Deep Learning & Neural Networks
Lecture 1

Kevin Duh

Graduate School of Information Science
Nara Institute of Science and Technology

Jan 14, 2014

The New York Times

Scientists See Promise in Deep-Learning Programs

Hao Zhang/The New York Times

A voice recognition program translated a speech given by Richard F. Rashid, Microsoft's top scientist, into Mandarin Chinese.

By JOHN MARKOFF
Published: November 23, 2012

FACEBOOK
TWITTER
GOOGLE+
SAVE

Using an artificial intelligence technique inspired by theories about how the brain recognizes patterns, technology companies are reporting startling gains in fields as diverse as computer vision, speech recognition and the identification of promising new molecules for designing drugs.

Google's Artificial Brain Learns to Find Cat Videos

BY WIRED UK 06.26.12 11:15 AM

Share 3.1k
Tweet 698
+1 558
Share 106
Pin it

By Liat Clark, Wired UK

When computer scientists at Google's mysterious X lab built a neural network of 16,000 computer processors with one billion connections and let it browse YouTube, it did what many web users might do — it began to look for cats.

WIRED.CO.UK

The "brain" simulation was exposed to 10 million randomly selected YouTube video thumbnails over the course of three days and, after being presented with a list of 20,000 different items, it began to recognize pictures of cats using a "deep learning" algorithm. This was despite being fed no information on

Facebook Launches Advanced AI Effort to Find Meaning in Your Posts

A technique called deep learning could help Facebook understand its users and their data better.

By Tom Simonite on September 20, 2013

Facebook is set to get an even better understanding of the 700 million people who use the social network to share details of their personal lives each day.

A new research group within the company is working on an emerging and powerful approach to artificial intelligence known as deep learning, which uses simulated networks of brain cells to process data. Applying this method to data shared on Facebook could allow for novel features and perhaps boost the company's ad targeting.

Deep learning has shown potential as the basis for software that could work out the emotions or events described in text even if they aren't explicitly referenced, recognize objects in photos, and make sophisticated predictions about people's likely future behavior.

WHY IT MATTERS

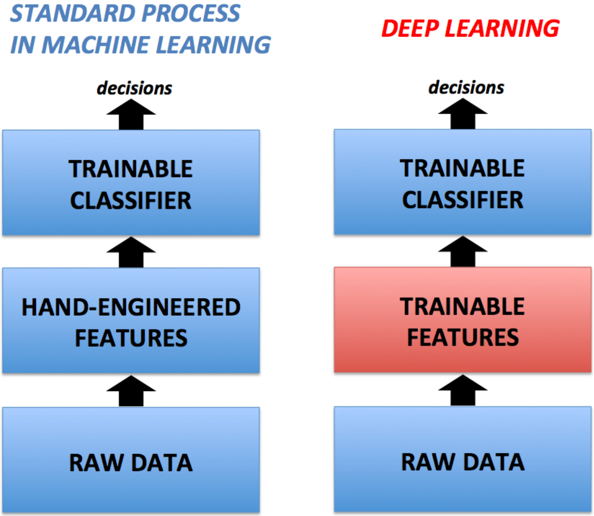
Facebook's piles of data on people's lives could allow it to push the boundaries of what can be done with the emerging AI technique

What is Deep Learning?

A family of methods that uses deep architectures to learn high-level feature representations

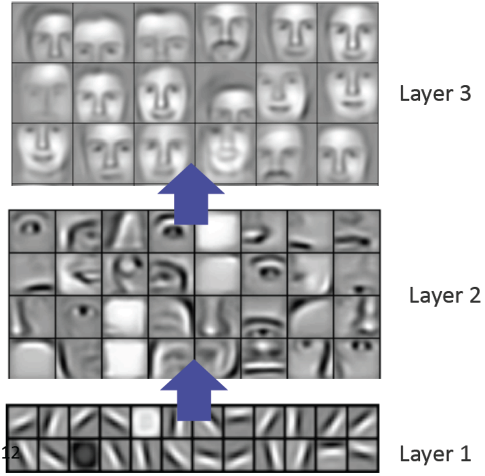
What is Deep Learning?

A family of methods that uses deep architectures to learn high-level feature representations



Example of Trainable Features

Hierarchical object-parts features in Computer Vision [Lee et al., 2009]



Course Outline

- Goal:
  To understand the foundations of neural networks and deep learning, at a level sufficient for reading recent research papers

Course Outline

- Goal:
  To understand the foundations of neural networks and deep learning, at a level sufficient for reading recent research papers
- Schedule:
  - Lecture 1 (Jan 14): Machine Learning background & Neural Networks
  - Lecture 2 (Jan 16): Deep Architectures (DBN, SAE)
  - Lecture 3 (Jan 21): Applications in Vision, Speech, Language
  - Lecture 4 (Jan 23): Advanced topics in optimization

Course Outline

- Goal:
  To understand the foundations of neural networks and deep learning, at a level sufficient for reading recent research papers
- Schedule:
  - Lecture 1 (Jan 14): Machine Learning background & Neural Networks
  - Lecture 2 (Jan 16): Deep Architectures (DBN, SAE)
  - Lecture 3 (Jan 21): Applications in Vision, Speech, Language
  - Lecture 4 (Jan 23): Advanced topics in optimization
- Prerequisites:
  - Basic calculus, probability, linear algebra

Course Material

- Course Website:
  http://cl.naist.jp/~kevinduh/a/deep2014/

- Useful References:
  1. Yoshua Bengio's [Bengio, 2009] short book: Learning Deep Architectures for AI[1]
  2. Yann LeCun & Marc'Aurelio Ranzato's ICML2013 tutorial[2]
  3. Richard Socher et. al.'s NAACL2013 tutorial[3]
  4. Geoff Hinton's Coursera course[4]
  5. Theano code samples[5]
  6. Chris Bishop's book Pattern Recognition and Machine Learning (PRML)[6]

---

[1] http://www.iro.umontreal.ca/~bengioy/papers/ftml.pdf
[2] http://techtalks.tv/talks/deep-learning/58122/
[3] http://www.socher.org/index.php/DeepLearningTutorial/
[4] https://www.coursera.org/course/neuralnets
[5] http://deeplearning.net/tutorial/
[6] http://research.microsoft.com/en-us/um/people/cmbishop/prml/

Grading

- The only criteria for grading:
  Are you actively participating and asking questions in class?
  - If you ask (or answer) 3+ questions, grade = A
  - If you ask (or answer) 2 questions, grade = B
  - If you ask (or answer) 1 question, grade = C
  - If you don't ask (or answer) any questions, you get no credit.

Best Advice I got while in Grad School

Always Ask Questions!

Best Advice I got while in Grad School

Always Ask Questions!

- If you don't understand, you must ask questions in order to understand.

Best Advice I got while in Grad School

Always Ask Questions!

- If you don't understand, you must ask questions in order to understand.
- If you understand, you will naturally have questions.

Best Advice I got while in Grad School

Always Ask Questions!

- If you don't understand, you must ask questions in order to understand.
- If you understand, you will naturally have questions.
- Having no questions is a sign that you are not thinking.

Today's Topics

1. Machine Learning background
   - Why Machine Learning is needed?
   - Main Concepts: Generalization, Model Expressiveness, Overfitting
   - Formal Notation

2. Neural Networks
   - 1-Layer Nets (Logistic Regression)
   - 2-Layer Nets and Model Expressiveness
   - Training by Backpropagation

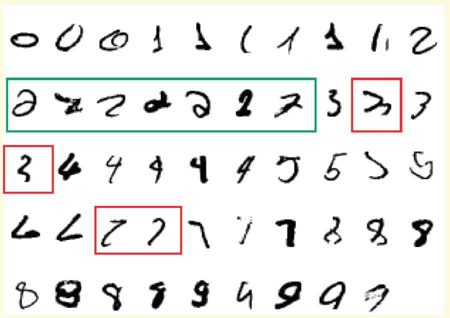
Today's Topics

1. Machine Learning background
   - Why Machine Learning is needed?
   - Main Concepts: Generalization, Model Expressiveness, Overfitting
   - Formal Notation
2. Neural Networks
   - 1-Layer Nets (Logistic Regression)
   - 2-Layer Nets and Model Expressiveness
   - Training by Backpropagation

Write a Program* to Recognize the Digit 2

This is hard to do manually!

```
bool recognizeDigitAs2(int** imagePixels){...}
```

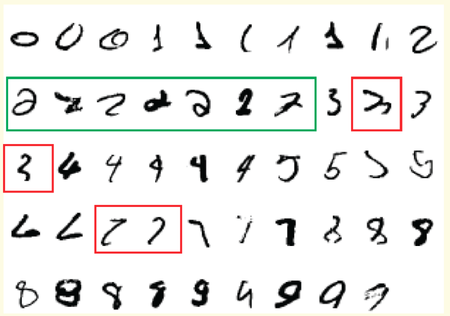


*example from Hinton's Coursera course

Write a Program* to Recognize the Digit 2

This is hard to do manually!

`bool recognizeDigitAs2(int** imagePixels){...}`

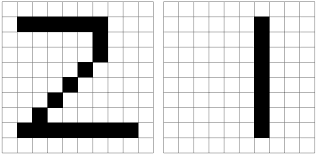


Machine Learning solution:

1. Assume you have a database (training data) of 2's and non-2's.
2. Automatically "learn" this function from data

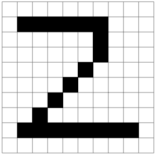
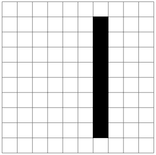
*example from Hinton's Coursera course

A Machine Learning Solution

Training data are represented as pixel matrices: 

Classifier is parameterized by weight matrix of same dimension.

A Machine Learning Solution

Training data are represented as pixel matrices:  

Classifier is parameterized by weight matrix of same dimension.

Training procedure:

1. When observe "2", add 1 to corresponding matrix elements
2. When observe "non-2", subtract 1 to corresponding matrix elements

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

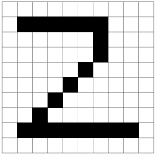
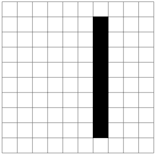
$\rightarrow$

0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0	0	0
0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0

$\rightarrow$

0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	-1	0	0	0
0	0	0	0	1	0	-1	0	0	0
0	0	0	1	0	0	-1	0	0	0
0	0	1	0	0	0	-1	0	0	0
0	1	1	1	1	1	-1	1	1	0
0	0	0	0	0	0	0	0	0	0

A Machine Learning Solution

Training data are represented as pixel matrices:  

Classifier is parameterized by weight matrix of same dimension.

Training procedure:

1. When observe "2", add 1 to corresponding matrix elements
2. When observe "non-2", subtract 1 to corresponding matrix elements

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

→

0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0	0	0
0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0

→

0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	-1	0	0	0
0	0	0	0	1	0	-1	0	0	0
0	0	0	1	0	0	-1	0	0	0
0	0	1	0	0	0	-1	0	0	0
0	1	1	1	1	1	-1	1	1	0
0	0	0	0	0	0	0	0	0	0

Test procedure: given new image, take sum of element-wise product.

If positive, predict "2"; else predict "non-2".

Today's Topics

1. Machine Learning background
   - Why Machine Learning is needed?
   - Main Concepts: Generalization, Model Expressiveness, Overfitting
   - Formal Notation
2. Neural Networks
   - 1-Layer Nets (Logistic Regression)
   - 2-Layer Nets and Model Expressiveness
   - Training by Backpropagation

Generalization $\neq$ Memorization

Key Issue in Machine Learning: Training data is limited

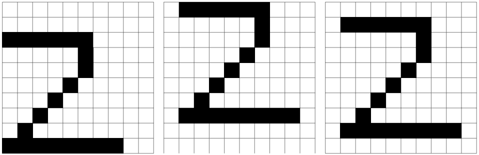
- If the classifier just memorizes the training data, it may perform poorly on new data
- "Generalization" is ability to extend accurate predictions to new data

Generalization $\neq$ Memorization

Key Issue in Machine Learning: Training data is limited

- If the classifier just memorizes the training data, it may perform poorly on new data
- "Generalization" is ability to extend accurate predictions to new data

E.g. consider shifted image:



Will this classifier generalize?

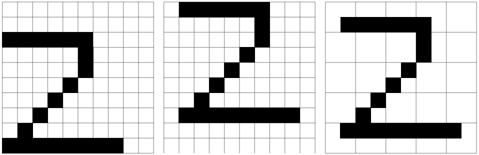
0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	-1	0	0	0
0	0	0	0	1	0	-1	0	0	0
0	0	0	1	0	0	-1	0	0	0
0	0	1	0	0	0	-1	0	0	0
0	1	1	1	1	1	-1	1	1	0
0	0	0	0	0	0	0	0	0	0

Generalization $\neq$ Memorization

One potential way to increase generalization ability:

- Discretize weight matrix with larger grids (fewer weights to train)

E.g. consider shifted image:



Now will this classifier generalize?

0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	-1	0	0	0
0	0	0	0	1	0	-1	0	0	0
0	0	0	1	0	0	-1	0	0	0
0	0	1	0	0	0	-1	0	0	0
0	1	1	1	1	1	-1	1	1	0
0	0	0	0	0	0	0	0	0	0

$\rightarrow$

0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	-1	0	0	0
0	0	0	0	1	0	-1	0	0	0
0	0	0	1	0	0	-1	0	0	0
0	0	1	0	0	0	-1	0	0	0
0	1	1	1	1	1	-1	1	1	0
0	0	0	0	0	0	0	0	0	0

$\rightarrow$

1	1	1	0	0
0	0	0	0	0
0	0	1	-1	0
0	1	0	-1	0
1	1	1	0	1

Model Expressiveness and Overfitting

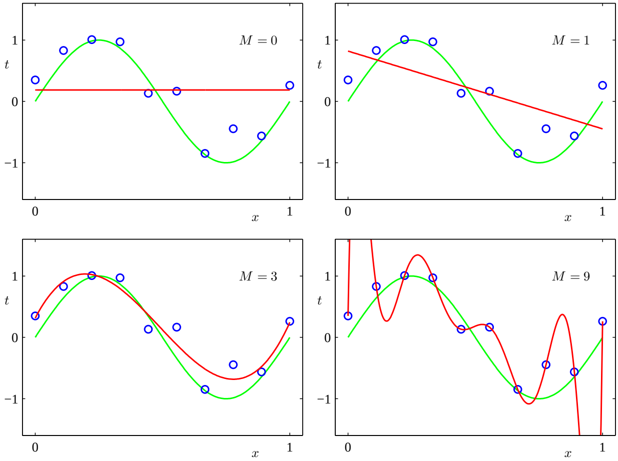
- A model with more weight parameters may fit training data better
- But since training data is limited, expressive model stand the risk of overfitting to peculiarities of the data.

$$\begin{array}{ccc} \text{Less Expressive Model} & \Longleftrightarrow & \text{More Expressive Model} \\ \text{(fewer weights)} & & \text{(more weights)} \end{array}$$

$$\text{Underfit training data} \Longleftrightarrow \text{Overfit training data}$$

Model Expressiveness and Overfitting

Fitting the training data (blue points: x_n)
with a polynomial model: $f(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M$
under squared error objective $\frac{1}{2}\sum_n (f(x_n) - t_n)^2$



from PRML Chapter 1 [Bishop, 2006]

Today's Topics

1. Machine Learning background
   - Why Machine Learning is needed?
   - Main Concepts: Generalization, Model Expressiveness, Overfitting
   - Formal Notation

2. Neural Networks
   - 1-Layer Nets (Logistic Regression)
   - 2-Layer Nets and Model Expressiveness
   - Training by Backpropagation

Basic Problem Setup in Machine Learning

- Training Data: a set of $(x^{(m)}, y^{(m)})_{m=\{1,2,..M\}}$ pairs, where input $x^{(m)} \in R^d$ and output $y^{(m)} = \{0, 1\}$
  - e.g. x=vectorized image pixels, y=2 or non-2
- Goal: Learn function $f : x \to y$ to predicts correctly on new inputs x.

Basic Problem Setup in Machine Learning

- Training Data: a set of $(x^{(m)}, y^{(m)})_{m=\{1,2,..M\}}$ pairs, where input $x^{(m)} \in R^d$ and output $y^{(m)} = \{0, 1\}$
  - e.g. x=vectorized image pixels, y=2 or non-2
- Goal: Learn function $f : x \to y$ to predicts correctly on new inputs x.
  - Step 1: Choose a function model family:
    - e.g. logistic regression, support vector machines, neural networks

Basic Problem Setup in Machine Learning

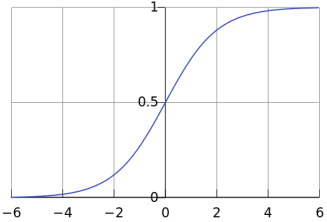
- Training Data: a set of $(x^{(m)}, y^{(m)})_{m=\{1,2,..M\}}$ pairs, where input $x^{(m)} \in R^d$ and output $y^{(m)} = \{0, 1\}$
  - e.g. x=vectorized image pixels, y=2 or non-2
- Goal: Learn function $f : x \to y$ to predicts correctly on new inputs x.
  - Step 1: Choose a function model family:
    - ★ e.g. logistic regression, support vector machines, neural networks
  - Step 2: Optimize parameters w on the Training Data
    - ★ e.g. minimize loss function $\min_w \sum_{m=1}^{M}(f_w(x^{(m)}) - y^{(m)})^2$

Today's Topics

1. Machine Learning background
   - Why Machine Learning is needed?
   - Main Concepts: Generalization, Model Expressiveness, Overfitting
   - Formal Notation
2. Neural Networks
   - 1-Layer Nets (Logistic Regression)
   - 2-Layer Nets and Model Expressiveness
   - Training by Backpropagation

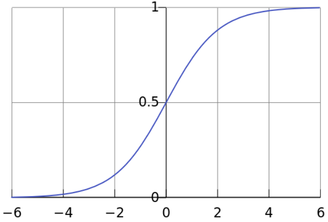
1-Layer Nets (Logistic Regression)

- Function model: $f(x) = \sigma(w^T \cdot x + b)$
  - Parameters: vector $w \in R^d$, b is scalar bias term
  - σ is a non-linearity, e.g. sigmoid: $\sigma(z) = 1/(1 + \exp(-z))$
  - For simplicity, sometimes write $f(x) = \sigma(w^T x)$ where $w = [w; b]$ and $x = [x; 1]$



1-Layer Nets (Logistic Regression)

- Function model: $f(x) = \sigma(w^T \cdot x + b)$
  - Parameters: vector $w \in R^d$, b is scalar bias term
  - σ is a non-linearity, e.g. sigmoid: $\sigma(z) = 1/(1 + \exp(-z))$
  - For simplicity, sometimes write $f(x) = \sigma(w^T x)$ where $w = [w; b]$ and $x = [x; 1]$



- Non-linearity will be important in expressiveness multi-layer nets. Other non-linearities, e.g., $tanh(z) = (e^z - e^{-z})/(e^z + e^{-z})$

Training 1-Layer Nets: Gradient

- Assume Squared-Error* $Loss(w) = \frac{1}{2}\sum_m(\sigma(w^T x^{(m)}) - y^{(m)})^2$

*An alternative is Cross-Entropy loss:
$\sum_m y^{(m)} \log(\sigma(w^T x^{(m)})) + (1 - y^{(m)}) \log(1 - \sigma(w^T x^{(m)}))$

Training 1-Layer Nets: Gradient

- Assume Squared-Error* $Loss(w) = \frac{1}{2}\sum_m (\sigma(w^T x^{(m)}) - y^{(m)})^2$
- Gradient: $\nabla_w Loss = \sum_m \left[\sigma(w^T x^{(m)}) - y^{(m)}\right] \sigma'(w^T x^{(m)}) x^{(m)}$

*An alternative is Cross-Entropy loss:
$\sum_m y^{(m)} \log(\sigma(w^T x^{(m)})) + (1 - y^{(m)}) \log(1 - \sigma(w^T x^{(m)}))$

Training 1-Layer Nets: Gradient

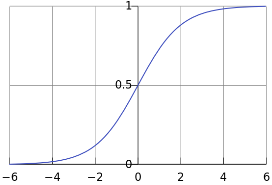
- Assume Squared-Error* $Loss(w) = \frac{1}{2}\sum_m (\sigma(w^T x^{(m)}) - y^{(m)})^2$
- Gradient: $\nabla_w Loss = \sum_m \left[\sigma(w^T x^{(m)}) - y^{(m)}\right] \sigma'(w^T x^{(m)}) x^{(m)}$
  - General form of gradient: $\sum_m Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)}$

*An alternative is Cross-Entropy loss:
$\sum_m y^{(m)} \log(\sigma(w^T x^{(m)})) + (1 - y^{(m)}) \log(1 - \sigma(w^T x^{(m)}))$

Training 1-Layer Nets: Gradient

- Assume Squared-Error* $Loss(w) = \frac{1}{2}\sum_m (\sigma(w^T x^{(m)}) - y^{(m)})^2$
- Gradient: $\nabla_w Loss = \sum_m \left[\sigma(w^T x^{(m)}) - y^{(m)}\right] \sigma'(w^T x^{(m)}) x^{(m)}$
  - General form of gradient: $\sum_m Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)}$
- Derivative of sigmoid $\sigma(z) = 1/(1+\exp(-z))$:

$$
\begin{aligned}
\sigma'(z) &= \frac{d}{dz}\left(\frac{1}{1+\exp(-z)}\right) \\
&= -\left(\frac{1}{1+\exp(-z)}\right)^2 \frac{d}{dz}(1+\exp(-z)) \\
&= -\left(\frac{1}{1+\exp(-z)}\right)^2 \exp(-z)(-1) \\
&= \left(\frac{1}{1+\exp(-z)}\right)\left(\frac{\exp(-z)}{1+\exp(-z)}\right) \\
&= \sigma(z)(1-\sigma(z))
\end{aligned}
$$



*An alternative is Cross-Entropy loss:
$\sum_m y^{(m)} \log(\sigma(w^T x^{(m)})) + (1-y^{(m)}) \log(1-\sigma(w^T x^{(m)}))$

Training 1-Layer Nets: Gradient Descent Algorithm

- General form of gradient: $\sum_m Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)}$
- Gradient descent algorithm:
  1. Initialize w
  2. Compute $\nabla_w Loss = \sum_m Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)}$
  3. $w \leftarrow w - \gamma(\nabla_w Loss)$
  4. Repeat steps 2-3 until some condition satisfied

Training 1-Layer Nets: Gradient Descent Algorithm

- General form of gradient: $\sum_m Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)}$
- Gradient descent algorithm:
  1. Initialize w
  2. Compute $\nabla_w Loss = \sum_m Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)}$
  3. $w \leftarrow w - \gamma(\nabla_w Loss)$
  4. Repeat steps 2-3 until some condition satisfied
- Stochastic gradient descent (SGD) algorithm:
  1. Initialize w
  2. for each sample $(x^{(m)}, y^{(m)})$ in training set:
  3. $\quad\quad w \leftarrow w - \gamma(Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)})$
  4. Repeat loop 2-3 until some condition satisfied

Training 1-Layer Nets: Gradient Descent Algorithm

- General form of gradient: $\sum_m Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)}$
- Gradient descent algorithm:
  1. Initialize w
  2. Compute $\nabla_w Loss = \sum_m Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)}$
  3. $w \leftarrow w - \gamma(\nabla_w Loss)$
  4. Repeat steps 2-3 until some condition satisfied
- Stochastic gradient descent (SGD) algorithm:
  1. Initialize w
  2. for each sample $(x^{(m)}, y^{(m)})$ in training set:
  3. $\quad w \leftarrow w - \gamma(Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)})$
  4. Repeat loop 2-3 until some condition satisfied
- Learning rate $\gamma > 0$ & stopping condition are important in practice

Intuition of SGD update

- for some sample $(x^{(m)}, y^{(m)})$:

$$w \leftarrow w - \gamma((\sigma(w^T x^{(m)}) - y^{(m)}) * \sigma'(w^T x^{(m)}) * x^{(m)})$$

$\sigma(w^T x^{(m)})$	$y^{(m)}$	*Error*	new w	new prediction
0	0	0	no change	0
1	1	0	no change	1
0	1	-1	$w + \gamma\sigma'(in^{(m)})x^{(m)}$	≥ 0
1	0	$+1$	$w - \gamma\sigma'(in^{(m)})x^{(m)}$	≤ 1

Intuition of SGD update

- for some sample $(x^{(m)}, y^{(m)})$:

$$w \leftarrow w - \gamma((\sigma(w^T x^{(m)}) - y^{(m)}) * \sigma'(w^T x^{(m)}) * x^{(m)})$$

$\sigma(w^T x^{(m)})$	$y^{(m)}$	*Error*	new w	new prediction
0	0	0	no change	0
1	1	0	no change	1
0	1	-1	$w + \gamma\sigma'(in^{(m)})x^{(m)}$	≥ 0
1	0	$+1$	$w - \gamma\sigma'(in^{(m)})x^{(m)}$	≤ 1

- $\left[w + \gamma\sigma'(in^{(m)})x^{(m)}\right]^T x^{(m)} = w^T x^{(m)} + \gamma\sigma'(in^{(m)})||x^{(m)}||^2 \geq w^T x^{(m)}$

Intuition of SGD update

- for some sample $(x^{(m)}, y^{(m)})$:

$$w \leftarrow w - \gamma((\sigma(w^T x^{(m)}) - y^{(m)}) * \sigma'(w^T x^{(m)}) * x^{(m)})$$

$\sigma(w^T x^{(m)})$	$y^{(m)}$	*Error*	new w	new prediction
0	0	0	no change	0
1	1	0	no change	1
0	1	-1	$w + \gamma\sigma'(in^{(m)})x^{(m)}$	≥ 0
1	0	$+1$	$w - \gamma\sigma'(in^{(m)})x^{(m)}$	≤ 1

- $\left[w + \gamma\sigma'(in^{(m)})x^{(m)}\right]^T x^{(m)} = w^T x^{(m)} + \gamma\sigma'(in^{(m)})||x^{(m)}||^2 \geq w^T x^{(m)}$
- $\sigma'(w^T x^{(m)})$ is near 0 when confident, near 0.25 when uncertain.

Intuition of SGD update

- for some sample $(x^{(m)}, y^{(m)})$:
$$w \leftarrow w - \gamma((\sigma(w^T x^{(m)}) - y^{(m)}) * \sigma'(w^T x^{(m)}) * x^{(m)})$$

$\sigma(w^T x^{(m)})$	$y^{(m)}$	*Error*	new w	new prediction
0	0	0	no change	0
1	1	0	no change	1
0	1	-1	$w + \gamma\sigma'(in^{(m)})x^{(m)}$	≥ 0
1	0	$+1$	$w - \gamma\sigma'(in^{(m)})x^{(m)}$	≤ 1

- $\left[w + \gamma\sigma'(in^{(m)})x^{(m)}\right]^T x^{(m)} = w^T x^{(m)} + \gamma\sigma'(in^{(m)})||x^{(m)}||^2 \geq w^T x^{(m)}$
- $\sigma'(w^T x^{(m)})$ is near 0 when confident, near 0.25 when uncertain.
- large γ = more aggressive updates; small γ = more conservative

Intuition of SGD update

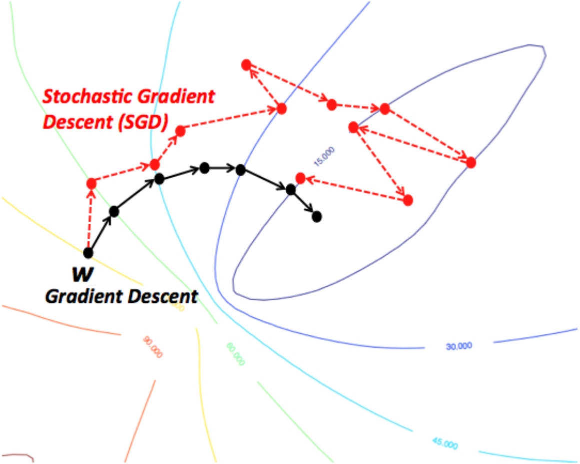
- for some sample $(x^{(m)}, y^{(m)})$:
$$w \leftarrow w - \gamma((\sigma(w^T x^{(m)}) - y^{(m)}) * \sigma'(w^T x^{(m)}) * x^{(m)})$$

$\sigma(w^T x^{(m)})$	$y^{(m)}$	*Error*	new w	new prediction
0	0	0	no change	0
1	1	0	no change	1
0	1	-1	$w + \gamma\sigma'(in^{(m)})x^{(m)}$	≥ 0
1	0	$+1$	$w - \gamma\sigma'(in^{(m)})x^{(m)}$	≤ 1

- $\left[w + \gamma\sigma'(in^{(m)})x^{(m)}\right]^T x^{(m)} = w^T x^{(m)} + \gamma\sigma'(in^{(m)})||x^{(m)}||^2 \geq w^T x^{(m)}$
- $\sigma'(w^T x^{(m)})$ is near 0 when confident, near 0.25 when uncertain.
- large γ = more aggressive updates; small γ = more conservative
- SGD improves classification for current sample, but no guarantee about others

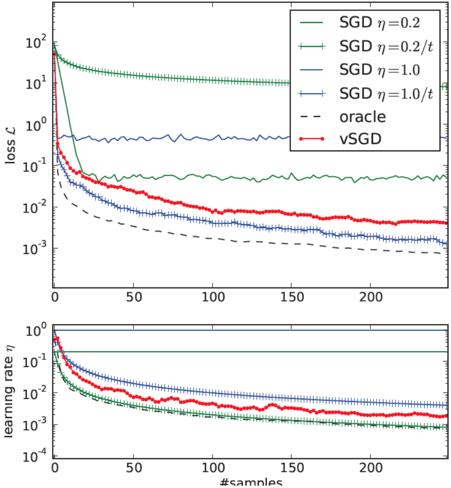
Geometric view of SGD update

- Loss objective contour plot: $\frac{1}{2}\sum_m(\sigma(w^T x^{(m)}) - y^{(m)})^2 + ||w||$
  - Gradient descent goes in steepest descent direction, but slower to compute per iteration for large datasets
  - SGD can be viewed as noisy descent, but faster per iteration
  - In practice, a good tradeoff is mini-batch SGD



Effect of Learning Rate γ on Convergence Speed

- SGD update: $w \leftarrow w - \gamma(Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)})$
  - Ideally, γ should be as large as possible without causing divergence.
  - Common heuristic: $\gamma(t) = \frac{\gamma_0}{1+\nu t} = O(1/t)$
- Analysis by [Schaul et al., 2013] (in plot, $\eta \equiv \gamma$):



Generalization issues: Regularization and Early-stopping

- Optimizing $Loss(w) = \frac{1}{2}\sum_m(\sigma(w^T x^{(m)}) - y^{(m)})^2$ on training data not necessarily leads to generalization.

Figures from Chapter 5, [Bishop, 2006]

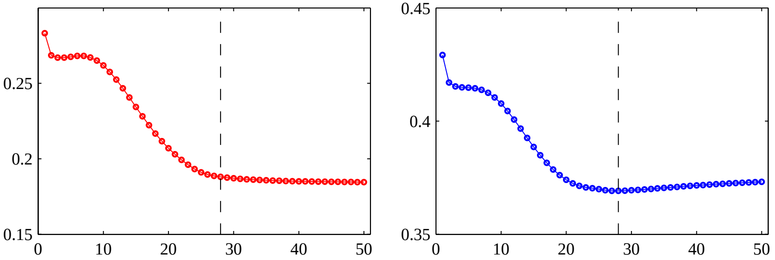
Generalization issues: Regularization and Early-stopping

- Optimizing $Loss(w) = \frac{1}{2}\sum_m(\sigma(w^T x^{(m)}) - y^{(m)})^2$ on training data not necessarily leads to generalization.
  1. Adding regularization: $Loss(w) = \frac{1}{2}\sum_m(\sigma(w^T x^{(m)}) - y^{(m)})^2 + ||w||$ reduces sensitivity to training input and decreases risk of overfitting

Figures from Chapter 5, [Bishop, 2006]

Generalization issues: Regularization and Early-stopping

- Optimizing $Loss(w) = \frac{1}{2}\sum_m(\sigma(w^T x^{(m)}) - y^{(m)})^2$ on training data not necessarily leads to generalization.
  1. Adding regularization: $Loss(w) = \frac{1}{2}\sum_m(\sigma(w^T x^{(m)}) - y^{(m)})^2 + ||w||$ reduces sensitivity to training input and decreases risk of overfitting
  2. Early Stopping:
     - ★ Prepare separate training and validation (development) data
     - ★ Optimize $Loss(w)$ on training but stop when $Loss(w)$ on validation stops improving



Figures from Chapter 5, [Bishop, 2006]

Summary

1. Given Training Data: $(x^{(m)}, y^{(m)})_{m=\{1,2,..M\}}$
2. Optimize a model $f(x) = \sigma(w^T \cdot x + b)$ under $Loss(w) = \frac{1}{2}\sum_m (\sigma(w^T x^{(m)}) - y^{(m)})^2$

Summary

1. Given Training Data: $(x^{(m)}, y^{(m)})_{m=\{1,2,..M\}}$
2. Optimize a model $f(x) = \sigma(w^T \cdot x + b)$ under $Loss(w) = \frac{1}{2}\sum_m(\sigma(w^T x^{(m)}) - y^{(m)})^2$
3. General form of gradient: $\sum_m Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)}$

Summary

1. Given Training Data: $(x^{(m)}, y^{(m)})_{m=\{1,2,..M\}}$
2. Optimize a model $f(x) = \sigma(w^T \cdot x + b)$ under $Loss(w) = \frac{1}{2}\sum_m (\sigma(w^T x^{(m)}) - y^{(m)})^2$
3. General form of gradient: $\sum_m Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)}$
4. SGD algorithm: for each sample $(x^{(m)}, y^{(m)})$ in training set,
$$w \leftarrow w - \gamma(Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)})$$

Summary

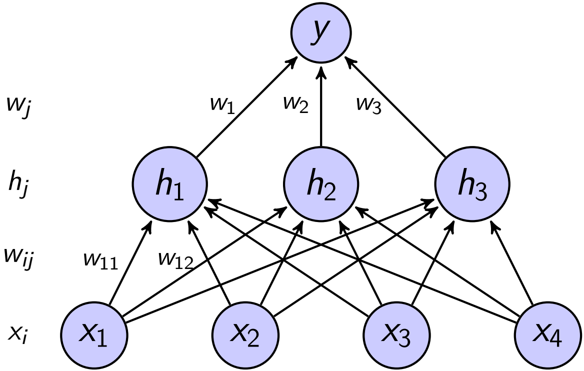
1. Given Training Data: $(x^{(m)}, y^{(m)})_{m=\{1,2,..M\}}$
2. Optimize a model $f(x) = \sigma(w^T \cdot x + b)$ under $Loss(w) = \frac{1}{2}\sum_m (\sigma(w^T x^{(m)}) - y^{(m)})^2$
3. General form of gradient: $\sum_m Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)}$
4. SGD algorithm: for each sample $(x^{(m)}, y^{(m)})$ in training set,
$$w \leftarrow w - \gamma(Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)})$$
5. Important issues:
   - Optimization speed/convergence: batch vs. mini-batch, learning rate γ
   - Generalization ability: regularization, early-stopping

Today's Topics

1. Machine Learning background
   - Why Machine Learning is needed?
   - Main Concepts: Generalization, Model Expressiveness, Overfitting
   - Formal Notation

2. Neural Networks
   - 1-Layer Nets (Logistic Regression)
   - 2-Layer Nets and Model Expressiveness
   - Training by Backpropagation

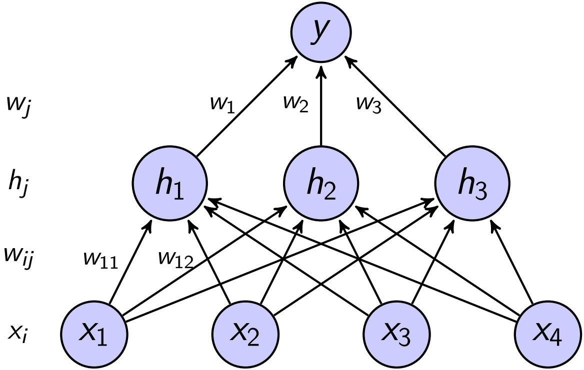
2-Layer Neural Networks



$$f(x) = \sigma(\textstyle\sum_j w_j \cdot h_j) = \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} x_i))$$

Called Multilayer Perceptron (MLP), but more like multilayer logistic regression

2-Layer Neural Networks



$$f(x) = \sigma(\sum_j w_j \cdot h_j) = \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} x_i))$$

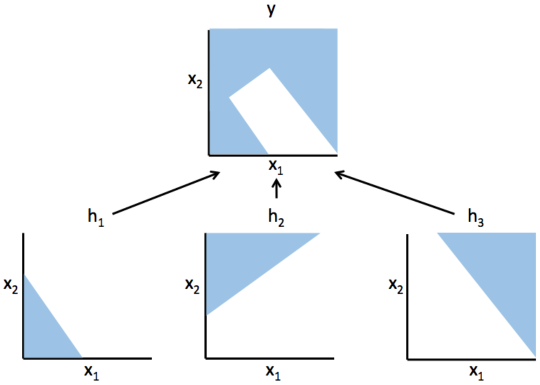
Hidden units h_j's can be viewed as new "features" from combining x_i's

---

Called Multilayer Perceptron (MLP), but more like multilayer logistic regression

Modeling complex non-linearities

- Given same number of units (with non-linear activation), a deeper architecture is more expressive than a shallow one [Bishop, 1995]
  - 1-layer nets only model linear hyperplanes
  - 2-layer nets are universal function approximators: given infinite hidden nodes, it can express any continuous function
  - $>$3-layer nets can do so with fewer nodes/weights

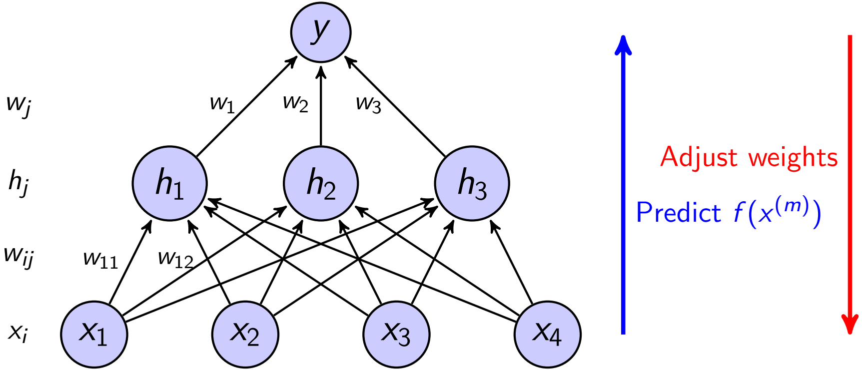


Today's Topics

1. Machine Learning background
   - Why Machine Learning is needed?
   - Main Concepts: Generalization, Model Expressiveness, Overfitting
   - Formal Notation

2. Neural Networks
   - 1-Layer Nets (Logistic Regression)
   - 2-Layer Nets and Model Expressiveness
   - Training by Backpropagation

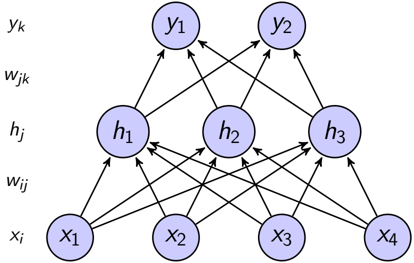
Training a 2-Layer Net with Backpropagation



1. For each sample, compute $f(x^{(m)}) = \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} x_i^{(m)}))$
2. If $f(x^{(m)}) \neq y^{(m)}$, back-propagate error and adjust weights $\{w_{ij}, w_j\}$.

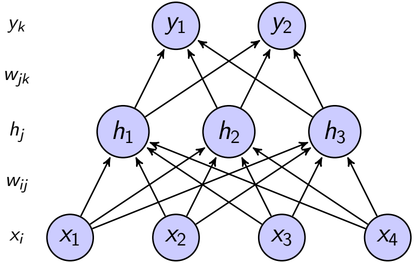
Derivatives of the weights

Assume two outputs (y_1, y_2) per input x,
and loss per sample: $Loss = \sum_k \frac{1}{2} \left[\sigma(in_k) - y_k\right]^2$



Derivatives of the weights

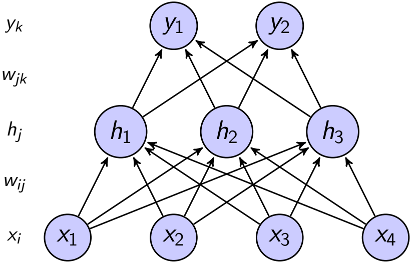
Assume two outputs (y_1, y_2) per input x,
and loss per sample: $Loss = \sum_k \frac{1}{2}\left[\sigma(in_k) - y_k\right]^2$



$$\frac{\partial Loss}{\partial w_{jk}} = \frac{\partial Loss}{\partial in_k}\frac{\partial in_k}{\partial w_{jk}} = \delta_k \frac{\partial(\sum_j w_{jk} h_j)}{\partial w_{jk}} = \delta_k h_j$$

Derivatives of the weights

Assume two outputs (y_1, y_2) per input x,
and loss per sample: $Loss = \sum_k \frac{1}{2} \left[\sigma(in_k) - y_k\right]^2$

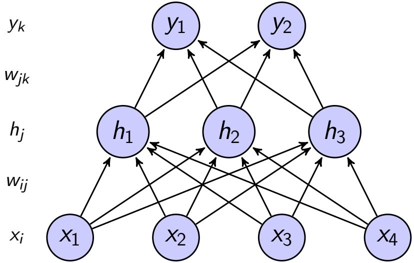


$$\frac{\partial Loss}{\partial w_{jk}} = \frac{\partial Loss}{\partial in_k} \frac{\partial in_k}{\partial w_{jk}} = \delta_k \frac{\partial(\sum_j w_{jk} h_j)}{\partial w_{jk}} = \delta_k h_j$$

$$\frac{\partial Loss}{\partial w_{ij}} = \frac{\partial Loss}{\partial in_j} \frac{\partial in_j}{\partial w_{ij}} = \delta_j \frac{\partial(\sum_j w_{ij} x_i)}{\partial w_{ij}} = \delta_j x_i$$

Derivatives of the weights

Assume two outputs (y_1, y_2) per input x,
and loss per sample: $Loss = \sum_k \frac{1}{2}\left[\sigma(in_k) - y_k\right]^2$



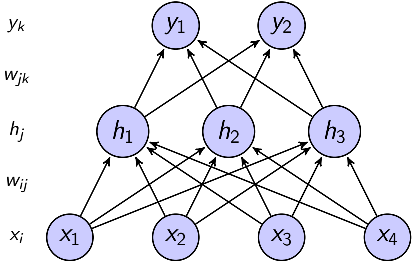
$$\frac{\partial Loss}{\partial w_{jk}} = \frac{\partial Loss}{\partial in_k}\frac{\partial in_k}{\partial w_{jk}} = \delta_k \frac{\partial(\sum_j w_{jk} h_j)}{\partial w_{jk}} = \delta_k h_j$$

$$\frac{\partial Loss}{\partial w_{ij}} = \frac{\partial Loss}{\partial in_j}\frac{\partial in_j}{\partial w_{ij}} = \delta_j \frac{\partial(\sum_j w_{ij} x_i)}{\partial w_{ij}} = \delta_j x_i$$

$$\delta_k = \frac{\partial}{\partial in_k}\left(\sum_k \frac{1}{2}\left[\sigma(in_k) - y_k\right]^2\right) = \left[\sigma(in_k) - y_k\right]\sigma'(in_k)$$

Derivatives of the weights

Assume two outputs (y_1, y_2) per input x,
and loss per sample: $Loss = \sum_k \frac{1}{2}\left[\sigma(in_k) - y_k\right]^2$



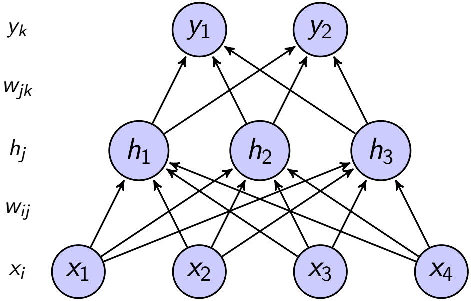
$$\frac{\partial Loss}{\partial w_{jk}} = \frac{\partial Loss}{\partial in_k}\frac{\partial in_k}{\partial w_{jk}} = \delta_k \frac{\partial(\sum_j w_{jk} h_j)}{\partial w_{jk}} = \delta_k h_j$$
$$\frac{\partial Loss}{\partial w_{ij}} = \frac{\partial Loss}{\partial in_j}\frac{\partial in_j}{\partial w_{ij}} = \delta_j \frac{\partial(\sum_j w_{ij} x_i)}{\partial w_{ij}} = \delta_j x_i$$
$$\delta_k = \frac{\partial}{\partial in_k}\left(\sum_k \frac{1}{2}\left[\sigma(in_k) - y_k\right]^2\right) = \left[\sigma(in_k) - y_k\right]\sigma'(in_k)$$
$$\delta_j = \sum_k \frac{\partial Loss}{\partial in_k}\frac{\partial in_k}{\partial in_j} = \sum_k \delta_k \cdot \frac{\partial}{\partial in_j}\left(\sum_j w_{jk}\sigma(in_j)\right) = \left[\sum_k \delta_k w_{jk}\right]\sigma'(in_j)$$

Backpropagation Algorithm

All updates involve some scaled error from output $*$ input feature:

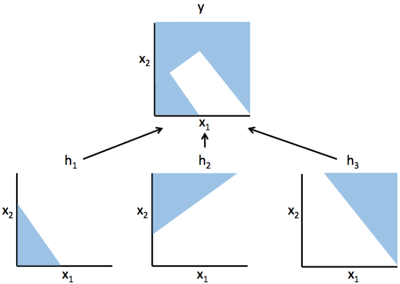
- $\frac{\partial Loss}{\partial w_{jk}} = \delta_k h_j$ where $\delta_k = [\sigma(in_k) - y_k]\,\sigma'(in_k)$
- $\frac{\partial Loss}{\partial w_{ij}} = \delta_j x_i$ where $\delta_j = [\sum_k \delta_k w_{jk}]\,\sigma'(in_j)$

After forward pass, compute δ_k from final layer, then δ_j for previous layer. For deeper nets, iterate backwards further.



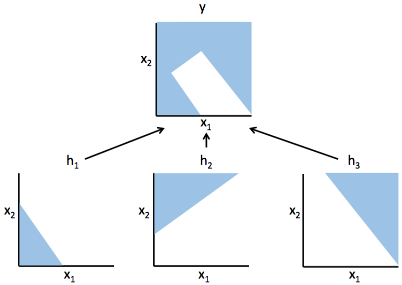
Summary

1. By extending from 1-layer to 2-layer net, we get dramatic increase in model expressiveness:



Summary

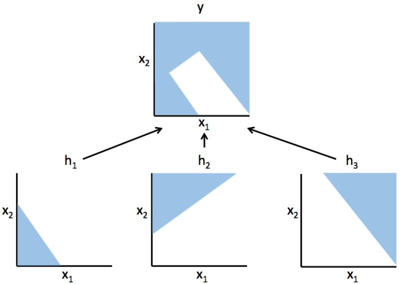
1. By extending from 1-layer to 2-layer net, we get dramatic increase in model expressiveness:



2. Backpropagation is an efficient way to train 2-layer nets:
   - Similar to SGD for 1-layer net, just more chaining in gradient

Summary

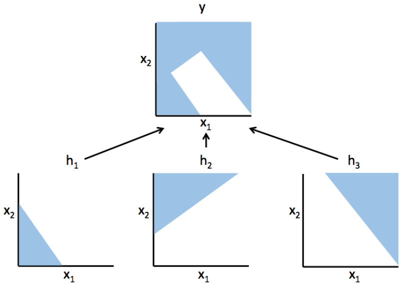
1. By extending from 1-layer to 2-layer net, we get dramatic increase in model expressiveness:



2. Backpropagation is an efficient way to train 2-layer nets:
   - Similar to SGD for 1-layer net, just more chaining in gradient
   - General form: update w_{ij} by $\delta_j x_i$, and δ_j is scaled/weighted sum of errors from outgoing layers

Summary

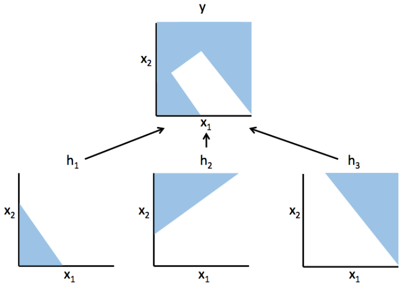
1. By extending from 1-layer to 2-layer net, we get dramatic increase in model expressiveness:



2. Backpropagation is an efficient way to train 2-layer nets:
   - Similar to SGD for 1-layer net, just more chaining in gradient
   - General form: update w_{ij} by $\delta_j x_i$, and δ_j is scaled/weighted sum of errors from outgoing layers
3. Ideally, we want even deeper architectures

Summary

1. By extending from 1-layer to 2-layer net, we get dramatic increase in model expressiveness:



2. Backpropagation is an efficient way to train 2-layer nets:
   - Similar to SGD for 1-layer net, just more chaining in gradient
   - General form: update w_{ij} by $\delta_j x_i$, and δ_j is scaled/weighted sum of errors from outgoing layers
3. Ideally, we want even deeper architectures
   - But Backpropagation becomes ineffective due to vanishing gradients
   - Deep Learning comes to the rescue! (next lecture)

References I

Bengio, Y. (2009).
Learning Deep Architectures for AI, volume Foundations and Trends in Machine Learning.
NOW Publishers.

Bishop, C. (1995).
Neural Networks for Pattern Recognition.
Oxford University Press.

Bishop, C. (2006).
Pattern Recognition and Machine Learning.
Springer.

Lee, H., Grosse, R., Ranganath, R., and Ng, A. (2009).
Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations.
In *ICML*.

Schaul, T., Zhang, S., and LeCun, Y. (2013).
No more pesky learning rates.
In *Proc. International Conference on Machine learning (ICML'13)*.