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

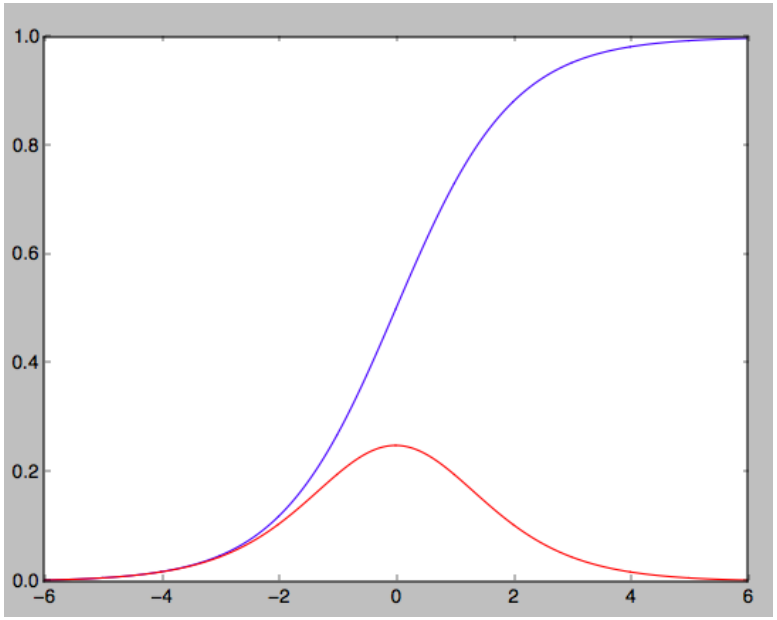
Causes of Vanishing and Exploding Gradients:

Activation function saturation

Repeated multiplication by network weights

Activation Function Saturation

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

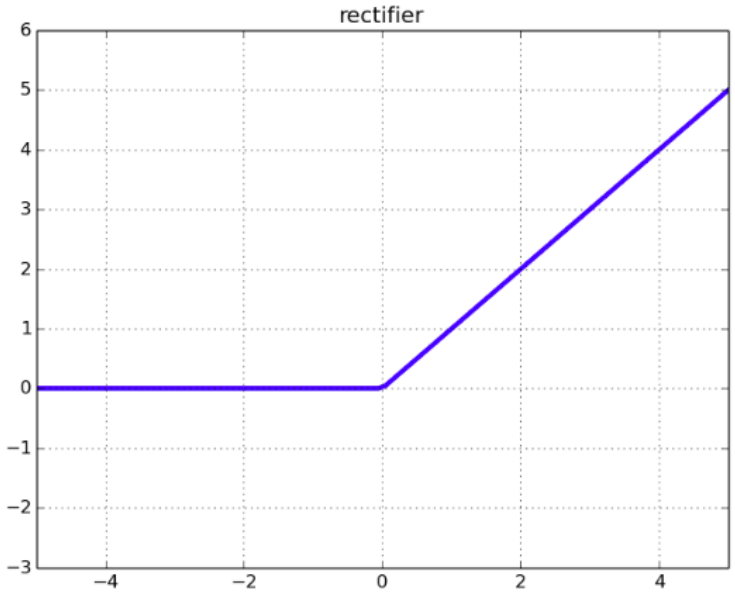


The gradient of this function is quite small for $|x| > 4$.

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.

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.

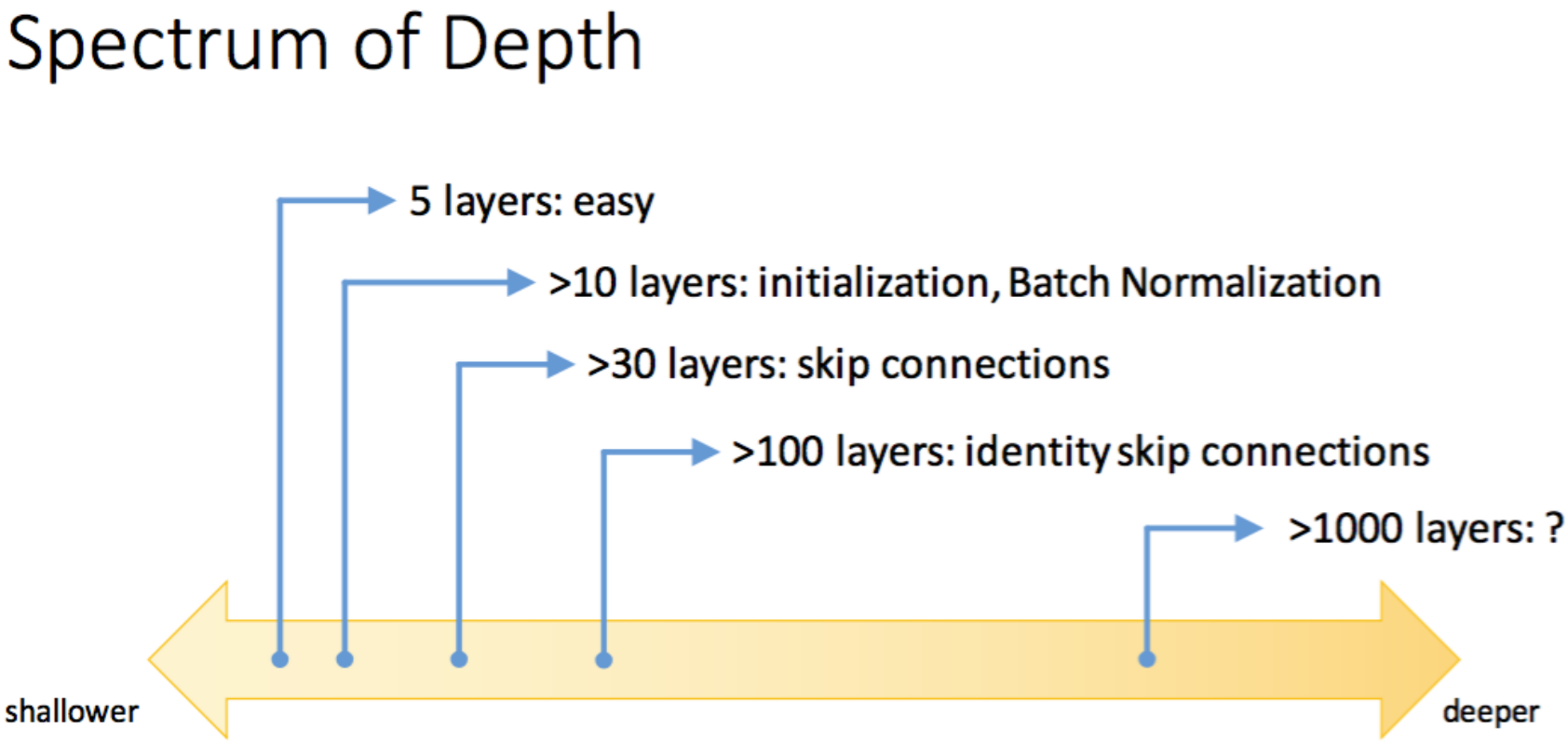
Methods for Maintaining Gradients

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.

EDF Implementation

```
def xavier(shape):
    sq = np.sqrt(3.0/np.prod(shape[:-1]))
    return np.random.uniform(-sq,sq,shape)
```

This assumes that we sum over all but the last index.

For example, an image convolution filter has shape $(W,\ W,\ C_1, C_2)$ and we sum over the first three indices.

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 $x[b, c]$ we define $\tilde{x}[b, c]$ as follows.

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

$$\hat{\mu}[c_x] = \frac{1}{B} \sum_b x[b, c]$$

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

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

Spatial Batch Normalization

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

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

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

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

Backpropagation Through Normalization

$$\tilde{x}[b,i,j,c_x] \;=\; (x[b,i,j,c_x] - \hat{\mu}[c_x])/\hat{\sigma}[c_x]$$

$$x.\mathrm{grad}[b,i,j,c_x] \;\texttt{+=}\; \tilde{x}.\mathrm{grad}[b,i,j,c_x]/\hat{\sigma}[c_x]$$

$$\hat{\mu}.\mathrm{grad}[c_x] \;\texttt{-=}\; \tilde{x}.\mathrm{grad}[b,i,j,c_x]/\hat{\sigma}[c_x]$$

$$\hat{\sigma}.\mathrm{grad}[c_x] \;\texttt{-=}\; \tilde{x}.\mathrm{grad}[b,i,j,c_x](x[b,i,j,c_x] - \hat{\mu}[c_x])/\hat{\sigma}[c_x]^2$$

Backpropagation Through Normalization

$$\hat{\mu}[c_x] \;=\; x[b, i, j, c_x]/(BIJ)$$

$$x.\text{grad}[b, i, j, c_x] \;\texttt{+=}\; \hat{\mu}.\text{grad}[c_x]/(BIJ)$$

Backpropagation Through Normalization

$$\hat{\sigma}[c_x] \;=\; \sqrt{\hat{s}[c_x]}$$

$$\hat{s}.\mathrm{grad}[c_x] \;\texttt{+=}\; \hat{\sigma}.\mathrm{grad}[c_x]/(2\hat{\sigma}[c_x])$$

$$\hat{s}[c_x] \;=\; \frac{1}{BIJ-1}(x[b,i,j,c_x]-\hat{\mu}[c_x])^2$$

$$x.\mathrm{grad}[b,i,j,c_x] \;\texttt{+=}\; \hat{s}.\mathrm{grad}[c_x]\,\frac{1}{BIJ-1}\,2(x[b,i,j,c_x]-\hat{\mu}[c_x])$$

$$\hat{\mu}.\mathrm{grad}[c_x] \;\texttt{-=}\; \hat{s}.\mathrm{grad}[c_x]\,\frac{1}{BIJ-1}\,2(x[b,i,j,c_x]-\hat{\mu}[c_x])$$

Adding an Affine Transformation

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

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

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

It can even undo the normaliztion.

Batch Normalization

Batch Normalization is is empirically successful in CNNs.

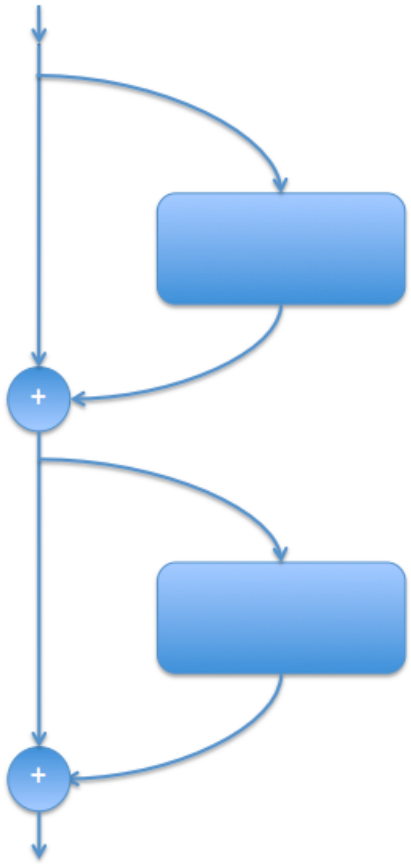
Not so successful in RNNs.

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



Here we have a "highway" with "diversions".

The highway path connects input to outputs and preserves gradients.

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

The resnet that won the Imagenet competition in 2015 had 152 diversions.

Residual Skip Connections

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

or

$$L \texttt{ += } D_i$$

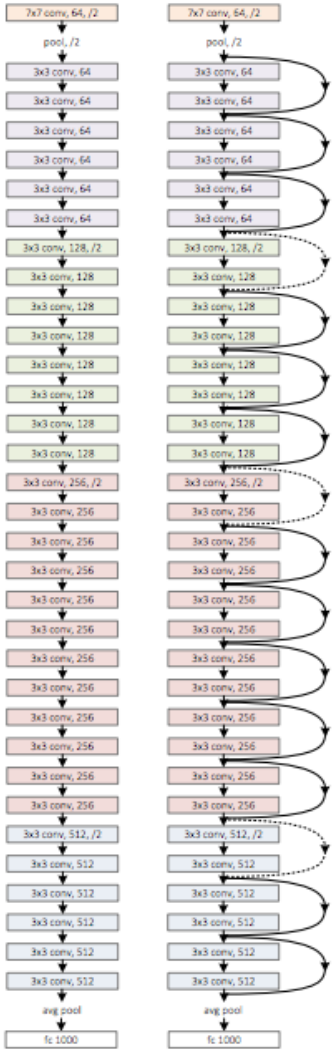
Here D_i "fits the residual of the identity function"

Resnet32

plain net

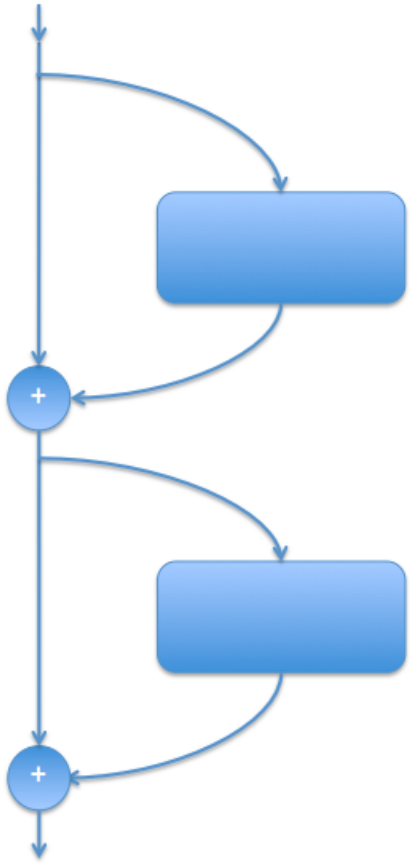
ResNet

er)



[Kaiming He]

Deep Residual Networks

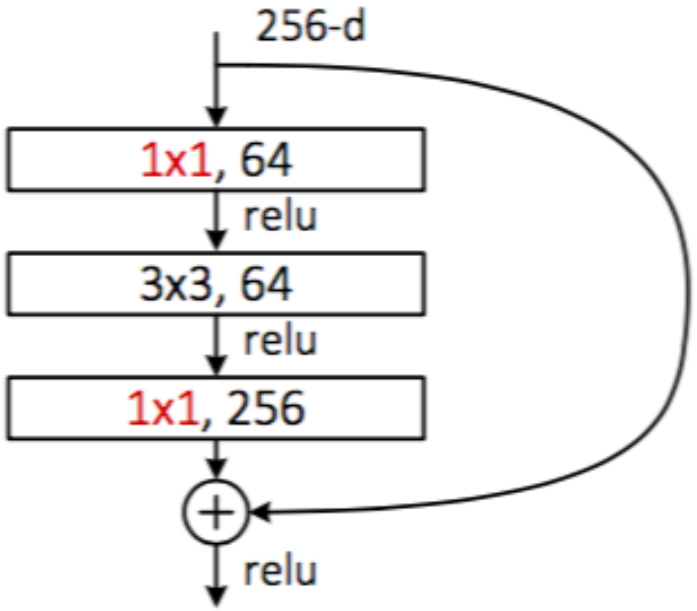


As with most of deep learning, not much is known about what resnets are actually doing.

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

They are capable of representing very general circuit topologies.

A Bottleneck Diversion



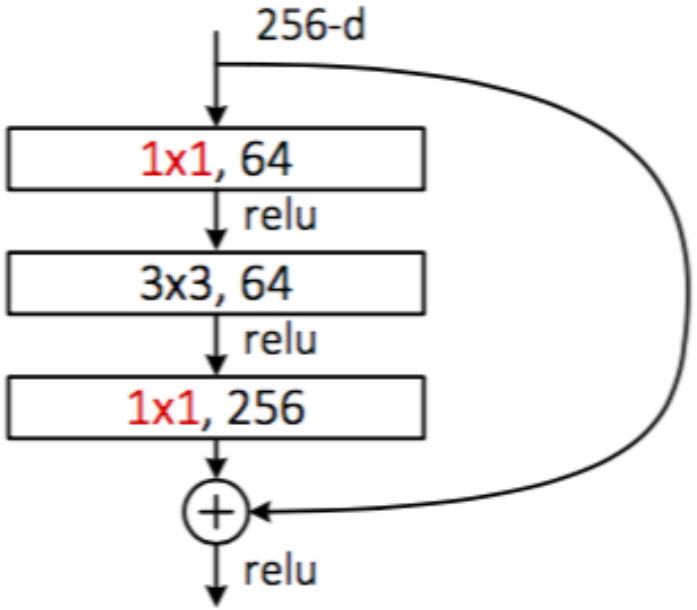
bottleneck

(for ResNet-50/101/152)

[Kaiming He]

This reduction in the number of channels used in the diversion suggests a modest update of the highway information.

Expressive Power



bottleneck

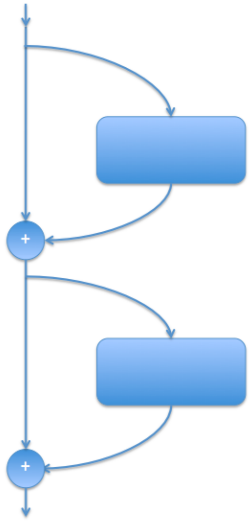
(for ResNet-50/101/152)

[Kaiming He]

This architecture can express fairly arbitrary state updates.

Each layer has data-flow parameters.

Gating gives Data-Dependent Data Flow



Residual: $y^{\ell+1} = y^{\ell} + d^{\ell}$

LSTM: $h^{t+1} = f^t \odot h^t + i^t \odot d^t$

GRU: $h^{t+1} = f^t \odot h^t + (1 - f^t) \odot d^t$

Resnet has data-flow parameters at each layer.

Gated RNNs use the same parameters at each time step but use gating for data-flow control.

Recurrent Neural Networks (RNNs)

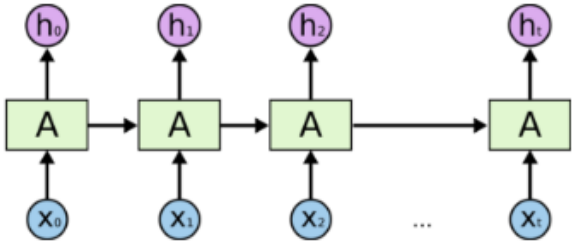
Speech Recognition

Machine Translation

Reading Comprehesion

Language Modeling

Vanilla RNNs



[Christopher Olah]

$$h[b, t{+}1, c_h] = \tanh(W[c'_h, c_h]h[b, t, c'_h]{+}W[c_x, c_h]x[b, t, c_x]{+}\beta[c_h])$$

or

$$h^{t+1} = \tanh(W^{h,h}h^t + W^{x,h}x^t + \beta)$$

or

$$h^{t+1} = \tanh(W^h[h^t, x^t] + \beta)$$

where $[x, y]$ denotes concatenation.

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: Highway Paths (Skip Connections)

Modern RNNs have highway paths (skip connections).

Unlike deep CNN layers such as Resnet, RNNs use the same parameters at each layer.

Probably for this reason, pure residual connections are not used.

Instead gating is used for data-dependent weighting.

Skip Connections

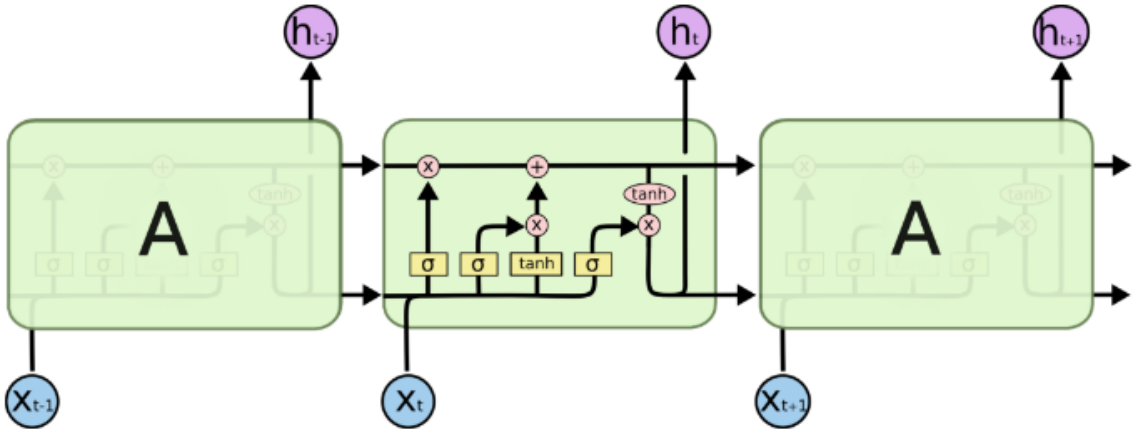
Residual: $L_{i+1} = L_i + D_i$

Forget Gates (LSTM): $L_{i+1} = F_i \odot L_i + I_i \odot D_i$

Convex Gates (GRU): $L_{i+1} = F_i \odot L_i + (1 - F_i) \odot D_i$

$\odot$ is Hadamard product. This is the same as NumPy element-wise product. However, the symbol $\odot$ is commonly used in the literature.

Long Short Term Memory (LSTM)



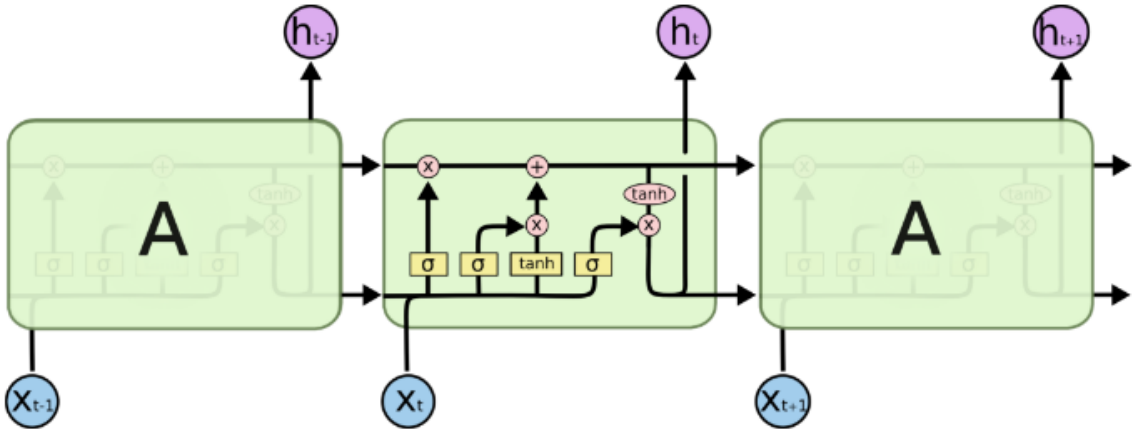
[figure: Christopher Olah]

[LSTM: Hochreiter&Shmidhuber, 1997]

$$c^{t+1} = f^t \odot c^t + i^t \odot d^t$$

Note that if $f^t = i^t = 1$ then we have a residual connection.

Long Short Term Memory (LSTM)



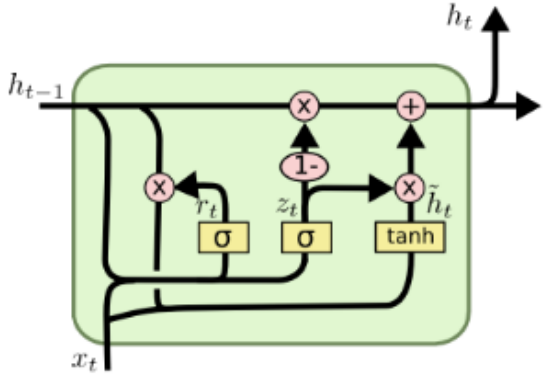
[Christopher Olah]

$$c^{t+1} = f^t \odot c^t + i^t \odot d^t$$

$$\begin{aligned} f^t &= \sigma(W^f[X^t, h^t] + b^f) \\ i^t &= \sigma(W^i[X^t, h^t] + b^i) \\ o^t &= \sigma(W^o[X^t, h^t] + b^0) \end{aligned}$$

$$\begin{aligned} d^t &= \tanh(W^d[X^t, h^t] + b^d) \\ h^{t+1} &= o^t \odot \tanh(c^{t+1}) \end{aligned}$$

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



[Christopher Olah]

$$h^{t+1} = f^t \odot h^t + (1 - f^t) \odot d^t$$

$$\begin{aligned} f^t &= \sigma(W^f[x^t, h^t] + b^f) \\ r^t &= \sigma(W^r[x^t, h^t] + b^r) \end{aligned}$$

$$d^t = \tanh(W^{x,d}x^t + r^t \odot (W^{h,d}h^t + b^{h,d}) + b^d)$$

GRUs vs. LSTMs

The GRU is simpler than the LSTM.

In TTIC31230 class projects 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.

Update Gate RNN (UGRNN)

$$h^{t+1} = f^t \odot h^t + (1 - f^t) \odot d^t$$

$$f^t = \sigma(W^f[x^t, h^t] + b^f)$$

$$d^t = \tanh(W^d[x^t, h^t] + b^d)$$

END