ECE 1508: Applied Deep Learning

Chapter 5: Skip Connection and Residual Networks

Ali Bereyhi

ali.bereyhi@utoronto.ca

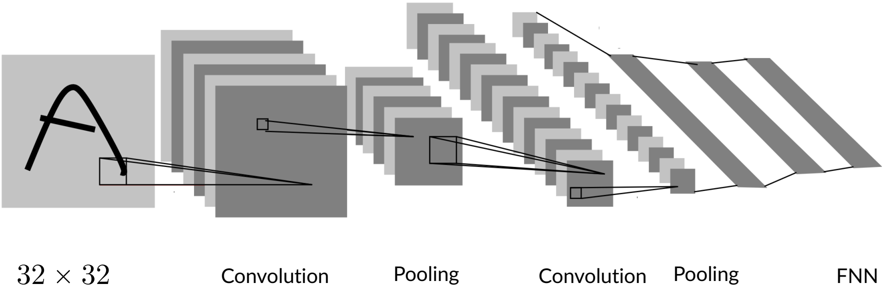
Department of Electrical and Computer Engineering

University of Toronto

Winter 2025

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: *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*

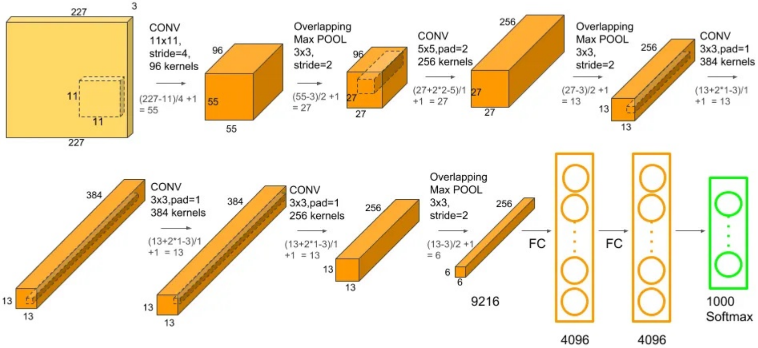
The winners in 2010 and 2011 used shallow NNs!

- *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*

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]

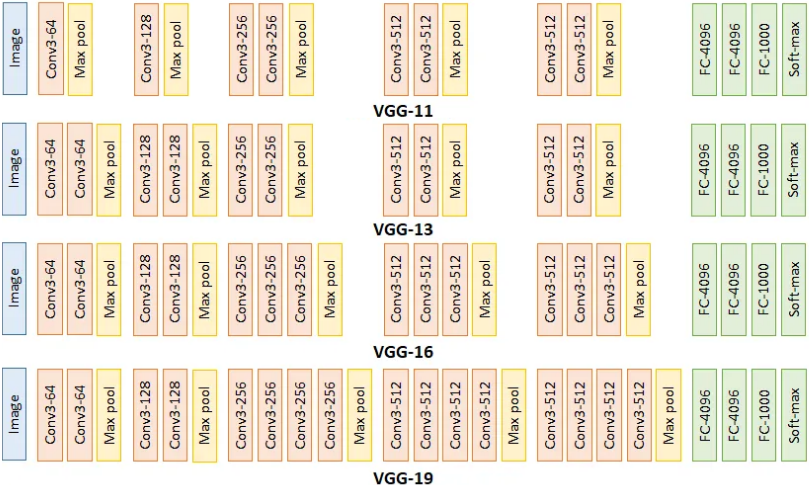
- *VGG-11*
- *VGG-13*
- *VGG-16*
- *VGG-19*

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



Error

10.4 %

9.9%

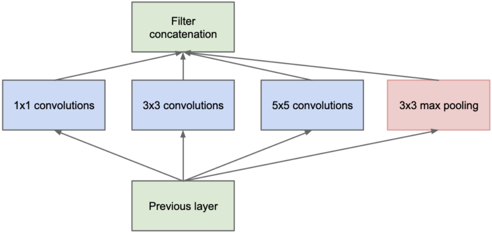
8.1%

8.0%

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*

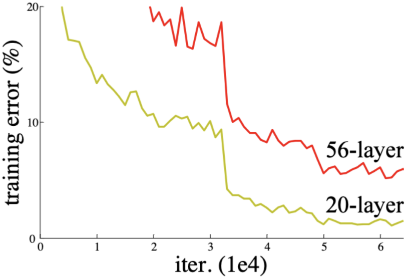
GoogLeNet won ILSVRC classification task

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

A Hurdle in Going Deeper

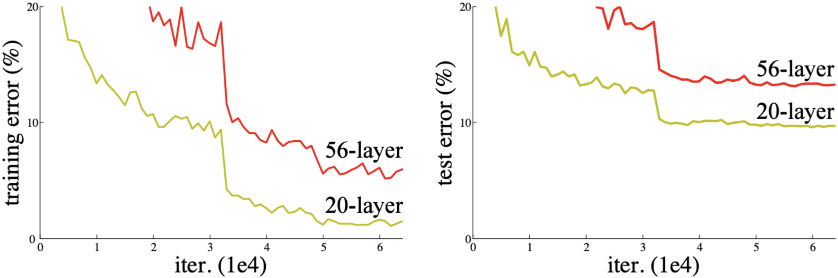
Though deep CNNs were doing good job, *as people kept going deeper they realized that the performance is getting saturated.* Later studies showed that *much deeper CNNs start to perform worse!*

- *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

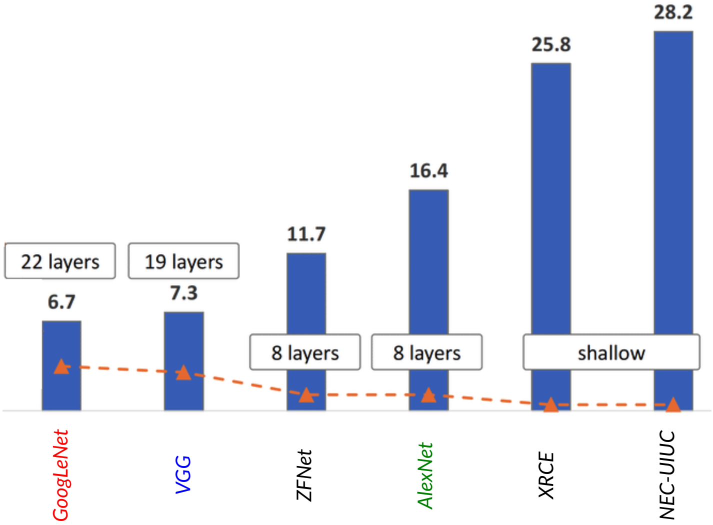
A Hurdle in Going Deeper



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

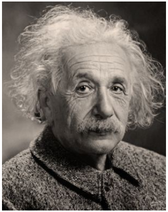
People started to blame the vanishing gradient behavior of deep NNs

Depth vs Accuracy: *ILSVRC Winners till 2014*



Problem of Depth

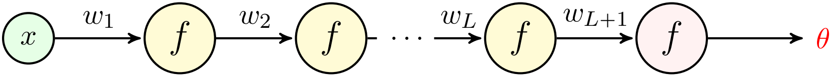
+ *But, what is the problem of vanishing gradient?*
– This is a general behavior in deep NNs: *as we go deeper, the gradients determined by backpropagation at initial layers get smaller and smaller, such that at some point they stop getting updated anymore, even though they should*
+ *How does it come?*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

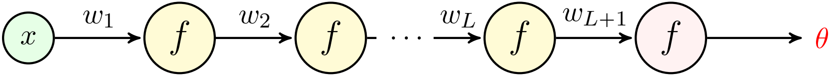


Let's write forward pass

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

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

Vanishing Gradients: *Simple Example*



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

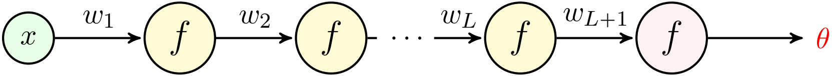
- $\theta = f\left(w_{L+1} y_L\right)$

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

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

$$\begin{aligned}\overleftarrow{y}_{L-1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_{L-1}} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_L}\frac{\mathrm{d}y_L}{\mathrm{d}y_L} &= \overleftarrow{y}_L w_L \dot{f}\left(w_L y_{L-1}\right) \\ &= \overleftarrow{\theta}\, w_{L+1} w_L \dot{f}\left(w_L y_{L-1}\right) \dot{f}\left(w_{L+1} y_L\right)\end{aligned}$$

Vanishing Gradients: *Simple Example*



As we keep on going backward, the multiplication terms expand

- $y_2 = f(w_2 y_1)$

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

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

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, 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*

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

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*

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

↳ *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*
  - ↳ *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*

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

Now, consider the following cases

- ***Case I:*** *We have a 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*

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!

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!