TTIC 31230, Fundamentals of Deep Learning

David McAllester

Architetural Elements That Improve SGD

ReLu

Initialization

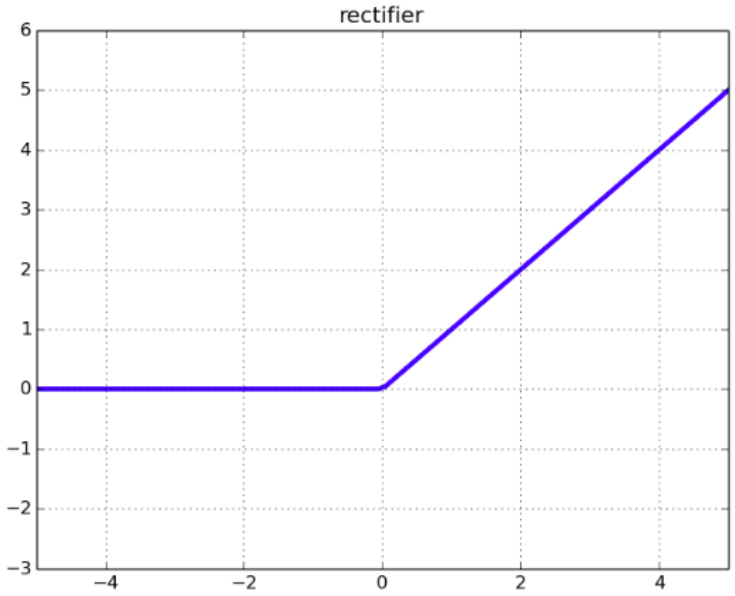
Normalization

Residual Connections

Gated RNNs

ReLu

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



The ReLu does not saturate at positive inputs.

Problems with Depth:

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 exponentially in network depth.

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

Problems with Depth:
Repeated Multiplication by Network Weights

Exploding activations cause exploding gradients.

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

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

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

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

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

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

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

Spatial Batch Normalization

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

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

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

Layer Normalization

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

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

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

Adding an Affine Transformation

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

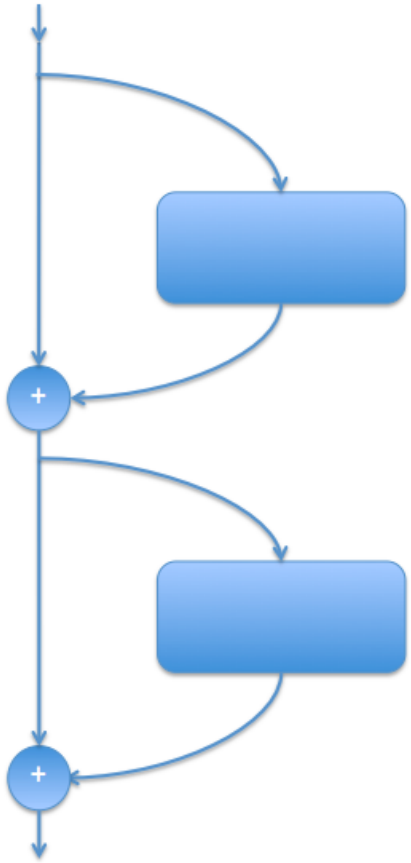
Here $\gamma[n]$ and $\beta[n]$ are parameters of the batch normalization.

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.

Residual Connections

Deep Residual Networks (ResNets) by Kaiming He 2015



A residual connections connects input to output directly and hence preserves gradients.

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

Residual Connections in CNNs

$$\tilde{L}_{\ell+1}[B, X, Y, N_{\text{out}}]$$

$$= \text{Conv}(K_{\ell+1}[N_{\text{out}}, \Delta X, \Delta Y, N_{\text{in}}], B_{\ell+1}[N_{\text{out}}], L_{\ell}[B, X, Y, N_{\text{in}}])$$

$$L_{\ell+1}[B, X, Y, N_{\text{out}}] = L_{\ell}[B, X, Y, N_{\text{in}}] + \tilde{L}_{\ell+1}[B, X, Y, N_{\text{out}}]$$

Capital letters indicate that complete tensors.

These equations require that the spacial dimension remains the same (stide 1) and $N_{\text{out}} = N_{\text{in}}$.

Residual Connections in CNNs

The residual connection typically skips over several layers, or in transformers, a complex multi-level unit.

$$
\begin{aligned}
& \tilde{L}_{\ell+1}[B, X, Y, N_{\text{out}}] \\
= \; & \text{Conv}(K_{\ell+1}[N, \Delta X, \Delta Y, N], B_{\ell+1}[N], L_{\ell}[B, X, Y, N])
\end{aligned}
$$

$$
\begin{aligned}
& \tilde{L}_{\ell+2}[B, X, Y, N] \\
= \; & \text{Conv}(K_{\ell+1}[N, \Delta X, \Delta Y, N], B_{\ell+1}[N], L_{\ell+1}[B, X, Y, N])
\end{aligned}
$$

$$
L_{\ell+2}[B, X, Y, N] = L_{\ell}[B, X, Y, N] + \tilde{L}_{\ell+2}[B, X, Y, N]
$$

Handling Spacial Reduction

Spacial reduction and neuron expansion is done without convolution.

$$L_{\ell+1}[b, x, y, j] = \begin{cases} L_{\ell}[b, s * x, s * y, j] & \text{for } j < N_{\ell} \\ 0 & \text{otherwise} \end{cases}$$

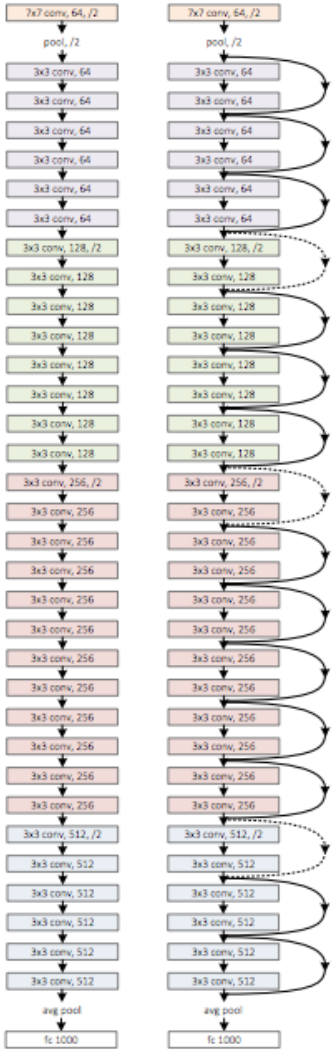
Residual connections are still placed around all convolutions.

Resnet32

plain net

ResNet

er)



[Kaiming He]

Deeper Versions use Bottleneck Residual Paths

We reduce the number of neurons to $N_{\text{bottle}} < N$ before doing the convolution.

$$
\begin{aligned}
U[B, X, Y, N_{\text{bottle}}] &= \text{Conv}(K^U[N_{\text{bottle}}, 1, 1, N_{\text{in}}], L_\ell[B, X, Y, N]) \\
V[B, X, Y, N_{\text{bottle}}] &= \text{Conv}(K^V[N_{\text{bottle}}, 3, 3, N_{\text{bottle}}], U[B, X, Y, N_{\text{bottle}}]) \\
R[B, X, Y, N] &= \text{Conv}(K^R[N, 1, 1, N_{\text{bottle}}], V[B, X, Y, N]) \\
L_{\ell+1} &= L_\ell + R
\end{aligned}
$$

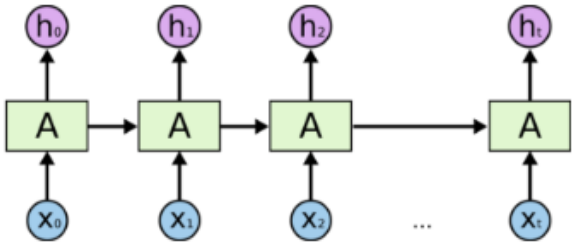
A General Residual Connection

$$y = x + R(x)$$

where $R(x)$ has the same shape as x.

Recurrent Neural Networks (RNNs)

Vanilla RNNs



$$h_{t+1}[b, n_{\text{out}}] = \sigma(W^{h,h}[n_{\text{out}}, N_{\text{in}}]h_t[N_{\text{in}}] + W^{h,x}[n_{\text{out}}, N_x]x_t[N_x] - B[n_{\text{out}}])$$

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.

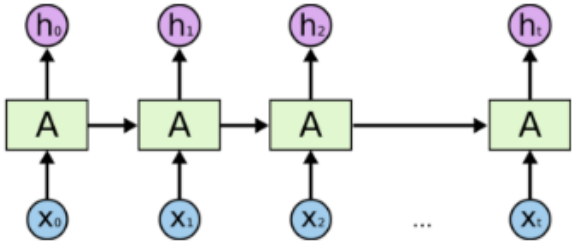
Exploding Gradients: Gradient Clipping

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

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

See `torch.nn.utils.clip_grad_norm`

Time as Depth



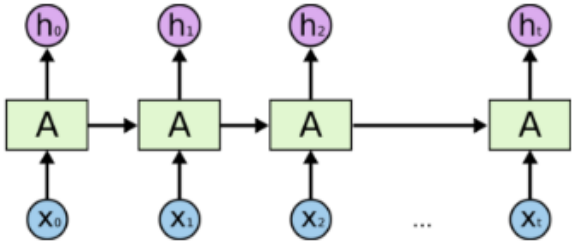
We would like the RNN to remember and use information from much earlier inputs.

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.

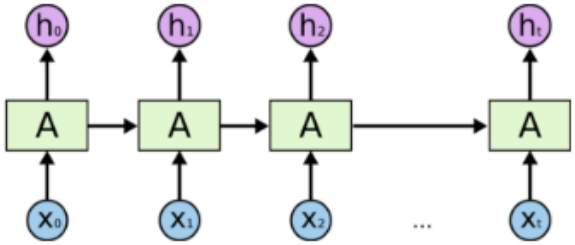
Skip Connections Through Time



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)



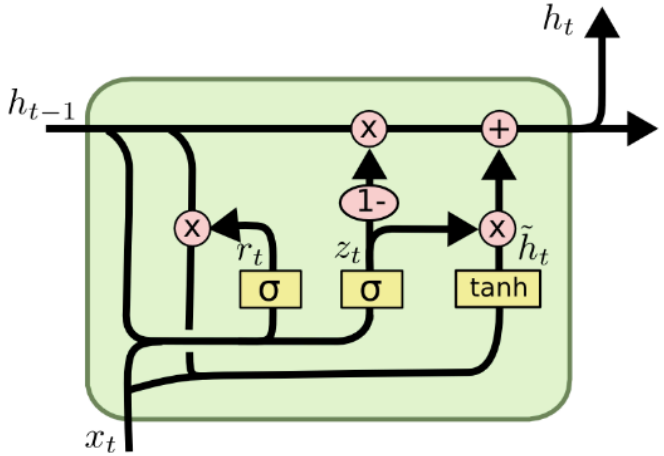
$$R_t[b, n_{\text{out}}] = \tanh(W^{h,R}[n_{\text{out}}, N_{\text{in}}]h_t[b, N_{\text{in}}] + W^{x,R}[n_{\text{out}}, N_{\text{in}}]x_t[b, N_{\text{in}}] - B^R[n_{\text{out}}])$$

$$G_t[b, n_{\text{out}}] = \sigma(W^{h,G}[n_{\text{out}}, N_{\text{in}}]h_t[b, N_{\text{in}}] + W^{x,G}[n_{\text{out}}, N_{\text{in}}]x_t[b, N_{\text{in}}] - B^G[n_{\text{out}}])$$

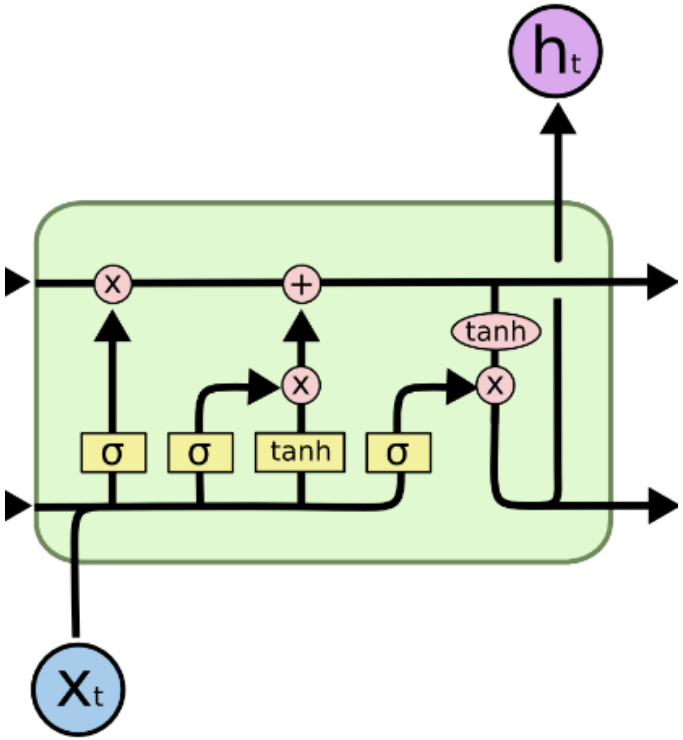
$$h_{t+1}[b, n] = G_t[b, n]h_t[b, n] + (1 - G_t[b, n])R_t[b, n]$$

$$h_{t+1}[b, N] = G_t[b, N] \odot h_t[b, N] + (1 - G_t[b, N]) \odot R_t[b, N]$$

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



Long Short Term Memory (LSTM)



[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

ReLu

Initialization

Normalization

Residual Connections

Gated RNNs

END