Demystifying Deep Learning

2017 Workshop on Deep Learning in Medicine

Mark Crowley
Assistant Professor
Electrical and Computer Engineering
University of Waterloo
mcrowley@uwaterloo.ca

Aug 28, 2017

Outline

My Background

- Waterloo : Assistant Professor, ECE Department since 2015
- PhD at UBC in Computer Science with Prof. David Poole
- Postdoc at Oregon State University
- UW ECE ML Lab: https://uwaterloo.ca/scholar/mcrowley/lab
- Waterloo Institute for Complexity and Innovation (WICI)

Good Problems vs Important Problems

- Playing Chess - *cool*

Good Problems vs Important Problems

- Playing Chess - *cool*
- Classifying cat videos - *surprising*

Good Problems vs Important Problems

- Playing Chess - *cool*
- Classifying cat videos - *surprising*
- Alpha Go **amazing!**

Good Problems vs Important Problems

- Playing Chess - *cool*
- Classifying cat videos - *surprising*
- Alpha Go **amazing!**
- Text and speech recognition, translation - **also handy!**

Good Problems vs Important Problems

- Playing Chess - *cool*
- Classifying cat videos - *surprising*
- Alpha Go **amazing!**
- Text and speech recognition, translation - **also handy!**
- Advertisement Ranking profit!!!

Good Problems vs Important Problems

- Playing Chess - *cool*
- Classifying cat videos - *surprising*
- Alpha Go **amazing!**
- Text and speech recognition, translation - **also handy!**
- Advertisement Ranking profit!!!
- no seriously though, how about something *important?*

Good Problems vs Important Problems

- Playing Chess - *cool*
- Classifying cat videos - *surprising*
- Alpha Go **amazing!**
- Text and speech recognition, translation - **also handy!**
- Advertisement Ranking profit!!!
- no seriously though, how about something *important?*
- Some of these can be important, but they have low risk if youre wrong.

My Research

Topics

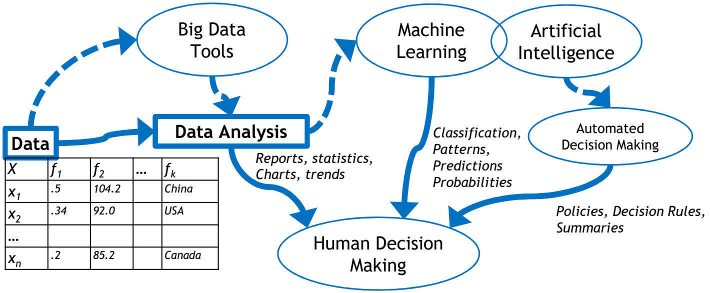
- **Computational Sustainability**
- **Deep Learning**
- Reinforcement Learning
- **Anomaly Detection**
- Natural Language Processing
- **Deep RL**
- Security
- PAC MDP Learning
- Graphical Models

Projects

- Forest Harvest Simulators
- **Forest Fire Satellite Images/Simulations**
- Invasive Species Simulations
- Embedded System Log Traces
- Driver Behaviour Modelling
- Formation Control and Learning in Remote Control Cars
- **diffusion MRI Brain Scans**
- Learning to Detect Anomalous Network Traffic
- **Therapeutic Chatbots for the Elderly**

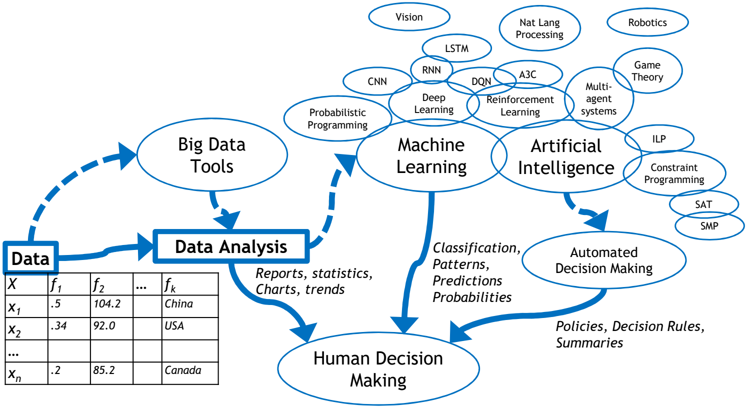
Outline

Data, Big Data, Machine Learning, AI, etc, etc,


Big Data Tools
Machine Learning
Artificial Intelligence
Data
Data Analysis
Automated Decision Making
Reports, statistics, Charts, trends
Classification, Patterns, Predictions Probabilities
Policies, Decision Rules, Summaries
Human Decision Making

X	f_1	f_2	...	f_k
x_1	.5	104.2		China
x_2	.34	92.0		USA
...				
x_n	.2	85.2		Canada

Data, Big Data, Machine Learning, AI, etc, etc,



X	f_1	f_2	...	f_k
x_1	.5	104.2		China
x_2	.34	92.0		USA
...				
x_n	.2	85.2		Canada

Machine Learning

- Supervised Learning
  - Also called: predictive, inferential
  - Given inputs and outputs
  - Model based: model or distribution is known
    - Parameter Estimation: distribution known, fitting parameters
    - Linear Regresion, least squares estimation
  - output is linear: regression, prediction
  - output is categorical (label) : **classification**
- Unsupervised Learning
  - Also called: descriptive, knowledge discovery
  - Given inputs only
  - output is categorical : clustering
- Reinforcement Learning
  - Input and output given only when action (prediction, choice, activity) provided.
  - Given feedback about utility of choice made.
  - No given model or labels ahead of time.
  - Learning is interactive.

Classification Definition

Definition:

- Classification is a learning method that uses training samples (with known class labels) to learn how to assign the samples into their proper classes. The task of the clasifier is to use the feature vector provide by the feature extractor to assign the object (datapoint) to a category.
- This learning can then be used to label test samples (with unknown labels).
- Classification can be facilitated if the features (or a subset of them) of samples in the same class share characteristics that discriminate them from other classes.
- It can be seen as the discrete form of Prediction, can you map the input values to a discrete set of output values rather than to a continuous number.

Clustering vs. Classification

Clustering

- Unsupervised
- Uses unlabeled data
- Organize patterns w.r.t. an optimization criteria
- Notion of similarity
- Hard to evaluate
- Example: K-means, Fuzzy C-means, Hierarchical

Classification

- Supervised
- Uses labeled data
- Requires training phase
- Domain sensitive
- Easy to evaluate
- Examples: Naive Bayes, KNN,SVM, Decision Trees

Classification Definition

- Given a dataset $X = \{x_1, x_2, \dots, x_n\}$ where each datapoint $x_i \in X$ contains F features denoted $x_{i,f}$,
- and given a set of classes $C = \{C_1, \dots, C_K\}$,
- the **classification problem** is to define a mapping $\delta(x_i) : X \to C$ where each x_i is assigned to one class.
- A **class**, C_j, contains precisely those points mapped to it, no more and no less.
- Note: the classes are predefined, nonoverlapping and partition the entire set of datapoints.

Outline

Linear Regression vs. Logistic Regression

- A simple type of **Generalized Linear Model**
- Linear regression learns a function to predict a continuous variable output of continous or discrete input variables

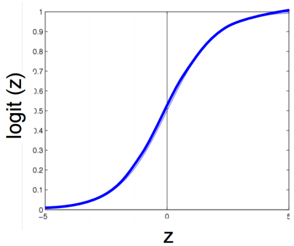
$$Y = b_0 + \sum (b_i X_i) + \epsilon$$

- Logistic regression predicts the probability of an outcome, the appropriate class for an input vector or the **odds** of one outcome being more likely than another.

Logistic Regression

Define probability of label being 1 by fitting a linear weight vector to the logistic (a.k.a sigmoid) function.

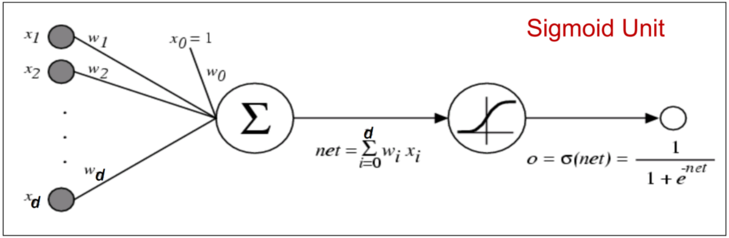
$$P(Y=1|X) = \frac{1}{1+e^{-(w_0+\sum_i w_i x_i)}}$$
$$= \frac{1}{1+\exp(-(w_0+\sum_i w_i x_i))}$$



Advantage of this is you can turn the continuous $[-\infty, \infty]$ feature information into $[0,1]$ and treat it like a probability.
bias: w_0 is called the bias, it basically adjusts the sigmoid curve to the left or right, biasing what the expected outcome is. The other weights w_i adjust the steepness of the curve in each dimension.

Logistic Regression as a Graphical Model

$$o(\mathbf{x}) = \sigma(w^T x_i) = \sigma(w_0 + \sum_i w_i x_i) = \frac{1}{1 + \exp(-(w_0 + \sum_i w_i x_i))}$$


Sigmoid Unit
x_1
x_2
x_d
w_1
w_2
w_d
$x_0 = 1$
w_0
Σ
$net = \sum_{i=0}^{d} w_i x_i$
$o = \sigma(net) = \frac{1}{1 + e^{-net}}$

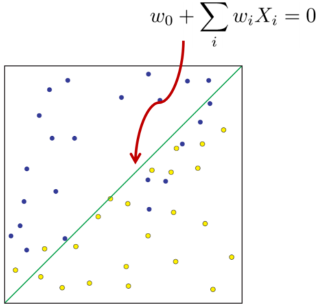
Logistic Regression Used as a Classifier

Logistic Regression can be used as a simple linear classifier.

- Compare probabilities of each class $P(Y = 0|X)$ and $P(Y = 1|X)$.
- Treat the halfway point on the sigmoid as the decision boundary.

$P(Y = 1|X) > 0.5$ classify X in class 1

$$w_0 + \sum_i w_i x_i = 0$$



Training Logistic Regression Model via Gradient Descent

- Can't easily perform Maximum Likelihood Estimation
- The negative log-likehood of the logistic function is given by *NLL* and it's gradient by *g*

$$NLL(w) = \sum_{i=1}^{N} \log\left(1 + \exp(-(w_0 + \sum_i w_i x_i))\right)$$

$$g = \frac{\partial}{\partial w} = \sum_i (\sigma(w^T x_i) - y_i) x_i$$

Then we can update the parameters iteratively

$$\theta_{k+1} = \theta_k - \eta_k g_k$$

where η_k is the learning rate or step size.

Support Vector Machines (SVMs)

- SVM learns a discriminant function and threshold to divide points into two classes.
- Originally this was a line $\mathbf{w}^T\mathbf{x} + b$ where weights $\mathbf{w}$ and offset b are learned so that possitive training points are above the line.
- More generally we can learn a nonlinear function using the **kernal trick**.

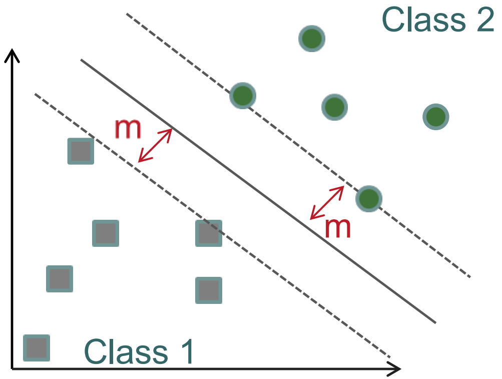
We rewrite the linear system as a dot product:

$$\mathbf{w}^T\mathbf{x} + b = b + \sum_{i=1}^{m} \alpha_i \mathbf{x}^T \mathbf{x}^{(i)}$$

where

- $\mathbf{x}^{(i)}$ is a training example
- α is a vector of coefficients

Support Vector Machines



Support Vector Machines (SVMs)

- More generally we can learn a nonlinear function using the **kernal trick**.

We rewrite the linear system as a dot product:

$$\mathbf{w}^T\mathbf{x} + b = b + \sum_{i=1}^{m} \alpha_i \mathbf{x}^T \mathbf{x}^{(i)}$$

$$f(x) = b + \sum_{i} \alpha_i k(\mathbf{x}, \mathbf{x}^{(i)})$$

where

- $\mathbf{x}^{(i)}$ is a training example
- α is a vector of coefficients
- $k(\mathbf{x}, \mathbf{x}^{(i)}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{x})^T$ is a kernal.

Pros and Cons of Kernal Methods

Pro:

- They map low dimensional linear space to higher dimensional nonlinear space.

Cons:

- Evaluation is expensive - linear in the weighted data points usd $\alpha_i k(\mathbf{x}\mathbf{x}^{(i)})$
  - kernal SVMs avoid this problem because most of $\alpha_i = 0$, only the **support vectors** $\mathbf{x}^{(i)}$ are computed.
- Training is expensive - for large datasets
- Do not Generalize well - the kernal determines what kind of general patterns it can learn.
- Choosing the kernal is hard - too general or too specific to the problem, how to choose it?

Outline

A Short History

40's Early work in NN goes back to the 40s with a simple model of the neuron by McCulloh and Pitt as a summing and thresholding devices.

A Short History

40's Early work in NN goes back to the 40s with a simple model of the neuron by McCulloh and Pitt as a summing and thresholding devices.

1958 Rosenblatt in 1958 introduced the **Perceptron**,a two layer network (one input layer and one output node with a bias in addition to the input features.

1969 Marvin Minsky: 1969. Perceptrons are 'just' linear, AI goes logical, beginning of "AI Winter"

A Short History

40's	Early work in NN goes back to the 40s with a simple model of the neuron by McCulloh and Pitt as a summing and thresholding devices.
1958	Rosenblatt in 1958 introduced the **Perceptron**,a two layer network (one input layer and one output node with a bias in addition to the input features.
1969	Marvin Minsky: 1969. Perceptrons are 'just' linear, AI goes logical, beginning of "AI Winter"
1980s	Neural Network resurgence: Backpropagation (updating weights by gradient descent)

A Short History

- 40's Early work in NN goes back to the 40s with a simple model of the neuron by McCulloh and Pitt as a summing and thresholding devices.
- 1958 Rosenblatt in 1958 introduced the **Perceptron**,a two layer network (one input layer and one output node with a bias in addition to the input features.
- 1969 Marvin Minsky: 1969. Perceptrons are 'just' linear, AI goes logical, beginning of "AI Winter"
- 1980s Neural Network resurgence: Backpropagation (updating weights by gradient descent)
- 1990s SVMs! Kernals can do **anything!** (no, they can't)

A Short History

1993 `LeNet 1` for digit recognition

A Short History

1993 `LeNet 1` for digit recognition
2003 Deep Learning (Convolutional Nets Dropout/RBMs, Deep Belief Networks)

A Short History

1993	`LeNet 1` for digit recognition
2003	Deep Learning (Convolutional Nets Dropout/RBMs, Deep Belief Networks)
1986, 2006	Restricted Boltzman Machines

A Short History

1993	`LeNet 1` for digit recognition
2003	Deep Learning (Convolutional Nets Dropout/RBMs, Deep Belief Networks)
1986, 2006	Restricted Boltzman Machines
2006	Neural Network outperform RBF SVM on MNIST handwriting dataset (Hinton et al.)

A Short History

1993	`LeNet 1` for digit recognition
2003	Deep Learning (Convolutional Nets Dropout/RBMs, Deep Belief Networks)
1986, 2006	Restricted Boltzman Machines
2006	Neural Network outperform RBF SVM on MNIST handwriting dataset (Hinton et al.)
2012	`AlexNet` for **ImageNet** challenge - this algorithm beat competition by error rate of 16% vs 26% for next best

A Short History

1993	`LeNet 1` for digit recognition
2003	Deep Learning (Convolutional Nets Dropout/RBMs, Deep Belief Networks)
1986, 2006	Restricted Boltzman Machines
2006	Neural Network outperform RBF SVM on MNIST handwriting dataset (Hinton et al.)
2012	`AlexNet` for **ImageNet** challenge - this algorithm beat competition by error rate of 16% vs 26% for next best • ImageNet : contains 15 million annotated images in over 22,000 categories.

A Short History

1993	`LeNet 1` for digit recognition
2003	Deep Learning (Convolutional Nets Dropout/RBMs, Deep Belief Networks)
1986, 2006	Restricted Boltzman Machines
2006	Neural Network outperform RBF SVM on MNIST handwriting dataset (Hinton et al.)
2012	`AlexNet` for **ImageNet** challenge - this algorithm beat competition by error rate of 16% vs 26% for next best

- ImageNet : contains 15 million annotated images in over 22,000 categories.
- ZFNet paper (2013) extends this and has good description of network structure

A Short History

1993 `LeNet 1` for digit recognition

2003 Deep Learning (Convolutional Nets Dropout/RBMs, Deep Belief Networks)

1986, 2006 Restricted Boltzman Machines

2006 Neural Network outperform RBF SVM on MNIST handwriting dataset (Hinton et al.)

2012 `AlexNet` for **ImageNet** challenge - this algorithm beat competition by error rate of 16% vs 26% for next best

- ImageNet : contains 15 million annotated images in over 22,000 categories.
- ZFNet paper (2013) extends this and has good description of network structure

2012-present Google Cat Youtube, speech recognition, self driving cars, computer defeats regional Go champion, ...

A Short History

1993 `LeNet 1` for digit recognition

2003 Deep Learning (Convolutional Nets Dropout/RBMs, Deep Belief Networks)

1986, 2006 Restricted Boltzman Machines

2006 Neural Network outperform RBF SVM on MNIST handwriting dataset (Hinton et al.)

2012 `AlexNet` for **ImageNet** challenge - this algorithm beat competition by error rate of 16% vs 26% for next best

- ImageNet : contains 15 million annotated images in over 22,000 categories.
- ZFNet paper (2013) extends this and has good description of network structure

2012-present Google Cat Youtube, speech recognition, self driving cars, computer defeats regional Go champion, ...

2014 `GoogLeNet` added many layers and introduced inception modules (allows parallel computation rather than serially

A “New” Age of Neural Networks

- Geoffrey Hinton (UofT, now Google Research), paper references
  - early work on back-propogation
  - Restricted Boltzman Machines - first deep architecture, idea for unsupervised pretraining of layers for RBMs was used for first successful use of deep networks.
  - Several applications of Deep Learning to image recognition, speech recognition,...
- Yoshua Bengio (University of Montreal, MILA, ElememntAI, ...)
  - ReLU improves efficiency
  - unsupervised representation learning of autoencoders
  - word2vec - learn general representation of words for classification, translation, etc.
  - General Adversarial Networks - an exciting new approach to training
- Yann LeCun (NYU, now Facebook AI Research)
  - Convolutional Neural Networks

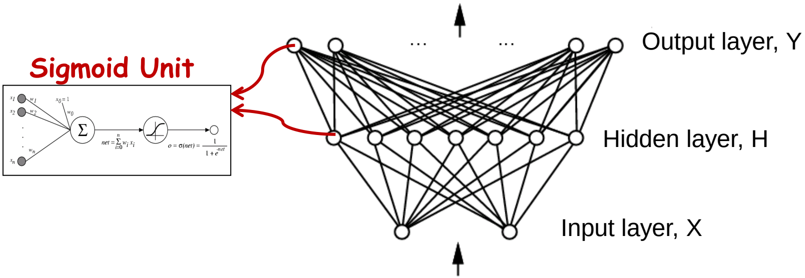
Neural Networks

- A Neural Network (NN) connects many nonlinear (classically logistic/sigmoid) units together into a network
- Central difference from the Perceptron: a layer of **hidden units**
- Also called: Multi-layer Perceptrons (MLP), Artificial Neural Networks (ANNs)

Neural Networks to learn $f : X \rightarrow Y$

- f can be a non-linear function
- **X** (vector of) continuous and/or discrete variables
- **Y** (vector of) continuous and/or discrete variables

Feedforward Neural networks - Represent f by network of non-linear (logistic/sigmoid/ReLU) units:



Basic Three Layer Neural Network

Input Layer

- vector data, each input collects **one feature**/dimension of the data and passes it on to the (first) hidden layer.

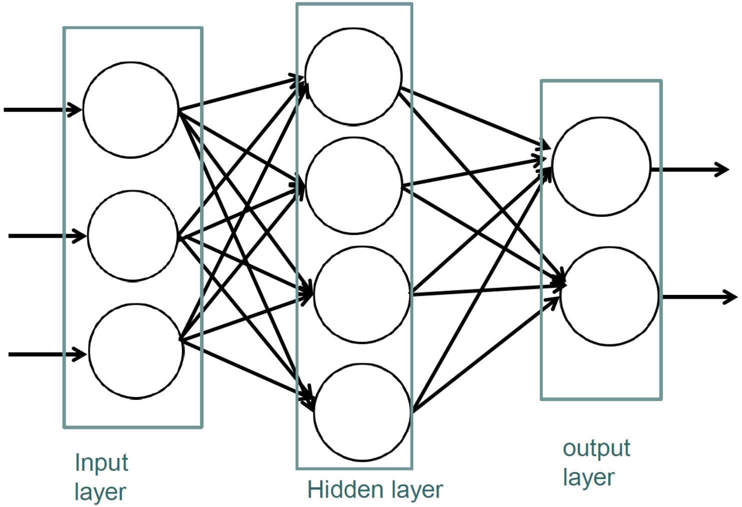
Hidden Layer

- Each hidden unit computes a weighted sum of all the units from the input layer (or any previous layer) and passes it through a **nonlinear activation function**.

Output Layer

- Each output unit computes a weighted sum of all the hidden units and passes it through a (possibly nonlinear) **threshold function**.

Three-layer Artificial Neural Network



Properties of Neural Networks

- Given a large enough layer of hidden units (or multiple layers) a NN can represent **any function**.
- One challenge is regularization: how many hidden units or layers to include in the network? Number of inputs and outputs are provided but internal structure can be arbitrary.
    - too few units, network will have too few parameters, may not be able to learn complex functions
    - too many units, network will be overparameterized and won't be forced to learn a generalizable model

Representation Learning: classic statistical machine learning is about learning functions to map input data to output. But Neural Networks, and especially Deep Learning, are more about learning **a representation** in order to perform classification or some other task.

Properties of Neural Networks

- The hidden layer will have a number of units (design parameter).
- The hidden layers allow extracting more complex features and facilitates solving complex problems.
- Connection between the units (neurons) of all the layers can be **forward**, backward or both.
- Units can be fully or partially connected.
- Different arrangements make different types or models of the NN.
- Each unit will have a **activation function** (a thresholding function) and connections between the units that have weights

Net Activation

$$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji}$$

where:

- i indexes the inputs units
- j indexes the hidden units
- W_{ji} denotes the weights on input to hidden layer ("synaptic weights")

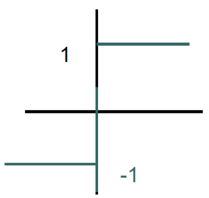
Hidden Layer: Adding Nonlinearity

- Each hidden unit emits an output that is a nonlinear **activation function** of its net activation.

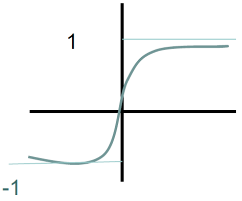
$$y_j = f(net_j)$$

- This is essential to neural networks power, if it's linear then it all becomes just linear regression.
- The output is thus thresholded through this nonlinear activation function.

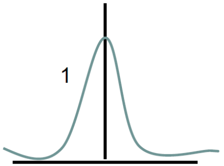
Activation Functions



a) Threshold



b) Sigmoid

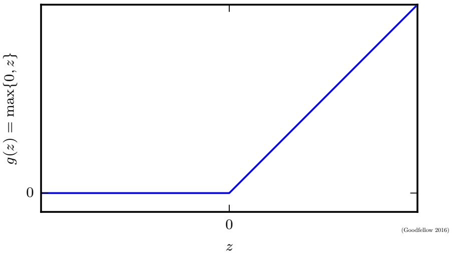


c) Gaussian

$$F_T(x) = \begin{cases} 1 & \text{if } x > \tau \\ -1 & \text{otherwise} \end{cases} \qquad F_S(x) = \frac{1}{\left(1 + e^{-cx}\right)} \qquad F_G(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

- `tanh` was another common function.
- `sigmoid` is now discourage except for final layer to obtain probabilities. Can **over-saturate** easily.
- ReLU is the new standard activation function to use.

Rectified Linear Activation



- **Rectified Linear Units (ReLU)** have become standard $\max(0, net_j)$
  - strong signals are alwasy easy to distinguish
  - most values are zero, deritive is mostly zero
  - they do not saturate as easily as sigmoid
- new Exponential linear units - evidence that they perform better than ReLU in some situations.

Output Layer

Regardless of activation function, the output layer then combines all the previous hidden layers.

$$net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0}$$

where:

- k indexes the output units
- n_H denotes the number of hidden units in the previous layer

Output Layer

The output layer adds *another* level of thresholding, usually nonlinear as well.

$$z_k = f(net_k)$$

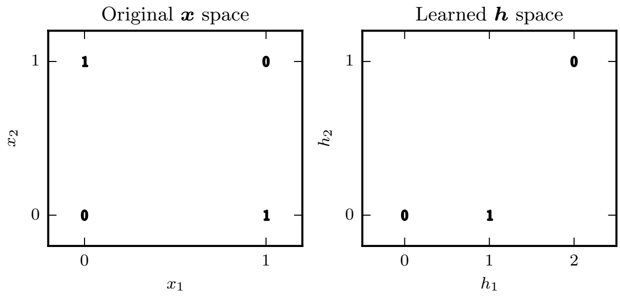
$f(net_k)$ could be any thresholding function such as the previous activation functions.

Outline

Failing on the XOR Problem

How is XOR a hard problem? It is if you're too linear. XOR is the simplest **nonlinear** classification problem.

- Two binary features x_1 and x_2 and four patterns to be assigned the correct labels `True` or `False`



(Goodfellow 2016)

Neural Nets Can Model XOR Problem

Use a simple `sign` activation function for hidden and output units:

$$\begin{aligned} f(net) &= 1 \text{ if } net \geq 0 \\ &= -1 \text{ if } net < 0 \end{aligned}$$

- First hidden unit:
  - y_1 computes $x_1 + x_2 + 0.5 = 0$
  - $y_1 = 1$ if $net_1 / ge0$, $y_1 = -1$ otherwise
- Second hidden unit:
  - y_1 computes $x_1 + x_2 - 1.5 = 0$
  - $y_2 = 1$ if $net_2 / ge0$, $y_2 = -1$ otherwise
- single output unit z_1:
  - emits $z_1 = 1$ iff $y_1 = 1$ and $y_2 = 1$.
  - y_1 models AND gate
  - y_2 models OR gate
  - z_1 models y_1 AND NOT y_2

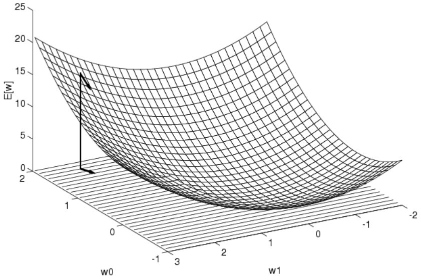
Full Neural Network Descriminant Function

The class of decision functions that can be described by a three-layer neural network are:

$$g_k(\mathbf{x}) = z_k = f\left(\sum_{j=1}^{n_H} w_{kj} f\left(\sum_{i=1}^{d} w_{ji} x_i + w_{j0}\right) + w_0\right)$$

- The **Universal Approximation Theorem** [Hornik, 1989] shows that this covers any possible decision function mapping continuous inputs to a finite set of classes to any desired level of accuracy.
- But it does not say how many hidden unites would be needed.
- In general, for a single layer it could be exponential (degrees of freedom).
- But, using more layers can reduce the number of hidden units needed and make it more generalizable.

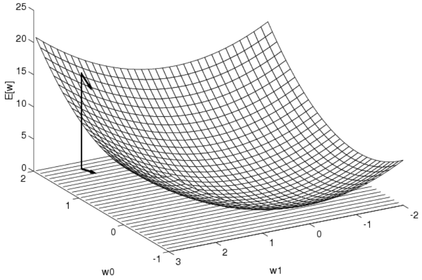
Gradient Descent



Error Function: Mean Squared Error, cross-entropy loss, etc.

(Slides from Tom Mitchell ML Course, CMU, 2010)

Gradient Descent

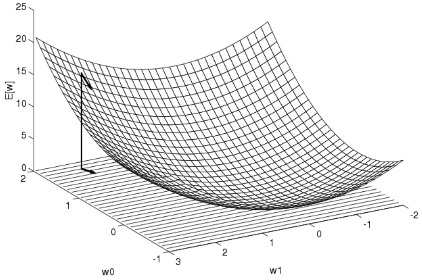


Error Function: Mean Squared Error, cross-entropy loss, etc.

Gradient: $\nabla E[\mathbf{w}] = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_d}\right]$

(Slides from Tom Mitchell ML Course, CMU, 2010)

Gradient Descent



Error Function: Mean Squared Error, cross-entropy loss, etc.

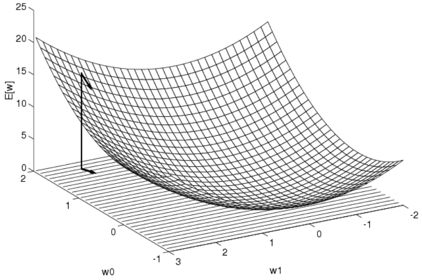
Gradient: $\nabla E[\mathbf{w}] = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_d}\right]$

Training Update Rule: $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$ where η is the training rate.

Note: For regression, others, this gradient is convex. In ANNs it is not. So we must solve iteratively

(Slides from Tom Mitchell ML Course, CMU, 2010)

Gradient Descent



Error Function: Mean Squared Error, cross-entropy loss, etc.

Gradient: $\nabla E[\mathbf{w}] = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_d}\right]$

Training Update Rule: $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$ where η is the training rate.

Note: For regression, others, this gradient is convex. In ANNs it is not. So we must solve iteratively

(Slides from Tom Mitchell ML Course, CMU, 2010)

Incremental Gradient Descent

Let error function be : $E_l[\mathbf{w}] = \frac{1}{2}(y^l - o^l)^2$
Do until satisfied:

- For each training example l in D
    1. Compute the gradient $\bigtriangledown E[\mathbf{w}]$
    2. update weights : $\mathbf{w} = \mathbf{w} - \eta \bigtriangledown E[\mathbf{w}]$

Note: can also use batch gradient descent on many points at once.

Backpropagation Algorithm

We need an iterative algorithm for getting the gradient efficiently.
For each training example:

1. Forward propagation: Input the training example to the network and compute outputs
2. Compute output units errors:
$$\delta_k^l = o_k^l(1 - o_k^l)(y_k^l - o_k^l)$$
3. Compute hidden units errors:
$$\delta_h^l = o_h^l(1 - o_h^l)\sum_k w_{h,k}\delta_k^l$$
4. Update network weights:
$$w_{i,j} = w_{i,j} + \Delta w_{i,j}^l = w_{i,j} + \eta\delta_j^l o_i^l$$

More about Backpropagation

- Still underlies all of Deep Learning
- Can be easily peformed on mulidimensional outputs.
- No guarantee of global optimal solution, only local, but often good enough.
- Can be paralellized very well, most deep learning tools now do this on GPUs automatically.
- Remember that it is minimizing errors on *training* examples, ability to generalize relies on network structure, training approach, other factors.

Outline

Problems with ANNs

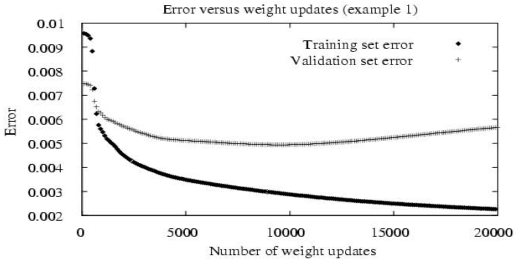
- Overfitting
- Very inneficient for images, timeseries, large numbers of inputs-outputs
- Overfitting
- Slow to train
- Hard to interpret the resulting model
- Overfitting

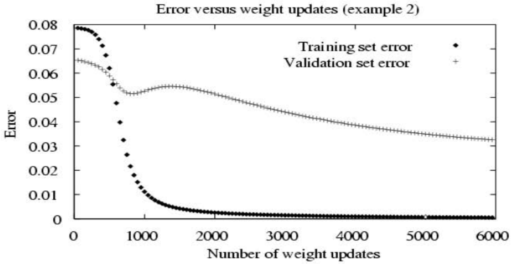
Heuristics for Improving Backpropagation

There are a number of useful heuristics for training Neural Networks that are useful in practice (maybe we'll learn more today):

- Less hidden nodes, just enough complexity to work, not too much to overfit.
- Train multiple networks with different sizes and search for the best design.
- Validation set: train on training set until error on validation set starts to rise, then evaluate on evaluation set.
- Try different activiation functions: tanh, ReLU, ELU, ...?
- Dropout (Hinton 2014) - randomly ignore certain units during training, don't update them via gradient descent, leads to hidden units that specialize
- Modify learning rate over time (cooling schedule)

The Threat of Overfitting





Regularization to Reduce Overfitting

- L2 regularization - augmenting the error function, add $\frac{1}{2}\lambda w^2$ to all weights in the neural network
  - Need to choose *lambda* carefully
  - Interpretation - heavily penalizing "peaky" weight vectors and preferring diffuse weight vectors.
  - Encourages network to use all its inputs instead of relying on strongest signal.
  - Also called **weight decay** since weights would reduce over time during gradient descent without input.
- L1 regularization - add $\lambda|w|$ to each weight error function (encourages all weights to zero)

Regularization to Reduce Overfitting

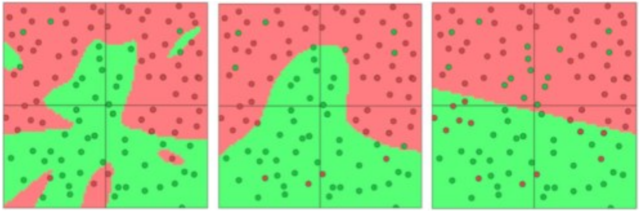
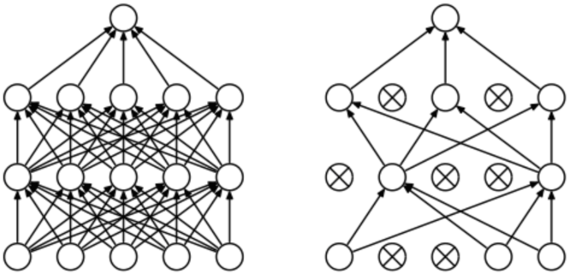


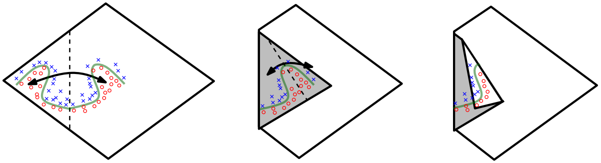
Figure: From http://www.kdnuggets.com/2015/04/preventing-overfitting-neural-networks.html/2

Dropout

- Dropout (Hinton 2014) - randomly ignore certain units during training, don't update them via gradient descent, leads to hidden units that specialize.
- With probability p don't include a weight in the gradient updates.
- Reduces overfitting by encouraging robustness of weights in the network.


Standard Neural Net After applying dropout.

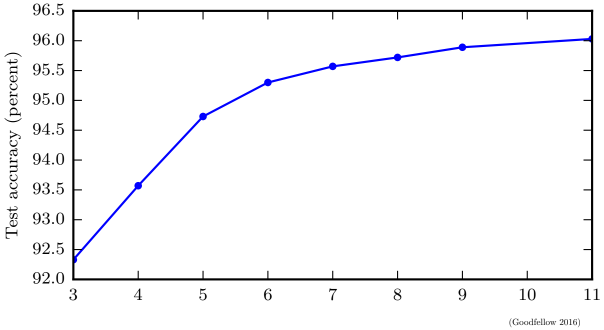
Advantage of Increased Depth



(Goodfellow 2016)

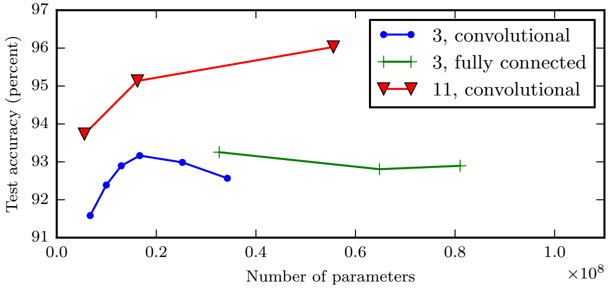
Using Hidden LeRU units, each hidden layer increases power, exponential advantage of additional layers.

Better Generalization with Greater Depth



Increasing accuracy from number of layers in detection of numbers in photos of adresses.

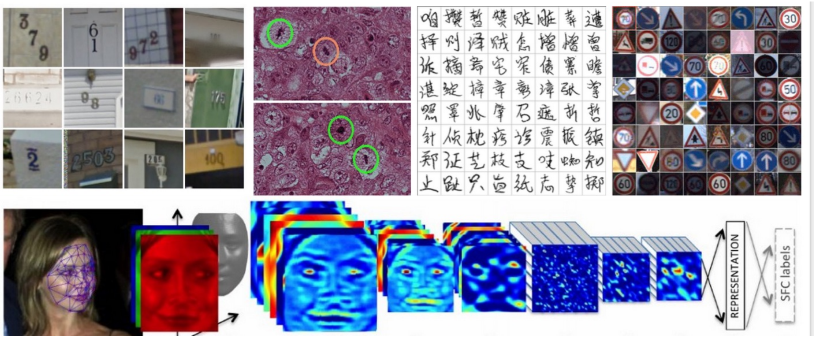
Large, Shallow Models Overfit More



(Goodfellow 2016)

Outline

Example Applications of CNNs



(Karpathy Blog, Oct, 25, 2015 - http:karpathy.github.io20151025selfie)

CNN History

After deep networks, ReLU's the third major innovation are Convolutional Neural Networks (CNNs).

- Work by Yann LeCun in 80s)
- **LeNet 1** (1993) for digit recognition
- **AlexNet** (2012) for ImageNet challenge - beat competition by error rate of 16% vs 26% for next best
- Cat videos (2012) Quoc Le - the Google cat youtube paper at ICML.
- **ZFNet** (2013), `GoogLeNet` (2014), ...

Benefits of CNNs

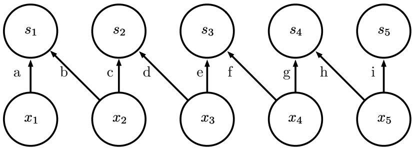
- **Simple** - repeated operation across each image - essentially a visual filter applied at every location.
  - Related to kernals discussed earlier, but automatically learns the right kernal mapping.
- **Efficient** - for images compared to a naive ANN approach.
  - Still takes a long time to train, but uses structure of images to reduce duplicated work.
- **Effective** - Computer vision applications are now mostly using some form of CNNs, but mostly for *low risk tasks*.
- **Natural?** - possible connection to animal visual system

What Makes a Deep Neural Network a CNN?

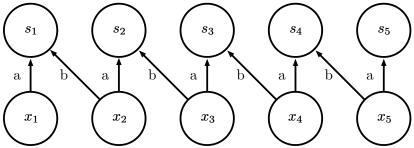
Essential Property: a neural network that uses convolution in place of general matrix multiplaction in at least one of its layers.
Additional properties:

- Sparse connectivity
- Parameter sharing (tied weights)
- Equivariant Representations

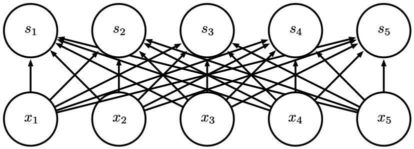
Kinds of Network Connectivity



Local connection



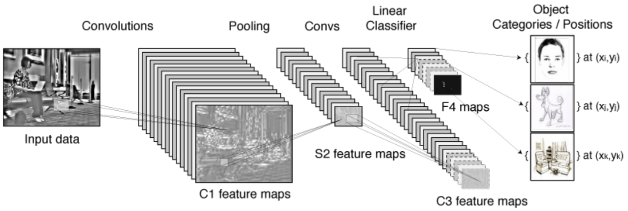
Convolution



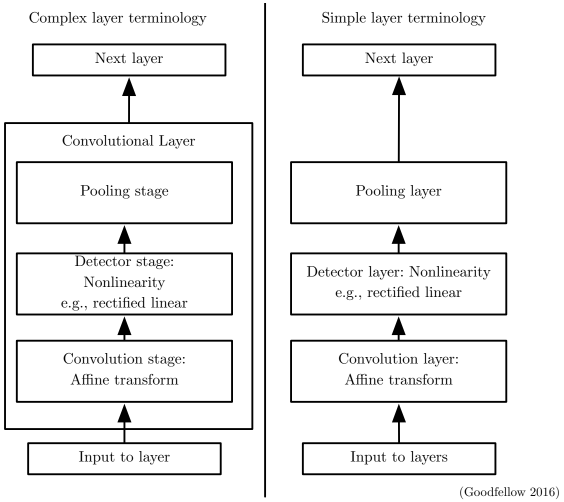
Fully connected

Convolutional Network Structure

- input data: image (eg. 256x256 pixels x3 channels RGB)



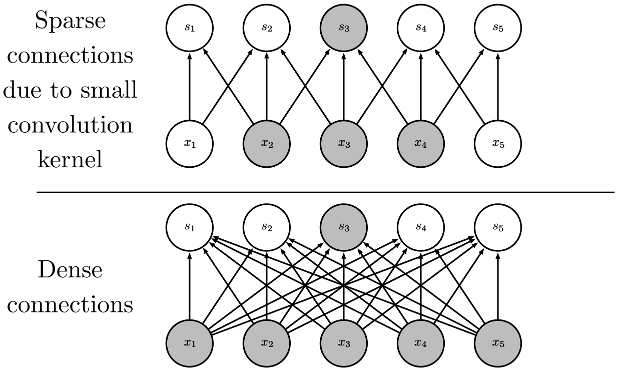
Convolutional Network Structure


Complex layer terminology
Next layer
Convolutional Layer
Pooling stage
Detector stage: Nonlinearity e.g., rectified linear
Convolution stage: Affine transform
Input to layer
Simple layer terminology
Next layer
Pooling layer
Detector layer: Nonlinearity e.g., rectified linear
Convolution layer: Affine transform
Input to layers

(Goodfellow 2016)

Sparse connectivity

- ANN: matrix multiplication for parameters mapping all inputs to all outputs.
- CNN: sparse weights, many input pixels (millions), small number of outputs (hundreds, thousands).



(Goodfellow 2016)

Convolution

A filter (kernal/tensor) is defined and passed across all the pixels, providing a response for each pixel that provides a weighted average over nearby pixels.

$$s(t) = \int x(a)w(t-a)da$$

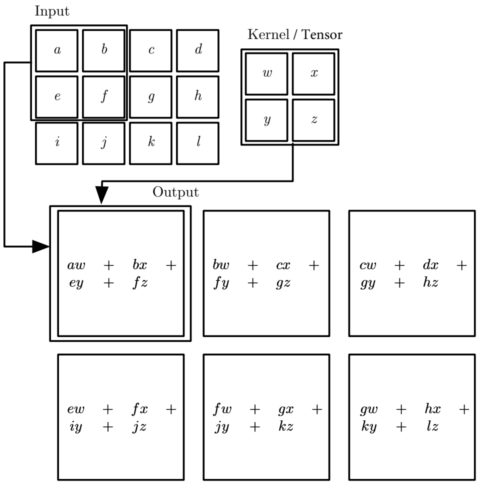
$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n)$$

where

- a - age of measurement in a timeseries
- I - 2D input image
- K - 2D kernal

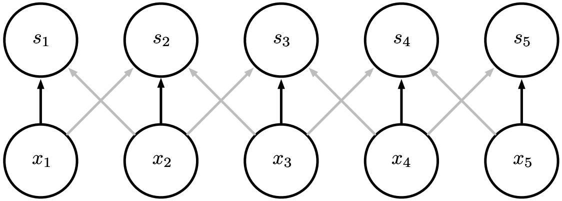
2D Convolution



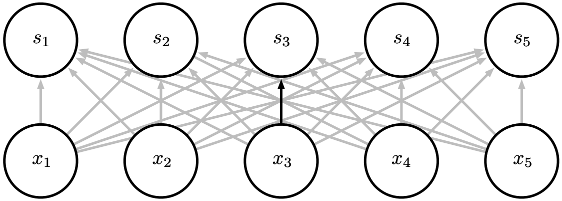
(Goodfellow 2016)

Parameter sharing

Convolution shares the same parameters across all spatial locations



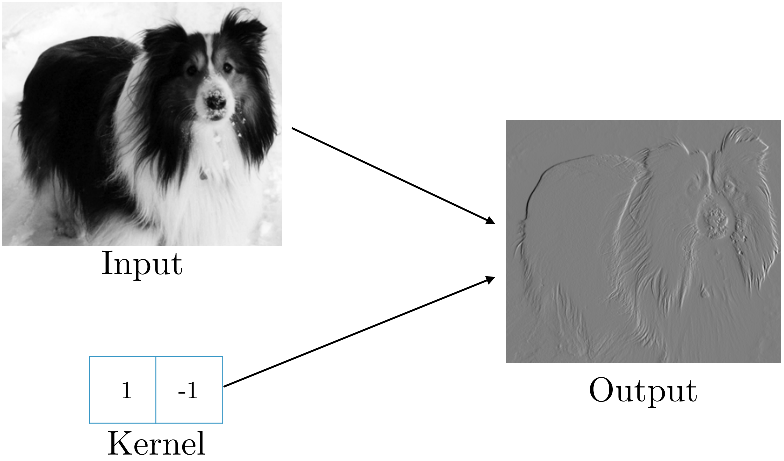
Traditional matrix multiplication does not share any parameters



(Goodfellow 2016)

A simple example

Edge detection by convolution with a kernal that subtracts the value from the neighbouring pixel on the left for every pixel.



(Goodfellow 2016)

Efficiency of Convolution

Input size: 320 by 280
Kernel size: 2 by 1
Output size: 319 by 280

	Convolution	**Dense matrix**	**Sparse matrix**
Stored floats	2	319*280*320*280 $>$ 8e9	2*319*280 = 178,640
Float muls or adds	319*280*3 = 267,960	$>$ 16e9	Same as convolution (267,960)

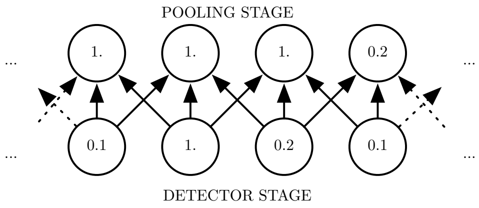
(Goodfellow 2016)

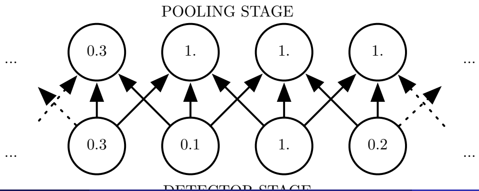
Max Pooling

- problem: Still too much density and repetition in the network.
- **observation:** Nearby pixels tend to represent the same thing/class/object.
- solution: Combine, or “pool”, responses from nearby nodes.
- Could use `average`, `median`, `minimum`, but `maximum` often works well.

Max Pooling and Invariance to Translation

Pooling refers to aproximating the outputs of a layer by aggregating nearby values.
This improves *invariance to small translations* in the input.





Cross-Channel Pooling and Invariance to Learned Transformations

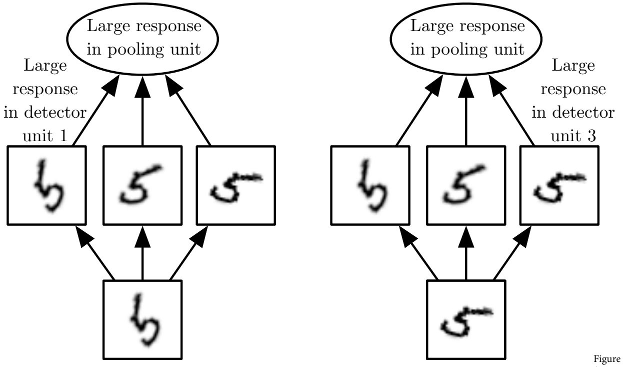
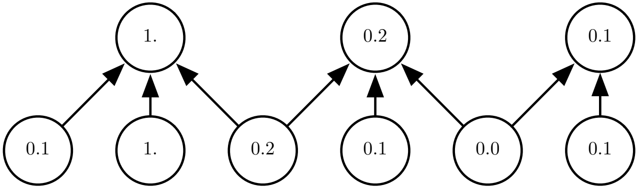

Large response in pooling unit
Large response in detector unit 1
Large response in pooling unit
Large response in detector unit 3

Figure 9.9
(Goodfellow 2016)

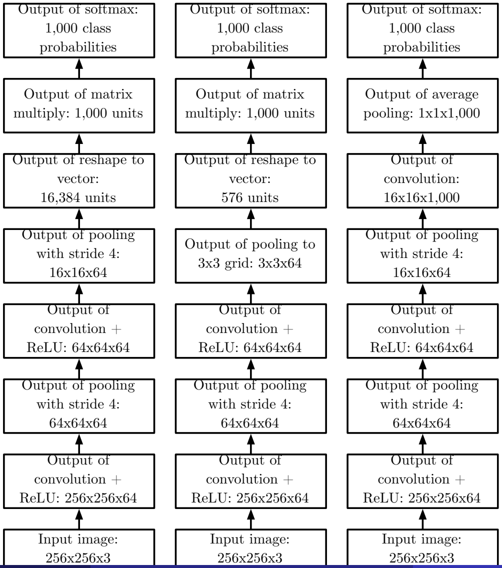
Pooling with Downsampling

We can also use pooling to reduce the size of each layer to create a hierarchical model.

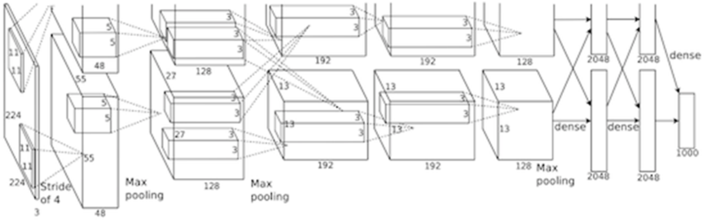


(Goodfellow 2016)

CNN Architecture Examples


Output of softmax: 1,000 class probabilities
Output of matrix multiply: 1,000 units
Output of reshape to vector: 16,384 units
Output of pooling with stride 4: 16x16x64
Output of convolution + ReLU: 64x64x64
Output of pooling with stride 4: 64x64x64
Output of convolution + ReLU: 256x256x64
Input image: 256x256x3
Output of softmax: 1,000 class probabilities
Output of matrix multiply: 1,000 units
Output of reshape to vector: 576 units
Output of pooling to 3x3 grid: 3x3x64
Output of convolution + ReLU: 64x64x64
Output of pooling with stride 4: 64x64x64
Output of convolution + ReLU: 256x256x64
Input image: 256x256x3
Output of softmax: 1,000 class probabilities
Output of average pooling: 1x1x1,000
Output of convolution: 16x16x1,000
Output of pooling with stride 4: 16x16x64
Output of convolution + ReLU: 64x64x64
Output of pooling with stride 4: 64x64x64
Output of convolution + ReLU: 256x256x64
Input image: 256x256x3

AlexNet

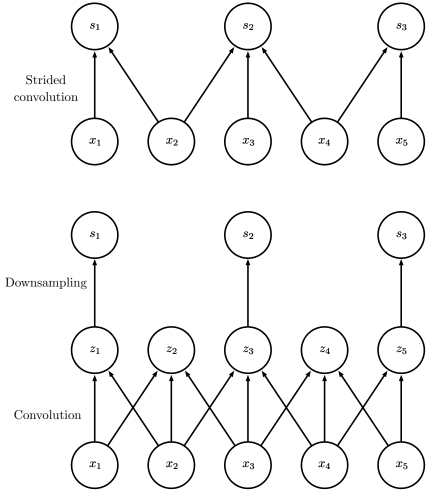


AlexNet architecture (May look weird because there are two different "streams". This is because the training process was so computationally expensive that they had to split the training onto 2 GPUs)

Other CNN Modification

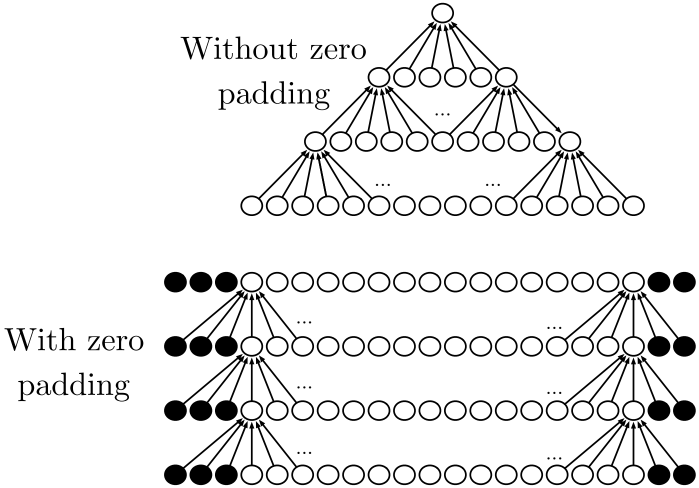
- Strides - number of pixels overlap between adjacent filters
- Zero padding - removing edge pixels from filter scan, can reduce size of network and deal with edge effects
- Alternate local connectivity options
- Partial connectivity between channels

Convolutions with Strides



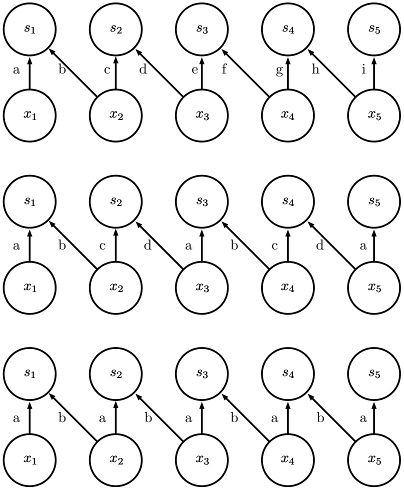
(Goodfellow 2016)

Effect of Zero Padding on Network Size



(Goodfellow 2016)

Tiled Convolution



Local connection (no sharing)

Tiled convolution (cycle between groups of shared parameters)

Convolution (one group shared everywhere)

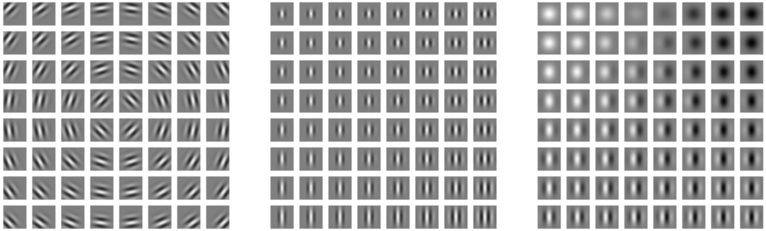
(Goodfellow 2016)

CNNs Learn Meaningful Filters

When trained on photographic images the filters learned by CNNs often correspond to prior image processing constructs or to identifiable real world patterns:

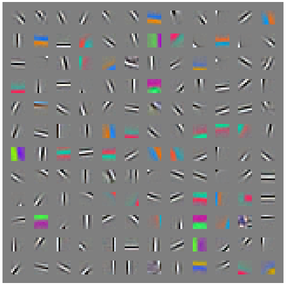
- low level stages learn filters like lines, curves, directional vectors. Similar to Gabor filters from animal visual systems.
- Unsupervised training - top layers can correspond to clusters of images

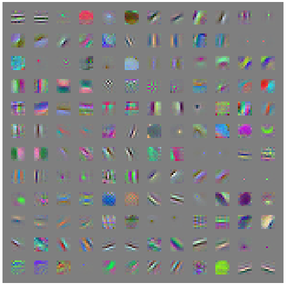
Gabor Functions



(Goodfellow 2016)

Gabor Like Learned Kernals Learned via CNNs





Unsupervised Training of CNNs

- Cat videos (2012) Quoc Le - the Google cat youtube paper at ICML.

Major Deep CNN Architectures

Spatial Transducer Net:	input size scales with output size, all layers are convolutional
All Convolutional Net:	no pooling layers, just use strided convolution to shrink representation size
Inception:	complex architecture designed to achieve high accuracy with low computational cost
ResNet:	blocks of layers with same spatial size, with each layers output added to the same buffer that is repeatedly updated. Very many updates = very deep net (over 150 layers!) but without vanishing gradient.

(Goodfellow, 2016)

Other Types of Deep Neural Networks

RBM:	Restricted Boltzman Machines (RBM) - older directed deep model.
RNN:	Recurrent Neural Networks (RNN) - allow links from outputs back to inputs, over time, good for time series learning
LSTM:	Long-Term Short-Term networks - more complex form of RNN
DeepRL:	Deep Reinforcement Learning
GAN:	General Adversarial Networks - train two networks at once

Other Types of Deep Neural Networks

RBM: Restricted Boltzman Machines (RBM) - older directed deep model.

RNN: Recurrent Neural Networks (RNN) - allow links from outputs back to inputs, over time, good for time series learning

LSTM: Long-Term Short-Term networks - more complex form of RNN

- integrate strategically remembering particular information from the past
- formalizes a process for *forgetting* information over time.

DeepRL: Deep Reinforcement Learning

GAN: General Adversarial Networks - train two networks at once

Other Types of Deep Neural Networks

RBM: Restricted Boltzman Machines (RBM) - older directed deep model.

RNN: Recurrent Neural Networks (RNN) - allow links from outputs back to inputs, over time, good for time series learning

LSTM: Long-Term Short-Term networks - more complex form of RNN

DeepRL: Deep Reinforcement Learning

- CNNs + Fully Connected Deep Network for learning a representation of a policy
- Reinforcement Learning for updating the policy through experience to make improved decision decisions
- Requires a value/reward function

GAN: General Adversarial Networks - train two networks at once

Other Types of Deep Neural Networks

RBM:	Restricted Boltzman Machines (RBM) - older directed deep model.
RNN:	Recurrent Neural Networks (RNN) - allow links from outputs back to inputs, over time, good for time series learning
LSTM:	Long-Term Short-Term networks - more complex form of RNN
DeepRL:	Deep Reinforcement Learning
GAN:	General Adversarial Networks - train two networks at once

General Adversarial Networks

- one network produces/hallucinates new answers (generative)
- second network distinguishes between the real and the generated answers (adversary/critic)
- similar in idea to Deep RL but does not require a value/reward function.
- could be used in combination with any other type: CNN, LSTM, RNN, ...
- Blog/Code: "GANS in 50 lines of code PyTorch code." easy way to get started

Summary

- Machine Learning
  - Learning models from data for prediction, **classification**, clustering and anomaly detection
  - Many models are linear, kernal methods allow mapping from low dimensional nonlinear to highdimensional linear, but have limitations.
- Artificial Neural Networks
  - Ideas have been around a long time.
  - Limitations: overfit easily, slow to train, not great for images
- Making Neural Networks Work
  - Regularization : Dropout
  - ReLU activation functions
  - More layers, more structure
- Convolutional Neural Networks
  - Basic use of convolutional filters
  - Many network architectures

Useful Books

All three ages of Machine Learning:

[Goodfellow, 2016]
Goodfellow, Bengio and Courville. *"Deep Learning"*, MIT Press, 2016.

- http://www.deeplearningbook.org/
- Website has free copy of book as pdf's.

[Murphy, 2012]
Kevin Murphy, *Machine Learning: A Probabilistic Perspective*, MIT Press, 2012.

[Duda, Pattern Classification, 2001]
R. O. Duda, P. E. Hart and D. G. Stork, *Pattern Classification (2nd ed.)*, John Wiley and Sons, 2001.

Useful Papers and Blogs

[lecun2015]
Y. LeCun, Y. Bengio, G. Hinton, L. Y., B. Y., and H. G., "Deep learning", *Nature*, vol. 521, no. 7553, pp. 436444, 2015. Great references at back with comments on seminal papers.

[bengio2009]
Y. Bengio, "Learning Deep Architectures for AI", *Foundations and Trends in Machine Learning*, vol. 2, no. 1. 2009. An earlier general referenceon the fundamentals of Deep Learning.

[krizhevsky2012]
A. Krizhevsky, G. E. Hinton, and I. Sutskever, "ImageNet Classification with Deep Convolutional Neural Networks", *Adv. Neural Inf. Process. Syst.* pp. 19, 2012. The beginning of the current craze.

[Karpathy, 2015]
Andrej Karpathy's Blog - `http://karpathy.github.io` Easy to follow explanations with code