Deep learning

6.5. Residual networks

François Fleuret
https://fleuret.org/ee559/
Nov 2, 2020
The “Highway networks” by Srivastava et al. (2015) use the idea of gating developed for recurrent units. It replaces a standard non-linear layer

\[ y = H(x; W_H) \]

with a layer that includes a “gated” pass-through

\[ y = T(x; W_T)H(x; W_H) + (1 - T(x; W_T))x \]

where \( T(x; W_T) \in [0, 1] \) modulates how much the signal should be transformed.

This technique allowed them to train networks with up to 100 layers.
The residual networks proposed by He et al. (2015) simplify the idea and use a building block with a **skip connection**.

Thanks to this structure, the parameters are optimized to learn a **residual**, that is the difference between the value before the block and the one needed after.
We can implement such a network for MNIST, composed of:

- A first convolution layer $\text{conv0}$ with kernels $1 \times 1$ to convert the tensor from $1 \times 28 \times 28$ to $\text{nb_channels} \times 28 \times 28$,
- a series of ResBlocks, each composed of two convolution layers and two batch normalization layers, that maintains the tensor size unchanged,
- an average poling layer $\text{avg}$ that produces an output of size $\text{nb_channels} \times 1 \times 1$,
- a fully connected layer $\text{fc}$ to make the final prediction.
class ResBlock(nn.Module):
    def __init__(self, nb_channels, kernel_size):
        super(ResBlock, self).__init__()

        self.conv1 = nn.Conv2d(nb_channels, nb_channels, kernel_size,
                               padding = (kernel_size-1)//2)
        self.bn1 = nn.BatchNorm2d(nb_channels)

        self.conv2 = nn.Conv2d(nb_channels, nb_channels, kernel_size,
                               padding = (kernel_size-1)//2)
        self.bn2 = nn.BatchNorm2d(nb_channels)

    def forward(self, x):
        y = self.bn1(self.conv1(x))
        y = F.relu(y)
        y = self.bn2(self.conv2(y))
        y += x
        y = F.relu(y)
        return y
class ResNet(nn.Module):
    def __init__(self, nb_channels, kernel_size, nb_blocks):
        super(ResNet, self).__init__()

        self.conv0 = nn.Conv2d(1, nb_channels, kernel_size = 1)

        self.resblocks = nn.Sequential(
            # A bit of fancy Python
            *(ResBlock(nb_channels, kernel_size) for _ in range(nb_blocks))
        )

        self.avg = nn.AvgPool2d(kernel_size = 28)
        self.fc = nn.Linear(nb_channels, 10)

    def forward(self, x):
        x = F.relu(self.conv0(x))
        x = self.resblocks(x)
        x = F.relu(self.avg(x))
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x
With 25 residual blocks, 16 channels, and convolution kernels of size $3 \times 3$, we get the following structure, with 117,802 parameters.

```python
ResNet(
    (conv0): Conv2d(1, 16, kernel_size=(1, 1), stride=(1, 1))
    (resblocks): Sequential(
        (0): ResBlock(
            (conv1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        ),
        ...
        (24): ResBlock(
            (conv1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        ),
    ),
    (avg): AvgPool2d(kernel_size=28, stride=28, padding=0)
    (fc): Linear(in_features=16, out_features=10, bias=True)
)
```
A technical point for a more general use of a residual architecture is to deal with convolution layers that change the activation map sizes or numbers of channels.

He et al. (2015) only consider:

- reducing the activation map size by a factor 2,
- increasing the number of channels.
To reduce the activation map size by a factor 2, the identity pass-through extracts $1/4$ of the activations over a regular grid (i.e. with a stride of 2),
To increase the number of channels from $C$ to $C'$, they propose to either:

- pad the original value with $C' - C$ zeros, which amounts to adding as many zeroed channels, or
- use $C'$ convolutions with a $1 \times 1 \times C$ filter, which corresponds to applying the same fully-connected linear model $\mathbb{R}^C \rightarrow \mathbb{R}^{C'}$ at every location.
Finally, He et al.’s residual networks are fully convolutional, which means they have no fully connected layers. We will come back to this.

Their one-before last layer is a per-channel global average pooling that outputs a $1d$ tensor, fed into a single fully-connected layer.
Figure 3. Example network architectures for ImageNet.

(He et al., 2015)
Performance on ImageNet.

Figure 4. Training on ImageNet. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

(He et al., 2015)
Veit et al. (2016) interpret a residual network as an ensemble, which explains in part its stability.

*E.g.*, with three blocks we have

\[
\begin{align*}
x_1 &= x_0 + f_1(x_0) \\
x_2 &= x_1 + f_2(x_1) \\
x_3 &= x_2 + f_3(x_2)
\end{align*}
\]

hence there are four "paths":

\[
\begin{align*}
x_3 &= x_2 + f_3(x_2) \\
    &= x_1 + f_2(x_1) + f_3(x_1 + f_2(x_1)) \\
    &= x_0 + f_1(x_0) + f_2(x_0 + f_1(x_0)) + f_3(x_0 + f_1(x_0) + f_2(x_0 + f_1(x_0))).
\end{align*}
\]

Veit et al. show that (1) performance reduction correlates with the number of paths removed from the ensemble, not with the number of blocks removed, (2) only gradients through shallow paths matter during train.
An extension of the residual network, is the **stochastic depth** network.

“Stochastic depth aims to shrink the depth of a network during training, while keeping it unchanged during testing. We can achieve this goal by randomly dropping entire ResBlocks during training and bypassing their transformations through skip connections.”

(Huang et al., 2016)
Shattered Gradient
Balduzzi et al. (2017) points out that depth “shatters” the relation between the input and the gradient wrt the input, and that Resnets mitigate this effect.

Since linear networks avoid this problem, they suggest to combine CReLU with a Looks Linear initialization that makes the network linear initially.
Let $\sigma(x) = \max(0, x)$, and

$$\Phi : \mathbb{R}^D \rightarrow \mathbb{R}^{2D}$$

the CReLU non-linearity, i.e.

$$\forall x \in \mathbb{R}^D, \ q = 1, \ldots, D, \left\{\begin{array}{ll}
\Phi(x)_{2q-1} &= \sigma(x_q), \\
\Phi(x)_{2q} &= \sigma(-x_q)
\end{array}\right.$$

and a weight matrix $\tilde{W} \in \mathbb{R}^{D' \times 2D}$ such that

$$\forall j = 1, \ldots, D', \ q = 1, \ldots, D, \ \tilde{W}_{j,2q-1} = -\tilde{W}_{j,2q} = W_{j,q}.$$

So two neighboring columns of $\Phi(x)$ are the $\sigma(\cdot)$ and $\sigma(-\cdot)$ of a column of $x$, and two neighboring columns of $\tilde{W}$ are a column of $W$ and its opposite.
From this we get, $\forall i = 1, \ldots, B$, $j = 1, \ldots, D'$:

$$
\left( \tilde{W}\Phi(x) \right)_j = \sum_{k=1}^{2D} \tilde{W}_{j,k} \Phi(x)_k
$$

$$
= \sum_{q=1}^{D} \tilde{W}_{j,2q-1}\Phi(x)_{2q-1} + \tilde{W}_{j,2q}\Phi(x)_{2q}
$$

$$
= \sum_{q=1}^{D} W_{j,q}\sigma(x_q) - W_{j,q}\sigma(-x_q)
$$

$$
= \sum_{q=1}^{D} W_{j,q} x_q
$$

$$
= (Wx)_j.
$$

Hence

$$
\forall x, \quad \tilde{W}\Phi(x) = Wx
$$

and doing this in every layer results in a linear network.
The Shattered Gradients Problem

The thinness of the architecture makes it particularly difficult for gradients to propagate at high depth. The reduction following the last reduction the representation is passed to the next. It is very common that correlations between gradients decays changes from exponential for feedforward architectures to sublinear for residual networks even when effective initialization and batch normalization are employed. Averaging gradients over mini-batches becomes analogous to integrating over white noise, which can lead to large differences in gradients.

The other networks compared a CReLU architecture with an orthogonal initialization. An interesting future direction is to investigate hybrid designs initializations that do not shatter such as the LL-init. An underappreciated consequence of the exponential growth in linear regions is the proliferation of discontinuities in the gradients of rectifier nets. An underappreciated consequence of the exponential growth in linear regions is the proliferation of discontinuities in the gradients of rectifier nets. An underappreciated consequence of the exponential growth in linear regions is the proliferation of discontinuities in the gradients of rectifier nets. An underappreciated consequence of the exponential growth in linear regions is the proliferation of discontinuities in the gradients of rectifier nets.

The numbers of filters per layer of the CReLU models were adjusted by a factor of \( \frac{\sqrt{2}}{2} \) entries of \( A \times W \times B \). The kernel is used in conjunction with the orthogonal initialization. The orthogonality of the kernel is improved by averaging directions. The other networks are increasingly uncorrelated. Shattered gradients play havoc with the optimization methods currently in use. The maximum depth of 198. The feedforward net with LL-init against an equivalent CReLU network, resnet, and a standard feedforward ReLU network. The other networks performed comparably to a resnet, suggesting that shattered gradients are a large part of the problem in training very deep networks even when effective initialization and batch normalization are employed. Averaging gradients over mini-batches becomes analogous to integrating over white noise, which can lead to large differences in gradients.

Updates were performed with Adam (Kingma & Ba, 2015). After every two modules following He et al. (2016a), the same as the basic ReLU model with skip-connections was performed; translating up to 4 pixels in any direction. Shattered gradients can also introduce numerical instabilities, since small differences in the input can lead to large differences in the output. In Monte Carlo simulations, the auto-scheduler that measures how quickly the loss on the training set has been decreasing over the last ten epochs, was measured. Averaging gradients over mini-batches becomes analogous to integrating over white noise, which can lead to large differences in gradients.

Average direction. Shattered gradients can also introduce numerical instabilities, since small differences in the input can lead to large differences in the output. In Monte Carlo simulations, the auto-scheduler that measures how quickly the loss on the training set has been decreasing over the last ten epochs, was measured. Averaging gradients over mini-batches becomes analogous to integrating over white noise, which can lead to large differences in gradients.

We briefly describe how we orthogonally initialize a kernel. First, set all the \( A \) to zero. Second, sample a random matrix \( B \). The kernel is used in conjunction with the orthogonal initialization. The orthogonality of the kernel is improved by averaging directions.

Figure 6: CIFAR-10 test accuracy. Comparison of test accuracy between networks of different depths with and without LL initialization.

(Balduzzi et al., 2017)
We can summarize the techniques which have enabled the training of very deep architectures:

- rectifiers to prevent the gradient from vanishing during the backward pass,
- dropout to force a distributed representation,
- batch normalization to dynamically maintain the statistics of activations,
- identity pass-through to keep a structured gradient and distribute representation,
- smart initialization to put the gradient in a good regime.
References


