TTIC 31230, Fundamentals of Deep Learning

David McAllester

Archiectural Elements That Improve SGD

ReLu

Initialization

Batch Normalization

Residual Networks

Gated RNNs

DNNs are Expressive and Trainable

Universality Assumption: DNNs are universally expressive (can model any function) and trainable (the desired function can be found by SGD).

Universal trainability is clearly false but can still guide architecture design.

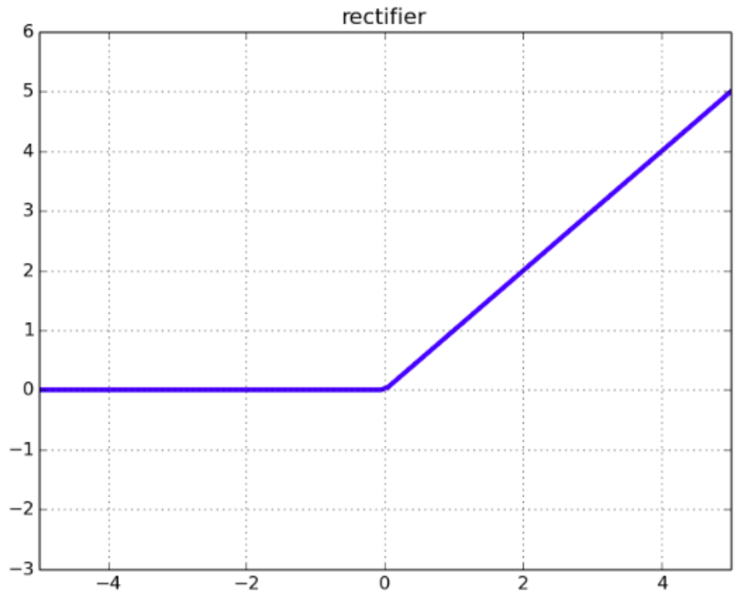
Equally expressive architectures differ in trainability.

We will consider architectural elements that improve SGD.

Very deep networks are particularly difficult for SGD.

Activation Function Saturation

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



The Relu does not saturate at positive inputs.

Repeated Multiplication by Network Weights

Consider

$$y = \sum_i w[i]x[i] \quad = \quad W[I]x[I]$$

If the weights are large the activations will grow exponenetially in depth.

If the weights are small the actvations will become exonentially small.

Repeated Multiplication by Network Weights

Exploding activations cause exploding gradients.

$$y \mathrel{\texttt{+=}} w[i]x[i]$$

$$w.grad \mathrel{\texttt{+=}} y.grad\; x[i]$$

The size of $w[i]$.grad is proportional to $x[i]$

He Initialization

Initialize weights to preserve zero-mean unit variance distributions values.

$$y = \sum_i w[i]x[i]$$

If we assume x_i has zero mean and unit variance then we want y to have zero mean and unit variance (over random training points).

He initialization randomly draws $w[i]$ from

$$\mathcal{N}(0, \sigma^2) \quad \sigma = \sqrt{1/N}$$

He Initialization

$$y = \sum_i w[i]x[i]$$

$$w[i] \sim \mathcal{N}(0, \sigma^2) \quad \sigma = \sqrt{1/N}$$

Assuming independence we have that y has zero mean and unit variance.

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 $L[b,x,y,n]$ to $\tilde{L}[b,x,y,n]$ as follows.

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

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

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

Layer Normalization

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

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

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

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

Adding an Affine Transformation

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

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

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

The affine transformation can undo the normaliztion but using ReLu activations the normalized value remains independent of scaling the weights and bias terms (thresholds) of the layer.

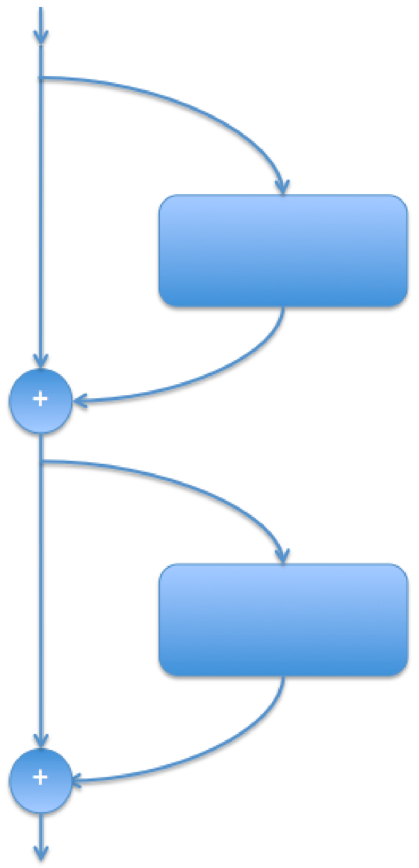
Batch Normalization

Spatial Batch Normalization is generally used in CNNs. Layer normalization is used in the transformer.

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.

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)

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

for b,x,y,j

$$L_{\ell+1}[b,x,y,j] \;=\; L_\ell[b,x,y,j]+R_{\ell+1}[b,x,y,j]$$

I will use capital letter indices to denote entire tensors and lower case letters for particular indeces.

Simple Residual Skip Connections in CNNs (stride 1)

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

for b, x, y, j

$$L_{\ell+1}[b,x,y,j] = L_\ell[b,x,y,j] + R_{\ell+1}[b,x,y,j]$$

Note that in the above equations $L_\ell[B,X,Y,J]$ and $R_{\ell+1}[B,X,Y,J]$ are the same shape.

In the actual ResNet $R_{\ell+1}$ is computed by two or three convolution layers.

Handling Spacial Reduction

Consider $L_\ell[B, X_\ell, Y_\ell, J_\ell]$ and $R_{\ell+1}[B, X_{\ell+1}, Y_{\ell+1}, J_{\ell+1}]$

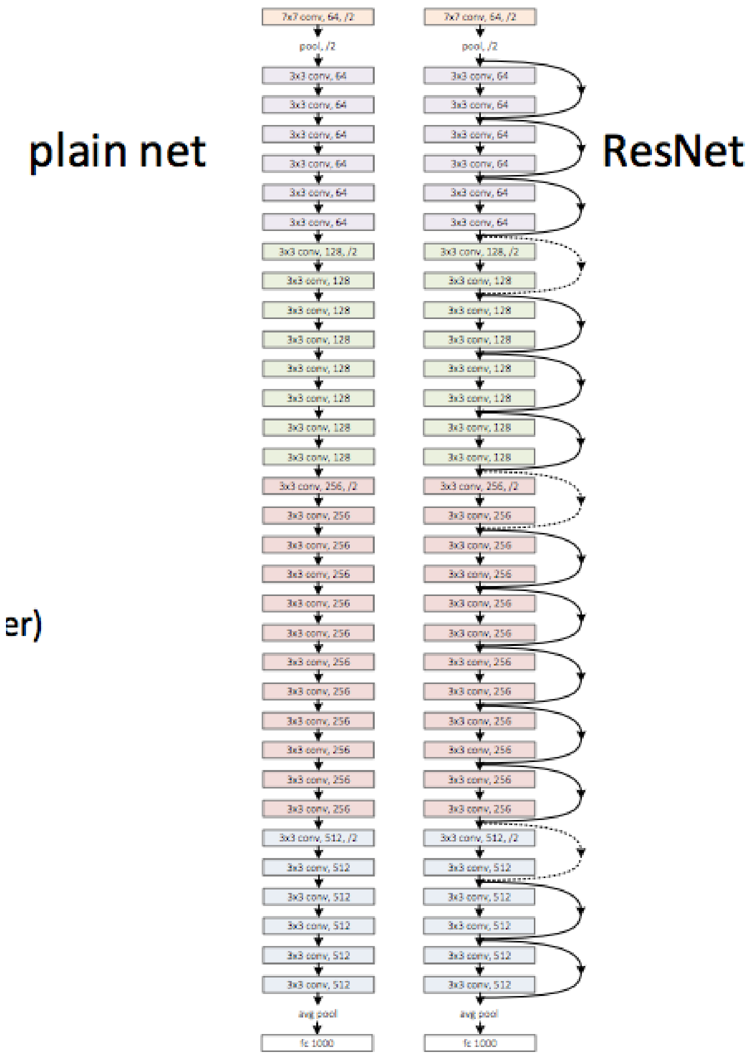
$$\begin{aligned} X_{\ell+1} &= X_\ell/s \\ Y_{\ell+1} &= Y_\ell/s \\ J_{\ell+1} &\geq J_\ell \end{aligned}$$

In this case we construct $\tilde{L}_\ell[B, X_{\ell+1}, Y_{\ell+1}, J_{\ell+1}]$

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

$$\begin{aligned} L_{\ell+1}[B, X_{\ell+1}, Y_{\ell+1}, J_{\ell+1}] = \; & \tilde{L}_\ell[B, X_{\ell+1}, Y_{\ell+1}, J_{\ell+1}] \\ & + R_{\ell+1}[B, X_{\ell+1}, Y_{\ell+1}, J_{\ell+1}] \end{aligned}$$

Resnet32



[Kaiming He]

Deeper Versions use Bottleneck Residual Paths

We reduce the number of features to $K < J$ before doing the convolution.

$$\begin{aligned}
U[B,X,Y,K] &= \mathrm{Conv}'(\Phi^A_{\ell+1}[1,1,J,K], L_\ell[B,X,Y,J]) \\
V[B,X,Y,K] &= \mathrm{Conv}'(\Phi^B_{\ell+1}[3,3,K,K], U[B,X,Y,K]) \\
R[B,X,Y,J] &= \mathrm{Conv}'(\Phi^R_{\ell+1}[1,1,K,J], V[B,X,Y,K]) \\
L_{\ell+1} &= L_\ell + R
\end{aligned}$$

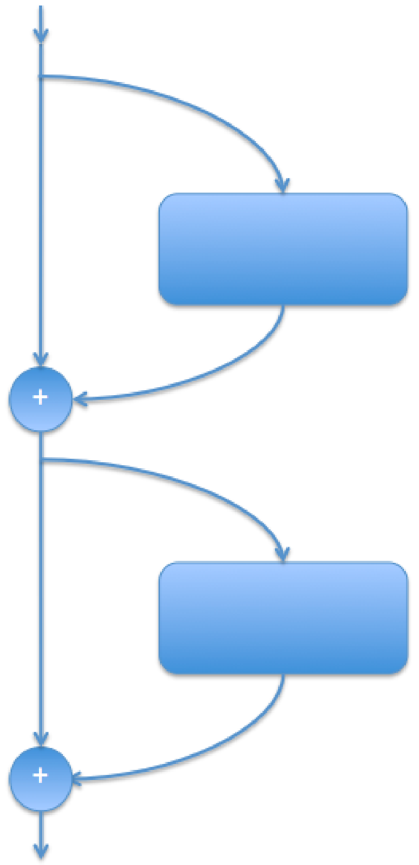
Here CONV$'$ may include batch normalization and/or an activation function.

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 $R(x)$.

Deep Residual Networks



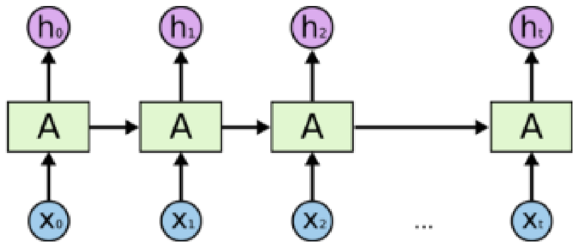
As with most of deep learning, not much is known about what resnets are actually doing.

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]

We use two input linear threshold units.

$$h_t[b,j] = \sigma\left(\left(\sum_i W^{h,h}[j,i]h_{t-1}[b,i]\right) + \left(\sum_k W^{x,h}[j,k]x_t[b,k]\right) - B[j]\right)$$

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

Exploding and Vanishing Gradients

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.

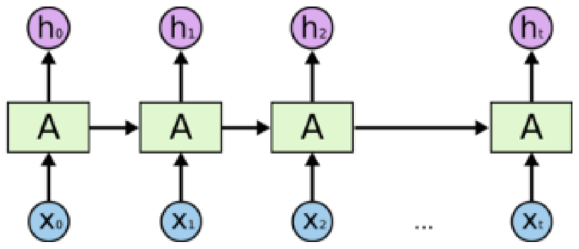
Exploding Gradients: Gradient Clipping

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

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

See `torch.nn.utils.clip_grad_norm`

Time as Depth



[Christopher Olah]

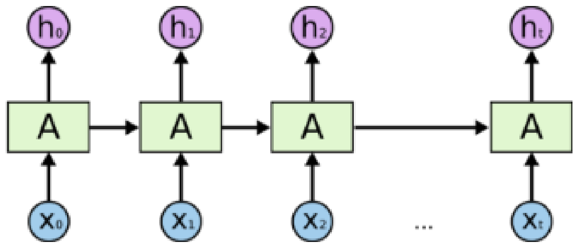
We would like the RNN to remember and use information from much earlier inputs.

All the issues with depth now occur through time.

However, for RNNs at each time step we use the same model parameters.

In CNNs at each layer uses its own model parameters.

Skip Connections Through Time



[Christopher Olah]

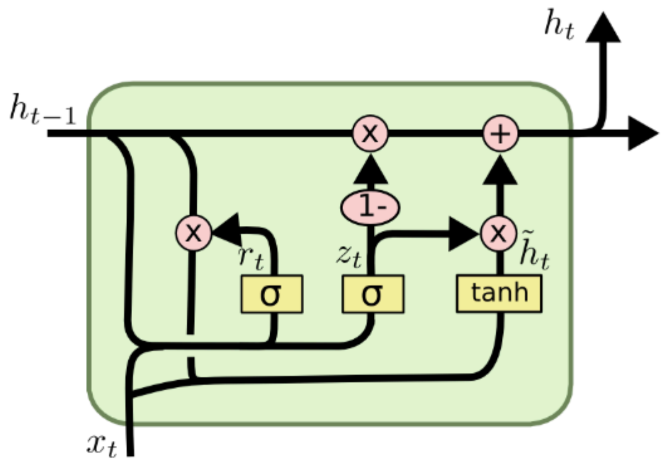
We would like to add skip connections through time.

However, We have to handle the fact that the same model parameters are used at every time step.

Update Gate RNN (UGRNN)

$$
\begin{aligned}
R_t[b,j] &= \tanh\left(\left(\sum_i W^{h,R}[j,i]h_{t-1}[b,i]\right) + \left(\sum_k W^{x,R}[j,k]x_t[b,k]\right) - B^R[j]\right) \\
G_t[b,j] &= \sigma\left(\left(\sum_i W^{h,G}[j,i]h_{t-1}[b,i]\right) + \left(\sum_k W^{x,G}[j,k]x_t[b,k]\right) - B^G[j]\right) \\
h_t[b,j] &= G_t[b,j]h_{t-1}[b,j] + (1 - G_t[b,j])R_t[b,j] \\
h_t[b,J] &= G_t[b,J] \odot h_{t-1}[b,J] + (1 - G_t[b,J]) \odot R_t[b,J] \quad \text{(alternate notation)} \\
\Phi &= (W^{h,R}, W^{x,R}, B^R, W^{h,G}, W^{x,G}, B^G)
\end{aligned}
$$

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

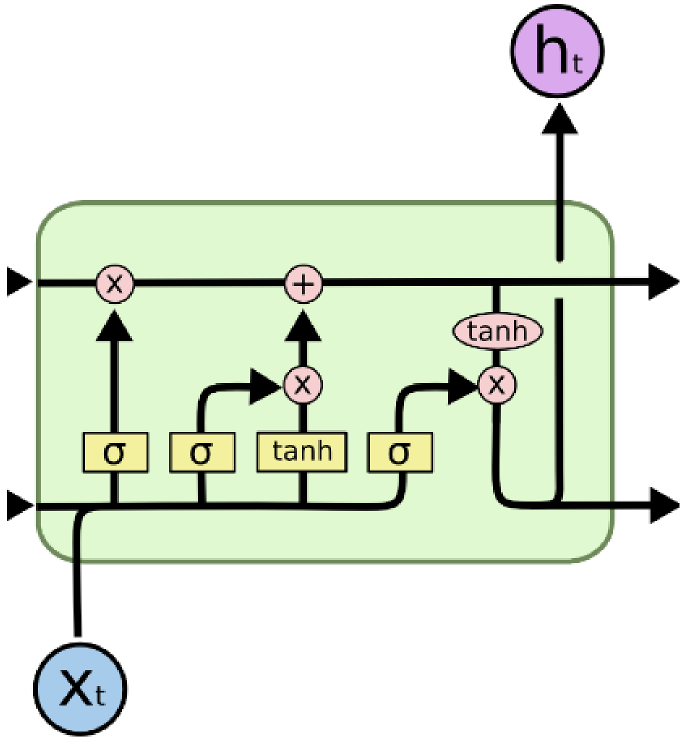


[Christopher Olah]

The right half is a UGRNN.

The GRU adds a gating on h_{t-1} before the tanh.

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.

Improving Trainability

Initialization

Batch Normalization

Residual Networks

Gated RNNs

END