# TTIC 31230, Fundamentals of Deep Learning

David McAllester, Winter 2019

Controlling Gradients

Vanishing and Exploding Gradients

Initialization

Batch Normalization

Residual Networks

Gated RNNs

## The Expressive Power of DNNs

Linear Threshold units generalize logical operations.

Consider Boolean Values $P, Q$ — numbers that are either close to 0 or close to 1.

$$P \wedge Q \approx \sigma(100 * P + 100 * Q - 150)$$

$$P \vee Q \approx \sigma(100 * P + 100 * Q - 50)$$

$$\neg P \approx \sigma(100 * (1 - P) - 50)$$

DNNs generalize digital circuits.

## 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).

The Universality assumption is clearly false but can still guide architecture design.

But equally expressive architectures differ in trainability. This lecture is on trainability.

## Vanishing and Exploding Gradients

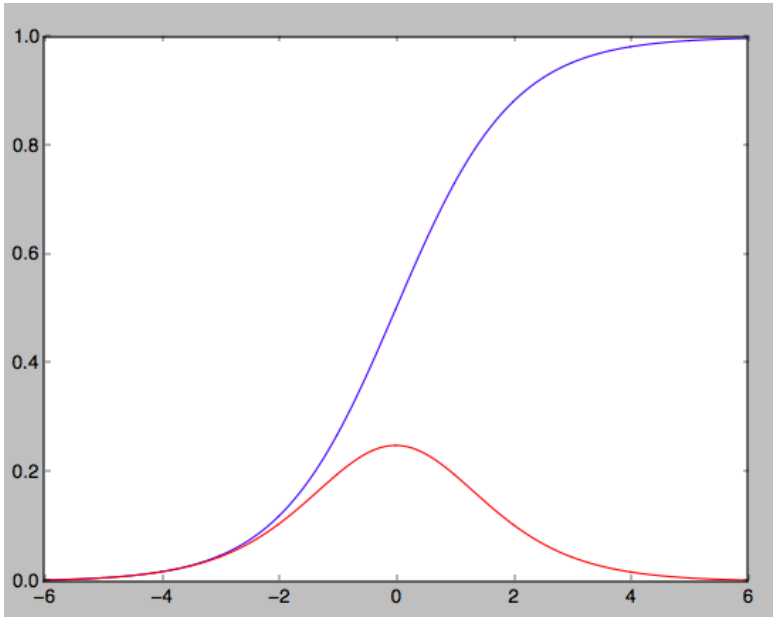
Causes of Vanishing and Exploding Gradients:

Activation function saturation

Repeated multiplication by network weights

## Activation Function Saturation

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

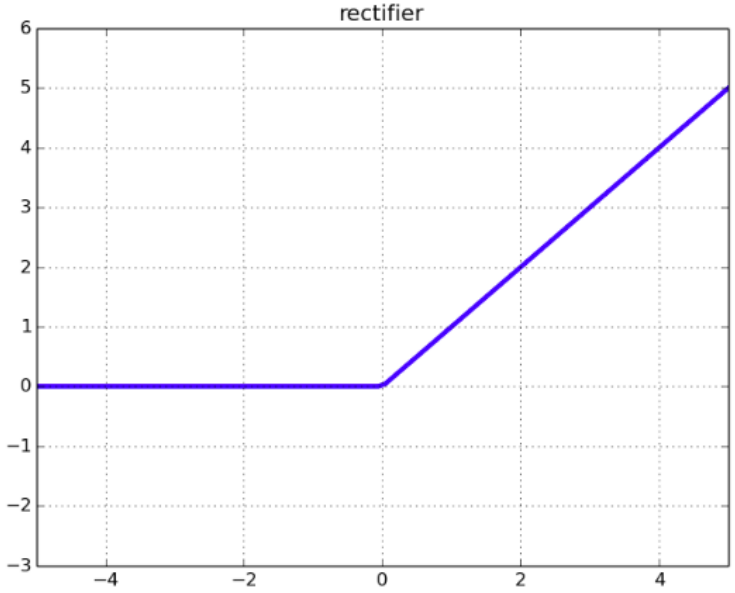



The gradient of this function is quite small for $|x| > 4$.

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.**

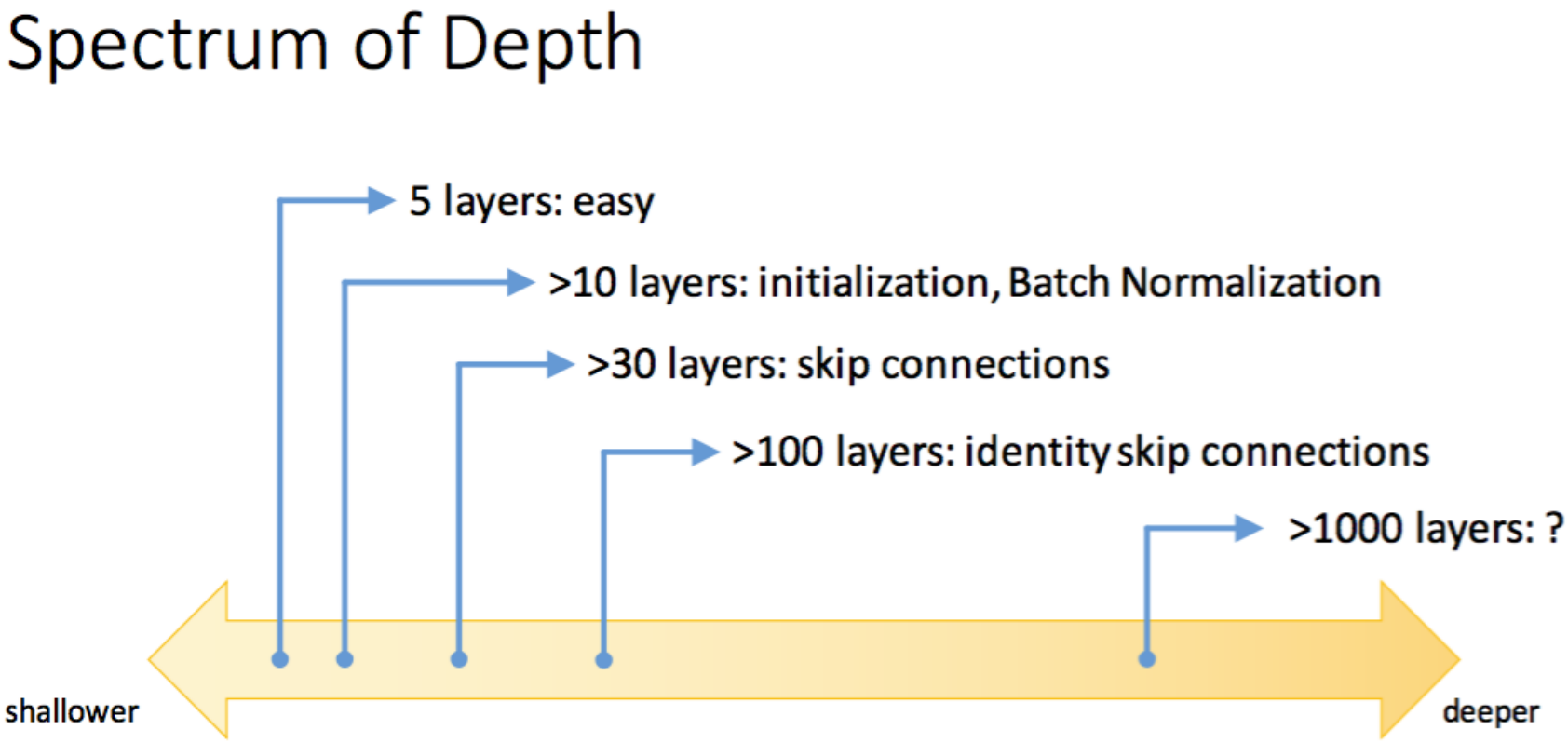
## Methods for Maintaining Gradients

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.

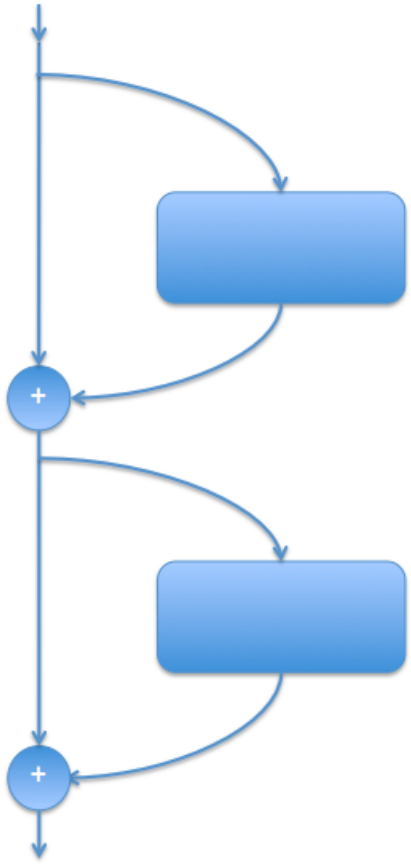
Not so successful in RNNs.

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)

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

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 indeces to denote entire tensors and lower case letters for particular indeces.

## Simple Residual Skip Connections in CNNs (stride 1)

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

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 $\tilde{L}_{\ell+1}[B, X_{\ell+1}, Y_{\ell+1}, J_{\ell+1}]$

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

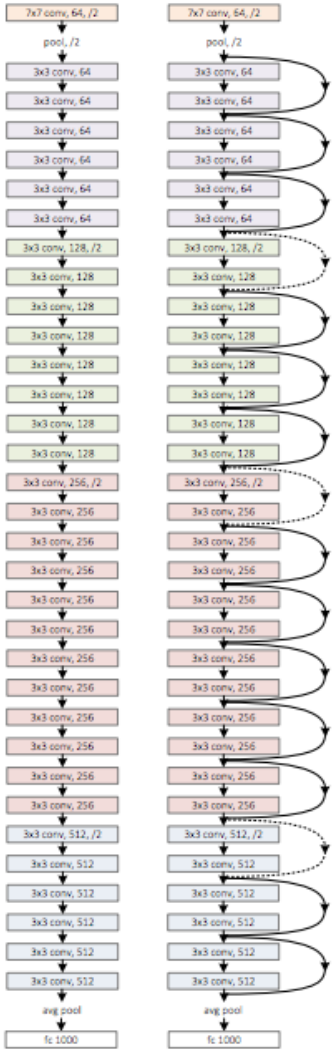
$$\begin{aligned} L_{\ell+1}[B, X_{\ell+1}, Y_{\ell+1}, J_{\ell+1}] = \; & \tilde{L}_{\ell+1}[B, X_{\ell+1}, Y_{\ell+1}, J_{\ell+1}] \\ & + R_{\ell+1}[B, X_{\ell+1}, Y_{\ell+1}, J_{\ell+1}] \end{aligned}$$

# Resnet32

plain net

ResNet

er)




[Kaiming He]

## Deeper Versions use Bottleneck Residual Paths

We reduce the number of channels to $K < J_\ell$ before doing the convolution.

$$A_\ell[B, X_\ell, Y_\ell, K] = \text{Conv}'(\Phi_\ell{}^A[1, 1, J_\ell, K], L_\ell[B, X_\ell, Y_\ell, J_\ell])$$

$$B_\ell[B, X_{\ell+1}, Y_{\ell+1}, K] = \text{Conv}'(\Phi_\ell{}^B[3, 3, K, K], A_\ell[B, X_\ell, Y_\ell, K], \text{ stride } s)$$

$$R_{\ell+1}[B, X_{\ell+1}, Y_{\ell+1}, J_{\ell+1}] = \text{Conv}'(\Phi_\ell{}^R[1, 1, K, J_{\ell+1}], B_\ell[B, X_{\ell+1}, Y_{\ell+1}, K])$$

$$L_{\ell+1} = \tilde{L}_{\ell+1} + R_{\ell+1}$$

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

# A General Resdual Connection

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

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

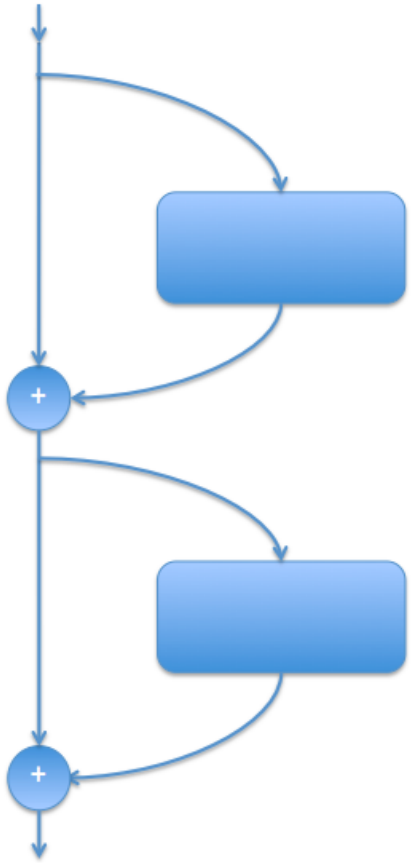
# DenseNet

For $u[I]$ and $v[J]$ we let $(u; w)[I + J]$ denote vector concatenation.

$$(u; v)[k] - \begin{cases} u[k] \text{ for } k < J \\ v[k - I] \text{ otherwise} \end{cases}$$

for $b, x, y$ $L_{\ell+1}[b, x, y, J_\ell + J_R] = L_\ell[b, x, y, J_\ell]; R[b, x, y, J_R]$

## Deep Residual Networks



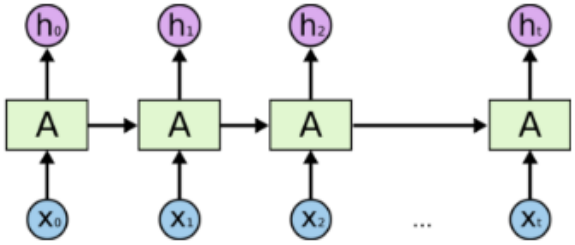
As with most of deep learning, not much is known about what resnets are actually doing.

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



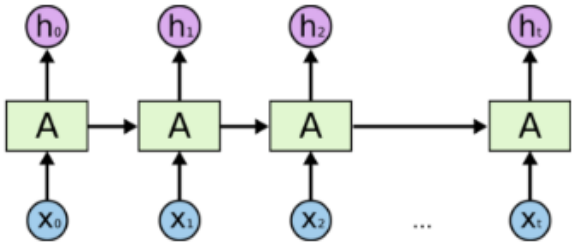

[Christopher Olah]

We use two input linear threshold units.

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

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

# 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.

## 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.

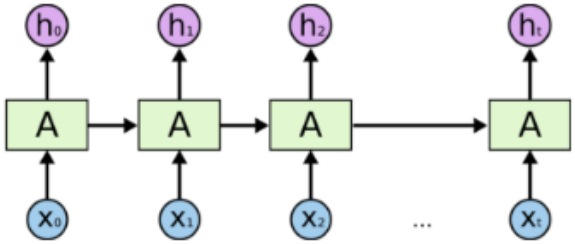
## Exploding Gradients: Gradient Clipping

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

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

See `torch.nn.utils.clip_grad_norm`

# Skip Connections Through Time




[Christopher Olah]

We would like to add residual connections through time.

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

## Gated Skip Connections

$$\text{Residual Skip: } L_{\ell+1} = \tilde{L}_\ell + R_{\Phi_{\ell+1}}(L_\ell)$$

Here $\Phi_{\ell+1}$ controls data independent data flow.

$$\text{Gated Skip: } h_{t+1} = G_t \odot h_t + (1 - G_t) \odot R_\Phi(x_t, h_t)$$

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

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

Here Gating allows data dependent data flow.

# Update Gate RNN (UGRNN)

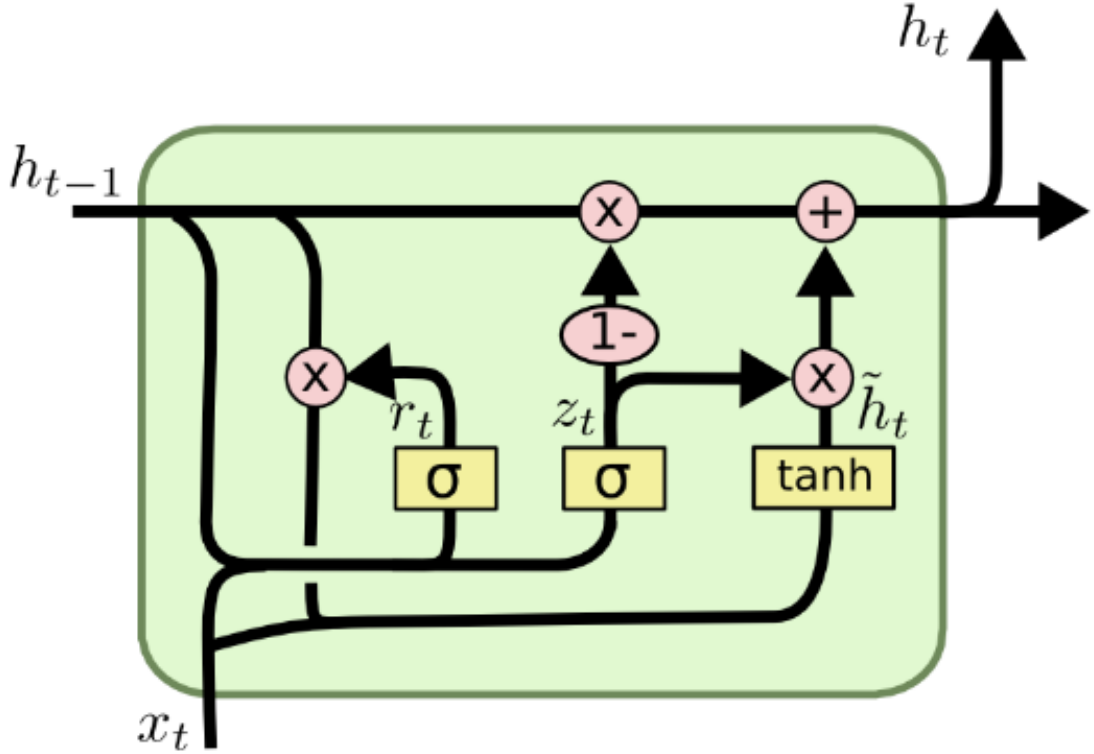
$$R_t[b,j] = \tanh\left(\left(\sum_i W^{h,R}[i,j]h_t[b,i]\right) + \left(\sum_k W^{x,R}[k,j]x_t[b,k]\right) - B^R[j]\right)$$

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

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

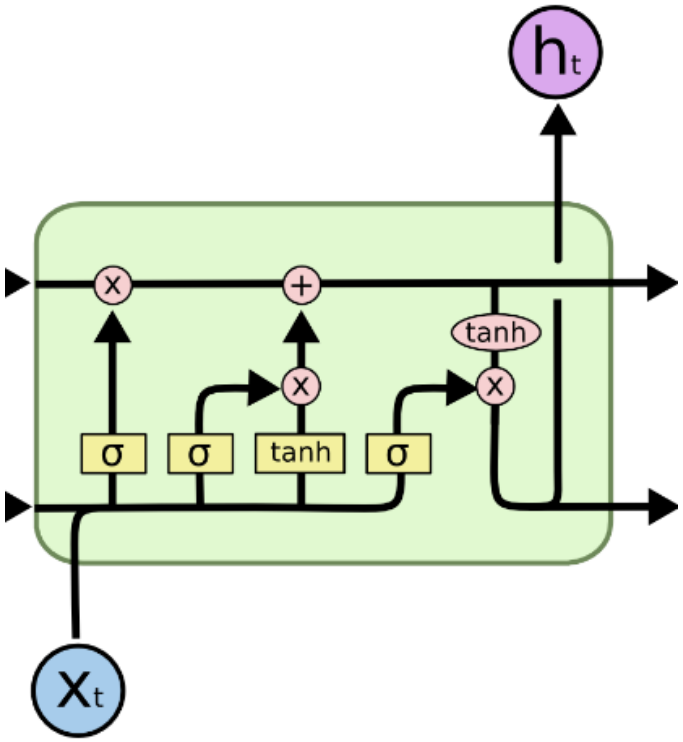
$$\Phi = (W^{h,R}, W^{x,R}, \beta^R, W^{h,G}, W^{x,G}, \beta^G)$$

# 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