TTIC 31230, Fundamentals of Deep Learning

David McAllester, Winter 2020

Trainability:

ReLU, Batch Normalization, Initialization,

and Residual Connections (ResNet)

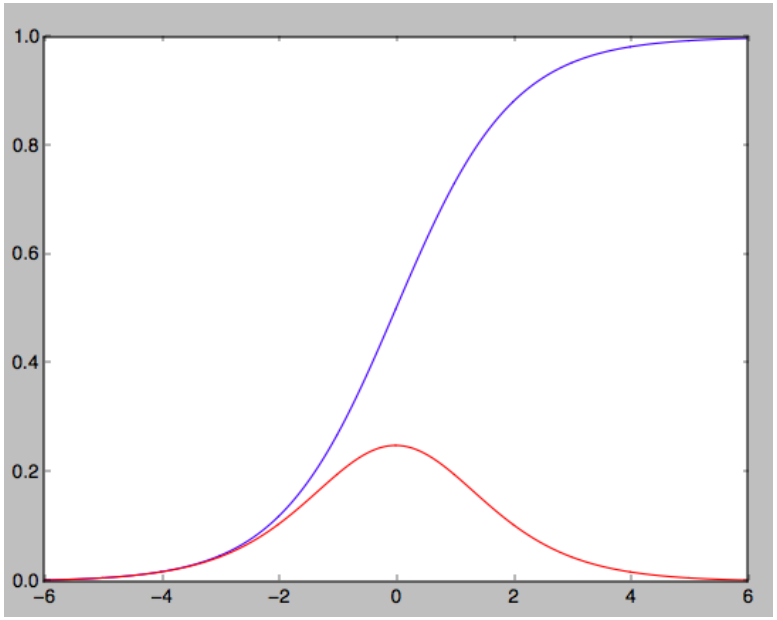
Trainability

The main issue in making deep neural networks trainable is maintaining meaningful gradients.

There are various difficulties.

Activation Function Saturation

Consider the sigmoid activation function $1/(1+e^{-x})$.

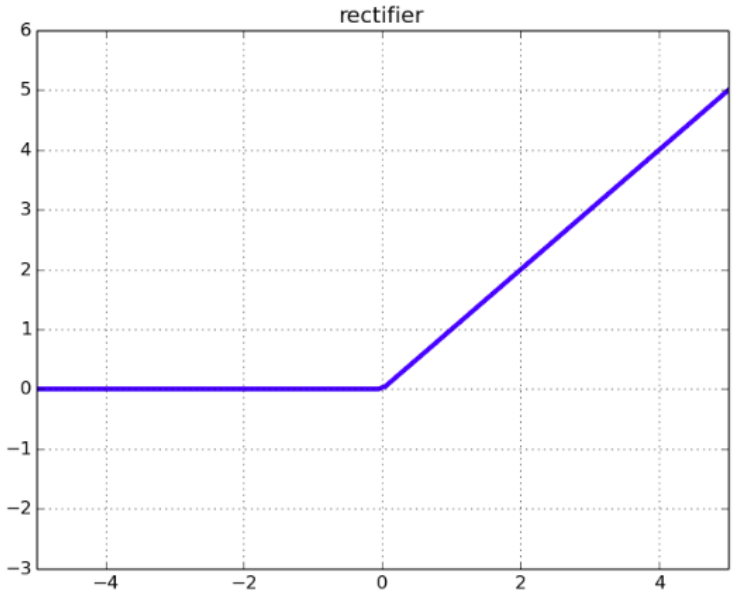


The gradient of this function is quite small for $|x| > 4$.

In deep networks backpropagation can go through many sigmoids and we get "vanishing gradients".

The Rectified Linear Unit Activation Function (ReLU)

$$\mathrm{ReLU}(x) = \max(x, 0)$$



The activation function $\mathrm{ReLU}(x)$ does not saturate for $x > 0$.

Repeated Multiplication by Network Weights

Consider a deep CNN.

$$L_{i+1} = \text{ReLU}(\text{Conv}(\Phi_i, L_i))$$

For i large, L_i has been multiplied by many weights.

If the weights are small then the neuron values, and hence the weight gradients, decrease exponentially with depth. **Vanishing Gradients.**

If the weights are large, and the activation functions do not saturate, then the neuron values, and hence the weight gradients, increase exponentially with depth. **Exploding Gradients.**

Repeated Multiplication by Network Weights

The problem of repeated multiplication by network weights can be addressed with careful initialization.

We want an initialization for which the values stay in the active regions of the activation functions — zero mean and unit variance.

Initialization

Consider a linear threshold unit

$$y[j] = \sigma(W[j, I]x[I] - B[j])$$

We want the scalar $y[j]$ to have zero mean and unit variance.

Xavier initialization initializes $B[j]$ to zero and randomly draws $W[j, i]$ from a uniform distribution on $\left(-\sqrt{3/I},\ \sqrt{3/I}\right)$.

Assuming $x[i]$ has zero mean and unit variance, this gives zero mean and unit variance for $W[j, I]x[I]$.

Batch Normalization

We can also enforce zero mean, unit variance, values dynamically with normalization layers.

In vision networks this is most commonly done with Batch Normalization.

Batch Normalization

Given a tensor $x[b, j]$ we define $\tilde{x}[b, j]$ as follows.

$$\hat{\mu}[j] = \frac{1}{B} \sum_{b} x[b, j]$$

$$\hat{\sigma}[j] = \sqrt{\frac{1}{B-1} \sum_{b} (x[b, j] - \hat{\mu}[j])^2}$$

$$\tilde{x}[b, j] = \frac{x[b, j] - \hat{\mu}[j]}{\hat{\sigma}[j]}$$

At test time a single fixed estimate of $\mu[j]$ and $\sigma[j]$ is used.

Spatial Batch Normalization

For CNNs we convert a tensor $x[b, x, y, j]$ to $\tilde{x}[b, x, y, j]$ as follows.

$$\hat{\mu}[j] = \frac{1}{BXY} \sum_{b,x,y} x[b, x, y, j]$$

$$\hat{\sigma}[j] = \sqrt{\frac{1}{BXY - 1} \sum_{b,x,y} (x[b, x, y, j] - \hat{\mu}[j])^2}$$

$$\tilde{x}[b, x, y, j] = \frac{x[b, x, y, j] - \hat{\mu}[j]}{\hat{\sigma}[j]}$$

Layer Normalization

For CNNs we convert a tensor $x[b, x, y, j]$ to $\tilde{x}[b, x, y, j]$ as follows.

$$\hat{\mu}[b, j] = \frac{1}{XY} \sum_{x,y} x[b, x, y, j]$$

$$\hat{\sigma}[b, j] = \sqrt{\frac{1}{XY - 1} \sum_{x,y} (x[b, x, y, j] - \hat{\mu}[j])^2}$$

$$\tilde{x}[b, x, y, j] = \frac{x[b, x, y, j] - \hat{\mu}[b, j]}{\hat{\sigma}[b, j]}$$

Adding an Affine Transformation

$$\breve{x}[b, x, y, j] = \gamma[j]\tilde{x}[b, x, y, j] + \beta[j]$$

Here $\gamma[j]$ and $\beta[j]$ are parameters of the batch normalization.

This allows the batch normlization to learn an arbitrary affine transformation (offset and scaling).

It can even undo the normaliztion.

Batch Normalization

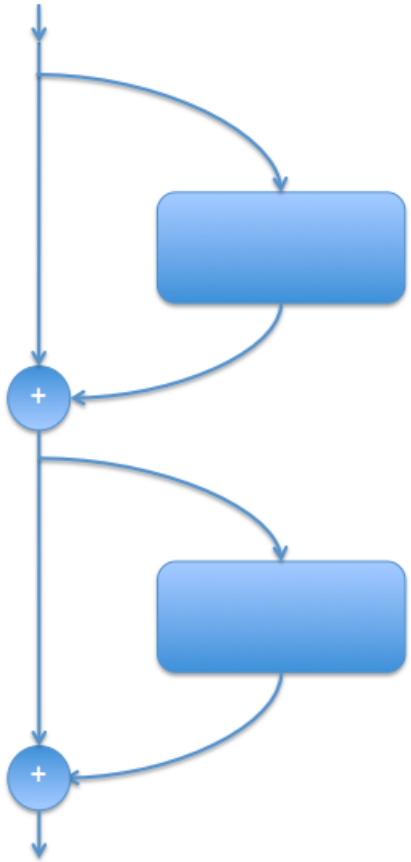
Batch Normalization appears to be generally useful in CNNs but is not always used.

Not so successful in RNNs.

It is typically used just prior to a nonlinear activation function.

It is intuitively justified in terms of "internal covariate shift": as the inputs to a layer change the zero mean unit variance property underlying Xavier initialization are maintained.

Residual Connections (ResNet)



A residual connection produces the sum of the previous layer and the new layer.

The residual connection connects input to output directly and hence preserves gradients.

Residual Connections were introduced in late 2015 (Kaiming He et al.) and are now standard in all areas of deep learning.

A Simple Residual Connection in CNNs (stride 1)

$$L_{\ell+1}[B, X, Y, J] = \text{Conv}(W_{\ell+1}[X, Y, J, J], B_{\ell+1}[J], L_{\ell}[B, X, Y, J])$$

$$+ \; L_{\ell}[B, X, Y, J]$$

(Recall that we use capital letter indices to denote entire tensors and lower case letters for particular indeces.)

Multiple Convolutions Between Additions

$$L_{\ell+1}[B, X, Y, J] = \text{Conv}(W_{\ell+1}[X, Y, J, J], B_{\ell+1}[J], L_{\ell}[B, X, Y, J])$$

$$L_{\ell+2}[B, X, Y, J] = \text{Conv}(W_{\ell+2}[X, Y, J, J], B_{\ell+2}[J], L_{\ell+1}[B, X, Y, J])$$

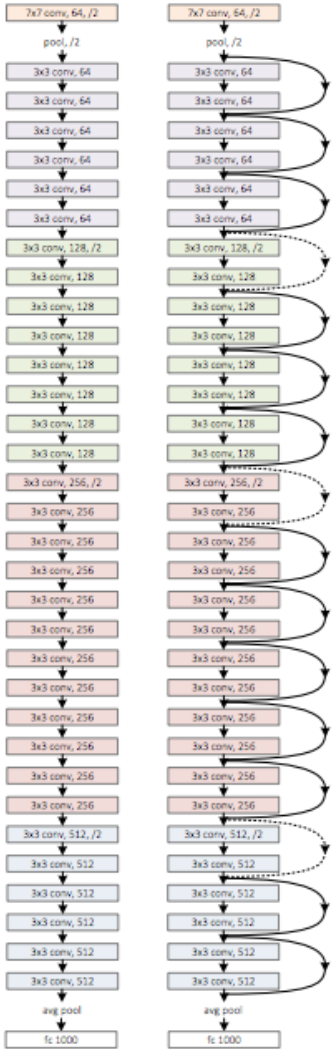
$$L_{\ell+3} = L_{\ell} + L_{\ell+2}$$

ResNet32

plain net

ResNet

er)



[Kaiming He]

Reducing Spacial Dimension

$$L_{\ell+1} = \text{Conv}(W_{\ell+1}, B_{\ell+1}, L_{\ell})$$

$$L_{\ell+2} = \text{Conv}(W_{\ell+2}, B_{\ell+2}, L_{\ell+1})$$

$$L_{\ell+3} = L_{\ell} + L_{\ell+2}$$

All these layers have the same shape.

Reducing Spacial Dimension

$$L_{\ell+1} = \text{Conv}(W_{\ell+1}, B_{\ell+1}, L_\ell, \text{ stride } s)$$

$$L_{\ell+2} = \text{Conv}(W_{\ell+2}, B_{\ell+2}, L_{\ell+1})$$

$$L_{\ell+3} = \tilde{L}_\ell + L_{\ell+2}$$

where

$$\begin{aligned} X_{\ell+1} &= X_\ell/s \\ Y_{\ell+1} &= Y_\ell/s \\ J_{\ell+1} &\geq J_\ell \end{aligned}$$

Reducing Spacial Dimension

$$L_{\ell+1} = \text{Conv}(W_{\ell+1}, B_{\ell+1}, L_\ell, \text{ stride } s)$$

$$L_{\ell+2} = \text{Conv}(W_{\ell+2}, B_{\ell+2}, L_{\ell+1})$$

$$L_{\ell+3} = \tilde{L}_\ell + L_{\ell+2}$$

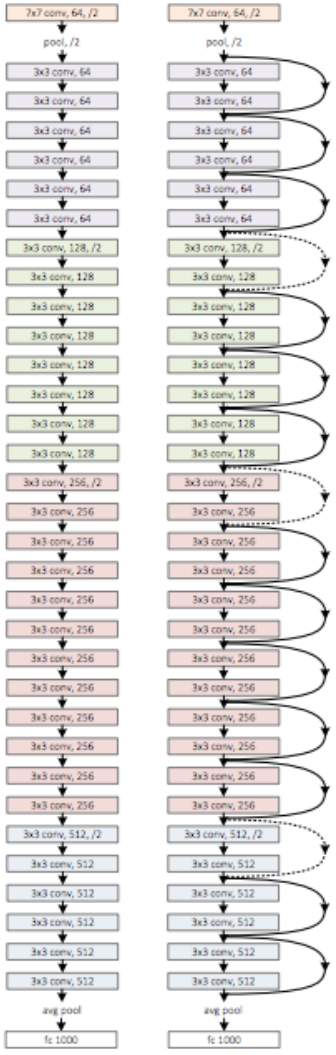
$$\tilde{L}_\ell[b, x, y, j] = \begin{cases} L_\ell[b, s*x, s*y, j] & \text{for } j < J_\ell \\ 0 & \text{otherwise} \end{cases}$$

ResNet32

plain net

ResNet

er)



[Kaiming He]

Deeper Versions use Bottleneck Layers

We reduce the number of neurons at each position in an intermediate "bottleneck" layer.

$$
\begin{aligned}
L_{\ell+1} &= \mathrm{Conv}(W_{\ell+1}, B_{\ell+1}, L_{\ell}) \\
L_{\ell+2} &= \mathrm{Conv}(W_{\ell+2}, B_{\ell+2}, L_{\ell+1}) \\
L_{\ell+3} &= \mathrm{Conv}(W_{\ell+3}, B_{\ell+3}, L_{\ell+2}) \\
L_{\ell+4} &= L_{\ell} + L_{\ell+3}
\end{aligned}
$$

Here $J_{\ell+4} = J_{\ell}$ but $J_{\ell+1} = J_{\ell+2} < J_{\ell}$.

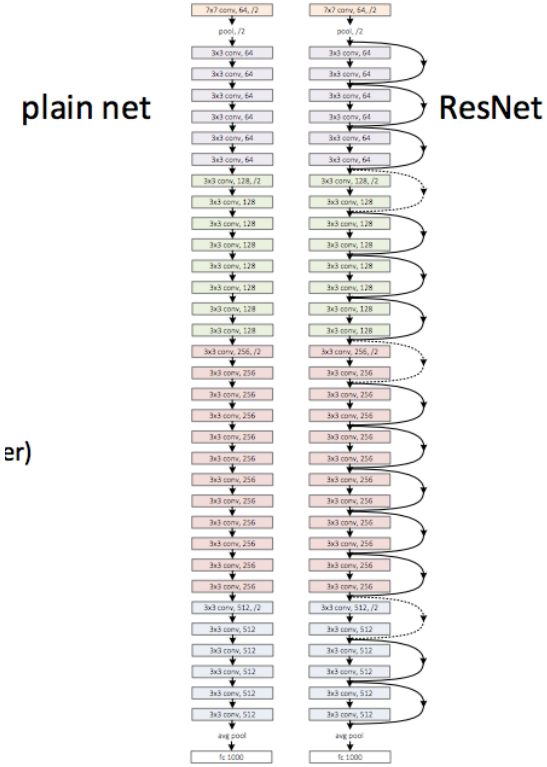
Reducing the number of neurons greatly reduces the size of the convolution weight tensor which has size $\Delta X \times \Delta Y \times J_{\ell} \times J_{\ell+1}$.

A General Residual Connection

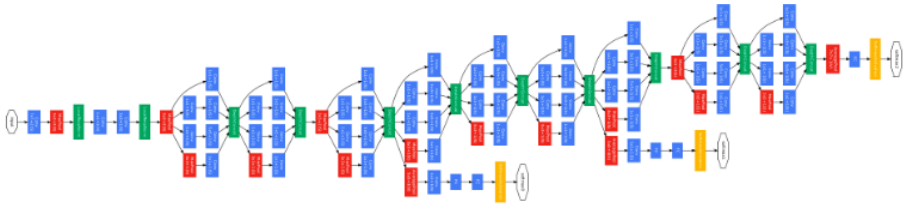
$$y = \tilde{x} + R(x)$$

Where $\tilde{x}$ is either x or a version of x adjusted to match the shape of $R(x)$.

ResNet Simplicity



[Kaiming He]



ResNet Power

ResNet gives powerful image classification.

ResNet is used in folding proteins.

ResNet is the network used in AlphaZero for Go, Chess and Shogi.

Residual connections are now universal in all forms of deep models such as RNNs and Transformers in language processing.

END