TTIC 31230, Fundamentals of Deep Learning

David McAllester, April 2017

Vanishing and Exploding Gradients

ReLUs

Xavier Initialization

Batch Normalization

Highway Architectures: Resnets, LSTMs and GRUs

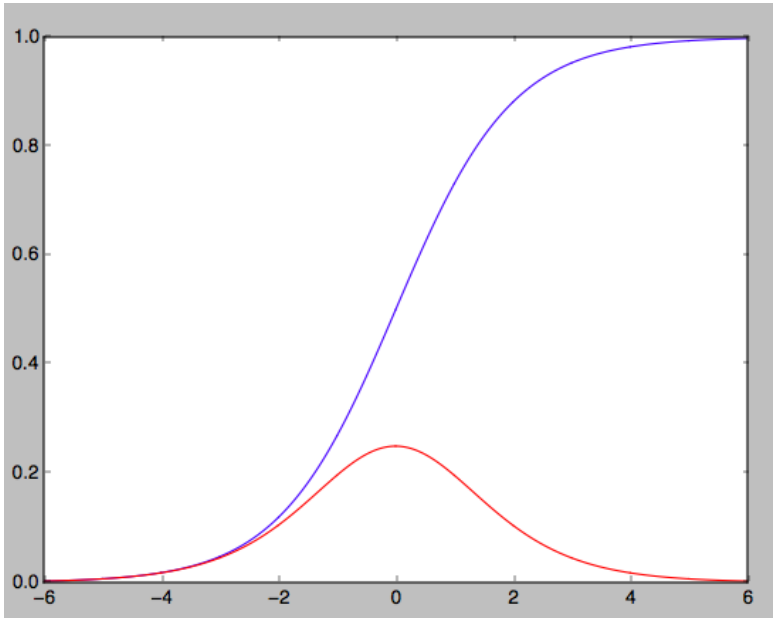
Causes of Vanishing and Exploding Gradients

Activation function saturation

Repeated multiplication by network weights

Activation Function Saturation

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

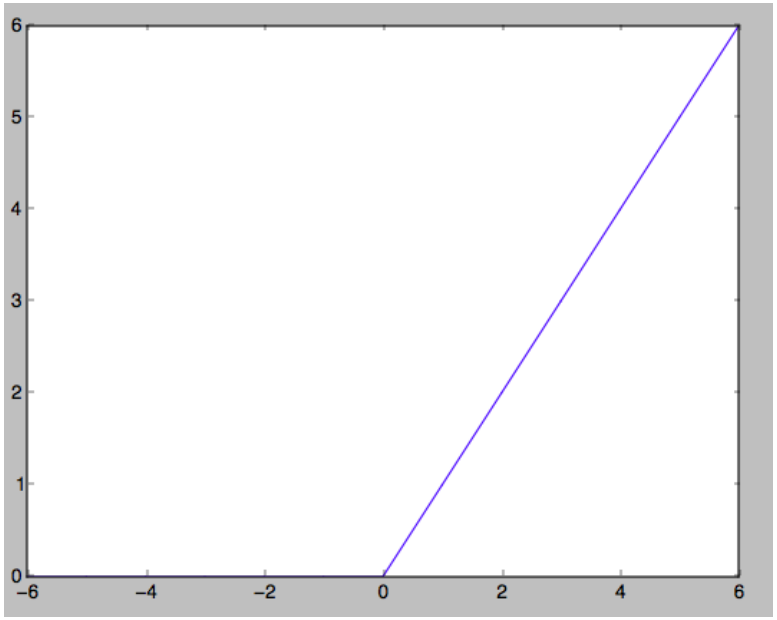


The gradient of this function is quite small for $|x| > 4$.

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.

$$L_{i+1} = \text{Relu}(\text{Conv}(L_i, F_i))$$

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.

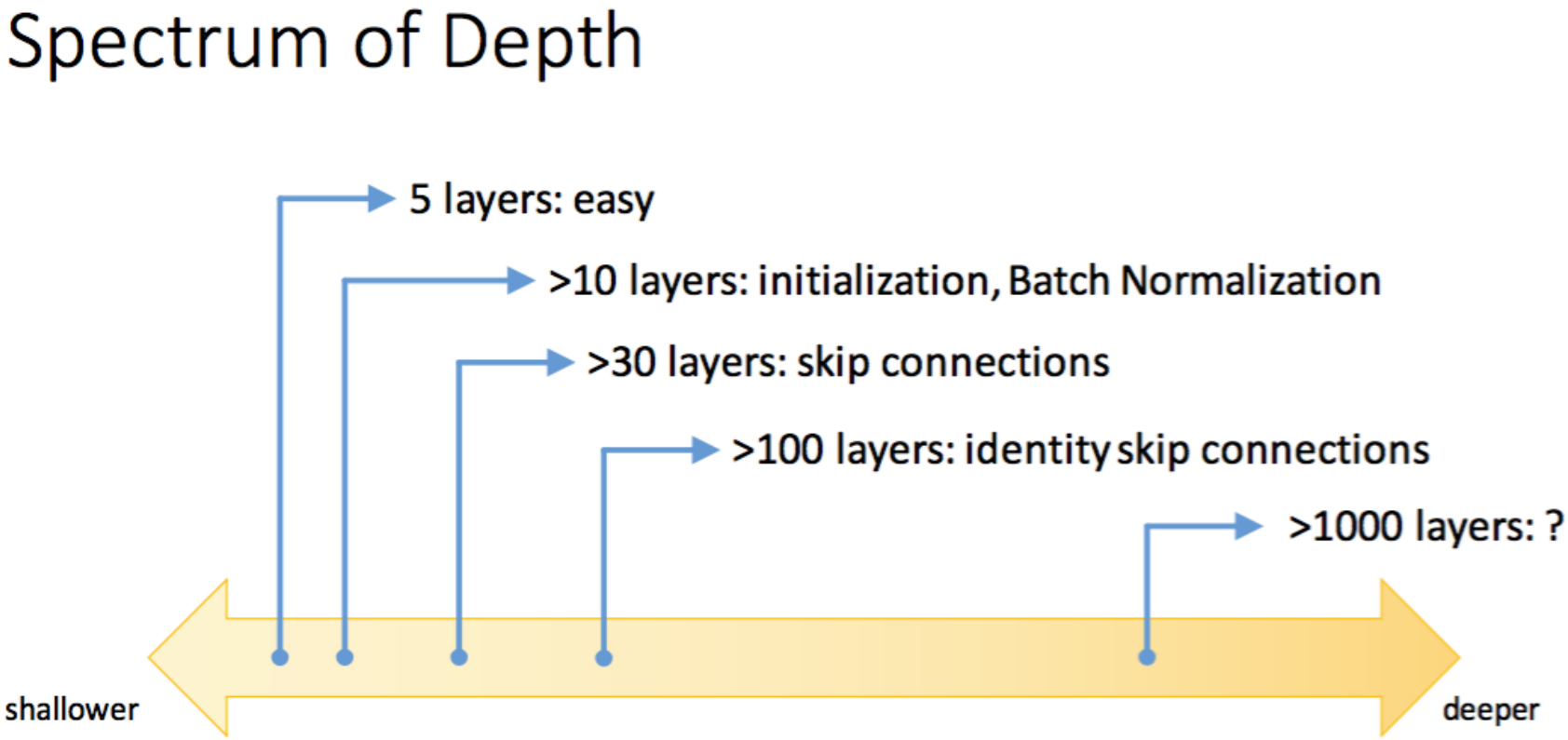
Methods for Maintaining Gradients

Initialization

Batch Normalization

Highway Architectures (Skip Connections)

Methods for Maintaining Gradients



[Kaiming He]

We will say "highway architecture" rather than "skip connections".

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 $(H,\ W,\ C_1, C_2)$ and we sum over the first three indices.

Kaiming 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

We first compute the mean and variance over the batch for each channel and renormalizes the channel value.

$$\text{Norm}(x) = \frac{x - \hat{\mu}}{\hat{\sigma}}$$

$$\hat{\mu} = \frac{1}{B}\sum_{b} x_b$$

$$\hat{\sigma} = \sqrt{\frac{1}{B-1}\sum_{b}(x_b - \hat{\mu})^2}$$

Backpropagation Through Normalization

Note that we can backpropgate through the normalization operation. Here the different batch elements are not independent.

At test time a single fixed estimate of μ and σ is used.

Adding an Affine Transformation

$$\text{BatchNorm}(x) = \gamma \text{Norm}(x) + \beta$$

Keep in mind that this done seperately for each channel.

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 empirically very successful in CNNs.

Not so successful in RNNs (RNNs are discussed below).

It is typically used just prior to a nonlinear activation function.

It was originally justified in terms of "covariant shift".

However, it can also be justified along the same lines as Xavier initialization — we need to keep the input to an activation function in the active region.

Normalization Interacts with SGD

Consider backpropagation through a weight layer.

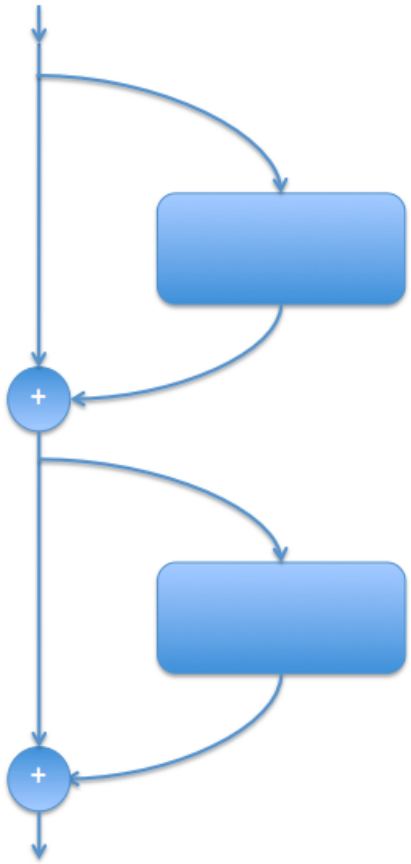
$$y.\text{value}[\cdots] \texttt{ += } w.\text{value}[\cdots]\, x.\text{value}[\ldots]$$

$$w.\text{grad}[\cdots] \texttt{ += } y.\text{grad}[\cdots]\, x.\text{value}[\cdots]$$

Replacing x by $x/\hat{\sigma}$ seems related to RMSProp for the update of w.

Highway Architectures

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.

Pure and Gated Highway Architectures

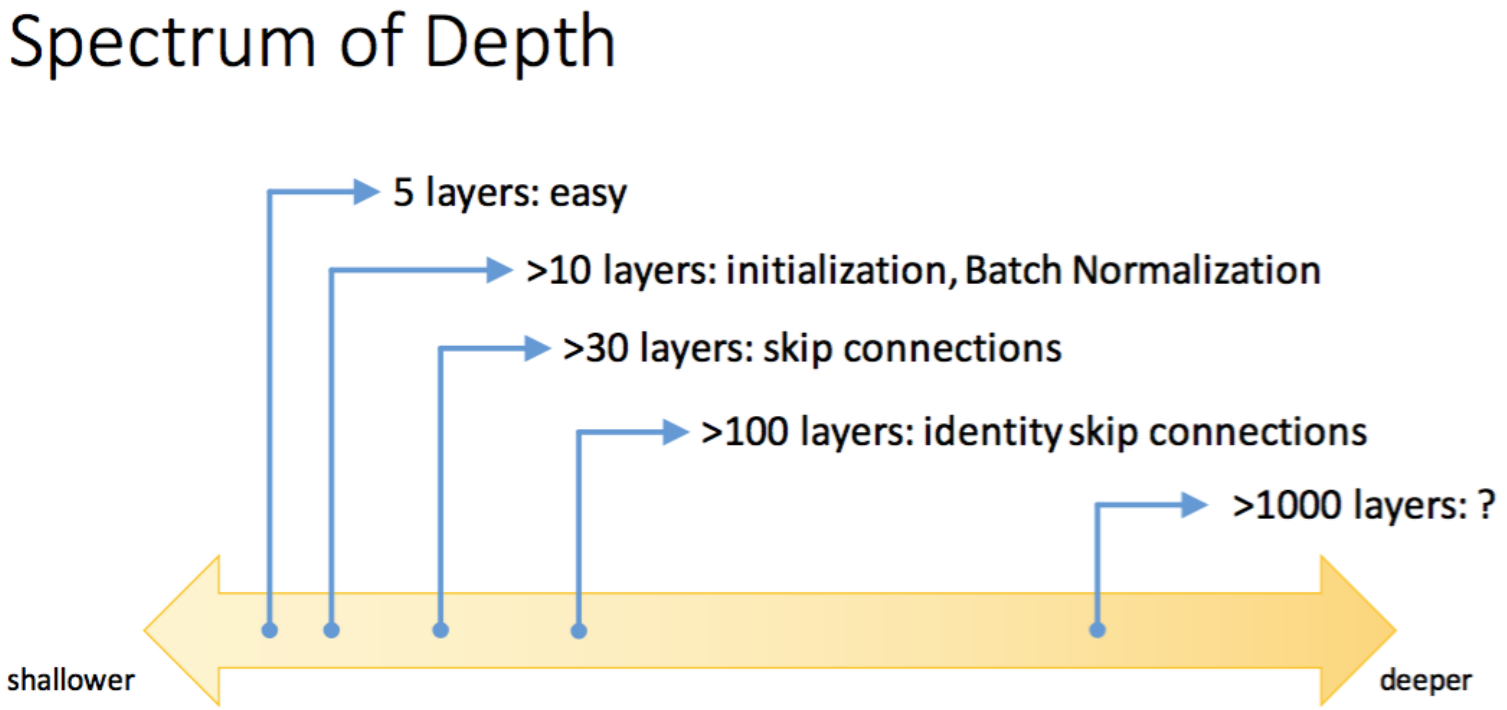
Pure (Resnet): $L_{i+1} = L_i + D_i$

Forget Gated (LSTM): $L_{i+1} = F_i * L_i + D_i$

Exclusively Gated (GRU): $L_{i+1} = F_i * L_i + (1 - F_i) * D_i$

In the pure case the "diversion" D_i "fits the residual of the identity function"

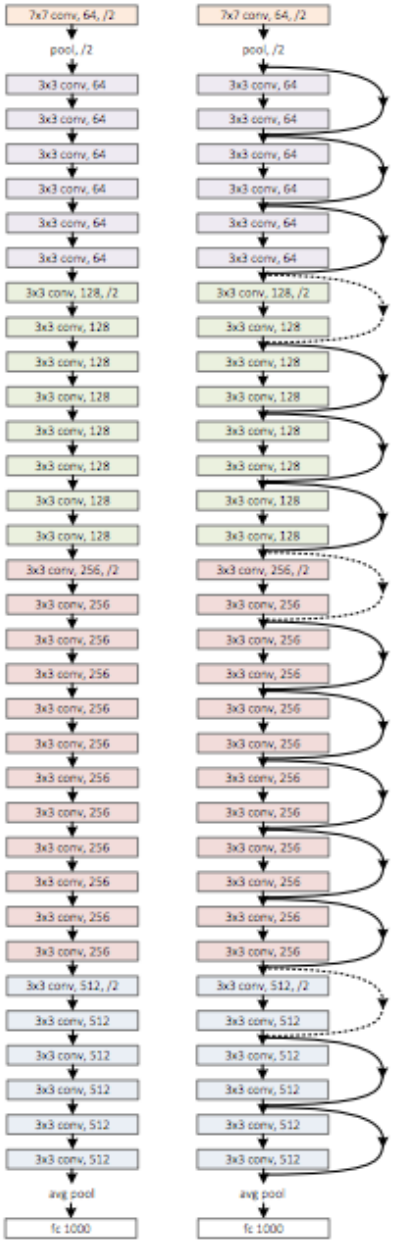
Methods for Maintaining Gradients



[Kaiming He]

Identity skip connections give a pure highway architecture.

plain net

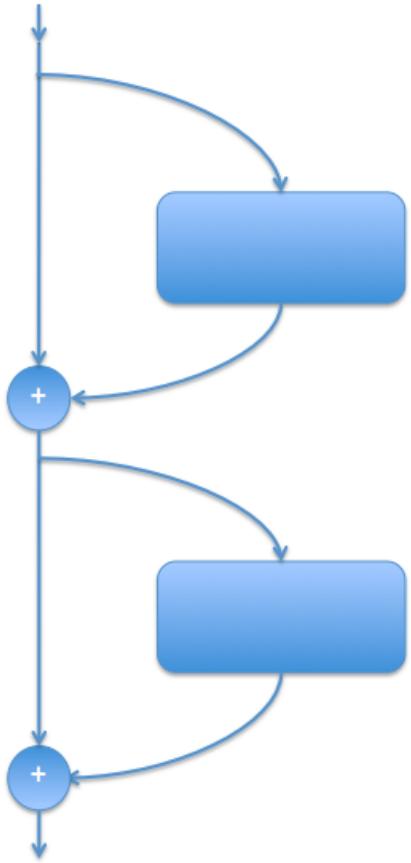


ResNet

er)

[Kaiming He]

Deep Residual Networks

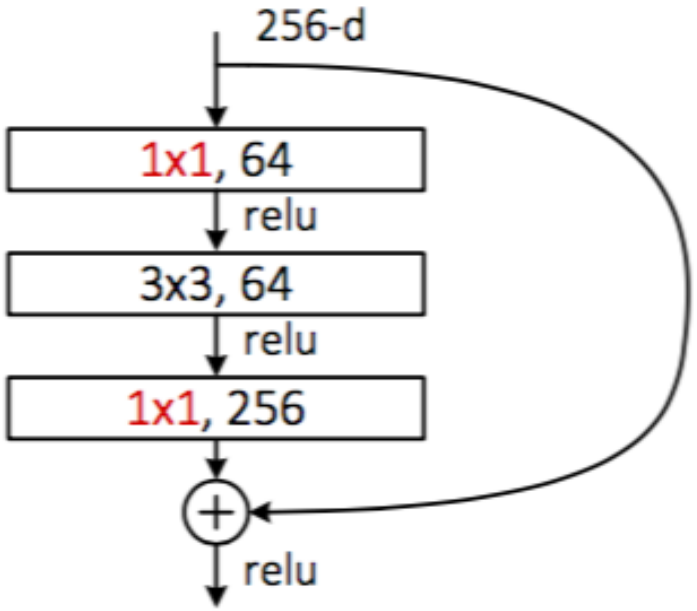


As with most of deep learning, not much is known about what resnets are actually doing.

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



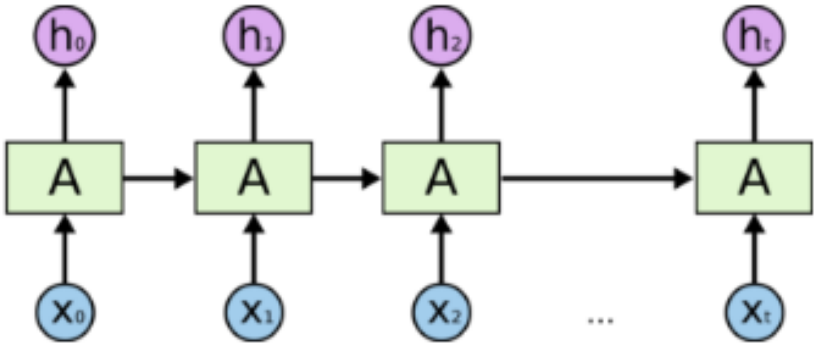
bottleneck
(for ResNet-50/101/152)

[Kaiming He]

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

Long Short Term Memory (LSTMs)

Recurrent Neural Networks



[Christopher Olah]

$$h_{t+1} = \tanh(W_h\, h_t + W_x x_t + \beta)$$

In EDF:

```
for t in range(T):
 H.append(None)
 H[t+1] = Sigmoid(ADD(VDot(Wh,H[t]), VDot(Wx,X[t]), Beta)
```

Gradient Clipping

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.

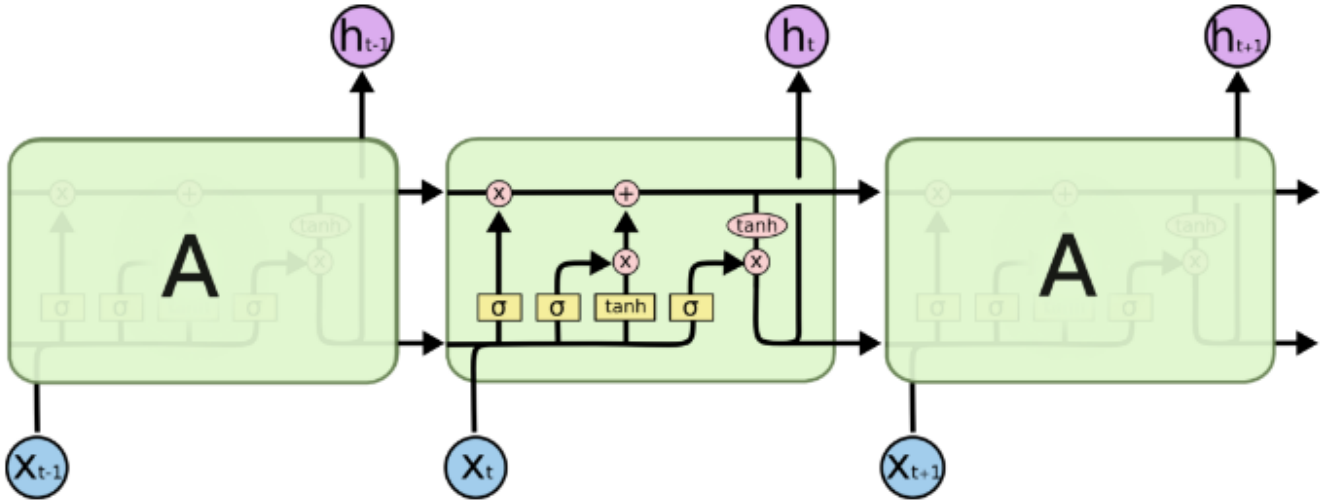
To handle exploding gradients we can define a gradient clipping component.

$$h_{t+1} = \text{clip}(\tanh(W_h\ h_t + W_x x_t + b))$$

The forward method of the clip component is the identity function.

The backward method shrinks the gradients to stay within some limit.

Long Short Term Memory (LSTM)



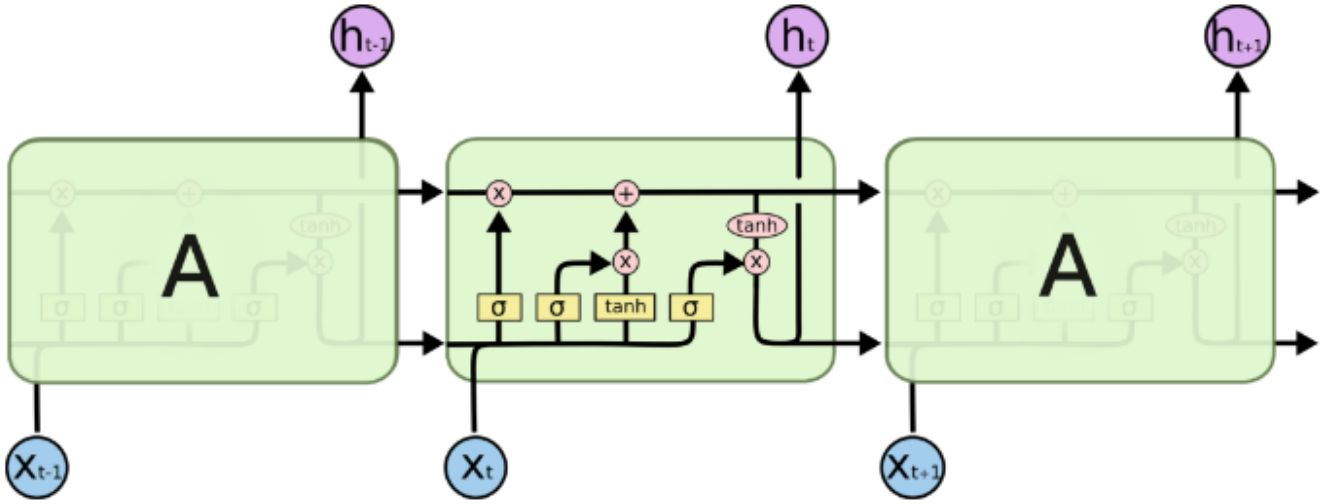
[Christopher Olah]

The highway path goes across the top and is called the Carry:

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

LSTM: $$C_{t+1} = F_t * C_t + D_t$$

Long Short Term Memory (LSTM)



[Christopher Olah]

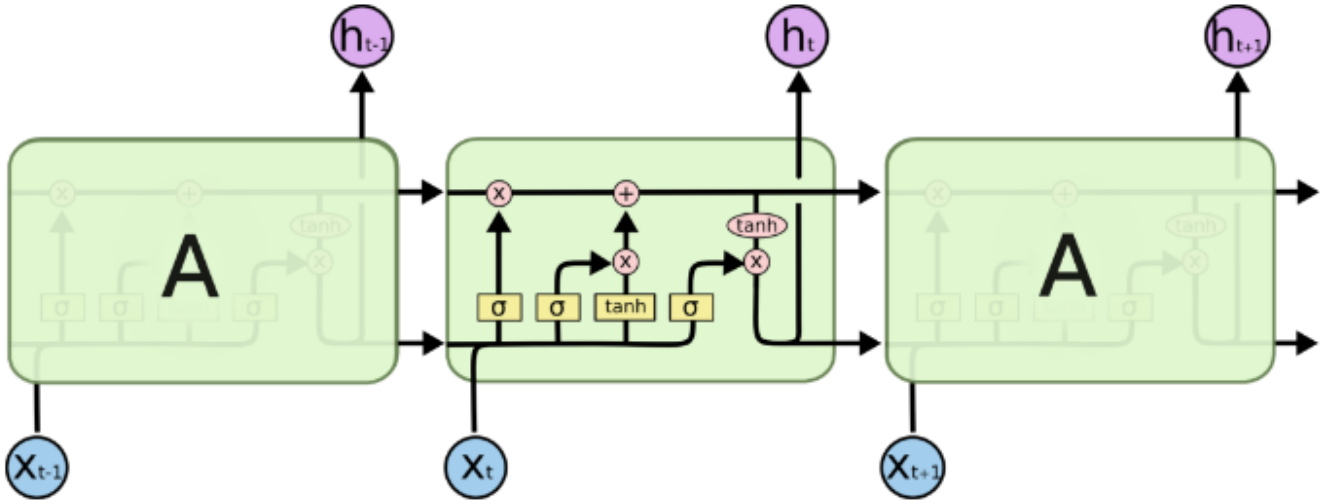
The highway path C (across the top of the figure) is gated.

$C_{t+1} = F_t * C_t + D_t$

The product $F * C$ is componentwise (as in Numpy).

We say F gates C. F is called the "forget gate".

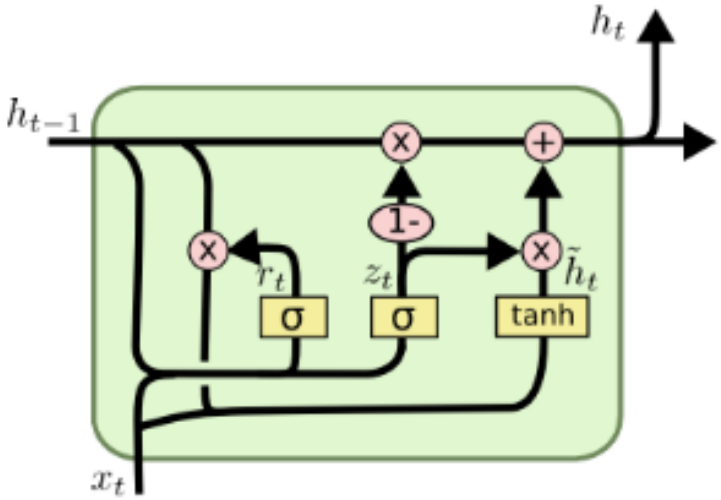
Long Short Term Memory (LSTM)



[Christopher Olah]

$$
\begin{aligned}
C_{t+1} &= F_t * C_t + D_t \\
F_t &= \sigma(W_F[X_t, H_t]) \\
D_t &= \sigma(W_I[X_t, H_t]) * \tanh(W_D[X_t, H_t]) \\
H_{t+1} &= \sigma(W_O[X_t, H_t]) * \tanh(W_H C_{t+1})
\end{aligned}
$$

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



[Christopher Olah]

The highway path is H.

$$
\begin{aligned}
H_{t+1} &= (1 - I_t) * H_t + I_t * D_t \\
I_t &= \sigma(W_I[X_t, H_t]) \\
D_t &= \tanh(W_D[X_t, \sigma(W_r[x_t, H_t]) * H_t])
\end{aligned}
$$

GRUs vs. LSTMs

In TTIC31230 class projects GRUs consistently outperformed LSTMs.

They are also conceptually simpler.

GRU Language Models

GRU Character Language Models

h_0 and c_0 are parameters.

For $t > 0$, we have h_t as defined by the GRU equations.

$$P(x_{t+1} \mid x_0, x_1, \ldots, x_t) = \text{Softmax}(W_o\, h_t)$$

The parameter W_o has shape (V, H) where V is the vocabulary size (number of characters in a character model) and H is the dimension of the hidden state vectors.

$$\ell(x_0, ..., \ x_{T-1}) = \sum_{t=0}^{T-1} \ \ln \ 1/P(x_t \mid x_0, \ldots, x_{T-1})$$

Generating from a Character Language Model

Repeatedly select x_{t+1} from $P(x_{t+1} \mid x_0, x_1, \ldots, x_t)$

END