TTIC 31230, Fundamentals of Deep Learning

David McAllester, Winter 2018

Controlling Gradients

Vanishing and Exploding Gradients

Initialization

Batch Normalization

Residual Networks

Gated RNNs

Vanishing and Exploding Gradients

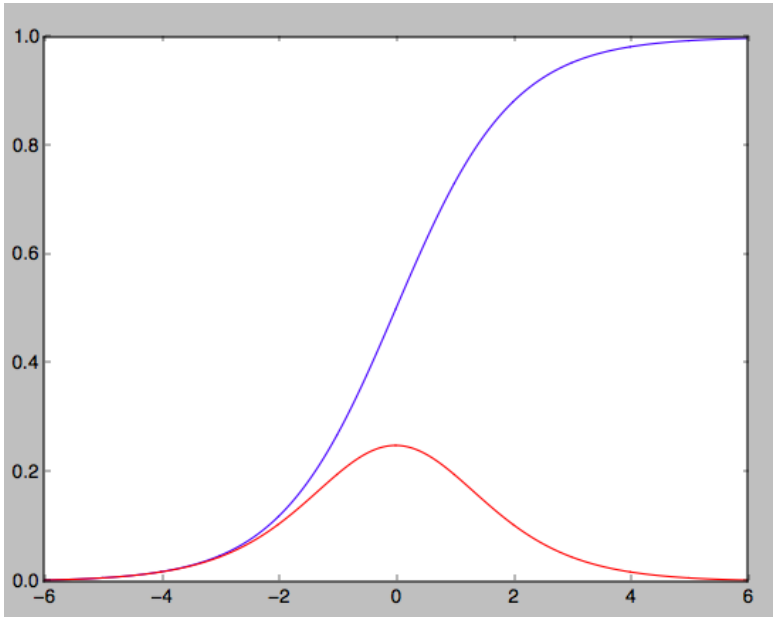
Causes of Vanishing and Exploding Gradients:

Activation function saturation

Repeated multiplication by network weights

Activation Function Saturation

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

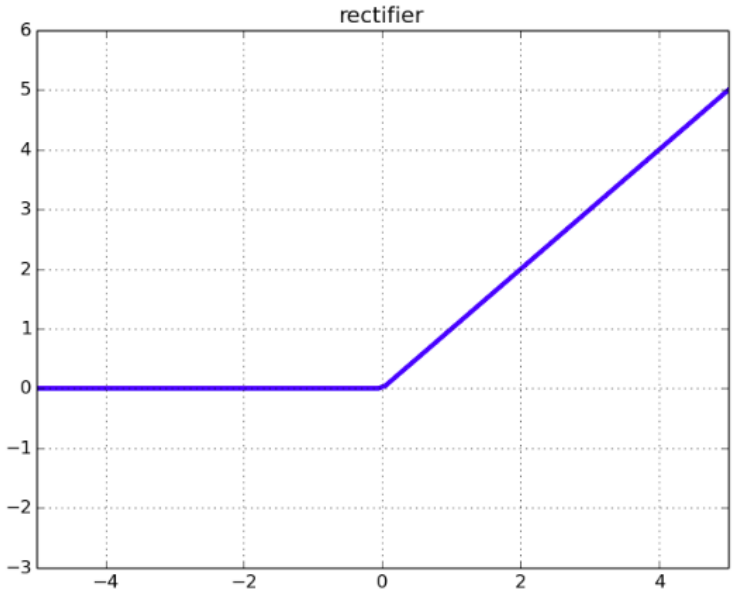


The gradient of this function is quite small for $|x| > 4$.

In deep networks backpropagation can go through many sigmoids and the gradient can "vanish"

Activation Function Saturation

$\text{Relu}(x) = \max(x, 0)$



The Relu does not saturate at positive inputs (good) but is completely saturated at negative inputs (bad).

Alternate variations of Relu still have small gradients at negative inputs.

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.**

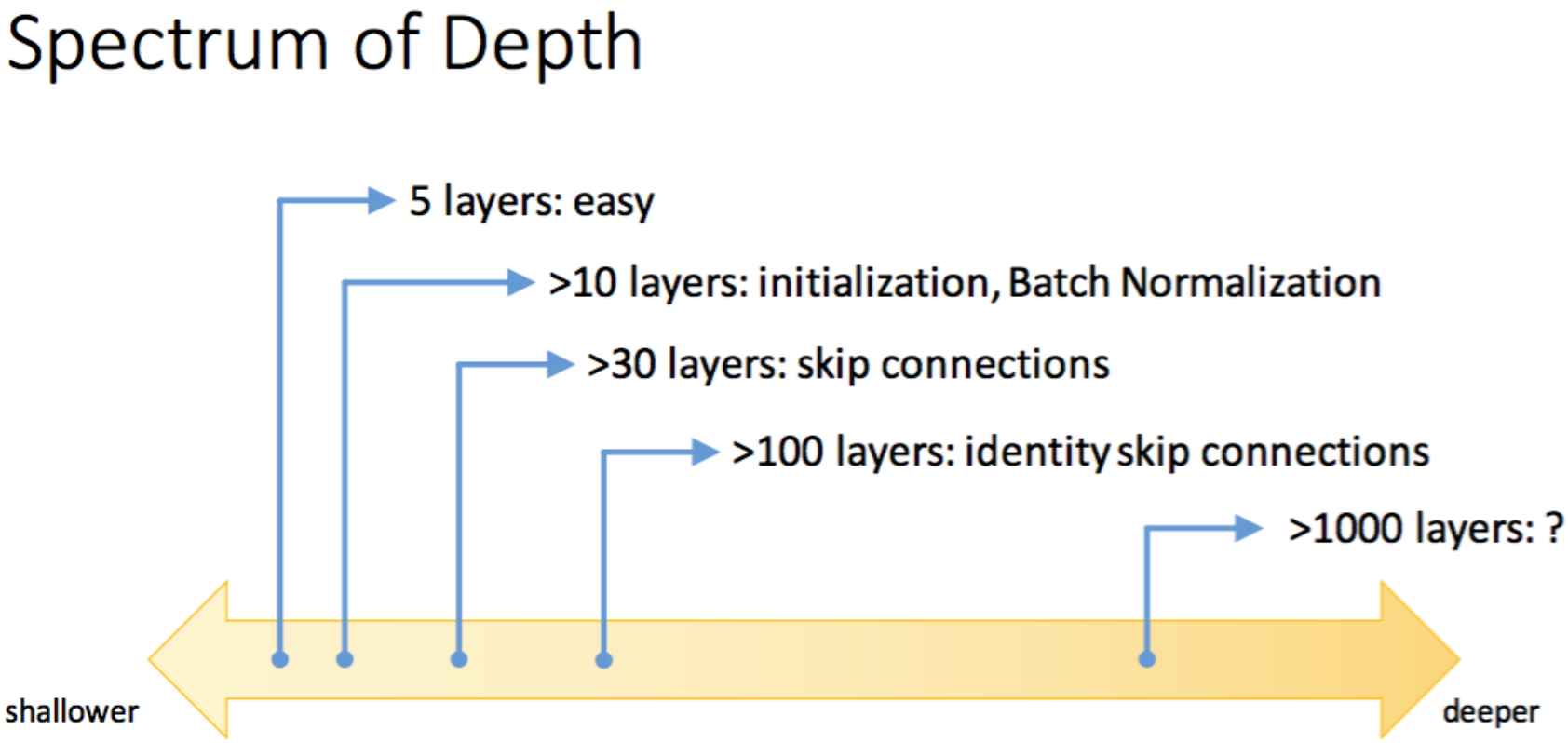
Methods for Maintaining Gradients

Initialization

Batch Normalization

Highway Architectures (Skip Connections)

Methods for Maintaining Gradients



Kaiming He

Initialization

Xavier Initialization

Initialize a weight matrix (or tensor) to preserve zero-mean unit variance distributions.

If we assume x_i has unit mean and zero variance then we want

$$y_j = \sum_{j=0}^{N-1} x_i w_{i,j}$$

to have zero mean and unit variance.

Xavier initialization randomly sets $w_{i,j}$ to be uniform in the interval $\left(-\sqrt{\frac{3}{N}}, \sqrt{\frac{3}{N}}\right)$.

Assuming independence this gives zero mean and unit variance for y_j.

He Initialization

A Relu nonlinearity reduces the variance.

Before a Relu nonlinearity it seems better to use the larger interval $\left(-\sqrt{\frac{6}{N}},\ \sqrt{\frac{6}{N}}\right)$.

Batch Normalization

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.

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

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

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

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.

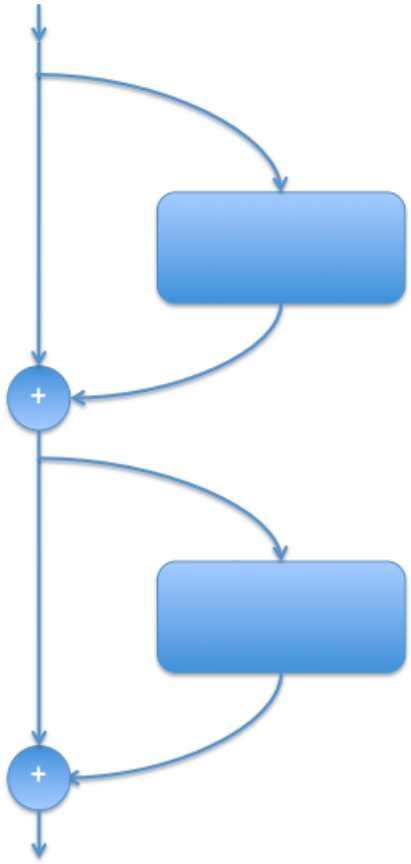
Not so successful in RNNs.

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.

Highway Architectures (Skip Connections)

Deep Residual Networks (ResNets) by Kaiming He 2015



A "skip connection" is adjusted by a "residual correction"

The skip connections connects input to output directly and hence preserves gradients.

ResNets were introduced in late 2015 (Kaiming He et al.) and revolutionized computer vision.

Simple Residual Skip Connections in CNNs (stride 1)

$$L_{i+1} = L_i + R_i$$

In ResNet R_i is computed by two or three convolution layers.

If all convolution layers are stride 1, and preserve the number of channels, then R_i and L_i are the same shape.

Handling Spacial Reduction

Let the shape of L_i be $[B, X_i, Y_i, J_i]$.

Let the shape of R_i be $[B, X_{i+1}, Y_{i+1}, J_{i+1}]$ where $X_{i+1} = X_i/s$ and $Y_{i+1} = Y_i/s$ for some stride s, and where $J_{i+1} \geq J_i$.

In this case we construct $\tilde{L}_i$ to have the same shape as R_i.

$$\tilde{L}_i[b, x, y, j] = \begin{cases} L_i[b, s * x, s * y, j] & \text{for } j < J_i \\ 0 & \text{otherwise} \end{cases}$$

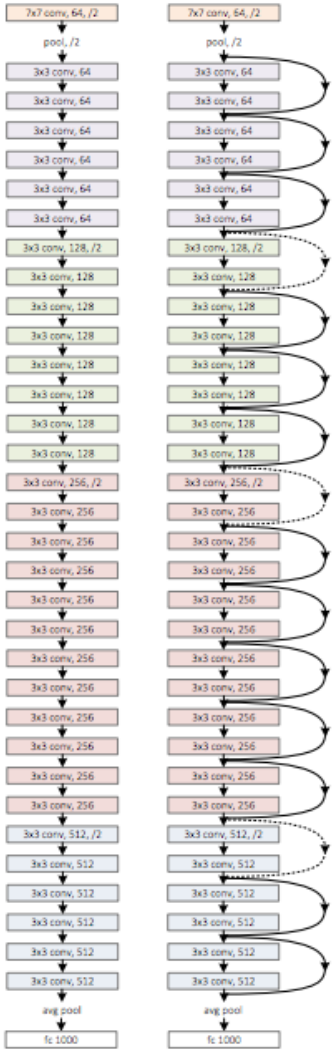
$$L_{i+1} = \tilde{L}_i + R_i$$

Resnet32

plain net

ResNet

er)



[Kaiming He]

Deeper Versions use Bottleneck Residual Paths

We reduce the number of channels to $K < J_i$ before doing the convolution.

$$A_i[B, X_i, Y_i, K] = \text{Conv}'(\Phi_i^A[1, 1, J_i, K], L_i[B, X_i, Y_i, J_i])$$

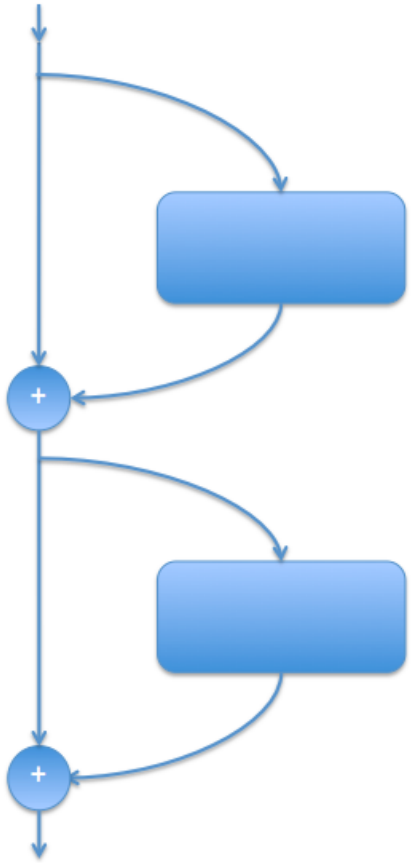
$$B_i[B, X_{i+1}, Y_{i+1}, K] = \text{Conv}'(\Phi_i^B[3, 3, K, K], A_i[B, X_i, Y_i, K], \text{ stride } s)$$

$$R_i[B, X_{i+1}, Y_{i+1}, J_{i+1}] = \text{Conv}'(\Phi_i^R[1, 1, K, J_{i+1}], B_i[B, X_{i+1}, Y_{i+1}, K])$$

$$L_{i+1} = \tilde{L}_i + R_i$$

Here CONV$'$ may include batch normalization and/or an activation function.

Deep Residual Networks



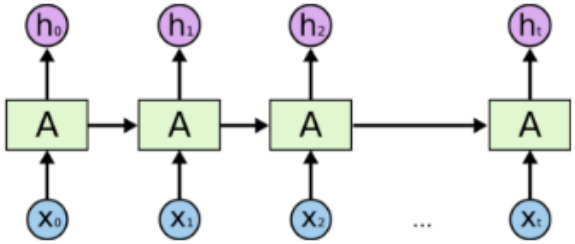
As with most of deep learning, not much is known about what resnets are actually doing.

For example, different residual paths might update disjoint channels making the networks shallower than they look.

They are capable of representing very general circuit topologies.

Recurrent Neural Networks (RNNs)

Vanilla RNNs



[Christopher Olah]

$$\tilde{h}_{t+1}[b,j] = \left(\sum_i W^{h,h}[i,j]h_t[b,i]\right) + \left(\sum_k W^{x,h}[k,j]x_t[b,k]\right) + \beta[j]$$

$$\text{Parameter } \Phi = (W^{h,h}[J,J],\ W^{x,h}[K,J],\ \beta[J])$$

Exploding and Vanishing Gradients

An RNN uses the same weights at every time step.

If we avoid saturation of the activation functions then we get exponentially growing or shrinking eigenvectors of the weight matrix.

Note that if the forward values are bounded by sigmoids or tanh then they cannot explode.

However the gradients can still explode.

Exploding Gradients: Gradient Clipping

We can dampen the effect of exploding gradients by clipping them before applying SGD.

$$W.\text{grad} = \begin{cases} W.\text{grad} & \text{if } ||W.\text{grad}|| \leq n_{\max} \\ \\ n_{\max}\, W.\text{grad}/||W.\text{grad}|| & \text{otherwise} \end{cases}$$

See `torch.nn.utils.clip_grad_norm`

Vanishing Gradients: Gated Skip Connections

Residual Skip: $L_{i+1} = \tilde{L}_i + R_i$
(layer-specific parameters)

Gated Skip: $h_{t+1} = G_t \odot h_t + (1 - G_t) \odot R_t$
(time-independent parameters)

$$(G \odot h)[b, j] = G[b, j] * h[b, j]$$

$$(1 - G_t)[b, j] = 1 - G_t[b, j]$$

Gating allows data dependent data flow.

Update Gate RNN (UGRNN)

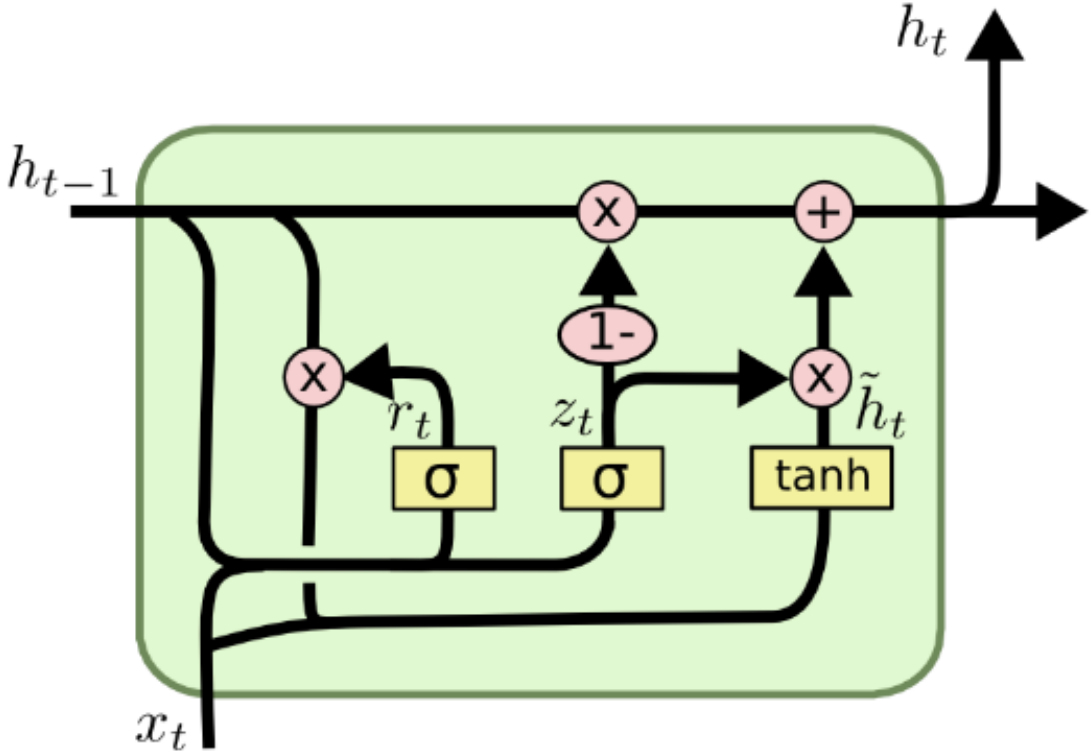
$$\tilde{R}_t[b,j] = \left(\sum_i W^{h,R}[i,j]h_t[b,i]\right) + \left(\sum_k W^{x,R}[k,j]x_t[b,k]\right) + \beta^R[j]$$

$$\tilde{G}_t[b,j] = \left(\sum_i W^{h,G}[i,j]h_t[b,i]\right) + \left(\sum_k W^{x,G}[k,j]x_t[b,k]\right) + \beta^G[j]$$

$$h_{t+1}[b,j] = \sigma(G_t[b,j])h_t[b,j] + (1-\sigma(G_t[b,j]))\tanh(R_t[b,j])$$

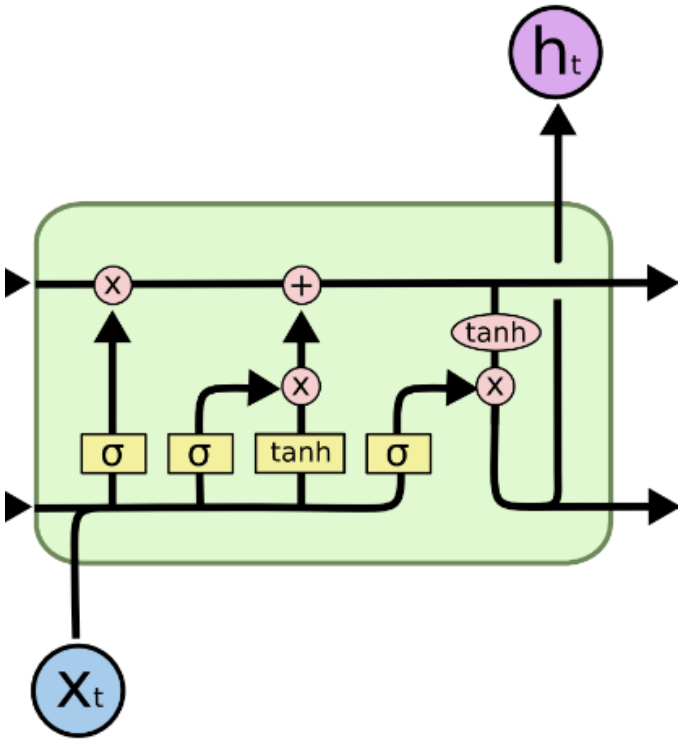
$$\Phi = (W^{h,R}, W^{x,R}, \beta^R, W^{h,G}, W^{x,G}, \beta^G)$$

Gated Recurrent Unity (GRU) by Cho et al. 2014



[Christopher Olah]

Long Short Term Memory (LSTM)



[figure: Christopher Olah]

[LSTM: Hochreiter&Shmidhuber, 1997]

UGRNN vs. GRUs vs. LSTMs

In class projects from previous years, GRUs consistently outperformed LSTMs.

A systematic study [Collins, Dickstein and Sussulo 2016] states:

> Our results point to the GRU as being the most learnable of gated RNNs for shallow architectures, followed by the UGRNN.

END