Deep Learning: An Engineer's Dream and a Mathematician's Nightmare

John McKay

Information Processing & Algorithms Laboratory
Dept. of Electrical Engineering
Pennsylvania State University

Dartmouth Applied Math Seminar, 2017

Contents

Why Give a Talk to Mathematicians on DL?

- Deep learning is dominating machine learning (image classification, object detection, text parsing, etc).
- Industry is keeping pace with/beating academia in terms of breakthroughs and adoption. We need mathematicians to provide rigor to modern strategies.

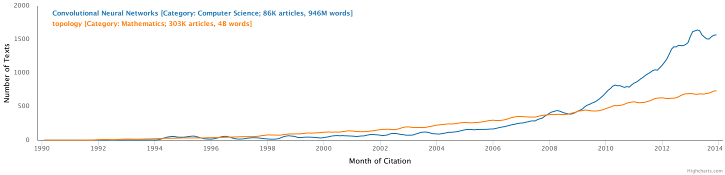


Figure: Per-month publication totals of papers on arXiv. Topology (orange) provided as a reference.

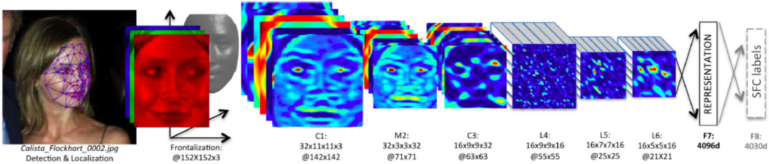
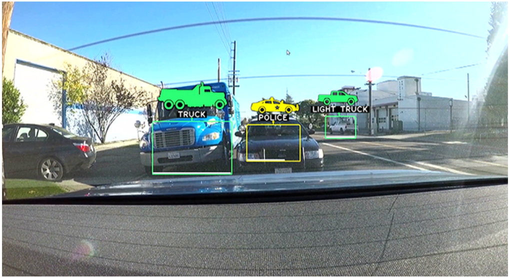
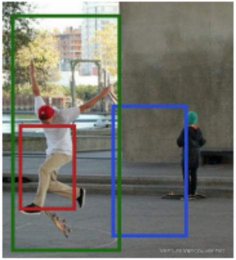


Figure 2. **Outline of the *DeepFace* architecture.** A front-end of a single convolution-pooling-convolution filtering on the rectified input, followed by three locally-connected layers and two fully-connected layers. Colors illustrate outputs for each layer. The net includes more than 120 million parameters, where more than 95% come from the local and fully connected layers.

Gatys et al. (2015), Taigman et al. (2014)



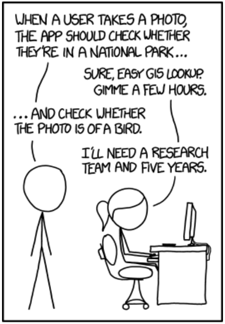
Two men are standing on a skateboard on a ramp outside on a sunny day. One man is wearing black pants, a white shirt and black pants. The man on the skateboard is wearing jeans. The man's arms are stretched out in front of him. The man is wearing a white shirt and black pants. The other man is wearing a white shirt and black pants.



Courtsey Nvidia,Krause et al. (2016)

This talk will. . .

- Provide a comprehensive introduction to Deep Learning
- Cover most of the modern ideas and trends
- Illuminate the issues underpinning popular strategies



IN CS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALLY IMPOSSIBLE.

Traditional Supervised Image Classification

Training

1. Input Labeled Data

↓

2. Extract Features

↓

3. Train Classifier

Traditional Supervised Image Classification

Training

1. Input Labeled Data

↓

2. Extract Features

↓

3. Train Classifier

Traditional Supervised Image Classification

Training

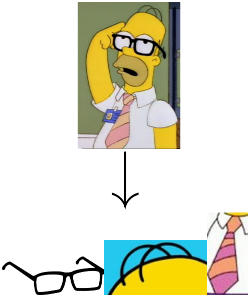
1. Input Labeled Data

↓

2. Extract Features

↓

3. Train Classifier



Traditional Supervised Image Classification

Training

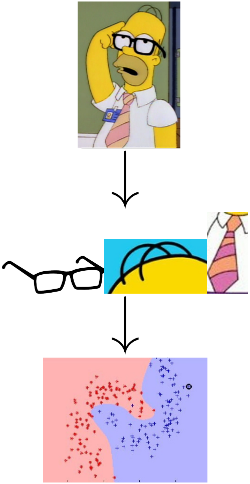
1. Input Labeled Data

↓

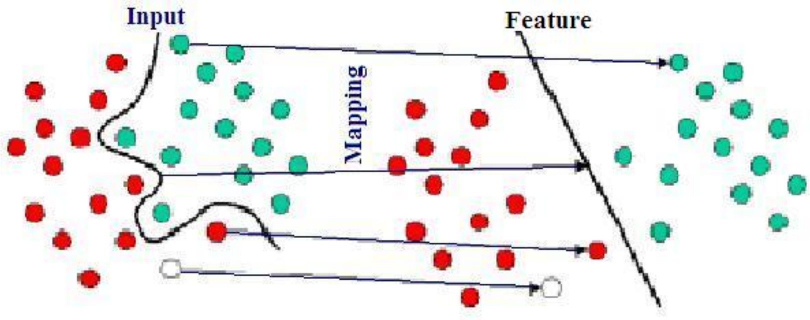
2. Extract Features

↓

3. Train Classifier



Training



http://www.cs.trincoll.edu

Feature Extraction

- Popular features: scale-invariant feature transforms (SIFT), speeded up robust features (SURF), sparse reconstruction-based classification (SRC) coefficients.
- They are hand crafted from analytical work trying to decipher invariant (translation, rotational, scale, etc.) and discriminatory image characteristics.

Lowe (1999)

Feature Extraction

- Popular features: scale-invariant feature transforms (SIFT), speeded up robust features (SURF), sparse reconstruction-based classification (SRC) coefficients.
- They are hand crafted from analytical work trying to decipher invariant (translation, rotational, scale, etc.) and discriminatory image characteristics.
- Why not have the computer do it?

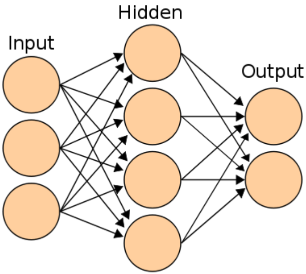
Lowe (1999)

Contents

Single Hidden Layer Network

- Input $\boldsymbol{x} \in \mathbb{R}^d$, NN output $\boldsymbol{a}^{(2)}$

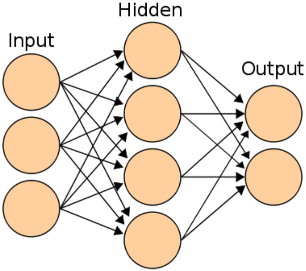
$$\boldsymbol{a}^{(2)} = f_2(f_1(\boldsymbol{x}))$$



Single Hidden Layer Network

- Input $\boldsymbol{x} \in \mathbb{R}^d$, NN output $\boldsymbol{a}^{(2)}$

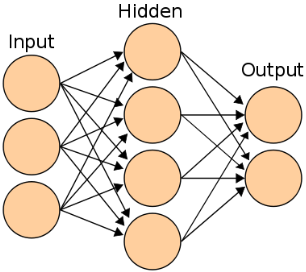
$$\boldsymbol{a}^{(2)} = W_2 \underbrace{\sigma(W_1 \boldsymbol{x} + \boldsymbol{b}_1)}_{\text{1st Layer (hidden)}} + \boldsymbol{b}_2$$



Single Hidden Layer Network

- Input $\boldsymbol{x} \in \mathbb{R}^d$, NN output $\boldsymbol{a}^{(2)}$

$$\boldsymbol{a}^{(2)} = \overbrace{W_2 \underbrace{\sigma(W_1 \boldsymbol{x} + \boldsymbol{b}_1)}_{\text{1st Layer (Hidden)}} + \boldsymbol{b}_2}^{\text{2nd Layer (Output)}}$$



Single Hidden Layer Network

- Input $\boldsymbol{x} \in \mathbb{R}^d$, NN output $\boldsymbol{a}^{(2)}$

$$\boldsymbol{a}^{(2)} = \overbrace{W_2\, \underbrace{\sigma(W_1 \boldsymbol{x} + \boldsymbol{b}_1)}_{\text{1st Layer (Hidden)}} + \boldsymbol{b}_2}^{\text{2nd Layer (Output)}}$$

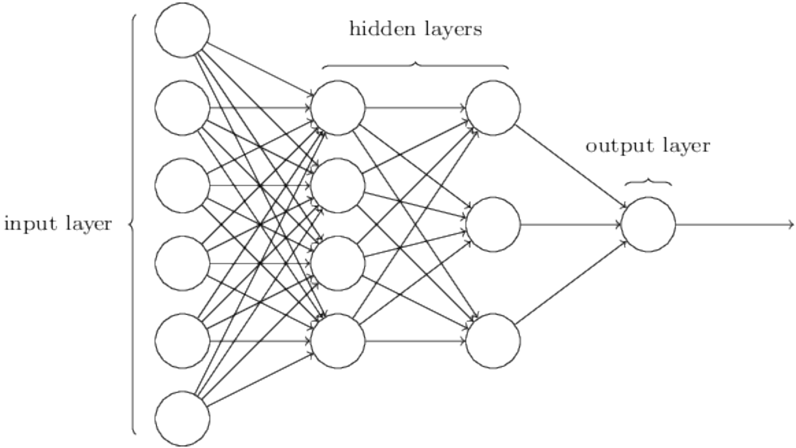
- W_1, W_2 are the weights and $\boldsymbol{b}_1$, $\boldsymbol{b}_2$ the biases.
- σ is an activation function (more on this later)

Single Hidden Layer Network

- Input $\boldsymbol{x} \in \mathbb{R}^d$, NN output $\boldsymbol{a}^{(2)}$

$$\boldsymbol{a}^{(2)} = \overbrace{W_2 \, \underbrace{\sigma(W_1 \boldsymbol{x} + \boldsymbol{b}_1)}_{\text{1st Layer (Hidden)}} + \boldsymbol{b}_2}^{\text{2nd Layer (Output)}}$$

- W_1, W_2 are the weights and $\boldsymbol{b}_1$, $\boldsymbol{b}_2$ the biases.
- σ is an activation function (more on this later)



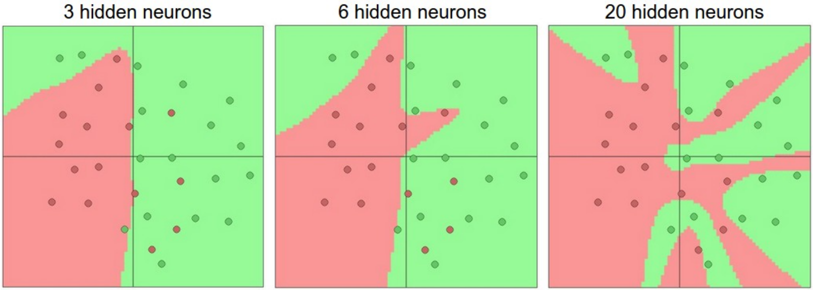
Nielsen (2015)

Universal Approximation Theorem

- Universal Approximation Theorem: given enough hidden layers, weight depth, and σ of a certain set of nonlinear functions, then a NN with linear output can represent any Borel measurable function mapping finite dimensional space to another.
- Nice, but representation$\neq$prediction.

Hornik et al. (1989),Leshno et al. (1993)

More Parameters, More Descriptiveness



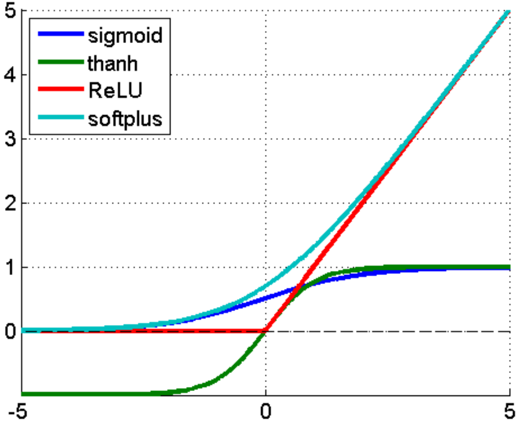
Li/Kaparthy/Johnson, Stanford CS231

Activation Functions

- Feature spaces require nonlinear activation functions.
- There is no one “best” choice for an activation function, to say nothing of how many layers have one specific activation function.
- Since 2009, the go-to: Rectified Linear Units (ReLU)

$$\sigma(\boldsymbol{z}) = \boldsymbol{a} \text{ such that } a_i = \max(0, z_i)$$

Activation Functions



imiloainf.wordpress.com (her typo)

Output Layer

- Nevermind, sigmoids are okay.
- For binary problem, sigmoid acts as a probability
- For multinomial, popular to use softmax

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

- softmax $\approx$ argmax function (one hot vector output)

Contents

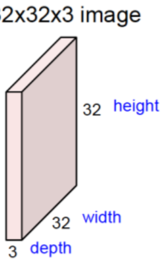
Convolutional Neural Networks

- Convolutions with filters are a way to do weight sharing and exploit sparse interactions.
- Fewer weights? Less to train/save.
- Modern CNNs are the most accurate algorithms humans have ever devised (by a bit).

LeCun et al. (1998)

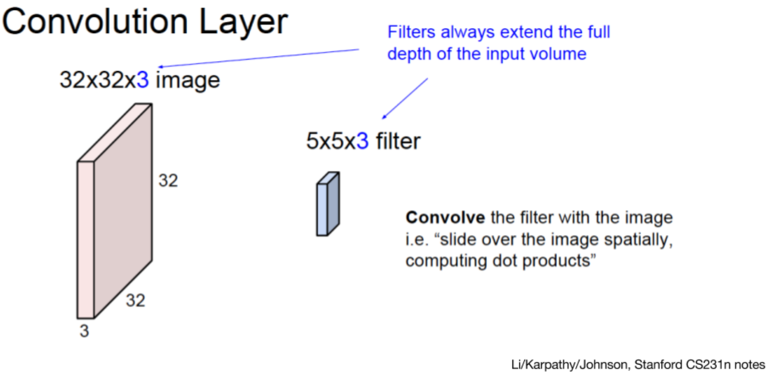
Convolutional Neural Network

Convolution Layer



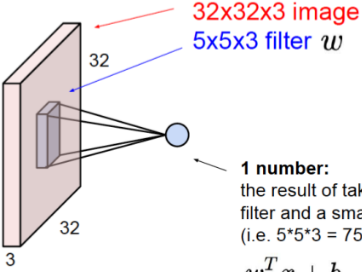
Li/Karpathy/Johnson, Stanford CS231n notes

Convolutional Neural Network



Convolutional Neural Network

Convolution Layer

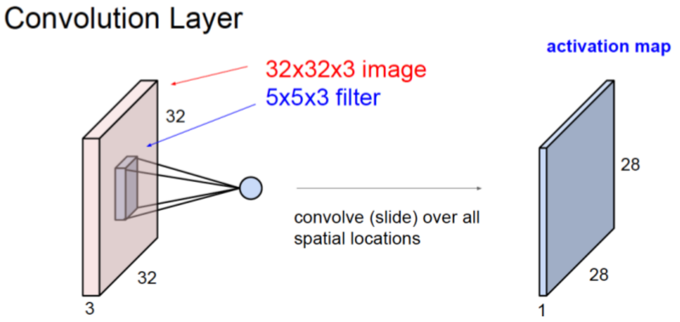


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

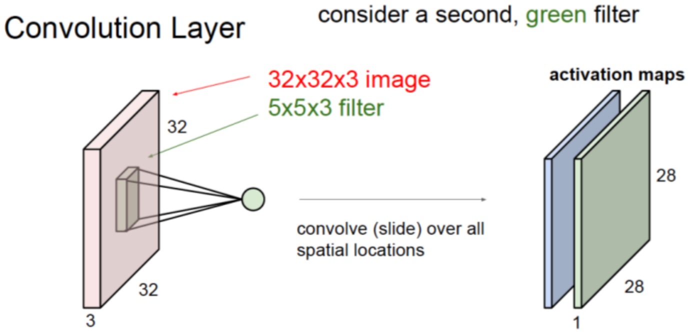
Li/Karpathy/Johnson, Stanford CS231n notes

Convolutional Neural Network



Li/Karpathy/Johnson, Stanford CS231n notes

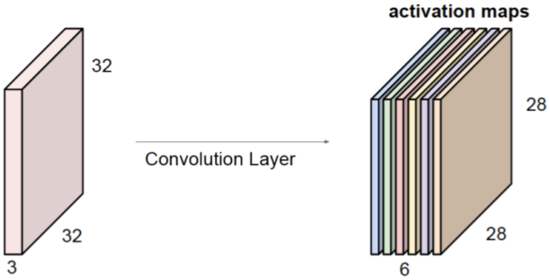
Convolutional Neural Network



Li/Karpathy/Johnson, Stanford CS231n notes

Convolutional Neural Network

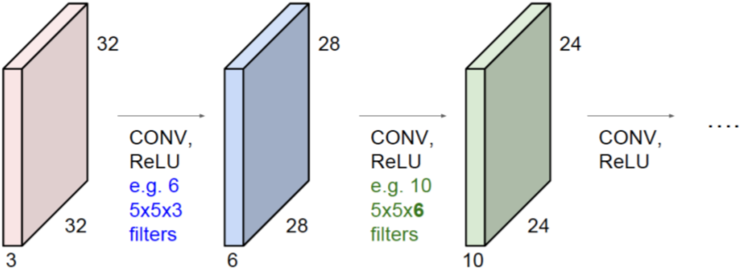
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

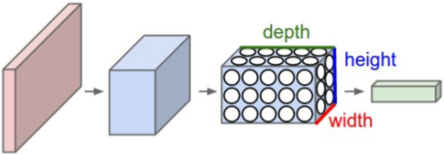
Li/Karpathy/Johnson, Stanford CS231n notes

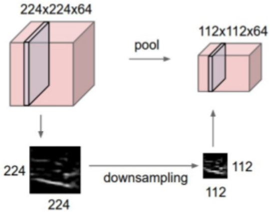
Convolutional Neural Network



Li/Karpathy/Johnson, Stanford CS231n notes

Convolutional Neural Network





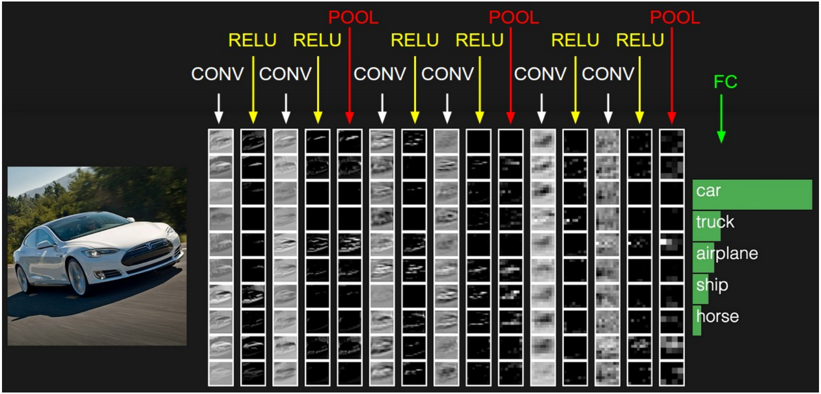
Convolutional layers

- Save more parameters than fully-connected layers
- Translation-invariant (useful for object recognition)
- Can use multiple sets of convolutional filters to extract features

Pooling layers:

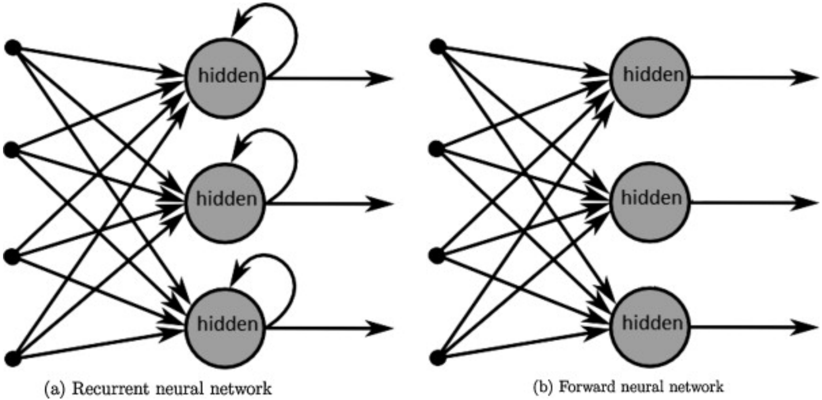
- Subsamples the (x,y) spatial dimensions

Li/Karpathy/Johnson, Stanford CS231n notes



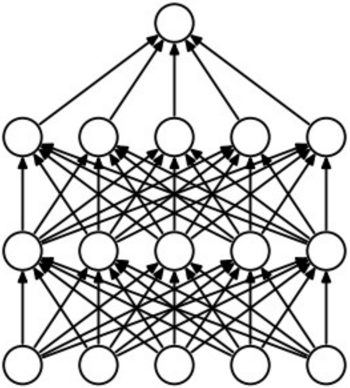
Li/Kaparthy/Johnson, Stanford CS231

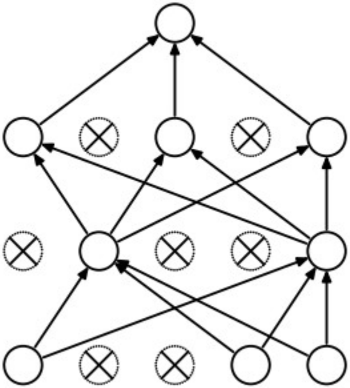
Recurrent Neural Networks



source

Regularization


(a) Standard Neural Net


(b) After applying dropout.

Li/Kaparthy/Johnson, Stanford CS231,Gal and Ghahramani (2015)

Contents

How to train Neural Networks?

- Suppose we are given a set $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ with known labels $y(\boldsymbol{x}_i) = y_i$. Training entails tuning parameters to minimize the error between y and the model's outputs for each $\boldsymbol{x}_i$.
- Cost functions represent a surrogate for classification error. We indirectly improve classification performance.
- In minimizing the cost function, gradient descent methods have become the go-to strategy. We will later discuss issues with this.
- What is the gradient of a NN w.r.t. its parameters?

Contents

Backpropagation

- Calculating the gradient was significant challenge until 1986 when Rumelhart *et al* proposed backpropagation.
- What was the problem before them?
  - Nested nonlinear activation functions and nontrivial cost functions are messy and require a lot of work for slight tweaks.
  - NNs with even a little depth involve several matrix multiplications. If one were to use a finite difference scheme, several forward passes through a network gets expensive.

Rumelhart et al. (1986)

Backpropagation

- Calculating the gradient was significant challenge until 1986 when Rumelhart *et al* proposed backpropagation.
- What was the problem before them?
  - Nested nonlinear activation functions and nontrivial cost functions are messy and require a lot of work for slight tweaks.
  - NNs with even a little depth involve several matrix multiplications. If one were to use a finite difference scheme, several forward passes through a network gets expensive.

Rumelhart et al. (1986)

Backpropagation

- Calculating the gradient was significant challenge until 1986 when Rumelhart *et al* proposed backpropagation.
- What was the problem before them?
  - Nested nonlinear activation functions and nontrivial cost functions are messy and require a lot of work for slight tweaks.
  - NNs with even a little depth involve several matrix multiplications. If one were to use a finite difference scheme, several forward passes through a network gets expensive.

Rumelhart et al. (1986)

Backpropagation

- As we will show, Backpropagation allows for us to compute the gradient of a NN at the cost of two forward passes regardless of the architecture.
- It plays mainly off of implementations of the chain rule.

Example Network: C cost function, L number of layers, input training $\boldsymbol{x} \in \mathbb{R}^m$ (n in total), $y(\boldsymbol{x})$ desired output

$$a_j^l(\boldsymbol{x}) = a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

Nielsen (2015)

Example Network: C cost function, L number of layers, input training $\boldsymbol{x} \in \mathbb{R}^m$ (n in total), $y(\boldsymbol{x})$ desired output

$$a_j^l(\boldsymbol{x}) = a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

C has two requirements: it can be written as a function of the network outputs $\boldsymbol{a}^L$ and arranged as a sum of cost functions for each training sample. Example:

$$C(\boldsymbol{a}^L) = \frac{1}{2n}\sum_{\boldsymbol{x}} ||\boldsymbol{y}(\boldsymbol{x}) - \boldsymbol{a}^L(\boldsymbol{x})||_2^2$$

Nielsen (2015)

Example Network: C cost function, L number of layers, input training $\boldsymbol{x} \in \mathbb{R}^m$ (n in total), $y(\boldsymbol{x})$ desired output

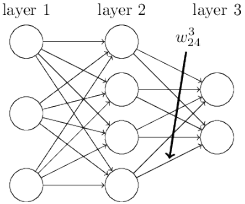
$$a_j^l(\boldsymbol{x}) = a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

$$a_j^1 = x_j$$

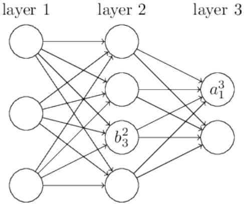
Nielsen (2015)

Example Network: C cost function, L number of layers, input training $\boldsymbol{x} \in \mathbb{R}^m$ (n in total), $y(\boldsymbol{x})$ desired output

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$



w_{jk}^l is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer



Nielsen (2015)

Example Network: C cost function, L number of layers, input training $\boldsymbol{x} \in \mathbb{R}^m$ (n in total), $y(\boldsymbol{x})$ desired output

$$a_j^l = \sigma\underbrace{\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)}_{\boldsymbol{z}^l}$$

Nielsen (2015)

Example Network: C cost function, L number of layers, input training $\boldsymbol{x} \in \mathbb{R}^m$ (n in total), $y(\boldsymbol{x})$ desired output

$$\boldsymbol{a}^l = \sigma\left(W^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l\right) = \sigma(\boldsymbol{z}^l)$$

Goal: find $\partial C / \partial w^l_{jk}$, $\partial C / \partial b^l_j$

Nielsen (2015)

$$\boldsymbol{a}^l = \sigma\left(W^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l\right) = \sigma(\boldsymbol{z}^l)$$

Let $\delta_j^l = \partial C / \partial z_j^l$

- Error in Output Layer

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{1}$$

-

$$\boldsymbol{\delta}^l = ((W^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot \sigma'(\boldsymbol{z}^l) \tag{2}$$

-

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{3}$$

-

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \tag{4}$$

$$\boldsymbol{a}^l = \sigma\left(W^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l\right) = \sigma(\boldsymbol{z}^l)$$

Let $\delta_j^l = \partial C / \partial z_j^l$

-
$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L) \tag{1}$$

- $\boldsymbol{\delta}^l$ in terms of $\boldsymbol{\delta}^{l+1}$

$$\boldsymbol{\delta}^l = ((W^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot \sigma'(\boldsymbol{z}^l) \tag{2}$$

-
$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{3}$$

-
$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{4}$$

$$\boldsymbol{a}^l = \sigma\left(W^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l\right) = \sigma(\boldsymbol{z}^l)$$

Let $\delta_j^l = \partial C / \partial z_j^l$

-
$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L) \tag{1}$$

-
$$\boldsymbol{\delta}^l = ((W^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot \sigma'(\boldsymbol{z}^l) \tag{2}$$

- Partial w.r.t. bias
$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{3}$$

-
$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{4}$$

$$\boldsymbol{a}^l = \sigma\left(W^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l\right) = \sigma(\boldsymbol{z}^l)$$

Let $\delta_j^l = \partial C / \partial z_j^l$

-
$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{1}$$

-
$$\boldsymbol{\delta}^l = ((W^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot \sigma'(\boldsymbol{z}^l) \tag{2}$$

-
$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{3}$$

- Partial w.r.t. weights
$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \tag{4}$$

Backpropagation Algorithm

1. Input $\boldsymbol{x}$ and solve for $\boldsymbol{a}^{(1)}$
2. Feedforward through the network, i.e. for $l = 2, \ldots, L$ find

$$\boldsymbol{z}^l = W^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l, \quad \boldsymbol{a}^l = \sigma(\boldsymbol{z}^l)$$

3. Find $\delta^L = \nabla_{\boldsymbol{a}^L} C \odot \sigma(\boldsymbol{z}^L)$
4. Backpropagate error by, for $l = L-1, L-2, \ldots, 2$, calculating

$$\delta^l = ((W^{l+1})^T \delta^{l+1}) \odot \sigma'(\boldsymbol{z}^l)$$

5. Gradients of Cost Function are found as

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l, \quad \frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

Nielsen (2015)

Backpropagation Algorithm

1. Input $\boldsymbol{x}$ and solve for $\boldsymbol{a}^{(1)}$
2. Feedforward through the network, i.e. for $l = 2, \ldots, L$ find

$$\boldsymbol{z}^l = W^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l, \;\; \boldsymbol{a}^l = \sigma(\boldsymbol{z}^l)$$

3. Find $\delta^L = \nabla_{\boldsymbol{a}^L} C \odot \sigma(\boldsymbol{z}^L)$
4. Backpropagate error by, for $l = L-1, L-2, \ldots, 2$, calculating

$$\delta^l = ((W^{l+1})^T \delta^{l+1}) \odot \sigma'(\boldsymbol{z}^l)$$

5. Gradients of Cost Function are found as

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l, \;\; \frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$$

Nielsen (2015)

Backpropagation Algorithm

1. Input $\boldsymbol{x}$ and solve for $\boldsymbol{a}^{(1)}$
2. Feedforward through the network, i.e. for $l = 2, \ldots, L$ find
$$\boldsymbol{z}^l = W^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l, \;\; \boldsymbol{a}^l = \sigma(\boldsymbol{z}^l)$$
3. Find $\delta^L = \nabla_{\boldsymbol{a}^L} C \odot \sigma(\boldsymbol{z}^L)$
4. Backpropagate error by, for $l = L-1, L-2, \ldots, 2$, calculating
$$\delta^l = ((W^{l+1})^T \delta^{l+1}) \odot \sigma'(\boldsymbol{z}^l)$$
5. Gradients of Cost Function are found as
$$\frac{\partial C}{\partial b^l_j} = \delta^l_j, \;\; \frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

Nielsen (2015)

Backpropagation Algorithm

1. Input $\boldsymbol{x}$ and solve for $\boldsymbol{a}^{(1)}$
2. Feedforward through the network, i.e. for $l = 2, \ldots, L$ find

$$\boldsymbol{z}^l = W^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l, \;\; \boldsymbol{a}^l = \sigma(\boldsymbol{z}^l)$$

3. Find $\boldsymbol{\delta}^L = \nabla_{\boldsymbol{a}^L} C \odot \sigma(\boldsymbol{z}^L)$
4. Backpropagate error by, for $l = L-1, L-2, \ldots, 2$, calculating

$$\boldsymbol{\delta}^l = ((W^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot \sigma'(\boldsymbol{z}^l)$$

5. Gradients of Cost Function are found as

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l, \;\; \frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

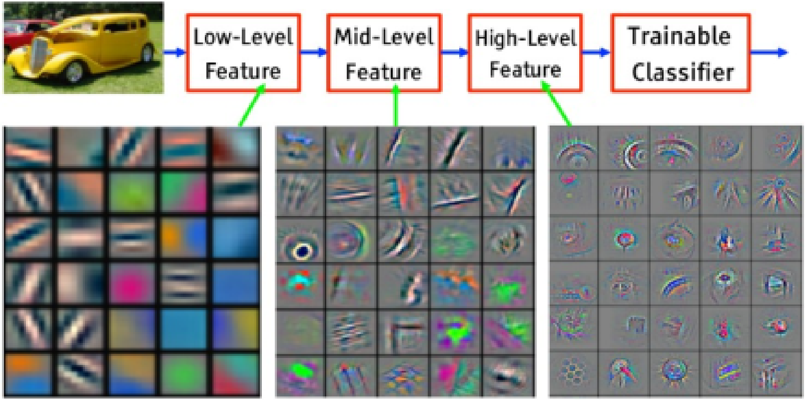
Nielsen (2015)

Backpropagation Algorithm

1. Input $\boldsymbol{x}$ and solve for $\boldsymbol{a}^{(1)}$
2. Feedforward through the network, i.e. for $l = 2, \ldots, L$ find
$$\boldsymbol{z}^l = W^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l, \;\; \boldsymbol{a}^l = \sigma(\boldsymbol{z}^l)$$
3. Find $\boldsymbol{\delta}^L = \nabla_{\boldsymbol{a}^L} C \odot \sigma(\boldsymbol{z}^L)$
4. Backpropagate error by, for $l = L-1, L-2, \ldots, 2$, calculating
$$\boldsymbol{\delta}^l = ((W^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot \sigma'(\boldsymbol{z}^l)$$
5. Gradients of Cost Function are found as
$$\frac{\partial C}{\partial b_j^l} = \delta_j^l, \;\; \frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

Nielsen (2015)

Learned Filters



Courtesy Yann LeCun

Contents

Gradient Descent

- Backpropagation → gradient
- (Batch) gradient descent:

$$W^l = W^l - \alpha_W \frac{\partial C}{\partial W^l}$$

$$\boldsymbol{b} = \boldsymbol{b}^l - \alpha_b \frac{\partial C}{\partial \boldsymbol{b}^l}$$

- Issue: C is the sum of costs associated with each training sample.
- Many problems use millions of training samples...

Gradient Descent

- Backpropagation $\rightarrow$ gradient
- (Batch) gradient descent:

$$W^l = W^l - \alpha_W \frac{\partial C}{\partial W^l}$$
$$\boldsymbol{b} = \boldsymbol{b}^l - \alpha_b \frac{\partial C}{\partial \boldsymbol{b}^l}$$

- Issue: C is the sum of costs associated with each training sample.
- Many problems use millions of training samples...

Stochastic Gradient Descent

- Randomly choose $m < n$ training samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$

$$C^{(m)} = \sum_{i=1}^{m} C_{x_i}$$

- Update step:

$$W_{k+1}^{l} = W_{k}^{l} - \frac{n\alpha_W}{m} \sum_{i=1}^{m} \frac{\partial C_{x_i}}{\partial W_k^l}$$

$$\boldsymbol{b}_{k+1} = \boldsymbol{b}^{l} - \frac{n\alpha_b}{m} \sum_{i=1}^{m} \frac{\partial C_{x_i}}{\partial \boldsymbol{b}_k^l}$$

Stochastic Gradient Descent

- Randomly choose $m < n$ training samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$

$$C^{(m)} = \sum_{i=1}^{m} C_{x_i}$$

$$W_{k+1}^{l} = W_{k}^{l} - \alpha_W \underbrace{\frac{n}{m} \sum_{i=1}^{m} \frac{\partial C_{x_i}}{\partial W_{k}^{l}}}_{\boldsymbol{g}_W}$$

$$\mathbb{E}[\boldsymbol{g}_W] = \nabla_W C$$

Stochastic Gradient

- Randomly choose $m < n$ training samples $\boldsymbol{x}_1, \dots, \boldsymbol{x}_m$

$$C^{(m)} = \sum_{i=1}^{m} C_{x_i}$$

$$W_{k+1}^{l} = W_k^l - \alpha_W \frac{n}{m} \sum_{i=1}^{m} \frac{\partial C_{x_i}}{\partial W_k^l}$$

- Learning rate α_W is typically fixed and "small" (but not "too small").

Stochastic Gradient "Descent"

- Randomly choose $m < n$ training samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$

$$C^{(m)} = \sum_{i=1}^{m} C_{x_i}$$

$$W_{k+1}^{l} = W_k^l - \alpha_W \frac{n}{m} \sum_{i=1}^{m} \frac{\partial C_{x_i}}{\partial W_k^l}$$

- Learning rate α_W is typically fixed and "small" (but not "too small").

Stochastic Gradient Descent

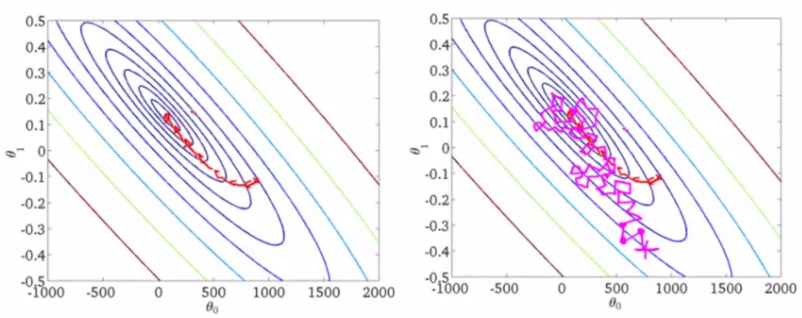


Figure: GD on left, SGD on right

Ng, Standford Machine Learning (Fall 2011)

Stochastic Gradient Descent

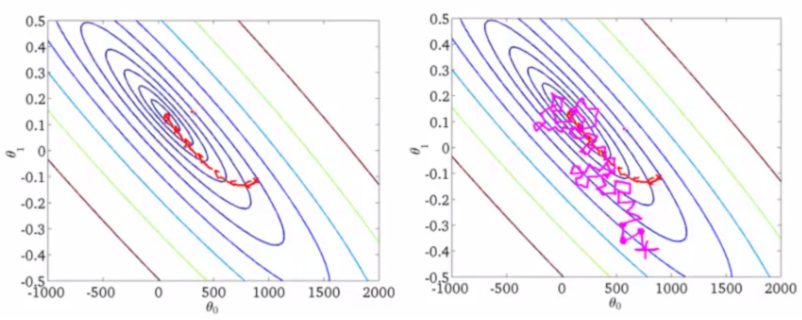
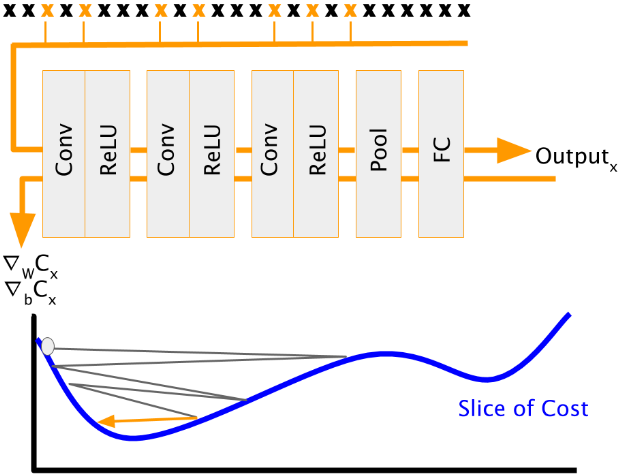


Figure: GD on left, SGD on right

Still acceptable - we just want “close enough”

Ng, Standford Machine Learning (Fall 2011)

Summarize Training



Quick Note on C

- Least squares is useful for examples, but usually not suitable for practice (slow with sigmoid output).
- Cross-entropy

$$C = \frac{1}{n} \sum_{\boldsymbol{x}} \boldsymbol{y}^T \ln(\boldsymbol{a}^L(\boldsymbol{x})) + (\mathbb{1} - \boldsymbol{y})^T \ln(\mathbb{1} - \boldsymbol{a}^L(\boldsymbol{x}))$$

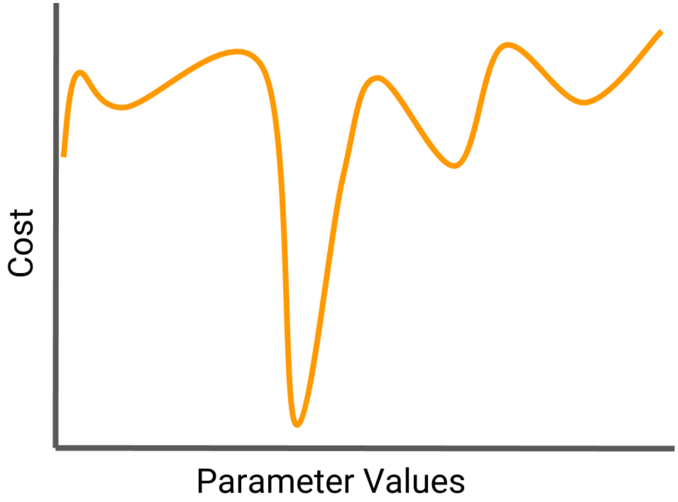
- Chosen for “nice” attributes, but no idea of resulting topology, which influences everything.

Contents

Optimization Issues

- Training for NNs remains one of the least analytically sound aspects.
- What does the cost function look like?
- Where are its local minima (if there are any)?
- How will SGD jump around?

Local Minima Problem

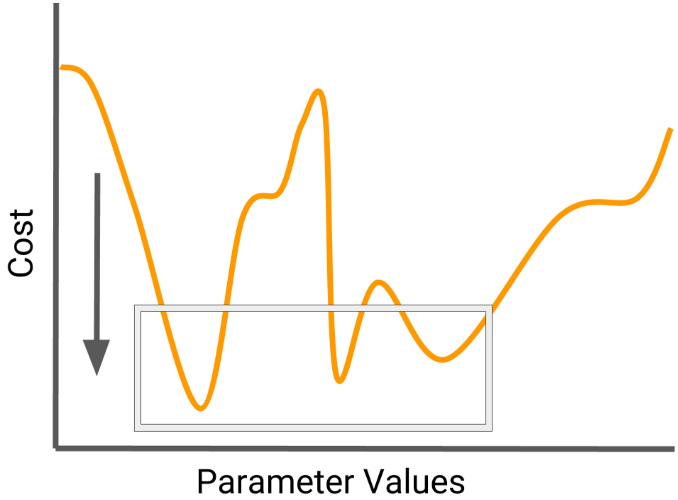


Sontag and Sussmann (1989),Brady et al. (1989),Gori and Tesi (1992)

Saddle Points

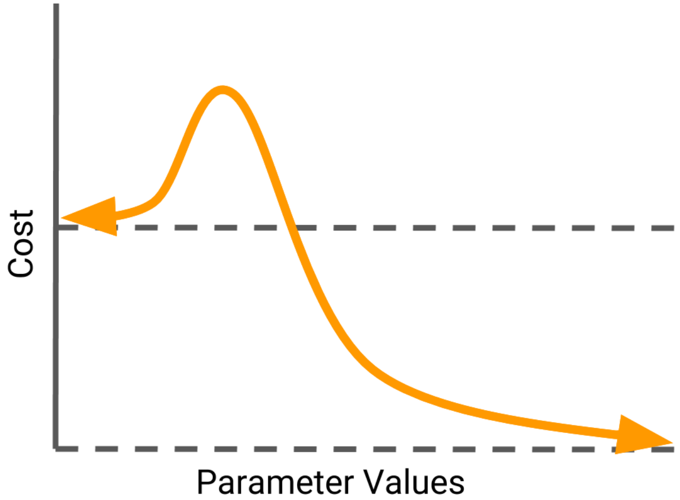
- Suppose $H = \nabla^2 C$ and λ_i its eigenvalues.
- $P(\lambda_i > 0) = \frac{1}{2} \Longrightarrow P(\lambda_1 > 0, \ldots, \lambda_K > 0) = (\frac{1}{2})^K$
- $(\frac{1}{2})^K \to 0$ as $K \to \infty$, meaning the probability that a critical point is a saddle point.

Dauphin et al. (2014)



Sontag and Sussmann (1989),Brady et al. (1989),Gori and Tesi (1992)

No Minima Problem



In Practice. . .

- Even in cases where local minimum is not found, results can be state-of-the-art; just want SGD to find “very small value.”
- Initialization of weights is a major area of research to try to place model in “good” area to find local minima (more on this later).
- Structural elements have adapted to (experimentally) fix these issues (ReLU, cross-entropy cost, convolutional layers, etc.).

Goodfellow et al. (2016)

Unstable Gradient vs. Architecture

Layers (Parameters*)	Mutli-Class Error	Top-5 Error
11 (133)	29.6	10.4
13 (133)	28.7	9.9
16 (134)	28.1	9.4
16 (138)	27.3	8.8
19 (144)	25.5	8.0

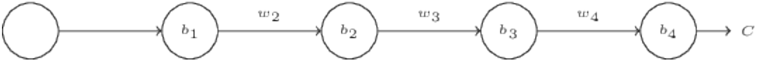
* Number in millions

Simonyan and Zisserman (2014)

Unstable Gradient

- Consider simplified model

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$

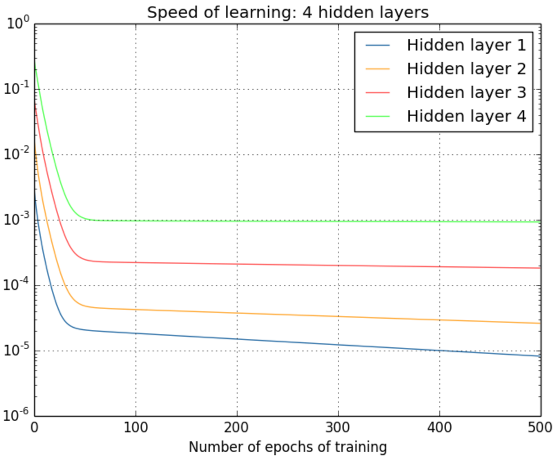


- Typical initialization: $w_i \sim \mathcal{N}(0, 1)$, $b_i = 1$

Nielsen (2015)

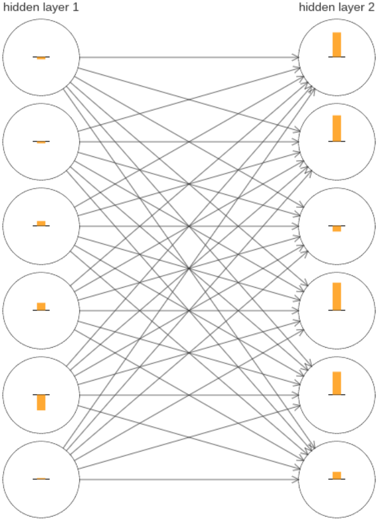
Vanishing Gradient

- First layers learn slower than later ones



Nielsen (2015)

Unstable Gradients for FFN



Nielsen (2015)

Overcoming Unstable Gradients

- Initialization (again)
  1. Random Walk Initialization[1]

$$\widetilde{W}^{\ell} = \omega W^{\ell} \text{ where } \omega_{\text{ReLU}} = (\sqrt{2}) \exp\left(\frac{1.2}{\max(N^{\ell}, 6) - 2.4}\right)$$

  2. Orthonormal Initialization[2]
  3. Unit Variance Initilization[3,4]

Sussillo and Abbott (2014)[1],Saxe et al. (2013)[2],He et al. (2015)[3],Mishkin and Matas (2015)[4]

Overcoming Unstable Gradients

- Student-teacher model
- Drop in layers as you go
- Change the cost function and/or activiation functions?

Romero et al. (2014)

Contents

Transfer Learning & Optimization Problems

- Transfer learning: taking a model primed for task X and applying to unrelated problem Y. For example, taking a CNN designed to discern amongst vehicles and applying it to classifying different animals without manipulating parameters.
- Transfer learning and its predecessor pretraining are unintentional solutions to the initialization problems.

Contents

Pretraining

- Pretraining revived deep neural networks from obscurity.
- Without convolutions or recursion, deep full connected networks could be trained and avoid local minima problems.
- Unsupervised pretraining has mostly been abandoned, but it inspired much of the modern work in transfer learning.

Bengio et al. (2007)

Unsupervised Pretraining

Let f be the identity function, X input data matrix (1 row per example), K number of iterations

for $k \in \{1, \ldots, K\}$

$$f^{(k)} = \mathcal{L}(X)$$

$$f = f^{(k)} \circ f$$

$$X = f^{(k)}(X)$$

The above is done for each layer. The input is the previous layer's output.

Goodfellow et al. (2016)

Unsupervised Pretraining

Unsupervised pretraining is thought to work because:

1. A model is sensitive to its initializations.
2. Learning the input distribution is useful.

Goodfellow et al. (2016)

Unsupervised Pretraining

Unsupervised pretraining is thought to work because:

1. A model is sensitive to its initializations.
   - This is the least mathematically understood part of pretraining.
   - It may start the model at local minima otherwise inaccessible based on the shape of the cost function.
   - How much of the initialization survives training?
2. Learning the input distribution is useful.

Goodfellow et al. (2016)

Unsupervised Pretraining

Unsupervised pretraining is thought to work because:

1. A model is sensitive to its initializations.
2. Learning the input distribution is useful.
   - Suppose car/motorcycle model; will pretraining spot wheels? Will its representation of a wheel be helpful?
   - There is no coherent theory as to if/how/why/when unsupervised features help with NN training.

Goodfellow et al. (2016)

Supervised Pretraining

$$\boldsymbol{x} \rightarrow \sigma(W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) \rightarrow y(\boldsymbol{x})$$

Bengio et al. (2007)

Supervised Pretraining

$$\underbrace{\boldsymbol{x} \to \sigma(W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})}_{\boldsymbol{x}^{(1)}} \to y(\boldsymbol{x})$$

$$\boldsymbol{x}^{(1)} \to \sigma(W^{(2)}\boldsymbol{x}^{(1)} + \boldsymbol{b}^{(2)}) \to y(\boldsymbol{x})$$

Bengio et al. (2007)

Supervised Pretraining

$$\underbrace{\boldsymbol{x} \to \sigma(W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})}_{\boldsymbol{x}^{(1)}} \to y(\boldsymbol{x})$$

$$\boldsymbol{x}^{(1)} \to \sigma(W^{(2)}\boldsymbol{x}^{(1)} + \boldsymbol{b}^{(2)}) \to y(\boldsymbol{x})$$

$$\vdots$$

$$\boldsymbol{x} \to \sigma(W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) \to \sigma(W^{(2)}\boldsymbol{a}^{(1)} + \boldsymbol{b}^{(2)}) \to \cdots \to y(\boldsymbol{x})$$

Bengio et al. (2007)

Contents

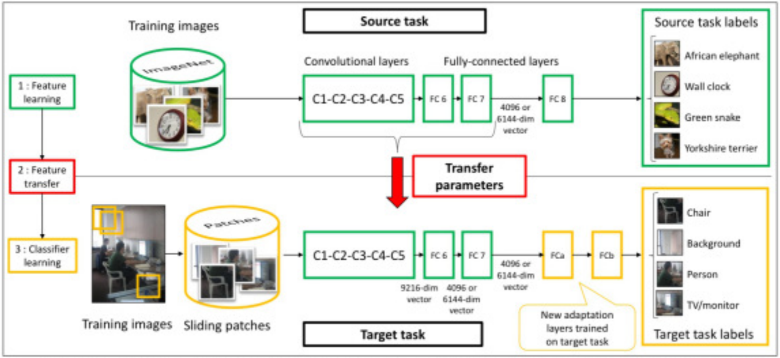
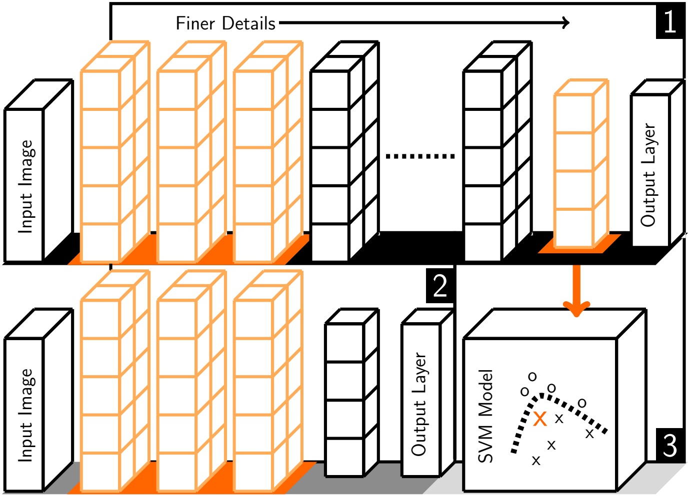
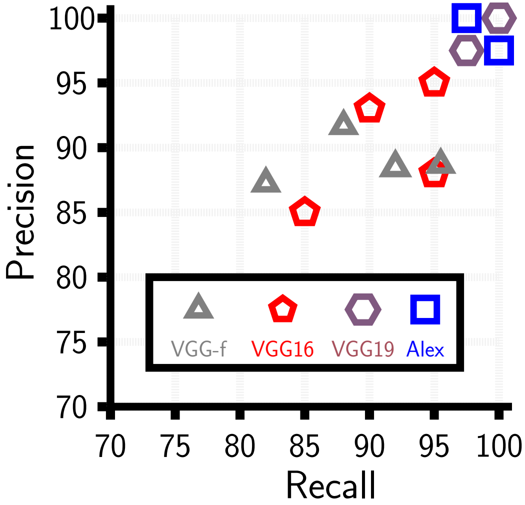


Figure 2: **Transferring parameters of a CNN.** First, the network is trained on the source task (ImageNet classification, top row) with a large amount of available labelled images. Pre-trained parameters of the internal layers of the network (C1-FC7) are then transferred to the target tasks (Pascal VOC object or action classification, bottom row). To compensate for the different image statistics (type of objects, typical viewpoints, imaging conditions) of the source and target data we add an adaptation layer (fully connected layers FCa and FCb) and train them on the labelled data of the target task.

Oquab et al. (2014)


Finer Details
1
Input Image
Output Layer
2
Input Image
Output Layer
SVM Model
o
o
o
o
X
x
x
x
x
3

McKay 2017



Precision
Recall
100
95
90
85
80
75
70
70
75
80
85
90
95
100
VGG-f
VGG16
VGG19
Alex

McKay 2017

Thank You

Acknolwedgements

- Dr. Anne Gelb
- Dr. Suren Jayasuriya
- iPal NN group: Tiep Vu, Hojjat Mousavi, Tiantong Guo

References I

Bengio, Y. et al. (2007). "Greedy layer-wise training of deep networks". In: *IN NIPS.*

Brady, M. L., R. Raghavan, and J. Slawny (1989). "Back propagation fails to separate where perceptrons succeed". In: *IEEE Transactions on Circuits and Systems* 36.5,

Dauphin, Y. N. et al. (2014). "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". In: *Advances in neural information processing systems*,

Gal, Y. and Z. Ghahramani (2015). "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning". In: *arXiv preprint arXiv:1506.02142* 2.

Gatys, L. A., A. S. Ecker, and M. Bethge (2015). "A neural algorithm of artistic style". In: *arXiv preprint arXiv:1508.06576.*

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press.

Gori, M. and A. Tesi (1992). "On the problem of local minima in backpropagation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.1,

He, K. et al. (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*,

Hornik, K., M. Stinchcombe, and H. White (1989). "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5,

References II

Krause, J. et al. (2016). "A Hierarchical Approach for Generating Descriptive Image Paragraphs". In: *arXiv preprint arXiv:1611.06607*.

LeCun, Y. et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11,

Leshno, M. et al. (1993). "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function". In: *Neural networks* 6.6,

Lowe, D. G. (1999). "Object recognition from local scale-invariant features". In: *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. Vol. 2. Ieee,

Mishkin, D. and J. Matas (2015). "All you need is a good init". In: *arXiv preprint arXiv:1511.06422*.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. `http://neuralnetworksanddeeplearning.com`. Determination Press.

Oquab, M. et al. (2014). "Learning and transferring mid-level image representations using convolutional neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*,

Romero, A. et al. (2014). "Fitnets: Hints for thin deep nets". In: *arXiv preprint arXiv:1412.6550*.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). "Learning representations by back-propagating errors". In: *Nature* 323,

References III

Saxe, A. M., J. L. McClelland, and S. Ganguli (2013). "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks". In: *arXiv preprint arXiv:1312.6120.*

Simonyan, K. and A. Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556.*

Sontag, E. D. and H. J. Sussmann (1989). "Backpropagation can give rise to spurious local minima even for networks without hidden layers". In: *Complex Systems* 3.1,

Sussillo, D. and L. Abbott (2014). "Random walk initialization for training very deep feedforward networks". In: *arXiv preprint arXiv:1412.6558.*

Taigman, Y. et al. (2014). "Deepface: Closing the gap to human-level performance in face verification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,*