Introduction to Neural Networks and Deep Learning

Deep Forward Neural Networks

Andres Mendez-Vazquez

August 22, 2020

Outline

Outline

For this initial analysis

We will look at the paper by Bengio

- "Learning deep architectures for AI", Foundations and trends in Machine Learning 2, 1 (2009), pp. 1--127.

And for this, we will look at Boolean functions

- After Shanon pointed out the fact they are useful to represent complex problems [1].

For this initial analysis

We will look at the paper by Bengio

- "Learning deep architectures for AI", Foundations and trends in Machine Learning 2, 1 (2009), pp. 1--127.

And for this, we will look at Boolean functions

- After Shanon pointed out the fact they are useful to represent complex problems [1].

Architecture

A two-layer circuit of logic gates can represent any boolean function [2]

- Any boolean function can be written as a sum of products, disjunctive normal form:
  - AND gates on the first layer with optional negation of inputs,
  - And OR gate on the second layer

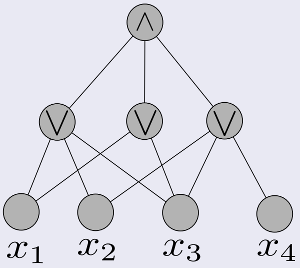
Example

Architecture

A two-layer circuit of logic gates can represent any boolean function [2]

- Any boolean function can be written as a sum of products, disjunctive normal form:
  - AND gates on the first layer with optional negation of inputs,
  - And OR gate on the second layer

Example



The Exponential Width

Here, we have a small problem

- There are functions computable with a polynomial-size logic gates circuit of depth k that require **exponential size** when restricted to depth $k-1$[3]
  - For Example

$$parity : (b_1, ..., b_d) \in \{0,1\}^d \mapsto \begin{cases} 1 & \text{if } \sum_{i=1}^{d} b_i \text{ is even} \\ -1 & \text{otherwise} \end{cases}$$

How this impact shallow learning in Machine Learning?

- Many of the results for boolean circuits can be generalized to architectures whose computational elements are linear threshold units

$$f(x) = 1_{wx+b>0}$$

  - The fan-in of a circuit is the maximum number of inputs of a particular element.

The Exponential Width

Here, we have a small problem

- There are functions computable with a polynomial-size logic gates circuit of depth k that require **exponential size** when restricted to depth $k-1$[3]
  - For Example

$$parity : (b_1, ..., b_d) \in \{0,1\}^d \mapsto \begin{cases} 1 & \text{if } \sum_{i=1}^{d} b_i \\ -1 & \text{otherwise} \end{cases} \text{ is even}$$

How this impact shallow learning in Machine Learning?

- Many of the results for boolean circuits can be generalized to architectures whose computational elements are linear threshold units

$$f(x) = 1_{wx+b>0}$$

  - The fan-in of a circuit is the maximum number of inputs of a particular element.

Therefore

How this impact shallow learning in Machine Learning?

- First, we define the concept of f_k function

Definition

- The function f_k is a function of N^{2k-2} variables. It is defined by a depth k circuit that is a tree. At the leaves of the tree there are unnegated variable. The i^{th} level from the bottom consists of $\wedge$-gates if i is even and otherwise it consists of $\vee$-gates.

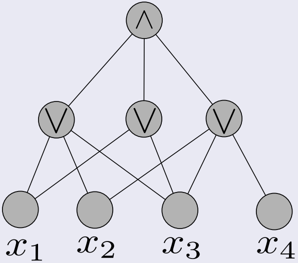
Therefore

How this impact shallow learning in Machine Learning?

- First, we define the concept of f_k function

Definition

- The function f_k is a function of N^{2k-2} variables. It is defined by a depth k circuit that is a tree. At the leaves of the tree there are unnegated variable, The i^{th} level from the bottom consists of $\wedge$-gates if i is even and otherwise it consists of $\vee$-gates.



An Important Theorem

Of particular interest is the following theorem

- Monotone weighted threshold circuits (i.e. multi-layer neural networks with linear threshold units and positive weights)

Theorem [4]

- A monotone weighted threshold circuit of depth $k - 1$ computing a function f_k has size at least 2^{cN} for some constant $c > 0$ and $N > N_0$.

An Important Theorem

Of particular interest is the following theorem

- Monotone weighted threshold circuits (i.e. multi-layer neural networks with linear threshold units and positive weights)

Theorem [4]

- A monotone weighted threshold circuit of depth $k-1$ computing a function f_k has size at least 2^{cN} for some constant $c > 0$ and $N > N_0$.

Meaning

This theorem does not fail any type of architecture

- But the question arises, Are the depth 1, 2 and 3 architectures (many Machine Learning algorithms) too shallow to represent efficiently more complicated functions?

What happens in Deep Architectures

- Bengio et al. argues that they can represent highly-varying functions [5]

Meaning

This theorem does not fail any type of architecture

- But the question arises, Are the depth 1, 2 and 3 architectures (many Machine Learning algorithms) too shallow to represent efficiently more complicated functions?

What happens in Deep Architectures

- Bengio et al. argues that they can represent highly-varying functions [5]

Outline

Highly-varying functions

Meaning

- We say that a function is highly-varying when a piecewise approximation of that function would require a large number of pieces.

Clearly

- Deeper Architectures can handle such functions in a easier way than shallow ones.

For Example

- The polynomial $\prod_{i=1}^{n} \sum_{j=1}^{m} a_{ij} x_j$ can be represented as a product of sums with only $O(nm)$ elements

Highly-varying functions

Meaning

- We say that a function is highly-varying when a piecewise approximation of that function would require a large number of pieces.

Clearly

- Deeper Architectures can handle such functions in a easier way than shallow ones.

For Example

- The polynomial $\prod_{i=1}^{n}\sum_{j=1}^{m} a_{ij}x_j$ can be represented as a product of sums with only $O(nm)$ elements

Highly-varying functions

Meaning

- We say that a function is highly-varying when a piecewise approximation of that function would require a large number of pieces.

Clearly

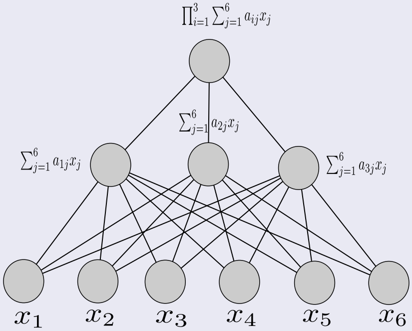
- Deeper Architectures can handle such functions in a easier way than shallow ones.

For Example

- The polynomial $\prod_{i=1}^{n}\sum_{j=1}^{m} a_{ij}x_j$ can be represented as a product of sums with only $O(nm)$ elements

Basically

We have a Perceptron Layer and a Product Second Layer

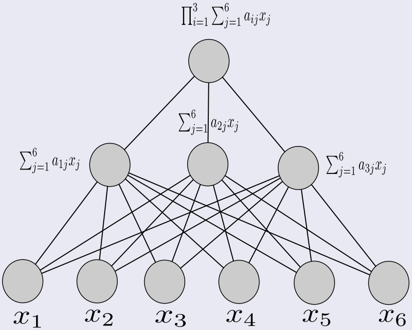


What if I do a product of sums

- What will happen?

Basically

We have a Perceptron Layer and a Product Second Layer



What if I do a product of sums

- What will happen?

Ok, we have a problem

Because for our case

$$\prod_{i=1}^{3}\sum_{j=1}^{6}a_{ij}x_j=\sum_{j=1}^{6}\prod_{i=1}^{3}a_{ij}x_j$$

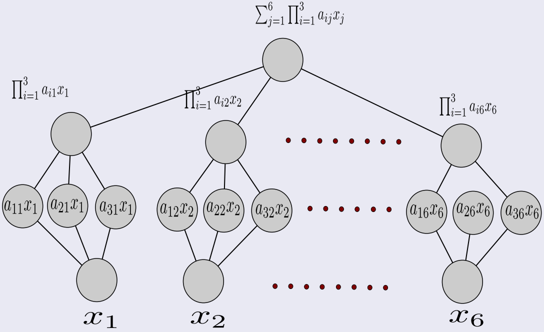
We have the following problem $O\left(n^m\right)$

Ok, we have a problem

Because for our case

$$\prod_{i=1}^{3}\sum_{j=1}^{6} a_{ij}x_j = \sum_{j=1}^{6}\prod_{i=1}^{3} a_{ij}x_j$$

We have the following problem $O(n^m)$



Actually

You could claim

- Machine Learning shallow learning depends on complex computational units to handle complex functions

Deep Learning

- Proposes simpler units but deeper structures to handle complex functions

What about both ideas together

- Complex adaptive units
- Deeper architectures to helps such units
  - It seems to be the case of the human brain...!!!

Actually

You could claim

- Machine Learning shallow learning depends on complex computational units to handle complex functions

Deep Learning

- Proposes simpler units but deeper structures to handle complex functions

What about both ideas together

- Complex adaptive units
- Deeper architectures to helps such units
  - It seems to be the case of the human brain...!!!

Actually

You could claim

- Machine Learning shallow learning depends on complex computational units to handle complex functions

Deep Learning

- Proposes simpler units but deeper structures to handle complex functions

What about both ideas together

- Complex adaptive units
- Deeper architectures to helps such units
  - It seems to be the case of the human brain...!!!

Outline

Local vs Non-Local Generalization

Something Notable

- A local estimator partitions the input space in regions

Thus, local estimators are based on matching local templates

- It can be thought of as having two levels...

The first level

- It is made of a set of templates which can be matched to the input.

Local vs Non-Local Generalization

Something Notable

- A local estimator partitions the input space in regions

Thus, local estimators are based on matching local templates

- It can be thought of as having two levels...

The first level

- It is made of a set of templates which can be matched to the input.

Local vs Non-Local Generalization

Something Notable

- A local estimator partitions the input space in regions

Thus, local estimators are based on matching local templates

- It can be thought of as having two levels...

The first level

- It is made of a set of templates which can be matched to the input.

Then

A template unit will output a value that indicates the degree of matching

$$K\left(x|\Theta\right)$$

The second level combines these values

- Typically a simple linear combination or product combination

$$L\left(x\right)=\sum_{i=1}^{k}K\left(x|\Theta_{i}\right)$$

Classic Example, the kernel machine

$$f\left(x\right)=b+\sum_{i=1}^{k}\alpha_{i}K\left(x,x_{i}\right)$$

Then

A template unit will output a value that indicates the degree of matching

$$K\left(x|\Theta\right)$$

The second level combines these values

- Typically a simple linear combination or product combination

$$L\left(x\right)=\sum_{i=1}^{k}K\left(x|\Theta_i\right)$$

Classic Example, the kernel machine

$$f\left(x\right)=b+\sum_{i=1}^{k}\alpha_i K\left(x,x_i\right)$$

Then

A template unit will output a value that indicates the degree of matching

$$K\left(x|\Theta\right)$$

The second level combines these values

- Typically a simple linear combination or product combination

$$L\left(x\right)=\sum_{i=1}^{k}K\left(x|\Theta_i\right)$$

Classic Example, the kernel machine

$$f\left(x\right)=b+\sum_{i=1}^{k}\alpha_iK\left(x,x_i\right)$$

As you can see

The Kernel has a local influence based on the support vectors

- For example the Gaussian Kernel

$$K\left(x, x_i\right) = \exp\left\{-\frac{\left\|x - x_i\right\|^2}{\sigma^2}\right\}$$

The Problem of Kernel

- The assumption that the target function is smooth or can be well approximated with a smooth function.

The limitations of a fixed generic kernel such as the Gaussian kernel

- They have motivated a lot of research in designing kernels [6, 7]

As you can see

The Kernel has a local influence based on the support vectors

- For example the Gaussian Kernel

$$K\left(x, x_i\right) = \exp\left\{-\frac{\left\|x - x_i\right\|^2}{\sigma^2}\right\}$$

The Problem of Kernel

- The assumption that the target function is smooth or can be well approximated with a smooth function.

The limitations of a fixed generic kernel such as the Gaussian kernel

- They have motivated a lot of research in designing kernels [6, 7]

As you can see

The Kernel has a local influence based on the support vectors

- For example the Gaussian Kernel

$$K(x, x_i) = \exp\left\{-\frac{\|x - x_i\|^2}{\sigma^2}\right\}$$

The Problem of Kernel

- The assumption that the target function is smooth or can be well approximated with a smooth function.

The limitations of a fixed generic kernel such as the Gaussian kernel

- They have motivated a lot of research in designing kernels [6, 7]

For Example, in supervised learning

If we have the training example (x_i, y_i)

- We want to build predictor that output something near y_i when any other sample is near x_i

Basically the situation when regularizing

- Bengio and Le Cun claim this is not enough [8, 9]

Although, It is possible to argue

- That such highly varying space is due to a lack of the correct feature selection process.

For Example, in supervised learning

If we have the training example (x_i, y_i)

- We want to build predictor that output something near y_i when any other sample is near x_i

Basically the situation when regularizing

- Bengio and Le Cun claim this is not enough [8, 9]

Although, It is possible to argue

- That such highly varying space is due to a lack of the correct feature selection process.

For Example, in supervised learning

If we have the training example (x_i, y_i)

- We want to build predictor that output something near y_i when any other sample is near x_i

Basically the situation when regularizing

- Bengio and Le Cun claim this is not enough [8, 9]

Although, It is possible to argue

- That such highly varying space is due to a lack of the correct feature selection process.

However

If you look at the parity problem

$$parity : (b_1, ..., b_d) \in \{0,1\}^d \mapsto \begin{cases} 1 & \text{if } \sum_{i=1}^{d} b_i \text{ is even} \\ -1 & \text{otherwise} \end{cases}$$

Theorem

- Let $f(x) = b + \sum_{i=1}^{2^d} \alpha_i K(x_i, x)$ be an affine combination of Gaussian with the same width σ centered on points $x_i \in \{-1,1\}^d$. If f solve the parity problem, then there are at least 2^{d-1} non-zero support vectors.

However

If you look at the parity problem

$$parity : (b_1, ..., b_d) \in \{0,1\}^d \mapsto \begin{cases} 1 & \text{if } \sum_{i=1}^{d} b_i \text{ is even} \\ -1 & \text{otherwise} \end{cases}$$

Theorem

- Let $f(\boldsymbol{x}) = b + \sum_{i=1}^{2^d} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x})$ be an affine combination of Gaussian with the same width σ centered on points $\boldsymbol{x}_i \in \{-1, 1\}^d$. If f solve the parity problem, then there are at least 2^{d-1} non-zero support vectors.

However

Although, this function is not a representative

- The kind of functions we are more interested in AI.

It suggest that local based estimators

- They are not enough, but still not a conclusive result

After all

- More Memory could be added to those systems

However

Although, this function is not a representative

- The kind of functions we are more interested in AI.

It suggest that local based estimators

- They are not enough, but still not a conclusive result

After all

- More Memory could be added to those systems

However

Although, this function is not a representative

- The kind of functions we are more interested in AI.

It suggest that local based estimators

- They are not enough, but still not a conclusive result

After all

- More Memory could be added to those systems

For example

Tensors have been used to add memory to SVM

$$\min_{\boldsymbol{U}_i^{(m)}, \boldsymbol{K}^{(m)}, \boldsymbol{\beta}, b} \gamma \sum_{i=1}^{N} \left\| \mathcal{X}_i - \left[\!\left[\boldsymbol{K}^{(1)} \boldsymbol{U}_i^{(1)}, \cdots, \boldsymbol{K}^{(M)} \boldsymbol{U}_i^{(M)} \right]\!\right] \right\|_F^2 + \cdots$$

$$\lambda \boldsymbol{\beta}^T \widehat{\boldsymbol{K}} \boldsymbol{\beta} + \sum_{i=1}^{N} \left[1 - y_i \left(\widehat{\boldsymbol{k}}_i^T \boldsymbol{\beta} + b \right) \right]_+$$

- $\boldsymbol{K}^{(m)}$ are kernel matrices defined on each mode to capture the nonlinear part.
- $\boldsymbol{U}^{(m)} = \left[\boldsymbol{u}_1^{(m)}, \ldots, \boldsymbol{u}_R^{(m)} \right]$ are factor matrices of size $I_m \times R_m$

However

A Problem

- You are limiting the Machine Learning operations to matrix additions and products and non-linear operations.
  - In a shallow way...

We need to add more complex functions

- After all deeper architectures construct complex functions layer by layer

However

A Problem

- You are limiting the Machine Learning operations to matrix additions and products and non-linear operations.
  - In a shallow way...

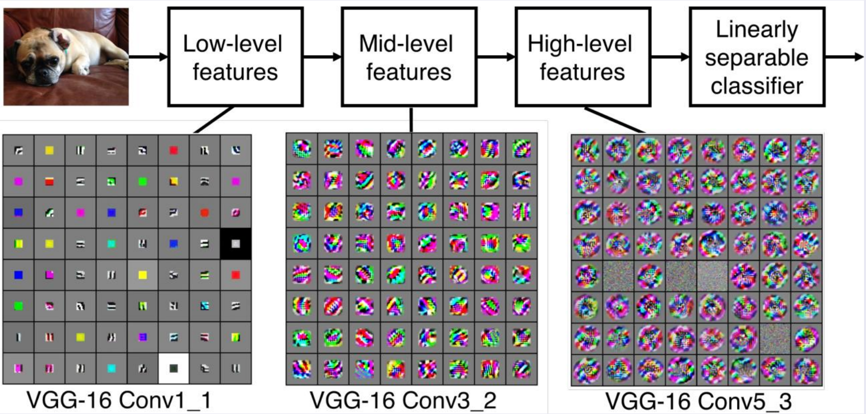
We need to add more complex functions

- After all deeper architectures construct complex functions layer by layer

Outline

By Using Weights in Certain Deep Learners

The Application of each Layer increase the complexity of the features



Outline

Some of the Models to be Reviewed of Models

Convolutional Neural Networks

- The classic model that started the phenomena of Neural Networks.

Auto Encoder

- How to generate novel features by funneling.

Boltzmann Machine

- Energy Based Models.

Some of the Models to be Reviewed of Models

Convolutional Neural Networks

- The classic model that started the phenomena of Neural Networks.

Auto Encoder

- How to generate novel features by funneling.

Boltzmann Machine

- Energy Based Models.

Some of the Models to be Reviewed of Models

Convolutional Neural Networks

- The classic model that started the phenomena of Neural Networks.

Auto Encoder

- How to generate novel features by funneling.

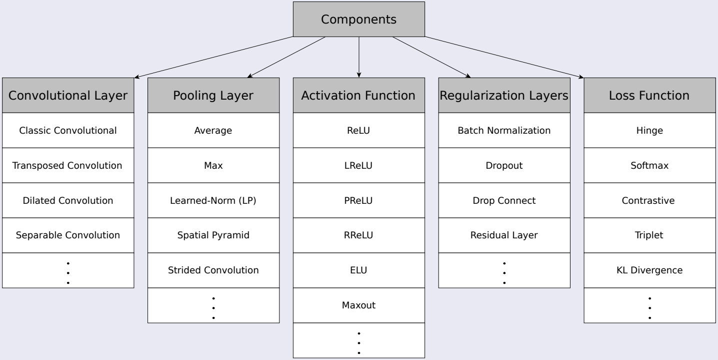
Boltzmann Machine

- Energy Based Models.

However

We will see that there are many possible architectures

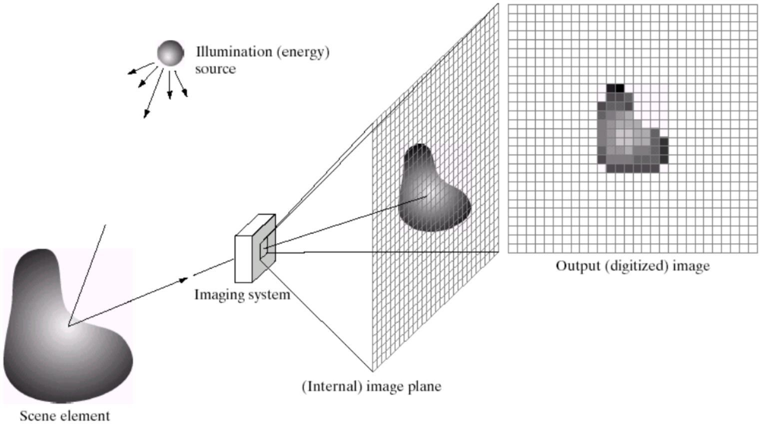
- And more with the different layers [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20] :



Outline

Outline

Digital Images as pixels in a digitized matrix



Further

Pixel values typically represent

- Gray levels, colours, heights, opacities etc

Something Notable

- Remember digitization implies that a digital image is an approximation of a real scene

Further

Pixel values typically represent

- Gray levels, colours, heights, opacities etc

Something Notable

- Remember digitization implies that a digital image is an approximation of a real scene

Therefore, we have the following process

Low Level Process

Input	Processes	Output
Image	Noise Removal	Improved Image
	Image Sharpening	

Example, Edge Detection

Therefore, we have the following process

Low Level Process

Input	Processes	Output
Image	Noise Removal	Improved Image
	Image Sharpening	

Example, Edge Detection

Then

Mid Level Process

Input	Processes	Output
Image	Object Recognition	Attributes
	Segmentation	

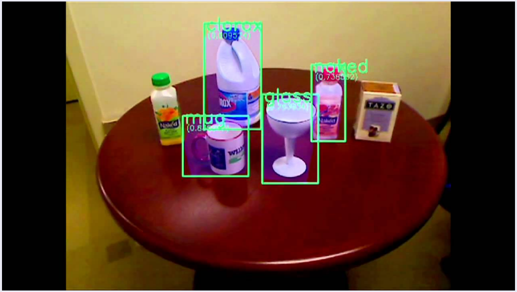
Object Recognition

Then

Mid Level Process

Input	Processes	Output
Image	Object Recognition	Attributes
	Segmentation	

Object Recognition



Therefore

It would be nice to automatize all these processes

- We would solve a lot of headaches when setting up such process

Why not to use the data sets

- By using a Neural Networks that replicates the process.

Therefore

It would be nice to automatize all these processes

- We would solve a lot of headaches when setting up such process

Why not to use the data sets

- By using a Neural Networks that replicates the process.

Convolutional Neural Networks History

Work by Hubel and Wiesel in the 1950s and 1960s

- They showed that cat and monkey visual cortexes contain neurons that individually respond to small regions of the visual field.

After all more studies about the visual cortex happened

- David H. Hubel and Torsten N. Wiesel (2005). Brain and visual perception: the story of a 25-year collaboration. Oxford University Press US. p. 106.

Convolutional Neural Networks History

Work by Hubel and Wiesel in the 1950s and 1960s

- They showed that cat and monkey visual cortexes contain neurons that individually respond to small regions of the visual field.

After all more studies about the visual cortex happened

- David H. Hubel and Torsten N. Wiesel (2005). Brain and visual perception: the story of a 25-year collaboration. Oxford University Press US. p. 106.

Neurocognitron (Circa 1980)

Kunihiko Fukushima [21]

- Proposed a Hierarchical Network for image recognition with a convolution!!!

But it used a function φ

$$\varphi\left(\frac{1+\sum_{k_{l-1}=1}^{K_{l-1}}\sum_{v\in S_l} a_l\left(k_{l-1},v,k_l\right)u_{cl-1}\left(k_{l=1},n+v\right)}{1+\frac{2r_l}{1+r_l}b_l\left(k_l\right)v_{Cl-1}\left(n\right)}-1\right)$$

With a Relu function

$$\varphi\left(x\right)=\begin{cases} x & x\geq 0\\ 0 & x<0\end{cases}$$

Neurocognitron (Circa 1980)

Kunihiko Fukushima [21]

- Proposed a Hierarchical Network for image recognition with a convolution!!!

But it used a function φ

$$\varphi\left(\frac{1+\sum_{k_{t-1}=1}^{K_{t-1}}\sum_{v\in S_l} a_l\left(k_{t-1},v,k_l\right)u_{cl-1}\left(k_{l=1},n+v\right)}{1+\frac{2r_l}{1+r_l}b_l\left(k_l\right)v_{Cl-1}\left(n\right)}-1\right)$$

With a Relu function

$$\varphi\left(x\right)=\begin{cases} x & x\geq 0\\ 0 & x<0\end{cases}$$

Neurocognitron (Circa 1980)

Kunihiko Fukushima [21]

- Proposed a Hierarchical Network for image recognition with a convolution!!!

But it used a function φ

$$\varphi\left(\frac{1+\sum_{k_{t-1}=1}^{K_{t-1}}\sum_{v\in S_l} a_l\left(k_{t-1},v,k_l\right)u_{cl-1}\left(k_{l=1},n+v\right)}{1+\frac{2r_l}{1+r_l}b_l\left(k_l\right)v_{Cl-1}\left(n\right)}-1\right)$$

With a Relu function

$$\varphi\left(x\right)=\begin{cases} x & x\geq 0\\ 0 & x<0\end{cases}$$

Furthermore (Circa 1993)

Weng et al. [22, 23]

- Proposed the use of Maxpooling to recognize 3D objects in 2D images

Yan LeCunn finally proposed the use of backpropagation [24]

- The Beginning of the Dream!!!

Furthermore (Circa 1993)

Weng et al. [22, 23]

- Proposed the use of Maxpooling to recognize 3D objects in 2D images

Yan LeCunn finally proposed the use of backpropagation [24]

- The Beginning of the Dream!!!

Convolutional Neural Networks

Basically they are deep learners based in convolutions or its variants

$$(f * g)(i,j) = \sum_{k=n}^{-n} \sum_{l=-n}^{n} f(k,l) \times g(i-k, j-l) \tag{1}$$

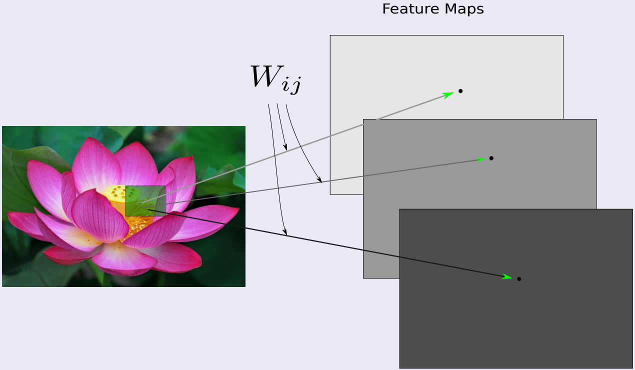
Basically Filters

Convolutional Neural Networks

Basically they are deep learners based in convolutions or its variants

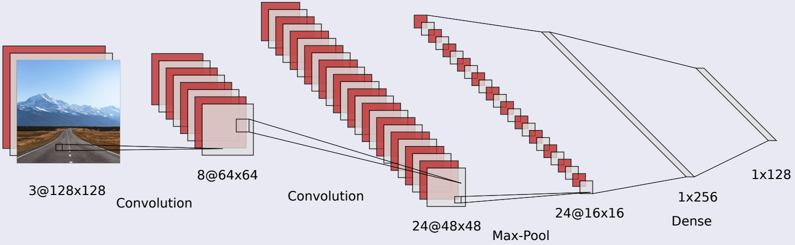
$$(f * g)(i,j) = \sum_{k=n}^{-n} \sum_{l=-n}^{n} f(k,l) \times g(i-k, j-l) \tag{1}$$

Basically Filters



Example of CNN

A Basic Convolutional Network



Outline

We know that

Many of the existing machine learning algorithms

- They depend on the quality of the input characteristics to generate a good model.

Not only that

- The amount of these variables is also important, given that performance tends to decline as the input dimensionality increases.

We know that

Many of the existing machine learning algorithms

- They depend on the quality of the input characteristics to generate a good model.

Not only that

- The amount of these variables is also important, given that performance tends to decline as the input dimensionality increases.

We have several techniques for that

Principal Component Analysis

$$L\left(\boldsymbol{u}_1\right) = \boldsymbol{u}_1^T S \boldsymbol{u}_1 + \lambda_1 \left(1 - \boldsymbol{u}_1^T \boldsymbol{u}_1\right)$$

Linear Locally Embeddings

$$\Phi\left(Y\right) = \sum_i \left| Y_i - \sum_j W_{ij} Y_j \right|^2$$

And recently

- Uniform Manifold Approximation and Projection for Dimension Reduction [25]

We have several techniques for that

Principal Component Analysis

$$L\left(\boldsymbol{u}_1\right) = \boldsymbol{u}_1^T S \boldsymbol{u}_1 + \lambda_1 \left(1 - \boldsymbol{u}_1^T \boldsymbol{u}_1\right)$$

Linear Locally Embeddings

$$\Phi\left(Y\right) = \sum_i \left| Y_i - \sum_j W_{ij} Y_j \right|^2$$

And recently

- Uniform Manifold Approximation and Projection for Dimension Reduction [25]

We have several techniques for that

Principal Component Analysis

$$L(\boldsymbol{u}_1) = \boldsymbol{u}_1^T S \boldsymbol{u}_1 + \lambda_1 \left(1 - \boldsymbol{u}_1^T \boldsymbol{u}_1\right)$$

Linear Locally Embeddings

$$\Phi(Y) = \sum_i \left| Y_i - \sum_j W_{ij} Y_j \right|^2$$

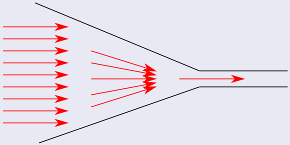
And recently

- Uniform Manifold Approximation and Projection for Dimension Reduction [25]

Therefore

We have the need to codify the original feature into better ones

- This can be done by a series of mappings that act as funnels, How?



Basically, we have a series of mappings

$$x \in \mathbb{R}^{n_1} \to f_1(x) \in \mathbb{R}^{n_2} \to f_2(x_1) \in \mathbb{R}^{n_3} \cdots \longrightarrow f_m(x_{m-1}) \in \mathbb{R}^{n_{m+1}}$$

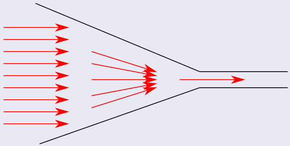
Where

$$n_1 < n_2 < \cdots < n_m < n_{m+1}$$

Therefore

We have the need to codify the original feature into better ones

- This can be done by a series of mappings that act as funnels, How?



Basically, we have a series of mappings

$$x \in \mathbb{R}^{n_1} \to f_1(x) \in \mathbb{R}^{n_2} \to f_2(x_1) \in \mathbb{R}^{n_3} \cdots \longrightarrow f_m(x_{m-1}) \in \mathbb{R}^{n_{m+1}}$$

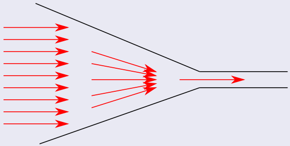
Where

$$n_1 < n_2 < \cdots < n_m < n_{m+1}$$

Therefore

We have the need to codify the original feature into better ones

- This can be done by a series of mappings that act as funnels, How?



Basically, we have a series of mappings

$$x \in \mathbb{R}^{n_1} \to f_1(x) \in \mathbb{R}^{n_2} \to f_2(x_1) \in \mathbb{R}^{n_3} \cdots \longrightarrow f_m(x_{m-1}) \in \mathbb{R}^{n_{m+1}}$$

Where

$$n_1 < n_2 < \cdots < n_m < n_{m+1}$$

Then, we can use linear mappings for this

With the following matrix functions

$$\sigma\left[f_{A_{i+1}}\left(x_i\right)\right] = \sigma\left(A_{i+1}x\right)$$

Therefore

- Therefore, we have the following architecture.

Then, we can use linear mappings for this

With the following matrix functions

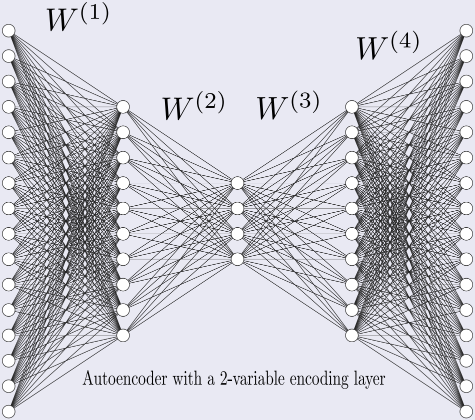
$$\sigma\left[f_{A_{i+1}}\left(x_i\right)\right] = \sigma\left(A_{i+1}x\right)$$

Therefore

- Therefore, we have the following architecture.

The Basic Auto Encoder Architecture

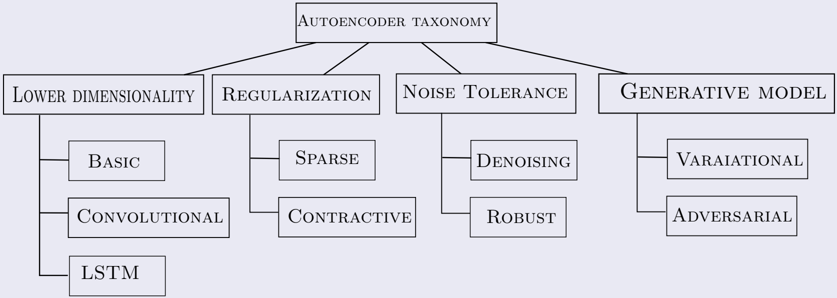
We have



Autoencoder with a 2-variable encoding layer

Taxonomy

Most popular Auto Encoders



Outline

The Basic Energy Models

We have that the Boltzmann Machines

- A Boltzmann machine is a network of units that are connected to each other

Here, we have N be the number of units

- Each unit takes a binary value in $\{0, 1\}$
  - Represented by a random variable X_i, $i = 1, ..., N$.

Additionally, it has parameters

- Bias b_i
- Weight w_{ij} between unit i and unit j, $(i, j) \in [1, N-1] \times [i+1, N]$

The Basic Energy Models

We have that the Boltzmann Machines

- A Boltzmann machine is a network of units that are connected to each other

Here, we have N be the number of units

- Each unit takes a binary value in $\{0, 1\}$
  - Represented by a random variable X_i, $i = 1, ..., N$.

Additionally, it has parameters

- Bias b_i
- Weight w_{ij} between unit i and unit j, $(i, j) \in [1, N-1] \times [i+1, N]$

The Basic Energy Models

We have that the Boltzmann Machines

- A Boltzmann machine is a network of units that are connected to each other

Here, we have N be the number of units

- Each unit takes a binary value in $\{0, 1\}$
  - Represented by a random variable X_i, $i = 1, ..., N$.

Additionally, it has parameters

- Bias b_i
- Weight w_{ij} between unit i and unit j, $(i, j) \in [1, N-1] \times [i+1, N]$

The Energy Based Structure

The energy of the Boltzmann machine is defined by

$$E_{W,\boldsymbol{b}}\left[\boldsymbol{x}\right] = -\sum_{i=1}^{N} b_i x_i - \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} w_{ij} x_i x_j = -\boldsymbol{b}^T\boldsymbol{x} - \boldsymbol{x}^T W \boldsymbol{x}$$

This allows to define a probability distribution

$$\mathbb{P}_{W,b}\left(\boldsymbol{x}\right) = \frac{\exp\left(-E_{W,b}\left[\boldsymbol{x}\right]\right)}{\sum_{\widetilde{\boldsymbol{x}}} \exp\left(-E_{W,b}\left[\widetilde{\boldsymbol{x}}\right]\right)}$$

The Energy Based Structure

The energy of the Boltzmann machine is defined by

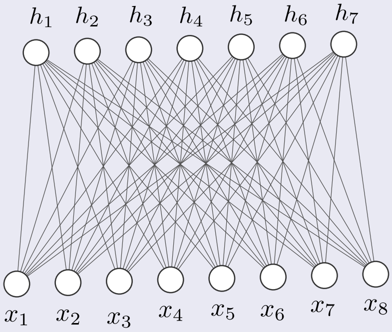
$$E_{W,\boldsymbol{b}}\left[\boldsymbol{x}\right] = -\sum_{i=1}^{N} b_i x_i - \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} w_{ij} x_i x_j = -\boldsymbol{b}^T \boldsymbol{x} - \boldsymbol{x}^T W \boldsymbol{x}$$

This allows to define a probability distribution

$$\mathbb{P}_{W,\boldsymbol{b}}\left(\boldsymbol{x}\right) = \frac{\exp\left(-E_{W,\boldsymbol{b}}\left[\boldsymbol{x}\right]\right)}{\sum_{\widetilde{\boldsymbol{x}}} \exp\left(-E_{W,\boldsymbol{b}}\left[\widetilde{\boldsymbol{x}}\right]\right)}$$

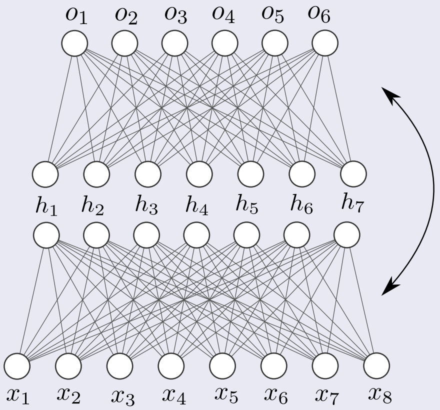
Example

Restricted Boltzmann Machines where the conectivity is layer by layer



Thus, using it as a basic model

We can stack them into a multiple layer model



Outline

Generative Adversarial Networks

They can be seen as an Accept-Reject MCMC Model

- However, they do not require Markov Chains with the classic problem:
  - The independence between the samples to generate ergodic probabilities (The real one)

As in the Accept-Reject

- The **generator network** tries to produce realistic-looking samples
- The **discriminator network** tries to figure out whether an image came from the training set or the generator network

Generative Adversarial Networks

They can be seen as an Accept-Reject MCMC Model

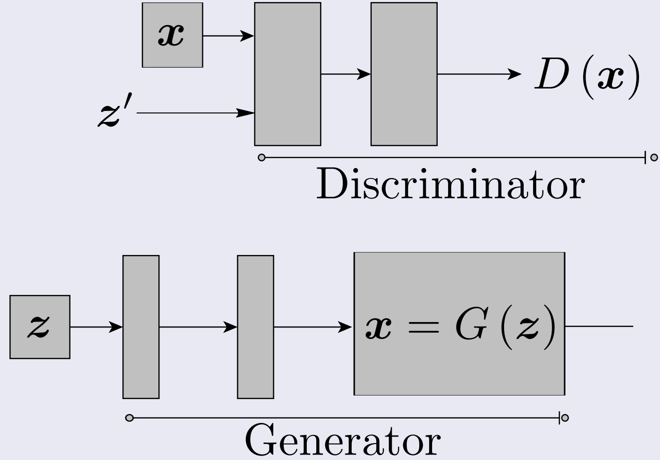
- However, they do not require Markov Chains with the classic problem:
    - The independence between the samples to generate ergodic probabilities (The real one)

As in the Accept-Reject

- The **generator network** tries to produce realistic-looking samples
- The **discriminator network** tries to figure out whether an image came from the training set or the generator network

Graphically

We have the following Basic Model



Here

There is a need to join both functions

- So, we can use the idea of Backpropagation to obtain the desired minimization.

How can we do this?

- We can define a sensible learning criterion when the dataset is not linearly separable

For this, we can use the **logistic cross-entropy loss** (We will explain more about this later)

$$\mathcal{L}_{LCE}(z,t) = L_{CE}(\sigma(z), t) = t\log\left(1+e^{-z}\right) + (1-t)\log\left(1+e^{z}\right)$$

Here

There is a need to join both functions

- So, we can use the idea of Backpropagation to obtain the desired minimization.

How can we do this?

- We can define a sensible learning criterion when the dataset is not linearly separable

For this, we can use the **logistic cross-entropy loss** (We will explain more about this later)

$$\mathcal{L}_{LCE}\left(z,t\right) = L_{CE}\left(\sigma\left(z\right),t\right) = t\log\left(1+e^{-z}\right) + \left(1-t\right)\log\left(1+e^{z}\right)$$

Here

There is a need to join both functions

- So, we can use the idea of Backpropagation to obtain the desired minimization.

How can we do this?

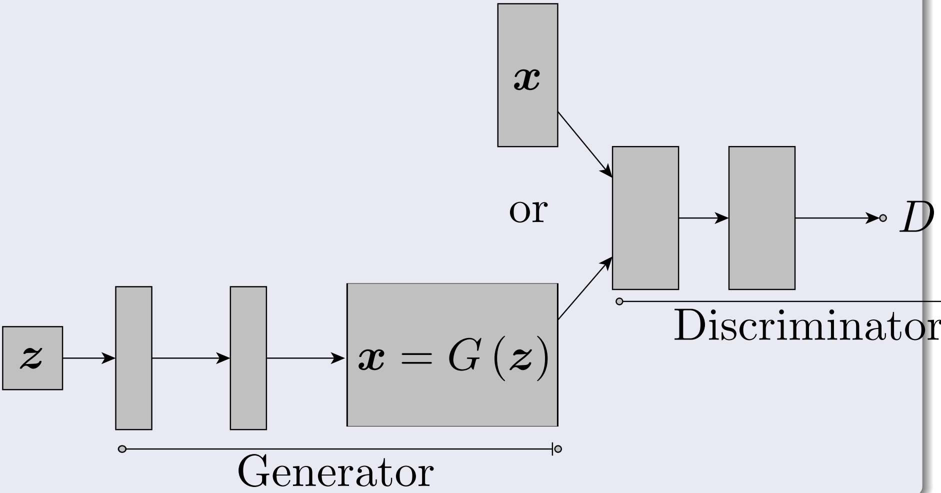
- We can define a sensible learning criterion when the dataset is not linearly separable

For this, we can use the **logistic cross-entropy loss** (We will explain more about this later)

$$\mathcal{L}_{LCE}\left(z,t\right) = L_{CE}\left(\sigma\left(z\right),t\right) = t\log\left(1+e^{-z}\right) + \left(1-t\right)\log\left(1+e^{z}\right)$$

Therefore, we have

The following architecture use this idea



In this basic Generator

D denote the discriminator's predicted probability of being data

$$\mathcal{J}_D = E_{\boldsymbol{x}\sim\mathcal{D}}\left[-\log D\left(\boldsymbol{x}\right)\right] + E_{\boldsymbol{z}}\left[-\log\left(1 - D\left(G\left(\boldsymbol{z}\right)\right)\right)\right]$$

One possible cost function for the generator

$$\mathcal{J}_G = -\mathcal{J}_D = const + E_{\boldsymbol{z}}\left[\log\left(1 - D\left(G\left(\boldsymbol{z}\right)\right)\right)\right]$$

In this basic Generator

D denote the discriminator's predicted probability of being data

$$\mathcal{J}_D = E_{\boldsymbol{x} \sim \mathcal{D}} \left[-\log D\left(\boldsymbol{x}\right) \right] + E_{\boldsymbol{z}} \left[-\log \left(1 - D\left(G\left(\boldsymbol{z}\right)\right)\right) \right]$$

One possible cost function for the generator

$$\mathcal{J}_G = -\mathcal{J}_D = const + E_{\boldsymbol{z}} \left[\log \left(1 - D\left(G\left(\boldsymbol{z}\right)\right)\right) \right]$$

Then using both functions

The minimax formulation

- Since the generator and discriminator are playing a zero-sum game against each other.

Basically

$$\max_{G} \min_{D} \mathcal{J}_D$$

There are other examples using the LSE [26]

$$\mathcal{J}_G = \frac{1}{N} \sum_{i=1}^{N} \left[G\left(\boldsymbol{z} \right) - \boldsymbol{x} \right]^2$$

Then using both functions

The minimax formulation

- Since the generator and discriminator are playing a zero-sum game against each other.

Basically

$$\max_{G} \min_{D} \mathcal{J}_D$$

There are other examples using the LSE [26]

$$\mathcal{J}_G = \frac{1}{N}\sum_{i=1}^{N} \left[G(z) - x\right]^2$$

Then using both functions

The minimax formulation

- Since the generator and discriminator are playing a zero-sum game against each other.

Basically

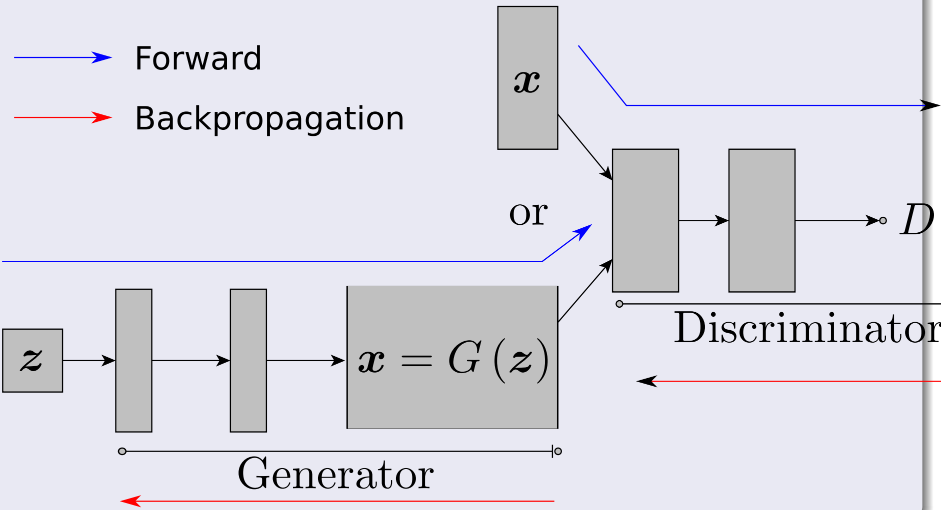
$$\max_{G} \min_{D} \mathcal{J}_D$$

There are other examples using the LSE [26]

$$\mathcal{J}_G = \frac{1}{N} \sum_{i=1}^{N} \left[G\left(\boldsymbol{z}\right) - \boldsymbol{x} \right]^2$$

Therefore, we have two updates

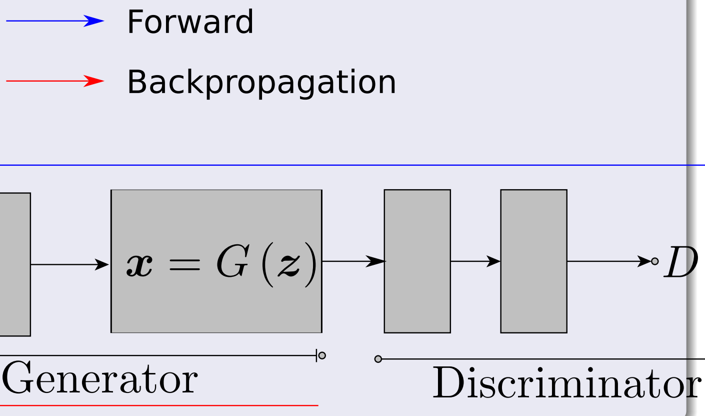
First update the Discriminator



Now

Update the Generator

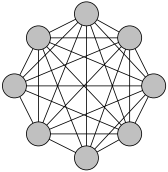
Backprop Derivatives Through the Discriminator, but do not change variables on it... only in the generator



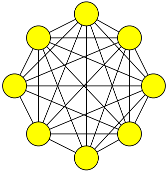
Outline

There Are Many More!!! Here a few more...

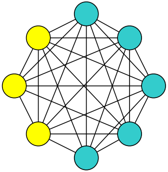
Markov Chain



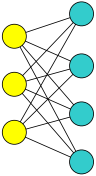
Hopfield Network



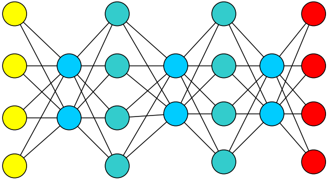
Boltzmann Machine



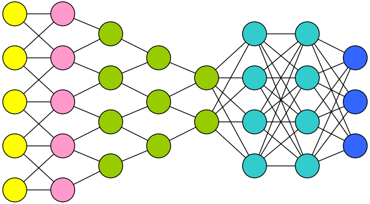
Restricted BM



Deep Belief Network

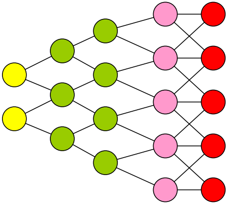


Convolutional Network

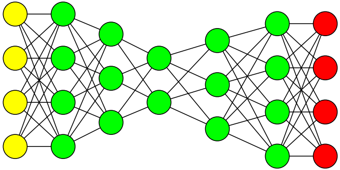


Furthermore

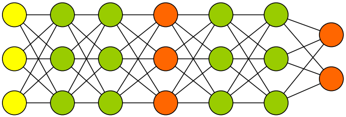
Deconvolutional Network



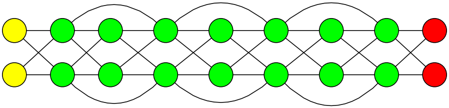
Autoencoder



Generative Adversarial Network



Deep Residual Network



Outline

As We know

In Recurrent Neural Networks, we have the problem

- Vanishing and Exploding Gradients

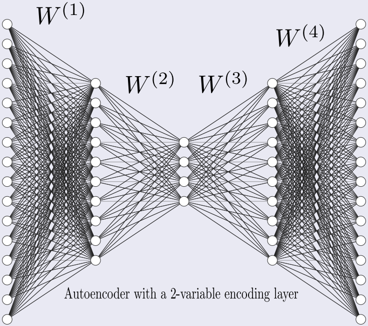
In the Deeper Architectures as encoder-decoder we have such phenomena

As We know

In Recurrent Neural Networks, we have the problem

- Vanishing and Exploding Gradients

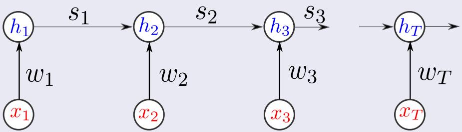
In the Deeper Architectures as encoder-decoder we have such phenomena



Autoencoder with a 2-variable encoding layer

Consider a simple encoder encoder network

We have this simplified version



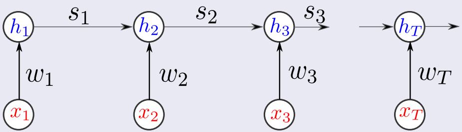
We have the following structure

$$h_t = w_t x_t + z_{t-1}$$

$$z_t = s_t h_t$$

Consider a simple encoder encoder network

We have this simplified version



We have the following structure

$$h_t = w_t x_t + z_{t-1}$$
$$z_t = s_t h_t$$

Backpropagation Rules

Then, we get the following backpropagation rules

$$\frac{\partial h_t}{\partial w_i} = \frac{\partial h_t}{\partial h_{t-1}} \times \frac{\partial h_{t-1}}{\partial h_{t-2}} \times ... \times \frac{\partial h_i}{\partial w_i}$$
$$\frac{\partial h_t}{\partial s_i} = \frac{\partial h_t}{\partial h_{t-1}} \times \frac{\partial h_{t-1}}{\partial h_{t-2}} \times ... \times \frac{\partial h_{i+1}}{\partial s_i}$$

Then, we have

By Using Our simplifying assumption that

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial \left(w_t x_t + s_{t-1} h_{t-1}\right)}{\partial h_{t-1}} = s_{t-1}$$

And for $\frac{\partial h_i}{\partial w_i}$

$$\frac{\partial h_i}{\partial w_i} = x_t$$

Finally, we have that

$$\frac{\partial h_t}{\partial w_i} = x_t \left[\prod_{k=t-1}^{i-1} s_k\right]$$

Then, we have

By Using Our simplifying assumption that

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial \left(w_t x_t + s_{t-1} h_{t-1}\right)}{\partial h_{t-1}} = s_{t-1}$$

And for $\frac{\partial h_i}{\partial w_i}$

$$\frac{\partial h_i}{\partial w_i} = x_t$$

Finally, we have that

$$\frac{\partial h_t}{\partial w_i} = x_t \left[\prod_{k=t-1}^{i-1} s_k\right]$$

Then, we have

By Using Our simplifying assumption that

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial \left(w_t x_t + s_{t-1} h_{t-1}\right)}{\partial h_{t-1}} = s_{t-1}$$

And for $\frac{\partial h_i}{\partial w_i}$

$$\frac{\partial h_i}{\partial w_i} = x_t$$

Finally, we have that

$$\frac{\partial h_t}{\partial w_i} = x_t \left[\prod_{k=t-1}^{i-1} s_k\right]$$

It is clear that

Unless the s_k's are near to 1

- You have the vanishing gradient if $s_k \in [0, 1)$ for all k.
- You have the exploding gradient if $s_k \in (1, +\infty]$ for all k.

Even with activation functions

- These terms tend to appear in the Deep Learners when Backpropagation is done

In the case of Forward

- We have many activation function that squash the signal...

It is clear that

Unless the s_k's are near to 1

- You have the vanishing gradient if $s_k \in [0, 1)$ for all k.
- You have the exploding gradient if $s_k \in (1, +\infty]$ for all k.

Even with activation functions

- These terms tend to appear in the Deep Learners when Backpropagation is done

In the case of Forward

- We have many activation function that squash the signal...

It is clear that

Unless the s_k's are near to 1

- You have the vanishing gradient if $s_k \in [0,1)$ for all k.
- You have the exploding gradient if $s_k \in (1,+\infty]$ for all k.

Even with activation functions

- These terms tend to appear in the Deep Learners when Backpropagation is done

In the case of Forward

- We have many activation function that squash the signal...

Outline

Instead of doing this

Let us to do the following

$$f\left(x\right) = 3.5x\left(1 - x\right)$$

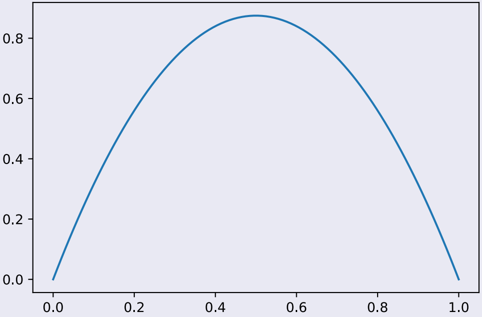
In the first composition, we get

Instead of doing this

Let us to do the following

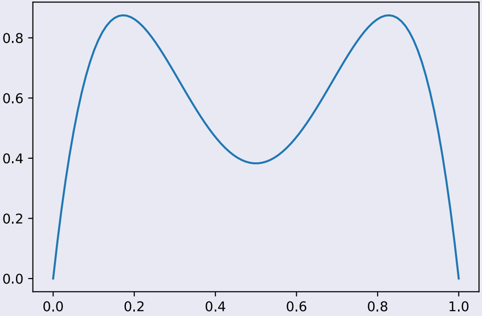
$$f\left(x\right) = 3.5x\left(1 - x\right)$$

In the first composition, we get



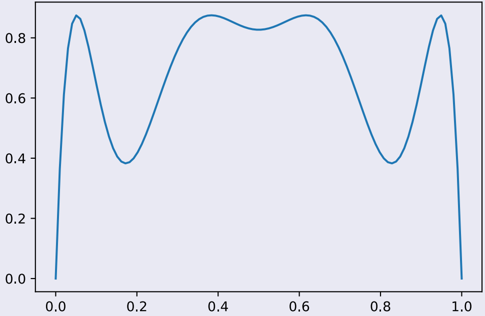
Now, as we compound the function

Second one, $y = f \circ f(x)$



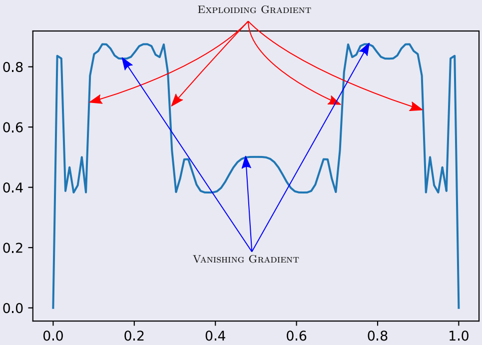
Now, as we increment iterations

Third one, $y = f \circ f \circ f\,(x)$



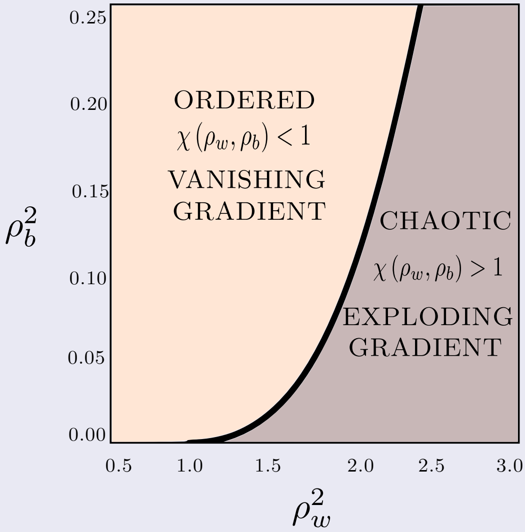
Finally

We see the increment in the gradient part negative or positive



Actually, we have

A Frontier defining the Vanishing and Exploding Gradient [27]



Outline

Actually

Eventually, the iterates go to infinity or zero OR

- They wind up at a fixed point...

A Fixed Point?

$$x = f(x)$$

Actually

Eventually, the iterates go to infinity or zero OR

- They wind up at a fixed point...

A Fixed Point?

$$x = f(x)$$

Basically

The fixed points can be thought

- Some fixed points repel the iterates; **these are called sources.**
- Other fixed points attract the iterates; **these are called sinks.**

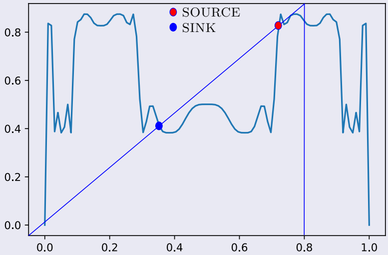
Basically $f'(x) < 1$ are sinks and $f'(x) > 1$ are sources

Basically

The fixed points can be thought

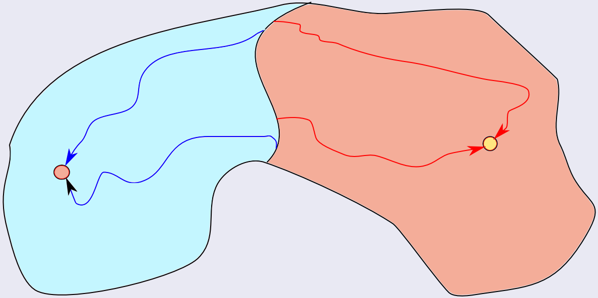
- Some fixed points repel the iterates; **these are called sources.**
- Other fixed points attract the iterates; **these are called sinks.**

Basically $f'(x) < 1$ are sinks and $f'(x) > 1$ are sources



Areas of attraction

Basically, we have that there are areas the pull in the iterations of the function



These fixed points

In Deep Structures as RNN without sigmoid functions

$$
\begin{aligned}
\boldsymbol{h}_t &= W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1} \\
\boldsymbol{y}_t &= V_{os}\boldsymbol{h}_t
\end{aligned}
$$

We have

$$
\boldsymbol{x}_t = V_{os}\left[W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1}\right]
$$

Therefore if $\boldsymbol{b} = V_{os}U_{ss}\boldsymbol{h}_{t-1}$

- Then, we have that

$$
\boldsymbol{x}_t = V_{os}W_{sd}\boldsymbol{x}_t + V_{os}U_{ss}\boldsymbol{h}_{t-1} = I\boldsymbol{x}_t + 0
$$

These fixed points

In Deep Structures as RNN without sigmoid functions

$$\boldsymbol{h}_t = W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1}$$
$$\boldsymbol{y}_t = V_{os}\boldsymbol{h}_t$$

We have

$$\boldsymbol{x}_t = V_{os}\left[W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1}\right]$$

Therefore if $\boldsymbol{b} = V_{os}U_{ss}\boldsymbol{h}_{t-1}$

- Then, we have that

$$\boldsymbol{x}_t = V_{os}W_{sd}\boldsymbol{x}_t + V_{os}U_{ss}\boldsymbol{h}_{t-1} = I\boldsymbol{x}_t + 0$$

These fixed points

In Deep Structures as RNN without sigmoid functions

$$
\begin{aligned}
\boldsymbol{h}_t &= W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1} \\
\boldsymbol{y}_t &= V_{os}\boldsymbol{h}_t
\end{aligned}
$$

We have

$$
\boldsymbol{x}_t = V_{os}\left[W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1}\right]
$$

Therefore if $\boldsymbol{b} = V_{os}U_{ss}\boldsymbol{h}_{t-1}$

- Then, we have that

$$
\boldsymbol{x}_t = V_{os}W_{sd}\boldsymbol{x}_t + V_{os}U_{ss}\boldsymbol{h}_{t-1} = I\boldsymbol{x}_t + 0
$$

Therefore

We have that

$$V_{os}W_{sd} \approx I \text{ and } \boldsymbol{h}_{t-1} \approx 0$$

They define an area

Where V_{os} and W_{sd}

- They are the inverse of each other

And the hidden state is almost zero

- Basically they fixed point converts a RNN without activation functions in a linear model

They define an area

Where V_{os} and W_{sd}

- They are the inverse of each other

And the hidden state is almost zero

- Basically they fixed point converts a RNN without activation functions in a linear model

Outline

Outline

Gradient Clipping

We prevent gradient from blowing up by rescaling to a certain value

$$\|\nabla_\theta L\| > \eta \implies \nabla_\theta L = \frac{\eta \nabla_\theta L}{\|\nabla_\theta L\|}$$

We have a series of nice analysis [28]

$$\min_{x \in \mathbb{R}^d} f(x)$$

Furthermore, we define a space

$$S = \{x | \exists y \text{ such that } f(y) \leq f(x_o), \text{ and } \|x - y\| \leq 1\}$$

Gradient Clipping

We prevent gradient from blowing up by rescaling to a certain value

$$\|\nabla_\theta L\| > \eta \Longrightarrow \nabla_\theta L = \frac{\eta \nabla_\theta L}{\|\nabla_\theta L\|}$$

We have a series of nice analysis [28]

$$\min_{x \in \mathbb{R}^d} f(x)$$

Furthermore, we define a space

$$S = \{x | \exists y \text{ such that } f(y) \leq f(x_o), \text{ and } \|x - y\| \leq 1\}$$

Gradient Clipping

We prevent gradient from blowing up by rescaling to a certain value

$$\|\nabla_\theta L\| > \eta \implies \nabla_\theta L = \frac{\eta \nabla_\theta L}{\|\nabla_\theta L\|}$$

We have a series of nice analysis [28]

$$\min_{x \in \mathbb{R}^d} f(x)$$

Furthermore, we define a space

$$S = \{x | \exists y \text{ such that } f(y) \leq f(x_o), \text{ and } \|x - y\| \leq 1\}$$

We have then for S

In $\mathbb{R}^2$ the following example

Assumptions

Assumption 1

- Function f is lower bounded by f^*

Assumption 2

- Function f is twice differentiable

Assumptions

Assumption 1

- Function f is lower bounded by f^*

Assumption 2

- Function f is twice differentiable

Then, there are the following proposals

The ordinary gradient descent

$$x_{k+1} = x_k - \eta \nabla f(x_k)$$

The Clipped Gradient Descent (CGD)

$$x_{k+1} = x_k - h_c \nabla f(x_k), \text{ where } h_c = \min\left\{\eta_c, \frac{\gamma\eta_c}{\|\nabla f(x)\|}\right\}$$

Normalized Gradient Descent (NGD)

$$x_{k+1} = x_k - h_n \nabla f(x_k), \text{ where } h_n = \frac{\eta_c}{\|\nabla f(x)\| + \beta}$$

Then, there are the following proposals

The ordinary gradient descent

$$x_{k+1} = x_k - \eta \nabla f(x_k)$$

The Clipped Gradient Descent (CGD)

$$x_{k+1} = x_k - h_c \nabla f(x_k), \text{ where } h_c = \min\left\{\eta_c, \frac{\gamma \eta_c}{\|\nabla f(x)\|}\right\}$$

Normalized Gradient Descent (NGD)

$$x_{k+1} = x_k - h_n \nabla f(x_k), \text{ where } h_n = \frac{\eta_c}{\|\nabla f(x)\| + \beta}$$

Then, there are the following proposals

The ordinary gradient descent

$$x_{k+1} = x_k - \eta \nabla f(x_k)$$

The Clipped Gradient Descent (CGD)

$$x_{k+1} = x_k - h_c \nabla f(x_k), \text{ where } h_c = \min\left\{\eta_c, \frac{\gamma \eta_c}{\|\nabla f(x)\|}\right\}$$

Normalized Gradient Descent (NGD)

$$x_{k+1} = x_k - h_n \nabla f(x_k), \text{ where } h_n = \frac{\eta_c}{\|\nabla f(x)\| + \beta}$$

Remark

Clipped GD and NGD are almost equivalent

- If we set $\gamma\eta_c = \eta_n$ and $\eta_c = \frac{\eta_n}{\beta}$ then

$$\frac{1}{2}h_c \leq h_n \leq 2h_c$$

A Natural Question

Definition

- The objective f is called L-smooth if $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$ for all $x, y \in \mathbb{R}^d$

This is equivalent under a twice differentiable f

$$\left\|\nabla^2 f(x)\right\| \leq L$$

Then, you get the following upper-bound

$$f(y) \approx f(x) + \nabla^T f(x)(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(x)(y - x)$$

A Natural Question

Definition

- The objective f is called L-smooth if $\|\nabla f(x) - \nabla f(y)\| \leq L \|x - y\|$ for all $x, y \in \mathbb{R}^d$

This is equivalent under a twice differentiable f

$$\left\|\nabla^2 f(x)\right\| \leq L$$

Then, you get the following upper-bound

$$f(y) \approx f(x) + \nabla^T f(x)(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(x)(y - x)$$

A Natural Question

Definition

- The objective f is called L-smooth if $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$ for all $x, y \in \mathbb{R}^d$

This is equivalent under a twice differentiable f

$$\left\|\nabla^2 f(x)\right\| \leq L$$

Then, you get the following upper-bound

$$f(y) \approx f(x) + \nabla^T f(x)(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(x)(y - x)$$

Then, it is possible to use the 3 Assumption

We have that

$$f(y) \leq f(x) + \nabla^T f(x)(y - x) + \frac{1}{2} L \|y - x\|^2$$

Then fixing all the other variables and assuming $y = x - h\nabla f(x)$

$$h^* = \arg\min_h \left[f(x) - h \|\nabla f(x)\|^2 + \frac{1}{2} L h^2 \|\nabla f(x)\|^2 \right] = \frac{1}{L}$$

Basically

- This choice of h leads to GD with a fixed step,

Then, it is possible to use the 3 Assumption

We have that

$$f(y) \leq f(x) + \nabla^T f(x)(y - x) + \frac{1}{2}L\|y - x\|^2$$

Then fixing all the other variables and assuming $y = x - h\nabla f(x)$

$$h^* = \arg\min_h \left[f(x) - h\|\nabla f(x)\|^2 + \frac{1}{2}Lh^2\|\nabla f(x)\|^2 \right] = \frac{1}{L}$$

Basically

- This choice of h leads to GD with a fixed step.

Then, it is possible to use the 3 Assumption

We have that

$$f(y) \leq f(x) + \nabla^T f(x)(y-x) + \frac{1}{2} L \|y-x\|^2$$

Then fixing all the other variables and assuming $y = x - h\nabla f(x)$

$$h^* = \arg\min_h \left[f(x) - h \|\nabla f(x)\|^2 + \frac{1}{2} L h^2 \|\nabla f(x)\|^2 \right] = \frac{1}{L}$$

Basically

- This choice of h leads to GD with a fixed step,

Now

Question

- "Is clipped gradient descent optimized for a different smoothness condition?"

Inspired in the equation

$$f(y) \leq f(x) + \nabla^T f(x)(y - x) + \frac{1}{2} L \|y - x\|^2$$

Assume

$$h^* = \frac{\eta}{\|\nabla f(x)\| + \beta}$$

Now

Question

- "Is clipped gradient descent optimized for a different smoothness condition?"

Inspired in the equation

$$f(y) \leq f(x) + \nabla^T f(x)(y - x) + \frac{1}{2}L\|y - x\|^2$$

Assume

$$h^* = \frac{\eta}{\|\nabla f(x)\| + \beta}$$

Now

Question

- “Is clipped gradient descent optimized for a different smoothness condition?”

Inspired in the equation

$$f(y) \leq f(x) + \nabla^T f(x)(y-x) + \frac{1}{2}L\|y-x\|^2$$

Assume

$$h^* = \frac{\eta}{\|\nabla f(x)\| + \beta}$$

Then, we have

Assume that such value optimize the equation

$$f(x) - h\left\|\nabla f(x)\right\|^2 + \frac{1}{2} L h^2 \left\|\nabla f(x)\right\|^2$$

Then, we have

$$L(x) = \frac{\left\|\nabla f(x)\right\| + \beta}{\eta}$$

Assumption 3 by using $\left\|\nabla^2 f(x)\right\| \leq L$

- (L_0, L_1)-smoothness. f is (L_0, L_1)-smooth, if there exist positive L_0 and L_1 such that $\left\|\nabla^2 f(x)\right\| \leq L_0 + L_1 \left\|\nabla f(x)\right\|$
  - $\nabla^2 f(x)$ is the Hessian

Then, we have

Assume that such value optimize the equation

$$f(x) - h\|\nabla f(x)\|^2 + \frac{1}{2}Lh^2\|\nabla f(x)\|^2$$

Then, we have

$$L(x) = \frac{\|\nabla f(x)\| + \beta}{\eta}$$

Assumption 3 by using $\|\nabla^2 f(x)\| \leq L$

- (L_0, L_1)-smoothness. f is (L_0, L_1)-smooth, if there exist positive L_0 and L_1 such that $\|\nabla^2 f(x)\| \leq L_0 + L_1\|\nabla f(x)\|$
  - $\nabla^2 f(x)$ is the Hessian

Then, we have

Assume that such value optimize the equation

$$f(x) - h\left\|\nabla f(x)\right\|^2 + \frac{1}{2}Lh^2\left\|\nabla f(x)\right\|^2$$

Then, we have

$$L(x) = \frac{\left\|\nabla f(x)\right\| + \beta}{\eta}$$

Assumption 3 by using $\left\|\nabla^2 f(x)\right\| \leq L$

- (L_0, L_1)-smoothness. $f$is (L_0, L_1)-smooth, if there exist positive L_0 and L_1 such that $\left\|\nabla^2 f(x)\right\| \leq L_0 + L_1\left\|\nabla f(x)\right\|$
  - $\nabla^2 f(x)$ is the Hessian

The final Theorem

Theorem (CGD) [28]

- Assume that Assumptions 1, 2, and 3 hold in set S. With parameters

$$\eta_c = \frac{1}{10L_o} \text{ and } \gamma = \min\left\{\frac{1}{\eta_c}, \frac{1}{10L_o\eta_c}\right\},$$

  - Then Clipped GD terminates in

$$\frac{20L_0\left(f\left(x_0\right)-f^*\right)}{\epsilon^2}+\frac{20\max\left\{1,L_1^2\right\}\left(f\left(x_0\right)-f^*\right)}{L_0} \text{ iterations}$$

Remarks

The paper

- It points out to a high correlation between the Jacobian and the Hessian

There are more work to be done

- Please read the paper...

Remarks

The paper

- It points out to a high correlation between the Jacobian and the Hessian

There are more work to be done

- Please read the paper...

Outline

Another way to stabilize the network

Data Normalization

- Standardization is the most popular form of preprocessing
  - Normally mean subtraction and subsequent scaling by the standard deviation.

Mean subtraction

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \text{ then } x_i^c = x_i - \mu$$

Finally

- Standardization refers to altering the data dimensions such that they are of approximately the same scale.

Another way to stabilize the network

Data Normalization

- Standardization is the most popular form of preprocessing
  - Normally mean subtraction and subsequent scaling by the standard deviation.

Mean subtraction

$$\mu = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i \text{ then } x_i^c = \boldsymbol{x}_i - \mu$$

Finally

- Standardization refers to altering the data dimensions such that they are of approximately the same scale.

Another way to stabilize the network

Data Normalization

- Standardization is the most popular form of preprocessing
  - Normally mean subtraction and subsequent scaling by the standard deviation.

Mean subtraction

$$\mu = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i \text{ then } x_i^c = \boldsymbol{x}_i - \mu$$

Finally

- Standardization refers to altering the data dimensions such that they are of approximately the same scale.

Therefore, we have that

Standardization

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \mu)^2$$

$$x_i^s = \frac{x_i - \mu}{\sigma}$$

However, there other tricks, Bengio et al [29]

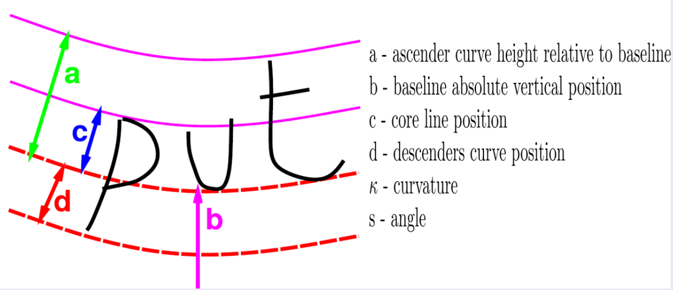
Therefore, we have that

Standardization

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \mu)^2$$

$$x_i^s = \frac{x_i - \mu}{\sigma}$$

However, there other tricks, Bengio et al [29]



Softmax Scaling

Thus

- All new features have zero mean and unit variance.

Further

- Other linear techniques limit the feature values in the range of $[0, 1]$ or $[-1, 1]$ by proper scaling.

However

- We can non-linear mapping. For example the softmax scaling.

Softmax Scaling

Thus

- All new features have zero mean and unit variance.

Further

- Other linear techniques limit the feature values in the range of $[0, 1]$ or $[-1, 1]$ by proper scaling.

However

- We can non-linear mapping. For example the softmax scaling.

Softmax Scaling

Thus

- All new features have zero mean and unit variance.

Further

- Other linear techniques limit the feature values in the range of $[0, 1]$ or $[-1, 1]$ by proper scaling.

However

- We can non-linear mapping. For example the softmax scaling.

Steps of Softmax Scaling

Softmax Scaling

- It consists of two steps

First one

$$y_{ik} = \frac{x_{ik} - \overline{x}_k}{\sigma} \quad (2)$$

Second one

$$\hat{x}_{ik} = \frac{1}{1 + \exp\{-y_{ik}\}} \quad (3)$$

Steps of Softmax Scaling

Softmax Scaling

- It consists of two steps

First one

$$y_{ik} = \frac{x_{ik} - \overline{x}_k}{\sigma} \quad (2)$$

Second one

$$\hat{x}_{ik} = \frac{1}{1 + \exp\{-y_{ik}\}} \quad (3)$$

Steps of Softmax Scaling

Softmax Scaling

- It consists of two steps

First one

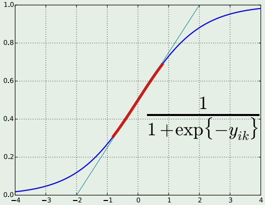
$$y_{ik} = \frac{x_{ik} - \overline{x}_k}{\sigma} \quad (2)$$

Second one

$$\hat{x}_{ik} = \frac{1}{1 + \exp\left\{-y_{ik}\right\}} \quad (3)$$

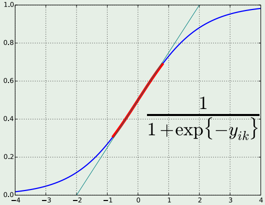
Explanation

Notice the red area is almost flat!!!



Explanation

Notice the red area is almost flat!!!

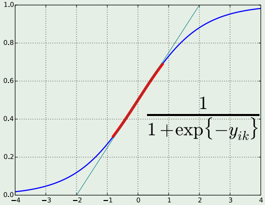


Thus, we have that

- The red region represents values of y inside of the region defined by the mean and variance (small values of y).
- Then, if we have those values x behaves as a linear function.

Explanation

Notice the red area is almost flat!!!



Thus, we have that

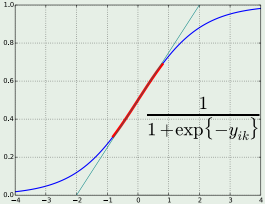
- The red region represents values of y inside of the region defined by the mean and variance (small values of y).
- Then, if we have those values x behaves as a linear function.

And values too away from the mean

- They are squashed by the exponential part of the function.

Explanation

Notice the red area is almost flat!!!



Thus, we have that

- The red region represents values of y inside of the region defined by the mean and variance (small values of y).
- Then, if we have those values x behaves as a linear function.

And values too away from the mean

- They are squashed by the exponential part of the function.

Outline

Here, the people at Google [17] around 2015

They commented in the "Internal Covariate Shift Phenomena"

- Due to the change in the distribution of each layer's input

They claim

- The min-batch forces to have those changes which impact on the learning capabilities of the network.

In Neural Networks, they define this

- Internal Covariate Shift as the change in the distribution of network activations due to the change in network parameters during training.

Here, the people at Google [17] around 2015

They commented in the "Internal Covariate Shift Phenomena"

- Due to the change in the distribution of each layer's input

They claim

- The min-batch forces to have those changes which impact on the learning capabilities of the network.

In Neural Networks, they define this

- Internal Covariate Shift as the change in the distribution of network activations due to the change in network parameters during training.

Here, the people at Google [17] around 2015

They commented in the "Internal Covariate Shift Phenomena"

- Due to the change in the distribution of each layer's input

They claim

- The min-batch forces to have those changes which impact on the learning capabilities of the network.

In Neural Networks, they define this

- Internal Covariate Shift as the change in the distribution of network activations due to the change in network parameters during training.

They gave the following reasons

Consider a layer with the input u that adds the learned bias b

- Then, it normalizes the result by subtracting the mean of the activation over the training data:

$$\widehat{\boldsymbol{x}} = \boldsymbol{x} - E\left[\boldsymbol{x}\right]$$

  - $\mathcal{X} = \{\boldsymbol{x}, ..., \boldsymbol{x}_N\}$ the data samples and $E\left[\boldsymbol{x}\right] = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i$

Now, if the gradient ignores the dependence of $E\left[x\right]$ on b

- Then $b = b + \Delta b$ where $\Delta b \propto -\frac{\partial l}{\partial \widehat{x}}$

Finally

$$u + (b + \Delta b) - E[u + (b + \Delta b)] = u + b - E[u + b]$$

They gave the following reasons

Consider a layer with the input u that adds the learned bias b

- Then, it normalizes the result by subtracting the mean of the activation over the training data:

$$\widehat{\boldsymbol{x}} = \boldsymbol{x} - E\left[\boldsymbol{x}\right]$$

  - $\mathcal{X} = \{\boldsymbol{x}, ..., \boldsymbol{x}_N\}$ the data samples and $E\left[\boldsymbol{x}\right] = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i$

Now, if the gradient ignores the dependence of $E\left[x\right]$ on b

- Then $b = b + \Delta b$ where $\Delta b \propto -\frac{\partial l}{\partial \widehat{x}}$

Finally

$$u + (b + \Delta b) - E[u + (b + \Delta b)] = u + b - E[u + b]$$

They gave the following reasons

Consider a layer with the input u that adds the learned bias b

- Then, it normalizes the result by subtracting the mean of the activation over the training data:

$$\widehat{\boldsymbol{x}} = \boldsymbol{x} - E\left[\boldsymbol{x}\right]$$

  - $\mathcal{X} = \{\boldsymbol{x}, ..., \boldsymbol{x}_N\}$ the data samples and $E\left[\boldsymbol{x}\right] = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i$

Now, if the gradient ignores the dependence of $E\left[\boldsymbol{x}\right]$ on b

- Then $b = b + \Delta b$ where $\Delta b \propto -\frac{\partial l}{\partial \widehat{x}}$

Finally

$$u + (b + \Delta b) - E[u + (b + \Delta b)] = u + b - E[u + b]$$

Then

The following will happen

- The update to b leads to **no change** in the output of the layer.

Therefore

- We need to integrate the normalization into the process of training.

Then

The following will happen

- The update to b leads to **no change** in the output of the layer.

Therefore

- We need to integrate the normalization into the process of training.

Normalization via Mini-Batch Statistic

It is possible to describe the normalization as a transformation layer

$$\widehat{\boldsymbol{x}} = Norm\left(\boldsymbol{x}, \mathcal{X}\right)$$

- Which depends on all the training samples $\mathcal{X}$ which also depends on the layer parameters

For back-propagation, we will need to generate the following terms

$$\frac{\partial Norm\left(\boldsymbol{x}, \mathcal{X}\right)}{\partial \boldsymbol{x}} \text{ and } \frac{\partial Norm\left(\boldsymbol{x}, \mathcal{X}\right)}{\partial \mathcal{X}}$$

Normalization via Mini-Batch Statistic

It is possible to describe the normalization as a transformation layer

$$\widehat{\boldsymbol{x}} = Norm\left(\boldsymbol{x}, \mathcal{X}\right)$$

- Which depends on all the training samples $\mathcal{X}$ which also depends on the layer parameters

For back-propagation, we will need to generate the following terms

$$\frac{\partial Norm\left(\boldsymbol{x}, \mathcal{X}\right)}{\partial \boldsymbol{x}} \text{ and } \frac{\partial Norm\left(\boldsymbol{x}, \mathcal{X}\right)}{\partial \mathcal{X}}$$

Normalization via Mini-Batch Statistic

Problem!!!

- whitening the layer inputs is expensive, as it requires computing the covariance matrix

$$Cov\left[\boldsymbol{x}\right] = E_{\boldsymbol{x}\in\mathcal{X}}\left[\boldsymbol{x}\boldsymbol{x}^T\right] \text{ and } E\left[\boldsymbol{x}\right]E\left[\boldsymbol{x}\right]^T$$

  - To produce the whitened activations

Therefore

A Better Options, we can normalize each dimension

$$\widehat{\boldsymbol{x}}^{(k)} = \frac{\boldsymbol{x}^{(k)} - \mu}{\sigma}$$

- with $\mu = E\left[\boldsymbol{x}^{(k)}\right]$ and $\sigma^2 = Var\left[\boldsymbol{x}^{(k)}\right]$

This allows to speed up convergence

- Simply normalizing each input of a layer may change what the layer can represent.

So, we need to insert a transformation in the network

- Which can represent the identity transform

Therefore

A Better Options, we can normalize each dimension

$$\widehat{\boldsymbol{x}}^{(k)} = \frac{\boldsymbol{x}^{(k)} - \mu}{\sigma}$$

- with $\mu = E\left[\boldsymbol{x}^{(k)}\right]$ and $\sigma^2 = Var\left[\boldsymbol{x}^{(k)}\right]$

This allows to speed up convergence

- Simply normalizing each input of a layer may change what the layer can represent.

So, we need to insert a transformation in the network

- Which can represent the identity transform

Therefore

A Better Options, we can normalize each dimension

$$\widehat{\boldsymbol{x}}^{(k)} = \frac{\boldsymbol{x}^{(k)} - \mu}{\sigma}$$

- with $\mu = E\left[\boldsymbol{x}^{(k)}\right]$ and $\sigma^2 = Var\left[\boldsymbol{x}^{(k)}\right]$

This allows to speed up convergence

- Simply normalizing each input of a layer may change what the layer can represent.

So, we need to insert a transformation in the network

- Which can represent the identity transform

The Transformation

The Linear transformation

$$\boldsymbol{y}^{(k)} = \gamma^{(k)}\widehat{\boldsymbol{x}}^{(k)} + \beta^{(k)}$$

The parameters $\gamma^{(k)}, \beta^{(k)}$

- This allow to recover the identity by setting $\gamma^{(k)} = \sqrt{Var\left[x^{(k)}\right]}$ and $\beta^{(k)} = E\left[x^{(k)}\right]$ if necessary.

The Transformation

The Linear transformation

$$\boldsymbol{y}^{(k)} = \gamma^{(k)}\widehat{\boldsymbol{x}}^{(k)} + \beta^{(k)}$$

The parameters $\gamma^{(k)}, \beta^{(k)}$

- This allow to recover the identity by setting $\gamma^{(k)} = \sqrt{Var\left[\boldsymbol{x}^{(k)}\right]}$ and $\beta^{(k)} = E\left[\boldsymbol{x}^{(k)}\right]$ if necessary.

Finally

Batch Normalizing Transform

Input: Values of $\boldsymbol{x}$ over a mini-batch: $\mathcal{B} = \{\boldsymbol{x}_{1\ldots m}\}$, Parameters to be learned: γ, β

Output: $\{y_i = BN_{\gamma,\beta}(\boldsymbol{x}_i)\}$

1. $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i$
2. $\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2$
3. $\widehat{x} = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$
4. $y_i = \gamma^{(k)}\widehat{x}_i + \beta = BN_{\gamma,\beta}(x_i)$

Finally

Batch Normalizing Transform

Input: Values of $\boldsymbol{x}$ over a mini-batch: $\mathcal{B} = \{\boldsymbol{x}_{1 \dots m}\}$, Parameters to be learned: γ, β

Output: $\{y_i = BN_{\gamma,\beta}\left(\boldsymbol{x}_i\right)\}$

1. $\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}_i$
2. $\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{x}_i - \mu\right)^2$
3. $\widehat{\boldsymbol{x}} = \frac{\boldsymbol{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$
4. $y_i = \gamma^{(k)} \widehat{\boldsymbol{x}}_i + \beta = BN_{\gamma,\beta}\left(\boldsymbol{x}_i\right)$

Finally

Batch Normalizing Transform

Input: Values of $\boldsymbol{x}$ over a mini-batch: $\mathcal{B} = \{\boldsymbol{x}_{1\ldots m}\}$, Parameters to be learned: γ, β

Output: $\{y_i = BN_{\gamma,\beta}(\boldsymbol{x}_i)\}$

1. $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i$
2. $\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (\boldsymbol{x}_i - \mu)^2$
3. $\widehat{\boldsymbol{x}} = \frac{\boldsymbol{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$
4. $y_i = \gamma^{(k)}\widehat{\boldsymbol{x}}_i + \beta = BN_{\gamma,\beta}(\boldsymbol{x}_i)$

Finally

Batch Normalizing Transform

Input: Values of $\boldsymbol{x}$ over a mini-batch: $\mathcal{B} = \{\boldsymbol{x}_{1\ldots m}\}$, Parameters to be learned: γ, β

Output: $\{y_i = BN_{\gamma,\beta}(\boldsymbol{x}_i)\}$

1. $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i$
2. $\sigma^2_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} (\boldsymbol{x}_i - \mu)^2$
3. $\widehat{\boldsymbol{x}} = \frac{\boldsymbol{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$
4. $y_i = \gamma^{(k)}\widehat{\boldsymbol{x}}_i + \beta = BN_{\gamma,\beta}(\boldsymbol{x}_i)$

Finally

Batch Normalizing Transform

Input: Values of $\boldsymbol{x}$ over a mini-batch: $\mathcal{B} = \{\boldsymbol{x}_{1\ldots m}\}$, Parameters to be learned: γ, β

Output: $\{y_i = BN_{\gamma,\beta}(\boldsymbol{x}_i)\}$

1. $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i$
2. $\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (\boldsymbol{x}_i - \mu)^2$
3. $\widehat{\boldsymbol{x}} = \frac{\boldsymbol{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$
4. $\boldsymbol{y}_i = \gamma^{(k)}\widehat{\boldsymbol{x}}_i + \beta = BN_{\gamma,\beta}(\boldsymbol{x}_i)$

Backpropagation

We have the following equations by using the loss function l

1. $\frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} = \frac{\partial l}{\partial \boldsymbol{y}_i} \times \gamma$
2. $\frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times (\boldsymbol{x}_i - \mu_{\mathcal{B}}) \times \left(-\frac{1}{2}\right) \times \left(\sigma_{\mathcal{B}}^2 + \epsilon\right)^{-\frac{3}{2}}$
3. $\frac{\partial l}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2+\epsilon}}\right) + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{\sum_{i=1}^{m} -2\times(\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m}$
4. $\frac{\partial l}{\partial \boldsymbol{x}_i} = \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2+\epsilon}} + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{2\times(\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m} + \frac{\partial l}{\partial \mu_{\mathcal{B}}} \times \frac{1}{m}$
5. $\frac{\partial l}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i} \times \widehat{\boldsymbol{x}}_i$
6. $\frac{\partial l}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i}$

Backpropagation

We have the following equations by using the loss function l

1. $\frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} = \frac{\partial l}{\partial \boldsymbol{y}_i} \times \gamma$
2. $\frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times (\boldsymbol{x}_i - \mu_{\mathcal{B}}) \times \left(-\frac{1}{2}\right) \times \left(\sigma_{\mathcal{B}}^2 + \epsilon\right)^{-\frac{3}{2}}$
3. $\frac{\partial l}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}\right) + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{\sum_{i=1}^{m} -2 \times (\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m}$
4. $\frac{\partial l}{\partial \boldsymbol{x}_i} = \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{2 \times (\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m} + \frac{\partial l}{\partial \mu_{\mathcal{B}}} \times \frac{1}{m}$
5. $\frac{\partial l}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i} \times \widehat{\boldsymbol{x}}_i$
6. $\frac{\partial l}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i}$

Backpropagation

We have the following equations by using the loss function l

1. $\frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} = \frac{\partial l}{\partial \boldsymbol{y}_i} \times \gamma$
2. $\frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times (\boldsymbol{x}_i - \mu_{\mathcal{B}}) \times \left(-\frac{1}{2}\right) \times \left(\sigma_{\mathcal{B}}^2 + \epsilon\right)^{-\frac{3}{2}}$
3. $\frac{\partial l}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2+\epsilon}}\right) + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{\sum_{i=1}^{m} -2\times(\boldsymbol{x}_i-\mu_{\mathcal{B}})}{m}$
4. $\frac{\partial l}{\partial \boldsymbol{x}_i} = \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2+\epsilon}} + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{2\times(\boldsymbol{x}_i-\mu_{\mathcal{B}})}{m} + \frac{\partial l}{\partial \mu_{\mathcal{B}}} \times \frac{1}{m}$
5. $\frac{\partial l}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i} \times \widehat{\boldsymbol{x}}_i$
6. $\frac{\partial l}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i}$

Backpropagation

We have the following equations by using the loss function l

1. $\frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} = \frac{\partial l}{\partial \boldsymbol{y}_i} \times \gamma$
2. $\frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times (\boldsymbol{x}_i - \mu_{\mathcal{B}}) \times \left(-\frac{1}{2}\right) \times \left(\sigma_{\mathcal{B}}^2 + \epsilon\right)^{-\frac{3}{2}}$
3. $\frac{\partial l}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2+\epsilon}}\right) + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{\sum_{i=1}^{m} -2\times(\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m}$
4. $\frac{\partial l}{\partial \boldsymbol{x}_i} = \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2+\epsilon}} + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{2\times(\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m} + \frac{\partial l}{\partial \mu_{\mathcal{B}}} \times \frac{1}{m}$
5. $\frac{\partial l}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i} \times \widehat{\boldsymbol{x}}_i$
6. $\frac{\partial l}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i}$

Backpropagation

We have the following equations by using the loss function l

1. $\frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} = \frac{\partial l}{\partial \boldsymbol{y}_i} \times \gamma$
2. $\frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times (\boldsymbol{x}_i - \mu_{\mathcal{B}}) \times \left(-\frac{1}{2}\right) \times \left(\sigma_{\mathcal{B}}^2 + \epsilon\right)^{-\frac{3}{2}}$
3. $\frac{\partial l}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}\right) + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{\sum_{i=1}^{m} -2 \times (\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m}$
4. $\frac{\partial l}{\partial \boldsymbol{x}_i} = \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{2 \times (\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m} + \frac{\partial l}{\partial \mu_{\mathcal{B}}} \times \frac{1}{m}$
5. $\frac{\partial l}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i} \times \widehat{\boldsymbol{x}}_i$
6. $\frac{\partial l}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i}$

Backpropagation

We have the following equations by using the loss function l

1. $\frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} = \frac{\partial l}{\partial \boldsymbol{y}_i} \times \gamma$
2. $\frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^m \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times (\boldsymbol{x}_i - \mu_{\mathcal{B}}) \times \left(-\frac{1}{2}\right) \times \left(\sigma_{\mathcal{B}}^2 + \epsilon\right)^{-\frac{3}{2}}$
3. $\frac{\partial l}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^m \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2+\epsilon}}\right) + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{\sum_{i=1}^m -2\times(\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m}$
4. $\frac{\partial l}{\partial \boldsymbol{x}_i} = \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2+\epsilon}} + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{2\times(\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m} + \frac{\partial l}{\partial \mu_{\mathcal{B}}} \times \frac{1}{m}$
5. $\frac{\partial l}{\partial \gamma} = \sum_{i=1}^m \frac{\partial l}{\partial \boldsymbol{y}_i} \times \widehat{\boldsymbol{x}}_i$
6. $\frac{\partial l}{\partial \beta} = \sum_{i=1}^m \frac{\partial l}{\partial \boldsymbol{y}_i}$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{ \boldsymbol{x}^{(k)} \right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

$N_{BN}^{tr} = N$ // Training BN network
for $k = 1 \ldots K$ do
    Add transformation $y^{(k)} = BN_{\gamma^{(k)}, \beta^{(k)}}\left(x^{(k)}\right)$ to N_{BN}^{tr}
    Modify each layer in N_{BN}^{tr} with input $x^{(k)}$ to take $y^{(k)}$instead
Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{ \gamma^{(k)}, \beta^{(k)} \right\}_{k=1}^{K}$
$N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
for $k = 1 \ldots K$ do
    Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
    $E[x] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[x] = \frac{m}{m-1}{}_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^{2}\right]$
    In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(x)$ with

$$y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \times x + \left[\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}}\right]$$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{x^{(k)}\right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1 \ldots K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}\left(x^{(k)}\right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $x^{(k)}$ to take $y^{(k)}$instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{\gamma^{(k)}, \beta^{(k)}\right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1 \ldots K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[x] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[x] = \frac{m}{m-1}{}_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^{2}\right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(x)$ with
11. $$y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \times x + \left[\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}}\right]$$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{x^{(k)}\right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1 \ldots K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}\left(x^{(k)}\right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $x^{(k)}$ to take $y^{(k)}$instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{\gamma^{(k)}, \beta^{(k)}\right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1 \ldots K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[x] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[x] = \frac{m}{m-1} E_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^2\right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(x)$ with
11. $$y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \times x + \left[\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}}\right]$$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{\boldsymbol{x}^{(k)}\right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1 \ldots K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}\left(\boldsymbol{x}^{(k)}\right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $x^{(k)}$ to take $y^{(k)}$instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{\gamma^{(k)}, \beta^{(k)}\right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1 \ldots K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[\boldsymbol{x}] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[\boldsymbol{x}] = \frac{m}{m-1}{}_{\mathcal{B}}\left[\sigma^2_{\mathcal{B}}\right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(\boldsymbol{x})$ with
11. $$y = \frac{\gamma}{\sqrt{Var[\boldsymbol{x}]+\epsilon}} \times \boldsymbol{x} + \left[\beta - \frac{\gamma E[\boldsymbol{x}]}{\sqrt{Var[\boldsymbol{x}]+\epsilon}}\right]$$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{\boldsymbol{x}^{(k)}\right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1...K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}\left(\boldsymbol{x}^{(k)}\right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $\boldsymbol{x}^{(k)}$ to take $\boldsymbol{y}^{(k)}$ instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{\gamma^{(k)}, \beta^{(k)}\right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1...K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[x] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[x] = \frac{m}{m-1}{}_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^2\right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(x)$ with
11. $$y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \times x + \left[\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}}\right]$$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{\boldsymbol{x}^{(k)}\right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1...K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}\left(\boldsymbol{x}^{(k)}\right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $\boldsymbol{x}^{(k)}$ to take $\boldsymbol{y}^{(k)}$instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{\gamma^{(k)}, \beta^{(k)}\right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1...K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[x] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[x] = \frac{m}{m-1}{}_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^2\right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(x)$ with
11. $$y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \times x + \left[\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}}\right]$$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{\boldsymbol{x}^{(k)}\right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1 ... K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}\left(\boldsymbol{x}^{(k)}\right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $\boldsymbol{x}^{(k)}$ to take $\boldsymbol{y}^{(k)}$instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{\gamma^{(k)}, \beta^{(k)}\right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1 ... K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[x] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[x] = \frac{m}{m-1}{}_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^2\right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(x)$ with
11. $$y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \times x + \left[\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}}\right]$$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{ \boldsymbol{x}^{(k)} \right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1 \ldots K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)}, \beta^{(k)}} \left(\boldsymbol{x}^{(k)} \right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $\boldsymbol{x}^{(k)}$ to take $\boldsymbol{y}^{(k)}$ instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{ \gamma^{(k)}, \beta^{(k)} \right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1 \ldots K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[x] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[x] = \frac{m}{m-1}_{\mathcal{B}} \left[\sigma_{\mathcal{B}}^2 \right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma, \beta}(x)$ with
11. $$y = \frac{\gamma}{\sqrt{Var[x] + \epsilon}} \times x + \left[\beta - \frac{\gamma E[x]}{\sqrt{Var[x] + \epsilon}} \right]$$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{\boldsymbol{x}^{(k)}\right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1 \ldots K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}\left(\boldsymbol{x}^{(k)}\right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $\boldsymbol{x}^{(k)}$ to take $\boldsymbol{y}^{(k)}$ instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{\gamma^{(k)}, \beta^{(k)}\right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1 \ldots K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[x] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[x] = \frac{m}{m-1} E_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^2\right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(x)$ with
11. $$y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \times x + \left[\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}}\right]$$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{\boldsymbol{x}^{(k)}\right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1 \ldots K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}\left(\boldsymbol{x}^{(k)}\right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $\boldsymbol{x}^{(k)}$ to take $\boldsymbol{y}^{(k)}$instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{\gamma^{(k)}, \beta^{(k)}\right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1 \ldots K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[\boldsymbol{x}] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[\boldsymbol{x}] = \frac{m}{m-1}{}_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^2\right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(x)$ with
11. $y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \times x + \left[\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}}\right]$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{\boldsymbol{x}^{(k)}\right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1...K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}\left(\boldsymbol{x}^{(k)}\right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $\boldsymbol{x}^{(k)}$ to take $\boldsymbol{y}^{(k)}$instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{\gamma^{(k)}, \beta^{(k)}\right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1...K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[\boldsymbol{x}] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[\boldsymbol{x}] = \frac{m}{m-1}{}_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^2\right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(\boldsymbol{x})$ with
11. $$\boldsymbol{y} = \frac{\gamma}{\sqrt{Var[\boldsymbol{x}]+\epsilon}} \times \boldsymbol{x} + \left[\beta - \frac{\gamma E[\boldsymbol{x}]}{\sqrt{Var[\boldsymbol{x}]+\epsilon}}\right]$$

However

Santurkar et al. [18]

- They found thats is not the covariance shift the one affected by it!!!

Santurkar et al. recognize that

- Batch normalization has been arguably one of the most successful architectural innovations in deep learning.

They used a standard Very deep convolutional network

- on CIFAR-10 with and without BatchNorm

However

Santurkar et al. [18]

- They found thats is not the covariance shift the one affected by it!!!

Santurkar et al. recognize that

- Batch normalization has been arguably one of the most successful architectural innovations in deep learning.

They used a standard Very deep convolutional network

- on CIFAR-10 with and without BatchNorm

However

Santurkar et al. [18]

- They found thats is not the covariance shift the one affected by it!!!

Santurkar et al. recognize that

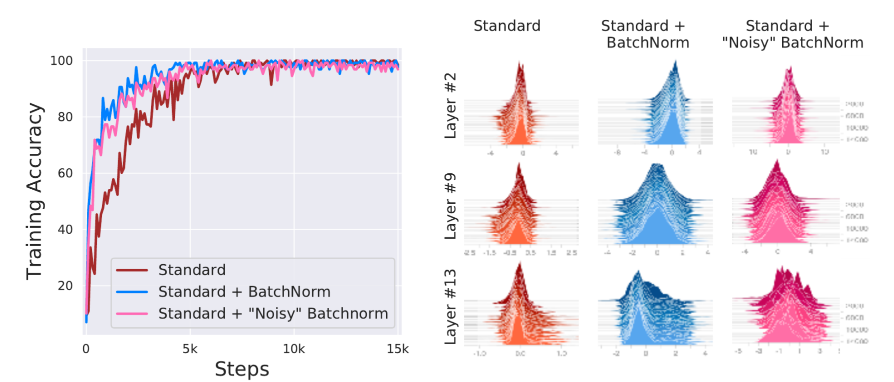
- Batch normalization has been arguably one of the most successful architectural innovations in deep learning.

They used a standard Very deep convolutional network

- on CIFAR-10 with and without BatchNorm

They found something quite interesting

The following facts



Actually Batch Normalization

It does not do anything to the Internal Covariate Shift

- Actually smooth the optimization manifold
  - It is not the only way to achieve it!!!

They suggest that

- "This suggests that the positive impact of BatchNorm on training might be somewhat serendipitous."

Actually Batch Normalization

It does not do anything to the Internal Covariate Shift

- Actually smooth the optimization manifold
  - It is not the only way to achieve it!!!

They suggest that

- "This suggests that the positive impact of BatchNorm on training might be somewhat serendipitous."

They actually have a connected result

To the analysis of gradient clipping!!!

- They are the same group

Theorem (The effect of BatchNorm on the Lipschitzness of the loss)

- For a BatchNorm network with loss $\hat{\mathcal{L}}$ and an identical non-BN network with (identical) loss $\mathcal{L}$,

$$\left\|\nabla_{y_j}\hat{\mathcal{L}}\right\|^2 \leq \frac{\gamma^2}{\sigma_j^2}\left[\left\|\nabla_{y_j}\mathcal{L}\right\|^2 - \frac{1}{m}\left\langle \mathbf{1}, \nabla_{y_j}\mathcal{L}\right\rangle^2 - \frac{1}{\sqrt{m}}\left\langle \nabla_{y_j}\mathcal{L}, \hat{y}_j\right\rangle^2\right]$$

They actually have a connected result

To the analysis of gradient clipping!!!

- They are the same group

Theorem (The effect of BatchNorm on the Lipschitzness of the loss)

- For a BatchNorm network with loss $\widehat{\mathcal{L}}$ and an identical non-BN network with (identical) loss $\mathcal{L}$,

$$\left\|\nabla_{\boldsymbol{y}_j}\widehat{\mathcal{L}}\right\|^2 \leq \frac{\gamma^2}{\sigma_j^2}\left[\left\|\nabla_{y_j}\mathcal{L}\right\|^2 - \frac{1}{m}\left\langle \mathbf{1}, \nabla_{y_j}\mathcal{L}\right\rangle^2 - \frac{1}{\sqrt{m}}\left\langle \nabla_{y_j}\mathcal{L}, \widehat{\boldsymbol{y}}_j\right\rangle^2\right]$$

Outline

Definition

Degradation Problem

- With the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly.

Something Notable

- Unexpectedly, such degradation is not caused by overfitting,

and adding more layers

- to a suitably deep model leads to higher training error,

Definition

Degradation Problem

- With the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly.

Something Notable

- Unexpectedly, such degradation is not caused by overfitting,

and adding more layers

- to a suitably deep model leads to higher training error,

Definition

Degradation Problem

- With the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly.

Something Notable

- Unexpectedly, such degradation is not caused by overfitting,

and adding more layers

- to a suitably deep model leads to higher training error,

Outline

Therefore, we need to deal with such problems

The Residual Network [16]

- He, Kaiming et al. - "Deep Residual Learning for Image Recognition"

Basically they got two layers doing something to an input

$$\mathcal{F}(\boldsymbol{x}) = A_2 A_1 \boldsymbol{x}$$

Then imagine you have an ideal mapping $\mathcal{H}(\boldsymbol{x})$

$$\mathcal{F}(\boldsymbol{x}) = \mathcal{H}(\boldsymbol{x}) - \boldsymbol{x} \Longrightarrow \mathcal{F}(\boldsymbol{x}) + \boldsymbol{x} = \mathcal{H}(\boldsymbol{x}) \Longrightarrow$$

Therefore, we need to deal with such problems

The Residual Network [16]

- He, Kaiming et al. - "Deep Residual Learning for Image Recognition"

Basically they got two layers doing something to an input

$$\mathcal{F}(\boldsymbol{x}) = A_2 A_1 \boldsymbol{x}$$

Then imagine you have an ideal mapping $\mathcal{H}(\boldsymbol{x})$

$$\mathcal{F}(\boldsymbol{x}) = \mathcal{H}(\boldsymbol{x}) - \boldsymbol{x} \Longrightarrow \mathcal{F}(\boldsymbol{x}) + \boldsymbol{x} = \mathcal{H}(\boldsymbol{x}) \Longrightarrow$$

Therefore, we need to deal with such problems

The Residual Network [16]

- He, Kaiming et al. - "Deep Residual Learning for Image Recognition"

Basically they got two layers doing something to an input

$$\mathcal{F}(\boldsymbol{x}) = A_2 A_1 \boldsymbol{x}$$

Then imagine you have an ideal mapping $\mathcal{H}(x)$

$$\mathcal{F}(\boldsymbol{x}) = \mathcal{H}(\boldsymbol{x}) - \boldsymbol{x} \Longrightarrow \mathcal{F}(\boldsymbol{x}) + \boldsymbol{x} = \mathcal{H}(\boldsymbol{x}) \Longrightarrow$$

Basically

This allows to

- Motivation for skipping over layers is to avoid the problem of vanishing gradients.

Something Notable

- In the simplest case, only the weights for the adjacent layer's connection are adapted.

Basically

This allows to

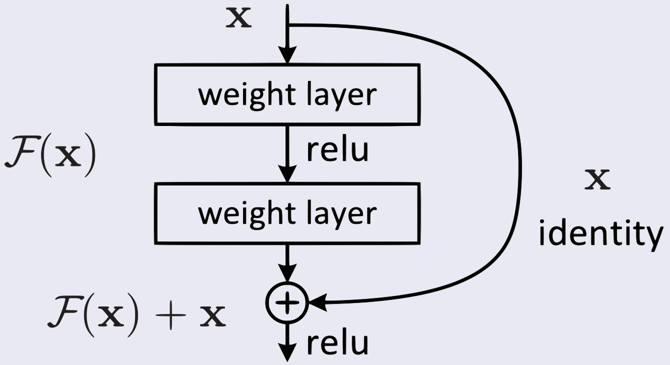
- Motivation for skipping over layers is to avoid the problem of vanishing gradients.

Something Notable

- In the simplest case, only the weights for the adjacent layer's connection are adapted.

Blocks of the Original RNN

We have



A Winner

Something Notable

- Winner of ILSVRC 2015 in image classification, detection, and localization, as well as Winner of MS COCO 2015 detection, and segmentation.

Outline

We have seen many concepts

Deep Forward Networks

- Although a simple idea

They represent a rich field of study

- Basically... From Lower Complexity Features toward more complex more informative!!!

In conclusion

- Deep Forward Networks look to have more expressibility than shallow learners.

We have seen many concepts

Deep Forward Networks

- Although a simple idea

They represent a rich field of study

- Basically... From Lower Complexity Features toward more complex more informative!!!

In conclusion

- Deep Forward Networks look to have more expressibility than shallow learners.

We have seen many concepts

Deep Forward Networks

- Although a simple idea

They represent a rich field of study

- Basically... From Lower Complexity Features toward more complex more informative!!!

In conclusion

- Deep Forward Networks look to have more expressibility than shallow learners.

C. E. Shannon, "A symbolic analysis of relay and switching circuits," *Electrical Engineering*, vol. 57, no. 12, pp. 713–723, 1938.

E. Mendelson, *Introduction to mathematical logic*. Chapman and Hall/CRC, 2009.

J. Hastad, "Almost optimal lower bounds for small depth circuits," in *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pp. 6–20, Citeseer, 1986.

J. Håstad and M. Goldmann, "On the power of small-depth threshold circuits," *Computational Complexity*, vol. 1, no. 2, pp. 113–129, 1991.

Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

M. Gönen and E. Alpaydın, "Multiple kernel learning algorithms," *Journal of machine learning research*, vol. 12, no. Jul, pp. 2211–2268, 2011.

G. R. G. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. I. Jordan, "Learning the kernel matrix with semidefinite programming," *J. Mach. Learn. Res.*, vol. 5, pp. 27–72, Dec. 2004.

Y. Bengio, O. Delalleau, and N. L. Roux, "The curse of highly variable functions for local kernel machines," in *Advances in neural information processing systems*, pp. 107–114, 2006.

Y. Bengio, Y. LeCun, *et al.*, "Scaling learning algorithms towards ai," *Large-scale kernel machines*, vol. 34, no. 5, pp. 1–41, 2007.

Z. Zhang, "Derivation of backpropagation in convolutional neural network (cnn)," *University of Tennessee, Knoxville, TN*, 2016.

X. Peng, H. Cao, and P. Natarajan, "Using convolutional encoder-decoder for document image binarization," in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 1, pp. 708–713, IEEE, 2017.

P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell, "Understanding convolution for semantic segmentation," in *2018 IEEE winter conference on applications of computer vision (WACV)*, pp. 1451–1460, IEEE, 2018.

V. Podlozhnyuk, "Image convolution with cuda," *NVIDIA Corporation white paper, June*, vol. 2097, no. 3, 2007.

X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, 2011.

I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

- S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.
- S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?," in *Advances in Neural Information Processing Systems*, pp. 2483–2493, 2018.
- C. Gulcehre, M. Moczulski, M. Denil, and Y. Bengio, "Noisy activation functions," in *International conference on machine learning*, pp. 3059–3068, 2016.
- S. Sharma, "Activation functions in neural networks," *Towards Data Science*, vol. 6, 2017.
- K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

J. J. Weng, N. Ahuja, and T. S. Huang, "Learning recognition and segmentation of 3-d objects from 2-d images," in *1993 (4th) International Conference on Computer Vision*, pp. 121–128, IEEE, 1993.

J. J. Weng, N. Ahuja, and T. S. Huang, "Learning recognition and segmentation using the cresceptron," *International Journal of Computer Vision*, vol. 25, no. 2, pp. 109–143, 1997.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.

- Y. Li, S. Liu, J. Yang, and M.-H. Yang, "Generative face completion," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3911–3919, 2017.
- J. Pennington, S. S. Schoenholz, and S. Ganguli, "The emergence of spectral universality in deep networks," *arXiv preprint arXiv:1802.09979*, 2018.
- J. Zhang, T. He, S. Sra, and A. Jadbabaie, "Analysis of gradient clipping and adaptive scaling with a relaxed smoothness condition," *arXiv preprint arXiv:1905.11881*, 2019.
- Y. Bengio and Y. Le Cun, "Word normalization for on-line handwritten word recognition," in *International Conference on Pattern Recognition*, pp. 409–409, IEEE COMPUTER SOCIETY PRESS, 1994.