ECE 1508S2: Applied Deep Learning

Chapter 5: Advancing Our Bag of Tools II

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Winter 2024

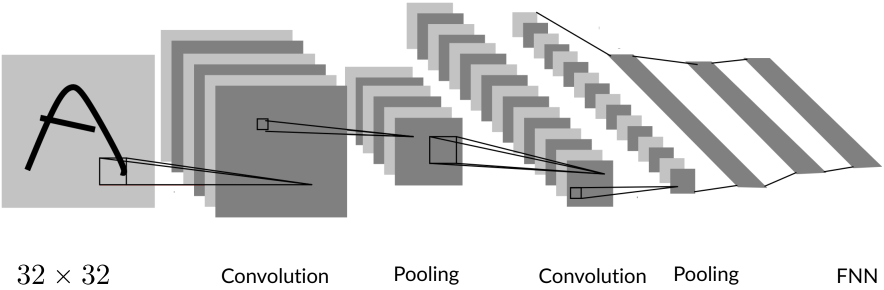
A Bit of History: *LeNet*

CNNs was first trained via gradient-based algorithms by *Yan LeCun* and his team: *they could develop backpropagation through CNNs and hence they were able to efficiently implement it*[1]

[1] Check out their paper at this link! The diagram is taken from the the paper

A Bit of History: *LeNet*

CNNs was first trained via gradient-based algorithms by *Yan LeCun* and his team: *they could develop backpropagation through CNNs and hence they were able to efficiently implement it*[1]



[1]Check out their paper at this link! The diagram is taken from the the paper

ILSVRC: *ImageNet Large Scale Visual Recognition Challenge*

The project ImageNet started a competition in 2010:

ILSVRC: *ImageNet Large Scale Visual Recognition Challenge*

The project ImageNet started a competition in 2010: *it introduced a training dataset of 1.2 million images from 1000 different labels. The proposed trained architectures are then validated by a test dataset. The winner is the model with minimal classification error*

ILSVRC: *ImageNet Large Scale Visual Recognition Challenge*

The project ImageNet started a competition in 2010: *it introduced a training dataset of 1.2 million images from 1000 different labels. The proposed trained architectures are then validated by a test dataset. The winner is the model with minimal classification error*

The winners in 2010 and 2011 used shallow NNs!

- *2010: Team NEC-UIUC*
- *2011: Team XRCE*

ILSVRC: *ImageNet Large Scale Visual Recognition Challenge*

The project ImageNet started a competition in 2010: *it introduced a training dataset of 1.2 million images from 1000 different labels. The proposed trained architectures are then validated by a test dataset. The winner is the model with minimal classification error*

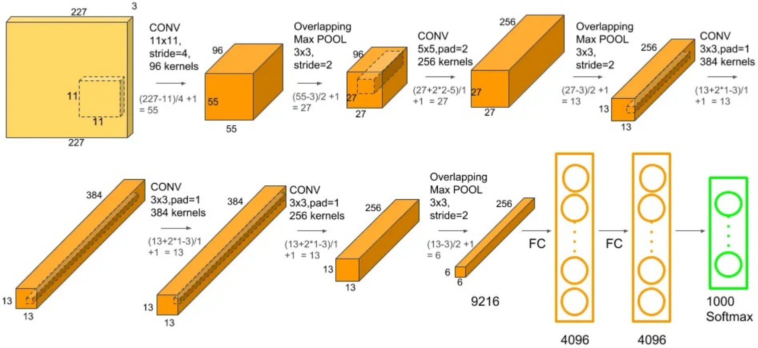
The winners in 2010 and 2011 used shallow NNs!

- *2010: Team NEC-UIUC*
- *2011: Team XRCE*

In 2012, Supervision from U of T trained a deep CNN and won ILSVRC

First Deep Winner: *AlexNet*

Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton proposed AlexNet *which could greatly reduce the classification error*[2]



[2] Check it out in their paper!

AlexNet to ZFNet

AlexNet was a deep CNN with 8 learnable layers

- *5 convolutional layers*
- *3 fully-connected layers*

[3] You may find details in their paper

AlexNet to ZFNet

AlexNet was a deep CNN with 8 learnable layers

- *5 convolutional layers*
- *3 fully-connected layers*

Zeiler and Fergus won ILSVRC in 2013 by *using the same architecture but doing accurate hyperparameter tuning*[3]

[3]You may find details in their paper

AlexNet to ZFNet

AlexNet was a deep CNN with 8 learnable layers

- *5 convolutional layers*
- *3 fully-connected layers*

Zeiler and Fergus won ILSVRC in 2013 by *using the same architecture but doing accurate hyperparameter tuning*[3]

At this point, going deeper considered as the key to success

[3] You may find details in their paper

VGG Architectures

Visual Geometry Group at Oxford University developed deeper CNNs: *a class of architectures*[4]

- *VGG-11*
- *VGG-13*
- *VGG-16*
- *VGG-19*

[4]Check VGG architectures out in their paper

VGG Architectures

Visual Geometry Group at Oxford University developed deeper CNNs: *a class of architectures*[4]

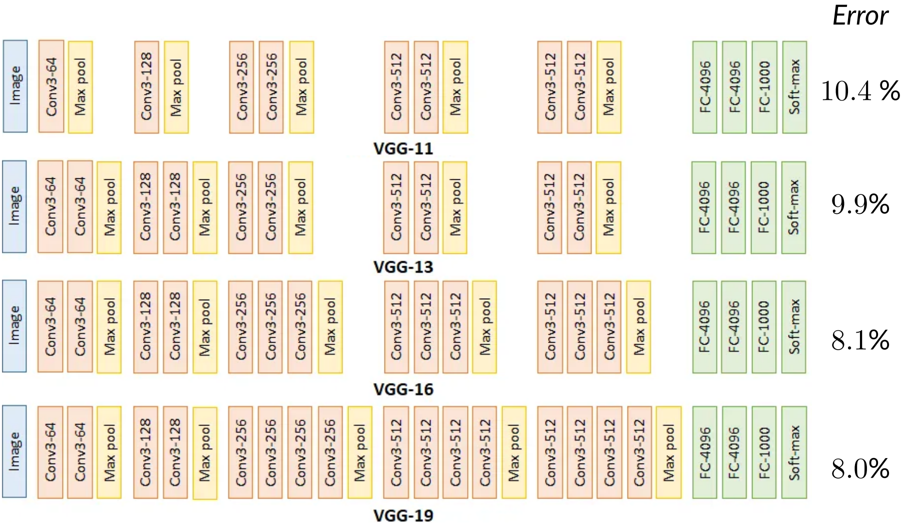
- *VGG-11*
- *VGG-13*
- *VGG-16*
- *VGG-19*

They could won ILSVRC localization task and took second place in classification: their results also showed an interesting finding

> *As we go deeper, the accuracy gets higher notably up to VGG-16; however, VGG-19 can only give marginal improvements!*

[4] Check VGG architectures out in their paper

VGG Architectures



GoogLeNet: *2014 Classification Winner*

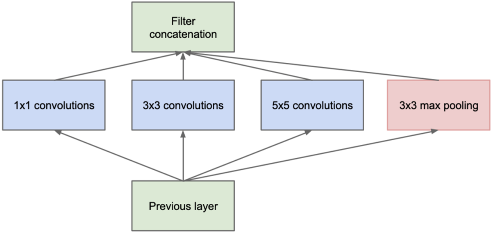
In 2014, Google introduced GoogLeNet: *a deep CNN which uses fully-connected layer only at the output and is purely based on CNN*[5]

[5]Check GoogLeNet in their paper. The diagram is taken from original paper

GoogLeNet: *2014 Classification Winner*

In 2014, Google introduced GoogLeNet: *a deep CNN which uses fully-connected layer only at the output and is purely based on CNN*[5]

- *They introduced a new module called "inception module"*
  ↳ *It's a collection of parallel convolutions and poolings*

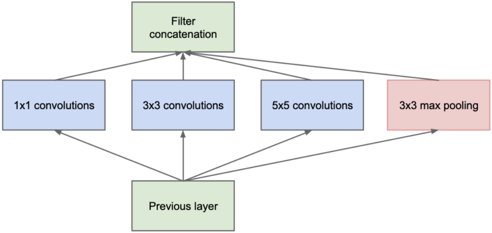


[5] Check GoogLeNet in their paper. The diagram is taken from original paper

GoogLeNet: *2014 Classification Winner*

In 2014, Google introduced GoogLeNet: *a deep CNN which uses fully-connected layer only at the output and is purely based on CNN*[5]

- *They introduced a new module called "inception module"*
  ↳ *It's a collection of parallel convolutions and poolings*



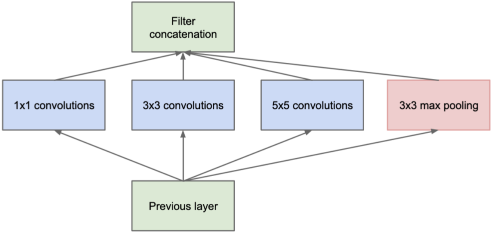
- *Since there is no fully-connected layer it has much less model parameters*
  ↳ *It hence requires less memory and is trained faster*

[5]Check GoogLeNet in their paper. The diagram is taken from original paper

GoogLeNet: *2014 Classification Winner*

In 2014, Google introduced GoogLeNet: *a deep CNN which uses fully-connected layer only at the output and is purely based on CNN*[5]

- *They introduced a new module called "inception module"*
  ↳ *It's a collection of parallel convolutions and poolings*



- *Since there is no fully-connected layer it has much less model parameters*
  ↳ *It hence requires less memory and is trained faster*

GoogLeNet won ILSVRC classification task

[5] Check GoogLeNet in their paper. The diagram is taken from original paper

A Hurdle in Going Deeper

Though deep CNNs were doing good job, *as people kept going deeper they realized that the performance is getting saturated*. Later studies showed that *much deeper CNNs start to perform worse!*

[6] Check it in this paper from which the figure is taken

A Hurdle in Going Deeper

Though deep CNNs were doing good job, *as people kept going deeper they realized that the performance is getting saturated*. Later studies showed that *much deeper CNNs start to perform worse!*

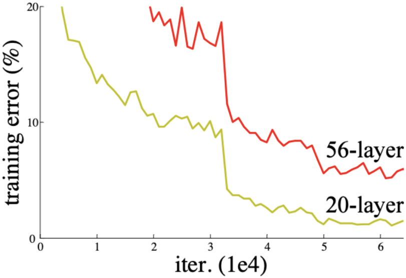
- *Initial guess for this behavior is overfitting*
  - ↳ *This was ruled out by Microsoft Research Lab by a study*[6]

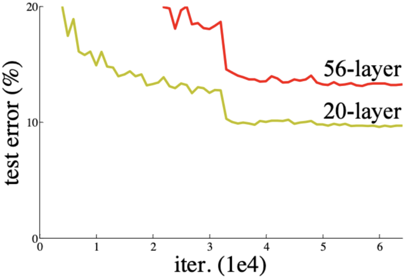
[6] Check it in this paper from which the figure is taken

A Hurdle in Going Deeper

Though deep CNNs were doing good job, *as people kept going deeper they realized that the performance is getting saturated.* Later studies showed that *much deeper CNNs start to perform worse!*

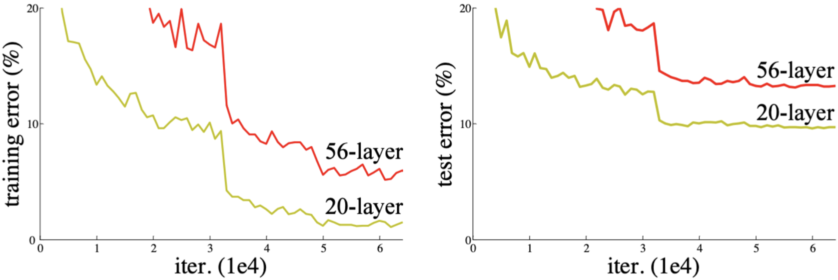
- *Initial guess for this behavior is overfitting*
  - ↳ *This was ruled out by Microsoft Research Lab by a study*[6]





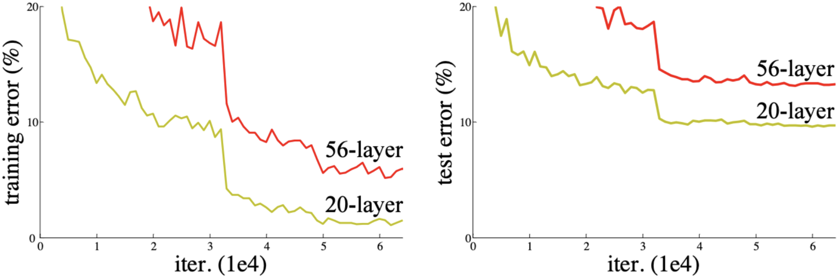
[6] Check it in this paper from which the figure is taken

A Hurdle in Going Deeper



+ *How do we see this conclusion in this figure?*

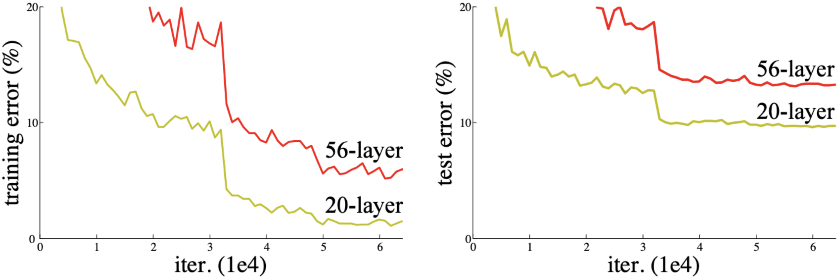
A Hurdle in Going Deeper



+ *How do we see this conclusion in this figure?*

– Well! *If it's coming from overfitting we should see better training and worst test risk. This is however not the case here!*

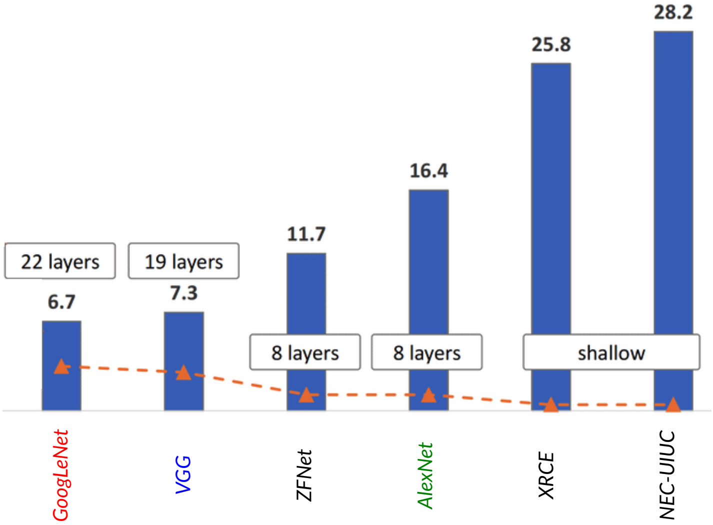
A Hurdle in Going Deeper



+ *How do we see this conclusion in this figure?*
– Well! *If it's coming from overfitting we should see better training and worst test risk. This is however not the case here!*

> *People started to blame the vanishing gradient behavior of deep NNs*

Depth vs Accuracy: *ILSVRC Winners till 2014*

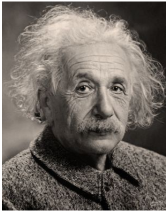


Problem of Depth

+ *But, what is the problem of vanishing gradient?*
- This is a general behavior in deep NNs: *as we go deeper, the gradients determined by backpropagation at initial layers get smaller and smaller, such that at some point they stop getting updated anymore, even though they should*
+ *How does it come?*

Problem of Depth

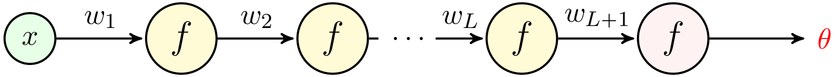
+ *But, what is the problem of vanishing gradient?*
– This is a general behavior in deep NNs: *as we go deeper, the gradients determined by backpropagation at initial layers get smaller and smaller, such that at some point they stop getting updated anymore, even though they should*
+ *How does it come?*
– Let's see it! But we follow Albert Einstein advice!



"Everything should be made as simple as possible, but not simpler!"

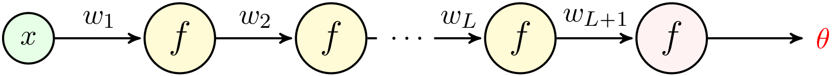
Vanishing Gradients: *Simple Example*

Consider the following dummy FNN: *an FNN has a single scalar input*, L *hidden layers and a single scalar output. All neurons are activated by function* $f(\cdot)$ *and have no bias*



Vanishing Gradients: *Simple Example*

Consider the following dummy FNN: *an FNN has a single scalar input*, L *hidden layers and a single scalar output. All neurons are activated by function* $f(\cdot)$ *and have no bias*



Let's write forward pass

- $y_1 = f(w_1 x)$
- $y_2 = f(w_2 y_1)$

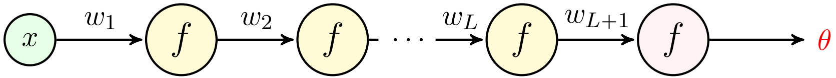
  $\cdots$
- $y_L = f(w_L y_{L-1})$
- $\theta = f(w_{L+1} y_L)$

Vanishing Gradients: *Simple Example*

$$x \xrightarrow{w_1} f \xrightarrow{w_2} f \cdots \xrightarrow{w_L} f \xrightarrow{w_{L+1}} f \longrightarrow \theta$$

Now, we go for backward *pass: we start with* $\overleftarrow{\theta} = \mathrm{d}\hat{R}/\mathrm{d}\theta$ *and go backward*

Vanishing Gradients: *Simple Example*

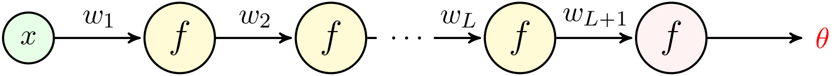


Now, we go for backward *pass: we start with* $\overleftarrow{\theta} = \mathrm{d}\hat{R}/\mathrm{d}\theta$ *and go backward*

- $\theta = f\left(w_{L+1} y_L\right)$

$$\overleftarrow{y}_L = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_L} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}\theta}\frac{\mathrm{d}\theta}{\mathrm{d}y_L}$$

Vanishing Gradients: *Simple Example*

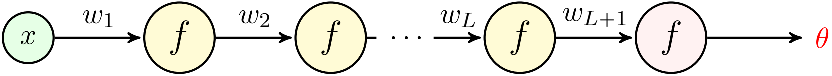


Now, we go for backward pass: we start with $\overleftarrow{\theta} = \mathrm{d}\hat{R}/\mathrm{d}\theta$ *and go backward*

- $\theta = f\left(w_{L+1} y_L\right)$

$$\overleftarrow{y}_L = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_L} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}\theta}\frac{\mathrm{d}\theta}{\mathrm{d}y_L} = \overleftarrow{\theta}\, w_{L+1} \dot{f}\left(w_{L+1} y_L\right)$$

Vanishing Gradients: *Simple Example*



Now, we go for backward *pass: we start with* $\overleftarrow{\theta} = \mathrm{d}\hat{R}/\mathrm{d}\theta$ *and go backward*

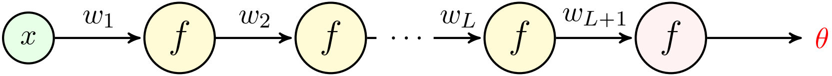
- $\theta = f\left(w_{L+1} y_L\right)$

$$\overleftarrow{y}_L = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_L} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}\theta}\frac{\mathrm{d}\theta}{\mathrm{d}y_L} = \overleftarrow{\theta}\, w_{L+1} \dot{f}\left(w_{L+1} y_L\right)$$

- $y_L = f\left(w_L y_{L-1}\right)$

$$\overleftarrow{y}_{L-1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_{L-1}} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_L}\frac{\mathrm{d}y_L}{\mathrm{d}y_L}$$

Vanishing Gradients: *Simple Example*



Now, we go for backward *pass: we start with* $\overleftarrow{\theta} = \mathrm{d}\hat{R}/\mathrm{d}\theta$ *and go backward*

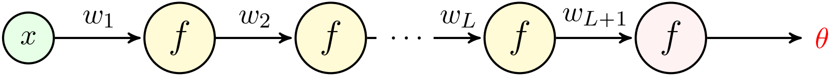
- $\theta = f\left(w_{L+1} y_L\right)$

$$\overleftarrow{y}_L = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_L} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}\theta}\frac{\mathrm{d}\theta}{\mathrm{d}y_L} = \overleftarrow{\theta}\, w_{L+1} \dot{f}\left(w_{L+1} y_L\right)$$

- $y_L = f\left(w_L y_{L-1}\right)$

$$\overleftarrow{y}_{L-1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_{L-1}} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_L}\frac{\mathrm{d}y_L}{\mathrm{d}y_L} = \overleftarrow{y}_L w_L \dot{f}\left(w_L y_{L-1}\right)$$

Vanishing Gradients: *Simple Example*



Now, we go for backward pass: we start with $\overleftarrow{\theta} = \mathrm{d}\hat{R}/\mathrm{d}\theta$ *and go backward*

- $\theta = f\left(w_{L+1} y_L\right)$

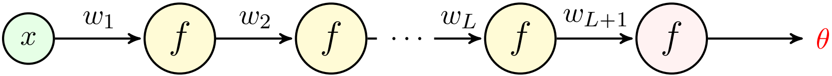
$$\overleftarrow{y}_L = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_L} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}\theta}\frac{\mathrm{d}\theta}{\mathrm{d}y_L} = \overleftarrow{\theta}\, w_{L+1} \dot{f}\left(w_{L+1} y_L\right)$$

- $y_L = f\left(w_L y_{L-1}\right)$

$$\overleftarrow{y}_{L-1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_{L-1}} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_L}\frac{\mathrm{d}y_L}{\mathrm{d}y_L} = \overleftarrow{y}_L w_L \dot{f}\left(w_L y_{L-1}\right)$$

$$= \overleftarrow{\theta}\, w_{L+1} w_L \dot{f}\left(w_L y_{L-1}\right) \dot{f}\left(w_{L+1} y_L\right)$$

Vanishing Gradients: *Simple Example*

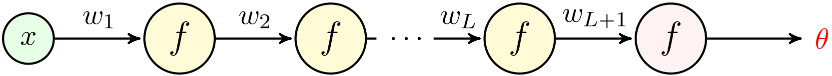


As we keep on going backward, the multiplication terms expand

- $y_2 = f\left(w_2 y_1\right)$

$$\overleftarrow{y}_1 = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_2}\frac{\mathrm{d}y_2}{\mathrm{d}y_1} = \overleftarrow{y_2} w_2 \dot{f}\left(w_2 y_1\right)$$

Vanishing Gradients: *Simple Example*



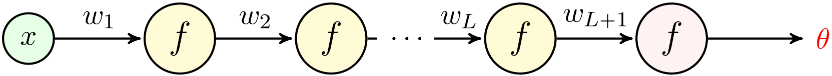
As we keep on going backward, the multiplication terms expand

- $y_2 = f\left(w_2 y_1\right)$

$$\overleftarrow{y}_1 = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_2}\frac{\mathrm{d}y_2}{\mathrm{d}y_1} = \overleftarrow{y_2} w_2 \dot{f}\left(w_2 y_1\right)$$

$$= \overleftarrow{\theta} \prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right)$$

Vanishing Gradients: *Simple Example*

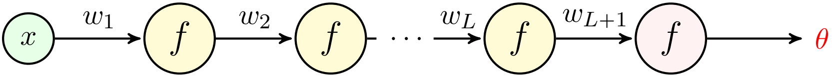


Now, let's compute derivative of loss with respect to the first weight w_1

- $y_1 = f(w_1 x)$

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_1}\frac{\mathrm{d}y_1}{\mathrm{d}w_1}$$

Vanishing Gradients: *Simple Example*

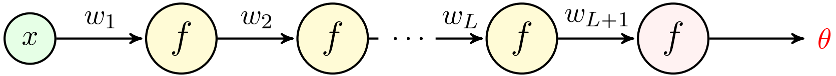


Now, let's compute derivative of loss with respect to the first weight w_1

- $y_1 = f(w_1 x)$

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_1}\frac{\mathrm{d}y_1}{\mathrm{d}w_1} = \overleftarrow{y_1} x \dot{f}(w_1 x)$$

Vanishing Gradients: *Simple Example*



Now, let's compute derivative of loss with respect to the first weight w_1

- $y_1 = f(w_1 x)$

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_1}\frac{\mathrm{d}y_1}{\mathrm{d}w_1} = \overleftarrow{y_1} x \dot{f}(w_1 x)$$

$$= \overleftarrow{\theta} x \dot{f}(w_1 x) \prod_{\ell=2}^{L+1} w_\ell \dot{f}(w_\ell y_{\ell-1})$$

Vanishing Gradients: *Simple Example*

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta} \underbrace{x\dot{f}\left(w_1 x\right) \prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right)}_{\text{accumulated in Backpropagation}}$$

Vanishing Gradients: *Simple Example*

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta} \underbrace{x\dot{f}\left(w_1 x\right) \prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right)}_{\text{accumulated in backpropagation}}$$

Now, consider the following cases

- ***Case I:*** *We have a sigmoid activation and all weight are smaller than 1*

Vanishing Gradients: *Simple Example*

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta} \underbrace{x \dot{f}\left(w_1 x\right) \prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right)}_{\text{accumulated in backpropagation}}$$

Now, consider the following cases

- ***Case I:*** *We have a sigmoid activation and all weight are smaller than 1*
  - ↳ *Note that for sigmoid* $\dot{f}\left(x\right) < 1$ *for any* x

Vanishing Gradients: *Simple Example*

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta} \underbrace{x \dot{f}(w_1 x) \prod_{\ell=2}^{L+1} w_\ell \dot{f}(w_\ell y_{\ell-1})}_{\text{accumulated in backpropagation}}$$

Now, consider the following cases

- ***Case I:*** *We have a sigmoid activation and all weight are smaller than 1*
  - ↳ *Note that for sigmoid* $\dot{f}(x) < 1$ *for any* x
  - ↳ *There is one number* $a < 1$ *that all weights and derivatives are smaller than, e.g.,* $a = 1 - 10^{-10}$

Vanishing Gradients: *Simple Example*

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta} \underbrace{x\dot{f}(w_1 x) \prod_{\ell=2}^{L+1} w_\ell \dot{f}(w_\ell y_{\ell-1})}_{\text{accumulated in backpropagation}}$$

Now, consider the following cases

- ***Case I:*** *We have a sigmoid activation and all weight are smaller than 1*
  - ↳ *Note that for sigmoid* $\dot{f}(x) < 1$ *for any* x
  - ↳ *There is one number* $a < 1$ *that all weights and derivatives are smaller than, e.g.,* $a = 1 - 10^{-10}$

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta} x\dot{f}(w_1 x) \prod_{\ell=2}^{L+1} w_\ell \dot{f}(w_\ell y_{\ell-1}) < \overleftarrow{\theta} x a^{2L+1}$$

Exploding Gradients: *Simple Example*

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta} \underbrace{x\dot{f}\left(w_1 x\right)\prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right)}_{\text{accumulated in backpropagation}}$$

Now, consider the following cases

- ***Case I:*** *We have a sigmoid activation and all weight are smaller than 1*

$$\lim_{L\uparrow\infty} \frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = 0$$

Exploding Gradients: *Simple Example*

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta} \underbrace{x\dot{f}(w_1 x) \prod_{\ell=2}^{L+1} w_\ell \dot{f}(w_\ell y_{\ell-1})}_{\text{accumulated in Backpropagation}}$$

Now, consider the following cases

- ***Case I:** We have a sigmoid activation and all weight are smaller than 1*

$$\lim_{L\uparrow\infty} \frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = 0$$

↳ *Backpropagation accumulation concentrates at zero!*

Gradient with respect to first layer vanishes as the network gets too deep

Exploding Gradients: *Simple Example*

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta} \underbrace{x\dot{f}\left(w_1 x\right) \prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right)}_{\text{accumulated in backpropagation}}$$

Now, consider the following cases

- ***Case II:*** *We have a ReLU activation and all weight are larger than 1*
  - ↳ *Assume $x > 0$; then, $\dot{f}\left(w_\ell y_{\ell-1}\right) = 1$ since all the sequence is positive*

Exploding Gradients: *Simple Example*

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta} \underbrace{x\dot{f}\left(w_1 x\right) \prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right)}_{\text{accumulated in backpropagation}}$$

Now, consider the following cases

- ***Case II:*** *We have a ReLU activation and all weight are larger than 1*
  - ↳ *Assume $x > 0$; then, $\dot{f}\left(w_\ell y_{\ell-1}\right) = 1$ since all the sequence is positive*
  - ↳ *There is one number $a > 1$ that all weights are larger than, e.g., $a = 1 + 10^{-10}$*

Exploding Gradients: *Simple Example*

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta} \underbrace{x\dot{f}\left(w_1 x\right) \prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right)}_{\text{accumulated in Backpropagation}}$$

Now, consider the following cases

- ***Case II:** We have a ReLU activation and all weight are larger than 1*
  - ↳ *Assume $x > 0$; then, $\dot{f}\left(w_\ell y_{\ell-1}\right) = 1$ since all the sequence is positive*
  - ↳ *There is one number $a > 1$ that all weights are larger than, e.g., $a = 1 + 10^{-10}$*

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta}\, x\dot{f}\left(w_1 x\right) \prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right) > \overleftarrow{\theta}\, x a^L$$

Vanishing Gradients: *Simple Example*

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta} \underbrace{x\dot{f}\left(w_1 x\right)\prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right)}_{\text{accumulated in Backpropagation}}$$

Now, consider the following cases

- ***Case I:*** *We have a ReLU activation and all weight are larger than 1*

$$\lim_{L\uparrow\infty} \frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} \to \infty$$

↳ *Backpropagation accumulation explodes!*

Gradient with respect to first layer explodes as the network gets too deep

Exploding-Vanishing Gradients: *Summary*

Moral of Story

As the network gets very deep, the gradients of initial layers can get extremely small or large

- *The vanishing occurs more frequently*
  - ↳ *Weights are often adjusted by optimizer to get small*
  - ↳ *Once they all get small, the gradient starts to vanish*
- *Weights and derivative of activation function are key deciders*
  - ↳ *We need them to mostly around 1*

Exploding-Vanishing Gradients: *Summary*

Moral of Story

As the network gets very deep, the gradients of initial layers can get extremely small or large

- *The vanishing occurs more frequently*
  - ↳ *Weights are often adjusted by optimizer to get small*
  - ↳ *Once they all get small, the gradient starts to vanish*
- *Weights and derivative of activation function are key deciders*
  - ↳ *We need them to mostly around 1*

The above observation also explains *why we had some specific preferences*

- We preferred ReLU activation in hidden layers
  - ↳ $\dot{\mathrm{ReLU}}(x) = 1$ *when* $x > 0$
- We preferred normalized features
  - ↳ *this can keep weights normalized*

Exploding-Vanishing Gradients: *Solution*

+ *But, is there any solution for that?*
– Yes! Actually, we already had some one them!

Exploding-Vanishing Gradients: *Solution*

+ *But, is there any solution for that?*
– Yes! Actually, we already had some one them!

In practice, we can use different approaches to control this behavior

- We use better activations in deep NNs
  ↳ *this is why in deep CNNs we use mostly ReLU*
- We apply batch-normalization
  ↳ *we already have discussed it!*
- We use skip connection
  ↳ *this helps us go even deeper!*

Let's understand what skip connection is!

Skip Connection: *Residual Learning*

The root idea of skip connection is as follows: *instead of learning a function*

we can learn its residual and add it up with the input

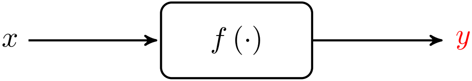
Skip Connection: *Residual Learning*

The root idea of skip connection is as follows: *instead of learning a function*

we can learn its residual and add it up with the input

Let's say we have input x and label y

With plain NNs we learn a function $f(\cdot)$ that relates x to y

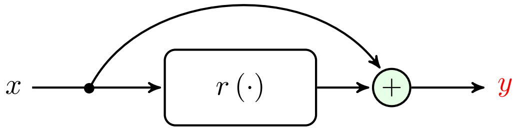


The NN approximates $f(\cdot)$ as best as it could

Skip Connection: *Residual Learning*

Let's say we have input x and label y

But, we can also learn a function $r(\cdot)$ that relates x to $y - x$: the end-to-end function is then given by

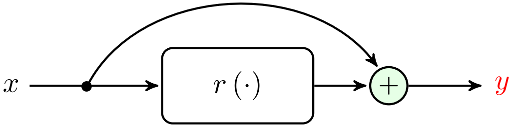


The NN now approximates the residual $r(\cdot)$ as best as it could

Skip Connection: *Residual Learning*

Let's say we have input x and label y

But, we can also learn a function $r(\cdot)$ that relates x to $y - x$: the end-to-end function is then given by

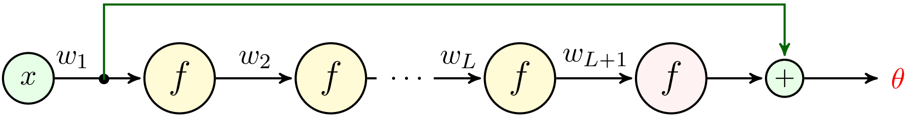


The NN now approximates the residual $r(\cdot)$ as best as it could

+ *But, why should it be any different this time?*
– Let's get back to our simple example

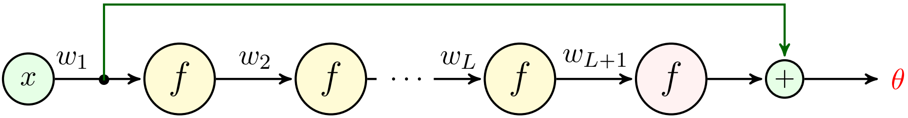
Skip Connection: *Simple Example*

Let's get back to our simple example: *this time we consider skip connection*



Skip Connection: *Simple Example*

Let's get back to our simple example: *this time we consider skip connection*

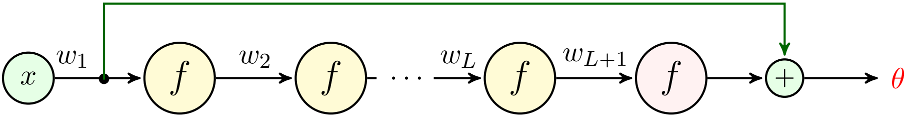


Let's see forward pass

- $y_1 = f(w_1 x)$
- $y_2 = f(w_2 y_1)$

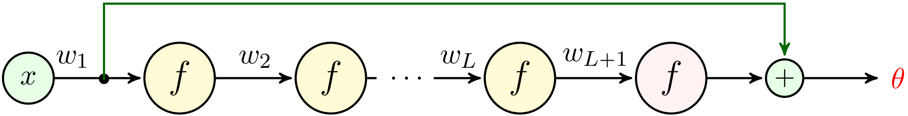
  $\cdots$
- $y_L = f(w_L y_{L-1})$
- $y_{L+1} = f(w_{L+1} y_L)$
- $\theta = y_{L+1} + w_1 x$

Skip Connection: *Simple Example*



What happens backward pass? We start with $\overleftarrow{\theta} = \mathrm{d}\hat{R}/\mathrm{d}\theta$ *again*

Skip Connection: *Simple Example*

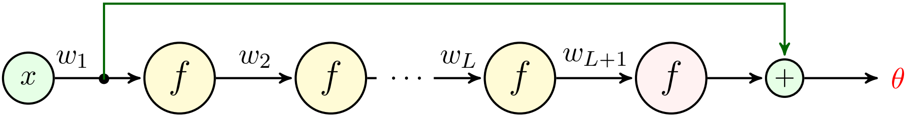


What happens backward pass? We start with $\overleftarrow{\theta} = \mathrm{d}\hat{R}/\mathrm{d}\theta$ *again*

- $\theta = y_{L+1} + w_1 x$

$$\overleftarrow{y}_{L+1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_{L+1}} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}\theta}\frac{\mathrm{d}\theta}{\mathrm{d}y_{L+1}} = \overleftarrow{\theta}$$

Skip Connection: *Simple Example*



What happens backward *pass? We start with* $\overleftarrow{\theta} = \mathrm{d}\hat{R}/\mathrm{d}\theta$ *again*

- $\theta = y_{L+1} + w_1 x$

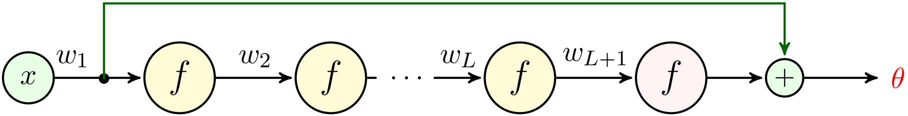
$$\overleftarrow{y}_{L+1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_{L+1}} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}\theta}\frac{\mathrm{d}\theta}{\mathrm{d}y_{L+1}} = \overleftarrow{\theta}$$

- $y_{L+1} = f\left(w_{L+1} y_L\right)$

$$\overleftarrow{y}_L = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_L} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_{L+1}}\frac{\mathrm{d}y_{L+1}}{\mathrm{d}y_L} = \overleftarrow{y}_{L+1} w_{L+1} \dot{f}\left(w_{L+1} y_L\right)$$

$$= \overleftarrow{\theta}\, w_{L+1} \dot{f}\left(w_{L+1} y_L\right)$$

Skip Connection: *Simple Example*



What happens backward pass? We start with $\overleftarrow{\theta} = \mathrm{d}\hat{R}/\mathrm{d}\theta$ *again*

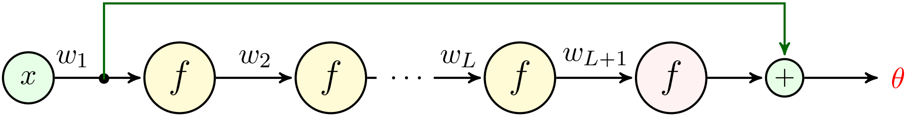
- $y_L = f(w_L y_{L-1})$

$$\overleftarrow{y}_{L-1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_{L-1}} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_L}\frac{\mathrm{d}y_L}{\mathrm{d}y_L} = \overleftarrow{y}_L w_L \dot{f}(w_L y_{L-1})$$

$$= \overleftarrow{\theta} w_{L+1} w_L \dot{f}(w_L y_{L-1}) \dot{f}(w_{L+1} y_L)$$

- $\ldots$

Skip Connection: *Simple Example*



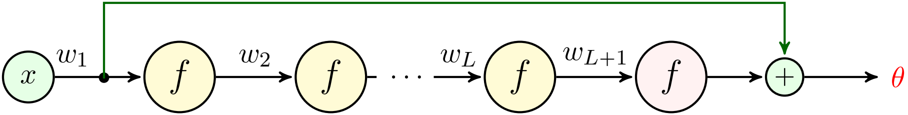
The backward pass looks the same up to the skip connection

- $y_2 = f\left(w_2 y_1\right)$

$$\overleftarrow{y}_1 = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_2}\frac{\mathrm{d}y_2}{\mathrm{d}y_1} = \overleftarrow{y_2} w_2 \dot{f}\left(w_2 y_1\right)$$

$$= \overleftarrow{\theta} \prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right)$$

Skip Connection: *Simple Example*

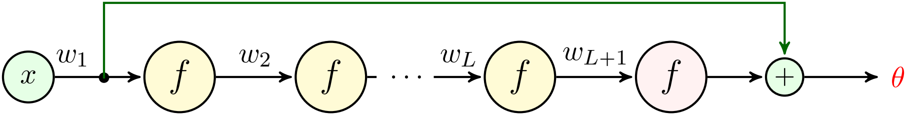


Now, let's compute derivative of loss with respect to the first weight w_1

- *Here, we have two links connected to* w_1 *in the network*
  - ↳ $y_1 = f(w_1 x)$
  - ↳ $\theta = y_{L+1} + w_1 x$

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1}$$

Skip Connection: *Simple Example*

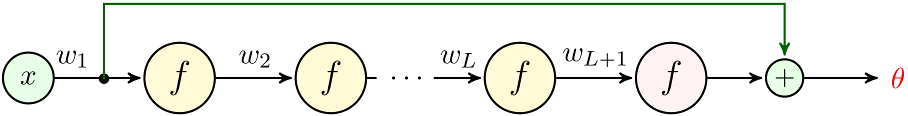


Now, let's compute derivative of loss with respect to the first weight w_1

- *Here, we have two links connected to* w_1 *in the network*
  - ↳ $y_1 = f(w_1 x)$
  - ↳ $\theta = y_{L+1} + w_1 x$

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_1}\frac{\mathrm{d}y_1}{\mathrm{d}w_1} + \frac{\mathrm{d}\hat{R}}{\mathrm{d}\theta}\frac{\mathrm{d}\theta}{\mathrm{d}w_1}$$

Skip Connection: *Simple Example*



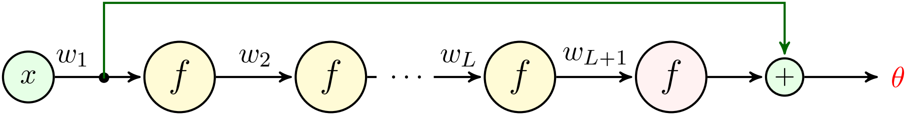
Now, let's compute derivative of loss with respect to the first weight w_1

- *Here, we have two links connected to w_1 in the network*
  - ↳ $y_1 = f(w_1 x)$
  - ↳ $\theta = y_{L+1} + w_1 x$

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_1}\frac{\mathrm{d}y_1}{\mathrm{d}w_1} + \frac{\mathrm{d}\hat{R}}{\mathrm{d}\theta}\frac{\mathrm{d}\theta}{\mathrm{d}w_1}$$

$$= \overleftarrow{y_1} x \dot{f}(w_1 x) + \overleftarrow{\theta} x$$

Skip Connection: *Simple Example*



Now, let's compute derivative of loss with respect to the first weight w_1

- *Here, we have two links connected to* w_1 *in the network*
  - ↳ $y_1 = f(w_1 x)$
  - ↳ $\theta = y_{L+1} + w_1 x$

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_1}\frac{\mathrm{d}y_1}{\mathrm{d}w_1} + \frac{\mathrm{d}\hat{R}}{\mathrm{d}\theta}\frac{\mathrm{d}\theta}{\mathrm{d}w_1}$$

$$= \overleftarrow{y_1} x \dot{f}(w_1 x) + \overleftarrow{\theta}\, x = \overleftarrow{\theta}\, x \left(\dot{f}(w_1 x) \prod_{\ell=2}^{L+1} w_\ell \dot{f}(w_\ell y_{\ell-1}) + 1 \right)$$

Skip Connection: *Simple Example*

With skip connection, the derivative of loss with respect to the first weight w_1 does not vanish

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta}\, x \left(\underbrace{\dot{f}\left(w_1 x\right) \prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right)}_{\text{accumulated in backpropagation}} + 1 \rightarrow \text{skip connection} \right)$$

Even if all weights and derivatives are smaller than one, as we get very deep $\overleftarrow{\theta}$ is still backpropagating

Skip Connection: *General Form*

Skip Connection

Skip connection refers to links that carry information from layer $\ell - s$ to layer ℓ for $s > 1$

Skip Connection: *General Form*

Skip Connection

Skip connection refers to links that carry information from layer $\ell - s$ to layer ℓ for $s > 1$

Let $\mathbf{Z}_\ell$ and $\mathbf{Y}_\ell = f(\mathbf{Z}_\ell)$ be outputs of layer ℓ before and after activation

- *This layer can be a convolution or fully-connected layer*
  - ↳ *If convolution, $\mathbf{Z}_\ell$ and $\mathbf{Y}_\ell$ are tensors*
  - ↳ *If fully-connected, $\mathbf{Z}_\ell$ and $\mathbf{Y}_\ell$ are vectors*

Skip Connection: *General Form*

Skip Connection

Skip connection refers to links that carry information from layer $\ell - s$ to layer ℓ for $s > 1$

Let $\mathbf{Z}_\ell$ and $\mathbf{Y}_\ell = f(\mathbf{Z}_\ell)$ be outputs of layer ℓ before and after activation

- *This layer can be a convolution or fully-connected layer*
  - ↳ *If convolution, $\mathbf{Z}_\ell$ and $\mathbf{Y}_\ell$ are tensors*
  - ↳ *If fully-connected, $\mathbf{Z}_\ell$ and $\mathbf{Y}_\ell$ are vectors*

With a skip connection of depth s, the output of layer ℓ is

$$\mathbf{Y}_\ell = f(\mathbf{Z}_\ell + \mathbf{W} \circ \mathbf{Y}_{\ell-s})$$

↳ $\mathbf{W}$ *is a linear transform that can be potentially learned*

↳ $\circ$ *is a kind of product that matches the dimensions*

Skip Connection: *General Form*

+ *What is exactly $\mathbf{W}$?*
– It is a set of weights, like other weighted components. But, *we don't really need it. We could set it to some fixed values and don't learn it at all*
+ *Why should we connect activated output to linear output?*
– There is actually no should here also!

Skip Connection: *General Form*

+ *What is exactly $\mathbf{W}$?*
– It is a set of weights, like other weighted components. But, *we don't really need it. We could set it to some fixed values and don't learn it at all*
+ *Why should we connect activated output to linear output?*
– There is actually no should here also!

In original proposal it was suggested to connect *activated output of a layer to the linear output of some next layers, i.e., add* $\mathbf{Y}_{\ell-s}$ to $\mathbf{Z}_{\ell}$

↳ *This combination is only a suggestion!*

↳ *Later suggestions proposed to connect linear output* $\mathbf{Z}_{\ell-s}$

↳ *Similar to batch normalization, best combination is found by experiment*

Residual Unit: *New Building Block via Skip Connection*

Skip connection allowed for training deeper NNs: *experiments showed that*

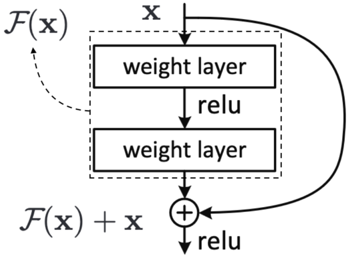
- *Using skip connection with batch normalization make results sensible*
  ↳ *Deeper networks show better training performance*
- *Skip connection seems to be a crucial component for going deep*

Residual Unit: *New Building Block via Skip Connection*

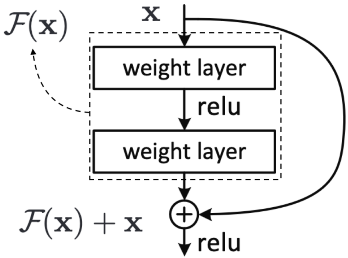
Skip connection allowed for training deeper NNs: *experiments showed that*

- *Using skip connection with batch normalization make results sensible*
  - ↳ *Deeper networks show better training performance*
- *Skip connection seems to be a crucial component for going deep*

These results led to introduction of a new building block called *residual unit*



Residual Unit

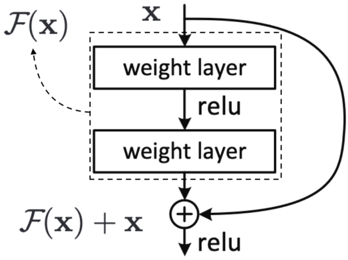


Residual Unit consists of

- *multiple (typically 2) weighted layers, e.g.,*
  - ↳ *convolutional layer*
  - ↳ *standard fully-connected layer*

The idea was proposed by Microsoft to win ILSVRC in this paper and then expanded in this paper

Residual Unit



Residual Unit consists of

- *multiple (typically 2) weighted layers, e.g.,*
  - ↳ *convolutional layer*
  - ↳ *standard fully-connected layer*
- *a skip connection that adds input of the unit to the output*
  - ↳ *generally a weighted connection*

$$\text{output} = \mathrm{ReLU}(\mathcal{F}(\mathbf{x}) + \mathbf{W}_{\mathrm{s}}\mathbf{x})$$

  - ↳ *experiments show that when* $\mathcal{F}(\mathbf{x})$ *and* $\mathbf{x}$ *are of same dimension* $\mathbf{W}_{\mathrm{s}} = \mathbf{I}$ *is a good choice*

The idea was proposed by Microsoft to win ILSVRC in this paper and then expanded in this paper

Forward Propagation through Residual Units

In original proposal, Residual Units are implemented by convolutional layers

1. *Input* $\mathbf{x} = \mathbf{X}$ *is a tensor with* C *channels*

Forward Propagation through Residual Units

In original proposal, Residual Units are implemented by convolutional layers

1. *Input* $\mathbf{x} = \mathbf{X}$ *is a tensor with* C *channels*
2. $\mathbf{X}$ *is given to a convolutional layer with* C *output channels, i.e.,* C *filters*

$$\mathbf{Z}_1 = \mathrm{Conv}\left(\mathbf{X} | \mathbf{W}_1^1, \ldots, \mathbf{W}_C^1\right)$$

Forward Propagation through Residual Units

In original proposal, Residual Units are implemented by convolutional layers

1. *Input* $\mathbf{x} = \mathbf{X}$ *is a tensor with* C *channels*
2. $\mathbf{X}$ *is given to a convolutional layer with* C *output channels, i.e.,* C *filters*

$$\mathbf{Z}_1 = \mathrm{Conv}\left(\mathbf{X}|\mathbf{W}_1^1, \ldots, \mathbf{W}_C^1\right)$$

3. $\mathbf{Z}_1$ *is then activated* $\mathbf{Y}_1 = f\left(\mathbf{Z}_1\right)$

Forward Propagation through Residual Units

In original proposal, Residual Units are implemented by convolutional layers

1. *Input* $\mathbf{x} = \mathbf{X}$ *is a tensor with* C *channels*
2. $\mathbf{X}$ *is given to a convolutional layer with* C *output channels, i.e.,* C *filters*

$$\mathbf{Z}_1 = \mathrm{Conv}\left(\mathbf{X}|\mathbf{W}_1^1, \ldots, \mathbf{W}_C^1\right)$$

3. $\mathbf{Z}_1$ *is then activated* $\mathbf{Y}_1 = f\left(\mathbf{Z}_1\right)$
4. $\mathbf{Y}_1$ *is given to another convolutional layer with* C *output channels*

$$\mathbf{Z}_2 = \mathrm{Conv}\left(\mathbf{Y}_1|\mathbf{W}_1^2, \ldots, \mathbf{W}_C^2\right)$$

Forward Propagation through Residual Units

In original proposal, Residual Units are implemented by convolutional layers

1. *Input* $\mathbf{x} = \mathbf{X}$ *is a tensor with* C *channels*
2. $\mathbf{X}$ *is given to a convolutional layer with* C *output channels, i.e.,* C *filters*

$$\mathbf{Z}_1 = \mathrm{Conv}\left(\mathbf{X} | \mathbf{W}_1^1, \dots, \mathbf{W}_C^1\right)$$

3. $\mathbf{Z}_1$ *is then activated* $\mathbf{Y}_1 = f\left(\mathbf{Z}_1\right)$
4. $\mathbf{Y}_1$ *is given to another convolutional layer with* C *output channels*

$$\mathbf{Z}_2 = \mathrm{Conv}\left(\mathbf{Y}_1 | \mathbf{W}_1^2, \dots, \mathbf{W}_C^2\right)$$

5. *Output* $\mathbf{Y}$ *is constructed by activating* $\mathbf{Z}_2$ *after skip connection*

$$\mathbf{Y} = f\left(\mathbf{Z}_2 + \mathbf{X}\right)$$

Backpropagation through Residual Units

Assume we have $\nabla_{\mathbf{Y}} \hat{R}$

1. $\nabla_{\mathbf{Z}_2} \hat{R}$ *is computed by entry-wise production with* $\dot{f}(\mathbf{Z}_2 + \mathbf{X})$

Backpropagation through Residual Units

Assume we have $\nabla_{\mathbf{Y}} \hat{R}$

1. $\nabla_{\mathbf{Z}_2} \hat{R}$ *is computed by entry-wise production with* $\dot{f}\left(\mathbf{Z}_2 + \mathbf{X}\right)$
2. $\nabla_{\mathbf{Y}_1} \hat{R}$ *is computed by convolution*

$$\nabla_{\mathbf{Y}_1} \hat{R} = \mathrm{Conv}\left(\nabla_{\mathbf{Z}_2} \hat{R} | \mathbf{W}_1^{2\dagger}, \ldots, \mathbf{W}_C^{2\dagger}\right)$$

Backpropagation through Residual Units

Assume we have $\nabla_{\mathbf{Y}} \hat{R}$

1. $\nabla_{\mathbf{Z}_2} \hat{R}$ *is computed by entry-wise production with* $\dot{f}\left(\mathbf{Z}_2 + \mathbf{X}\right)$
2. $\nabla_{\mathbf{Y}_1} \hat{R}$ *is computed by convolution*

$$\nabla_{\mathbf{Y}_1} \hat{R} = \mathrm{Conv}\left(\nabla_{\mathbf{Z}_2} \hat{R} | \mathbf{W}_1^{2\dagger}, \ldots, \mathbf{W}_C^{2\dagger}\right)$$

3. $\nabla_{\mathbf{Z}_1} \hat{R}$ *is computed by entry-wise production with* $\dot{f}\left(\mathbf{Z}_1\right)$

Backpropagation through Residual Units

Assume we have $\nabla_{\mathbf{Y}} \hat{R}$

1. $\nabla_{\mathbf{Z}_2} \hat{R}$ *is computed by entry-wise production with* $\dot{f}\left(\mathbf{Z}_2 + \mathbf{X}\right)$
2. $\nabla_{\mathbf{Y}_1} \hat{R}$ *is computed by convolution*

$$\nabla_{\mathbf{Y}_1} \hat{R} = \operatorname{Conv}\left(\nabla_{\mathbf{Z}_2} \hat{R} | \mathbf{W}_1^{2\dagger}, \ldots, \mathbf{W}_C^{2\dagger}\right)$$

3. $\nabla_{\mathbf{Z}_1} \hat{R}$ *is computed by entry-wise production with* $\dot{f}\left(\mathbf{Z}_1\right)$
4. $\nabla_{\mathbf{X}} \hat{R}$ *is given by a new chain rule*

$$\begin{aligned}\nabla_{\mathbf{X}} \hat{R} &= \underbrace{\nabla_{\mathbf{Z}_1} \hat{R} \circ \nabla_{\mathbf{X}} \mathbf{Z}_1}_{\text{Backward convolution}} + \nabla_{\mathbf{Y}} \hat{R} \underbrace{\circ}_{\odot} \underbrace{\nabla_{\mathbf{X}} \mathbf{Y}}_{\dot{f}(\mathbf{Z}_2 + \mathbf{X})} \\ &= \operatorname{Conv}\left(\nabla_{\mathbf{Z}_1} \hat{R} | \mathbf{W}_1^{1\dagger}, \ldots, \mathbf{W}_C^{1\dagger}\right) \underbrace{+ \nabla_{\mathbf{Y}} \hat{R} \odot \dot{f}\left(\mathbf{Z}_2 + \mathbf{X}\right)}_{\text{avoids vanishing gradients}}\end{aligned}$$

Residual Networks

We can now look at *Residual Unit* as a single block in a deep NN

- *We build an architecture by cascading them*
  - ↳ *Intuitively we can go deeper now, since we use skip connection*
- *We also add other kinds of layers that we know, e.g.,*
  - ↳ *convolutional layers, pooling layers, fully-connected layers*
- *We can do whatever we have done before to train them efficiently, e.g.,*
  - ↳ *dropout or other regularization techniques*
  - ↳ *input and batch normalization*
  - ↳ *more advanced optimizers*

Residual Networks

We can now look at *Residual Unit* as a single block in a deep NN

- *We build an architecture by cascading them*
  - ↳ *Intuitively we can go deeper now, since we use skip connection*
- *We also add other kinds of layers that we know, e.g.,*
  - ↳ *convolutional layers, pooling layers, fully-connected layers*
- *We can do whatever we have done before to train them efficiently, e.g.,*
  - ↳ *dropout or other regularization techniques*
  - ↳ *input and batch normalization*
  - ↳ *more advanced optimizers*

These kinds of NNs are called Residual Networks or shortly ResNet

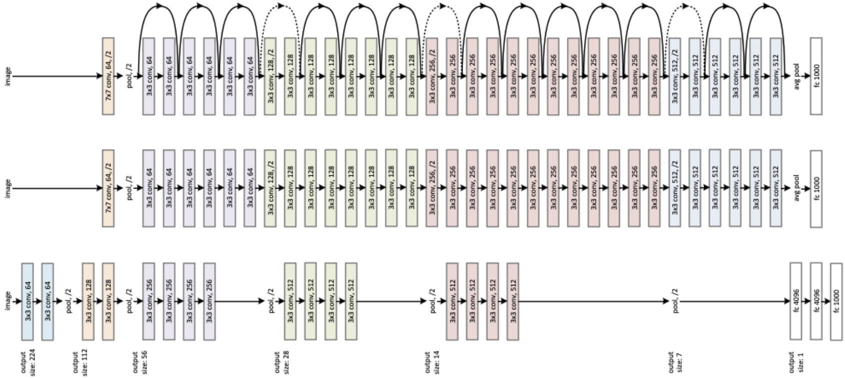
↳ *they were proposed in the original paper introduced Residual Unit*

↳ *they are nowadays a kind of benchmark in many applications*

Residual Networks: *A Nice Experiment*

ResNet inventors did a nice experiment to show the impact of skip connection[7]

- *They investigated 3 architectures: 34-layer ResNet, 34-layer Plain CNN (no skip connection) and benchmark VGG architectures*



[7] Find the details in their paper

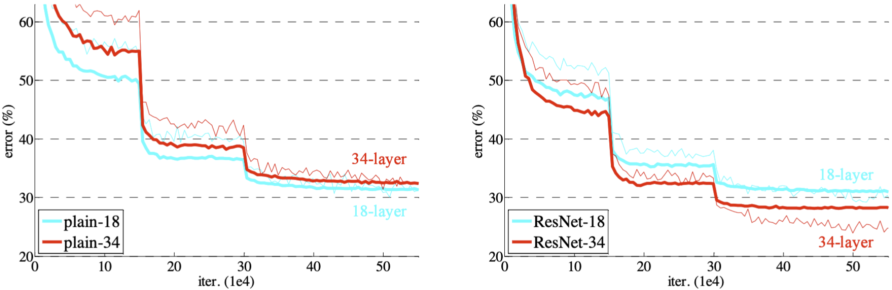
Residual Networks: *A Nice Experiment*

The results was interesting: *the depth challenge is now efficiently addressed, i.e., by going deep we always get better in performance*

model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40

Residual Networks: *A Nice Experiment*

Recall that *the depth challenge was different from overfitting! With deeper architectures, plain CNNs are showing worse "training" performance. With ResNet, this is not the case anymore*



> *The above figures show training error meaning that*
>
> *the left figure cannot simply show overfitting!*

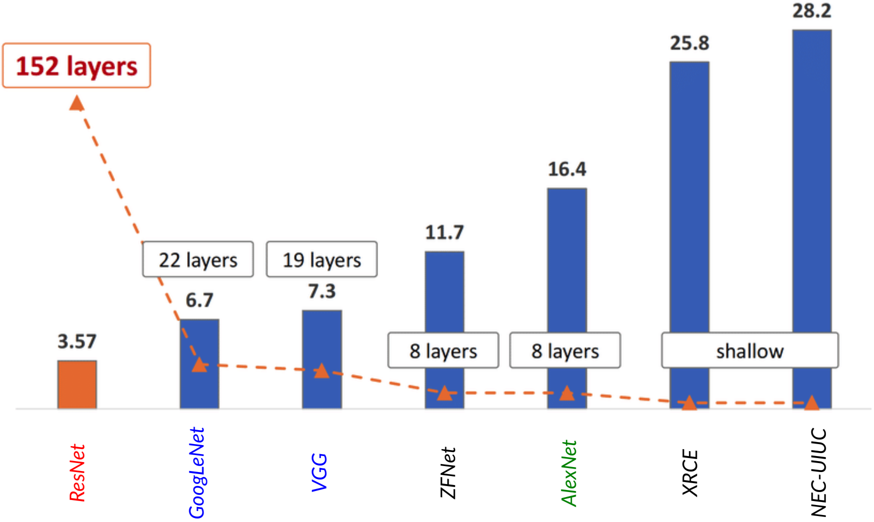
Residual Networks: *ILSVRC*

Getting rid of this undesired behavior *made it easy to go as deep as we want*

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

This way ResNet won ILSVRC in 2015!

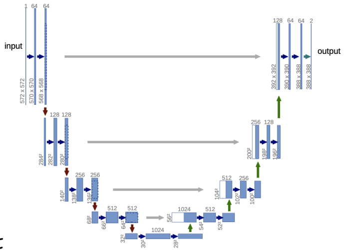
Depth vs Accuracy: *ILSVRC Winners till 2015*



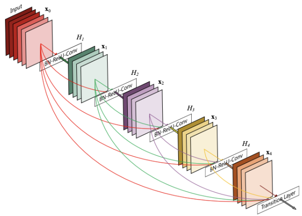
Other Architectures with Skip Connection

ResNet is not the only architecture with skip connection

- *ResNet made a breakthrough by using short skip connection*
- *U-Net uses long and short skip connections*[8]
- *DenseNet uses long and short skip connections densely to go even deeper*[9]



U-Net



DenseNet

[8] Check the original proposal of U-Net here

[9] Check the original proposal of DenseNet here

Notes on Implementation

Many architectures have been already implemented and pre-trained in PyTorch

- *They are implemented using basic blocks in PyTorch*
- *They are pre-trained for ImageNet classification*
  - ↳ *They get 3-channel 224×224 tensor as input*
  - ↳ *They return 1000-dimensional output vector*

We can access them either through `torchvision.models`

```
import torchvision.models as Models
model = Models.
```

```
resnet
ResNet
resnet101
ResNet101_Weights
resnet152
ResNet152_Weights
resnet18
ResNet18_Weights
resnet34
ResNet34_Weights
```

Notes on Implementation

For instance, we could load ResNet-50 with its pre-trained weights

```
from torchvision.models import resnet50, ResNet50_Weights

# Old weights with accuracy 76.130%
resnet50(weights=ResNet50_Weights.IMAGENET1K_V1)

# New weights with accuracy 80.858%
resnet50(weights=ResNet50_Weights.IMAGENET1K_V2)

# Best available weights (currently alias for IMAGENET1K_V2)
# Note that these weights may change across versions
resnet50(weights=ResNet50_Weights.DEFAULT)

# Strings are also supported
resnet50(weights="IMAGENET1K_V2")

# No weights - random initialization
resnet50(weights=None)
```

We could alternatively use `torch.hub` module to load pre-trained models

Notes on Implementation

+ *What is we are using it for other applications with different data size?*
– We could add or modify layer to it; *for instance,*

> *Say we want to use a pre-trained* ResNet *to classify MNIST: we could replace the first convolutional layer with single-channel 28×28 input and the same number of output channels. We further replace the output layer with a fully-connected layer with 10 classes*

+ *But it does not perform well! Does it?!*
– Of course not! It has been trained for ImageNet and there is no reason to work with MNIST! But, *we can start from those weights and do normal training for several epochs*
  ↳ *It probably gets trained much faster as compared to random initialization*
  ↳ *This idea is studied in a much broader sense in Transfer Learning*