top-1 and top-5 error rates

# ImageNet Samples

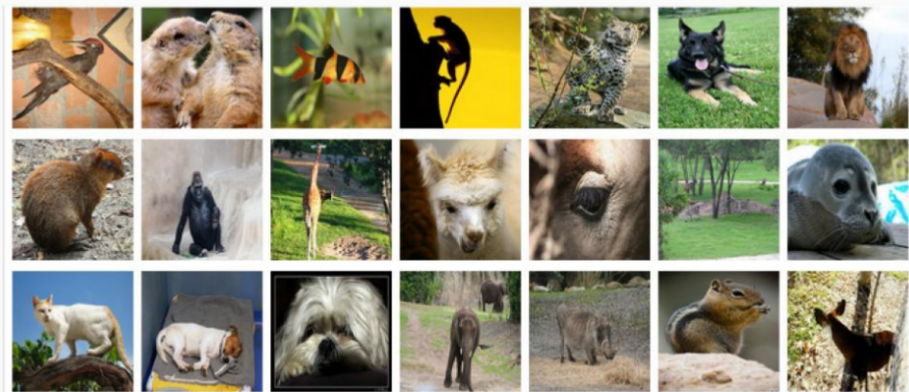


Figure: Sample Images from ImageNet

# CNN Architecture for ImageNet: AlexNet

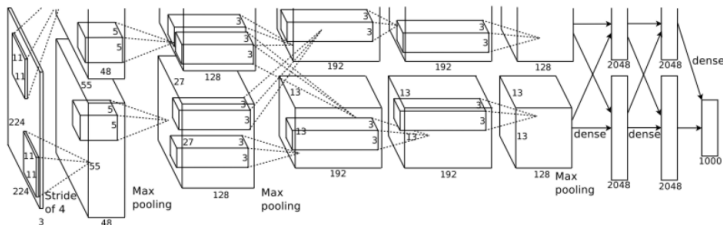
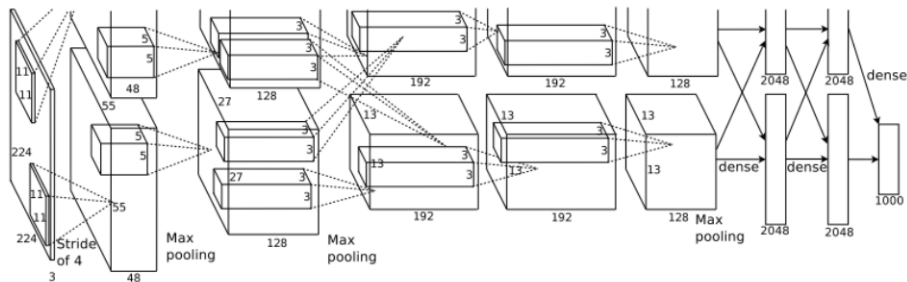


Figure: AlexNet Architecture for ImageNet [Krizhevsky et al., 2012]

- Images were down-sampled to a fixed resolution of 256x256
- Two GPUs were used — “columnar” impact on the CNN architecture (communications only in a few layers)
- ReLU activation was employed
- Dropout mechanism was used to reduce overfitting

# CNN Architecture for ImageNet: AlexNet



- Eight layers with weights: 5 convolutional + 3 fully connected
- Output Layer: 1000-way Softmax
- Two preceding fully connected hidden layers had 4096 neurons each
- Stochastic gradient descent to maximize the average log-probability of correct label
- Achieved an error rate of 15.3% in ILSVRC-2012

# Other CNN Architectures for ImageNet

- ZF-Net [Zeiler and Fergus, 2013]
- VGG-Net [Simonyan and Zisserman, 2014]
- GoogLeNet [Szegedy et al., 2015a]
- Google Inception Networks  
[Szegedy et al., 2015b, Szegedy et al., 2016]
- Microsoft ResNet [He et al., 2015, He et al., 2016]

- 1 Introduction
- 2 Neural Networks
- 3 Deep Neural Networks
- 4 Architectures for Deep Learning**
  - Convolutional Neural Networks
  - Recurrent Neural Networks
- 5 Tools for Deep Learning
- 6 Summary



# Hopfield Network

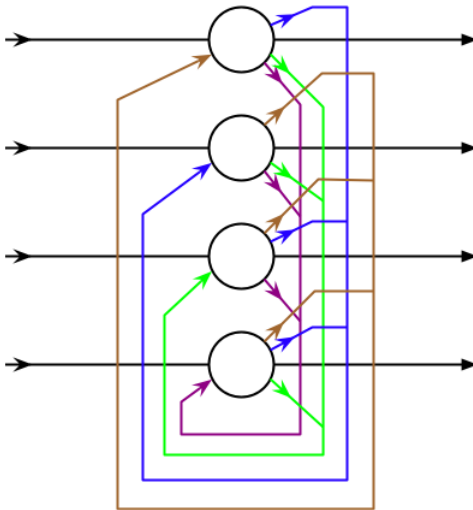


Figure: Hopfield Network (Figure adopted from Wikipedia)

# RNN for Sequence to Sequence

- Feed-back mechanism permits information to persist by passing essence of history to process the current input
- Suitable for sequence generation tasks

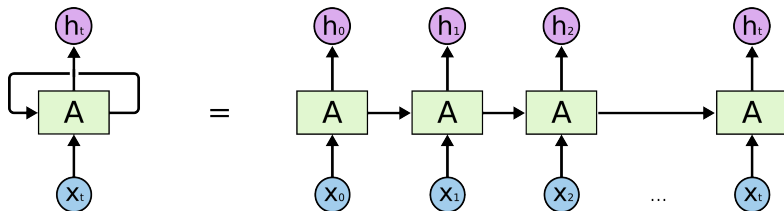


Figure: RNN Unrolled (Figure adopted from blog of Christopher Olah)

# RNN — Short-Term and Long-Term Context

- RNNs are particularly useful for “Short-Term” context, where “gap” between context and current output is short
- “ . . . when the sun rises in the — ”

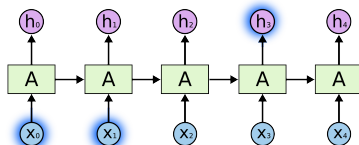


Figure: RNN Short-Term Context (Figure adopted from blog of Christopher Olah)

# RNN — Short-Term and Long-Term Context

- However, performance of RNN goes down when “long-term” context is required — error Back-propagated through time either explodes or vanishes [Hochreiter and Schmidhuber, 1997]
- “Constanzo Beschi arrived in Madurai in 1711 and had spent several years in Tamilnadu since then. Born in Castiglione delle Stiviere, in the Duchy of Mantua, Beschi got his secondary education in the Jesuits High School at Mantua. (blah ... blah ... blah) He speaks fluent —”

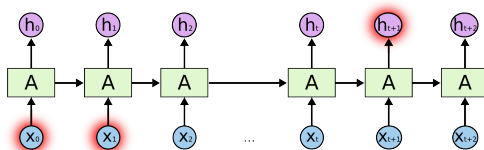
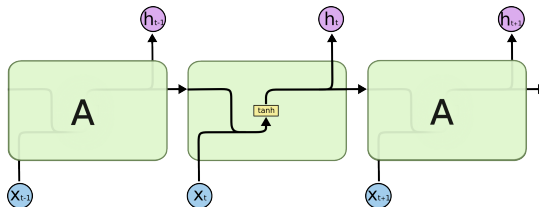
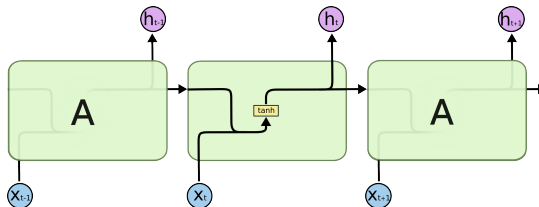


Figure: RNN Long-Term Context (Figure adopted from blog of Christopher Olah)

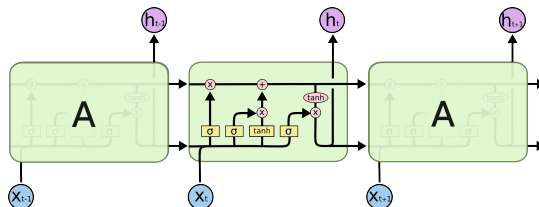
- Standard RNN



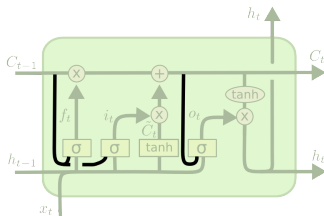
- Standard RNN



- LSTM with “gates” — ensures constant error flow (constant error carrousel) [Hochreiter and Schmidhuber, 1997]



# LSTM — Variations

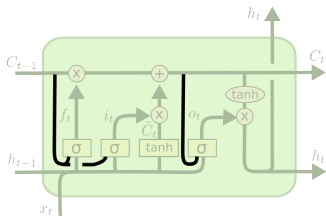


$$f_t = \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o)$$

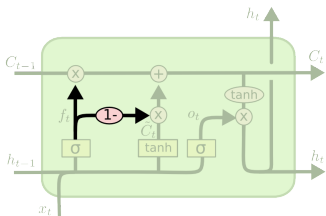
# LSTM — Variations



$$f_t = \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$

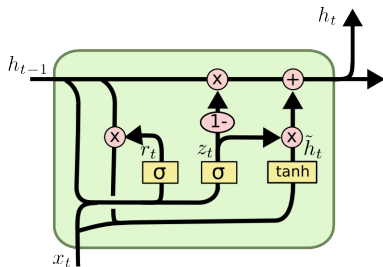
$$i_t = \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o)$$



$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t$$





$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

# Recurrent Seq2Seq Architectures

- Encode variable-length sequence to a fixed size representation vector and decode it to a variable-length sequence  
[Sutskever et al., 2014, Bahdanau et al., 2014, Cho et al., 2014]

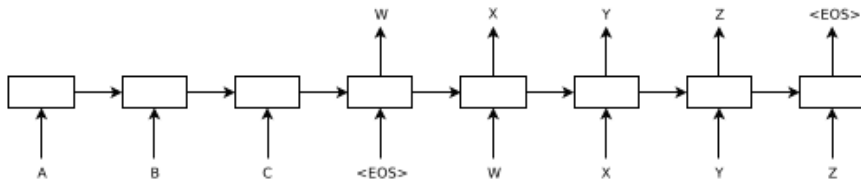


Figure: LSTM based seq2seq encoder-decoder [Sutskever et al., 2014]

# Recurrent Seq2Seq Architectures

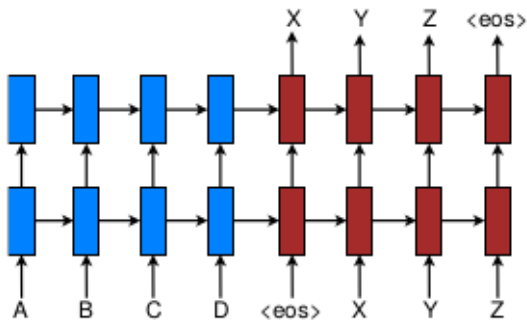


Figure: Stacked seq2seq encoder-decoder [Luong et al., 2015]

- Attention mechanism [Bahdanau et al., 2014, Luong et al., 2015]
- Bi-Directional encoders  
[Bahdanau et al., 2014, Zhou et al., 2016, Wu et al., 2016]
- Residual connections [Zhou et al., 2016, Wu et al., 2016]

# Outline

- 1 Introduction
- 2 Neural Networks
- 3 Deep Neural Networks
- 4 Architectures for Deep Learning
  - Convolutional Neural Networks
  - Recurrent Neural Networks
- 5 Tools for Deep Learning
- 6 Summary

- TensorFlow

- <https://www.tensorflow.org/>
- Open-source software library (Python) originally developed by Google Brain Team

- Theano

- <http://deeplearning.net/software/theano/>
- A Python library for fast mathematical computations

- CNTK — Microsoft Cognitive Toolkit

- <https://docs.microsoft.com/en-us/cognitive-toolkit/>
- Networks can be defined and evaluated in Python, C++, BrainScript
- Models can be evaluated in C# as well

- Keras — frontend for TensorFlow, CNTK, and Theano

- <https://keras.io/>
- Python front-end for TensorFlow, Theano, or CNTK

- Torch

- <http://torch.ch/>
- C/CUDA implementation with LuaJIT front-end
- PyTorch provides python front-end
- Embeddable, with ports to iOS, Android and FPGA back-ends

- Caffe

- <http://caffe.berkeleyvision.org/>
- A deep learning framework (C++) developed by the Berkeley Vision and Learning Center
- PyCaffe provides Python front-end

- MXNet

- <http://mxnet.io/>
- Efficient C++ implementation with multiple frontends

- Blocks and Lasagne — frontends for Theano

- <https://blocks.readthedocs.io/en/latest/>
- <https://lasagne.readthedocs.io/en/latest/>
- Provide frameworks to build and manage NN models

# Outline

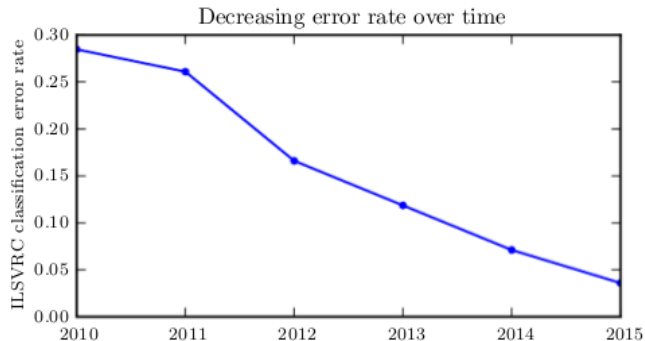
- 1 Introduction
- 2 Neural Networks
- 3 Deep Neural Networks
- 4 Architectures for Deep Learning
  - Convolutional Neural Networks
  - Recurrent Neural Networks
- 5 Tools for Deep Learning
- 6 Summary



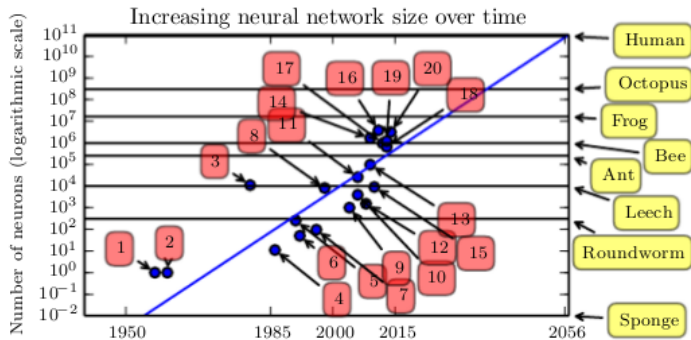
# Summary

- Machine learning is about using the right features to build the right models that achieve the right tasks
- In this talk, we have focused on recent developments that led to deep learning architectures
- In particular, we have discussed about Convolutional Neural Networks, RNN, LSTM, and Seq2seq architectures
- We have highlighted certain techniques, such as new activation functions, data augmentation, dropout regularization, stochastic gradient descend, parameter sharing, GPU-based computations, etc., that have made deep learning possible

# Error Rate Drop



# Growth of Neurons



## Growth Rate

Size of neural networks have doubled in size roughly every 2.4 years

# References I



Abadi, M. et al. (2015).

TensorFlow: Large-scale machine learning on heterogeneous systems.  
Software available from <https://www.tensorflow.org/>.



Abu-Mostafa, Y. S., Magdon-Ismail, M., and Lin, H.-T. (2012).

*Learning from Data — A Short Course.*  
AMLbook.



Alpaydin, E. (2010).




*Introduction to Machine Learning.*  
The MIT Press, Second edition.



Anderson, J. A. (1995).

*An introduction to neural networks.*  
The MIT Press.

# References II

-  Bahdanau, D., Cho, K., and Bengio, Y. (2014).  
Neural machine translation by jointly learning to align and translate.  
*CoRR*, abs/1409.0473.
-  Bengio, Y. (2009).  
Learning deep architectures for AI.  
*Foundations and Trends in Machine Learning*, 2(1):1–127.
-  Bengio, Y., Courville, A., and Vincent, P. (2013).  
Representation learning: A review and new perspectives.  
*IEEE Transactions and Pattern Recognition and Machine Intelligence*, 35(8):1798–1828.
-  Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017).  
Julia: A fresh approach to numerical computing.  
*SIAM Review*, 59:65–98.  
Software available from <https://julialang.org/>.

# References III



Blum, A. and Rivest, R. L. (1989).

Training a 3-node neural net is NP-complete.

In *Advances in neural information processing systems*, volume 1, pages 494–501. Morgan-Kaufmann.



Chatfield, K., Simonyan, K., Vedaldi, A., and Zisserman, A. (2014).

Return of the devil in the details: Delving deep into convolutional nets.

*CoRR*, abs/1405.3531.



Cho, K., van Merriënboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014).

Learning phrase representations using RNN encoder–decoder for statistical machine translation.



# References V



Goodfellow, I., Bengio, Y., and Courville, A. (2016).

*Deep Learning*.

MIT Press.

[www.deeplearningbook.org](http://www.deeplearningbook.org).



Graves, A. and Schmidhuber, J. (2005).

Framewise phoneme classification with bidirectional LSTM and other neural network architectures.

*Neural Networks*, 18(5–6):602–610.



Greff, K., Srivatsava, R. K., and Koutnik, J. (2017).

LSTM: A search space odyssey.

*IEEE Transactions on Neural Networks and Learning Systems*.







Hassoun, M. H. (1995).

*Fundamentals of Artificial Neural Networks*.

The MIT Press.



# References VI

-  He, K., Zhang, X., Ren, S., and Sun, J. (2015).  
Deep residual learning for image recognition.  
*CoRR*, [abs/1512.03385](#).
-  He, K., Zhang, X., Ren, S., and Sun, J. (2016).  
Identity mappings in deep residual networks.  
*CoRR*, [abs/1603.05027](#).
-  Hertz, J., Krogh, A., and Palmer, R. G. (1991).  
*Introduction to the theory of neural computing*.  
Addison-Wesley Publishing Company.
-  Hochreiter, S. and Schmidhuber, J. (1997).  
Long Short-Term Memory.  
*Neural Computation*, 9(8):1735–1780.

# References VII



Hornik, K., Stinchcomb, M., and White, H. (1989).  
Multilayer feedforward networks are universal approximators.  
*Neural Networks*, 2(5):359–366.



Jaderberg, M., Simonyan, K., Zisserman, A., and Kavukcuoglu, K. (2015).  
Spatial transformer networks.  
In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 2017–2025. Curran Associates Inc.



Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009).  
What is the best multi-stage architecture for object recognition?  
In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153.

# References VIII



Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014).

Caffe: Convolutional architecture for fast feature embedding.  
*CoRR*, abs/1408.5093.

Software available from <http://caffe.berkeleyvision.org/>.



Jordan, M. I. and Mitchell, T. M. (2015).

Machine learning: Trends, perspectives, and prospects.  
*Science*, 349(6245):255–260.



Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015).

An empirical exploration of recurrent network architectures.

In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 2342–2350. JMLR.org.

# References IX



Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012).  
Imagenet classification with deep convolutional neural networks.  
In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q.,  
editors, *Advances in Neural Information Processing Systems 25*, pages  
1097–1105. Curran Associates, Inc.  
<http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.



LeCun, Y. (1989).  
Generalization and network design strategies.  
Technical Report CRG-TR-89-4, University of Toronto, Canada.



LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998).  
Gradient-based learning applied to document recognition.  
In *Proceedings of the IEEE*, volume 86, pages 2278–2324.

# References X



LeCun, Y., Kavukcuoglu, K., and Farabet, C. (2010).

Convolutional networks and applications in vision.

In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 253–256. IEEE.



Lin, M., Chen, Q., and Yan, S. (2013).

Network in network.

*CoRR*, abs/1312.4400.



Luong, M., Pham, H., and Manning, C. D. (2015).

Effective approaches to attention-based neural machine translation.

*CoRR*, abs/1508.04025.



Mitchell, T. M. (1997).

*Machine Learning*.

McGraw-Hill.

# References XI



Nair, V. and Hinton, G. E. (2010).

Rectified linear units improve restricted boltzmann machines.

*In Proceedings of 27th International Conference on Machine Learning.*



Nowlan, S. J. and Hinton, G. E. (1992).

Simplifying neural networks by soft weight-sharing.

*Neural Computation*, 4:473–493.



Ramachandran, P., Zoph, B., and Le, Q. V. (2017).

Searching for activation functions.

*CoRR*, abs/1710.05941.




Rumelhart, D. E., Hinton, G. E., and Willams, R. J. (1986).


Learning representations by back-propagating errors.

*Nature*, pages 533–536.





doi:10.1038/323533a0.

 Russel, S. and Norvig, P. (2009).  
*Artificial Intelligence — A Modern Approach*.  
Prentice Hall, Third edition.




 Schmidhuber, J. (2014).  
Deep learning in neural networks: An overview.  
Technical report, Istituto Dalle Molle di Studi sull'Intelligenza  
Artificiale.

 Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and  
LeCun, Y. (2013).  
Overfeat: Integrated recognition, localization, and detection using  
convolutional networks.  
*CoRR*, abs/1312.6229.

# References XIII

-  Simonyan, K. and Zisserman, A. (2014).  
Very deep convolutional networks for large-scale image recognition.  
*CoRR*, abs/1409.1556.
-  Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014).  
Dropout: A simple way to prevent neural networks from overfitting.  
*Journal of Machine Learning Research*, 15:1929–1958.
-  Sutskever, I., Vinyals, O., and Le, Q. V. (2014).  
Sequence to sequence learning with neural networks.  
*CoRR*, abs/1409.3215.
-  Szegedy, C., Ioffe, S., and Vanhoucke, V. (2016).  
Inception-v4, inception-resnet and the impact of residual connections on learning.  
*CoRR*, abs/1602.07261.



-  Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015a).  
Going deeper with convolutions.  
*In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9. IEEE.
-  Szegedy, C., Vanhoucke, V., Ioffe, S., and Shlens, J. (2015b).  
Rethinking the inception architecture for computer vision.  
*CoRR*, abs/1512.00567.
-  Technology, M. and Research Division (2017).  
The Microsoft Cognitive Toolkit.  
<https://www.microsoft.com/en-us/cognitive-toolkit/>.



Theano Development Team (2016).

Theano: A Python framework for fast computation of mathematical expressions.

*CoRR*, abs/1605.02688.

Software available from

<http://deeplearning.net/software/theano/>.



Theodoridis, S. and Koutroumbas, K. (2009).

*Pattern Recognition*.

Academic Press, Fourth edition.



Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016).

Google's neural machine translation system: Bridging the gap between human and machine translation.

*CoRR*, abs/1609.08144.



Zeiler, M. D. and Fergus, R. (2013).

Visualizing and understanding convolutional networks.

*CoRR*, abs/1311.2901.



Zhou, J., Cao, Y., Wang, X., Li, P., and Xu, W. (2016).  
Deep recurrent models with fast-forward connections for neural  
machine translation.  
*CoRR*, [abs/1606.04199](https://arxiv.org/abs/1606.04199).

# QUESTIONS?