Residual Networks
And
Attention Models

cs273b
Recitation 11/11/2016

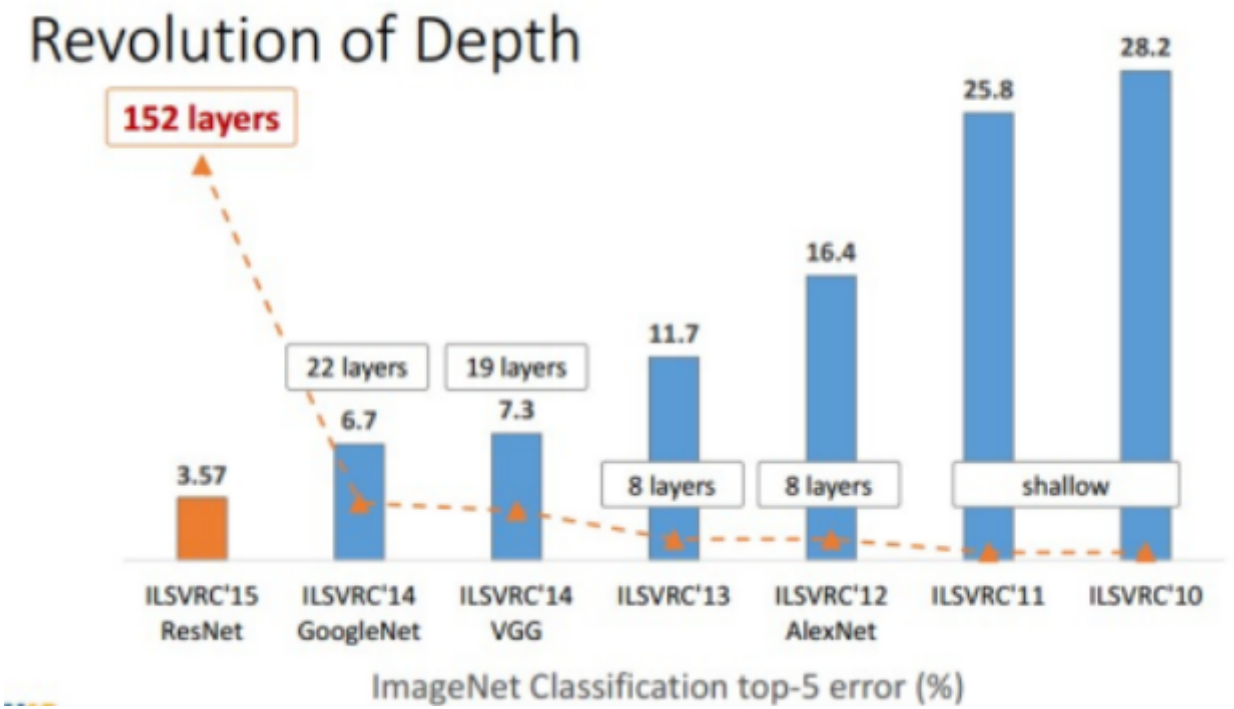
Anna Shcherbina

Introduction to ResNets

- Introduced in 2015 by Microsoft Research
  - Deep Residual Learning for Image Recognition (He, Zhang, Ren, Sun) https://arxiv.org/pdf/1512.03385v1.pdf
  - Won first place on 2015 ILSVRC classification task with an error of 3.57%
- Ease the training of networks that are substantially deeper than those used previously

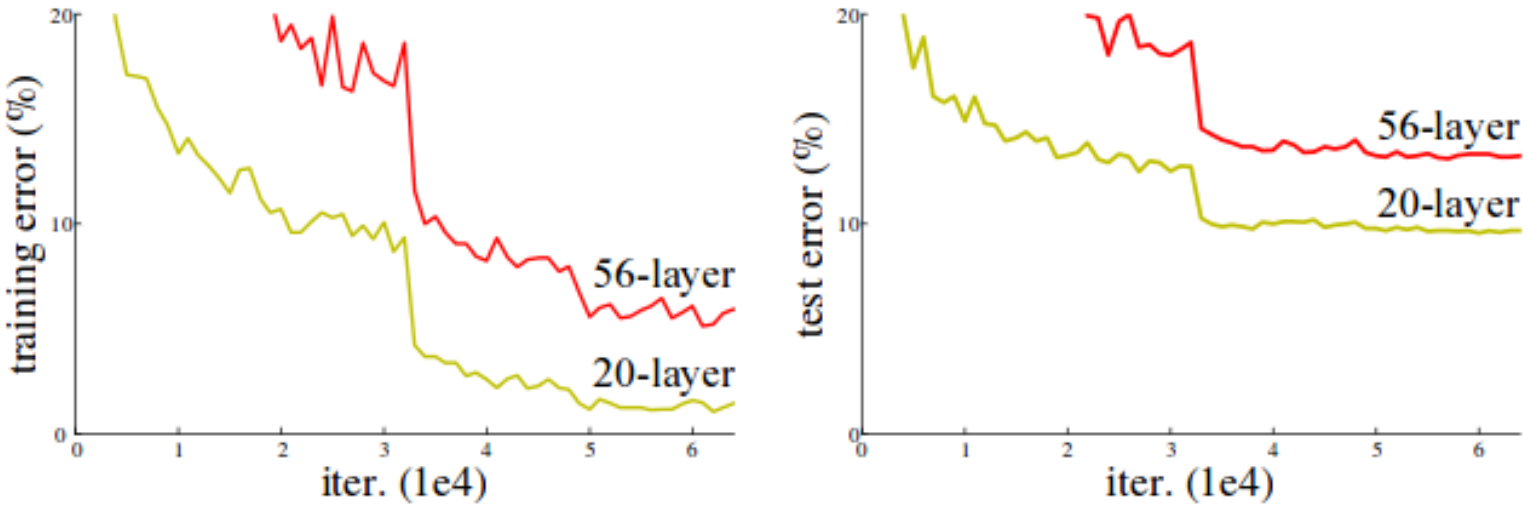
Deep Convolutional Neural Networks: Advantages

- Levels of features (low/mid/high-level features) can be enriched by increasing the number of stacked layers in a neural network
- So… is learning better networks as easy as stacking more layers?
  - Depth of ImageNet winning algorithms over the past 5 years suggests that this may indeed be the case



Deep Convolutional Neural Networks: Limitations

- Stacking network layers deeper and deeper exposes the problem of vanishing and exploding gradients:
  - This has largely been solved for networks to ~30 layers of depth
    - Normalized initialization (i.e. Xavier initialization)
    - Intermediate batch normalization layers between subsequent convolution layers
  - **Degradation problem persists**: As network depth increases, accuracy gets saturated, and then degrades rapidly.



More on the degradation problem

- Degradation problem is NOT caused by over-fitting to the training data.
  - The training error increases as well as the test error!

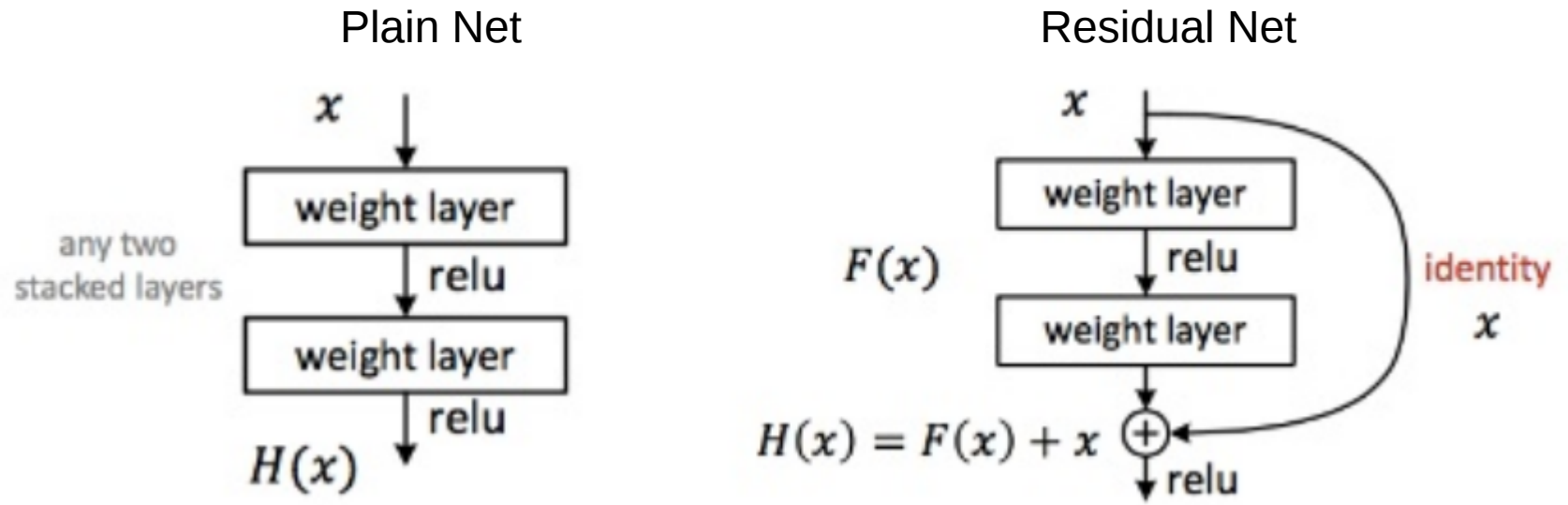
- The real cause is that the back propagation algorithm is inherently flawed

  - As we proceed from the output layer to the input layer via chain rule differentiation, the errors become smaller
  - After ~20 layers, underflow errors cause loss of information

Solution by Construction

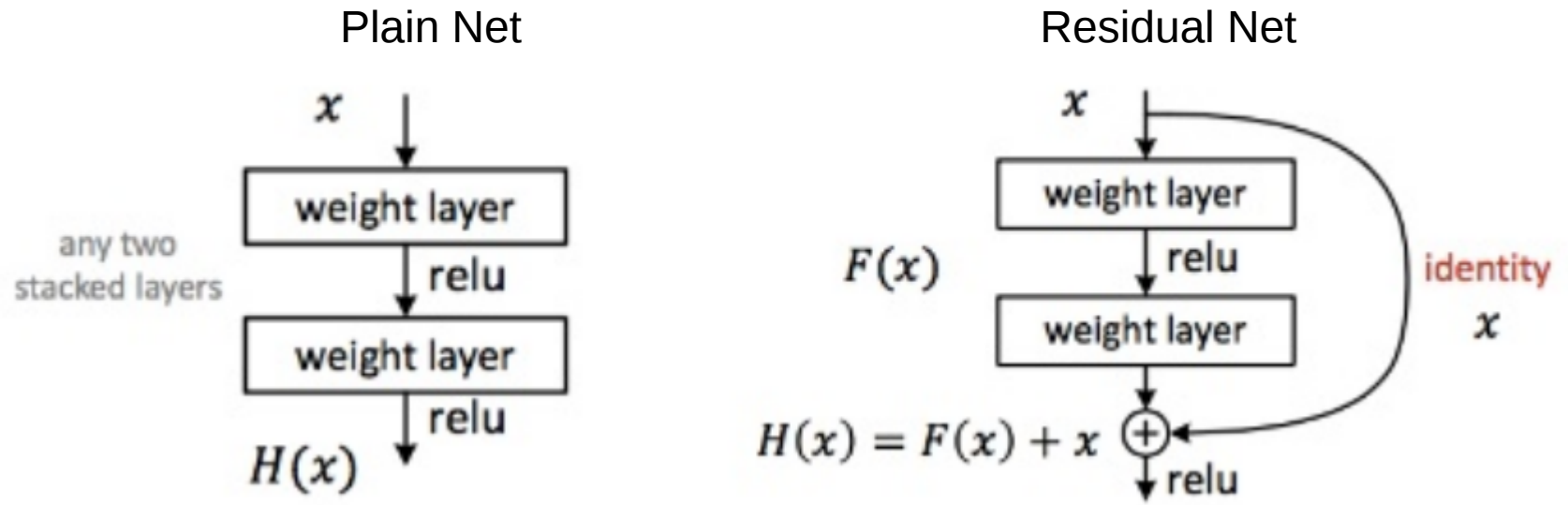
- We build a deep neural network as follows:
  - Copy layers from a learned shallower model.
  - Add layers that are *identity mapping* from the layers in the shallower model.
  - True, this model produces training error that is no higher than the shallow model counterpart
  - *But:* empirically, this constructed model serves as an upper bound for performance; current solvers are not able to find models where adding layers would actually improve performance compared to this constructed model.

The Deep Residual Learning Framework: Theory



- A small number of stacked layers are made to fit a residual mapping
- Formally:
  - The desired underlying mapping is denoted as H(x)
  - We let the stacked nonlinear layers fit another rmapping: F(x)=H(x)-x
  - Original mapping is recast to F(x)+x
- It is easier to optimize the residual mapping than to optimize the original unreferenced mapping
  - i.e. if an identify mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers

The Deep Residual Learning Framework: Implementation with Shortcut Connections

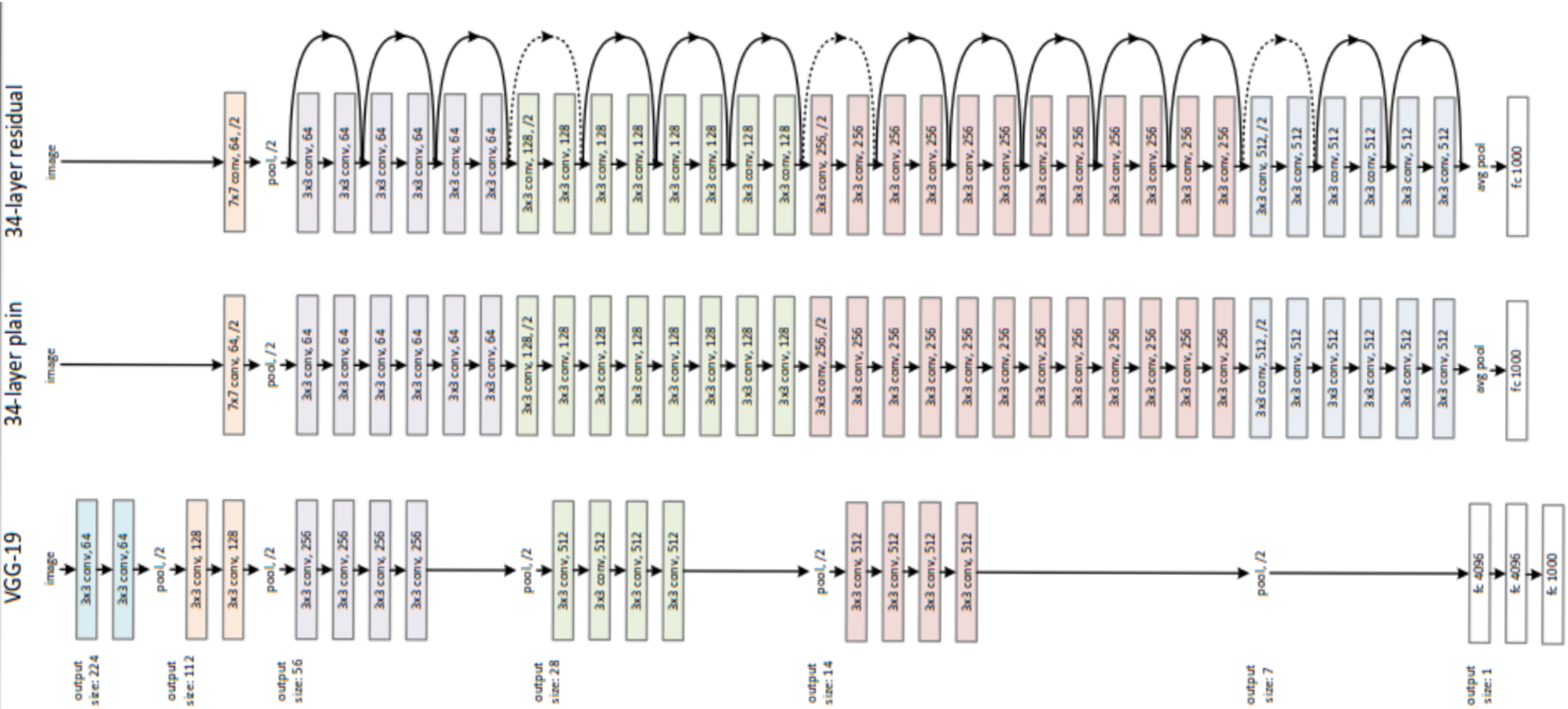


- Add shortcut connections for feed-forward neural nets to implement F(x)+x
  - Shortcut connections skip one or more layers
  - Shortcut connections perform identity mapping x→x
  - Outputs of shortcut connections are added to the outputs of the stacked layers

  - The dimensions of x and F(x) may not be equal!
    - If input/output channels are changed
    - Linearly project x to match F(x) by performing a 1x1 convolution

- Shortcut connections add no extra parameters or computational complexity!

Creating a 34-layer ResNet



- Solid lines – inputs and outputs have same dimensions
- Dotted shortcuts increase dimensions, two options for dealing with this
  - Shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions (no extra params)
  - Match dimensions by performing 1x1 convolutions

Evaluating a 34-layer ResNet on the ImageNet dataset

- 1000 classes
- 1.28 million training images
- 50k validation images

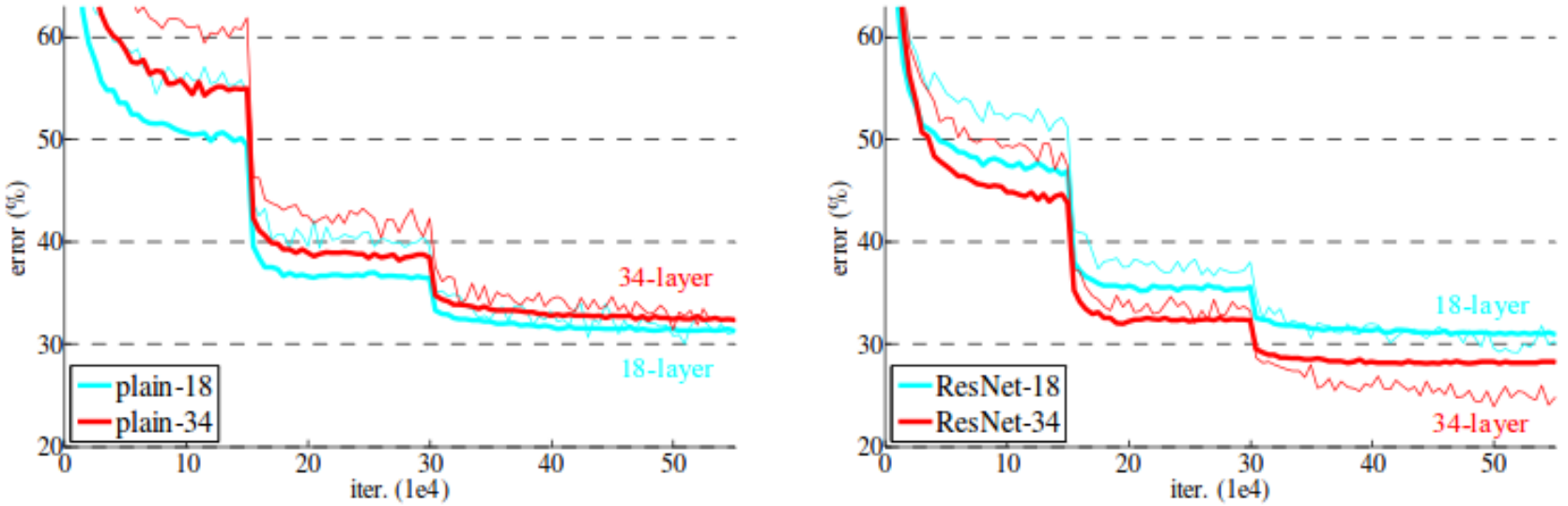


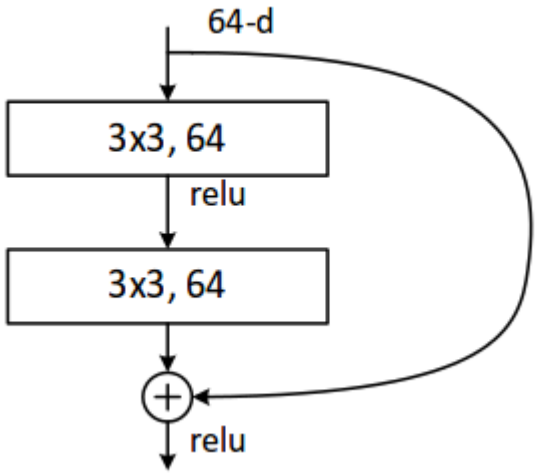
Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

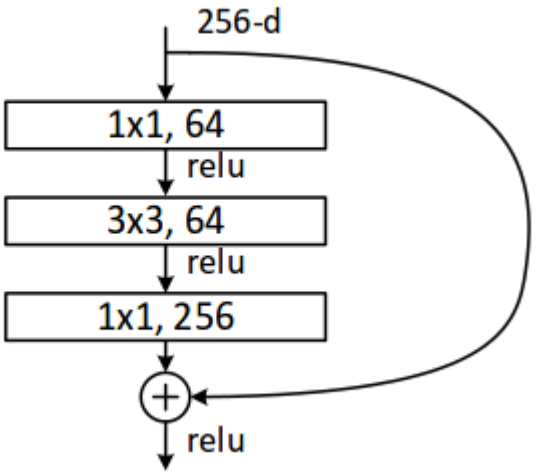
Table 2. Top-1 error (%, 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.

Bottleneck design

All 3x3



Bottleneck
(for ResNet-50/101/152)



- Bottleneck design reduces training time
- For each residual function, use a stack of 3 layers, instead of 2.
  - 1x1 conv layers are responsible for reducing and then restoring dimensions
  - 3x3 layer is a bottle neck with smaller input/output dimensions
- Similar numbers of parameters
  - (3x3x3)x64x64 + (3x3x3)x64x64 = 221184 (left)
  - (1x1x3)x265x64 + (3x3x3)x64x64 + (1x1x3)x64x256= 208896 (right)
- More width in bottleneck design

Exploring over 1000 layers

- “Deep Residutal Learning for Image Recognition” paper explored a ResNet model with 1202 layers

- No optimization difficulty

- Training error < 0.1%
- Test error 7.93%

- Test results is worse than for 110-layer ResNet → overfitting is a problem

But that's not all for ResNets...

- He et al's original technique only approximated identity connections between layers.
- Layers still used ReLu's (nonlinear activations) after the identity connections.
- Such nonlinearities impede gradient propagation.
- “Identity Mappings in Deep Residual Networks” by He et al. proposes the identity connection concept
  - https://arxiv.org/pdf/1603.05027v3.pdf

Residual Unit Formulation

$$\mathbf{y}_l = h(\mathbf{x}_l) + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l),$$
$$\mathbf{x}_{l+1} = f(\mathbf{y}_l),$$

- X_l is the input to the l-th unit
- X_l+1 is the output from the l-th unit
- F is a residual function

- Original formulation:
  - h(x_l) is an identity mapping
  - f is a ReLU

- Updated formulation:
  - Both h(x_l) and f are identity mappings
- We can re-write the residual unit formulation as follows:

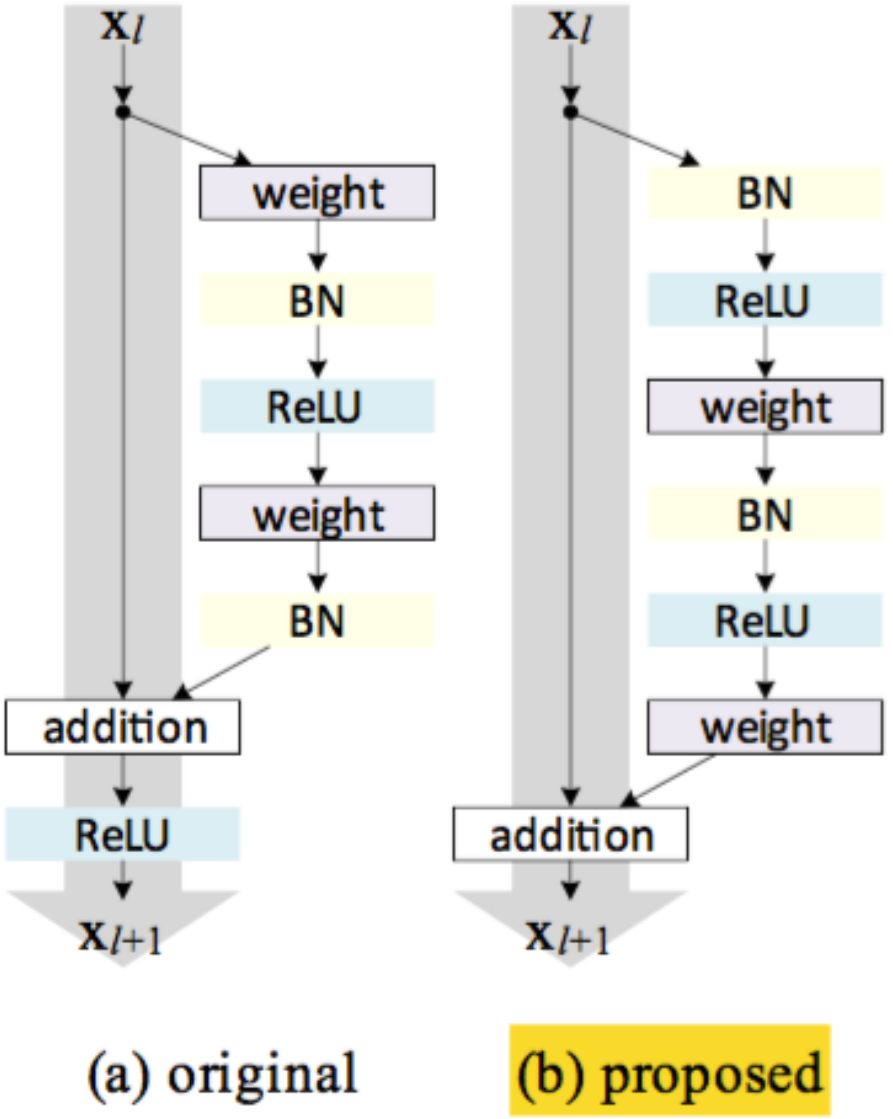
$$\mathbf{x}_{l+1} = \mathbf{x}_l + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l).$$

Recursively $(\mathbf{x}_{l+2} = \mathbf{x}_{l+1} + \mathcal{F}(\mathbf{x}_{l+1}, \mathcal{W}_{l+1}) = \mathbf{x}_l + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l) + \mathcal{F}(\mathbf{x}_{l+1}, \mathcal{W}_{l+1}),$ etc.) will have:

$$\mathbf{x}_L = \mathbf{x}_l + \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i),$$

(for any deeper unit L and any shallower unit l)

Residual Unit Formulation



- BN (bath norm) and ReLU are viewed as 'pre-activation' functions of the weight layers rather than 'post-activation' functions, as was the original design

Updated Residual Unit Formulation has Nice Mathematical Properties

$$\mathbf{x}_L = \mathbf{x}_l + \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)$$

- The feature X_L of any deeper unit L can be represented as the features x_l of any shallower unit l plus a residual function of the form $\sum_{i=l}^{L-1} \mathcal{F}$
- The feature $\mathbf{x}_L = \mathbf{x}_0 + \sum_{i=0}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)$ is the **summation** of the outputs of all preceding residual functions (plus x0). This is in contrast to a “plain network”, where a feature xL is a series of matrix-vector products $\prod_{i=0}^{L-1} W_i \mathbf{x}_0$ (ignoring BN and ReLU)

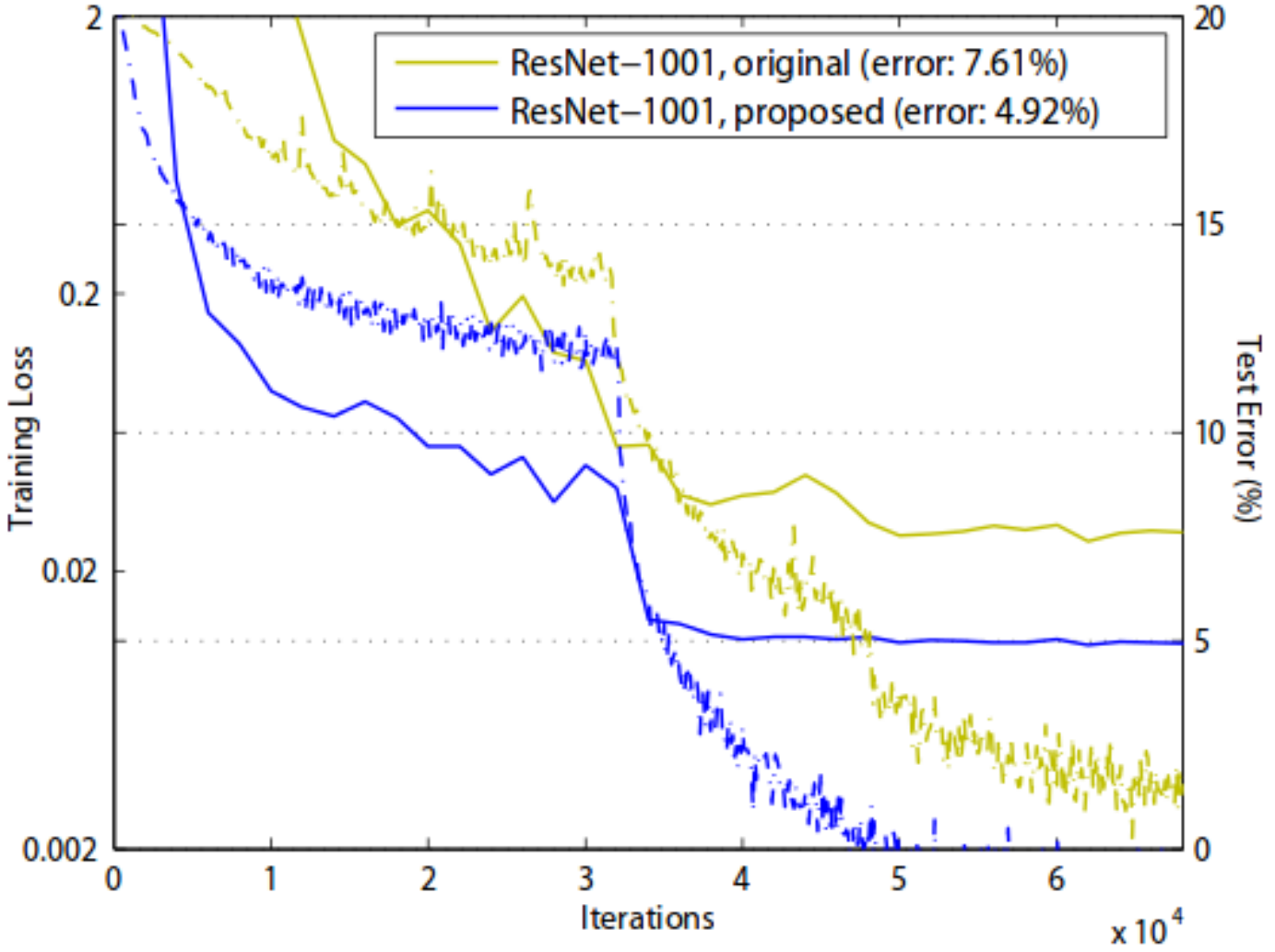
Updated Residual Unit Formulation
Backpropagation Properties

$$\frac{\partial \mathcal{E}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\left(1 + \frac{\partial}{\partial \mathbf{x}_l}\sum_{i=l}^{L-1}\mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)\right)$$

($\mathcal{E}$ s the loss function)

- The gradient can be decomposed into two additive terms
  - $\frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}$ Propagates information directly, without concerning any weight layers
    - ensures that information is directly propagated back to any shallower unit l
  - $\frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\left(\frac{\partial}{\partial \mathbf{x}_l}\sum_{i=l}^{L-1}\mathcal{F}\right)$ propagates through the weight layers
  - This formulation makes the gradient unlikely to be canceled out during a mini-batch → gradient of a layer does not vanish when the weights are small

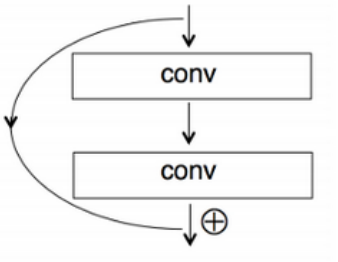
Updated Residual Unit Formulation Allows for Deeper Networks



- Less prone to overfitting on training data for ResNet-1001
- Evaluated on CIFAR-10 dataset

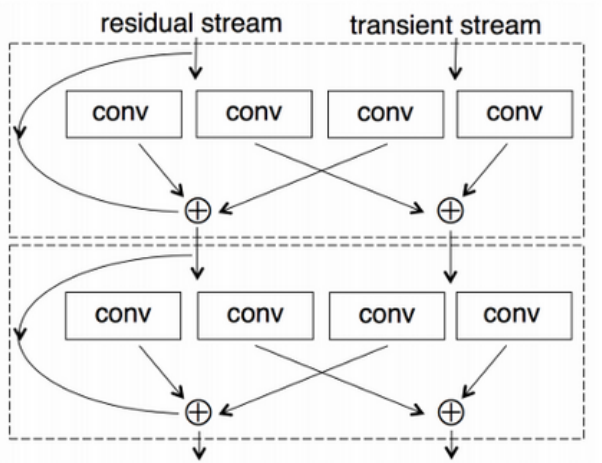
Expanding on the ResNet architecture: Resnet in Resnet

- “Resnet in Resnet: Generalizing Residual Architectures” by Targ et al
- For every ResNet layer, a traditional convolution layer is placed next to it
- Parallel layers are allowed to exchange information
- Fractal ResNet structure



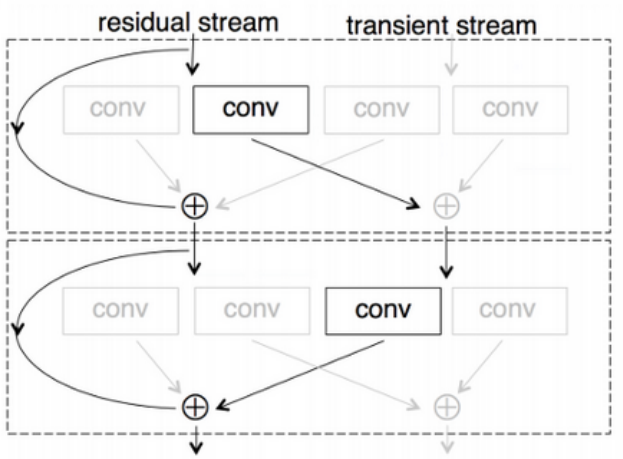
(a)

2-layer ResNet block



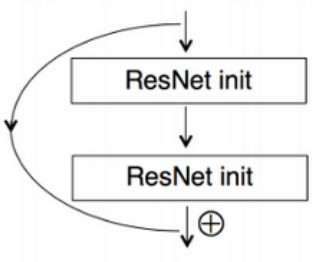
(b)

2 generalized residual blocks (ResNet Init)



(c)

2-layer ResNet block from 2 generalized residual blocks (grayed out connections are 0).

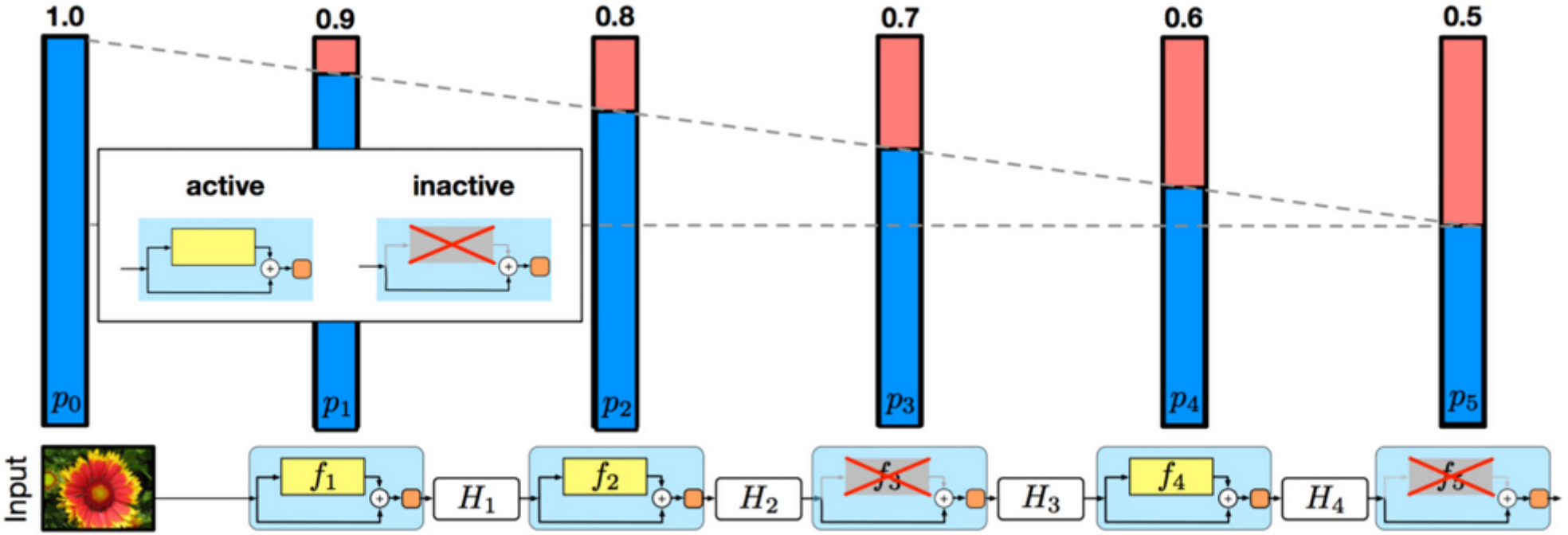


(d)

2-layer RiR block

Expanding on the ResNet architecture: Deep Networks with Stochastic Depth

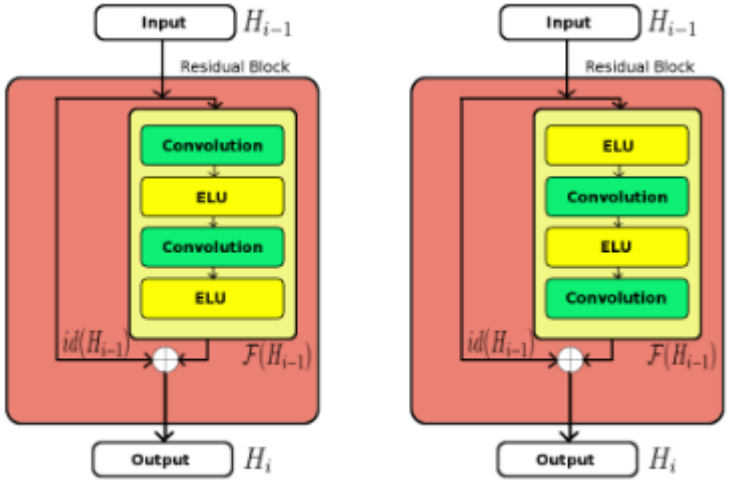
- “Deep Networks with Stochastic Depth” by Huang et al
- Modifies dropout to remove layers from the network during training time
- Entire layers get dropped at random
- Remaining layers compensate for missing inputs by relying on lower inputs
- At each training step, layers get reactivated and new ones get zero-ed
- Network depth shrinks during training, entire depth of model used at inference time



Linear decay of layer dropout probabilities in stochastic depth.

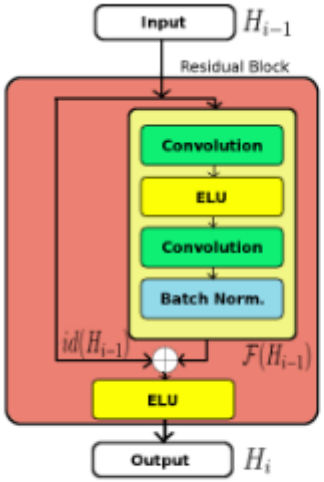
Expanding on the ResNet architecture: Deep Residual Networks with Exponential Linear Unit

- “Deep Residual Networks with Exponential Linear Unit” by Shah et al
- Exponential linear units are used as an alternative to ReLUs

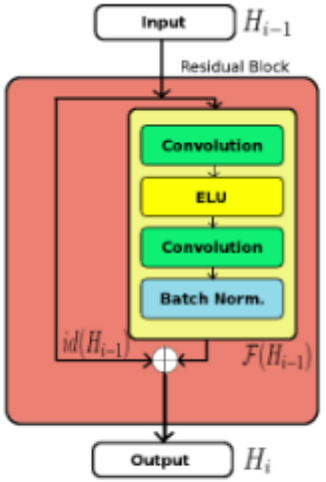


(a) Conv-ELU-Conv-ELU

(b) ELU-Conv-ELU-Conv



(c) Conv-ELU-Conv-BN and ELU after Addition



(d) Conv-ELU-Conv-BN and No ELU after Addition

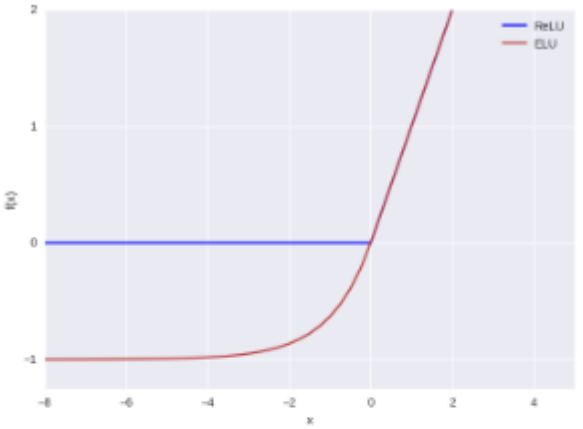
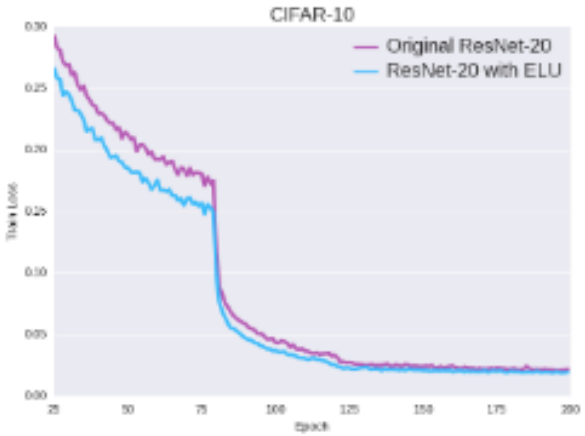


Figure 2: The rectified linear unit (ReLU) and Exponential Linear Unit (ELU, $\alpha = 1.0$)

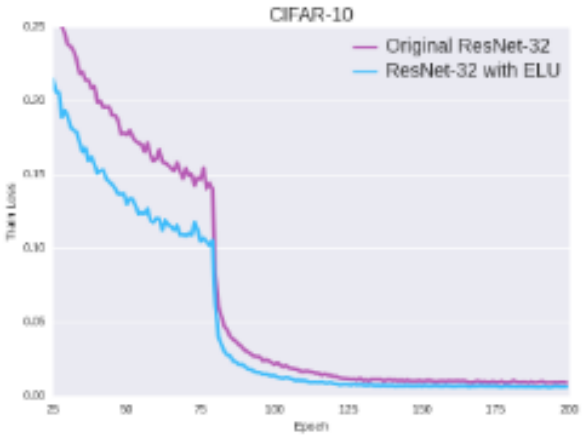
Exponential Linear Unit (ELU) [9] alleviates the vanishing gradient problem and also speeds up learning in deep neural networks which leads to higher classification accuracies. The *exponential linear unit* (ELU) is

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

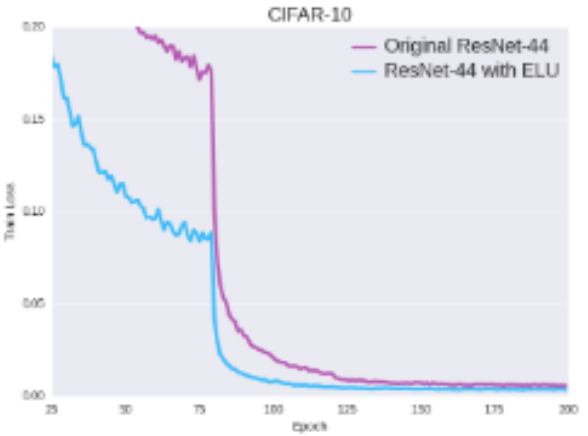
Expanding on the ResNet architecture: Deep Residual Networks with Exponential Linear Unit



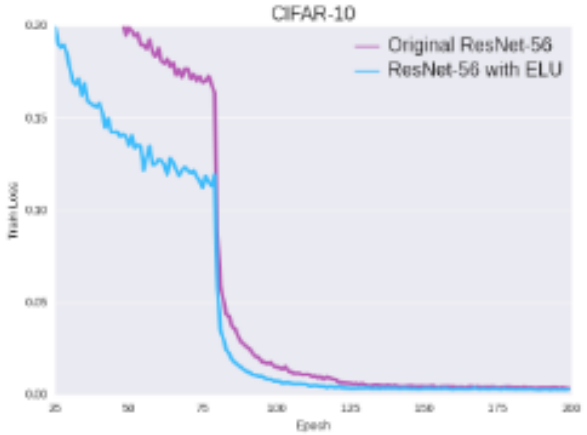
(a) 20-layers



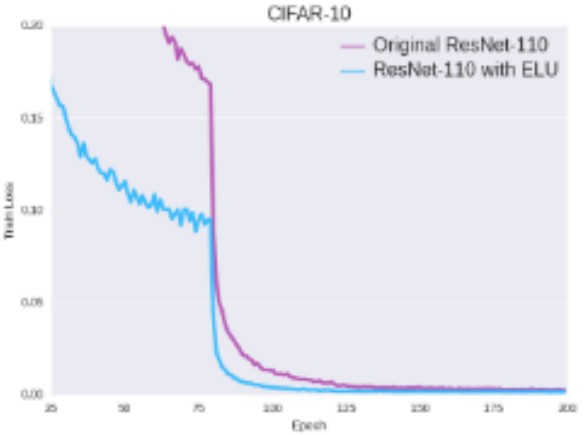
(b) 32-layers



(c) 44-layers



(d) 56-layers



(e) 110-layers

Figure 5: Comparison of the learning behavior of our model and the original ResNet model on CIFAR-10 dataset. We compare the results for 20, 32, 44, 56 and 110-layers and show that our model significantly outperforms the original ResNet model.

Attention Models
And their Applications

Attention Mechanisms are Rooted in the Study of Human Visual Attention

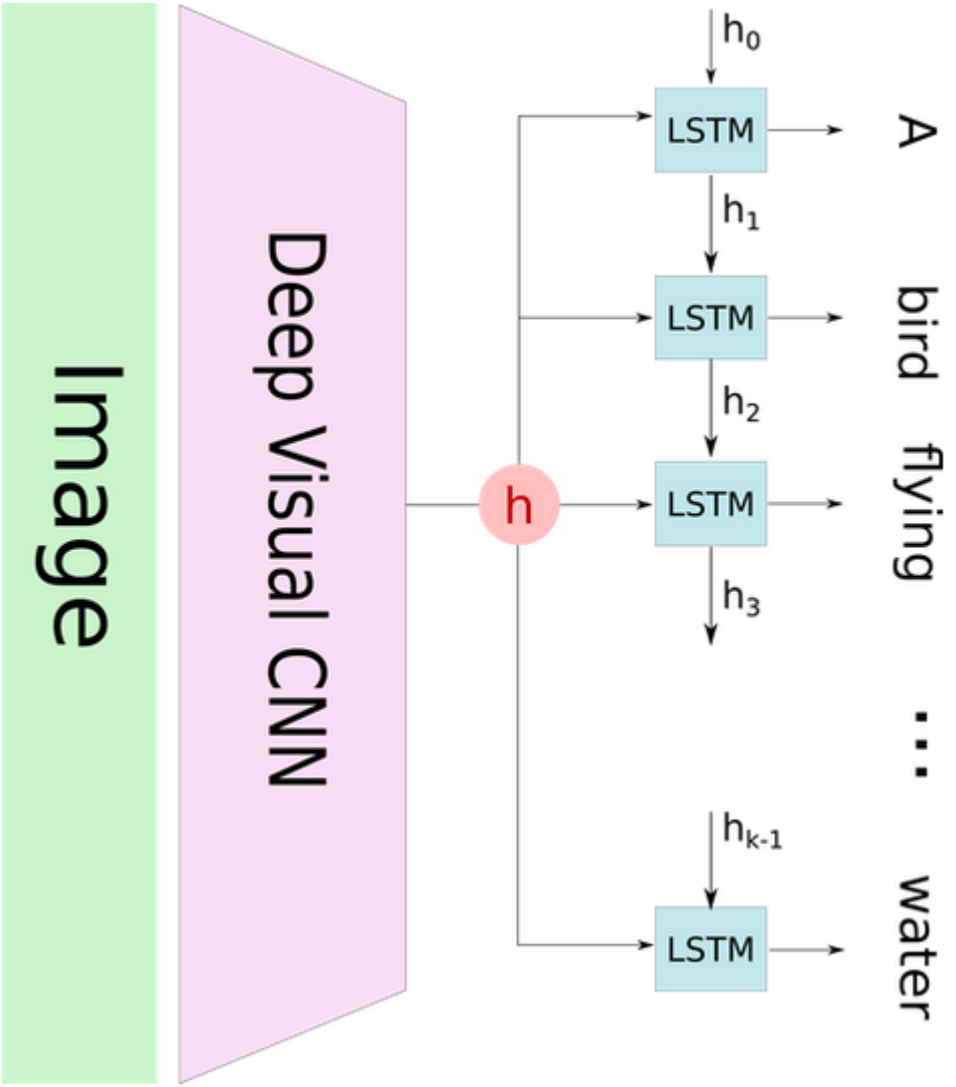
- Animals focus on specific parts of their visual inputs to compute the adequate response

- Select the most pertinent piece of information, rather than using all available information
  - Most info is actually irrelevant for computing a visual response!

Attention for Image Captioning: A Motivating Example

Classic approach:

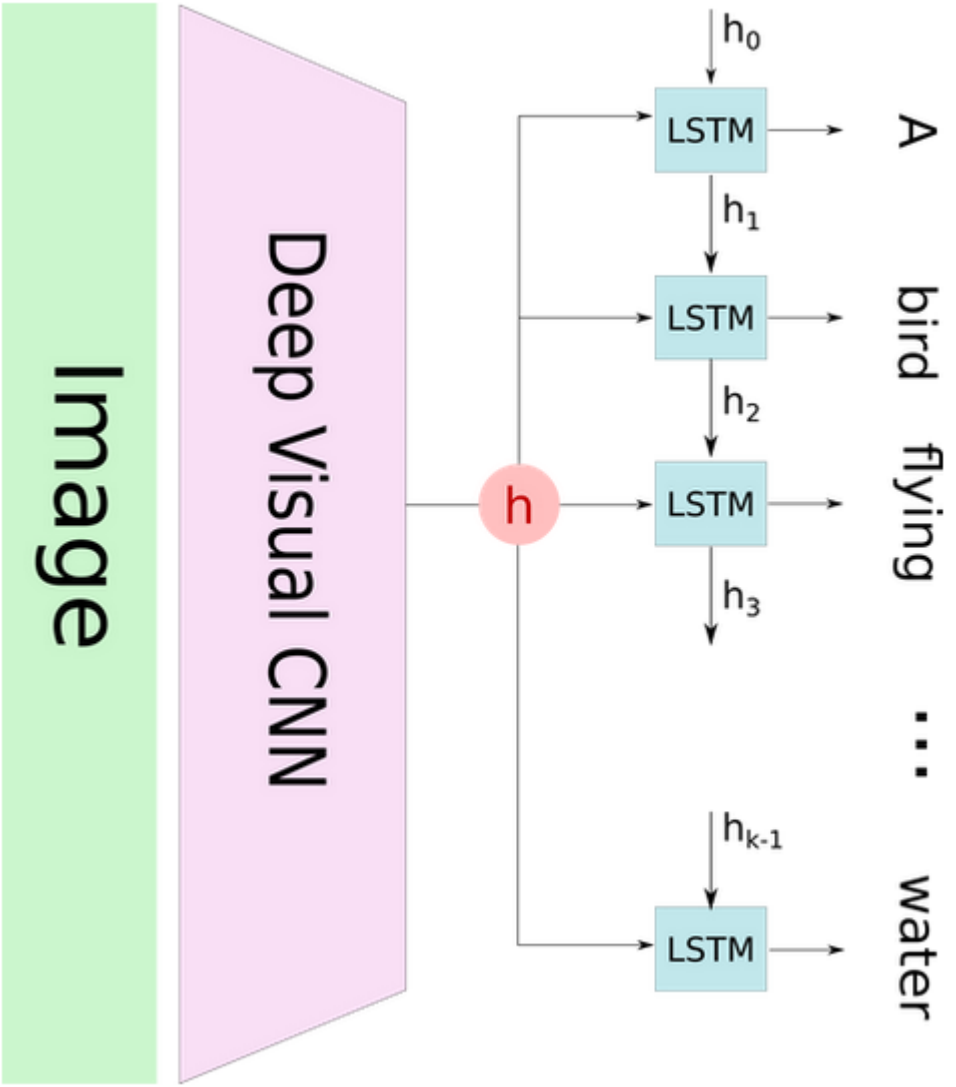
- Pre-trained CNN used to encode image and produce hidden state h
- RNN used to decode the hidden state and recursively generate each word in the image caption



Attention for Image Captioning: Limitations of the Classical Model

Problems with the classical approach

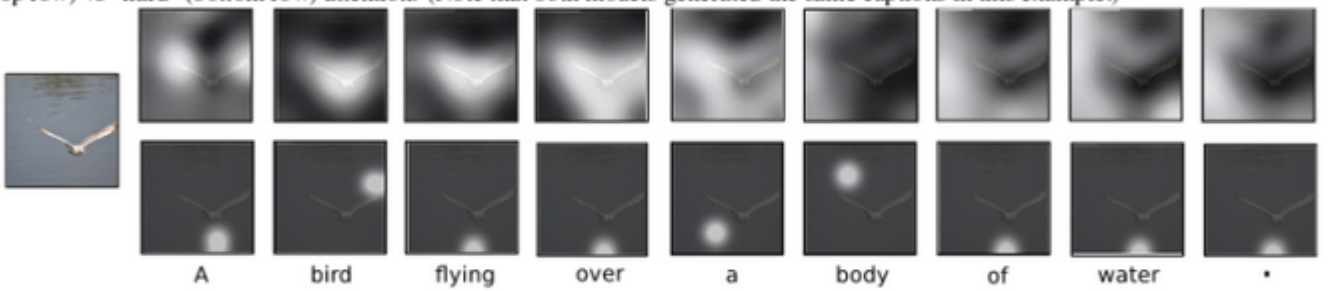
- When the model generates the next word in a caption, the word usually describes only part of the image
- Using the whole representation of the image h to condition the generation of each word cannot efficiently produce different words for different parts of the image



Attention for Image Captioning:
Attention mechanisms overcome limitations of the classical model

- Attention mechanisms remove this problem
- Image is first divided into n parts
- CNN used to compute representation of each part h_1… h_n
- When RNN generates new word, the attention mechanism focuses on the relevant part of the image
- Decoder uses only specific part of image

Figure 2. Attention over time. As the model generates each word, its attention changes to reflect the relevant parts of the image. "soft" (top row) vs "hard" (bottom row) attention. (Note that both models generated the same captions in this example.)

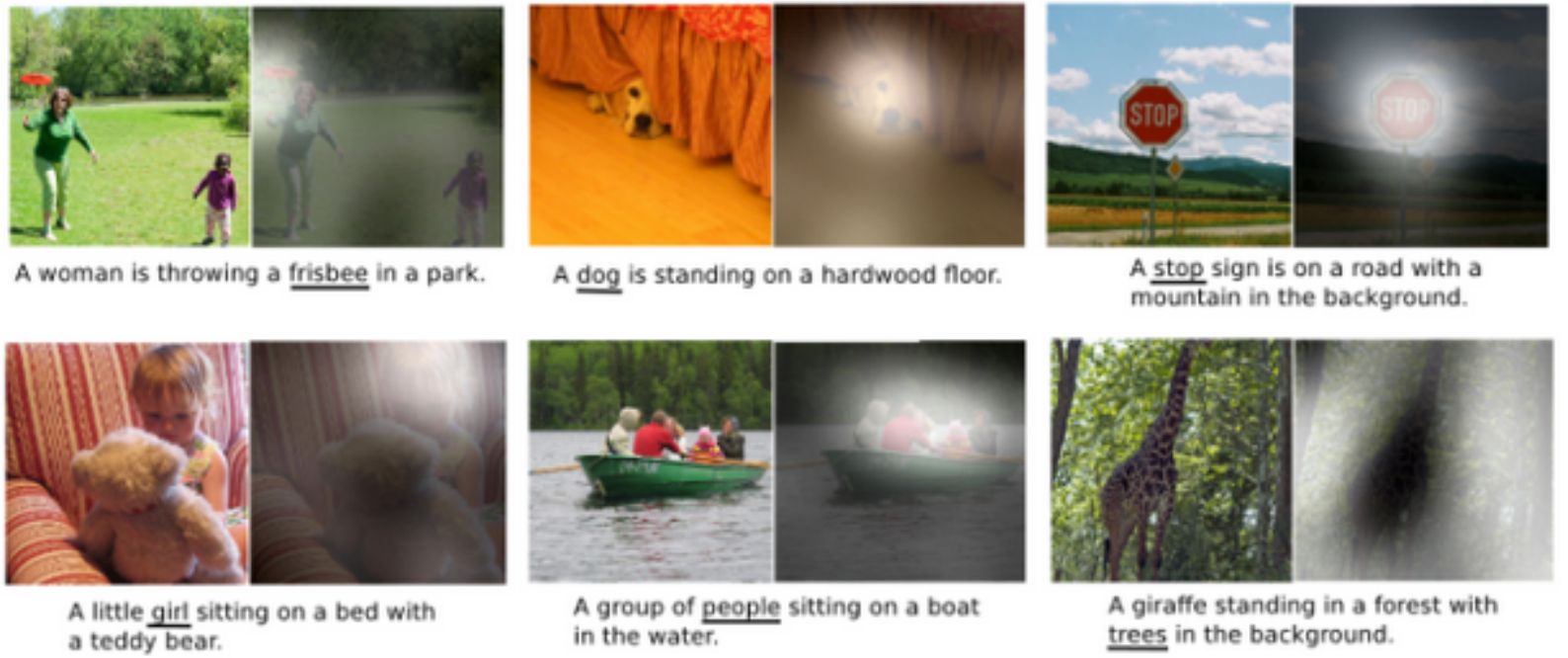


Vinyals, Oriol, et al. « Show and tell: A neural image caption generator. » arXiv preprint arXiv:1411.4555 (2014).

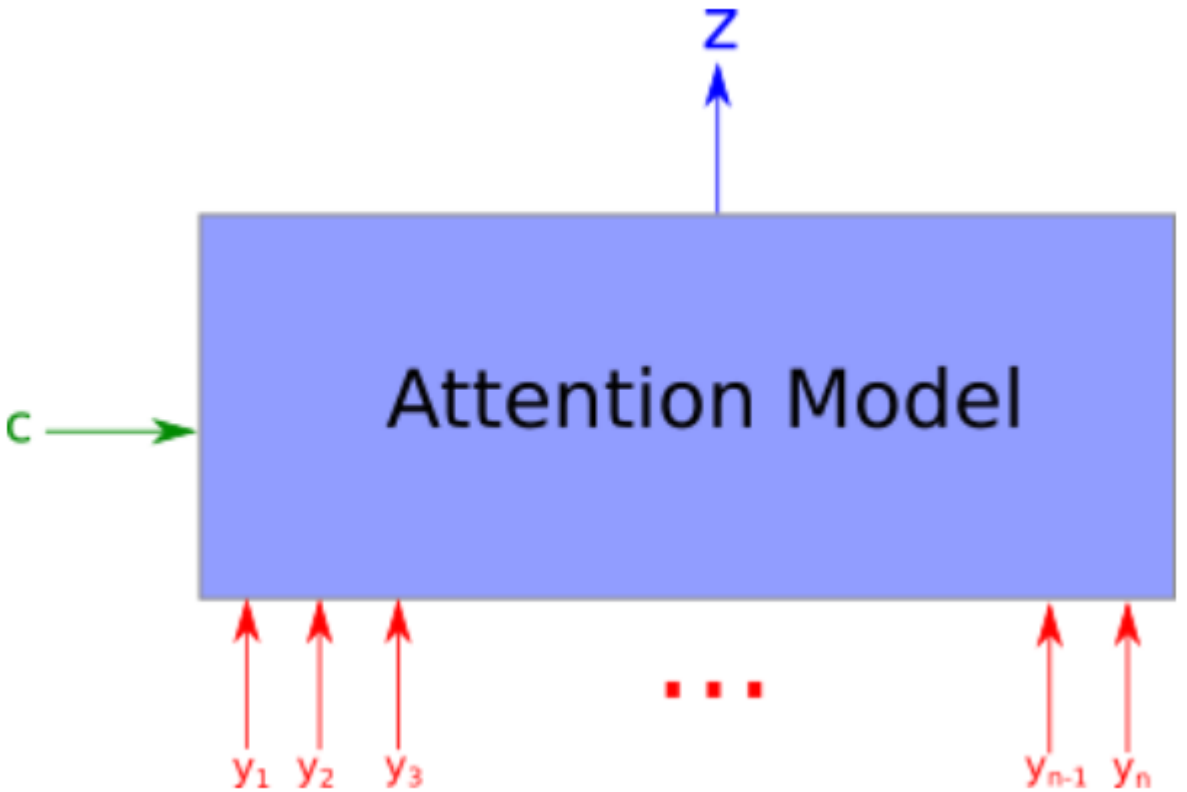
Attention for Image Captioning: A Motivating Example

- Attention lets us look at the “relevant” part of these images to generate the underlined words:

Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)



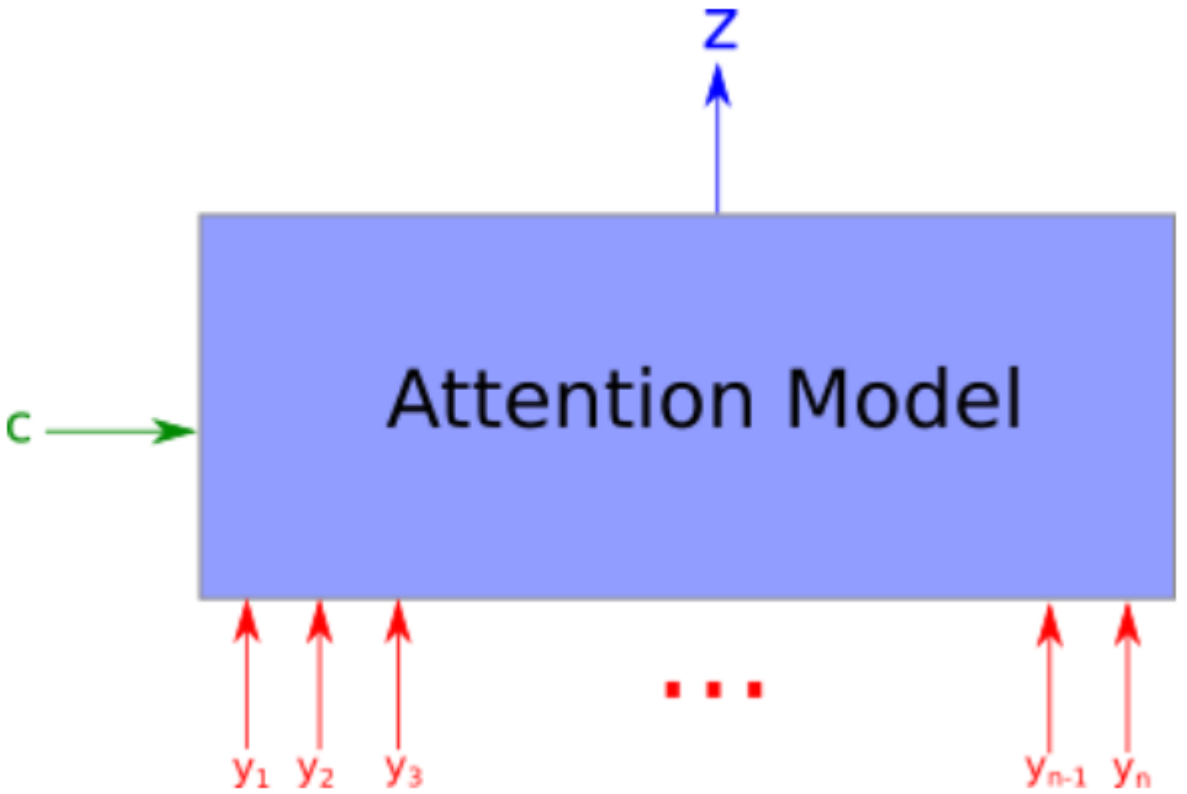
Attention Models:
General Formulation



- Inputs:
  - n arguments (y_1 … y_n)
    - In the image caption example, y_i = h_i (CNN representation for part of an image)
  - Context C

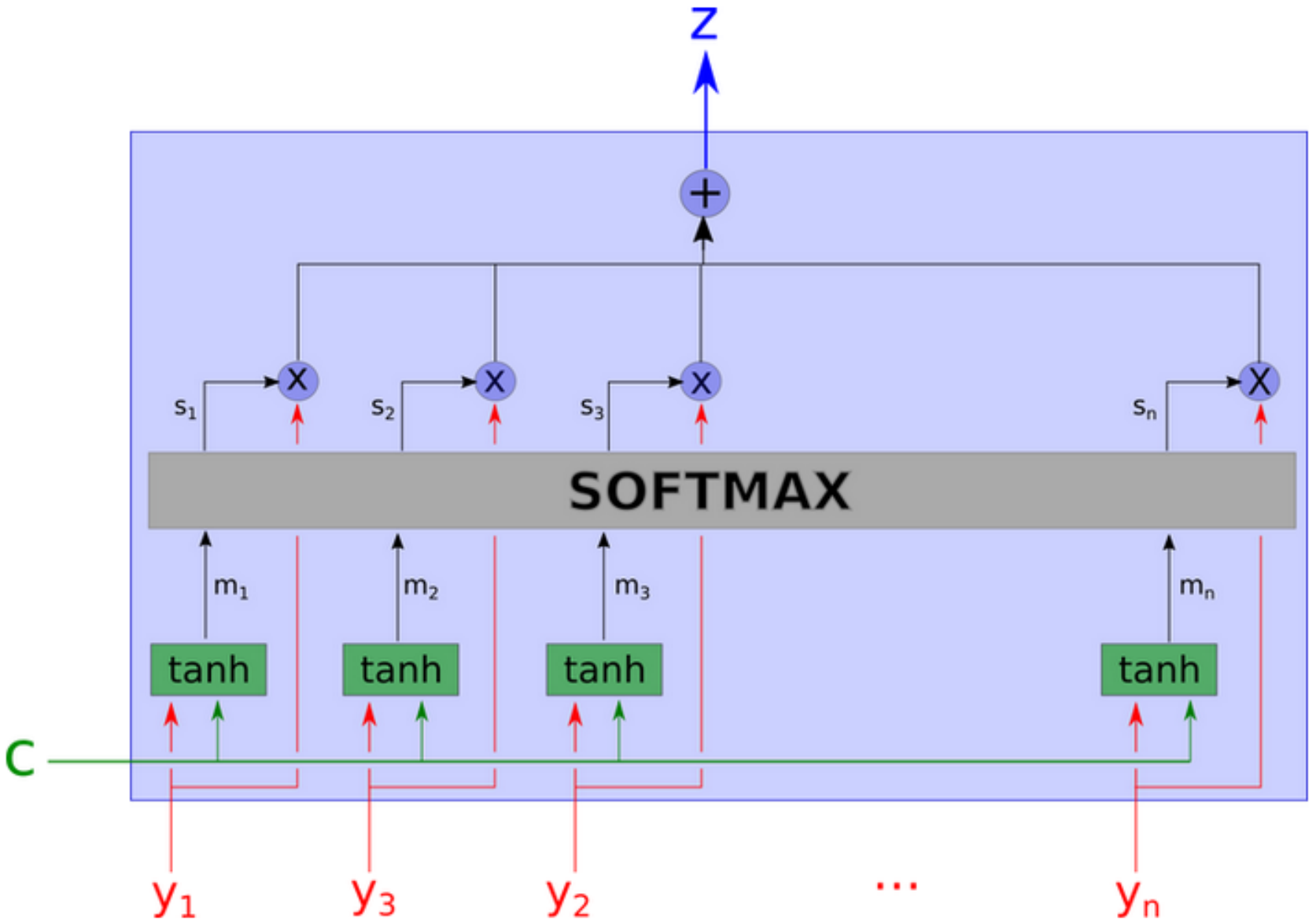
- Outputs:
  - Vector z, which is the summary of y_i, focusing on the information linked in context C
  - Specifically, a weighted arithmetic mean of y_i, with weights chosen by relevance of each y_i, given the context C

Attention Models:
General Formulation

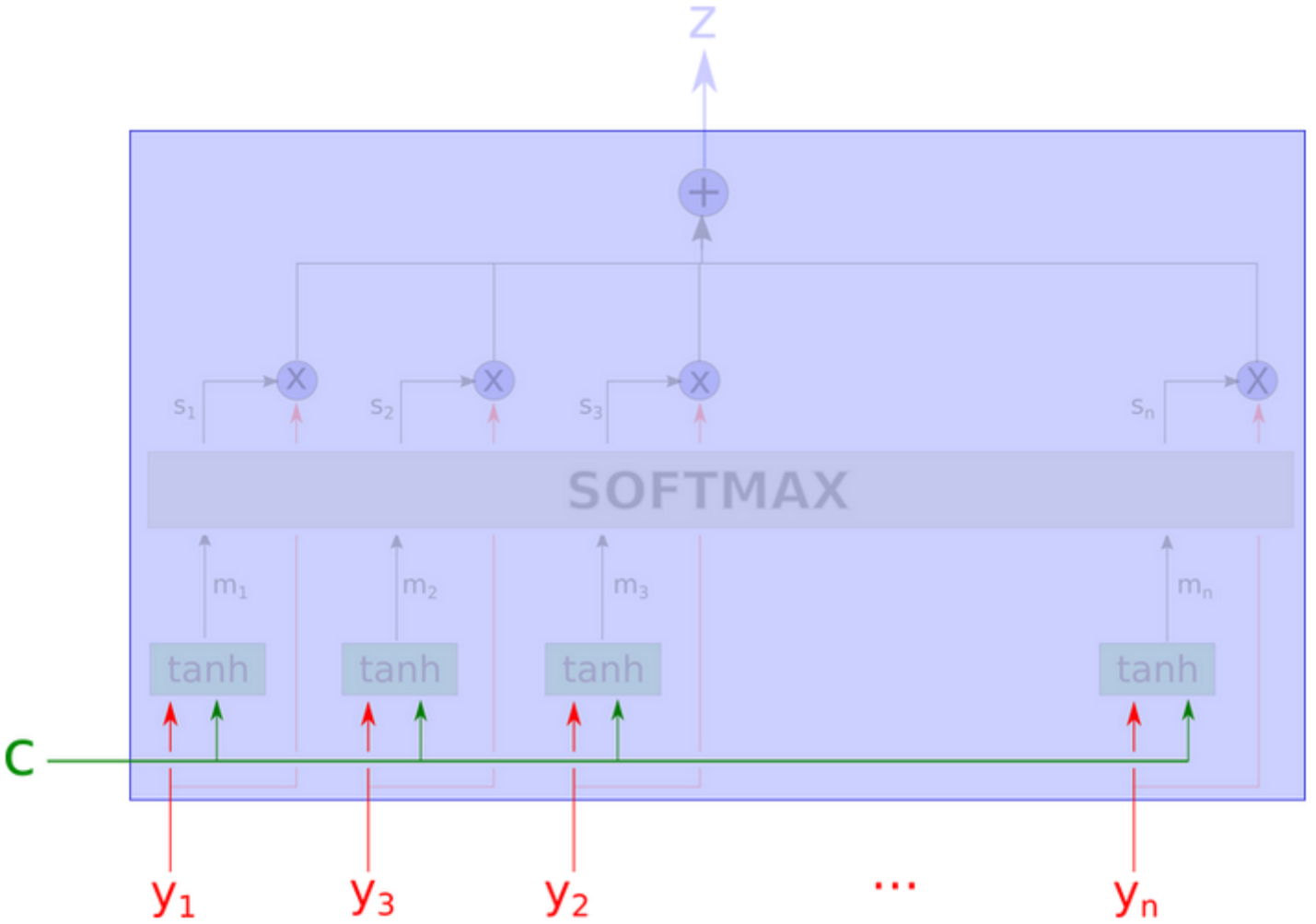


- For our motivating example:
  - Context (C) → beginning of the generated sentence
  - y_i → representation of part of an image (h_i)
  - Z → representation of the filtered image, with a filtering putting the focus on the interesting part of the image for the current word

Attention Models:
Looking Inside the Black Box

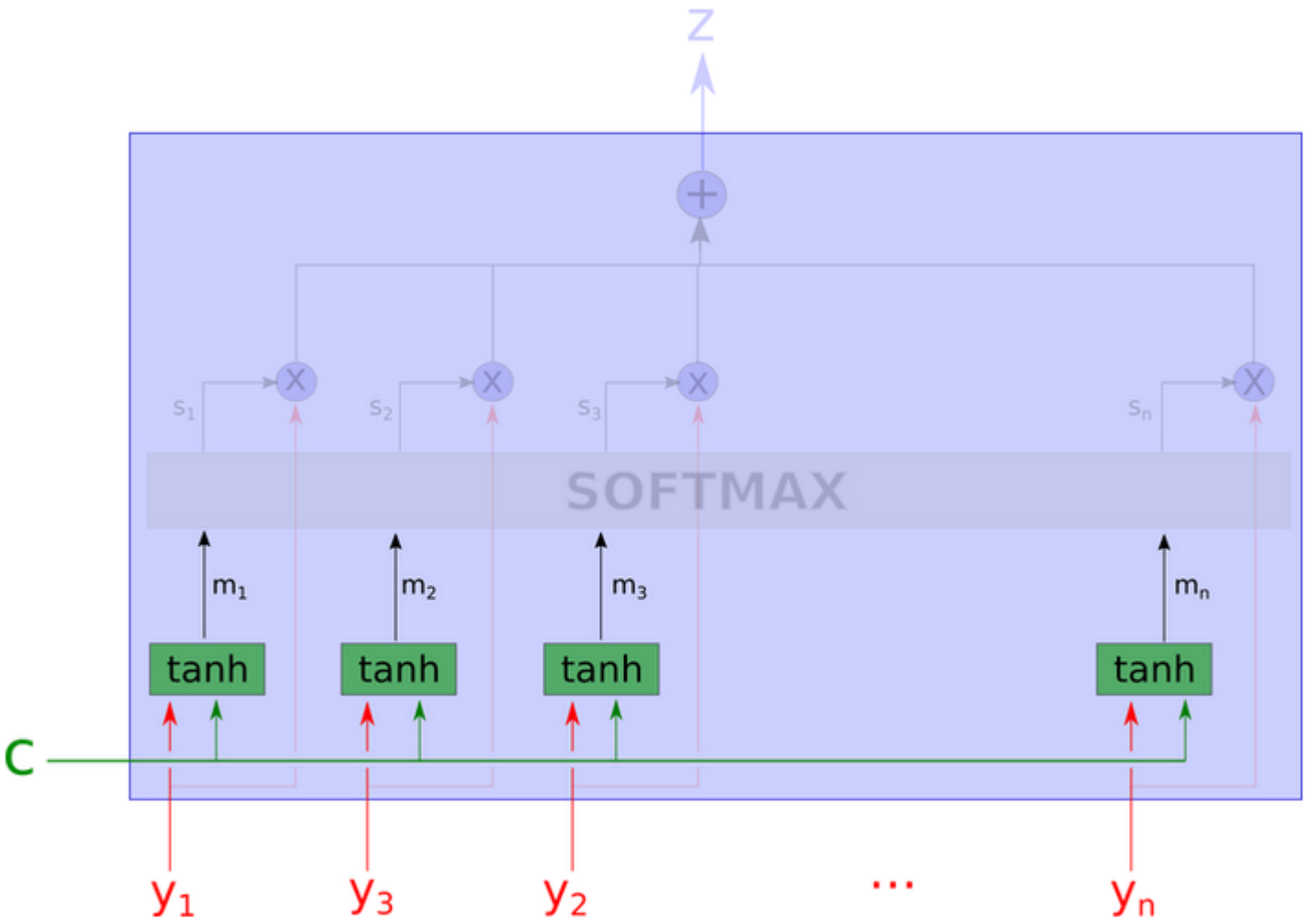


Looking Inside the Black Box: Inputs to the Attention Model



Looking Inside the Black Box: tanh layer

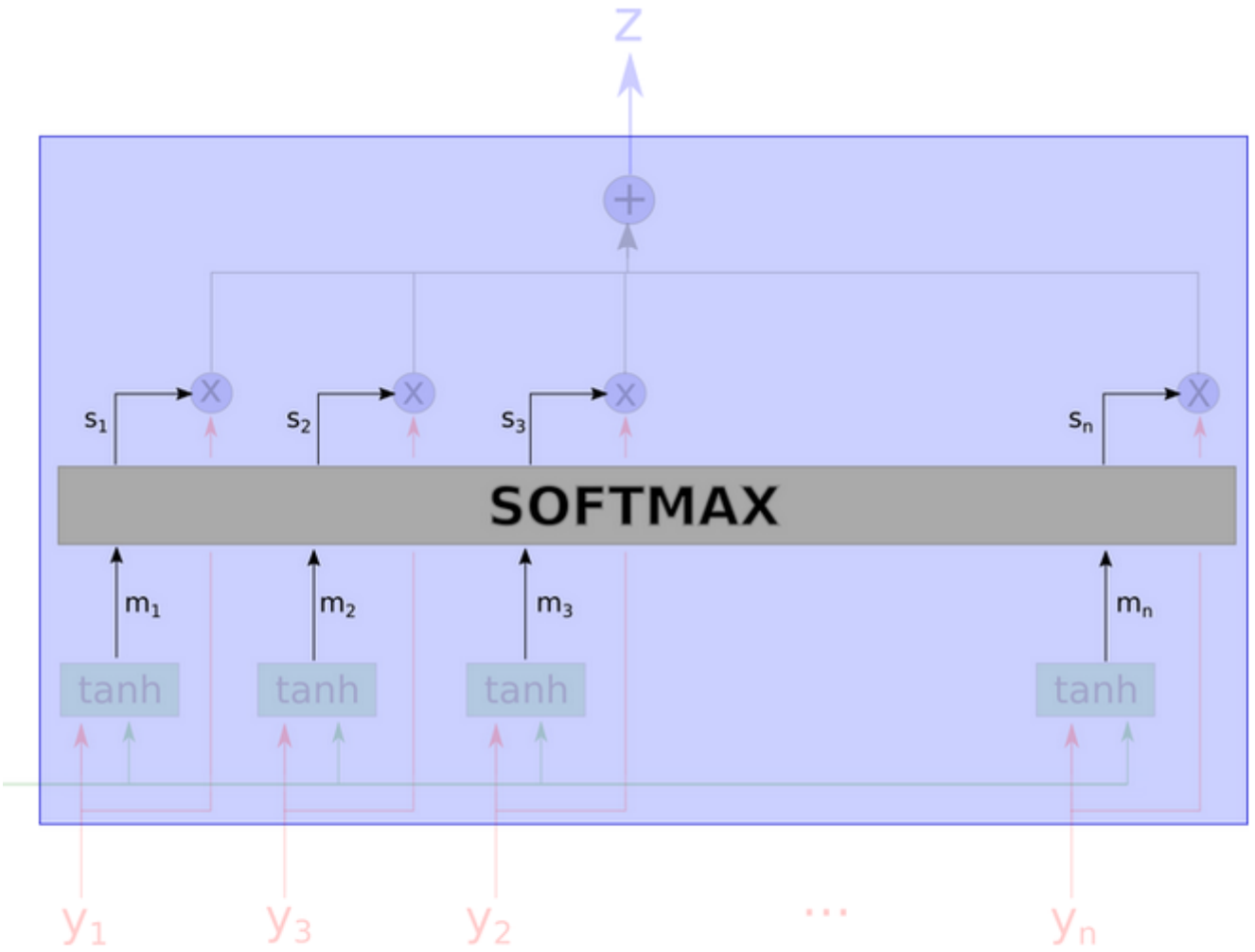
- m_1 … m_n are computed with a tanh layer
- This is effectively an aggregation of the values of y_i and c.
- Each m_i is computed independently without looking at the other y_j for j != i



$$m_i = \tanh\left(W_{cm}c + W_{ym}y_i\right)$$

Looking Inside the Black Box: softmax layer

- Compute each weight using a softmax
- s_i are the softmax of the m_i projected on a learned direction
- Softmax can be thought of as the max of the “relevance” of the variables, according to the context

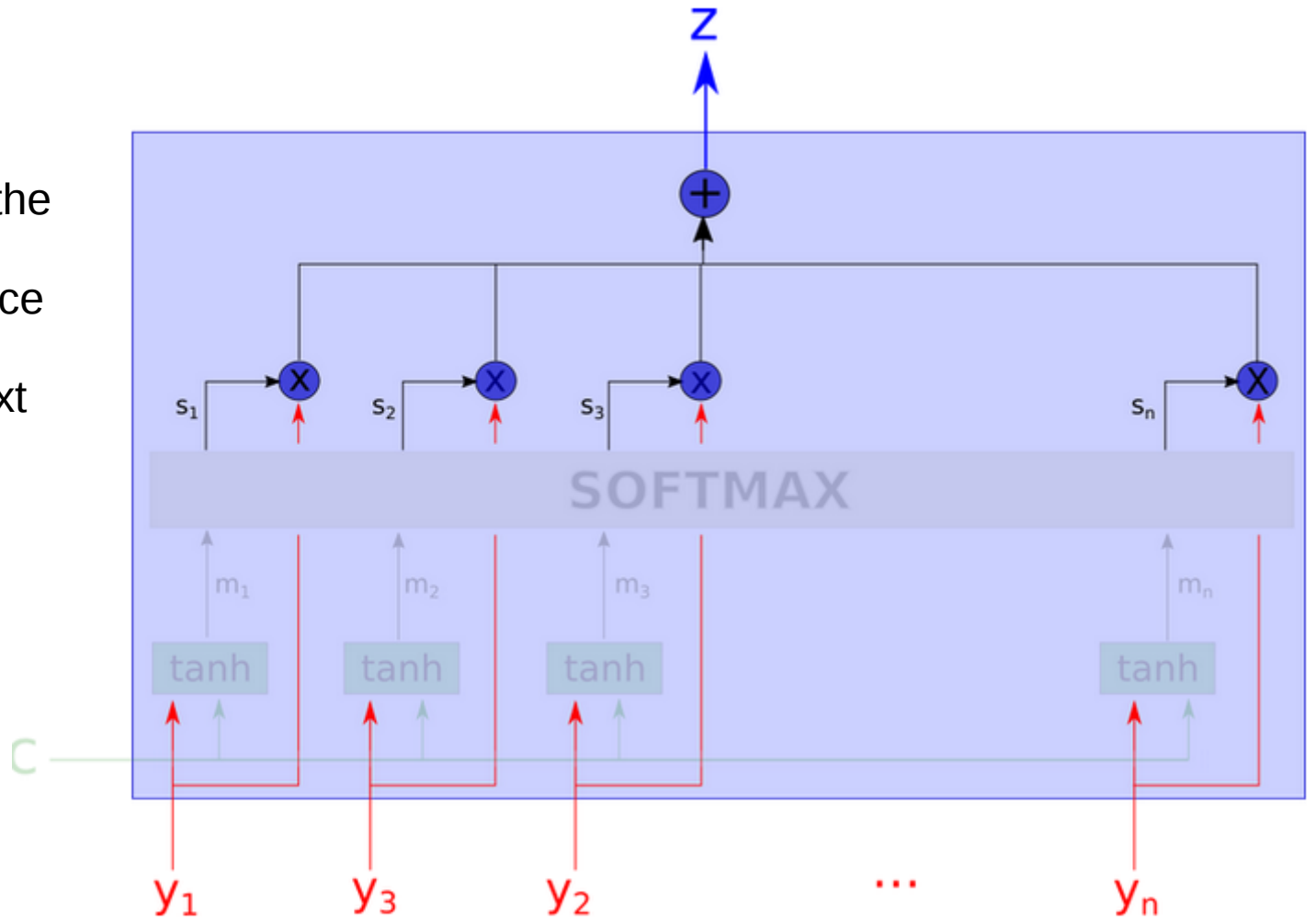


$$s_i \propto \exp\left(\langle w_m, m_i \rangle\right)$$

$$\sum_i s_i = 1$$

Looking Inside the Black Box: output Z

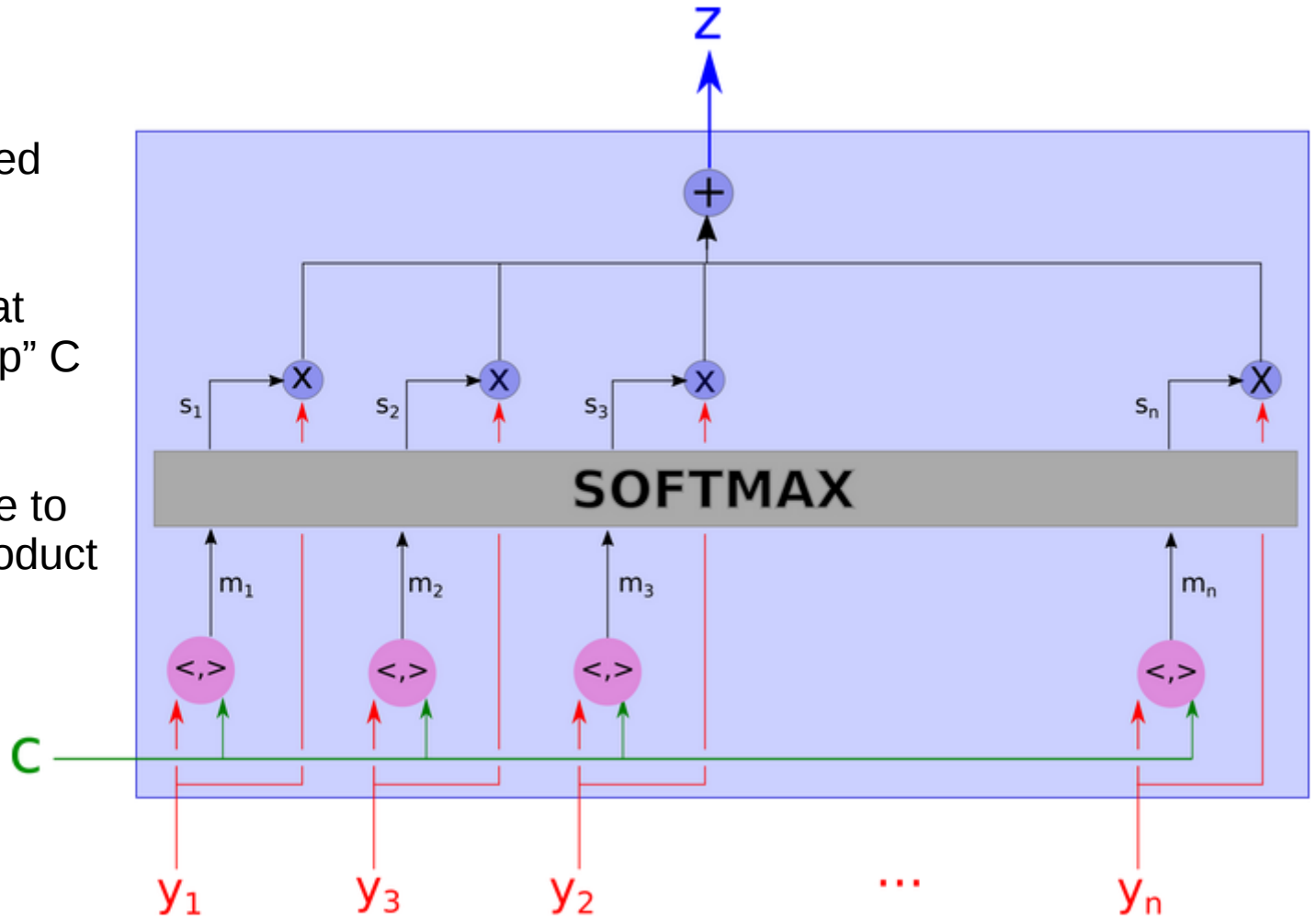
- Output Z is a weighted arithmetic mean of all the y_i, where the weight represents the relevance of each variable according to the context C.



$$z = \sum_i s_i y_i$$

Architecture Modifications:
You don't HAVE to use a tanh layer to compute relevance

- tanh layer can be replaced by any other network
- The only constraint is that this function must “mix up” C and y_i
- For example, it's possible to simply compute a dot product between C and y_i



Attention Model with an other computation method for relevance.

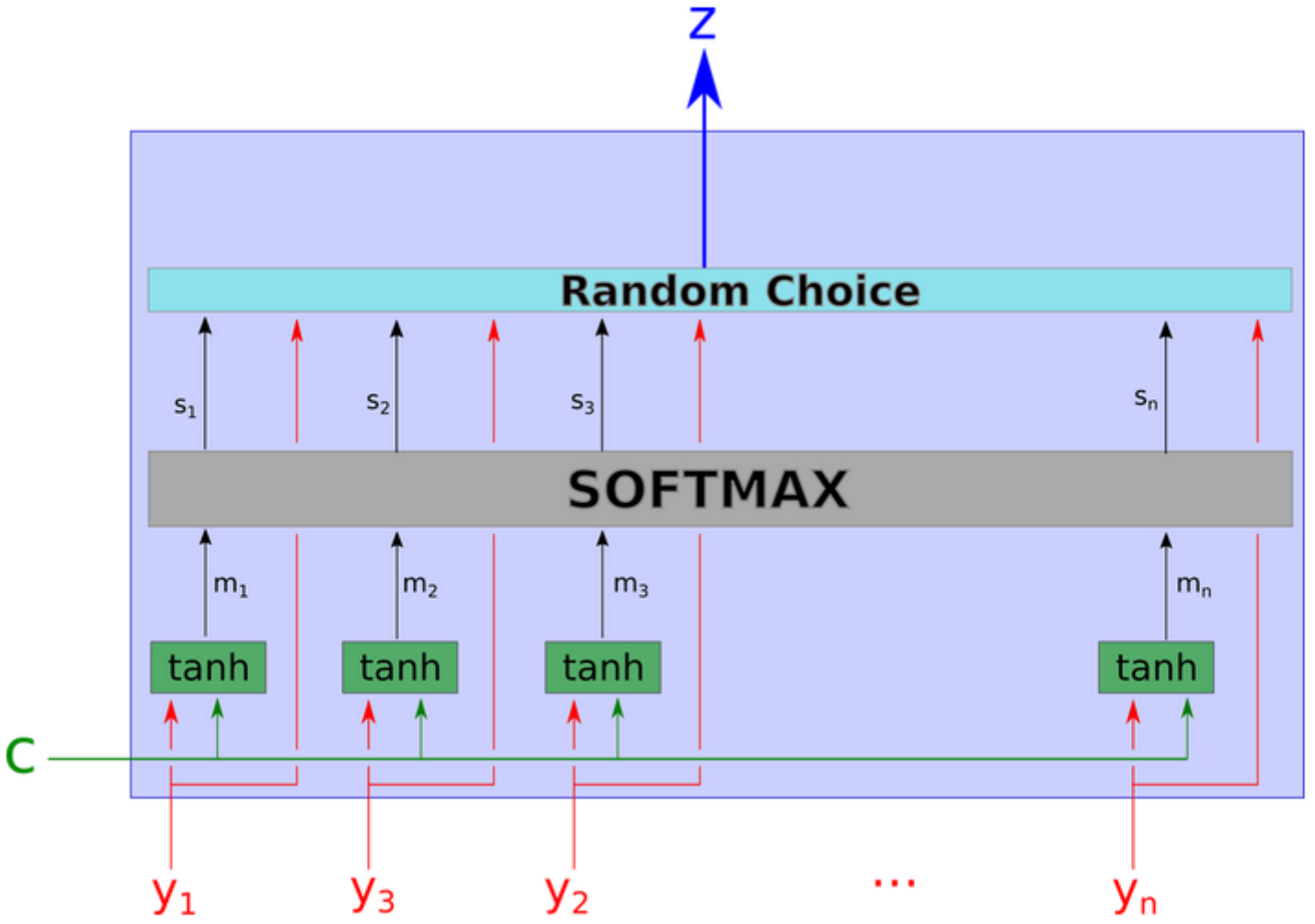
Soft Attention vs Hard Attention

- **Soft attention:**
  - Fully differentiable deterministic mechanism
  - Can be plugged into an existing system
  - Gradients are propagated through the attention mechanism at the same time as they are propagated through the rest of the network
  - This is the mechanism that we have been discussing so far

- **Hard attention:**
  - Stochastic process
  - System samples a hidden state y_i with probability s_i (not all hidden states are used as inputs for decoding)
  - Gradients are estimated by Monte Carlo sampling

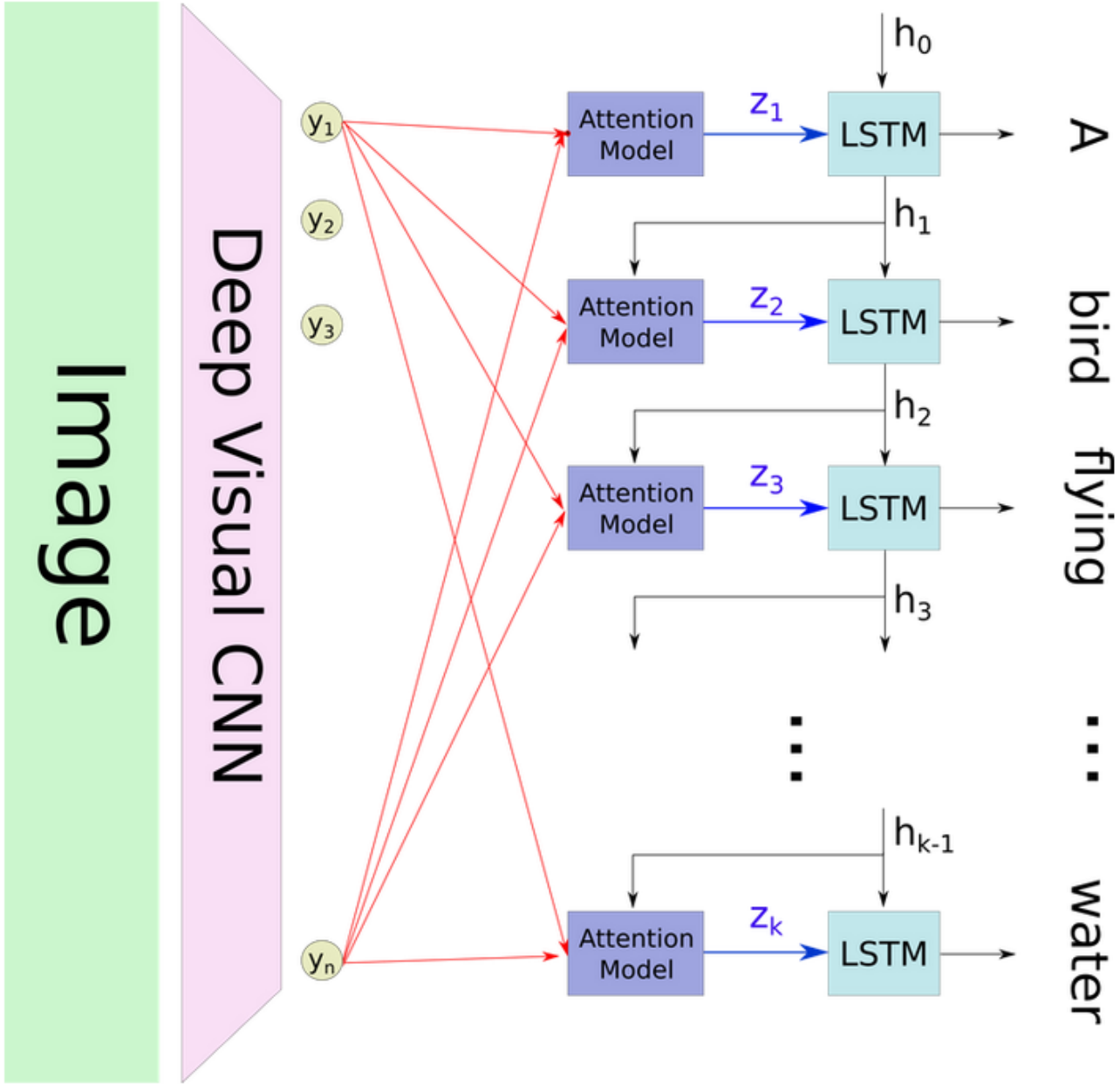
- Both systems have pros/cons, but the trend is to focus on soft attention mechanisms as the gradient can directly be computed instead of estimated through a stochastic process

Hard Attention



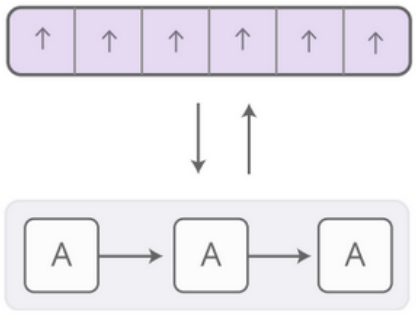
A Hard Attention model. The output is a random choice of one of the y_i, with probability s_i.

Now We Have a Deeper Understanding of the Image Captioning System We Examined Earlier

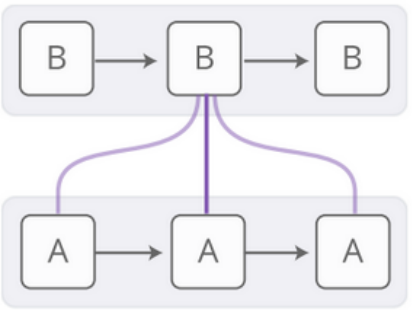


Attention model for image captioning

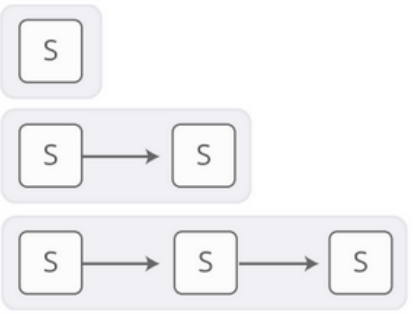
Applications of Attention Mechanisms



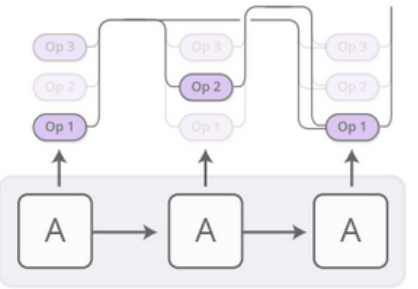
Neural Turing Machines
have external memory that they can read and write to.



Attentional Interfaces
allow RNNs to focus on parts of their input.



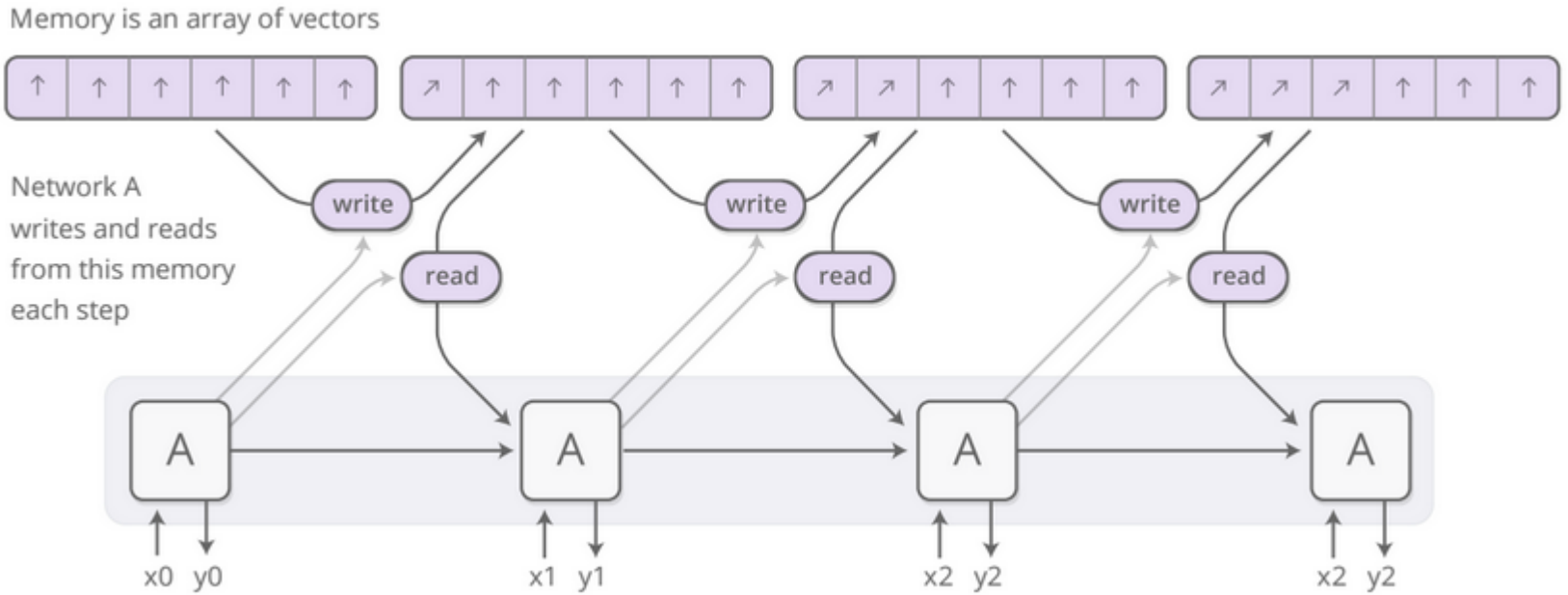
Adaptive Computation Time
allows for varying amounts of computation per step.



Neural Programmers
can call functions, building programs as they run.

Neural Turing Machines Combine an RNN with an External Memory Bank

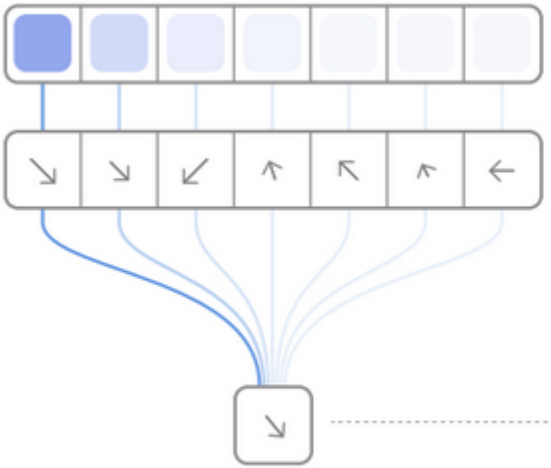
- Graves et al, 2014
- Memory is an array of vectors



- We want read/write operations to be differentiable with respect to the location we read/write to
  - This lets us learn where to read/write

- At each step, NTM's read/write everywhere, but to different extents
  - RNN computes an attention distribution, which describes how to spread out the amount we care about different memory positions

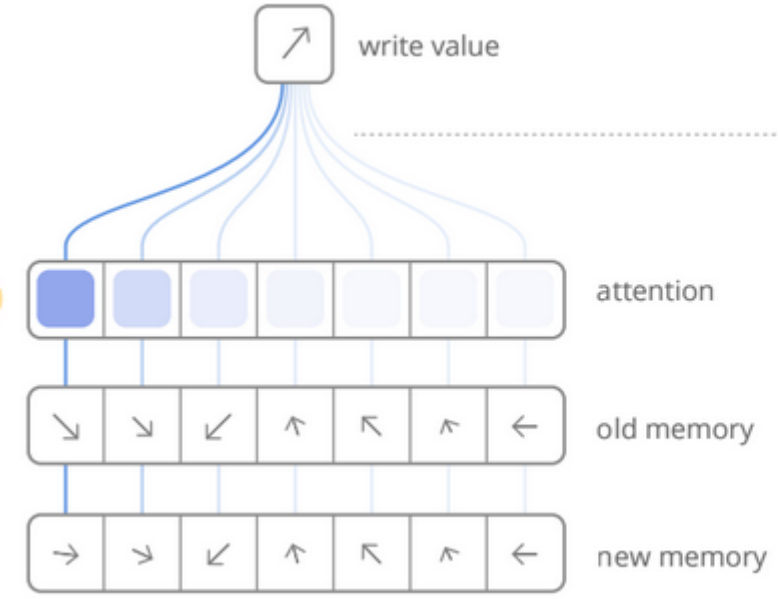
Neural Turing Machines Combine an RNN with an External Memory Bank



The RNN gives an attention distribution which describe how we spread out the amount we care about different memory positions

The read result is a weighted sum.

$$r \leftarrow \sum_i a_i M_i$$



Instead of writing to one location, we write everywhere, just do different extents.

The RNN gives an attention distribution, describing how much we should change each memory position towards the write value.

$$M_i \leftarrow a_i w + (1-a_i) M_i$$

Neural Turing Machines Combine an RNN with an External Memory Bank

- Content-based attention:
  - Allows NTM's to search through their memory and focus on places that match what they're looking for

- Location-based attention:
  - Allows relative movement in memory, enabling the NTM to loop

The Capability to Read/Write Allows NTM's To Perform Many Simple Algorithms

- Learn to store a long sequence in memory, and then loop over it, repeating it back repeatedly
- Learn to mimic a lookup table
- Advent of the neural GPU allows NTM's to add and multiply numbers (Kaiser & Sutskever, 2015)
- Neural random access machines work based on pointers.