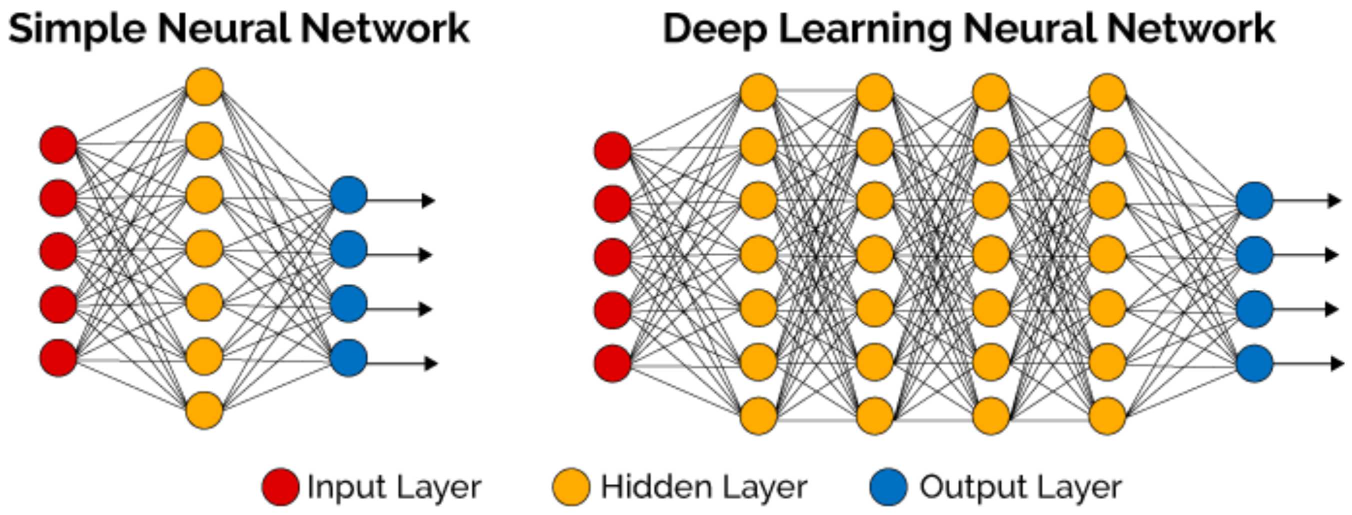
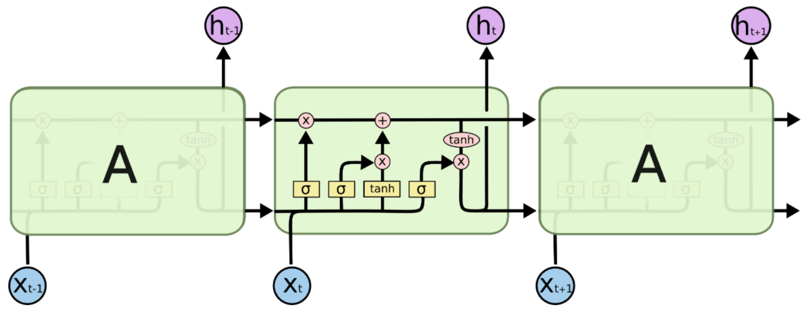
DSCI 552, Machine Learning for Data Science

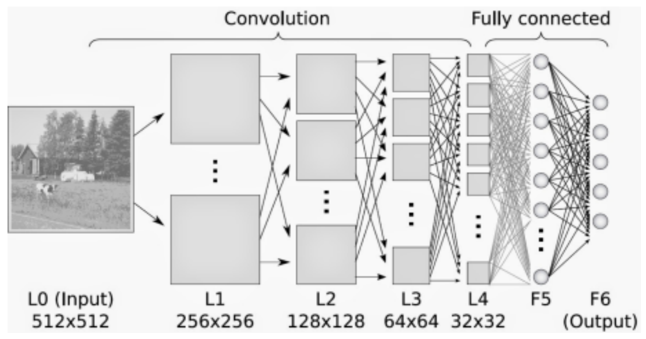
University of Southern California

M. R. Rajati, PhD

Lesson 10
Neural Networks and Deep Learning (Appendices)







Appendix

Unsupervised Learning/ Feature Learning Using NNs

&

Pre-Training

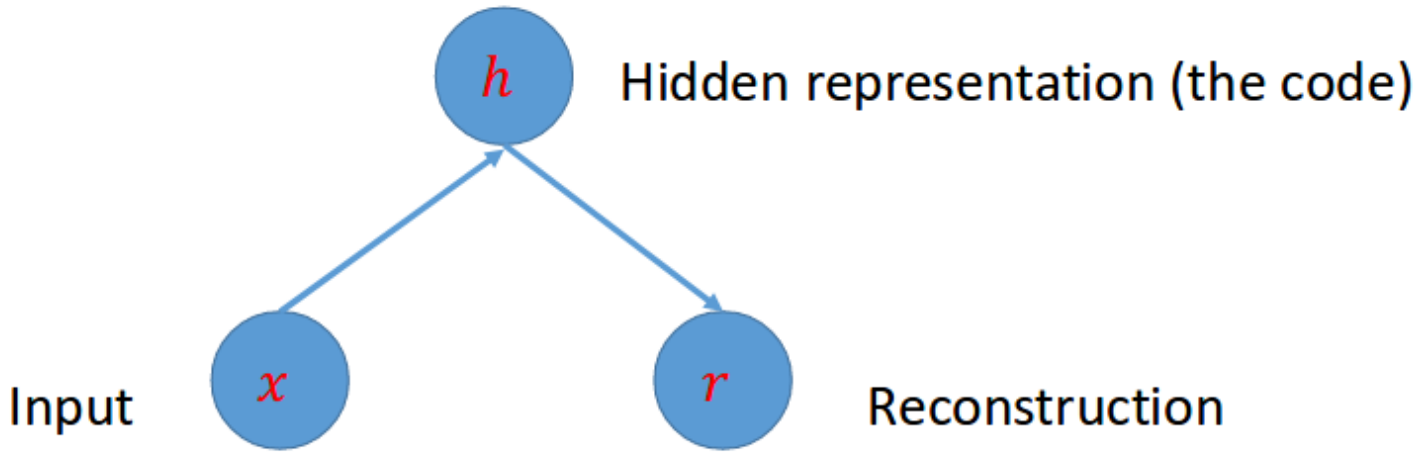
Unsupervised Learning/ Feature Learning Using NNs

- One can use MLPs to learn low dimensional representations of high dimensional data
- This corresponds to dimensionality reduction or learning features from data
- Similar in nature to PCA

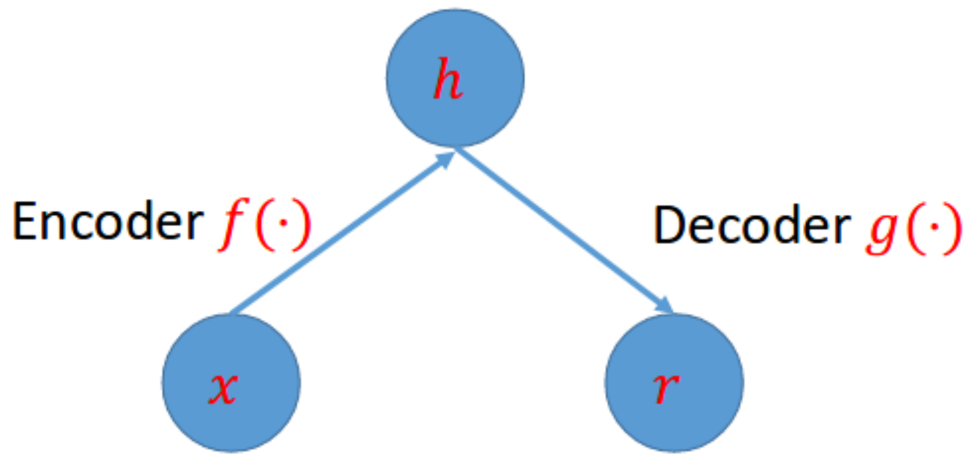
Autoencoders

- Neural network trained to attempt to copy its input to its output
- Contains two parts:
- Encoder: map the input to a hidden representation
- Decoder: map the hidden representation to the output

Autoencoders



Autoencoders



$$h = f(x), r = g(h) = g(f(x))$$

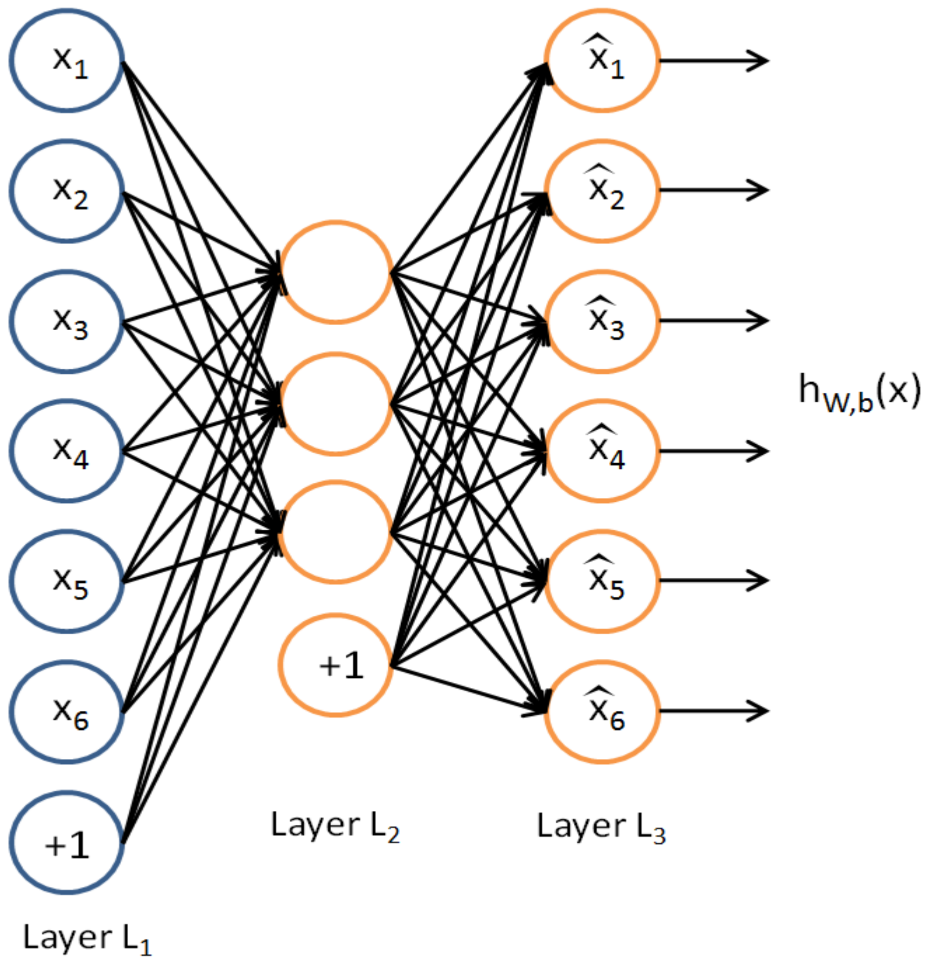
Why Learn the Identity Function?

- We do not really care about copying
- NN will NOT be able to copy exactly but strives to do so
- Autoencoder:
  - forced to select which aspects to preserve and thus
  - hopefully can learn useful properties of the data

Undercomplete Autoencoder

- Constrain the code to have smaller dimension than the input
- Training: minimize a loss function (e.g., MSE)
- **e** = **g**(**f**(**x**)) - **r**
- When both f and g are linear and MSE is used, the code contains just the Principal Components of the data

Undercomplete Autoencoder and PCA



Undercomplete Autoencoder

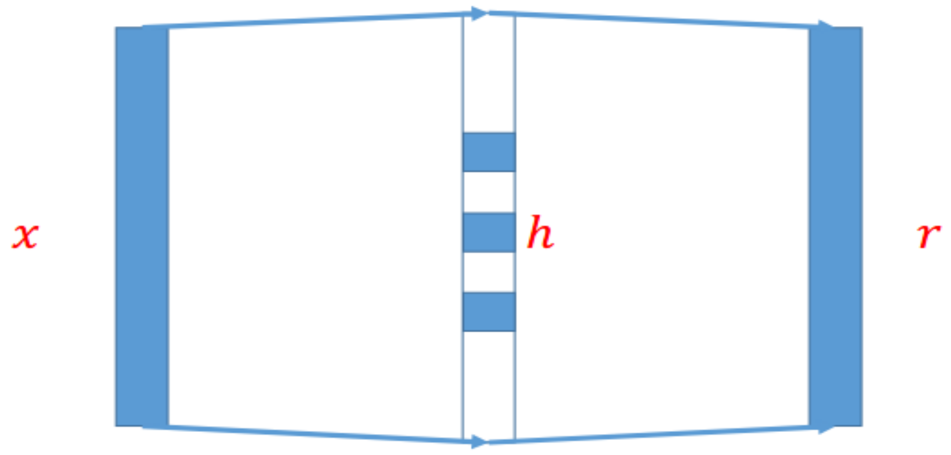
- Example of nonlinear encoder and decoder
- Capacity should not be too large
- Suppose given data
- **x**(1), **x**(2),..., **x**(N)
- Encoder maps **x**(*i*) to *i*
- Decoder maps *i* to **x**(*i*)
- One dimensional h suffices for perfect reconstruction

Regularization

- Often not performed.
- Regularized autoencoders: add regularization term that encourages the model to have other properties
- Sparsity of the representation (sparse autoencoder)
- Robustness to noise or to missing inputs (denoising autoencoder)
- Smallness of the derivative of the representation (smoothness)

Sparse Autoencoder

- Code is constrained to be sparse
- Code does not need to be “low dimensional”
- L1 or Probabilistic Regularization is used



Denoising Autoencoder

- In order to force the hidden layer to discover more robust features and prevent it from simply learning the identity, we train the auto-encoder to reconstruct the input from a corrupted version of it.
- **e** = **g**(**f**(**x**+**noise**)) – **r**

Denoising Auto-encoder

- The denoising auto-encoder is a *stochastic version of the auto-encoder.*
- It does two things: try to encode the input (preserve the information about the input), and try to undo the effect of a corruption process stochastically applied to the input of the auto-encoder.

Visualizing Auto-encoder

- Having trained a (sparse) auto-encoder, we would now like to visualize the function learned by the algorithm, to try to understand what it has learned.
- Consider the case of training an auto-encoder on 10×10 images, so that p=100.

Visualizing Autoencoder

- Each neuron *i* computes a function of the input:

$$a_i^{(1)} = f\left(\sum_{j=1}^{100} w_{ij}\, x_j + b_i\right)$$

- We will visualize the function computed by neuron *i*—which depends on the parameters $w_{ij}^{(1)}$ (ignoring the bias term for now)—using a 2D image.

Visualizing Autoencoder

- Think of $a_i^{(1)}$ as some non-linear feature of the input **x**.

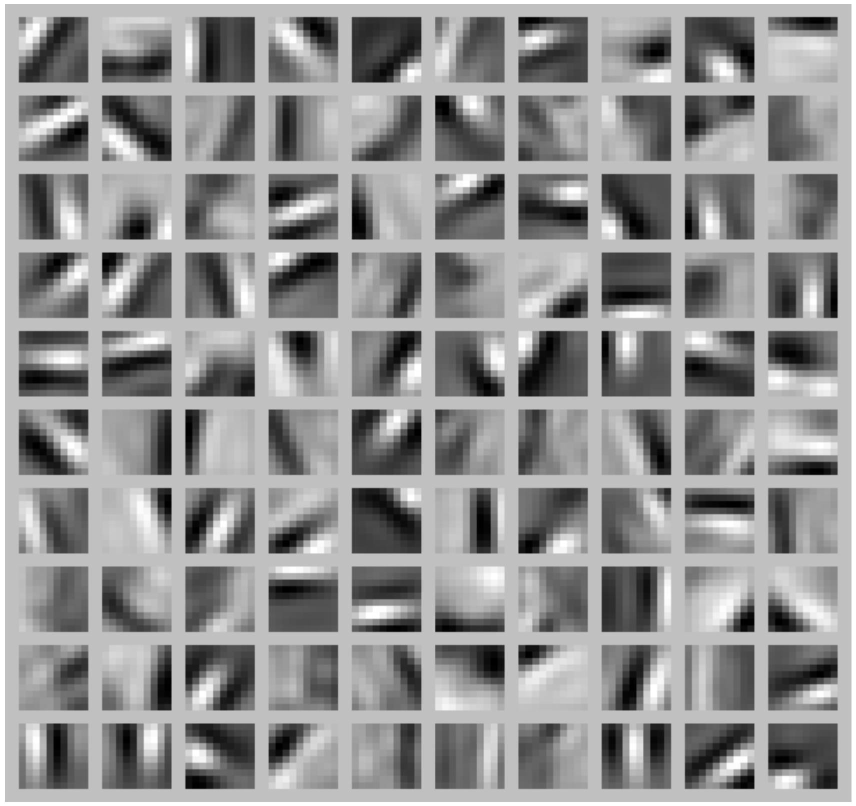
- What input image **x** would cause $a_i^{(1)}$ to be maximally activated?
- Less formally, what is the feature that neuron *i* is looking for?

Visualizing Autoencoder

- If we have an auto-encoder with 100 neurons, then we our visualization will have 100 such images—one per neuron.
- By examining these 100 images, we can try to understand what the ensemble of hidden units is learning.

Visualizing Autoencoder

- When we do this for a sparse autoencoder (trained with 100 neurons on 10x10 pixel inputs) we get the following result



Visualizing Autoencoder

- Each square in the figure shows the (normalized) input image **x** that maximally actives one of 100 neurons.
- Different neurons have learned to detect edges at different positions and orientations in the image.

Visualizing Autoencoder

- These features are useful for such tasks as object recognition and other vision tasks.
- When applied to other input domains (such as audio), this algorithm also learns useful representations/features for those domains too.

Problems with Using Very Deep Networks?

- A famous problem with training very deep neural networks with sigmoid hidden units is *vanishing gradients*
- When the number of layers is large, net signals that go into each neuron can be very negative or very positive

Problems with Using Very Deep Networks?

- Neurons become *saturated,* i.e. their output becomes close to 1 or -1
- Magnitudes of gradients in saturation areas are very small, so the weight updates will be very small and the network cannot learn from new data.

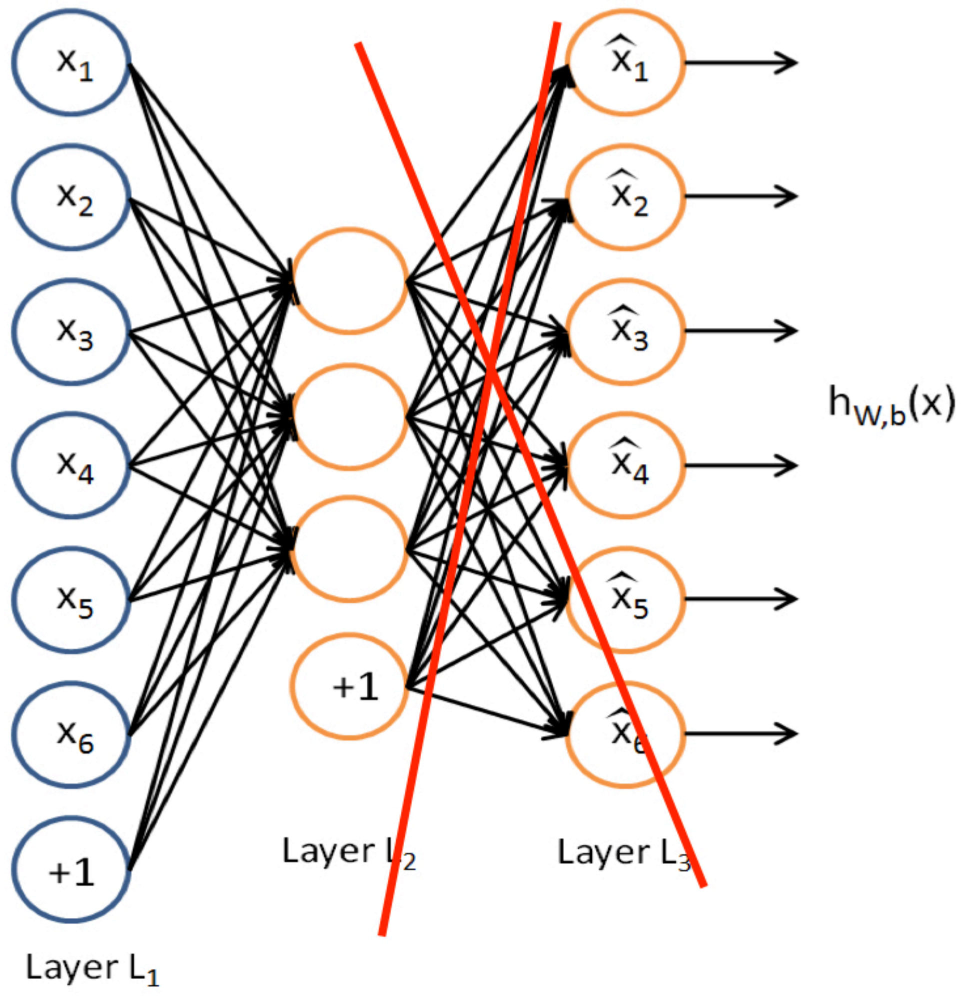
Remedy: Pre-training

- In 2006, Hinton et al showed that pre-training with auto-encoders can remedy the problem of vanishing gradients.
- In basic application of backpropagation to update the weights, weights are initialized randomly
- That can easily cause gradients to vanish

Remedy: Pre-training

- Instead, one can use layer-by-layer pre-training with auto-encoders
- In this method, an auto-encoder is used to learn a representation of the data.
- The weights of its output layer are discarded, but the weights from the input to hidden layer are kept as initial weights of the first layer.

Remedy: Pre-training



Remedy: Pre-training

- The outputs of the hidden layer can then be used to train a second auto-encoder.
- Again, the output weights of this auto-encoder are discarded.
- Only the weights from the input to hidden neurons of the second auto-encoder are used as initial weights of the second layer

Remedy: Pre-training

- This process continues by adding layers and training an auto-encoder for the output of the last layer.
- Layer by layer pre-training made training deep networks possible
- The initial pre-trained weights act as good "guesses" that guide the algorithm to minimum error.

Is Pre-training Outdated?

- Some think so!
- Reddit User:

"…but the last few years the pre-training approach has been largely obsoleted.
Nowadays, deep neural networks are a lot more similar to their 80's cousins. Instead of pre-training, the difference is now in the activation functions and regularisation methods used (and sometimes in the optimisation algorithm, although much more rarely)."

Is Pre-training Outdated?

- Some think so!
- See this discussion by a Famous Reddit User and the responses: (https://www.reddit.com/r/MachineLearning/comments/22u1yt/is_deep_learning_basically_just_neural_networks/cgqgy9w/)

"…but the last few years the pre-training approach has been largely obsoleted.
Nowadays, deep neural networks are a lot more similar to their 80's cousins. Instead of pre-training, the difference is now in the activation functions and regularisation methods used (and sometimes in the optimisation algorithm, although much more rarely)."

Is Pre-training Outdated?

- Some think so!
- Famous Reddit User:

"…the "pre-training era", which started around 2006, ended in the early '10s when people started using rectified linear units (ReLUs), and later dropout, and discovered that pre-training was no longer beneficial for this type of networks."

Is Pre-training Outdated?

- Some think so!
- Famous Reddit User:

“ReLUs (and modern variants such as maxout) suffer significantly less from the vanishing gradient problem. Dropout is a strong regulariser that helps ensure the solution we get generalises well. These are precisely the two issues pre-training sought to solve, but they are now solved in different ways.”

Is Pre-training Outdated?

- Some think so!
- Nonetheless, it is still used by the industry to some extent.
- It is very useful if labeled data is not enough to avoid overfitting by using dropout.
- It can be useful when unlabeled data is abundant: https://arxiv.org/pdf/1412.6597.pdf

Appendix

Adversarial Training

Adversarial Training

- Machine learning techniques were originally designed for stationary environments in which the training and test data are assumed to be generated from the same (although possibly unknown) distribution.

Adversarial Training

- In the presence of intelligent and adaptive adversaries, however, this working hypothesis is likely to be violated to at least some degree (depending on the adversary).

Adversarial Training

- In fact, a malicious adversary can carefully manipulate the input data exploiting specific vulnerabilities of learning algorithms to compromise the whole system security.

Do nets have Human-level understanding?

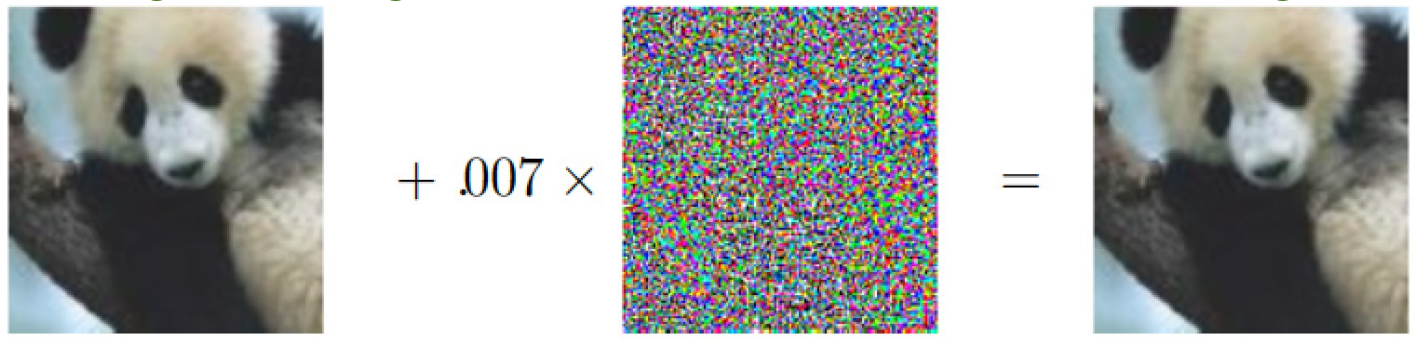
- In many cases, neural networks have begun to reach human level performance when evaluated on an i.i.d. test set
  - Have they reached human level understanding?
- To probe the level of understanding we can probe examples that model misclassifies
  - Even neural networks that perform at human level accuracy have a 100% error rate on examples intentionally constructed!

Adversarial examples

- An optimization procedure is used to search for an input ***x'*** near data point ***x*** such that the model output is very different at ***x'***
  - In many cases, ***x'*** can be so similar to ***x*** that a human observer cannot tell the difference between the original example and the adversarial example
  - But the network makes a highly different prediction

Adversarial Example Generation

We add to ***x*** an imperceptibly small vector
Its elements are equal to the sign of the elements of the gradient of the cost function with respect to the input. It changes Googlenet's classification of the image



$\boldsymbol{x}$

y ="panda"
with 58% confidence

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

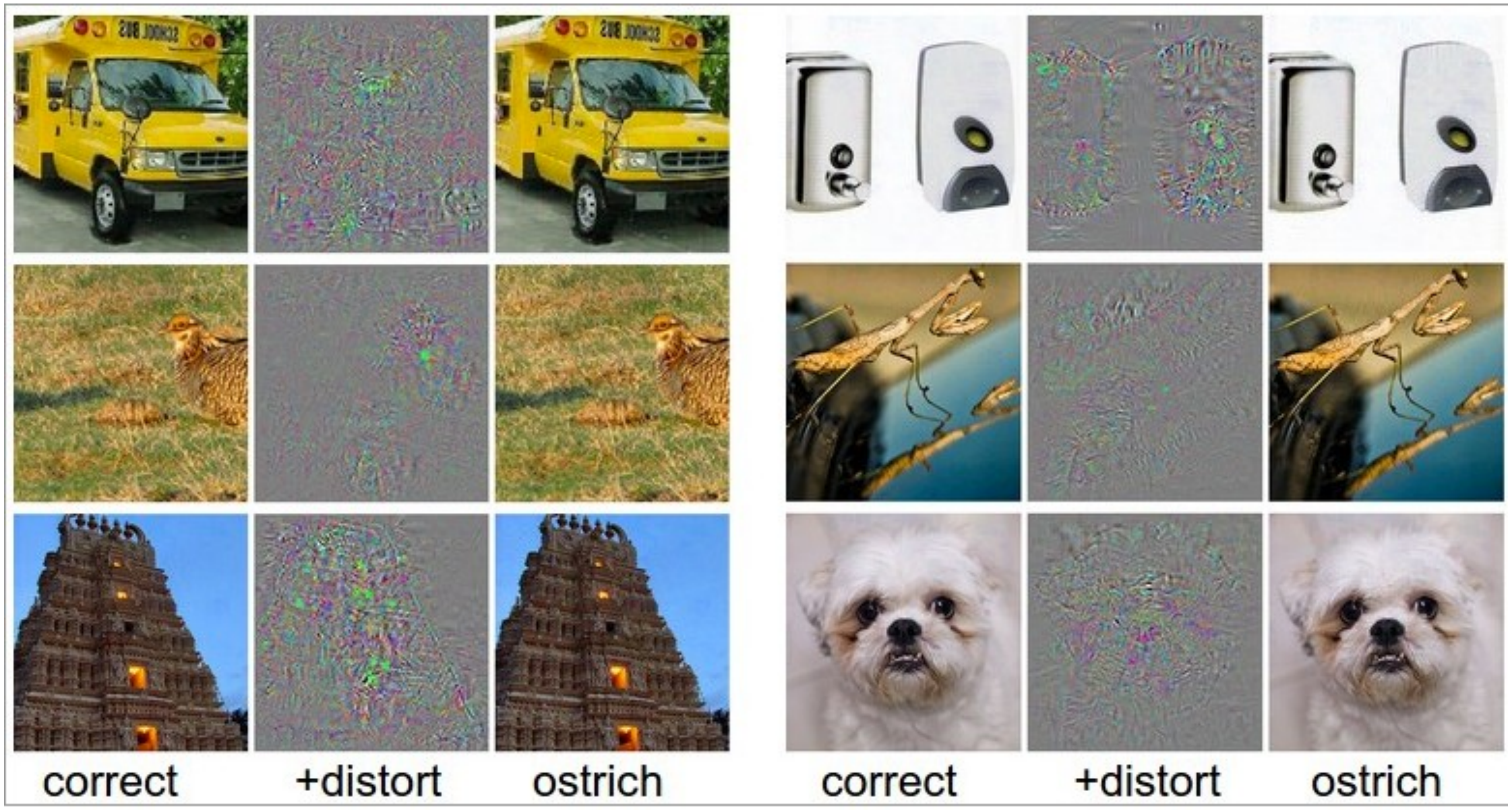
y ="nermatode"
With 8.2% confidence

$\boldsymbol{x} + \epsilon\,\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

y ="gibbon"
With 99% confiden4ce

https://arxiv.org/pdf/1412.6572.pdf

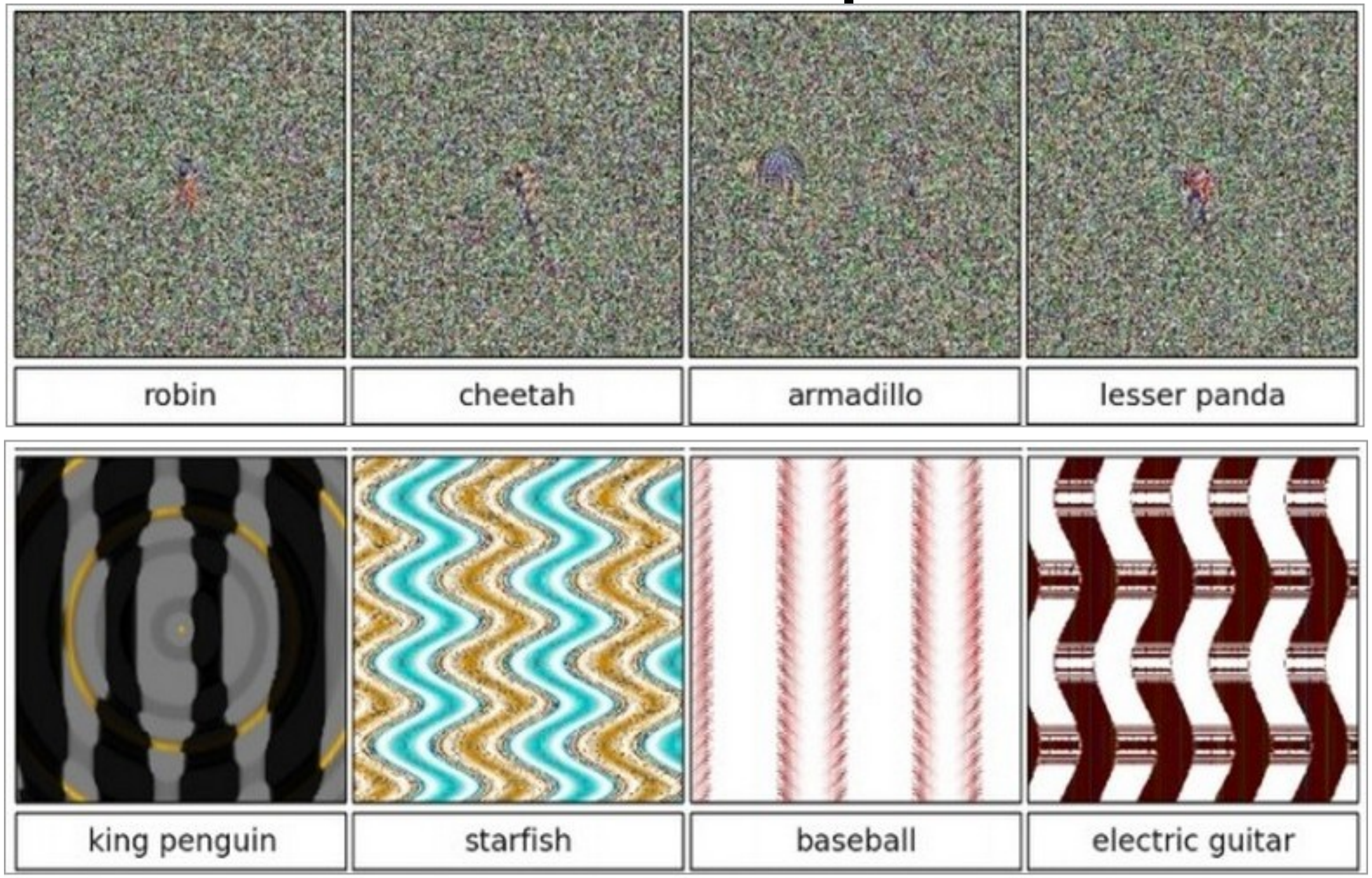
More examples



Take a correctly classified image (left image in both columns), and add a tiny distortion (middle) to fool the ConvNet with the resulting image (right).

http://karpathy.github.io/2015/03/30/breaking-convnets/

Some more examples



These images are classified with >99.6% confidence as the shown class by a Convolutional Network.

http://karpathy.github.io/2015/03/30/breaking-convnets/

Uses of adversarial training

- Adversarial examples have many implications
  - E.g., they are useful in computer security
    - Adversarial examples are hard to defend against
  - They are interesting in the context of regularization
    - Using adversarially perturbed samples we can reduce error rate on test set

Cause of adversarial examples

- Primary cause is excessive linearity
  - Neural networks are built primarily out of linear building blocks
    - The overall function often proves to be (close to) linear
  - Linear functions are easy to optimize
  - But the value of a linear function can change rapidly with numerous inputs
  - If we change input by ε then a linear functions with weights $\boldsymbol{w}$ can change by $\varepsilon||\boldsymbol{w}||$ which can be very large in high-dimensional spaces

Adversarial Training

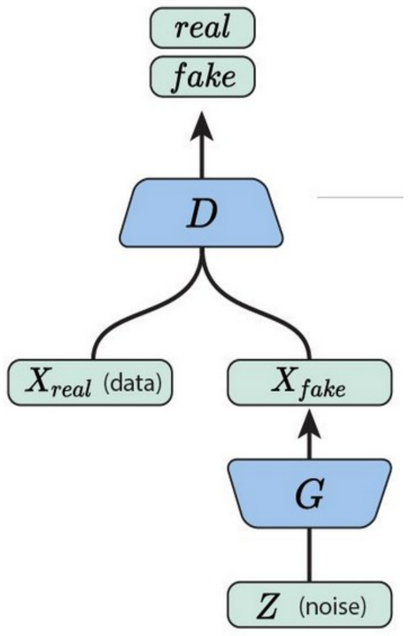
- Adversarial training discourages highly sensitive local behavior
- By encouraging network to be locally constant in the neighborhood of the training data
- This can be seen as a way of explicitly introducing a local constancy prior into supervised neural nets

Adversarial Training

- Locally constant behavior is robust to adversarial examples in comparison to linear models such as logistic regression.
- Neural networks are able to represent functions that can range from nearly linear to nearly locally constant
  - Thus can capture linear trends as well as learning to resist local perturbation

Generative Adversarial Network

- GANs are a way to make a generative model by having two neural networks compete with each other



The discriminator tries to distinguish genuine data from forgeries created by the generator

The generator turns random noise into imitations of the data, in an attempt to fool the discriminator

Generative Adversarial Network

- Training the discriminator involves presenting it with samples from the dataset, until it reaches some level of accuracy.
- Typically the generator is seeded with a randomized input

Generative Adversarial Network

- Thereafter, samples synthesized by the generator are evaluated by the discriminator.
- Backpropagation is applied in both networks so that the generator produces better images, while the discriminator becomes more skilled at flagging synthetic images.

Appendix

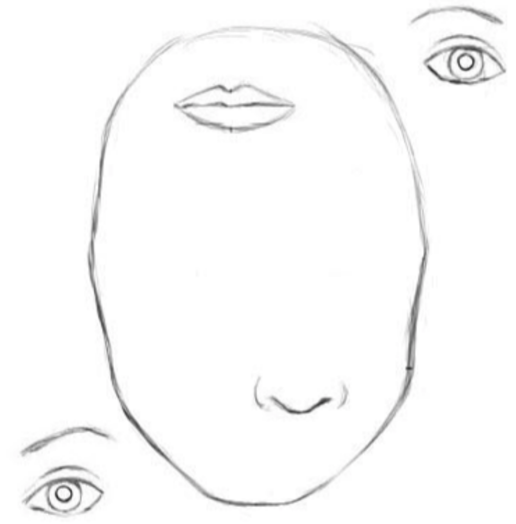
Capsule Networks

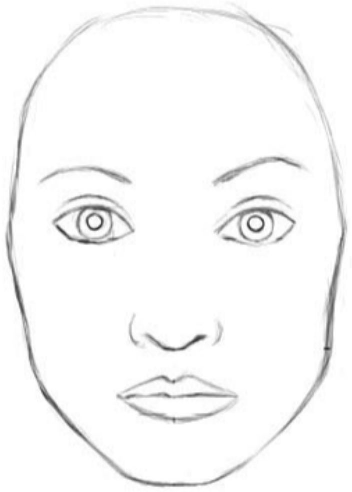
CNNs Have Important Drawbacks

- For a CNN, a mere presence of some objects can be a very strong indicator to consider that there is a pattern in the image.
- Orientational and relative spatial relationships between these components are not very important to a CNN.

Orientation? Spatial Relationships?

- Not considered important by design!

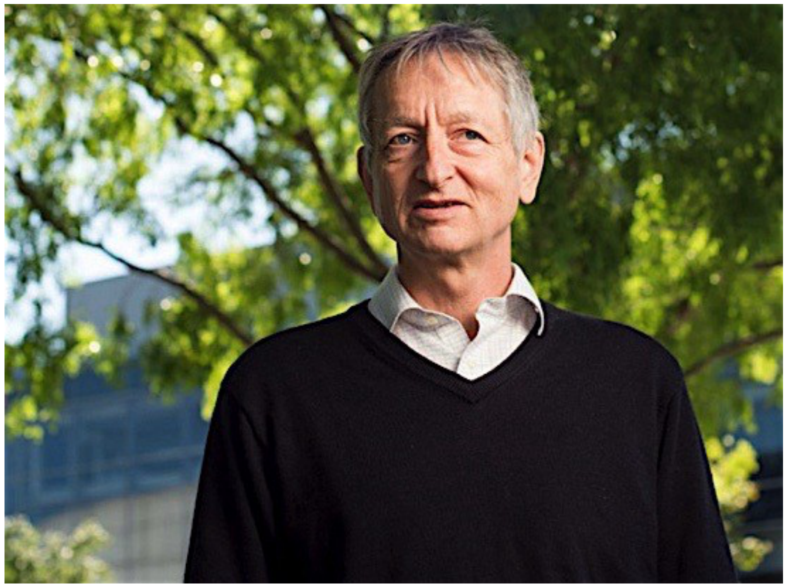




- To a CNN, both pictures are similar, since they both contain similar elements

Max Pooling: to praise or to blame?

- *Hinton: "The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster."*



Max Pooling: to praise or to blame?

- Hinton:
  - Brains deconstruct a hierarchical representation of the world around us from visual information received by eyes and try to match it with already learned patterns and stored relationships.
    - **Representation of objects in the brain does not depend on view angle**

Max Pooling: to praise or to blame?

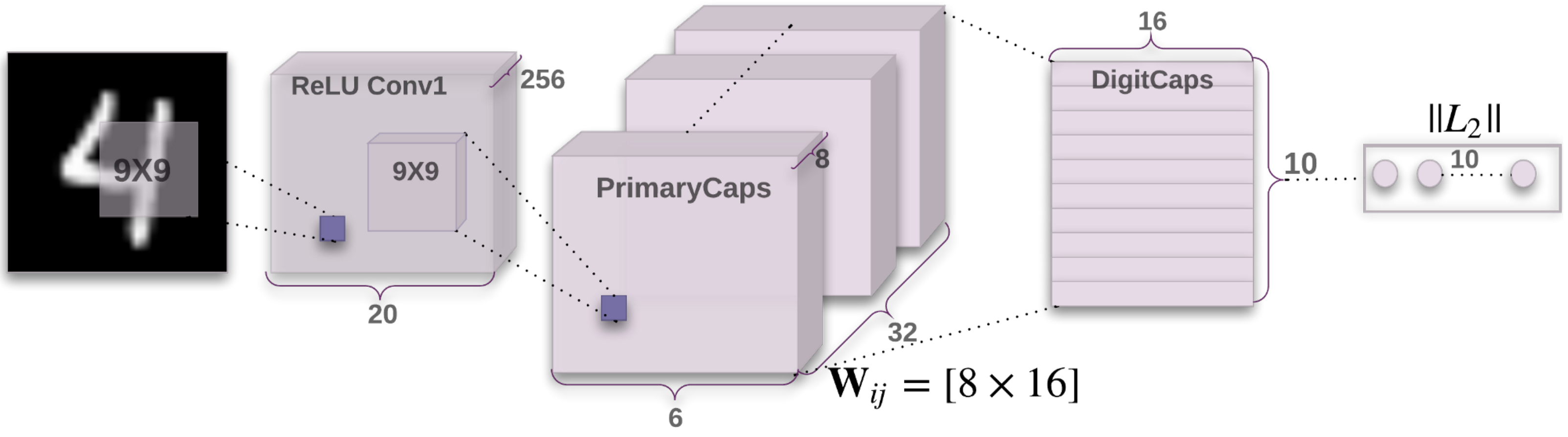
- Hinton:
  - Brains deconstruct a hierarchical representation of the world around us from visual information received by eyes and try to match it with already learned patterns and stored relationships.

Max Pooling: to praise or to blame?

- **Representation of objects in the brain does not depend on view angle**
- A CNN is easily confused when viewing an image in a different orientation.
- One way to combat this is with excessive training of all possible angles, but this takes a lot of time and seems counter intuitive.
- CNN is susceptible to adversarial patterns

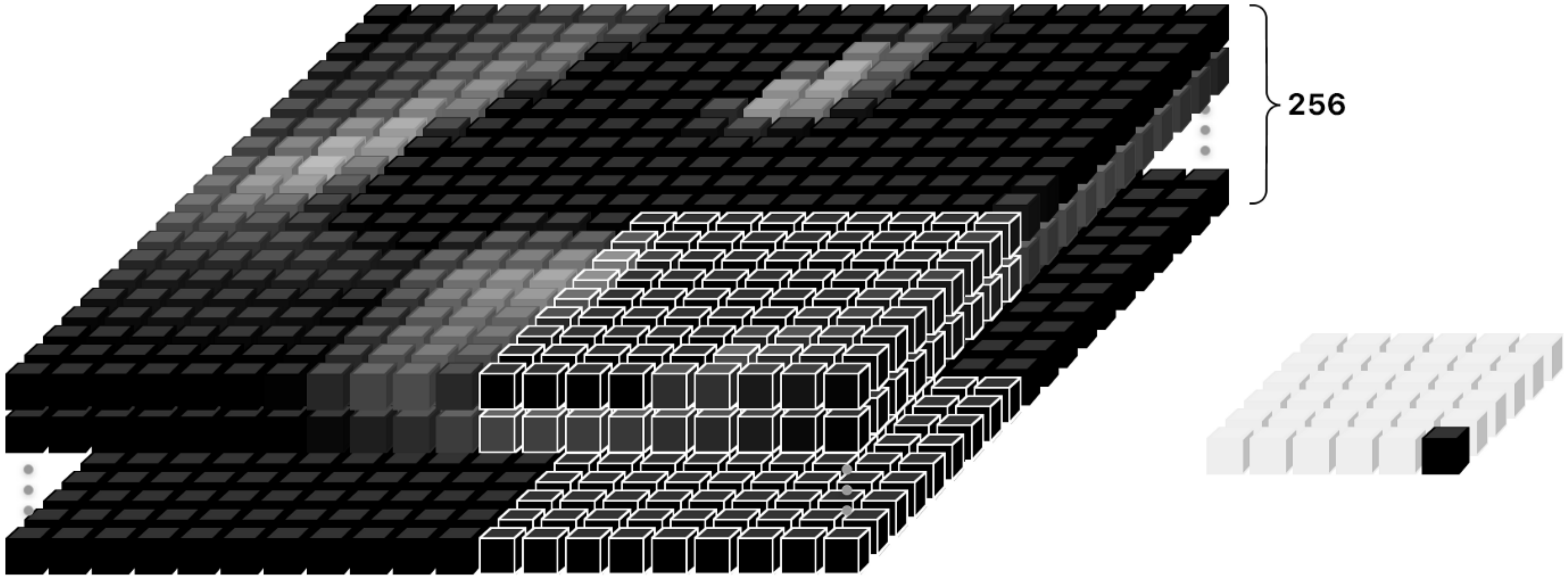
Capsule Networks

- The first part of CapsNet is a traditional convolutional layer passed through ReLU. Depth = 256.



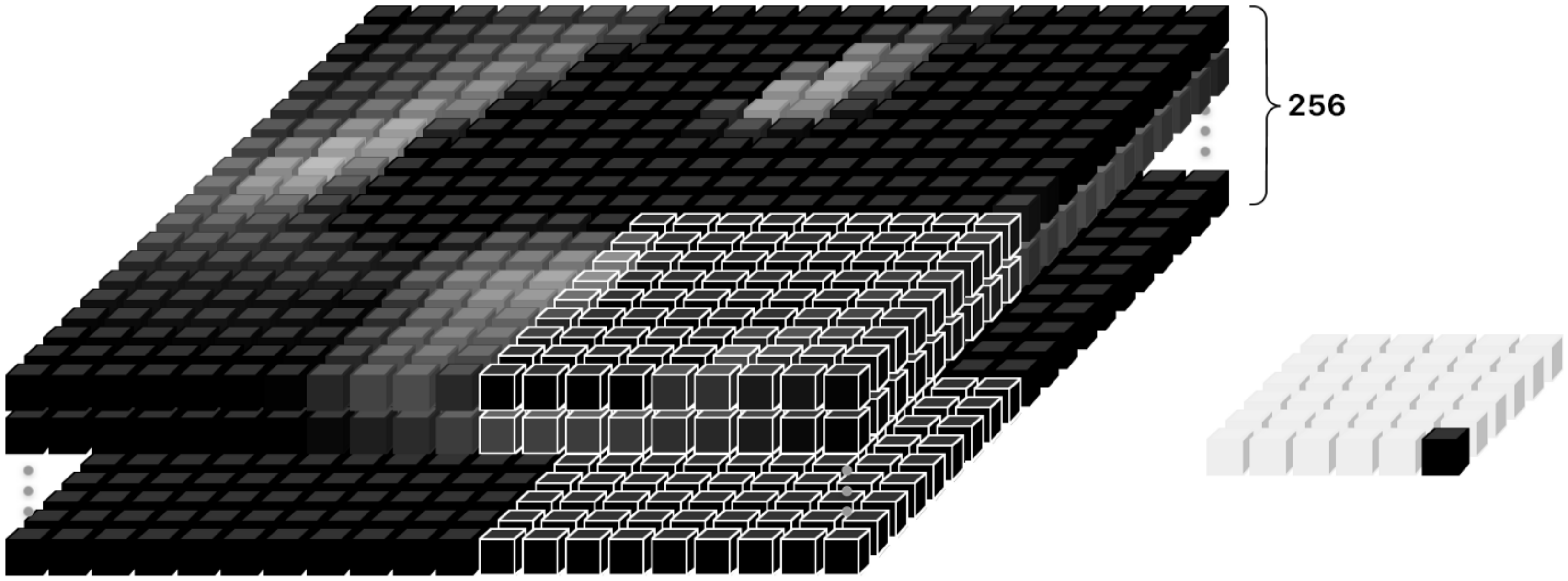
Primary Caps

- The PrimaryCaps layer starts off as a normal convolution layer, but this time we are convolving over the stack of outputs from the previous convolutions.



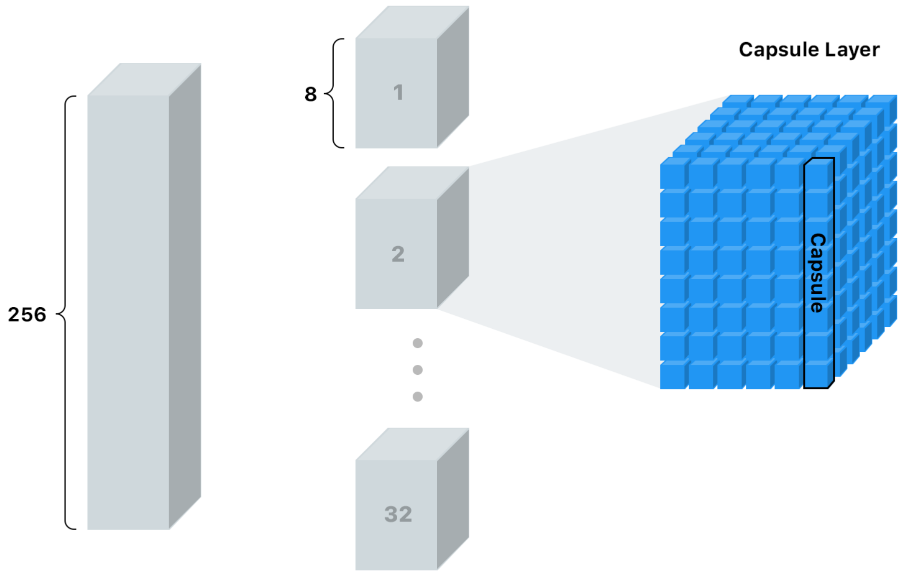
Primary Caps

- 256 6x6 feature maps
- Strides are >1, e.g. 2, to rapidly reduce dimensions
- Looking for slightly more complex shapes from the features (e.g. edges) found earlier



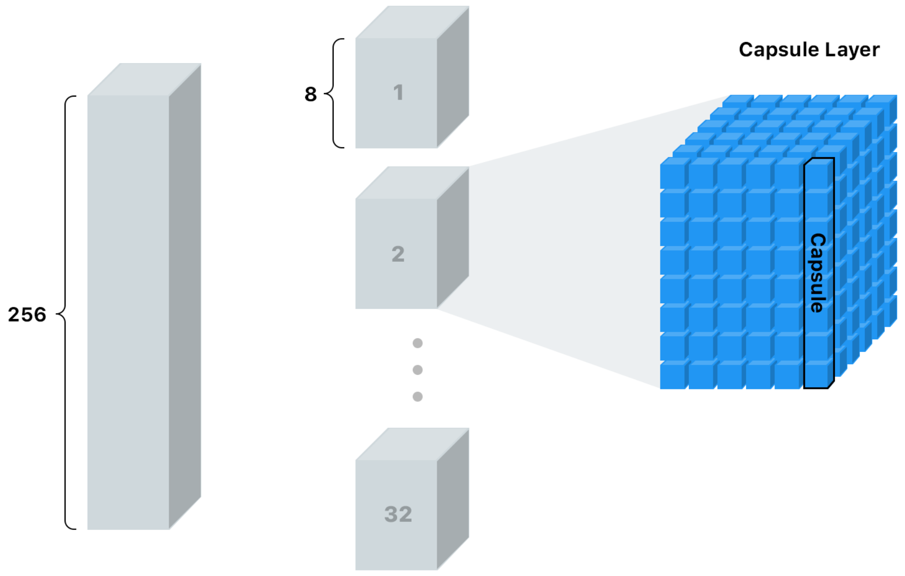
Primary Caps

- The capsule operation reshapes 256 6x6 outputs to 32 8x6x6 outputs
- This is basically slicing the cube into 32 smaller cubes, called “capsules”



Squashing (Instead of ReLU)

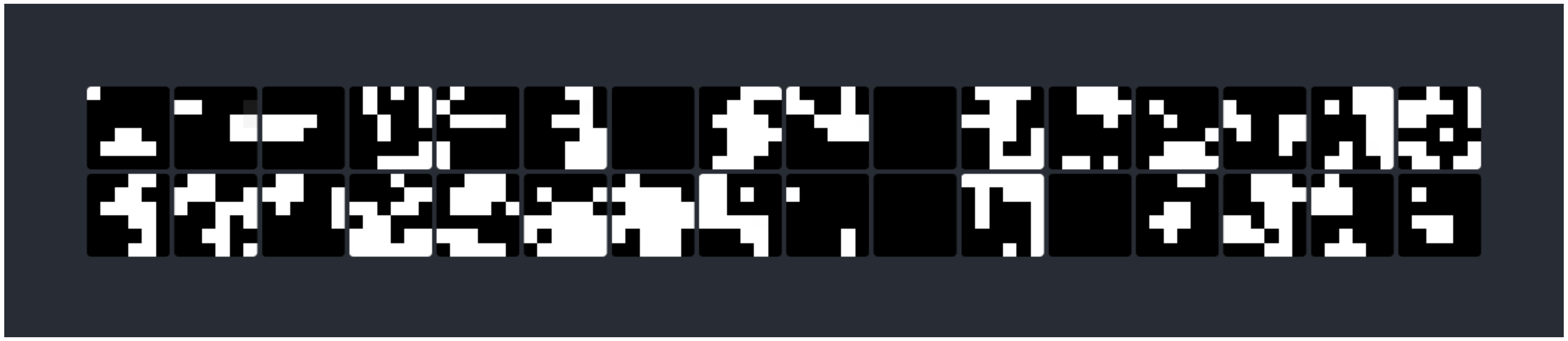
Each capsule is squashed (each vector of 8 elements is re-scaled so that its length is between 0 and 1 but its direction is not changed.)



$$\mathbf{v}_j = \frac{||\mathbf{s}_j||^2}{1 + ||\mathbf{s}_j||^2} \frac{\mathbf{s}_j}{||\mathbf{s}_j||}$$

Squashing (Instead of ReLU)

32 6X6 squares. Each pixel represents the length of a vector of 8 elements.



Routing by Agreement

The capsules in the next layers also try to detect objects and their pose, but they work very differently, using an algorithm called routing by agreement. This is where most of the magic of CapsNets lies.
https://www.youtube.com/watch?v=pPN8d0E3900

https://www.youtube.com/watch?v=YqazfBLLV4U

Appendix

- Representations of Natural Language Inputs and Outputs

Representations of Natural Language Inputs and Outputs

- When words are output at each time step, generally the output consists of a softmax vector $\mathbf{y}^{(t)}$ with *K elements* where K is the size of the vocabulary.
- Words are encoded with a binary (one hot) coding.

Representations of Natural Language Inputs and Outputs

- A softmax layer is an element-wise logistic function that is normalized so that all of its components sum to one.
- Intuitively, these outputs correspond to the probabilities that each word is the correct output at that time step.

Representations of Natural Language Inputs and Outputs

- Another simple architecture feeds the input one character at a time, and generates output one character at a time.
- Output is a softmax layer and characters are encoded with a binary (one-hot) encoding.

Representations of Natural Language Inputs and Outputs

- One hot encoding is inefficient, requiring as many bits as the vocabulary is large.
- It offers no direct way to capture different aspects of similarity between words in the encoding itself.

Distributed Representation of Natural Language

- It is common now to model words with a distributed representation using a meaning vector .
- The meanings are either learned given a large corpus of supervised data, or are based on word co-occurrence statistics.

Word Vectors

- Freely available code to produce word vectors from these statistics include GloVe and word2vec

Word2vec

- **Word2vec** is a group of related models used to produce word embeddings.
- These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words.

Word2vec

- **Word2vec**:
- **Input:** a large corpus of text
- **Output**: a vector space of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space.
- Words sharing common contexts in the corpus are located close to one another in the space

Evaluation of RNNs for Language Modeling: Obstacles

- A serious obstacle: outputs are variable length sequences of words.
- Captioning or translation: multiple correct translations.
- A labeled dataset may contain **multiple reference** translations for each example.
  - Comparing against such a gold standard is harder than applying performance measures to binary classification.

N-Grams

- An ***n*-gram** is a sequence of *n* items from a given sample of text or speech.
  - phonemes, syllables, letters, words or base pairs according to the application.
- When the items are words, *n*-grams may also be called ***shingles***

BLEU score

- It is the geometric mean of the n-gram precisions for all values of *n* between 1 and some upper limit *N*.
- In practice, 4 is a typical value for *N*.

BLEU score: Brevity

Precision can be made high by offering excessively short translations, so the BLEU score includes a brevity penalty *B*:

$$B = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}.$$

c is average the length of the candidate translations and *r* the average length of the reference translations.

BLEU score

The BLEU Score is:

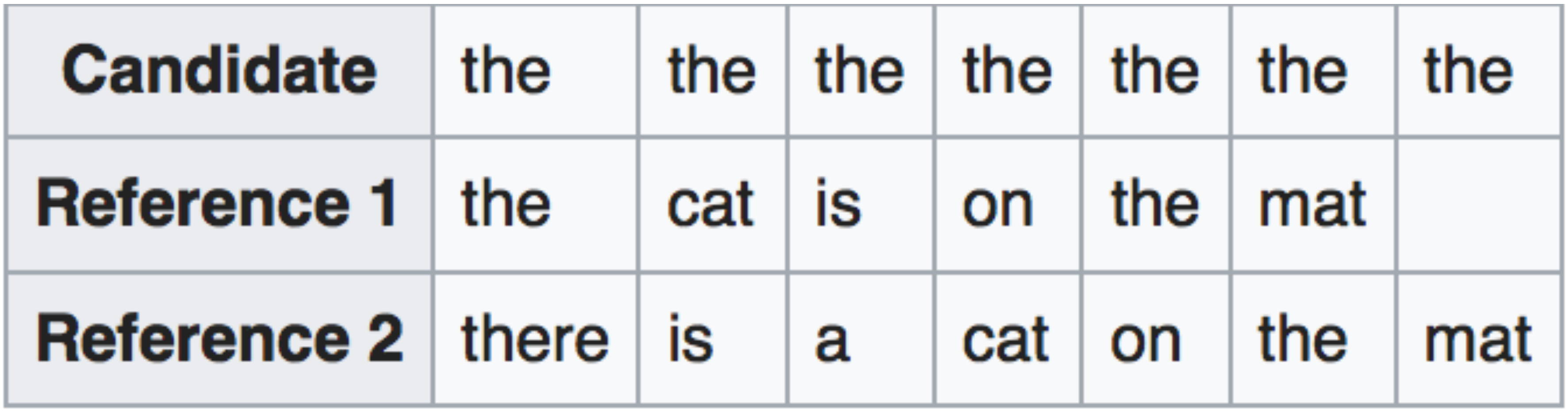
$$BLEU = B \cdot \exp\left(\frac{1}{N}\sum_{n=1}^{N} \log p_n\right)$$

BLEU score

$$BLEU = B \cdot \exp\left(\frac{1}{N}\sum_{n=1}^{N}\log p_n\right)$$

p_n : the modied n-gram precision, which is the number of n-grams
in the candidate translation that occur in any of the reference translations,
divided by the total number of n-grams in the candidate translation.

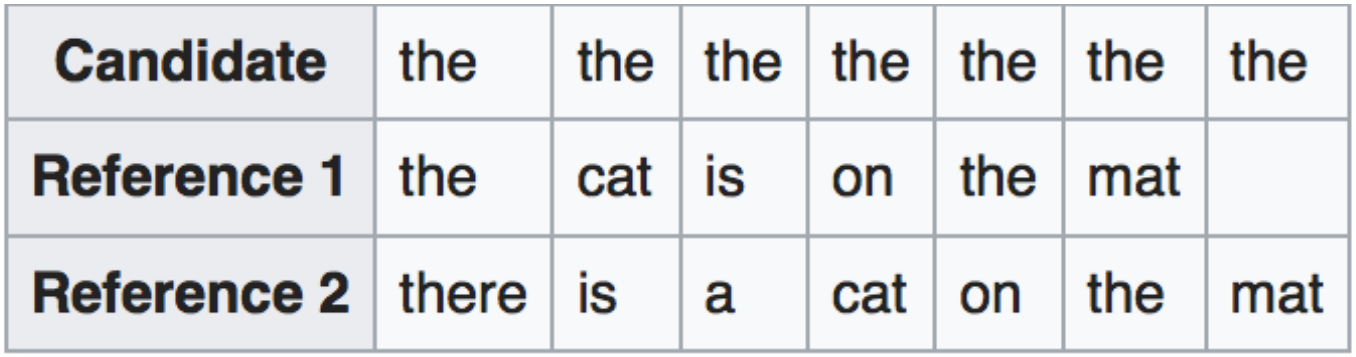
Unigram Metric



Candidate	the	the	the	the	the	the	the
Reference 1	the	cat	is	on	the	mat	
Reference 2	there	is	a	cat	on	the	mat

Of the seven words in the candidate translation, all of them appear in the reference translations. Thus the candidate text is given a unigram precision of **Words in the cand. found in ref/ total number of words in cand.=7/7**

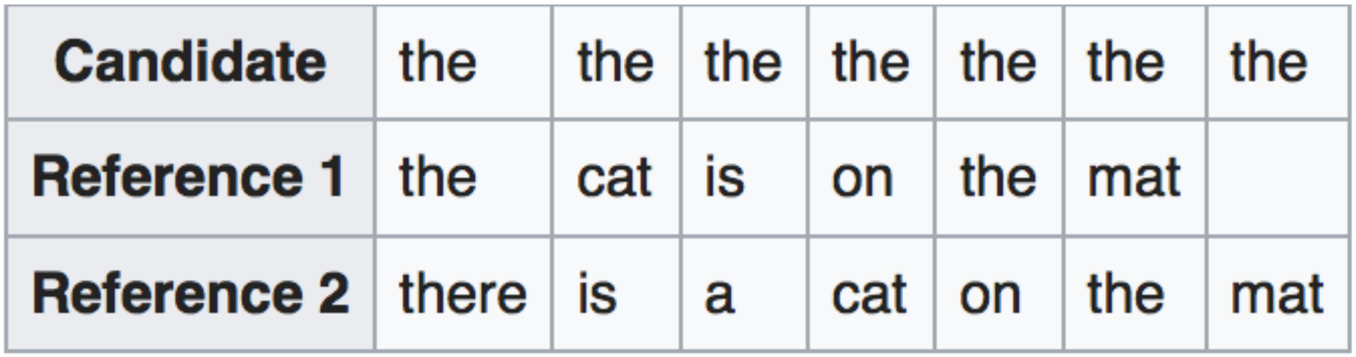
BLEU Modified Precision



Candidate	the	the	the	the	the	the	the
Reference 1	the	cat	is	on	the	mat	
Reference 2	there	is	a	cat	on	the	mat

For each word in the candidate translation, the algorithm takes its maximum total count m_{max} in any of the reference translations. In the example above, the word "the" appears twice in reference 1, and once in reference 2. Thus m_{max} =2

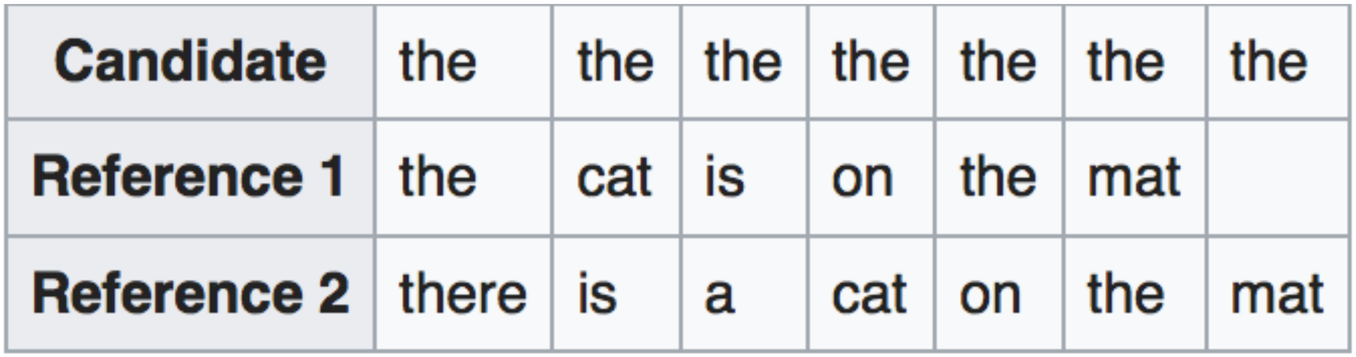
BLEU Modified Precision



Candidate	the	the	the	the	the	the	the
Reference 1	the	cat	is	on	the	mat	
Reference 2	there	is	a	cat	on	the	mat

For the candidate translation, the count m_w of each word is clipped to a maximum of m_{max} for that word

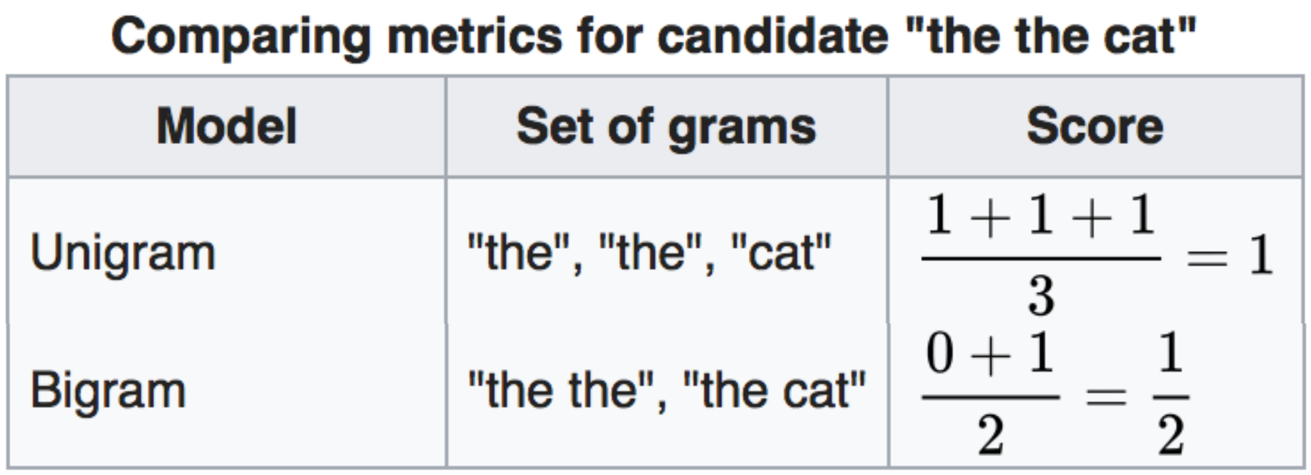
BLEU Modified Precision



Candidate	the	the	the	the	the	the	the
Reference 1	the	cat	is	on	the	mat	
Reference 2	there	is	a	cat	on	the	mat

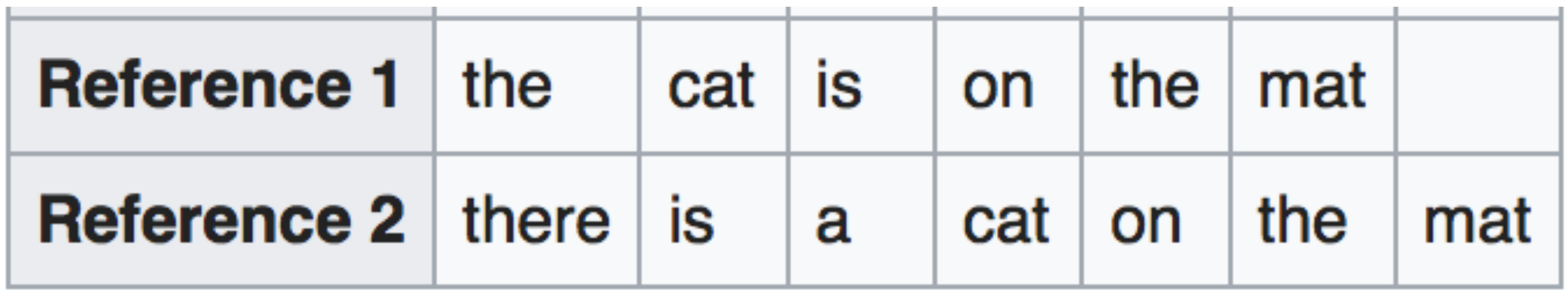
These clipped counts are then summed over all distinct words in the candidate. This sum is then divided by the total number of words (n-grams) in the candidate translation. In the above example, the modified unigram precision score would be:
2/7

BLEU Modified Precision



Comparing metrics for candidate "the the cat"

Model	Set of grams	Score
Unigram	"the", "the", "cat"	$\frac{1+1+1}{3} = 1$
Bigram	"the the", "the cat"	$\frac{0+1}{2} = \frac{1}{2}$



Reference 1	the	cat	is	on	the	mat		
Reference 2	there	is	a	cat	on	the	mat	

Research Other Methods

What are other evaluation methods?
Research them!
Example: METEOR

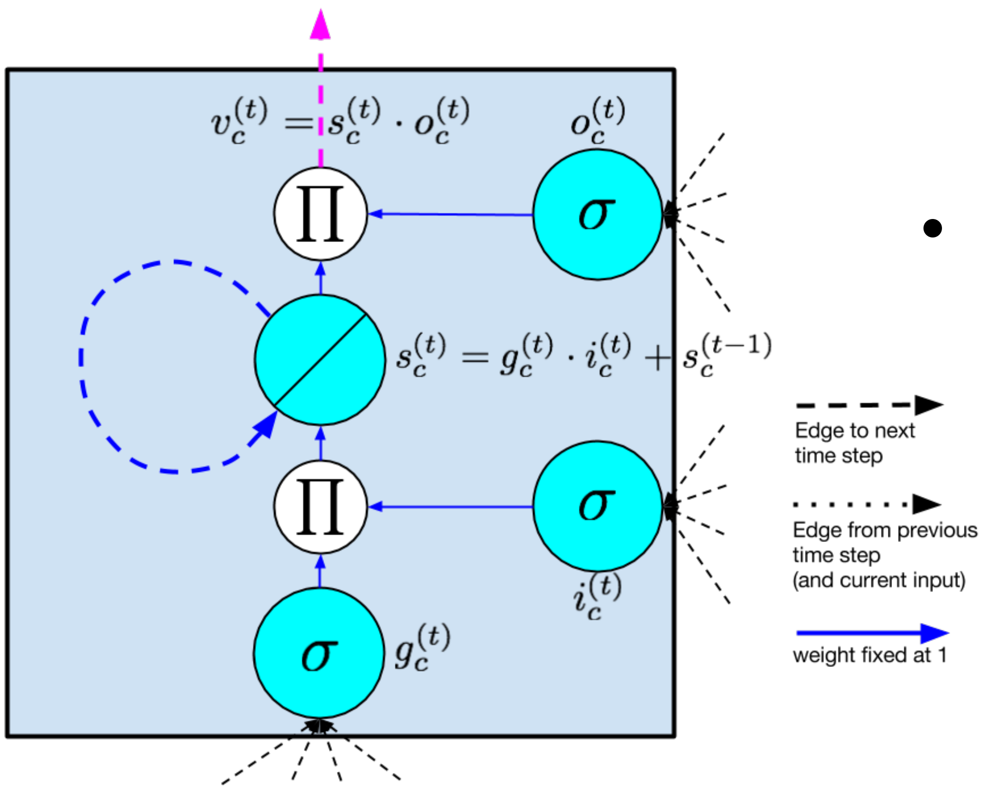
Appendix

Another Representation of LSTM

The LSTM

This model resembles
a standard recurrent neural
network with a hidden layer, but
each ordinary Neuron in the
hidden layer is replaced by a
memory cell.

The LSTM: Memory Cell



$v_c^{(t)} = s_c^{(t)} \cdot o_c^{(t)}$
$o_c^{(t)}$
Π
σ
$s_c^{(t)} = g_c^{(t)} \cdot i_c^{(t)} + s_c^{(t-1)}$
Π
σ
$i_c^{(t)}$
σ
$g_c^{(t)}$
Edge to next time step
Edge from previous time step (and current input)
weight fixed at 1

- One LSTM memory cell. The self-connected node is the internal state **s** .
- The diagonal line indicates that the identity link function is applied.
- The blue dashed line is the recurrent edge, which has fixed unit weight. Nodes marked Π output the product of their inputs. All edges into and from Π nodes also have fixed unit weight.

The LSTM: Memory Cell

- Each memory cell contains a node with a self-connected recurrent edge of fixed weight one, ensuring that the gradient can pass across many time steps without vanishing or exploding.
- To distinguish references to a memory cell and not an ordinary neuron, we use the subscript c .

The LSTM: Input Node

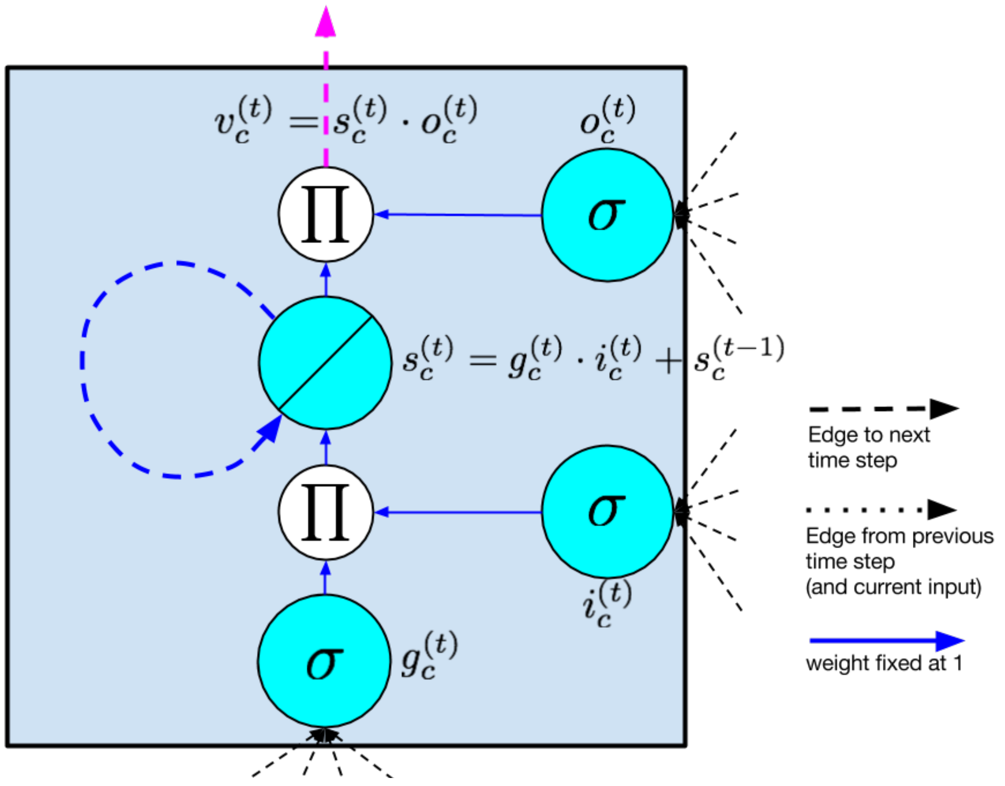
- Labeled g_c, it takes activation from the input layer $\mathbf{x}^{(t)}$ at the current time step and (along recurrent edges) from the hidden layer at the previous time step $\mathbf{h}^{(t\text{-}1)}$.
- Typically, the summed weighted input is run through a *tanh* (or sigmoid) activation function.

The LSTM: Input Gate

- Gates are a distinctive feature of the LSTM approach.
- A gate is a sigmoidal unit that, like the input node, takes activation from the current data point $\mathbf{x}^{(t)}$ as well as from the hidden layer at the previous time step.

The LSTM: Input Gate

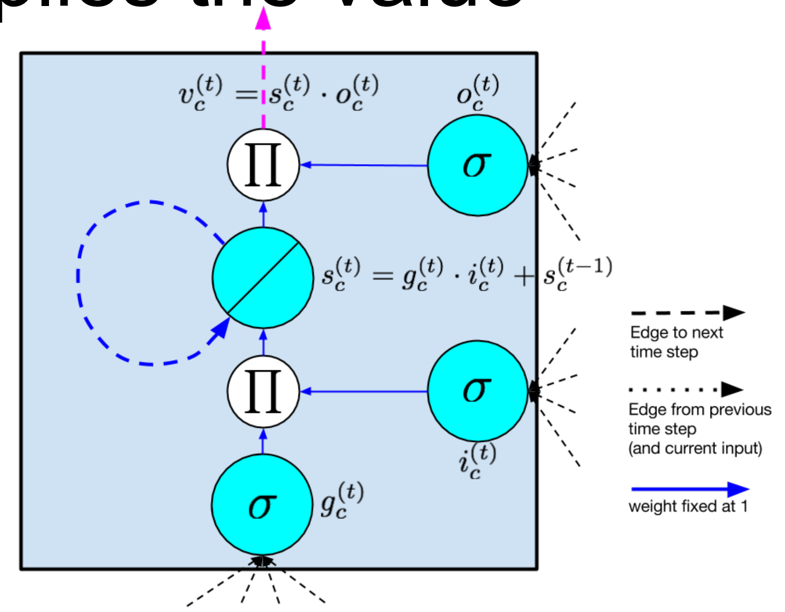
- A gate is so-called because its value is used to multiply the value of another node.



$v_c^{(t)} = s_c^{(t)} \cdot o_c^{(t)}$
$o_c^{(t)}$
Π
σ
$s_c^{(t)} = g_c^{(t)} \cdot i_c^{(t)} + s_c^{(t-1)}$
Π
σ
$i_c^{(t)}$
σ
$g_c^{(t)}$
Edge to next time step
Edge from previous time step (and current input)
weight fixed at 1

The LSTM: Input Gate

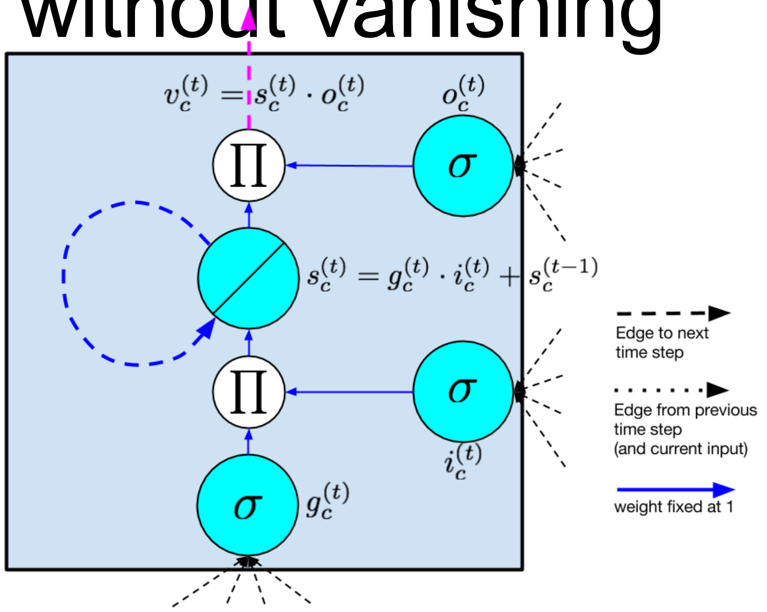
- It is a gate because if its value is zero, then flow from the other node is cut off.
- If the value of the gate is one, all flow is passed through. The value of the input gate *i_c* multiplies the value of the input node.



$v_c^{(t)} = s_c^{(t)} \cdot o_c^{(t)}$
$o_c^{(t)}$
Π
σ
$s_c^{(t)} = g_c^{(t)} \cdot i_c^{(t)} + s_c^{(t-1)}$
Π
σ
$i_c^{(t)}$
σ
$g_c^{(t)}$
Edge to next time step
Edge from previous time step (and current input)
weight fixed at 1

The LSTM: Internal State

- The heart of the memory cell is a node s_c with linear activation
- s_c has a self-connected recurrent edge with fixed unit weight. Because this edge spans adjacent time steps with constant weight, error can flow across time steps without vanishing or exploding.



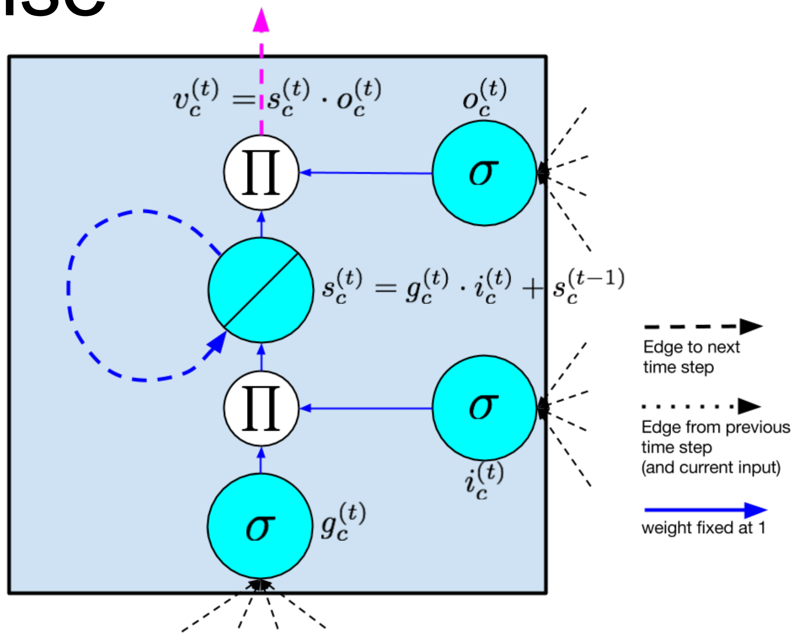
$v_c^{(t)} = s_c^{(t)} \cdot o_c^{(t)}$
$o_c^{(t)}$
Π
σ
$s_c^{(t)} = g_c^{(t)} \cdot i_c^{(t)} + s_c^{(t-1)}$
Π
σ
$i_c^{(t)}$
σ
$g_c^{(t)}$
Edge to next time step
Edge from previous time step (and current input)
weight fixed at 1

The LSTM: Internal State

- This constant edge is often called the constant error carousel.
- In vector notation, the update for the internal state is:

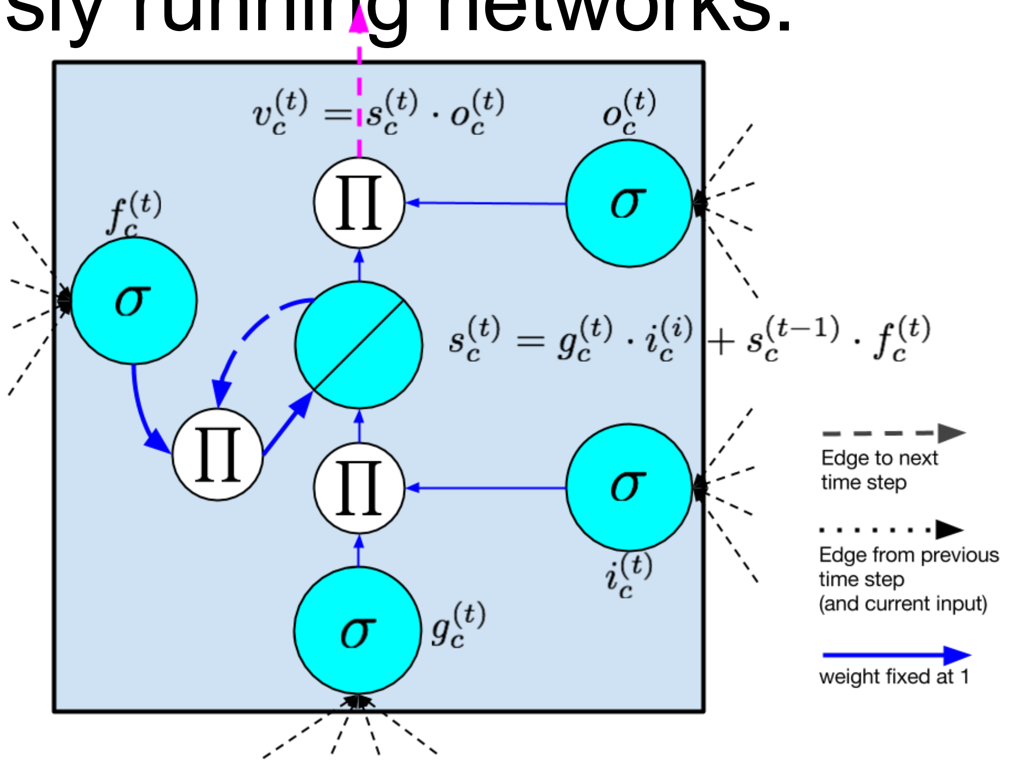
  $\mathbf{s}^{(t)} = \mathbf{g}^{(t)} \odot \mathbf{i}^{(t)} + \mathbf{s}^{(t-1)}$

  where $\odot$ is elementwise multiplication.


$v_c^{(t)} = s_c^{(t)} \cdot o_c^{(t)}$
$o_c^{(t)}$
Π
σ
$s_c^{(t)} = g_c^{(t)} \cdot i_c^{(t)} + s_c^{(t-1)}$
Π
σ
$i_c^{(t)}$
σ
$g_c^{(t)}$
Edge to next time step
Edge from previous time step (and current input)
weight fixed at 1

The LSTM: Forget Gate

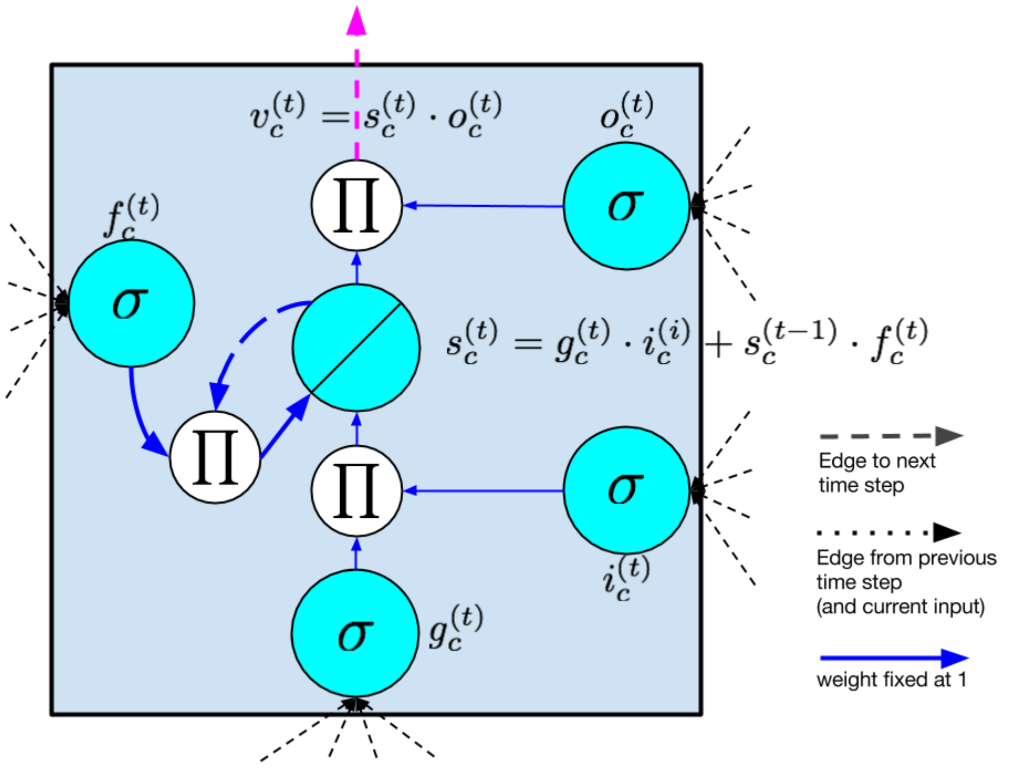
- Forget gates provide a method by which the network can learn to flush the contents of the internal state.
- This is especially useful in continuously running networks.



$v_c^{(t)} = s_c^{(t)} \cdot o_c^{(t)}$
$o_c^{(t)}$
$f_c^{(t)}$
σ
Π
$s_c^{(t)} = g_c^{(t)} \cdot i_c^{(i)} + s_c^{(t-1)} \cdot f_c^{(t)}$
$i_c^{(t)}$
$g_c^{(t)}$
Edge to next time step
Edge from previous time step (and current input)
weight fixed at 1

The LSTM: Forget Gate

- With forget gates, the equation to calculate the internal state on the forward pass is
  $\mathbf{s}^{(t)} = \mathbf{g}^{(t)} \odot \mathbf{i}^{(t)} + \mathbf{f}^{(t)} \odot \mathbf{s}^{(t\text{-}1)}$


$v_c^{(t)} = s_c^{(t)} \cdot o_c^{(t)}$
$o_c^{(t)}$
$f_c^{(t)}$
$s_c^{(t)} = g_c^{(t)} \cdot i_c^{(i)} + s_c^{(t-1)} \cdot f_c^{(t)}$
$i_c^{(t)}$
$g_c^{(t)}$
σ
$\prod$
Edge to next time step
Edge from previous time step (and current input)
weight fixed at 1

The LSTM: Output Gate

The value $\mathbf{v}_c$ ultimately produced by a memory cell is the value of the internal state $\mathbf{s}_c$ multiplied by the value of the output gate $\mathbf{o}_c$.

$$\mathbf{v}^{(t)} = \mathbf{s}^{(t)} \odot \mathbf{o}^{(t)}$$

It is customary that the internal state first be run through a tanh activation function, as this gives the output of each cell the same dynamic range as an ordinary tanh hidden unit.

The LSTM: Output Gate

However, in other neural network research, rectified linear units, which have a greater dynamic range, are easier to train. Thus it seems plausible that the nonlinear function on the internal state might be omitted.

The LSTM: Equations

$$\mathbf{g}^{(t)} = \boldsymbol{\phi}(\mathbf{W}_{gx}{}^{\mathbf{T}}\,\mathbf{x}^{(t)} + \mathbf{W}_{gh}{}^{\mathbf{T}}\,\mathbf{h}^{(t-1)} + \mathbf{b_g})$$

$$\mathbf{i}^{(t)} = \boldsymbol{\sigma}(\mathbf{W}_{ix}{}^{\mathbf{T}}\,\mathbf{x}^{(t)} + \mathbf{W}_{ih}{}^{\mathbf{T}}\,\mathbf{h}^{(t-1)} + \mathbf{b_i})$$

$$\mathbf{f}^{(t)} = \boldsymbol{\sigma}(\mathbf{W}_{fx}{}^{\mathbf{T}}\,\mathbf{x(t)} + \mathbf{W}_{fh}{}^{\mathbf{T}}\,\mathbf{h}^{(t-1)} + \mathbf{b_f})$$

$$\mathbf{o}^{(t)} = \boldsymbol{\sigma}(\mathbf{W}_{ox}{}^{\mathbf{T}}\,\mathbf{x}^{(t)} + \mathbf{W}_{oh}{}^{\mathbf{T}}\,\mathbf{h}^{(t-1)} + \mathbf{b_o})$$

$$\mathbf{s}^{(t)} = \mathbf{g(t)} \odot \mathbf{i}^{(t)} + \mathbf{s}^{(t-1)} \odot \mathbf{f}^{(t)}$$

$$\mathbf{h}^{(t)} = \boldsymbol{\phi}(\mathbf{s}^{(t)}) \odot \mathbf{o}^{(t)}$$

ϕ:tanh

σ: sigmoid

The LSTM: How It Works

Intuitively, in terms of the forward pass, the LSTM can learn when to let activation into the internal state. As long as the input gate takes value zero, no activation can get in.

$$\mathbf{s}^{(t)} = \mathbf{g(t)} \odot \mathbf{i}^{(\underline{t})} + \mathbf{s}^{(t\text{-}1)} \odot \mathbf{f}^{(\underline{t})}$$

The LSTM: How It Works

Similarly, the output gate learns when to let the value out. When both gates are closed , the activation is trapped in the memory cell, neither growing nor shrinking, nor affecting the output at intermediate
time steps.

The LSTM: How It Works

- In terms of the backwards pass, the constant error carousel enables the gradient to propagate back across many time steps, neither exploding nor vanishing.

The LSTM: How It Works

- The gates are learning when to let error in, and when to let it out.
- In practice, the LSTM has shown a superior ability to learn long range dependencies as compared to simple RNNs.