

# Introduction to Deep Learning

## Regularization and Loss Functions

Andres Mendez-Vazquez

July 6, 2025

# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
  - Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
  - Random Dropout Probability
  - Projecting Noise into Input Space
  - Augmenting by Noise
  - Co-adaptation/Overfitting
  - Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
  - For More in Normalization

# Outline

1

## Problems with Deeper Architectures

### The Degradation Problem

- The Vanishing and Exploding Gradients
- An Example of The Vanishing and Exploding Gradient
- Vanishing and Exploding Gradient as Fixed Points

2

## The Problem with Overfitting

- Intuition from Overfitting
- The Idea of Regularization
- The Horrible Landscape of Optimization in Deep Learning

3

## Methods for Improving Learning in Deep Networks

- Gradient Clipping for Exploding Gradient
- The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
- Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
- Random Dropout Probability
- Projecting Noise into Input Space
- Augmenting by Noise
- Co-adaptation/Overfitting
- Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
- For More in Normalization

# Definition

## Degradation Problem

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

# 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,

# 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

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - **The Vanishing and Exploding Gradients**
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
- Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
- Random Dropout Probability
- Projecting Noise into Input Space
- Augmenting by Noise
- Co-adaptation/Overfitting
- Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
- For More in Normalization

## As We know

In Recurrent Neural Networks, we have the problem

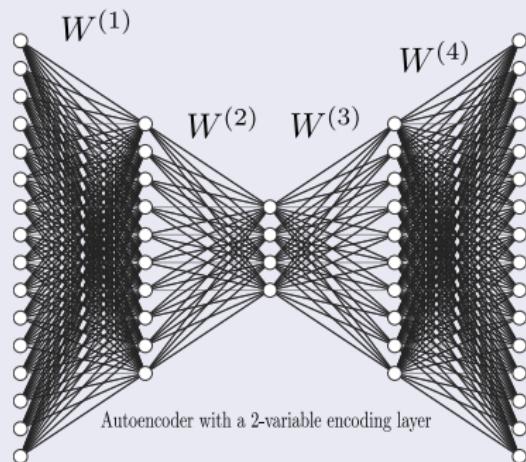
- Vanishing and Exploding Gradients

# As We know

In Recurrent Neural Networks, we have the problem

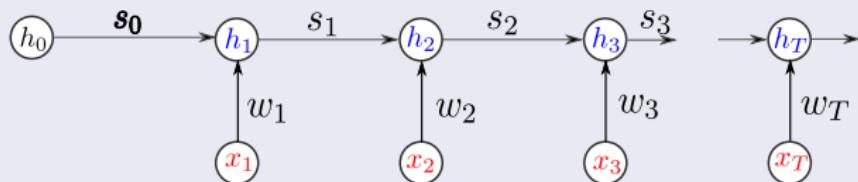
- Vanishing and Exploding Gradients

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



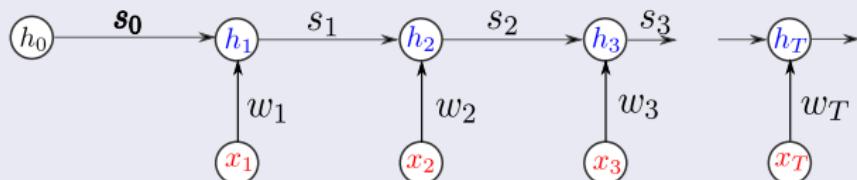
# Consider a simple encoder encoder network

We have this simplified version



# Consider a simple encoder encoder network

We have this simplified version



We have the following structure

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

$$z_t = s_{t-1} h_{t-1}$$

# Backpropagation Rules

Then, we get the following backpropagation rules by chain rule

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

Then, we have

By Using Our simplifying assumption that

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

Then, we have

By Using Our simplifying assumption that

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

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

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

Then, we have

By Using Our simplifying assumption that

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

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  $k$ .

# 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

## 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

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
  - Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
- Random Dropout Probability
  - Projecting Noise into Input Space
  - Augmenting by Noise
  - Co-adaptation/Overfitting
- Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
- For More in Normalization

## We have noticed that many layers in Deep Learning

They are similar, and can be problematic. For example,

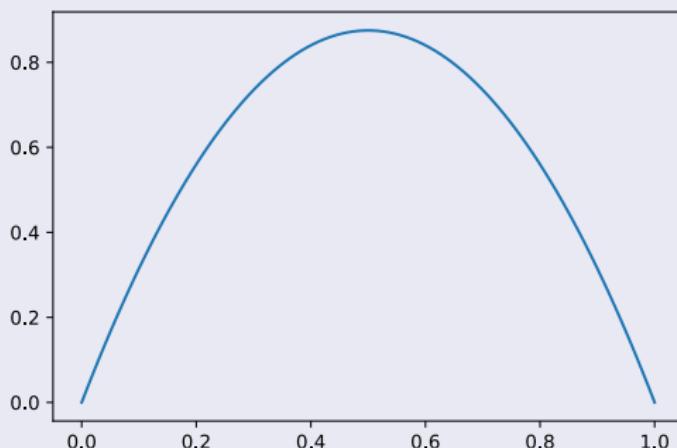
$$f(x) = 3.5x(1 - x)$$

We have noticed that many layers in Deep Learning

They are similar, and can be problematic. For example,

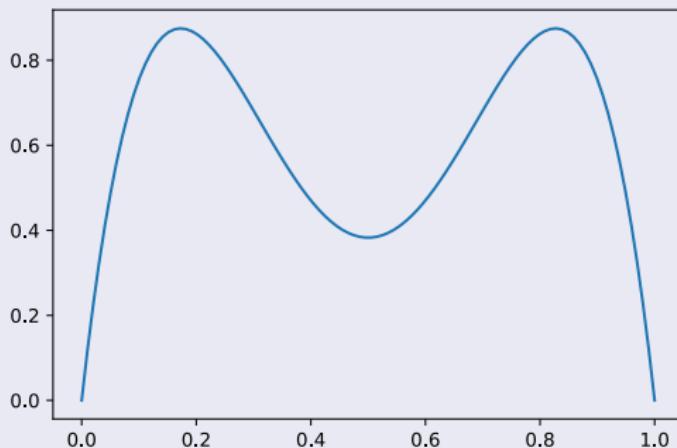
$$f(x) = 3.5x(1 - x)$$

In the first composition, we get



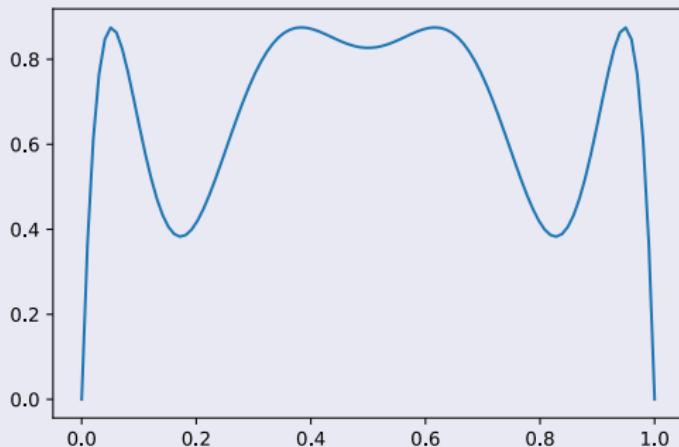
Now, as we compound the function

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



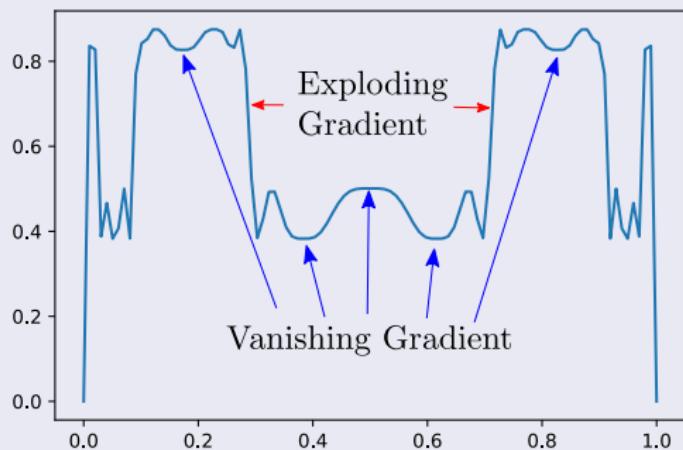
Now, as we increment iterations

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



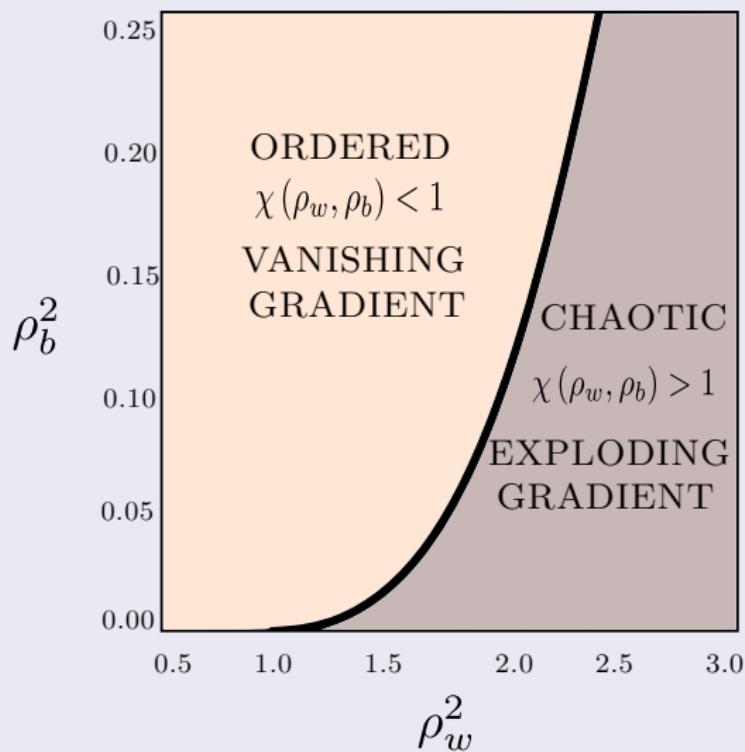
# Finally

We see the increment in the gradient part negative or positive



Actually, we have

## A Frontier defining the Vanishing and Exploding Gradient [1]



# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - **Vanishing and Exploding Gradient as Fixed Points**
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
  - Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
- Random Dropout Probability
  - Projecting Noise into Input Space
  - Augmenting by Noise
  - Co-adaptation/Overfitting
- Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
- For More in Normalization

# Vanishing and Exploding Gradient as Fixed Points

We can see them as

- A fixed point...

# Vanishing and Exploding Gradient as Fixed Points

We can see them as

- A fixed point...

A Fixed Point?

$$x = f(x)$$

# Basically

The fixed points can be thought

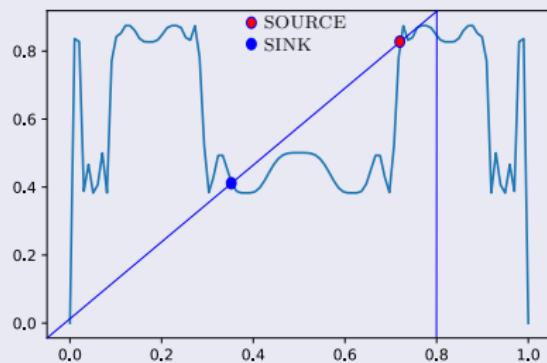
- Some fixed points repel the iterates; **these are called sources.**
- Other fixed points attract the iterates; **these are called sinks.**

# Basically

The fixed points can be thought

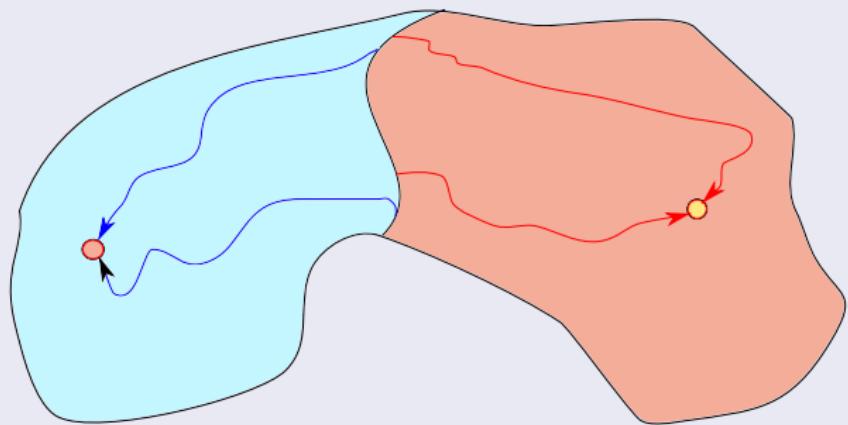
- Some fixed points repel the iterates; **these are called sources.**
- Other fixed points attract the iterates; **these are called sinks.**

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

$$\mathbf{h}_t = W_{sd} \mathbf{x}_t + U_{ss} \mathbf{h}_{t-1}$$

$$\mathbf{y}_t = V_{os} \mathbf{h}_t$$

# These fixed points

In Deep Structures as RNN without sigmoid functions

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

We have

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

## These fixed points

In Deep Structures as RNN without sigmoid functions

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

We have

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

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

- Then, we have that

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

# Therefore

We have that

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

## They define an area

Where  $V_{os}$  and  $W_{sd}$

- They are the inverse of each other

## 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

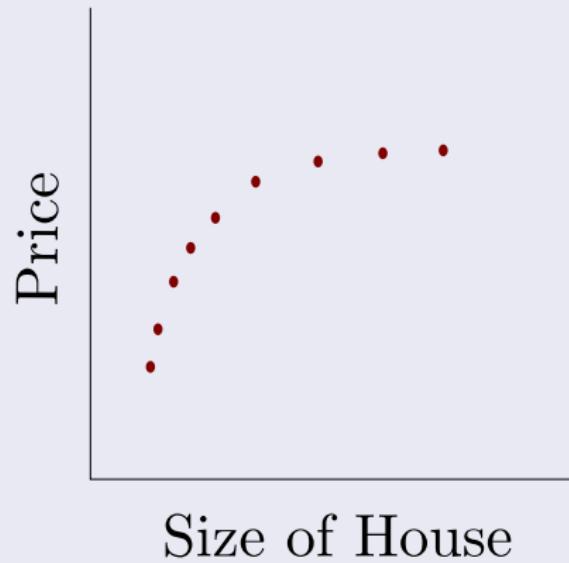
# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
  - Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
- Random Dropout Probability
  - Projecting Noise into Input Space
  - Augmenting by Noise
  - Co-adaptation/Overfitting
- Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
- For More in Normalization

# The house example (From Andrew Ng Course)

Imagine the following data set



Now assume that we use a regressor

For the fitting

$$\frac{1}{2} \sum_{i=1}^N (h_{\theta}(x_i) - y_i)^2$$

Now assume that we use a regressor

For the fitting

$$\frac{1}{2} \sum_{i=1}^N (h_{\theta}(x_i) - y_i)^2$$

We can then run one of our machine to see what minimize better the previous equation

Question: Did you notice that I did not impose any structure to  $h_w(x)$ ?

## Then, First fitting

What about using  $h_1(x) = \theta_0 + \theta_1x + \theta_2x^2$ ?



## Second fitting

What about using  $h_2(x) = \theta_0 + \theta_1x + \theta_2x^2 + \theta_3x^3 + \theta_4x^4 + \theta_5x^5$ ?



# Therefore, we have a problem

We get weird over fitting effects!!!

What do we do? What about minimizing the influence of  $\theta_3, \theta_4, \theta_5$ ?

# Therefore, we have a problem

We get weird over fitting effects!!!

What do we do? What about minimizing the influence of  $\theta_3, \theta_4, \theta_5$ ?

How do we do that?

$$\min_{\theta} \frac{1}{2} \sum_{i=1}^N (h_{\theta}(x_i) - y_i)^2$$

What about integrating those values to the cost function? Ideas

# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
  - Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
- Random Dropout Probability
  - Projecting Noise into Input Space
  - Augmenting by Noise
  - Co-adaptation/Overfitting
- Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
- For More in Normalization

We have

Regularization intuition is as follow

Small values for parameters  $\theta_0, \theta_1, \theta_2, \dots, \theta_n$

We have

Regularization intuition is as follow

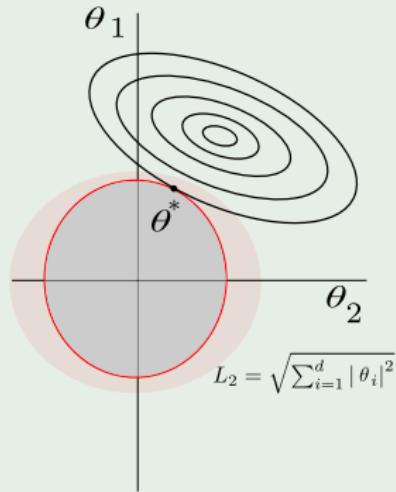
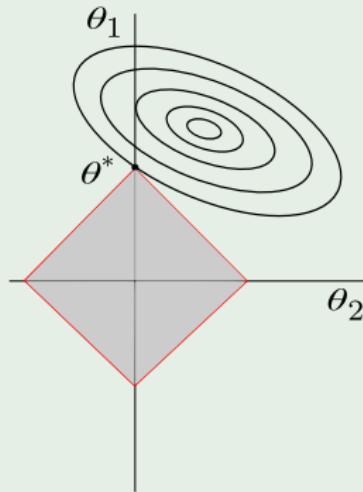
Small values for parameters  $\theta_0, \theta_1, \theta_2, \dots, \theta_n$

It implies

- ① "Simpler" function
- ② Less prone to overfitting

## Graphically

Yes the circle defined as  $\|\theta\|_2 = \sqrt{\sum_{i=1}^d |\theta_i|^2}$



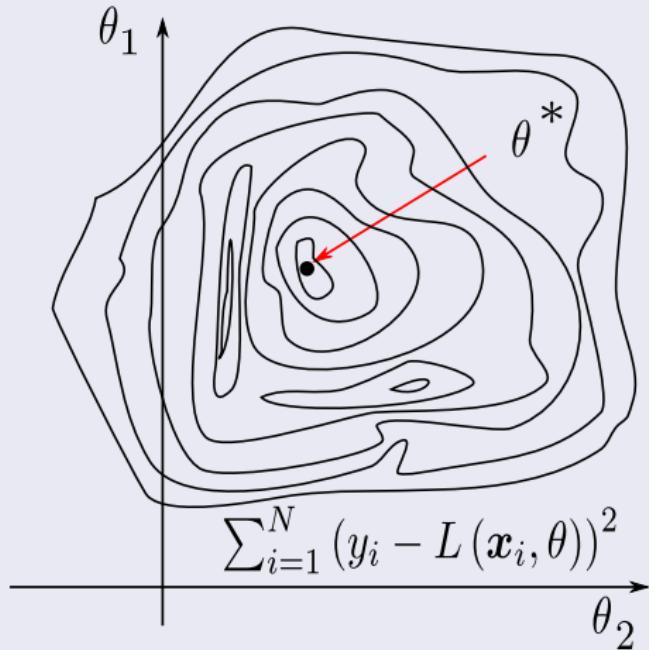
# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
  - Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
- Random Dropout Probability
  - Projecting Noise into Input Space
  - Augmenting by Noise
  - Co-adaptation/Overfitting
- Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
- For More in Normalization

# The Horrible Landscape

The Landscape is highly variable



# Over-fitting?

## Basically (Intuition)

$(y_i - L(\mathbf{x}_i, \theta))^2 = 0$  for  $i \in Training$

$(y_j - L(\mathbf{x}_i, \theta))^2 \gg 0$  for  $i \in Validation$

# Over-fitting?

## Basically (Intuition)

$$(y_i - L(\mathbf{x}_i, \theta))^2 = 0 \text{ for } i \in \text{Training}$$

$$(y_j - L(\mathbf{x}_i, \theta))^2 \gg 0 \text{ for } i \in \text{Validation}$$

On the other side, you have BIAS==Simplification

- Then, Regularization is an operator moving the model toward a bias

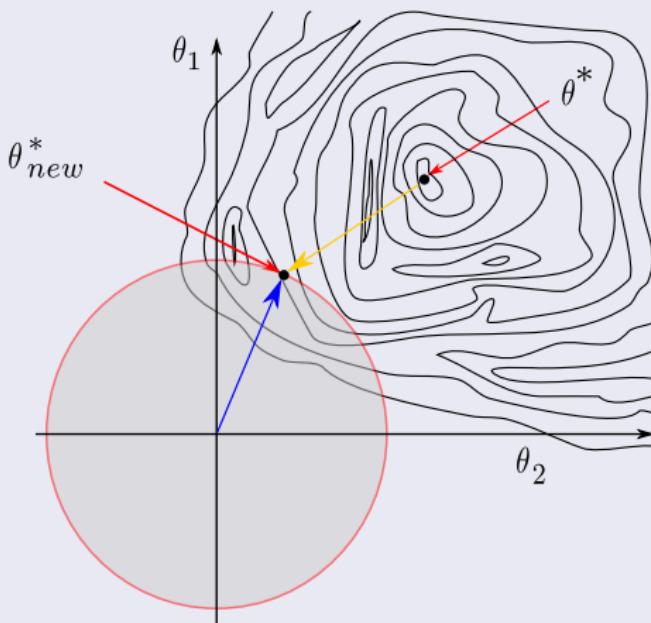
However, we do not want too much simplification

Look at this, the worst case Bias toward Red



Basically this simplification is due to the constrained optimization landscape

Basically our constraint is too Euclidean for the Optimization Landscape



# Well-Posed Problem

## Definition by Hadamard (Circa 1902)

- Models of physical phenomena should have the following properties
  - ① A solution exists,
  - ② The solution is unique,
  - ③ The solution's behavior changes continuously with the initial conditions.

# Well-Posed Problem

## Definition by Hadamard (Circa 1902)

- Models of physical phenomena should have the following properties
  - ① A solution exists,
  - ② The solution is unique,
  - ③ The solution's behavior changes continuously with the initial conditions.

## Any other problem that fails in any of this conditions

- It is considered an Ill-Posed Problem.

It seems to be that

The Deep Learners are highly ill-posed problems

- Ridge and LASSO have two possible effects

It seems to be that

The Deep Learners are highly ill-posed problems

- Ridge and LASSO have two possible effects

Too much simplification

- The Deep Learners losses power of representation.
  - ▶ Weights are eliminated

## It seems to be that

The Deep Learners are highly ill-posed problems

- Ridge and LASSO have two possible effects

Too much simplification

- The Deep Learners losses power of representation.
  - ▶ Weights are eliminated

The constraints forces the  $\theta's$

- They are forced to live in a too smooth optimization landscape

# Outline

1

## Problems with Deeper Architectures

- The Degradation Problem
- The Vanishing and Exploding Gradients
- An Example of The Vanishing and Exploding Gradient
- Vanishing and Exploding Gradient as Fixed Points

2

## The Problem with Overfitting

- Intuition from Overfitting
- The Idea of Regularization
- The Horrible Landscape of Optimization in Deep Learning

3

## Methods for Improving Learning in Deep Networks

### ● Gradient Clipping for Exploding Gradient

- The Correlation between Gradient Norm and Local Smoothness of Deep Learners

●

## Gaussian Noise on Hidden Units for Regularization

- Application into a Decoder/Encoder

● Dropout as Regularization

- Introduction
- Dropout Process
- Dropout as Bagging/Bootstrap Aggregation
- Beyond an Empirical Probabilities, LASSO and Data Flow
- Random Dropout Probability
- Projecting Noise into Input Space
- Augmenting by Noise
- Co-adaptation/Overfitting
- Batch normalization
- Improving the Google Layer Normalization
- Layer Normalization in RNN
- Invariance Under Weights and Data Transformations
- For More in Normalization

## Gradient Clipping AKA accelerate Learning

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

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

# Gradient Clipping AKA accelerate Learning

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 [2]

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

# Gradient Clipping AKA accelerate Learning

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 [2]

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

In general this problem is intractable

- Instead of seeking a global optimum, we try to get an  $\epsilon$ -stationary point

$$\|\nabla f(x)\| \leq \epsilon$$

## Now

We assume the following conditions holds in the neighborhood of the sublevel set  $S^1$  in an initial point  $x_0$

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

# Conditions

## Assumption 1

- Function  $f$  is lower bounded by  $f^*$

# Conditions

## Assumption 1

- Function  $f$  is lower bounded by  $f^*$

## Assumption 2

- Function  $f$  is twice differentiable

# They introduced a relaxed smoothness assumption

## Assumption 3

- $(L_0, L_1)$ -smoothness
  - ▶  $f$  is  $(L_0, L_1)$ -smooth, if there exist  $L_0 > 0$  and  $L_1 > 0$  such that
$$\|\nabla^2 f(x)\| \leq L_0 + L_1 \|\nabla f(x)\|$$

# They introduced a relaxed smoothness assumption

## Assumption 3

- $(L_0, L_1)$ -smoothness
  - ▶  $f$  is  $(L_0, L_1)$ -smooth, if there exist  $L_0 > 0$  and  $L_1 > 0$  such that
$$\|\nabla^2 f(x)\| \leq L_0 + L_1 \|\nabla f(x)\|$$

This is equivalent to

$$\limsup_{\delta \rightarrow 0} \frac{\|\nabla f(x) - \nabla f(x + \delta)\|}{\|\delta\|} \leq L_1 \|\nabla f(x)\| + L_0$$

- This implies Lipschitz continuous

# What?

## Definition (Lipschitz continuous)

- A function  $f : S \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$  is Lipschitz continuous at  $x \in S$  if there is a constant  $C$  such that

$$\|\nabla f(y) - \nabla f(x)\| \leq C \|y - x\|$$

for all  $y \in S$  sufficiently near to  $x$ .

# What?

## Definition (Lipschitz continuous)

- A function  $f : S \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$  is Lipschitz continuous at  $x \in S$  if there is a constant  $C$  such that

$$\|\nabla f(y) - \nabla f(x)\| \leq C \|y - x\|$$

for all  $y \in S$  sufficiently near to  $x$ .

## Using these assumptions

- They did an interesting analysis of the gradient clipping strategies.

Then, there are the following proposals

The ordinary gradient descent (Baseline case)

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

Then, there are the following proposals

The ordinary gradient descent (Baseline case)

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

The Clipped Gradient Descent (CGD)

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

Then, there are the following proposals

### The ordinary gradient descent (Baseline case)

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

### 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

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

Is clipped gradient descent optimized for a different smoothness condition?

## Definition

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

Is clipped gradient descent optimized for a different smoothness condition?

### 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  $f$

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

Is clipped gradient descent optimized for a different smoothness condition?

### 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  $f$

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

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

It is possible to prove the following upper bound

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

Then, it is possible to use the 3 Assumption

It is possible to prove the following upper bound

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

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

$$\begin{aligned}&= f(x) + \nabla^T f(x)(y - x) + \frac{1}{2}L\|y - x\|^2 \\&= f(x) - h\|\nabla f(x)\|^2 + \frac{1}{2}L\|h\nabla f(x)\|^2 \\&= f(x) - \|\nabla f(x)\|^2 \left[ h - \frac{1}{2}Lh^2 \right]\end{aligned}$$

## Then

We have the following situation to minimize this bound

$$h - \frac{1}{2}Lh^2 = 0 \Rightarrow 2 = Lh \Rightarrow h = \frac{2}{L}$$

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

## Then

We have the following situation to minimize this bound

$$h - \frac{1}{2}Lh^2 = 0 \Rightarrow 2 = Lh \Rightarrow h = \frac{2}{L}$$

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

This raises the following question

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

## Now

As step sizes for clipped GD and NGD are related by a constant factor

- For this assume:

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

- Optimizes the quadratic function (The part with  $h$  can be make 0):

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

Then, we have

By substitution

$$h - \frac{L(x)h^2}{2} = \left[ \frac{\eta}{\|\nabla f(x)\| + \beta} \right] - \frac{L(x)}{2} \left[ \frac{\eta}{\|\nabla f(x)\| + \beta} \right]^2 = 0$$

Then, we have

By substitution

$$h - \frac{L(x)h^2}{2} = \left[ \frac{\eta}{\|\nabla f(x)\| + \beta} \right] - \frac{L(x)}{2} \left[ \frac{\eta}{\|\nabla f(x)\| + \beta} \right]^2 = 0$$

Then, we have that

$$L(x) = \frac{2(\|\nabla f(x)\| + \beta)}{\eta} = L'_1 \|\nabla f(x)\| + L'_0$$

Then, we have

By substitution

$$h - \frac{L(x)h^2}{2} = \left[ \frac{\eta}{\|\nabla f(x)\| + \beta} \right] - \frac{L(x)}{2} \left[ \frac{\eta}{\|\nabla f(x)\| + \beta} \right]^2 = 0$$

Then, we have that

$$L(x) = \frac{2(\|\nabla f(x)\| + \beta)}{\eta} = L'_1 \|\nabla f(x)\| + L'_0$$

For this we have the definition

- $(L_0, L_1)$ -smoothness

Then, we have

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

# The final Theorem

## Theorem (Clipped Gradient Descent) [2]

- 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

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

# Therefore

## Gradient Clipping

- Accelerate the convergence of Network

# Therefore

## Gradient Clipping

- Accelerate the convergence of Network

## However, you still have a Question

- Is this optimal the best Optimal?

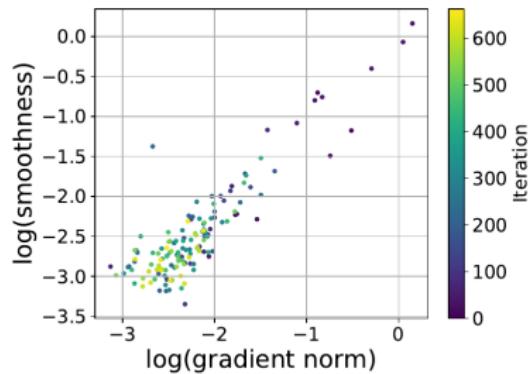
# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

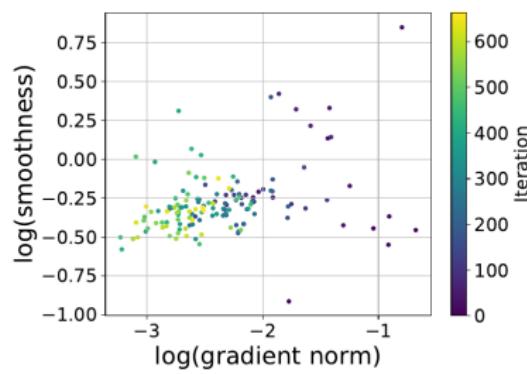
- Gaussian Noise on Hidden Units for Regularization
- Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
- Random Dropout Probability
- Projecting Noise into Input Space
- Augmenting by Noise
- Co-adaptation/Overfitting
- Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
- For More in Normalization

# Although, we can see an empirical relation

From Zhang et al. [2] (LSTM-based language)



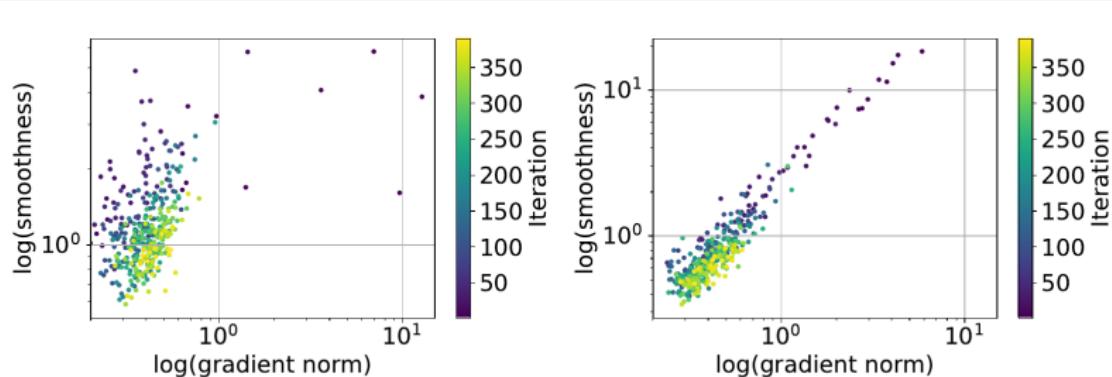
(a) Learning rate 30, with clipping.



(b) Learning rate 2, without clipping.

## Another Example

Here is another example (ResNet20 Training)



(b) Learning rate 1, without clipping. (c) Learning rate 5, with clipping.

## Remarks

### The paper

- It points out to a high correlation between the Jacobian and the Hessian when clipping is applied

# Remarks

## The paper

- It points out to a high correlation between the Jacobian and the Hessian when clipping is applied

## There are more work to be done

- The proof about these are not complete...

# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
  - Application into a Decoder/Encoder
  - Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
  - Random Dropout Probability
  - Projecting Noise into Input Space
  - Augmenting by Noise
  - Co-adaptation/Overfitting
  - Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
  - For More in Normalization

### For many years

- Dataset augmentation has been a standard regularization technique used to reduce overfitting while training supervised learning models

## DeVris and Taylor [6]

For many years

- Dataset augmentation has been a standard regularization technique used to reduce overfitting while training supervised learning models

For Example, LeCun et al. [3] when training the LeNet5

- They applied a series of transformations to the input images in order to improve the robustness of the model.

## DeVris and Taylor [6]

### For many years

- Dataset augmentation has been a standard regularization technique used to reduce overfitting while training supervised learning models

### For Example, LeCun et al. [3] when training the LeNet5

- They applied a series of transformations to the input images in order to improve the robustness of the model.

### Unfortunately

- Dataset augmentation is not as straightforward to apply in all domains as it is for images.
  - ▶ After all you have what is known as Out-Of-Distribution [4, 5]

## For Example

In voice detection, adding

- ① Gaussian noise to the input,

## For Example

In voice detection, adding

- ① Gaussian noise to the input,
- ② Shifting the pitch of the audio signal,

## For Example

In voice detection, adding

- ① Gaussian noise to the input,
- ② Shifting the pitch of the audio signal,
- ③ Time stretching,

## For Example

In voice detection, adding

- ① Gaussian noise to the input,
- ② Shifting the pitch of the audio signal,
- ③ Time stretching,
- ④ Varying the loudness of the audio signal,

## For Example

In voice detection, adding

- ① Gaussian noise to the input,
- ② Shifting the pitch of the audio signal,
- ③ Time stretching,
- ④ Varying the loudness of the audio signal,
- ⑤ Applying random frequency filters,

## For Example

In voice detection, adding

- ① Gaussian noise to the input,
- ② Shifting the pitch of the audio signal,
- ③ Time stretching,
- ④ Varying the loudness of the audio signal,
- ⑤ Applying random frequency filters,
- ⑥ Interpolating between samples in input space.

## For Example

In voice detection, adding

- ① Gaussian noise to the input,
- ② Shifting the pitch of the audio signal,
- ③ Time stretching,
- ④ Varying the loudness of the audio signal,
- ⑤ Applying random frequency filters,
- ⑥ Interpolating between samples in input space.

Actually, only the following techniques worked out

- Pitch shifting and random frequency filtering

## But Again

### Data Augmentation is a complex business

- We can see it in papers as [7, 8, 9]
  - ▶ Shorten, Connor, and Taghi M. Khoshgoftaar. "A survey on image data augmentation for deep learning." Journal of big data 6.1 (2019): 1-48.

# But Again

Data Augmentation is a complex business

- We can see it in papers as [7, 8, 9]
  - ▶ Shorten, Connor, and Taghi M. Khoshgoftaar. "A survey on image data augmentation for deep learning." Journal of big data 6.1 (2019): 1-48.

Thus, why not introduce the noise into the Deep Learning

Instead of  $\mathbf{x} + \epsilon$  with  $\epsilon \sim p_{Noise}(\mathbf{x}|\Theta)$



$$f(\mathbf{x}) = h \circ r + \epsilon \circ g(\mathbf{x}) = h(r(g(\mathbf{x})) + \epsilon)$$

They did something different

- First learning a data representation
- Then applying transformations to samples mapped to that representation.

## DeVris and Taylor [6]

They did something different

- First learning a data representation
- Then applying transformations to samples mapped to that representation.

They hypothesized

- It is better to apply transformations/noise at encoded/latent space

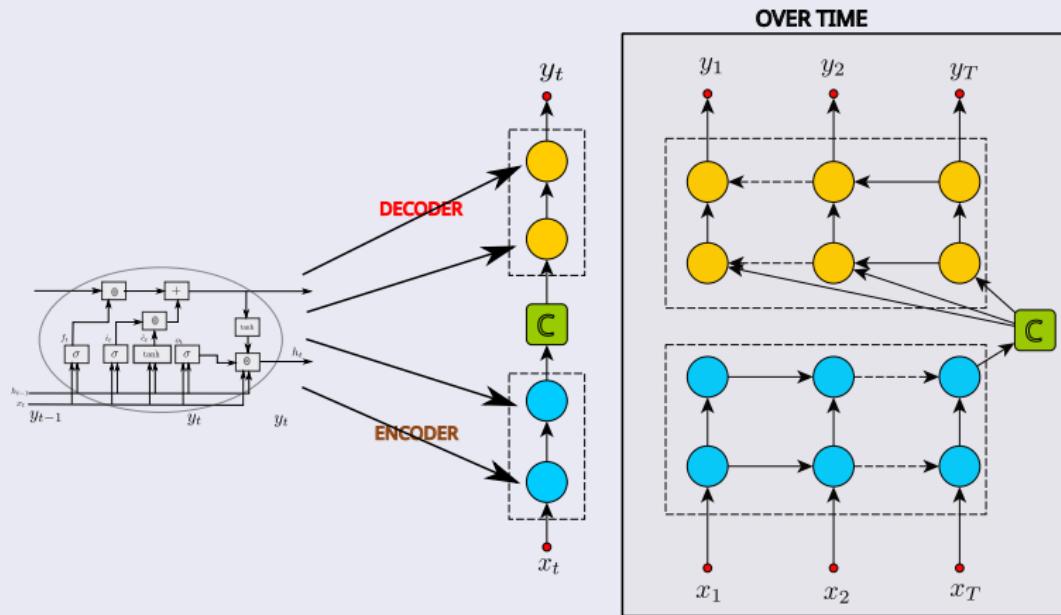
# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
  - Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
  - Random Dropout Probability
  - Projecting Noise into Input Space
  - Augmenting by Noise
  - Co-adaptation/Overfitting
  - Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
  - For More in Normalization

# Decoder/Encoder Part

We have a Decoder and Encoder Architecture



## Basically

They used a context  $C$  to pass information between the encoder and decoder

- Here is where the authors performed the augmentation

## Basically

They used a context  $C$  to pass information between the encoder and decoder

- Here is where the authors performed the augmentation

## Basically

- At the context, something like the embeddings at document level.

# Here

We have a K-coding symbol set

- The Encoder and Decoder are based in a novel hidden unit.

# Here

We have a K-coding symbol set

- The Encoder and Decoder are based in a novel hidden unit.

We have the following configuration per row element  $j$

$$r_j = \sigma \left( [\mathbf{W}_{rx}]_j + [\mathbf{U}_r \mathbf{h}_{t-1}]_j \right) \leftarrow \text{Reset Gate}$$

- $\sigma$  a sigmoid function

## Here

We have a K-coding symbol set

- The Encoder and Decoder are based in a novel hidden unit.

We have the following configuration per row element  $j$

$$r_j = \sigma \left( [\mathbf{W}_{r\mathbf{x}}]_j + [\mathbf{U}_{r\mathbf{h}_{t-1}}]_j \right) \leftarrow \text{Reset Gate}$$

- $\sigma$  a sigmoid function

The Update gate

$$z_j = \sigma \left( [\mathbf{W}_{z\mathbf{x}}]_j + [\mathbf{U}_{z\mathbf{h}_{t-1}}]_j \right)$$

# Where

## The Activation Gate update

$$h_j^t = z_j h_j^{t-1} + (1 - z_j) \tilde{h}_j^t$$

- Where  $\tilde{h}_j^t = \phi \left( [\mathbf{Wx}]_j + [\mathbf{U}(\mathbf{r} \odot \mathbf{h}_{t-1})]_j \right)$

# Where

## The Activation Gate update

$$h_j^t = z_j h_j^{t-1} + (1 - z_j) \tilde{h}_j^t$$

- Where  $\tilde{h}_j^t = \phi \left( [\mathbf{Wx}]_j + [\mathbf{U}(\mathbf{r} \odot \mathbf{h}_{t-1})]_j \right)$

## In this formulation

- When the reset gate is close to 0, the hidden state is forced to ignore the previous hidden state!!!

## Finally, at output

We have a probability of producing a symbol of a set of at the Decoder

$$p(y_t | y_{t-1}, \dots, y_1, \mathbf{c}) = \frac{\exp(W_o \mathbf{h}_t + U_o y_{t-1} + \mathbf{c}_{t-1})}{\sum_{j=1}^K \exp(W_j \mathbf{h}_t + U_o y_{t-1} + \mathbf{c}_{t-1})}$$

## Finally, at output

We have a probability of producing a symbol of a set of at the Decoder

$$p(y_t|y_{t-1}, \dots, y_1, \mathbf{c}) = \frac{\exp(W_o \mathbf{h}_t + U_o y_{t-1} + \mathbf{c}_{t-1})}{\sum_{j=1}^K \exp(W_j \mathbf{h}_t + U_o y_{t-1} + \mathbf{c}_{t-1})}$$

## Then, at the Encoder

- The encoder learns to predict the next symbol  $x_t$  based in the previous  $x_{t-1}, x_{t-2}, \dots, x_1$  by using the maximization

$$\max_{\theta} \frac{1}{N} \sum_{n=1}^N p(\mathbf{y}_n | \mathbf{x}_n)$$

## Here, the Noise

Generate noise by drawing from

- A Gaussian distribution with **zero mean** and **per-element standard deviation** calculated across all context vectors in the dataset

$$c'_i = c_i + \gamma X, \quad X \sim N(0, \sigma_i^2)$$

## Here, the Noise

Generate noise by drawing from

- A Gaussian distribution with **zero mean** and **per-element standard deviation** calculated across all context vectors in the dataset

$$c'_i = c_i + \gamma X, \quad X \sim N(0, \sigma_i^2)$$

We can generate this using a more direct approach

- For each sample in the dataset, we find its  $K$  nearest neighbors in feature space, then

$$\mathbf{c}' = (\mathbf{c}_k - \mathbf{c}_j) \lambda + \mathbf{c}_j$$

- $\lambda = 0.5$

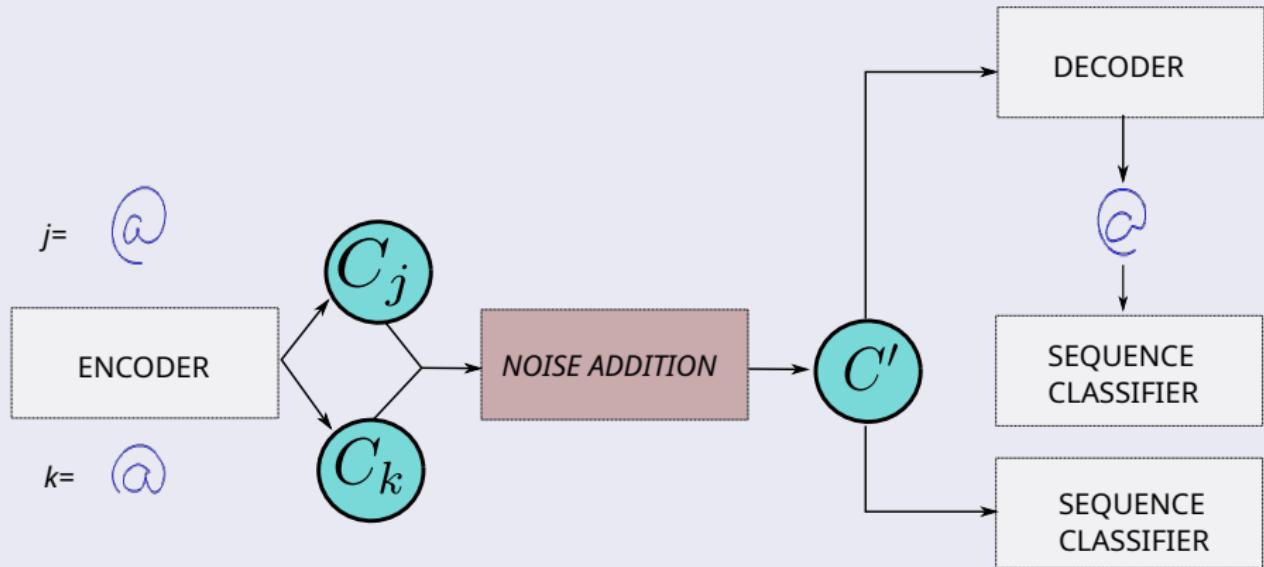
## Then

Once this new augmented context vectors with noise are ready

- As input for a learning task,
- They can be decoded to generate new sequences

Finally, we have

The following architecture where two symbols  $j$  and  $k$  are encoded



# Results

Not so much improvement when comparing with simple augmentation

Image Size	Description	Test Error	Test Error (Reconstructions of original data)
32 × 32	Original dataset	$8.59 \pm 0.24$	-
24 × 24	Center crop	$11.28 \pm 0.25$	$18.54 \pm 0.38$
24 × 24	Center crop + extrapolation	$13.90 \pm 0.22$	$17.69 \pm 0.39$
24 × 24	Simple data augmentation	<b><math>7.33 \pm 0.17</math></b>	$13.60 \pm 0.17$
24 × 24	Simple data augmentation + extrapolation	$8.80 \pm 0.24$	<b><math>12.00 \pm 0.23</math></b>

## Why is this happening?

It is the same problem at the exit point

- We are regularizing at the encoded input space... but the architecture is still there...

# Why is this happening?

It is the same problem at the exit point

- We are regularizing at the encoded input space... but the architecture is still there...

Therefore

- It is necessary to do something quite different...

# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
- Application into a Decoder/Encoder
- Dropout as Regularization**
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
  - Random Dropout Probability
  - Projecting Noise into Input Space
  - Augmenting by Noise
  - Co-adaptation/Overfitting
  - Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
  - For More in Normalization

# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
- Application into a Decoder/Encoder
- 4 Dropout as Regularization**
  - Introduction**
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
  - Random Dropout Probability
  - Projecting Noise into Input Space
  - Augmenting by Noise
  - Co-adaptation/Overfitting
  - Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
  - For More in Normalization

# Regularization in Deep Forward

## In Layers of a Deep Forward

- We want to find an estimation  $\hat{x}_t^r$  to an input at  $x_0 \in \mathbb{R}^d$  in layer  $t$  satisfying

# Regularization in Deep Forward

## In Layers of a Deep Forward

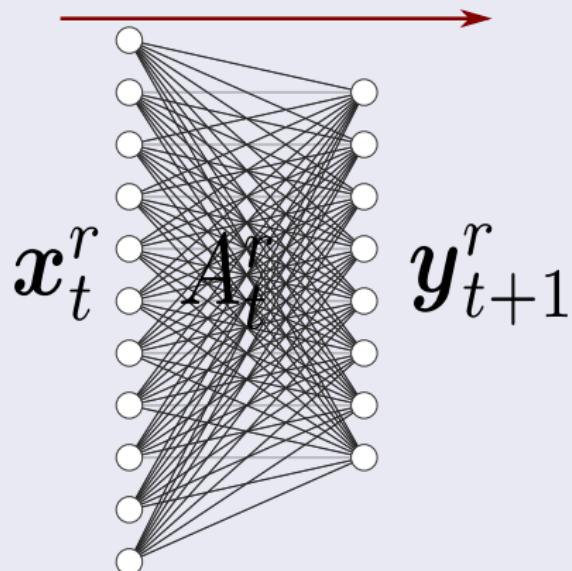
- We want to find an estimation  $\mathbf{x}_t^r$  to an input at  $\mathbf{x}_0 \in \mathbb{R}^d$  in layer  $t$  satisfying

$$\sigma(A_t^r \mathbf{x}_t) = \mathbf{y}_{t+1}$$

We can see this

## A flow of information

FORWARD FLOW OF INFORMATION



## In all such situations

The vector  $x_t$  is generated by  $y_{t+1}$  using back-propagation

$$A_t^r = A_t^{r-1} - \eta \frac{\partial L(A_T^{r-1}, \dots, A_0^{r-1}, x_0)}{\partial A_t^{r-1}}$$

## In all such situations

The vector  $x_t$  is generated by  $y_{t+1}$  using back-propagation

$$A_t^r = A_t^{r-1} - \eta \frac{\partial L(A_T^{r-1}, \dots, A_0^{r-1}, x_0)}{\partial A_t^{r-1}}$$

It is usually a meaningless bad approximation

- to  $x^*$  optimal at layer  $t$  for all possible inputs  $x'_0 s.$

## Then

We can see the Deep Forward Network as

$$y_T = \sigma(A_T \sigma(A_{T-1} \sigma(A_{T-2} (\dots \sigma(A_0 x_0)))))$$

# Then

We can see the Deep Forward Network as

$$y_T = \sigma(A_T \sigma(A_{T-1} \sigma(A_{T-2} (\dots \sigma(A_0 x_0)))))$$

## Here

- The  $\sigma$  is applied to the generated vectors point wise...

# The Jacobian of the Gradient Descent

Here, we assume a Least Squared Error cost function

$$\frac{\partial L \left( A_T^{r-1}, \dots, A_0^{r-1}, x_0^i \right)}{\partial A_t^{r-1}} = - \left( z^i - y_T \right) \times \sigma' \left( A_{T-1}^r \mathbf{x}_{T-1} \right) \times \frac{\partial A_{T-1}^r \mathbf{x}_{T-1}}{\partial \mathbf{x}_{T-1}} \times \dots \times \sigma' \left( A_t^r \mathbf{x}_t \right) \times \frac{\partial A_t^r \mathbf{x}_t}{\partial \mathbf{x}_t}$$

# The Jacobian of the Gradient Descent

Here, we assume a Least Squared Error cost function

$$\frac{\partial L \left( A_T^{r-1}, \dots, A_0^{r-1}, x_0^i \right)}{\partial A_t^{r-1}} = - \left( z^i - y_T \right) \times \sigma' \left( A_{T-1}^r \mathbf{x}_{T-1} \right) \times \frac{\partial A_{T-1}^r \mathbf{x}_{T-1}}{\partial \mathbf{x}_{T-1}} \times \dots \times \sigma' \left( A_t^r \mathbf{x}_t \right) \times \frac{\partial A_t^r \mathbf{x}_t}{\partial \mathbf{x}_t}$$

Where

$$\sigma' \left( A_k^r \mathbf{x}_k \right) = \begin{pmatrix} \sigma' \left( \mathbf{a}_{1k}^r \mathbf{x}_k \right) & 0 & \cdots & 0 \\ 0 & \sigma' \left( \mathbf{a}_{2k}^r \mathbf{x}_k \right) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma' \left( \mathbf{a}_{Mk}^r \mathbf{x}_k \right) \end{pmatrix}$$

## What will happen in the following situation?

Imagine that  $A'_k$ s are diagonal matrix

$$A_k^r = \begin{pmatrix} a_{1k} & 0 & \cdots & 0 \\ 0 & a_{2k} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{Mk} \end{pmatrix}$$

# What will happen in the following situation?

Imagine that  $A'_k$ s are diagonal matrix

$$A_k^r = \begin{pmatrix} a_{1k} & 0 & \cdots & 0 \\ 0 & a_{2k} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{Mk} \end{pmatrix}$$

Therefore, we have

$$\sigma' (A_k^r \mathbf{x}_k) = \begin{pmatrix} \sigma' (a_{1k}^r x_{1k}) & 0 & \cdots & 0 \\ 0 & \sigma' (a_{2k}^r x_{2k}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma' (a_{Mk}^r x_{2k}) \end{pmatrix}$$

Then, we have that

First

$$\sigma' (A_{T-1}^r \mathbf{x}_{T-1}) \times \frac{\partial A_{T-1}^r \mathbf{x}_{T-1}}{\partial \mathbf{x}_{T-1}} \times \dots \times \sigma' (A_t^r \mathbf{x}_t) \times \frac{\partial A_t^r \mathbf{x}_t}{\partial \mathbf{x}_t} = *$$

Then, we have that

First

$$\sigma' \left( A_{T-1}^r \mathbf{x}_{T-1} \right) \times \frac{\partial A_{T-1}^r \mathbf{x}_{T-1}}{\partial \mathbf{x}_{T-1}} \times \dots \times \sigma' \left( A_t^r \mathbf{x}_t \right) \times \frac{\partial A_t^r \mathbf{x}_t}{\partial \mathbf{x}_t} = *$$

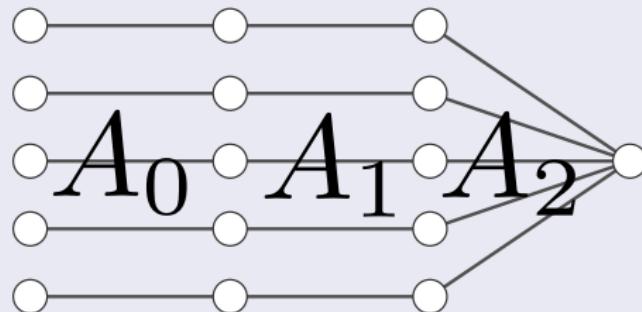
Then, we have that

$$* = \begin{pmatrix} \prod_{k=T-1}^t \sigma' \left( a_{1k}^r x_{1k} \right) a_{1k} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \prod_{k=T-1}^t \sigma' \left( a_{Mk}^r x_{2k} \right) a_{2k} \end{pmatrix}$$

# Actually

## Choosing Matrices in such way

- It is like a heavy simplification of the Deep Forward Network



# Something happens with the LASSO and Ridge

## At the top of the Optimization Cost Function

- We do not know how such shallow regularization can affect the Neural Network

## So heavy regularization

- It can not be a so good idea...

# Something happens with the LASSO and Ridge

## At the top of the Optimization Cost Function

- We do not know how such shallow regularization can affect the Neural Network

## So heavy regularization

- It can not be a so good idea...

## We need a new way of doing stuff

- For example, we could do the following...

# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
- Application into a Decoder/Encoder
- **Dropout as Regularization**
  - Introduction
  - **Dropout Process**
    - Dropout as Bagging/Bootstrap Aggregation
    - Beyond an Empirical Probabilities, LASSO and Data Flow
  - Random Dropout Probability
  - Projecting Noise into Input Space
  - Augmenting by Noise
  - Co-adaptation/Overfitting
  - Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
  - For More in Normalization

# Dropout

It was introduced by Hinton and Google [10]

- To avoid the problem of over-fitting

# Dropout

It was introduced by Hinton and Google [10]

- To avoid the problem of over-fitting

You can see it as a regularization

- From [11] “Dropout training as adaptive regularization” by Wager et al.

He comments that with unlimited computations

- “the best way to “regularize” a fixed-sized model is to average the predictions of all possible settings of the parameters”

He comments that with unlimited computations

- “the best way to “regularize” a fixed-sized model is to average the predictions of all possible settings of the parameters”

Something like Boosting [12]

- By Using simpler and smaller models

# Problem

We have Deep Architectures with thousands of parameters and hyperparameters

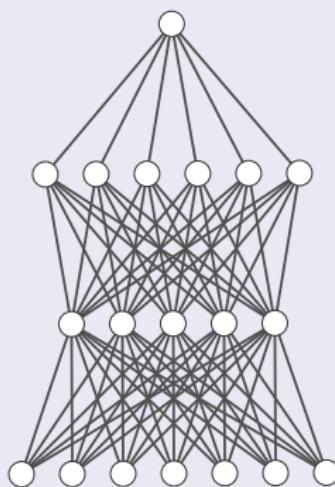
- Therefore, we have a problem!!! We need to solve this in some way!!!

# Problem

We have Deep Architectures with thousands of parameters and hyperparameters

- Therefore, we have a problem!!! We need to solve this in some way!!!

What if we fix our architecture



# How it works?

You have forward layers

$$z_i^{l+1} = W_i^{l+1} \mathbf{x}^l + b_i^{l+1}$$

$$x_i^{l+1} = \sigma(z_i^{l+1})$$

# How it works?

You have forward layers

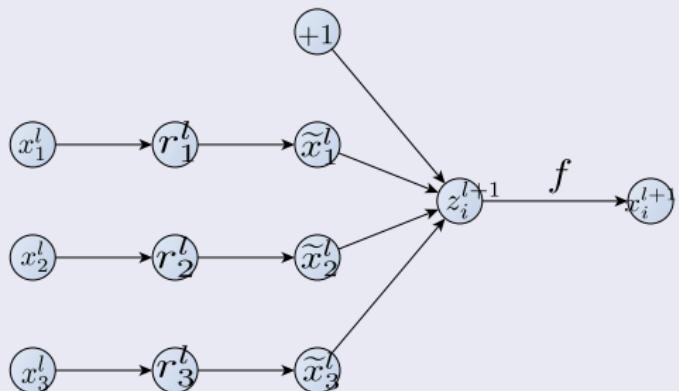
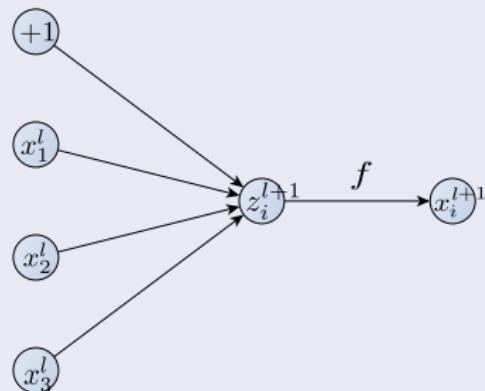
$$\begin{aligned}z_i^{l+1} &= W_i^{l+1} \mathbf{x}^l + b_i^{l+1} \\x_i^{l+1} &= \sigma(z_i^{l+1})\end{aligned}$$

With dropout, the feed-forward operation becomes

$$\begin{aligned}r_j^l &\sim \text{Bernoulli}(p) \\ \tilde{\mathbf{x}}^l &= \mathbf{r}^l \odot \mathbf{x}^l \\ z_i^{l+1} &= W_i^{l+1} \tilde{\mathbf{x}}^l + b_i^{l+1} \\ x_i^{l+1} &= \sigma(z_i^{l+1})\end{aligned}$$

# The Network

It looks like a series of gates



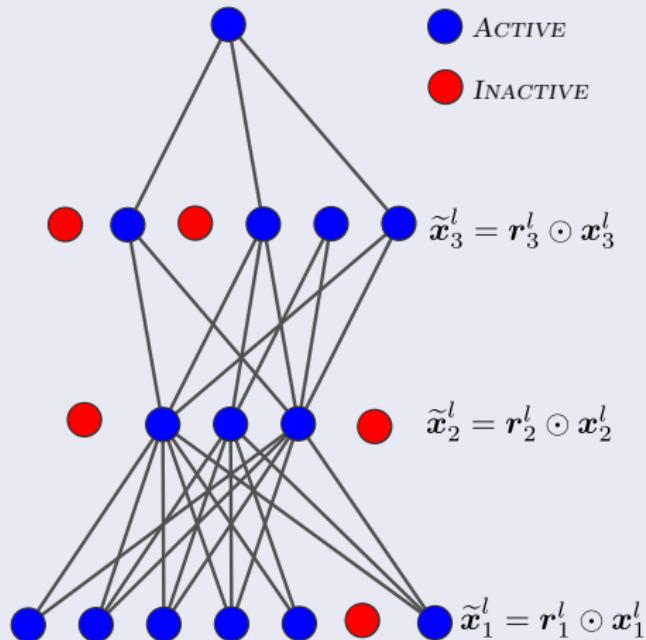
## Therefore

We have that sampling is done in a Bernoulli to generate the  $r^l$ , a vector of Bernoulli random variables

- Then, the layers are thinned by the wise multiplication with the nodes at each layer

Then, we erase randomly connections through the network

We generate sparser version with input layer such that  $p_{1j}^1 \rightarrow 1.0$



## Then assuming a Multilayer Perceptron

We have the following Architecture without bias to simplify with a single output

$$\min \frac{1}{N} \sum_{i=1}^N (z_i - t_i)^2$$

$$z_i = \sigma_1 (W_{oh} \mathbf{y}_i)$$

$$\mathbf{y}_i = \sigma_2 (W_{hi} \mathbf{x}_i)$$

## Then assuming a Multilayer Perceptron

We have the following Architecture without bias to simplify with a single output

$$\min \frac{1}{N} \sum_{i=1}^N (z_i - t_i)^2$$

$$z_i = \sigma_1 (W_{oh} \mathbf{y}_i)$$

$$\mathbf{y}_i = \sigma_2 (W_{hi} \mathbf{x}_i)$$

Then, we get the following network after the sampling

$$L (W_{oh}, W_{hi}) = (t - z)^2$$

$$z = \sigma_1 \left( W_{oh} \left( \mathbf{r}^2 \odot \mathbf{y} \right) \right)$$

$$\mathbf{y} = \sigma_2 \left( W_{hi} \left( \mathbf{r}^1 \odot \mathbf{x} \right) \right)$$

Then, we have that

## The Backpropagation at hidden weights

$$\frac{\partial L}{\partial W_{oh}} = -2(t - z) \times \frac{\partial \sigma'_1(\text{net}_{oh})}{\partial \text{net}_{oh}} \times (\mathbf{r}^2 \odot \mathbf{y})$$

Then, we have that

## The Backpropagation at hidden weights

$$\frac{\partial L}{\partial W_{oh}} = -2(t - z) \times \frac{\partial \sigma'_1(\text{net}_{oh})}{\partial \text{net}_{oh}} \times (\mathbf{r}^2 \odot \mathbf{y})$$

Basically

$$(W_{oh}^{t+1})_j = \begin{cases} (W_{oh}^t)_j + \eta 2(t - z) \times \frac{\partial \sigma'_1(\text{net}_{oh})}{\partial \text{net}_{oh}} (\mathbf{y})_j & \text{if } r_j = 1 \\ (W_{oh}^t)_j & \text{if } r_j = 0 \end{cases}$$

## However, At Testing

There are a exponential number of possible sparse networks

- A neural net with  $n$  units, can be seen as a collection of  $2^n$  possible thinned neural networks.

## However, At Testing

There are a exponential number of possible sparse networks

- A neural net with  $n$  units, can be seen as a collection of  $2^n$  possible thinned neural networks.

## Assuming

- These networks all share weights so that the total number of parameters is still  $O(n^2)$  given that you this many connections

$$\frac{n(n-1)}{2} = O(n^2)$$

## However, At Testing

There are a exponential number of possible sparse networks

- A neural net with  $n$  units, can be seen as a collection of  $2^n$  possible thinned neural networks.

## Assuming

- These networks all share weights so that the total number of parameters is still  $O(n^2)$  given that you this many connections

$$\frac{n(n-1)}{2} = O(n^2)$$

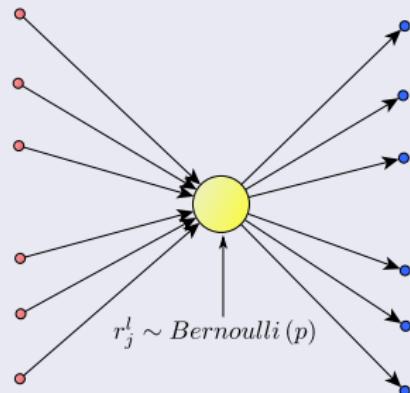
Problem, we cannot average such amount of sub-networks

- We average over the different passes to obtain a  $p$  for each node in the network
  - ▶ Meaning the probability of being active in the network.

$$p_{ik} = \frac{\text{\#of subnets where node } ik \text{ was active}}{\text{\#Of total subnets}}$$

Then, we have

At Training



# The mixture of the models

We know that

$$E(w_{ik}) = \sum_{m=1}^M w_{ik}^m p(w_{ik}^m | \text{BackProp}_M, \mathbf{X})$$

# The mixture of the models

We know that

$$E(w_{ik}) = \sum_{m=1}^M w_{ik}^m p(w_{ik}^m | \text{BackProp}_M, \mathbf{X})$$

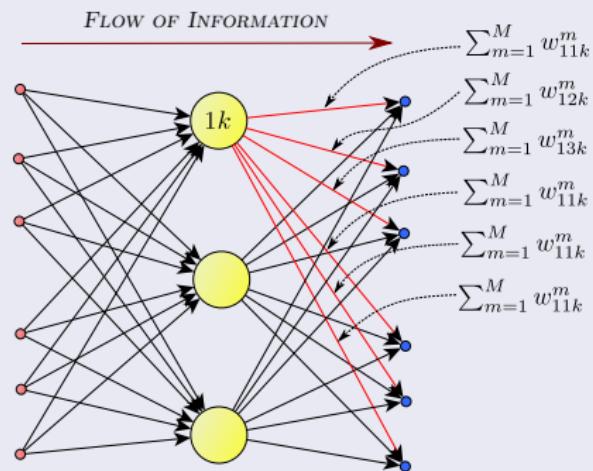
Clearly, we need to get  $p(w_{ik}^m | \text{BackProp}_M, \mathbf{X})$

- A simple solution, we can use

$$p_{ik} = \frac{\text{\#of subnets where node } ik \text{ was active}}{\text{\#Of total subnets}}$$

# Therefore, Using the fact that Forward has a Flow of Information

Add flow of information between all the different generated trained networks



## Mathematically

We have the following ideas

- Each node has associated matrices for exit weights

$$W_{out} = \begin{pmatrix} \frac{1}{m} \sum_{i=1}^m w_{i1k}^m & \approx E[w_{i1k}] \\ \frac{1}{m} \sum_{i=1}^m w_{i2k}^m & \approx E[w_{i2k}] \\ \vdots \\ \frac{1}{m} \sum_{i=1}^m w_{iJk}^m & \approx E[w_{iJk}] \end{pmatrix}$$

## Mathematically

We have the following ideas

- Each node has associated matrices for exit weights

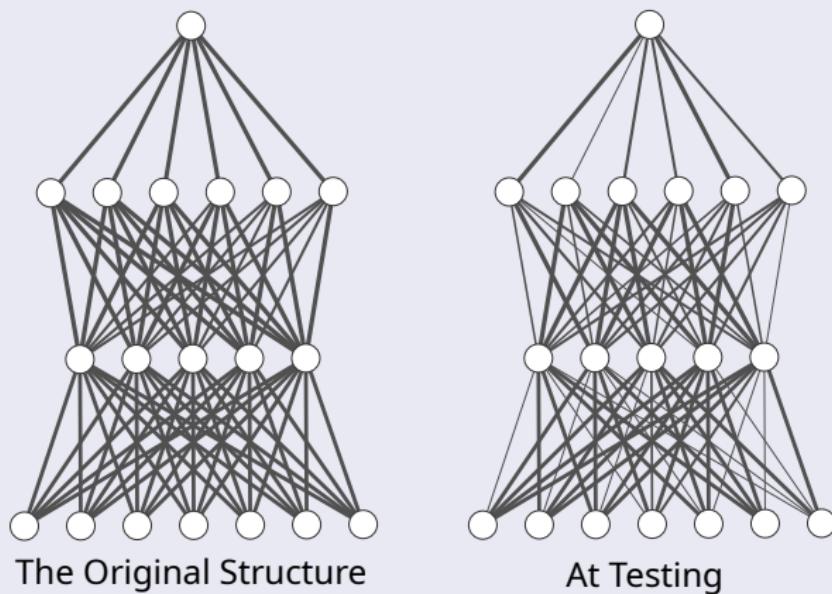
$$W_{out} = \begin{pmatrix} \frac{1}{m} \sum_{i=1}^m w_{i1k}^m & \approx E[w_{i1k}] \\ \frac{1}{m} \sum_{i=1}^m w_{i2k}^m & \approx E[w_{i2k}] \\ \vdots \\ \frac{1}{m} \sum_{i=1}^m w_{iJk}^m & \approx E[w_{iJk}] \end{pmatrix}$$

Then use the probability  $p$  to get the new final weights

$$p_{ik} W_{out} = \begin{pmatrix} E[w_{i1k}] p_{ik} \\ E[w_{i2k}] p_{ik} \\ \vdots \\ E[w_{iJk}] p_{ik} \end{pmatrix}$$

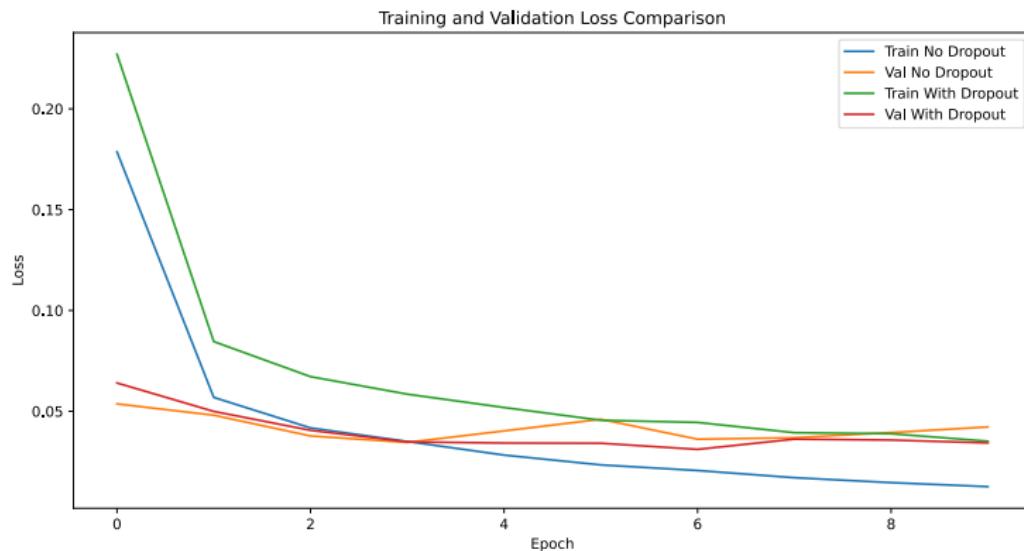
Then

We have the following structure where thinner lines represent smaller weights



# Example

## CNN with Dropout



# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
- Application into a Decoder/Encoder
- Dropout as Regularization**
  - Introduction
  - Dropout Process
- Dropout as Bagging/Bootstrap Aggregation**
  - Beyond an Empirical Probabilities, LASSO and Data Flow
- Random Dropout Probability
- Projecting Noise into Input Space
- Augmenting by Noise
- Co-adaptation/Overfitting
- Batch normalization
- Improving the Google Layer Normalization
- Layer Normalization in RNN
- Invariance Under Weights and Data Transformations
- For More in Normalization

# Why dropout?

Srivastava et al. [10]

- A motivation for dropout comes from the theory of evolution!!!
  - ▶ Yes a original network and after a mutated one!!!

# Why dropout?

Srivastava et al. [10]

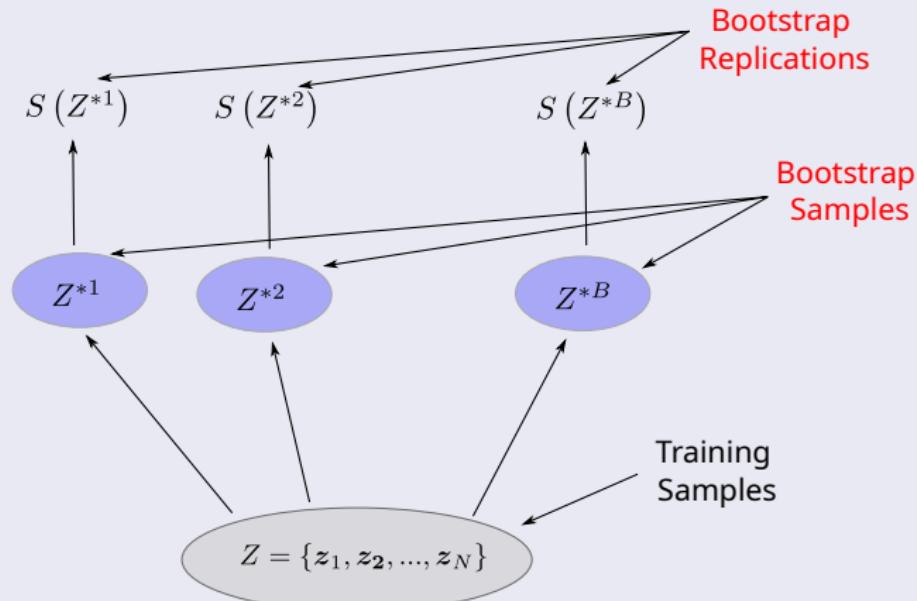
- A motivation for dropout comes from the theory of evolution!!!
  - ▶ Yes a original network and after a mutated one!!!

The most accepted interpretation of dropout

- It is implicitly bagging at test time a large number of neural networks which share parameters.

# Bagging/Bootstrap Aggregation

## Schematic of the Bootstrap Aggregation process [12]



# Thus

Use each of them to train a copy  $y_b (\boldsymbol{x})$  of a predictive regression model to predict a single continuous variable

$$y_{com} (\boldsymbol{x}) = \frac{1}{B} \sum_{b=1}^B y_b (\boldsymbol{x})$$

# Results

We have that

Method	CIFAR-10 Error	CIFAR-100 Error
CNN+max pooling (hand tuned)	15.60%	43.48%
CNN+stochastic pooling (Zeiler and Fergus, 2013)	15.13%	42.51%
CNN+max pooling (Snoek et al., 2012)	14.98%	-
CNN+max pooling + dropout fully connected layers	14.32%	41.26%
CNN+max pooling + dropout in all layers	12.61%	37.20%
CNN+maxout (Goodfellow et al., 2013)	11.68%	38.57%

# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
- Application into a Decoder/Encoder
- Dropout as Regularization**
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
- Beyond an Empirical Probabilities, LASSO and Data Flow**
  - Random Dropout Probability
  - Projecting Noise into Input Space
  - Augmenting by Noise
  - Co-adaptation/Overfitting
  - Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
  - For More in Normalization

## Given the previous ideas

### Why not to use the Data Flow for Sparsity?

- Basically, we can assume that a pattern exist in the data you are looking at
  - ▶ The shifts on the weights are not so great...

## Given the previous ideas

### Why not to use the Data Flow for Sparsity?

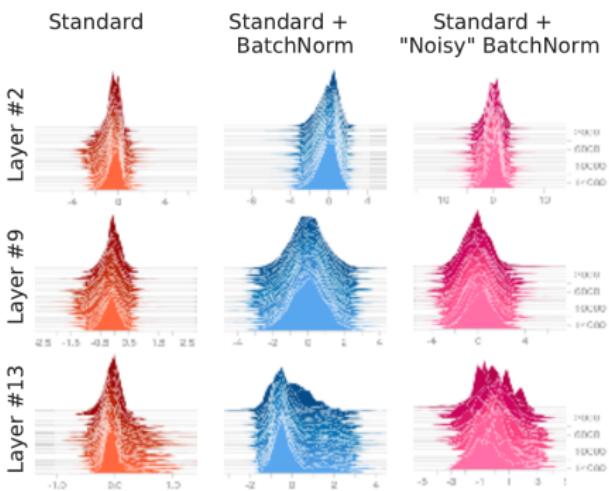
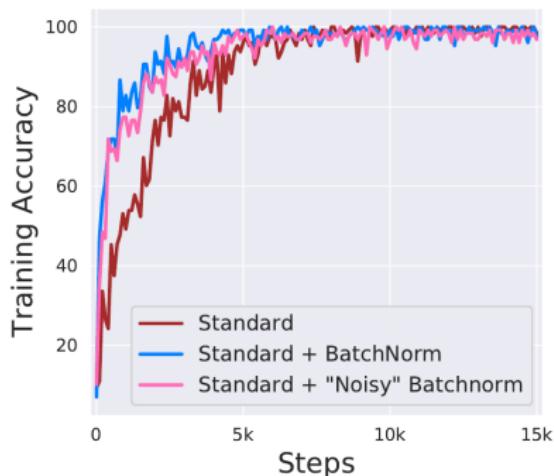
- Basically, we can assume that a pattern exist in the data you are looking at
  - ▶ The shifts on the weights are not so great...

$p_{ik}$  is too broad because it does not represent the real  
 $p(w_{ik}^m | \text{BackProp}_M, \mathbf{X})$

- Actually, you should use the min-batch values,  $\mathbf{x}_t$  and  $\mathbf{y}_{t+1}$ , to generate the real distribution

## Based in the paper

"How does batch normalization help optimization?", in Advances in Neural Information Processing Systems (2018), pp. 2483--2493.



Then, we can use a Gaussian Distribution to model this

Actually, the paper is telling us that, given the noise that is injected at each time step  $t$

$$\mu^t \sim U(-n_\mu, n_\mu)$$

$$\sigma^t \sim U(1, n)$$

## Why not use for the Data for enforcing Sparsity?

We have

$$p(\mathbf{y}^{l+1} | \mathbf{x}^l, W) = \mathcal{N}\left(\sigma\left(W\mathbf{x}^l\right), \sigma^2 I\right)$$

$$p(\sigma^2) \propto "constant"$$

$$p(W^l | \tau) = \prod_{i=1}^d \mathcal{N}\left(w_j^l | 0, \tau_j^l\right) = \mathcal{N}\left(W^l | 0, (\Upsilon(\tau))^{-1}\right)$$

$$p(\tau | \gamma) = \left(\frac{\gamma}{2}\right)^d \prod_{i=1}^d \exp\left\{-\frac{\gamma}{2}\tau_i\right\}$$

- With  $\Upsilon(\tau) = \text{diag}(\tau_1^{-1}, \dots, \tau_d^{-1})$  is the diagonal matrix with the inverse variances of all the  $w_i$ 's.

## How do we build such distribution

Given that each  $w_i$  has a zero-mean Gaussian prior

$$p(w_i|\tau_i) = \mathcal{N}(w_i|0, \tau_i) \quad (1)$$

## How do we build such distribution

Given that each  $w_i$  has a zero-mean Gaussian prior

$$p(w_i|\tau_i) = \mathcal{N}(w_i|0, \tau_i) \quad (1)$$

Where  $\tau_i$  has the following exponential hyper-prior

$$p(\tau_i|\gamma) = \frac{\gamma}{2} \exp\left\{-\frac{\gamma}{2}\tau_i\right\} \text{ for } \tau_i \geq 0 \quad (2)$$

## How do we build such distribution

Given that each  $w_i$  has a zero-mean Gaussian prior

$$p(w_i|\tau_i) = \mathcal{N}(w_i|0, \tau_i) \quad (1)$$

Where  $\tau_i$  has the following exponential hyper-prior

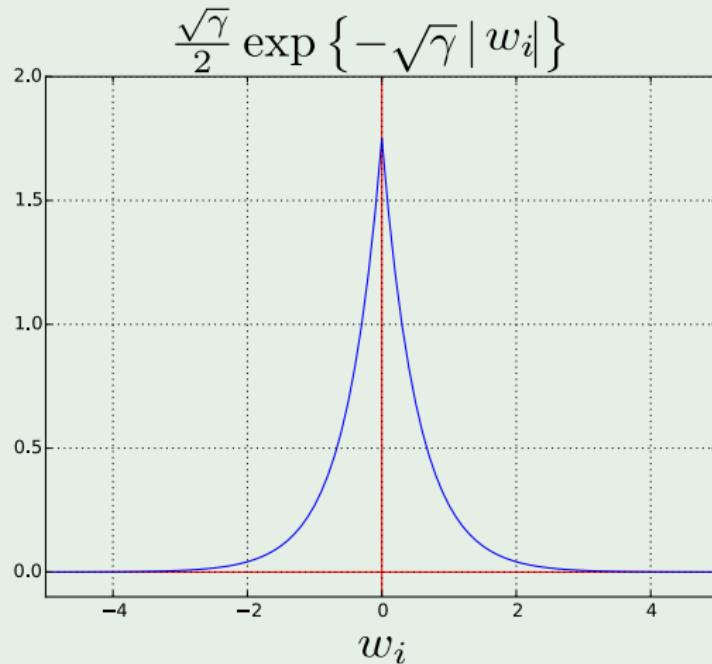
$$p(\tau_i|\gamma) = \frac{\gamma}{2} \exp\left\{-\frac{\gamma}{2}\tau_i\right\} \text{ for } \tau_i \geq 0 \quad (2)$$

Then, we have

$$w_i \sim p(w_i|\gamma) = \int_0^{\infty} p(w_i|\tau_i) p(\tau_i|\gamma) d\tau_i = \frac{\sqrt{\gamma}}{2} \exp\{-\sqrt{\gamma}|w_i|\} \quad (3)$$

## Example

### The double exponential



## Then using the Monte Carlo Method

We have

$$E [W^t | f(W_b^{tl} \mathbf{x}_b), \sigma^2 I] = \frac{p(\sigma^2)}{B} \sum_{b=1}^B \mathcal{N}(f(W_b^{tl} \mathbf{x}_b), \sigma^2 I) p(W_b^{tl} | \tau_i) p(\tau_i | \gamma)$$

## Then using the Monte Carlo Method

We have

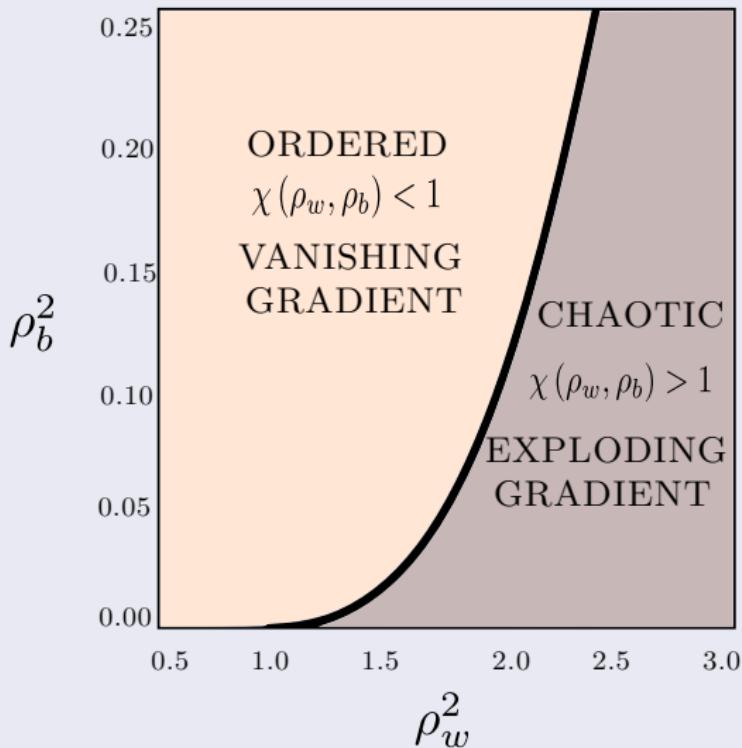
$$E [W^t | f(W_b^{tl} \mathbf{x}_b), \sigma^2 I] = \frac{p(\sigma^2)}{B} \sum_{b=1}^B \mathcal{N}(f(W_b^{tl} \mathbf{x}_b), \sigma^2 I) p(W_b^{tl} | \tau_i) p(\tau_i | \gamma)$$

Then, we use the mini batch per epoch to decide if we drop a weight

- Basically, the previous ideas

## We are using the following idea

Basically, we are using the fact that



Thus, we have that

The layer output can be bounded by

$$\mathcal{N} \left( f \left( W_b^{tl} \mathbf{x}_b \right), \sigma^2 I \right)$$

Thus, we have that

The layer output can be bounded by

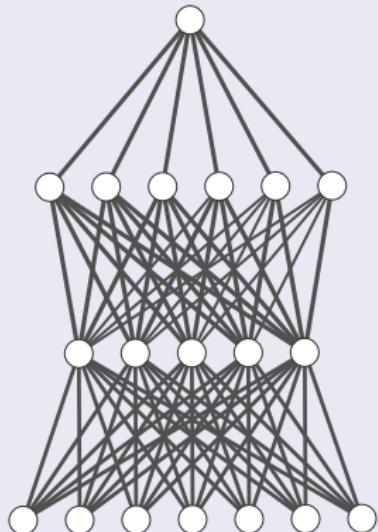
$$\mathcal{N} \left( f \left( W_b^{tl} \mathbf{x}_b \right), \sigma^2 I \right)$$

The other part of the equation is the sparsity part

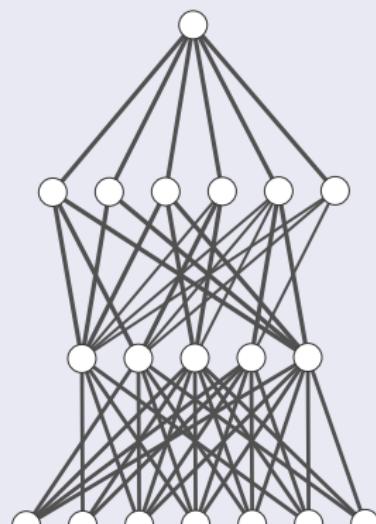
$$p \left( W_b^{tl} | \tau_i \right) p \left( \tau_i | \gamma \right)$$

## As the process progress

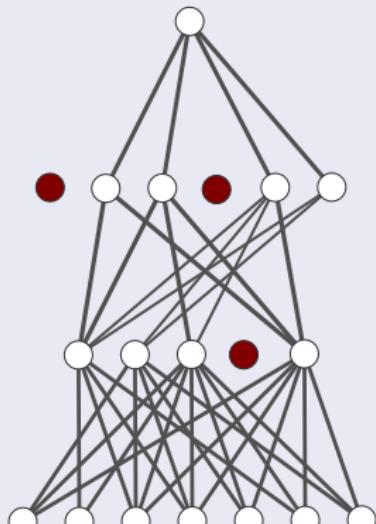
Once the weights fall below certain level we shutdown the weight



The Original Structure



After Some Epochs



More Epochs

# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
- Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
- Random Dropout Probability
  - Projecting Noise into Input Space
  - Augmenting by Noise
  - Co-adaptation/Overfitting
  - Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
  - For More in Normalization

The main goal when using dropout

- It is to regularize the neural network we are training

## The main goal when using dropout

- It is to regularize the neural network we are training

## Those random modifications of the network's structure

- They are believed to avoid co-adaptation of neurons by making it impossible for two subsequent neurons to rely solely on each other [10]

## Therefore

We have a function that projects from a dimensional space to another

$$h(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

## Therefore

We have a function that projects from a dimensional space to another

$$h(\mathbf{x}) = W\mathbf{x} + \mathbf{b}$$

Then, given an activation function and its noisy version where  
 $M \sim \mathcal{B}(p_h)$

- Where  $\text{rect}(h) = \max(0, h)$

$$\sigma(h) = \text{rect}(h) \tag{4}$$

$$\tilde{\sigma}(h) = M \odot \text{rect}(h) \tag{5}$$

## We have the following

We have that

- Eq. 5 denotes the activation with dropout during training
- Eq. 4 the equation of the activation at test time.

## We have the following

We have that

- Eq. 5 denotes the activation with dropout during training
- Eq. 4 the equation of the activation at test time.

Additionally, Srivastava et al. [10]

- He mentions to use an scaling of  $\sigma(h)$

$$p_{ijk} = \frac{\text{\#of subnets where node } ijk \text{ was active}}{\text{\#Of total subnets}}$$

# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
- Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
- Random Dropout Probability
- Projecting Noise into Input Space
  - Augmenting by Noise
  - Co-adaptation/Overfitting
- Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
  - For More in Normalization

# Data Augmentation

In many previous works [6, 3]

- It has been shown that augmenting data by using domain specific transformations helps in learning better models

# Data Augmentation

In many previous works [6, 3]

- It has been shown that augmenting data by using domain specific transformations helps in learning better models

Therefore, the main idea

- It is to map input data to output labels

# Data Augmentation

In many previous works [6, 3]

- It has been shown that augmenting data by using domain specific transformations helps in learning better models

Therefore, the main idea

- It is to map input data to output labels

One way to learn such a mapping function

- It is to augment the data using noise:
  - ▶ Hypothesis!!! Noise based regularization techniques seems to be increasing training data coverage as augmentation

# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
- Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
- **Random Dropout Probability**
  - Projecting Noise into Input Space
- **Augmenting by Noise**
  - Co-adaptation/Overfitting
- Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
  - For More in Normalization

## Augmenting by Noise [13]

We assume that for a given  $\tilde{\sigma}(h)$ , there is an optimal  $\mathbf{x}^*$

$$(\sigma \circ h)(\mathbf{x}^*) = \text{rect}(h(\mathbf{x}^*)) \approx M \odot \text{rect}(h) = (\tilde{\sigma} \circ h)(\mathbf{x}^*)$$

## Augmenting by Noise [13]

We assume that for a given  $\tilde{\sigma}(h)$ , there is an optimal  $\mathbf{x}^*$

$$(\sigma \circ h)(\mathbf{x}^*) = \text{rect}(h(\mathbf{x}^*)) \approx M \odot \text{rect}(h) = (\tilde{\sigma} \circ h)(\mathbf{x}^*)$$

This  $\mathbf{x}^*$  can be found by minimizing by stochastic gradient descent

$$L(\mathbf{x}, \mathbf{x}^*) = [(\sigma \circ h)(\mathbf{x}^*) - (\tilde{\sigma} \circ h)(\mathbf{x})]^2$$

## Extending to $n$ layers

For this, we define

$$\tilde{g}^{(i)}(\mathbf{x}) = \left[ \tilde{\sigma}^{(i)} \circ h^{(i)} \circ \dots \circ \tilde{\sigma}^{(1)} \circ h^{(1)} \right] (\mathbf{x})$$

$$g^{(i)}(\mathbf{x}^*) = \left[ \sigma^{(i)} \circ h^{(i)} \circ \dots \circ \sigma^{(1)} \circ h^{(1)} \right] (\mathbf{x}^*)$$

## Extending to $n$ layers

For this, we define

$$\tilde{g}^{(i)}(\mathbf{x}) = \left[ \tilde{\sigma}^{(i)} \circ h^{(i)} \circ \dots \circ \tilde{\sigma}^{(1)} \circ h^{(1)} \right](\mathbf{x})$$

$$g^{(i)}(\mathbf{x}^*) = \left[ \sigma^{(i)} \circ h^{(i)} \circ \dots \circ \sigma^{(1)} \circ h^{(1)} \right](\mathbf{x}^*)$$

Then, it is possible to compute the back propagation projection corresponding to all hidden layer activation's at once

$$L\left(\mathbf{x}, \mathbf{x}^{(1)*}, \dots, \mathbf{x}^{(n)*}\right) = \sum_{i=1}^n \lambda_i \left[ g^{(i)}\left(\mathbf{x}^{(i)*}\right) - \tilde{g}^{(i)}(\mathbf{x}) \right]^2$$

# However

## Small Problem

- It is possible to show by contradiction that one is unlikely to find a single  $\mathbf{x}^* = \mathbf{x}^{(1)*} = \dots = \mathbf{x}^{(n)*}$ 
  - ▶ Such that you can significantly reduce  $L$

# Proof of the unlikeness of $\boldsymbol{x}^* = \boldsymbol{x}^{(1)*} = \dots = \boldsymbol{x}^{(n)*}$

By the associative property of function composition

$$g^{(i)}(\boldsymbol{x}^*) = (\sigma^{(i)} \circ h^{(i)}) (g^{(i-1)}(\boldsymbol{x}^*))$$

## Proof of the unlikeness of $\mathbf{x}^* = \mathbf{x}^{(1)*} = \dots = \mathbf{x}^{(n)*}$

By the associative property of function composition

$$g^{(i)}(\mathbf{x}^*) = (\sigma^{(i)} \circ h^{(i)}) (g^{(i-1)}(\mathbf{x}^*))$$

Suppose there exist  $\mathbf{x}^* = \mathbf{x}^{(1)*} = \dots = \mathbf{x}^{(n)*}$  an such that

$$(\sigma^{(i)} \circ h^{(i)}) (g^{(i-1)}(\mathbf{x}^*)) = (\tilde{\sigma}^{(i)} \circ h^{(i)}) (\tilde{g}^{(i-1)}(\mathbf{x}))$$

$$(\sigma^{(i-1)} \circ h^{(i-1)}) (g^{(i-2)}(\mathbf{x}^*)) = (\tilde{\sigma}^{(i-1)} \circ h^{(i-1)}) (\tilde{g}^{(i-2)}(\mathbf{x}))$$

# Then

Based on the previous equations

$$g^{(i-1)}(\mathbf{x}^*) = \tilde{g}^{(i-1)}(\mathbf{x})$$

Then

Based on the previous equations

$$g^{(i-1)}(\mathbf{x}^*) = \tilde{g}^{(i-1)}(\mathbf{x})$$

Then, we get

$$(\sigma^{(i)} \circ h^{(i)}) (g^{(i-1)}(\mathbf{x}^*)) = (\tilde{\sigma}^{(i)} \circ h^{(i)}) (g^{(i-1)}(\mathbf{x}))$$

Then

Based on the previous equations

$$g^{(i-1)}(\mathbf{x}^*) = \tilde{g}^{(i-1)}(\mathbf{x})$$

Then, we get

$$(\sigma^{(i)} \circ h^{(i)})\left(g^{(i-1)}(\mathbf{x}^*)\right) = (\tilde{\sigma}^{(i)} \circ h^{(i)})\left(g^{(i-1)}(\mathbf{x})\right)$$

Finally

$$\text{rect}\left(h^{(i)}\left(g^{(i-1)}(\mathbf{x}^*)\right)\right) = M^{(i)} \odot \text{rect}\left(h^{(i)}\left(g^{(i-1)}(\mathbf{x})\right)\right)$$

## Therefore

This is only true if  $M^{(i)} = 1$  or no modification

- When  $\text{rect}_j \left( h^{(i)} \left( g^{(i-1)}(\mathbf{x}^*) \right) \right) > 0$

## Therefore

This is only true if  $M^{(i)} = 1$  or no modification

- When  $\text{rect}_j \left( h^{(i)} \left( g^{(i-1)}(\mathbf{x}^*) \right) \right) > 0$

This only happens with a probability  $p_{(i)}^{d_{(i)} s_{(i)}}$

- Where:
  - ▶  $p_{(i)}$  is the Bernoulli success probability.
  - ▶  $d_{(i)}$  is the number of hidden units.
  - ▶  $s_{(i)}$  is the mean sparsity level at  $i$  (Mean percentage of active hidden units).

# Which is quite low!!!

This probability is very low for standard hyper-parameters values

- With  $p_{(i)} = 0.5$ ,  $d_{(i)} = 1000$  and  $s_{(i)} = 0.15$

$$p_{(i)}^{d_{(i)} s_{(i)}} = 10^{-47}$$

# However

## Fortunately

- It is easy to find a different  $x^*$  for each hidden layer

# However

## Fortunately

- It is easy to find a different  $\mathbf{x}^*$  for each hidden layer

by providing multiple inputs

$$(\mathbf{x}, \mathbf{x}^{(1)*}, \mathbf{x}^{(2)*}, \dots, \mathbf{x}^{(n)*})$$

## However

### Fortunately

- It is easy to find a different  $\mathbf{x}^*$  for each hidden layer

by providing multiple inputs

$$(\mathbf{x}, \mathbf{x}^{(1)*}, \mathbf{x}^{(2)*}, \dots, \mathbf{x}^{(n)*})$$

## However

- This raises the question whether we can train the network deterministically on the  $\mathbf{x}^{(i)*}$  instead of using dropout

# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
- Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
- Random Dropout Probability
  - Projecting Noise into Input Space
  - Augmenting by Noise
- Co-adaptation/Overfitting
  - Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
  - For More in Normalization

# Co-adaptation/Overfitting

## Definition

- Co-adaptation is the accumulation of interacting genes in the gene pool of a population by selection.
  - ▶ Selection pressures on one of the genes will affect its interacting proteins, after which compensatory changes occur.

# Co-adaptation/Overfitting

## Definition

- Co-adaptation is the accumulation of interacting genes in the gene pool of a population by selection.
  - ▶ Selection pressures on one of the genes will affect its interacting proteins, after which compensatory changes occur.

## In Neural Networks

- In neural network, co-adaptation means that some neurons are highly dependent on others:
  - ▶ Getting into over-fitting!!!

# Question

We have that

- Question: Can we train the network deterministically on  $x^{(i)*}$ ?

# Question

We have that

- Question: Can we train the network deterministically on  $\mathbf{x}^{(i)*}$ ?

This is not trivial given that

- Dropout is not effectively applied to every layer at the same time when using

$$(\mathbf{x}, \mathbf{x}^{(1)*}, \mathbf{x}^{(2)*}, \dots, \mathbf{x}^{(n)*})$$

## Question

We have that

- Question: Can we train the network deterministically on  $\mathbf{x}^{(i)*}$ ?

This is not trivial given that

- Dropout is not effectively applied to every layer at the same time when using

$$(\mathbf{x}, \mathbf{x}^{(1)*}, \mathbf{x}^{(2)*}, \dots, \mathbf{x}^{(n)*})$$

- The gradients of the linear projections will differ greatly, different from dropout!!!

# Therefore

We can then

- Modifying the probability distribution is the most straightforward way to improve the set of transformations.

# Therefore

We can then

- Modifying the probability distribution is the most straightforward way to improve the set of transformations.

For example

- A simple way to vary the transformation magnitude randomly is to replace  $p_{hij}$  by a random variable!!!

# Therefore

## Define

$$M_{hij} \sim \mathcal{B}(\rho_h) \text{ (Bernoulli)}$$
$$\rho_h \sim U(0, p_h) \text{ (Uniform)}$$

- where  $h$  defines the layer,  $i$  the sample, and  $j$  the layer's neuron.

# Therefore

## Define

$$M_{hij} \sim \mathcal{B}(\rho_h) \text{ (Bernoulli)}$$
$$\rho_h \sim U(0, p_h) \text{ (Uniform)}$$

- where  $h$  defines the layer,  $i$  the sample, and  $j$  the layer's neuron.

Here, the authors use the same  $\rho$  for all the layers of the neurons, then

$$\tilde{f}(h) = \frac{1}{1-\rho} M \odot rect(h)$$

# Experiments

MLP network with two hidden layers with three architectures

- 2500-625,
- 2500-1250,
- 2500-2500.

# Experiments

MLP network with two hidden layers with three architectures

- 2500-625,
- 2500-1250,
- 2500-2500.

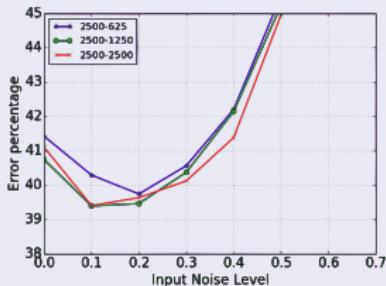
Data Sets where the networks are trained on

- MNIST and CIFAR-10 datasets

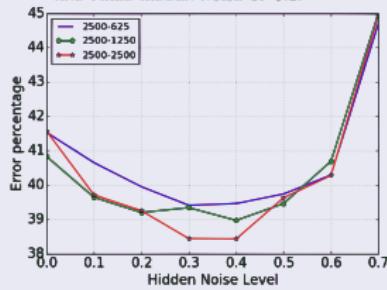
# Results

## Something Notable

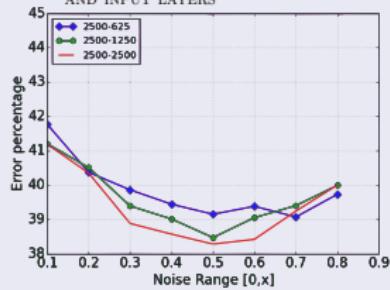
USING DROPOUT WITH VARYING INPUT NOISE  
AND FIXED HIDDEN NOISE OF 0.5.



USING DROPOUT WITH VARYING INPUT NOISE  
AND FIXED HIDDEN NOISE OF 0.2.



USING RANDOM-DROPOUT WITH VARYING  
NOISE RANGE  $[0, x]$  USED AT HIDDEN  
AND INPUT LAYERS



# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
- Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
- Random Dropout Probability
- Projecting Noise into Input Space
- Augmenting by Noise
- Co-adaptation/Overfitting
- **Batch normalization**
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
  - For More in Normalization

Here, the people at Google [14] around 2015

They commented in the “Internal Covariate Shift Phenomena”

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

Here, the people at Google [14] 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.

Here, the people at Google [14] 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 activation’s 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:

$$\hat{\mathbf{x}} = \mathbf{x} - E[\mathbf{x}]$$

- ▶  $\mathcal{X} = \{\mathbf{x}, \dots, \mathbf{x}_N\}$  the data samples and  $E[\mathbf{x}] = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i$

## 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:

$$\hat{\mathbf{x}} = \mathbf{x} - E[\mathbf{x}]$$

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

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

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

## 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:

$$\hat{x} = x - E[x]$$

►  $\mathcal{X} = \{x, \dots, x_N\}$  the data samples and  $E[x] = \frac{1}{N} \sum_{i=1}^N x_i$

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

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

Finally

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

# Then

The following will happen

- The update to  $b$  by  $\Delta b$  leads to **no change** in the output of the layer.

## Then

The following will happen

- The update to  $b$  by  $\Delta b$  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

$$\hat{\mathbf{x}} = \text{Norm}(\mathbf{x}, \mathcal{X})$$

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

# Normalization via Mini-Batch Statistic

It is possible to describe the **normalization** as a transformation layer

$$\hat{\mathbf{x}} = \text{Norm}(\mathbf{x}, \mathcal{X})$$

- 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

$$\frac{\partial \text{Norm}(\mathbf{x}, \mathcal{X})}{\partial \mathbf{x}} \text{ and } \frac{\partial \text{Norm}(\mathbf{x}, \mathcal{X})}{\partial \mathcal{X}}$$

# Definition of Whitening

## Whitening

- Suppose  $X$  is a random (column) vector with non-singular covariance matrix  $\Sigma$  and mean 0.

# Definition of Whitening

## Whitening

- Suppose  $X$  is a random (column) vector with non-singular covariance matrix  $\Sigma$  and mean 0.

## Then

- Then the transformation  $Y = WX$  with a whitening matrix  $W$  satisfying the condition  $W^T W = \Sigma^{-1}$  yields the whitened random vector  $Y$  with unit diagonal covariance.

## Such Normalization

It could be used for all layer

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

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

- ▶ To produce the whitened activations

## Therefore

A Better Options, we can normalize each input layer

$$\hat{\mathbf{x}}^{(k)} = \frac{\mathbf{x}^{(k)} - \mu}{\sigma}$$

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

## Therefore

A Better Options, we can normalize each input layer

$$\hat{\mathbf{x}}^{(k)} = \frac{\mathbf{x}^{(k)} - \mu}{\sigma}$$

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

This allows to speed up convergence

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

## Therefore

A Better Options, we can normalize each input layer

$$\hat{\mathbf{x}}^{(k)} = \frac{\mathbf{x}^{(k)} - \mu}{\sigma}$$

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

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

$$\mathbf{y}^{(k)} = \gamma^{(k)} \hat{\mathbf{x}}^{(k)} + \beta^{(k)}$$

# The Transformation

## The Linear transformation

$$\mathbf{y}^{(k)} = \gamma^{(k)} \hat{\mathbf{x}}^{(k)} + \beta^{(k)}$$

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

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

# Finally

## Batch Normalizing Transform

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_{1\dots m}\}$ , Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = BN_{\gamma, \beta}(x_i)\}$

# Finally

## Batch Normalizing Transform

**Input:** Values of  $\mathbf{x}$  over a mini-batch:  $\mathcal{B} = \{\mathbf{x}_1 \dots \mathbf{x}_m\}$ , Parameters to be learned:  $\gamma, \beta$

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

①  $\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i$

# Finally

## Batch Normalizing Transform

**Input:** Values of  $\mathbf{x}$  over a mini-batch:  $\mathcal{B} = \{\mathbf{x}_1 \dots \mathbf{x}_m\}$ , Parameters to be learned:  $\gamma, \beta$

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

$$① \mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i$$

$$② \sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^m (\mathbf{x}_i - \mu)^2$$

# Finally

## Batch Normalizing Transform

**Input:** Values of  $\mathbf{x}$  over a mini-batch:  $\mathcal{B} = \{\mathbf{x}_1 \dots \mathbf{x}_m\}$ , Parameters to be learned:  $\gamma, \beta$

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

$$① \mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i$$

$$② \sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^m (\mathbf{x}_i - \mu)^2$$

$$③ \hat{\mathbf{x}} = \frac{\mathbf{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

# Finally

## Batch Normalizing Transform

**Input:** Values of  $\mathbf{x}$  over a mini-batch:  $\mathcal{B} = \{\mathbf{x}_1 \dots \mathbf{x}_m\}$ , Parameters to be learned:  $\gamma, \beta$

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

①  $\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i$

②  $\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^m (\mathbf{x}_i - \mu)^2$

③  $\hat{\mathbf{x}} = \frac{\mathbf{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$

④  $\mathbf{y}_i = \gamma^{(k)} \hat{\mathbf{x}}_i + \beta = BN_{\gamma, \beta}(\mathbf{x}_i)$

# Backpropagation

We have the following equations by using the loss function  $l$

$$\textcircled{1} \quad \frac{\partial l}{\partial \hat{x}_i} = \frac{\partial l}{\partial y_i} \times \gamma$$

# Backpropagation

We have the following equations by using the loss function  $l$

$$\textcircled{1} \quad \frac{\partial l}{\partial \hat{x}_i} = \frac{\partial l}{\partial y_i} \times \gamma$$

$$\textcircled{2} \quad \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^m \frac{\partial l}{\partial \hat{x}_i} \times (\mathbf{x}_i - \mu_{\mathcal{B}}) \times \left(-\frac{1}{2}\right) \times (\sigma_{\mathcal{B}}^2 + \epsilon)^{-\frac{3}{2}}$$

# Backpropagation

We have the following equations by using the loss function  $l$

$$① \frac{\partial l}{\partial \hat{x}_i} = \frac{\partial l}{\partial y_i} \times \gamma$$

$$② \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^m \frac{\partial l}{\partial \hat{x}_i} \times (\mathbf{x}_i - \mu_{\mathcal{B}}) \times \left(-\frac{1}{2}\right) \times (\sigma_{\mathcal{B}}^2 + \epsilon)^{-\frac{3}{2}}$$

$$③ \frac{\partial l}{\partial \mu_{\mathcal{B}}} = \left( \sum_{i=1}^m \frac{\partial l}{\partial \hat{x}_i} \times \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{\sum_{i=1}^m -2 \times (\mathbf{x}_i - \mu_{\mathcal{B}})}{m}$$

# Backpropagation

We have the following equations by using the loss function  $l$

$$\textcircled{1} \quad \frac{\partial l}{\partial \hat{x}_i} = \frac{\partial l}{\partial y_i} \times \gamma$$

$$\textcircled{2} \quad \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^m \frac{\partial l}{\partial \hat{x}_i} \times (\mathbf{x}_i - \mu_{\mathcal{B}}) \times \left(-\frac{1}{2}\right) \times (\sigma_{\mathcal{B}}^2 + \epsilon)^{-\frac{3}{2}}$$

$$\textcircled{3} \quad \frac{\partial l}{\partial \mu_{\mathcal{B}}} = \left( \sum_{i=1}^m \frac{\partial l}{\partial \hat{x}_i} \times \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{\sum_{i=1}^m -2 \times (\mathbf{x}_i - \mu_{\mathcal{B}})}{m}$$

$$\textcircled{4} \quad \frac{\partial l}{\partial x_i} = \frac{\partial l}{\partial \hat{x}_i} \times \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{2 \times (\mathbf{x}_i - \mu_{\mathcal{B}})}{m} + \frac{\partial l}{\partial \mu_{\mathcal{B}}} \times \frac{1}{m}$$

# Backpropagation

We have the following equations by using the loss function  $l$

$$\textcircled{1} \quad \frac{\partial l}{\partial \hat{x}_i} = \frac{\partial l}{\partial y_i} \times \gamma$$

$$\textcircled{2} \quad \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^m \frac{\partial l}{\partial \hat{x}_i} \times (\mathbf{x}_i - \mu_{\mathcal{B}}) \times \left(-\frac{1}{2}\right) \times (\sigma_{\mathcal{B}}^2 + \epsilon)^{-\frac{3}{2}}$$

$$\textcircled{3} \quad \frac{\partial l}{\partial \mu_{\mathcal{B}}} = \left( \sum_{i=1}^m \frac{\partial l}{\partial \hat{x}_i} \times \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{\sum_{i=1}^m -2 \times (\mathbf{x}_i - \mu_{\mathcal{B}})}{m}$$

$$\textcircled{4} \quad \frac{\partial l}{\partial \mathbf{x}_i} = \frac{\partial l}{\partial \hat{x}_i} \times \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{2 \times (\mathbf{x}_i - \mu_{\mathcal{B}})}{m} + \frac{\partial l}{\partial \mu_{\mathcal{B}}} \times \frac{1}{m}$$

$$\textcircled{5} \quad \frac{\partial l}{\partial \gamma} = \sum_{i=1}^m \frac{\partial l}{\partial y_i} \times \hat{\mathbf{x}}_i$$

# Backpropagation

We have the following equations by using the loss function  $l$

$$\textcircled{1} \quad \frac{\partial l}{\partial \hat{x}_i} = \frac{\partial l}{\partial y_i} \times \gamma$$

$$\textcircled{2} \quad \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^m \frac{\partial l}{\partial \hat{x}_i} \times (\mathbf{x}_i - \mu_{\mathcal{B}}) \times \left(-\frac{1}{2}\right) \times (\sigma_{\mathcal{B}}^2 + \epsilon)^{-\frac{3}{2}}$$

$$\textcircled{3} \quad \frac{\partial l}{\partial \mu_{\mathcal{B}}} = \left( \sum_{i=1}^m \frac{\partial l}{\partial \hat{x}_i} \times \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{\sum_{i=1}^m -2 \times (\mathbf{x}_i - \mu_{\mathcal{B}})}{m}$$

$$\textcircled{4} \quad \frac{\partial l}{\partial \mathbf{x}_i} = \frac{\partial l}{\partial \hat{x}_i} \times \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{2 \times (\mathbf{x}_i - \mu_{\mathcal{B}})}{m} + \frac{\partial l}{\partial \mu_{\mathcal{B}}} \times \frac{1}{m}$$

$$\textcircled{5} \quad \frac{\partial l}{\partial \gamma} = \sum_{i=1}^m \frac{\partial l}{\partial y_i} \times \hat{\mathbf{x}}_i$$

$$\textcircled{6} \quad \frac{\partial l}{\partial \beta} = \sum_{i=1}^m \frac{\partial l}{\partial y_i}$$

# Training Batch Normalization Networks

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

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

# Training Batch Normalization Networks

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

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

①  $N_{BN}^{tr} = N$  // Training BN network

# Training Batch Normalization Networks

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

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

- ①  $N_{BN}^{tr} = N$  // Training BN network
- ② for  $k = 1 \dots K$  do

# Training Batch Normalization Networks

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

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

- ①  $N_{BN}^{tr} = N$  // Training BN network
- ② for  $k = 1 \dots K$  do
- ③     Add transformation  $y^{(k)} = BN_{\gamma^{(k)}, \beta^{(k)}}(\mathbf{x}^{(k)})$  to  $N_{BN}^{tr}$

# Training Batch Normalization Networks

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

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

- ①  $N_{BN}^{tr} = N$  // Training BN network
- ② for  $k = 1 \dots K$  do
- ③     Add transformation  $y^{(k)} = BN_{\gamma^{(k)}, \beta^{(k)}}(\mathbf{x}^{(k)})$  to  $N_{BN}^{tr}$
- ④     Modify each layer in  $N_{BN}^{tr}$  with input  $\mathbf{x}^{(k)}$  to take  $y^{(k)}$  instead

# Training Batch Normalization Networks

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

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

- ①  $N_{BN}^{tr} = N$  // Training BN network
- ② for  $k = 1 \dots K$  do
- ③     Add transformation  $y^{(k)} = BN_{\gamma^{(k)}, \beta^{(k)}}(\mathbf{x}^{(k)})$  to  $N_{BN}^{tr}$
- ④     Modify each layer in  $N_{BN}^{tr}$  with input  $\mathbf{x}^{(k)}$  to take  $y^{(k)}$  instead
- ⑤ Train  $N_{BN}^{tr}$  to optimize the parameters  $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^K$

# Training Batch Normalization Networks

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

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

- ①  $N_{BN}^{tr} = N$  // Training BN network
- ② for  $k = 1 \dots K$  do
- ③     Add transformation  $y^{(k)} = BN_{\gamma^{(k)}, \beta^{(k)}}(\mathbf{x}^{(k)})$  to  $N_{BN}^{tr}$
- ④     Modify each layer in  $N_{BN}^{tr}$  with input  $\mathbf{x}^{(k)}$  to take  $y^{(k)}$  instead
- ⑤ Train  $N_{BN}^{tr}$  to optimize the parameters  $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^K$
- ⑥  $N_{BN}^{inf} = N_{BN}^{tr}$  // Inference BN network with frozen parameters

# Training Batch Normalization Networks

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

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

- ①  $N_{BN}^{tr} = N$  // Training BN network
- ② for  $k = 1 \dots K$  do
- ③     Add transformation  $y^{(k)} = BN_{\gamma^{(k)}, \beta^{(k)}}(\mathbf{x}^{(k)})$  to  $N_{BN}^{tr}$
- ④     Modify each layer in  $N_{BN}^{tr}$  with input  $\mathbf{x}^{(k)}$  to take  $y^{(k)}$  instead
- ⑤ Train  $N_{BN}^{tr}$  to optimize the parameters  $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^K$
- ⑥  $N_{BN}^{inf} = N_{BN}^{tr}$  // Inference BN network with frozen parameters
- ⑦ for  $k = 1 \dots K$  do

# Training Batch Normalization Networks

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

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

- ①  $N_{BN}^{tr} = N$  // Training BN network
- ② for  $k = 1 \dots K$  do
- ③     Add transformation  $y^{(k)} = BN_{\gamma^{(k)}, \beta^{(k)}}(\mathbf{x}^{(k)})$  to  $N_{BN}^{tr}$
- ④     Modify each layer in  $N_{BN}^{tr}$  with input  $\mathbf{x}^{(k)}$  to take  $y^{(k)}$  instead
- ⑤ Train  $N_{BN}^{tr}$  to optimize the parameters  $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^K$
- ⑥  $N_{BN}^{inf} = N_{BN}^{tr}$  // Inference BN network with frozen parameters
- ⑦ for  $k = 1 \dots K$  do
- ⑧     Process multiple training mini-batches  $\mathcal{B}$ , each of size  $m$ , and average over them

# Training Batch Normalization Networks

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

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

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

# Training Batch Normalization Networks

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

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

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

## However

Santurkar et al. [15]

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

## However

Santurkar et al. [15]

- They found that 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.

# However

Santurkar et al. [15]

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

Santurkar et al. recognize that

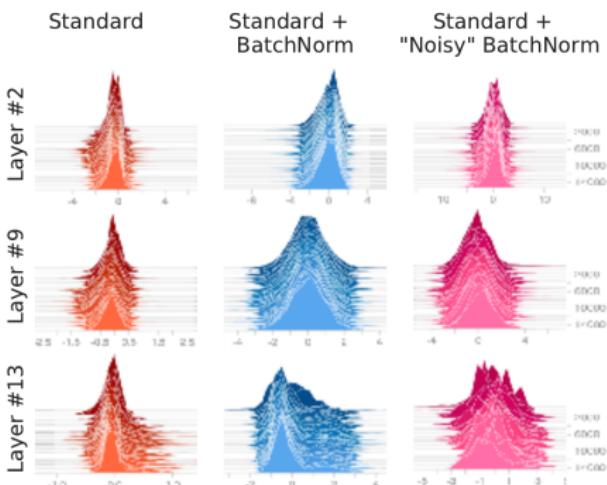
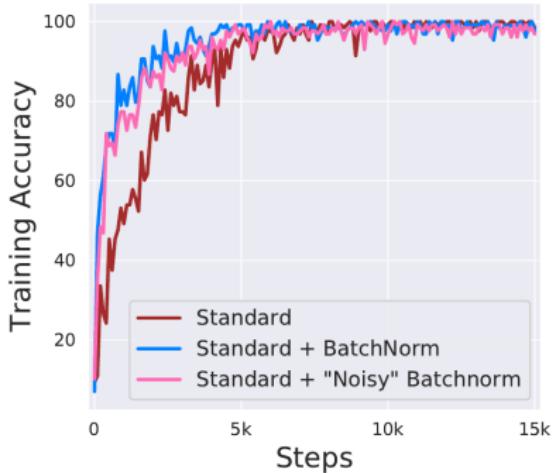
- Batch normalization has been arguably one of the most successful architectural innovations in deep learning.

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!!!

# 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

# 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  $\mathcal{L}$ ,

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

# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
- Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
- Random Dropout Probability
- Projecting Noise into Input Space
- Augmenting by Noise
- Co-adaptation/Overfitting
- Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
- For More in Normalization

## Remember

Using Min-Batch inputs, we have

$$\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i$$

# Remember

Using Min-Batch inputs, we have

$$\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i$$

And Variance

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^m (\mathbf{x}_i - \mu_{\mathcal{B}})^2$$

## Therefore, Ba et al. [16]

We get the mean over the output of the layer  $l$  with  $H$  number of hidden units

$$\mu^l = \frac{1}{H} \sum_{i=1}^H y_i^l$$

- Basically, do the forward process then add over the output  $y_i^l = w_i^{lT} h^l$  where  $h_i^{l+1} = f(y_i^l + b_i^l)$

## Therefore, Ba et al. [16]

We get the mean over the output of the layer  $l$  with  $H$  number of hidden units

$$\mu^l = \frac{1}{H} \sum_{i=1}^H y_i^l$$

- Basically, do the forward process then add over the output  $y_i^l = w_i^{lT} h^l$  where  $h_i^{l+1} = f(y_i^l + b_i^l)$

Then the standard deviation layer  $l$

$$\sigma^l = \sqrt{\frac{1}{H} \sum_{i=1}^H (y_i^l - \mu^l)^2}$$

## Remarks

We have that

- All the hidden units in a layer share the same normalization terms  $\mu$  and  $\sigma$ 
  - ▶ but different training cases have different normalization terms.

## Remarks

We have that

- All the hidden units in a layer share the same normalization terms  $\mu$  and  $\sigma$ 
  - ▶ but different training cases have different normalization terms.

Layer normalization does not impose any constraint

- On the size of a mini-batch and it can be used in the pure on-line regime with batch size 1.

# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
- Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
- Random Dropout Probability
- Projecting Noise into Input Space
- Augmenting by Noise
- Co-adaptation/Overfitting
- **Batch normalization**
  - Improving the Google Layer Normalization
  - **Layer Normalization in RNN**
  - Invariance Under Weights and Data Transformations
  - For More in Normalization

# The Flow of Information through time

First, the new  $h^t$  with a gain vector  $g$

$$h^t = f \left[ \frac{g}{\sigma^t} \odot (y^t - \mu^t) + b \right]$$

# The Flow of Information through time

First, the new  $h^t$  with a gain vector  $g$

$$h^t = f \left[ \frac{\mathbf{g}}{\sigma^t} \odot (\mathbf{y}^t - \mu^t) + b \right]$$

The Temporal Layer Mean Normalization

$$\mu^t = \frac{1}{H} \sum_{i=1}^H y_i^t$$

# The Flow of Information through time

First, the new  $h^t$  with a gain vector  $g$

$$h^t = f \left[ \frac{\mathbf{g}}{\sigma^t} \odot (\mathbf{y}^t - \mu^t) + b \right]$$

The Temporal Layer Mean Normalization

$$\mu^t = \frac{1}{H} \sum_{i=1}^H y_i^t$$

The Temporal Layer STD Normalization

$$\sigma^t = \sqrt{\frac{1}{H} \sum_{i=1}^H (y_i^t - \mu^t)^2}$$

# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
- Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
- Random Dropout Probability
- Projecting Noise into Input Space
- Augmenting by Noise
- Co-adaptation/Overfitting
- **Batch normalization**
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - **Invariance Under Weights and Data Transformations**
- For More in Normalization

## Weight re-scaling and re-centering

Observe that under batch normalization and weight normalization

- Any re-scaling to the incoming weights  $w_i$  of a single neuron has no effect on the normalized summed inputs to a neuron.

# Weight re-scaling and re-centering

Observe that under batch normalization and weight normalization

- Any re-scaling to the incoming weights  $w_i$  of a single neuron has no effect on the normalized summed inputs to a neuron.

Meaning

- If the weight vector is scaled by  $\delta_i$  the two scalars  $\mu$  and  $\sigma$  will also be scaled by  $\delta$

# Weight re-scaling and re-centering

Observe that under batch normalization and weight normalization

- Any re-scaling to the incoming weights  $w_i$  of a single neuron has no effect on the normalized summed inputs to a neuron.

Meaning

- If the weight vector is scaled by  $\delta_i$  the two scalars  $\mu$  and  $\sigma$  will also be scaled by  $\delta$

Properties

- The batch and weight normalization are invariant to the re-scaling of the weights.

# In the other hand

## Layer normalization

- It is not invariant to the individual scaling of the single weight vectors.

## In the other hand

### Layer normalization

- It is not invariant to the individual scaling of the single weight vectors.

### However

- Layer normalization is invariant to scaling of the entire weight matrix.

## In the other hand

### Layer normalization

- It is not invariant to the individual scaling of the single weight vectors.

### However

- Layer normalization is invariant to scaling of the entire weight matrix.
- Also it is invariant to a shift to all of the incoming weights in the weight matrix.

# How?

Imagine the following

- Let there be two sets of model parameters  $\theta, \theta'$  with weight matrices

$$W' = \delta W + 1\gamma^T$$

We have

Given that  $y_i^l = w_i^{lT} \mathbf{x}^l$

$$y_i'^l = (\delta W + 1\gamma^T)_i \mathbf{x}^l$$

We have

Given that  $y_i^l = w_i^{lT} \mathbf{x}^l$

$$y_i'^l = (\delta W + 1\gamma^T)_i \mathbf{x}^l$$

Then, we have

$$\mu'^l = \frac{\delta}{H} \sum_{i=1}^H W_i \mathbf{x}^l + \frac{1}{H} \sum_{i=1}^H (1\gamma^T)_i \mathbf{x}^l = \delta\mu + (1\gamma^T)_i \mathbf{x}^l$$

Now

## Standard Deviation

$$\sigma' = \sqrt{\frac{1}{H} \sum_{i=1}^H (y_i'^l - \mu')^2} = \delta \sqrt{\frac{1}{H} \sum_{i=1}^H (y_i^l - \mu)^2}$$

Now

## Standard Deviation

$$\sigma' = \sqrt{\frac{1}{H} \sum_{i=1}^H (y_i'^l - \mu')^2} = \delta \sqrt{\frac{1}{H} \sum_{i=1}^H (y_i^l - \mu)^2}$$

Finally, Under Layer Normalization, we have the same output

$$\begin{aligned}\mathbf{h}' &= f \left[ \frac{\mathbf{g}}{\sigma'} (\mathbf{W}' \mathbf{x} - \mu') + \mathbf{b} \right] \\ &= f \left[ \frac{\mathbf{g}}{\sigma'} \left( [\delta \mathbf{W} + 1 \gamma^T] \mathbf{x} - \mu' \right) + \mathbf{b} \right] \\ &= f \left[ \frac{\mathbf{g}}{\sigma} (\mathbf{W} \mathbf{x} - \mu) + \mathbf{b} \right] = \mathbf{h}\end{aligned}$$

# Remarks

## Something Notable

- if normalization is only applied to the input before the weights, the model will not be invariant to re-scaling and re-centering of the weights.

# Data re-scaling and re-centering

We can show

- All the normalization methods are invariant to re-scaling the dataset

# Data re-scaling and re-centering

We can show

- All the normalization methods are invariant to re-scaling the dataset

Layer normalization is invariant to re-scaling of individual training cases

$$h'_i = f \left[ \frac{g_i}{\sigma'} \left( w_i^T \mathbf{x}' - \mu' \right) + b_i \right] = f \left[ \frac{g_i}{\delta\sigma} \left( \delta w_i^T \mathbf{x} - \delta\mu \right) + b_i \right] = h_i$$

## Additionally

Layer Normalization has a relation with the Fisher Information Matrix

$$F(\theta) = E_{\mathbf{x} \sim P(\mathbf{x}), y \sim P(y|\mathbf{x})} \left[ \frac{\partial \log P(y|\mathbf{x})}{\partial \theta} \left( \frac{\partial \log P(y|\mathbf{x})}{\partial \theta} \right)^T \right]$$

## Additionally

Layer Normalization has a relation with the Fisher Information Matrix

$$F(\theta) = E_{\mathbf{x} \sim P(\mathbf{x}), y \sim P(y|\mathbf{x})} \left[ \frac{\partial \log P(y|\mathbf{x})}{\partial \theta} \left( \frac{\partial \log P(y|\mathbf{x})}{\partial \theta} \right)^T \right]$$

Basically, we can write the generalized linear model as

$$\log P(y|\mathbf{x}, w, b) = \frac{(a+b)y - \eta(a+b)}{\Phi} + c(y, \Phi)$$

$$E[y|\mathbf{x}] = f(a+b) = f(w^T \mathbf{x} + b)$$

$$Var[y|\mathbf{x}] = \Phi f'(a+b)$$

# The curvature of a Riemannian manifold

It is entirely captured by its Riemannian metric

$$ds^2 \approx \frac{1}{2} \delta^T F(\theta) \delta$$

- where,  $\delta$  is a small change to the parameters.

# The curvature of a Riemannian manifold

It is entirely captured by its Riemannian metric

$$ds^2 \approx \frac{1}{2} \delta^T F(\theta) \delta$$

- where,  $\delta$  is a small change to the parameters.

Then, under Layer Normalization, we have

$$F(\theta) = \frac{1}{\Phi^2} E_{x \sim P(\mathbf{x})} \begin{bmatrix} Cov(y_1, y_2 | \mathbf{x}) \frac{(a_1 - \mu)^2}{\sigma^2} & \dots & Cov(y_1, y_H | \mathbf{x}) \frac{(a_1 - \mu)(a_H - \mu)}{\sigma^2} \\ \vdots & \ddots & \vdots \\ Cov(y_H, y_1 | \mathbf{x}) \frac{(a_1 - \mu)(a_H - \mu)}{\sigma^2} & \dots & Cov(y_H, y_H | \mathbf{x}) \frac{(a_H - \mu)^2}{\sigma^2} \end{bmatrix}$$

## Where

We have that  $a_i = w_i^T \mathbf{x}$

- We project the gradient updates to the gain parameter  $\delta_{gi}$  of the  $i^{th}$  neuron to its weight vector as

$$\frac{\delta_{gi}\delta_{gj}}{2\Phi^2} E_{x \sim P(\mathbf{x})} \left[ Cov(y_i, y_j | \mathbf{x}) \frac{(a_1 - \mu)(a_H - \mu)}{\sigma^2} \right]$$

## Where

We have that  $a_i = w_i^T \mathbf{x}$

- We project the gradient updates to the gain parameter  $\delta_{gi}$  of the  $i^{th}$  neuron to its weight vector as

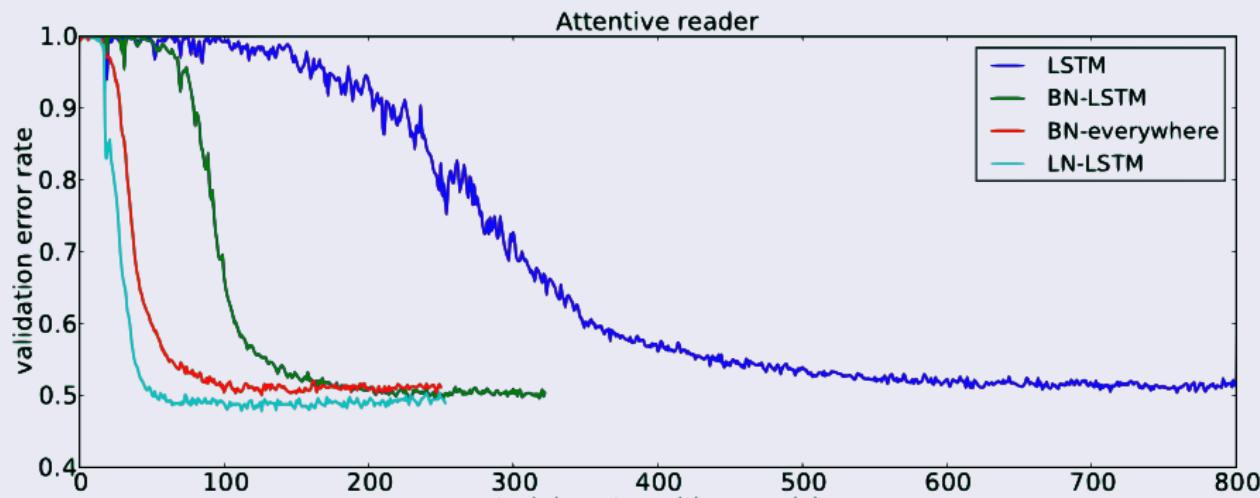
$$\frac{\delta_{gi}\delta_{gj}}{2\Phi^2} E_{x \sim P(\mathbf{x})} \left[ Cov(y_i, y_j | \mathbf{x}) \frac{(a_1 - \mu)(a_H - \mu)}{\sigma^2} \right]$$

## Basically

- We have that the normalization layer is more robust to the scaling of the input and parameters

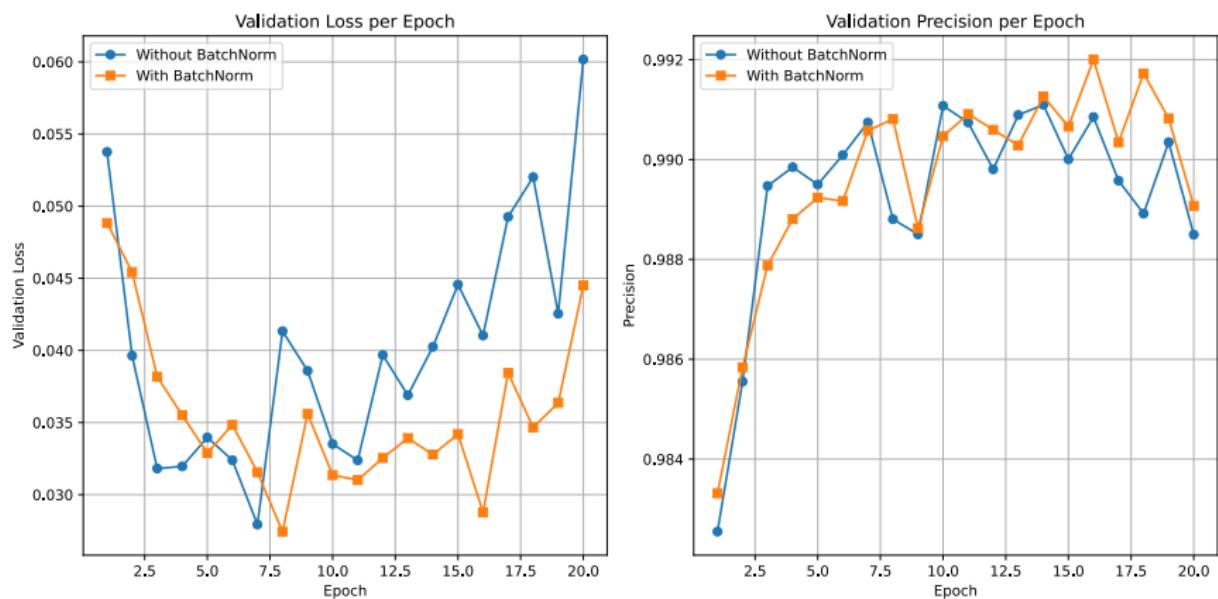
# Results

## In a LSTM



# Example

## BatchNormalization Comparison



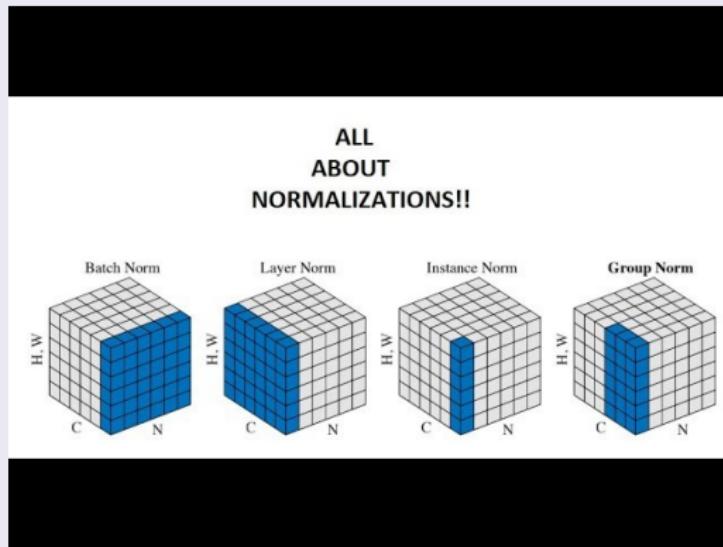
# Outline

- 1 Problems with Deeper Architectures
  - The Degradation Problem
  - The Vanishing and Exploding Gradients
  - An Example of The Vanishing and Exploding Gradient
  - Vanishing and Exploding Gradient as Fixed Points
- 2 The Problem with Overfitting
  - Intuition from Overfitting
  - The Idea of Regularization
  - The Horrible Landscape of Optimization in Deep Learning
- 3 Methods for Improving Learning in Deep Networks
  - Gradient Clipping for Exploding Gradient
  - The Correlation between Gradient Norm and Local Smoothness of Deep Learners

- Gaussian Noise on Hidden Units for Regularization
- Application into a Decoder/Encoder
- Dropout as Regularization
  - Introduction
  - Dropout Process
  - Dropout as Bagging/Bootstrap Aggregation
  - Beyond an Empirical Probabilities, LASSO and Data Flow
- Random Dropout Probability
- Projecting Noise into Input Space
- Augmenting by Noise
- Co-adaptation/Overfitting
- Batch normalization
  - Improving the Google Layer Normalization
  - Layer Normalization in RNN
  - Invariance Under Weights and Data Transformations
- **For More in Normalization**

# We have more types

We have this



# Conclusions

There is still a lot to understand on the Deep Learning Architectures

- The Last 10 years have shown us a lot on the need of regularization...

# Conclusions

There is still a lot to understand on the Deep Learning Architectures

- The Last 10 years have shown us a lot on the need of regularization...

Therefore

- When connecting with the paper
  - ▶ “How Does Batch Normalization Help Optimization?” by Santurkar, Tsipras, Ilyas and Madry

# Conclusions

There is still a lot to understand on the Deep Learning Architectures

- The Last 10 years have shown us a lot on the need of regularization...

Therefore

- When connecting with the paper
  - ▶ “How Does Batch Normalization Help Optimization?” by Santurkar, Tsipras, Ilyas and Madry

We have the if we were able to connect these normalizations

- With the building of the Jacobian on the Gradient Descent, we could improve
  - ▶ The speed of optimization + The regularization properties of such Gradient Descent

-  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. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
-  S. Bulusu, B. Kailkhura, B. Li, P. K. Varshney, and D. Song, "Anomalous example detection in deep learning: A survey," *IEEE Access*, vol. 8, pp. 132330–132347, 2020.
-  J. Kauffmann, K.-R. Müller, and G. Montavon, "Towards explaining anomalies: a deep taylor decomposition of one-class models," *Pattern Recognition*, vol. 101, p. 107198, 2020.

- T. DeVries and G. W. Taylor, "Dataset augmentation in feature space," *arXiv preprint arXiv:1702.05538*, 2017.
- C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of big data*, vol. 6, no. 1, pp. 1–48, 2019.
- C. Shorten, T. M. Khoshgoftaar, and B. Furht, "Text data augmentation for deep learning," *Journal of big Data*, vol. 8, no. 1, pp. 1–34, 2021.
- Q. Wen, L. Sun, F. Yang, X. Song, J. Gao, X. Wang, and H. Xu, "Time series data augmentation for deep learning: A survey," *arXiv preprint arXiv:2002.12478*, 2020.
- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

-  S. Wager, S. Wang, and P. S. Liang, "Dropout training as adaptive regularization," in *Advances in neural information processing systems*, pp. 351–359, 2013.
-  T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics, Springer New York, 2009.
-  X. Bouthillier, K. Konda, P. Vincent, and R. Memisevic, "Dropout as data augmentation," *arXiv preprint arXiv:1506.08700*, 2015.
-  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.



J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *arXiv preprint arXiv:1607.06450*, 2016.