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

Introduction

Machine Learning

Artificial Neural Networks

Making Neural Networks Work

Convolutional Neural Networks

Summary
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*?
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 you're 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

Introduction

Machine Learning
- Landscape of Big Data/AI/ML
- Classification
- Linear Classifiers
- Beyond Linear Classifiers

Artificial Neural Networks

Making Neural Networks Work

Convolutional Neural Networks
Data, Big Data, Machine Learning, AI, etc, etc,

<table>
<thead>
<tr>
<th>Data</th>
<th>Big Data Tools</th>
<th>Machine Learning</th>
<th>Artificial Intelligence</th>
</tr>
</thead>
</table>

Data Analysis

Reports, statistics, Charts, trends

Policies, Decision Rules, Summaries

Classification, Patterns, Predictions Probabilities

Human Decision Making

- $X \ f_1 \ f_2 \ ... \ f_k$
- $x_1 \ .5 \ 104.2 \ China$
- $x_2 \ .34 \ 92.0 \ USA$
- $\ldots$
- $x_n \ .2 \ 85.2 \ Canada$
Data, Big Data, Machine Learning, AI, etc, etc,

<table>
<thead>
<tr>
<th>Data</th>
<th>Data Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>f₁  f₂  ...  fₖ</td>
</tr>
<tr>
<td>X₁</td>
<td>0.5  104.2  China</td>
</tr>
<tr>
<td>X₂</td>
<td>0.34 92.0  USA</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Xₙ</td>
<td>0.2  85.2  Canada</td>
</tr>
</tbody>
</table>

- Reports, statistics, Charts, trends
- Classification, Patterns, Predictions, Probabilities
- Policies, Decision Rules, Summaries
- Human Decision Making
- Automated Decision Making
- Machine Learning
- Artificial Intelligence
- Big Data Tools
- Multi-agent systems
- Game Theory
- Natural Language Processing
- Robotics
- Vision
- CNN
- LSTM
- RNN
- DQN
- ILP
- Constraint Programming
- SAT
- SMP
- A3C
- Reinforcement Learning
- Games
- Deep Learning
- Probabilistic Programming
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 Regression, 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 classifier 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 $X = \{x_1, x_2, \ldots, x_n\}$ where each datapoint $x_i \in X$ contains $F$ features denoted $x_i,f$,
- and given a set of classes $C = \{C_1, \ldots, C_K\}$,
- 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

Introduction

Machine Learning
- Landscape of Big Data/AI/ML
- Classification
- Linear Classifiers
- Beyond Linear Classifiers

Artificial Neural Networks

Making Neural Networks Work

Convolutional Neural Networks
Linear Regression vs. Logistic Regression

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

$$Y = b_0 + \sum (b_iX_i) + \epsilon$$

- 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.

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

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

\[ o(x) = \sigma(w^T x_i) = \sigma(w_0 + \sum_i w_ix_i) = \frac{1}{1 + \exp(-(w_0 + \sum_i w_ix_i))} \]
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-likelihood of the logistic function is given by $NLL$ and its 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 $\eta_k$ 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 positive training points are above the line.
- More generally we can learn a nonlinear function using the kernel 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
- $\alpha$ is a vector of coefficients
Support Vector Machines

- Optimal separation:
  1. No errors.
  2. The distance from the closest vector to the hyper plane is maximized.

- The vectors closest to the hyper plane are called support vectors.
Support Vector Machines (SVMs)

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

We rewrite the linear system as a dot product:

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

\[
f(\mathbf{x}) = b + \sum_i \alpha_i k(\mathbf{x}, \mathbf{x}^{(i)})
\]

where

- \(\mathbf{x}^{(i)}\) is a training example
- \(\alpha\) is a vector of coefficients
- \(k(\mathbf{x}, \mathbf{x}^{(i)}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{x})^T\) is a kernel.
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 $\alpha_i k(xx^{(i)})$
  - Kernal SVMs avoid this problem because most of $\alpha_i = 0$, only the support vectors $x^{(i)}$ 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

Introduction

Machine Learning

Artificial Neural Networks
- History and Overview
- Classic Neural Network Basics
- Neural Networks as Universal Approximators
- Training Neural Networks - Backpropagation

Making Neural Networks Work

Convolutional Neural Networks
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"
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! Kernels 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 Boltzmann 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 Boltzmann 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 Boltzmann 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 Boltzmann 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 Boltzmann 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, ElementAl, …)
  - 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

Input layer

Hidden layer

Output layer
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 an 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.

\[ y_j = f(\text{net}_j) \]

- 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

- **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 always easy to distinguish
  - most values are zero, derivative 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
The output layer adds another level of thresholding, usually nonlinear as well.

\[ z_k = f(\text{net}_k) \]

\( f(\text{net}_k) \) could be any thresholding function such as the previous activation functions.
Outline

Introduction

Machine Learning

Artificial Neural Networks
- History and Overview
- Classic Neural Network Basics
- Neural Networks as Universal Approximators
- Training Neural Networks - Backpropagation

Making Neural Networks Work

Convolutional Neural Networks
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.

(Left) A linear model applied directly to the original input cannot implement the XOR function. When $x_1 = 0$, the model’s output must increase as $x_2$ increases. When $x_1 = 1$, the model’s output must decrease as $x_2$ increases. A linear model must apply a fixed coefficient $w_2$ to $x_2$. The linear model therefore cannot use the value of $x_1$ to change the coefficient on $x_2$ and cannot solve this problem.

(Right) In the transformed space represented by the features extracted by a neural network, a linear model can now solve the problem. In our example solution, the two points that must have output 1 have been collapsed into a single point in feature space. In other words, the nonlinear features have mapped both $x = [1, 0]^T$ and $x = [0, 1]^T$ to a single point in feature space, $h = [1, 0]^T$. The linear model can now describe the function as increasing in $h_1$ and decreasing in $h_2$.

In this example, the motivation for learning the feature space is only to make the model capacity greater so that it can fit the training set. In more realistic applications, learned representations can also help the model to generalize.
Neural Nets Can Model XOR Problem

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

\[ f(\text{net}) = \begin{cases} 1 & \text{if } \text{net} \geq 0 \\ -1 & \text{if } \text{net} < 0 \end{cases} \]

- **First hidden unit:**
  - \( y_1 \) computes \( x_1 + x_2 + 0.5 = 0 \)
  - \( y_1 = 1 \) if \( \text{net}_1 \geq 0 \), \( y_1 = -1 \) otherwise

- **Second hidden unit:**
  - \( y_1 \) computes \( x_1 + x_2 - 1.5 = 0 \)
  - \( y_2 = 1 \) if \( \text{net}_2 \geq 0 \), \( y_2 = -1 \) 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 Discriminant Function

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

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

- 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 units 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.
Gradient Descent

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

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

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

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

Training Update Rule: $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$ where $\eta$ 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: \( \nabla E[w] = \left[ \frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_d} \right] \)

Training Update Rule: \( \Delta w_i = -\eta \frac{\partial E}{\partial w_i} \) where \( \eta \) 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: \( E_l[w] = \frac{1}{2}(y^l - o^l)^2 \)

Do until satisfied:

- For each training example \( l \) in \( D \)
  1. Compute the gradient \( \nabla E[w] \)
  2. update weights: \( w = w - \eta \nabla E[w] \)

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**:  
   \[ \delta^l_k = o^l_k(1 - o^l_k)(y^l_k - o^l_k) \]
3. **Compute hidden units errors**:  
   \[ \delta^l_h = o^l_h(1 - o^l_h) \sum_k w_{h,k}\delta^l_k \]
4. **Update network weights**:  
   \[ w_{i,j} = w_{i,j} + \Delta w_{i,j} = w_{i,j} + \eta \delta^l_j o^l_i \]
More about Backpropagation

- Still underlies all of Deep Learning
- Can be easily performed on multidimensional outputs.
- No guarantee of global optimal solution, only local, but often good enough.
- Can be parallelized 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

Introduction

Machine Learning

Artificial Neural Networks

Making Neural Networks Work

- Improving Performance
- The Threat of Overfitting
- Regularization
- Going Deep

Convolutional Neural Networks
Problems with ANNs

- Overfitting
- Very inefficient 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 activation 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

**Error versus weight updates (example 1)**

- **Training set error**: 
- **Validation set error**

**Error versus weight updates (example 2)**

- **Training set error**: 
- **Validation set error**
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.
Advantage of Increased Depth

Using Hidden LeRU units, each hidden layer increases power, exponential advantage of additional layers.
Better Generalization with Greater Depth

**Figure 6.6**: Empirical results showing that deeper networks generalize better when used to transcribe multi-digit numbers from photographs of addresses. Data from Goodfellow et al. (2014d). The test set accuracy consistently increases with increasing depth. See figure 6.7 for a control experiment demonstrating that other increases to the model size do not yield the same effect.

Another key consideration of architecture design is exactly how to connect a pair of layers to each other. In the default neural network layer described by a linear transformation via a matrix $W$, every input unit is connected to every output unit. Many specialized networks in the chapters ahead have fewer connections, so that each unit in the input layer is connected to only a small subset of units in the output layer. These strategies for reducing the number of connections reduce the number of parameters and the amount of computation required to evaluate the network, but are often highly problem-dependent. For example, convolutional networks, described in chapter 9, use specialized patterns of sparse connections that are very effective for computer vision problems. In this chapter, it is difficult to give much more specific advice concerning the architecture of a generic neural network. Subsequent chapters develop the particular architectural strategies that have been found to work well for different application domains.
Large, Shallow Models Overfit More

Figure 6.7: Deeper models tend to perform better. This is not merely because the model is larger. This experiment from Goodfellow et al. (2016d) shows that increasing the number of parameters in layers of convolutional networks without increasing their depth is not nearly as effective at increasing test set performance. The legend indicates the depth of network used to make each curve and whether the curve represents variation in the size of the convolutional or the fully connected layers. We observe that shallow models in this context overfit at around 20 million parameters while deep ones can benefit from having over 60 million. This suggests that using a deep model expresses a useful preference over the space of functions the model can learn. Specifically, it expresses a belief that the function should consist of many simpler functions composed together. This could result either in learning a representation that is composed in turn of simpler representations (e.g., corners defined in terms of edges) or in learning a program with sequentially dependent steps (e.g., first locate a set of objects, then segment them from each other, then recognize them).
Outline

Introduction

Machine Learning

Artificial Neural Networks

Making Neural Networks Work

Convolutional Neural Networks
- Convolution Operation
- Pooling
- Examples
- Other CNN Modifications
- Interpreting CNN Models
- Major Architectures
Example Applications of CNNs

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 multiplication 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 Neural Networks

Convolutional Network Structure

- input data: image (e.g. 256x256 pixels x3 channels RGB)
Convolutional Neural Networks

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).

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

![Sparse connections due to small convolution kernel](image)

![Dense connections](image)

(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.

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

\[
s(t) = (x \ast w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t - a)
\]

\[
S(i, j) = (I \ast K)(i, j) = \sum_{m} \sum_{n} I(m, n)K(i - m, j - n)
\]

where

- \(a\) - age of measurement in a timeseries
- \(I\) - 2D input image
- \(K\) - 2D kernal
2D Convolution

Figure 9.1: An example of 2-D convolution without kernel-flipping. In this case we restrict the output to only positions where the kernel lies entirely within the image, called "valid" convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor.
Parameter sharing

Convolution shares the same parameters across all spatial locations.

Traditional matrix multiplication does not share any parameters.

(Convolutional Neural Networks)
A simple example

Edge detection by convolution with a kernel that subtracts the value from the neighbouring pixel on the left for every pixel.
### Efficiency of Convolution

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

<table>
<thead>
<tr>
<th></th>
<th>Convolution</th>
<th>Dense matrix</th>
<th>Sparse matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stored floats</td>
<td>2</td>
<td>319<em>280</em>320*280</td>
<td>2<em>319</em>280 = 178,640</td>
</tr>
<tr>
<td>Float muls or adds</td>
<td>319<em>280</em>3 = 267,960</td>
<td>&gt; 16e9</td>
<td>Same as convolution (267,960)</td>
</tr>
</tbody>
</table>

(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 approximating 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

Figure 9.9: Example of learned invariances: Apooling unit over multiple features that are learned with separate parameters can learn to be invariant to transformations of the input. Here we show how a set of three learned filters and a max pooling unit can learn to become invariant to rotation. All three filters are intended to detect a hand-written 5. Each filter attempts to match a slightly different orientation of the 5. When a 5 appears in the input, the corresponding filter will match it and cause a large activation in a detector unit. The max pooling unit then has a large activation regardless of which detector unit was activated. We show here how the network processes two different inputs, resulting in two different detector units being activated. The effect on the pooling unit is roughly the same either way. This principle is leveraged by maxout networks (Goodfellow et al., 2013a) and other convolutional networks. Max pooling over spatial positions is naturally invariant to translation; this multi-channel approach is only necessary for learning other transformations.

Figure 9.10: Pooling with downsampling. Here we use max-pooling with a pool width of three and a stride between pools of two. This reduces the representation size by a factor of two, which reduces the computational and statistical burden on the next layer. Note that the rightmost pooling region has a smaller size, but must be included if we do not want to ignore some of the detector units.
We can also use pooling to reduce the size of each layer to create a hierarchical model.
CNN Architecture Examples

Input image: 256x256x3

Output of convolution + ReLU: 256x256x64

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: 16,384 units

Output of average pooling: 1x1x1,000

Output of convolution: 16x16x1,000

Output of pooling with stride 4: 16x16x64

Output of pooling to 3x3 grid: 3x3x64

Output of reshaping to vector: 576 units

Output of matrix multiply: 1,000 units

Output of softmax: 1,000 class probabilities

Figure 9.11: Examples of architectures for classification with convolutional networks. The specific strides and depths used in this figure are not advisable for real use; they are designed to be very shallow in order to fit onto the page. Real convolutional networks also often involve significant amounts of branching, unlike the chain structures used here for simplicity.

(Left) A convolutional network that processes a fixed image size. After alternating between convolution and pooling for a few layers, the tensor for the convolutional feature map is reshaped to flatten out the spatial dimensions. The rest of the network is an ordinary feedforward network classifier, as described in chapter 6.

(Center) A convolutional network that processes a variable-size image, but still maintains a fully connected section. This network uses a pooling operation with variably-sized pools but a fixed number of pools, in order to provide a fixed-size vector of 576 units to the fully connected portion of the network.

(Right) A convolutional network that does not have any fully connected weight layer. Instead, the last convolutional layer outputs one feature map per class. The model presumably learns a map of how likely each class is to occur at each spatial location. Averaging a feature map down to a single value provides the argument to the softmax classifier at the top.
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 Modifications

- 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

Strided convolution

Downsampling

Convolution

(Goodfellow 2016)
Effect of Zero Padding on Network Size

Without zero padding:

With zero padding:

(Goodfellow 2016)
Tiled Convolution

Local connection
(no sharing)

Tiled convolution
(cycle between groups of shared parameters)

Convolution
(one group shared everywhere)

Figure 9.16: A comparison of locally connected layers, tiled convolution, and standard convolution. All three have the same sets of connections between units, when the same size of kernel is used. This diagram illustrates the use of a kernel that is two pixels wide. The differences between the methods lies in how they share parameters.

(Top) Locally connected layer has no sharing at all. We indicate that each connection has its own weight by labeling each connection with a unique letter.

(Center) Tiled convolution has a set of different kernels. Here we illustrate the case of $t = 2$. One of these kernels has edges labeled "a" and "b," while the other has edges labeled "c" and "d." Each time we move one pixel to the right in the output, we move on to using a different kernel. This means that, unlike the locally connected layer, neighboring units in the output have different parameters. Unlike the locally connected layer, after we have gone through all available kernels, we cycle back to the first kernel. If two output units are separated by a multiple of $t$ steps, then they share parameters.

(Bottom) Traditional convolution is equivalent to tiled convolution with $t = 1$. There is only one kernel and it is applied everywhere, as indicated in the diagram by using the kernel with weights labeled "a" and "b" 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.
Figure 9.18: Gabor functions with a variety of parameter settings. White indicates large positive weight, black indicates large negative weight, and the background gray corresponds to zero weight.

- **Left:** Gabor functions with different values of the parameters that control the coordinate system: $x_0$, $y_0$, and $\beta$. Each Gabor function in this grid is assigned a value of $x_0$ and $y_0$ proportional to its position in its grid, and $\beta$ is chosen so that each Gabor filter is sensitive to the direction radiating out from the center of the grid.

- **Center:** Gabor functions with different Gaussian scale parameters $x$ and $y$. Gabor functions are arranged in increasing width (decreasing $x$) as we move left to right through the grid, and increasing height (decreasing $y$) as we move top to bottom. For the other two plots, the values are fixed to $1.5 \times$ the image width.

- **Right:** Gabor functions with different sinusoid parameters $f$ and $\alpha$. As we move top to bottom, $f$ increases, and as we move left to right, $\alpha$ increases. For the other two plots, $\alpha$ is fixed to 0 and $f$ is fixed to $5 \times$ the image width.

Some of the most striking correspondences between neuroscience and machine learning come from visually comparing the features learned by machine learning models with those employed by V1. Olshausen and Field (1996) showed that a simple unsupervised learning algorithm, sparse coding, learns features with receptive fields similar to those of simple cells. Since then, we have found that an extremely wide variety of statistical learning algorithms learn features with Gabor-like functions when applied to natural images. This includes most deep learning algorithms, which learn these features in their first layer. Figure 9.19 shows some examples. Because so many different learning algorithms learn edge detectors, it is difficult to conclude that any specific learning algorithm is the "right" model of the brain just based on the features that it learns (though it can certainly be a bad sign if an algorithm does not learn some sort of edge detector when applied to natural images). These features are an important part of the statistical structure of natural images and can be recovered by many different approaches to statistical modeling. See Hyvärinen et al. (2009) for a review of the field of natural image statistics.
Gabor Like Learned Kernels Learned via CNNs

Figure 9.19: Many machine learning algorithms learn features that detect edges or specific colors of edges when applied to natural images. These feature detectors are reminiscent of the Gabor functions known to be present in primary visual cortex. (Left) Weights learned by an unsupervised learning algorithm (spike and slab sparse coding) applied to small image patches. (Right) Convolution kernels learned by the first layer of a fully supervised convolutional maxout network. Neighboring pairs of filters drive the same maxout unit.

9.11 Convolutional Networks and the History of Deep Learning

Convolutional networks have played an important role in the history of deep learning. They are a key example of a successful application of insights obtained by studying the brain to machine learning applications. They were also some of the first deep models to perform well, long before arbitrary deep models were considered viable. Convolutional networks were also some of the first neural networks to solve important commercial applications and remain at the forefront of commercial applications of deep learning today. For example, in the 1990s, the neural network research group at AT&T developed a convolutional network for reading checks (LeCun et al., 1998b). By the end of the 1990s, this system deployed by NEC was reading over 10% of all the checks in the US. Later, several OCR and handwriting recognition systems based on convolutional nets were deployed by Microsoft (Simard et al., 2003). See chapter 12 for more details on such applications and more modern applications of convolutional networks. See LeCun et al. (2010) for a more in-depth history of convolutional networks up to 2010.

Convolutional networks were also used to win many contests. The current intensity of commercial interest in deep learning began when Krizhevsky et al. (2012) won the ImageNet object recognition challenge, but convolutional networks...
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 Boltzmann 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 Boltzmann 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 Boltzmann 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 Boltzmann 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, kernel methods allow mapping from low dimensional nonlinear to high-dimensional 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
All three ages of Machine Learning:

[Goodfellow, 2016]
- http://www.deeplearningbook.org/
- Website has free copy of book as pdf’s.

[Murphy, 2012]

[Duda, Pattern Classification, 2001]
Useful Papers and Blogs

[lecun2015]

[bengio2009]

[krizhevsky2012]

[Karpathy, 2015]