法律声明

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

关注 小象学院

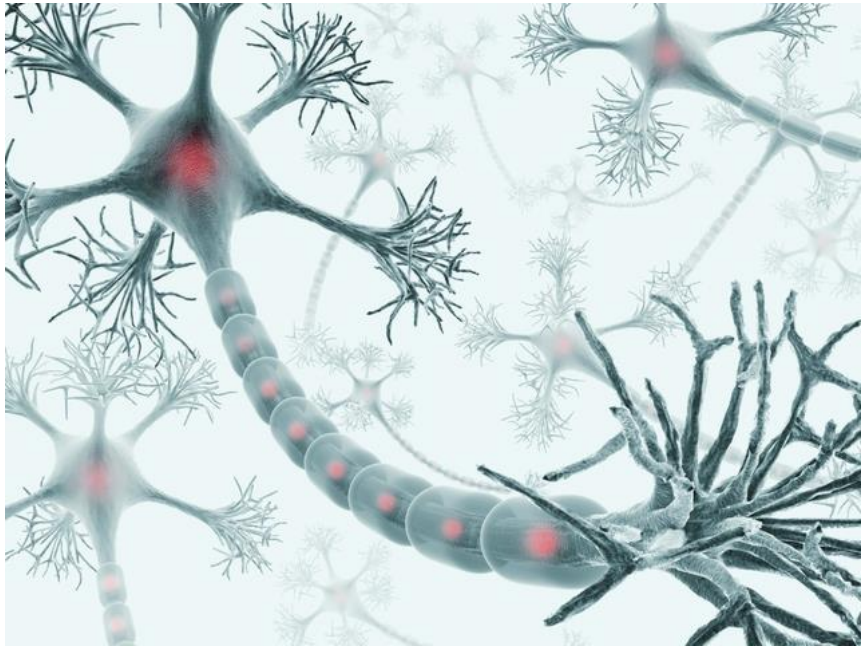
Machine Learning

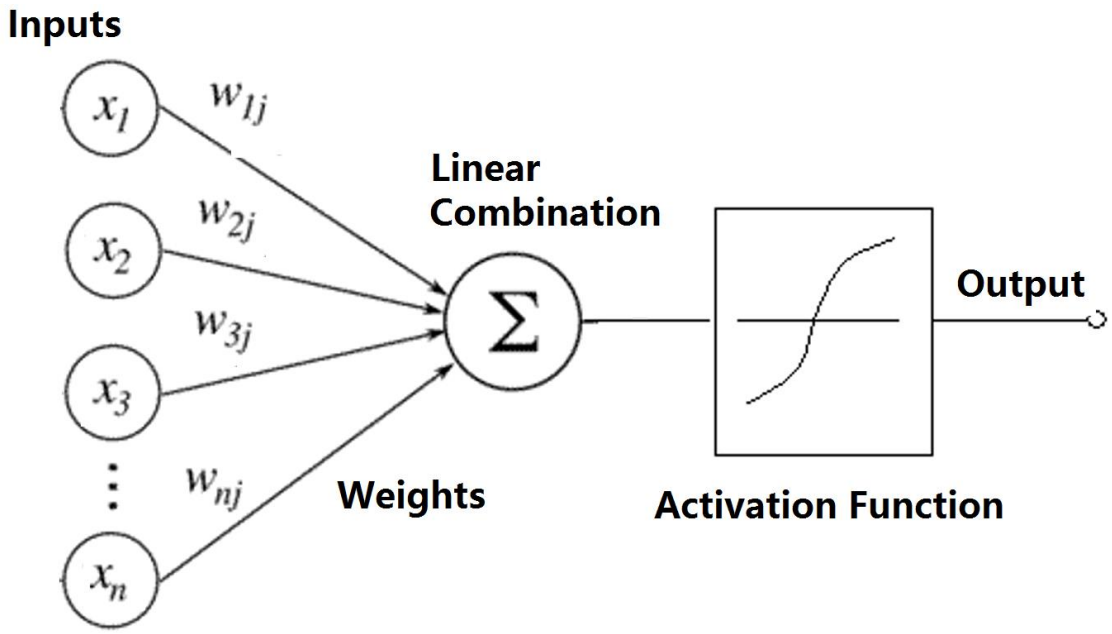
Neural Networks and Deep Learning

Perceptron

The Neuron

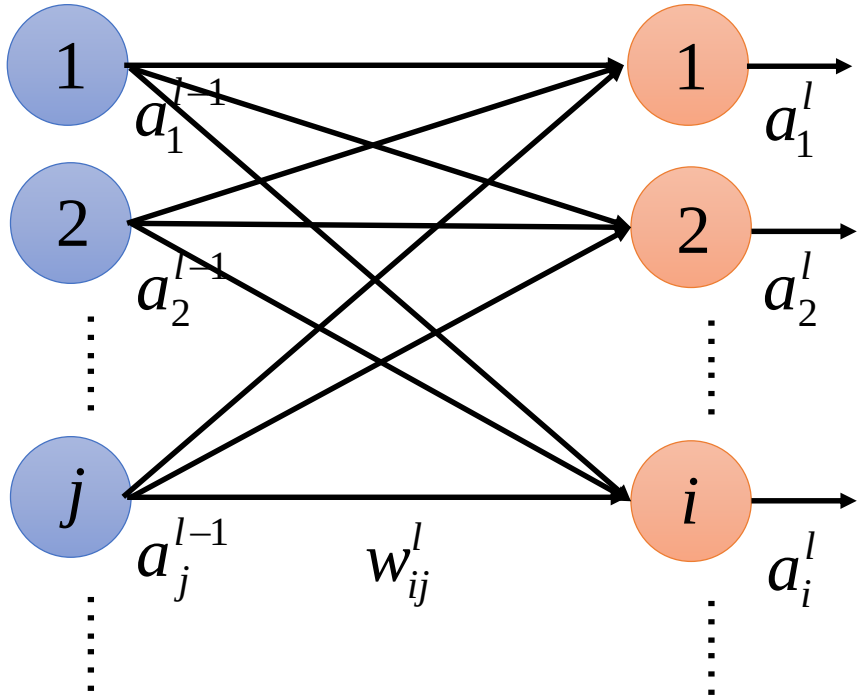
Neural Networks (Artificial Neural Networks, Precisely) are always with story of its biological counterpart, however, with the advancement of A.I., we now seriously believe it is a mathematical model of “imitating”.





Full Connected Networks

Fully Connected Feed-Forward Neural Networks

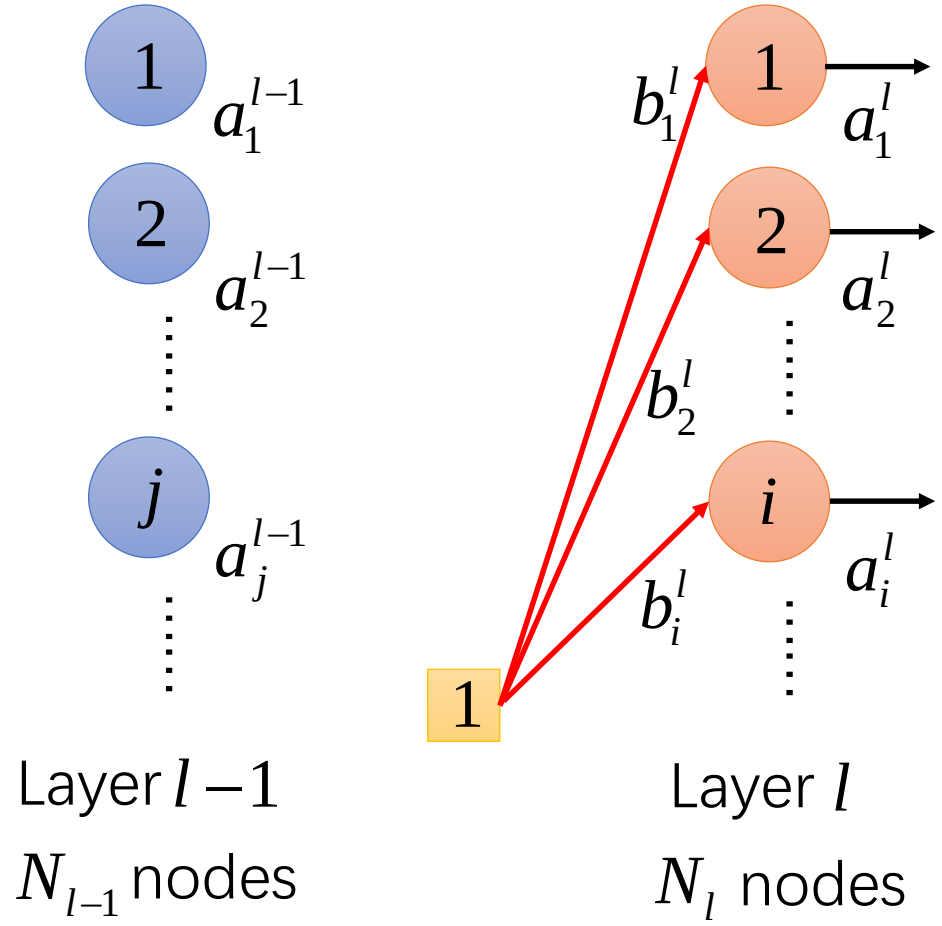


w_{ij}^l → Layer $l-1$ to Layer l

$$W^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \\ & & \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \end{matrix} \begin{matrix} \overbrace{\quad\quad\quad} \\ \\ \\ \\ \end{matrix} \begin{matrix} N_{l-1} \\ \\ \\ \\ \end{matrix}$$

$$\left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} N_l$$

Bias

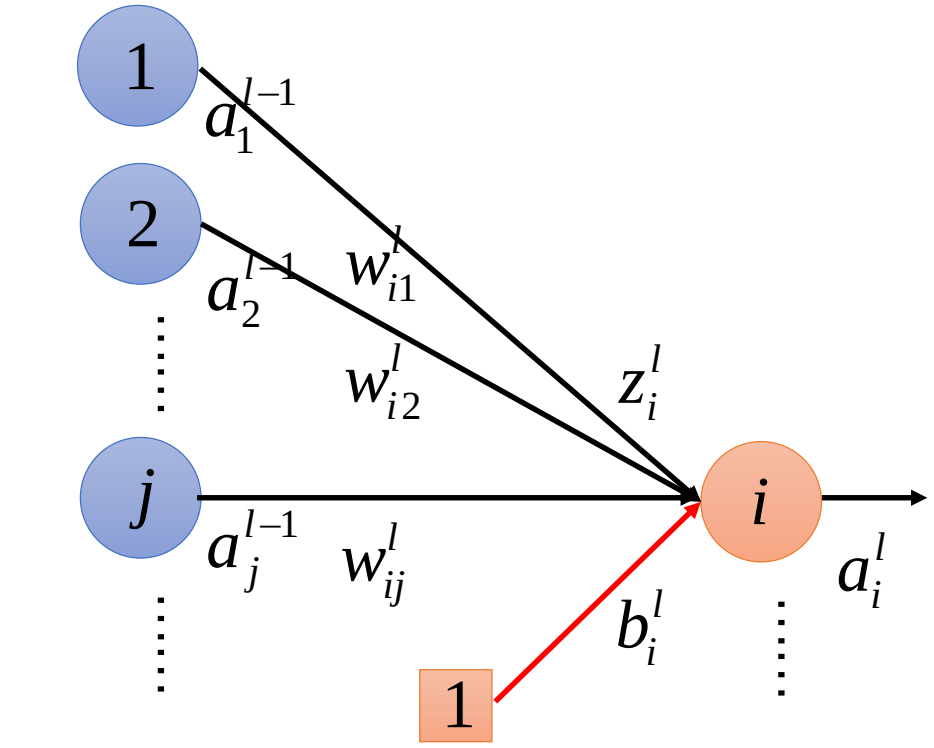


b_i^l : bias for neuron i at layer l

$$b^l = \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

bias for all neurons in layer l

Linear Combination



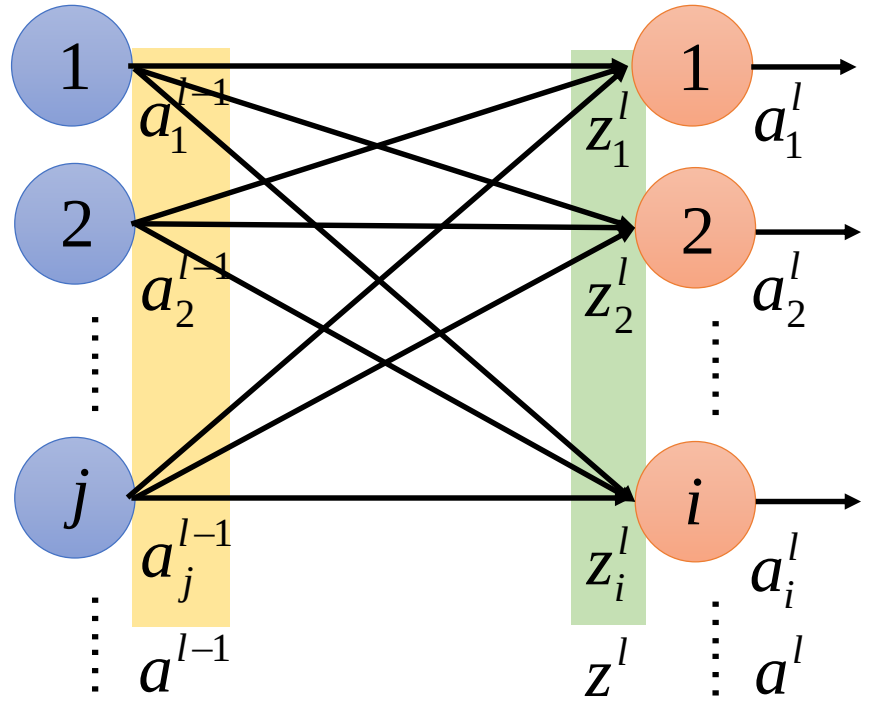
z_i^l : input of the activation function for neuron i at layer l

z^l : input of the activation function all the neurons in layer l

$$z_i^l = w_{i1}^l a_1^{l-1} + w_{i2}^l a_2^{l-1} \ldots + b_i^l$$

$$z_i^l = \sum_{j=1}^{N_{l-1}} w_{ij}^l a_j^{l-1} + b_i^l$$

Inputs-Outputs Relations



Layer $l-1$

N_{l-1} nodes

Layer l

N_l nodes

$$z_1^l = w_{11}^l a_1^{l-1} + w_{12}^l a_2^{l-1} + \cdots + b_1^l$$

$$z_2^l = w_{21}^l a_1^{l-1} + w_{22}^l a_2^{l-1} + \cdots + b_2^l$$

$$\vdots$$

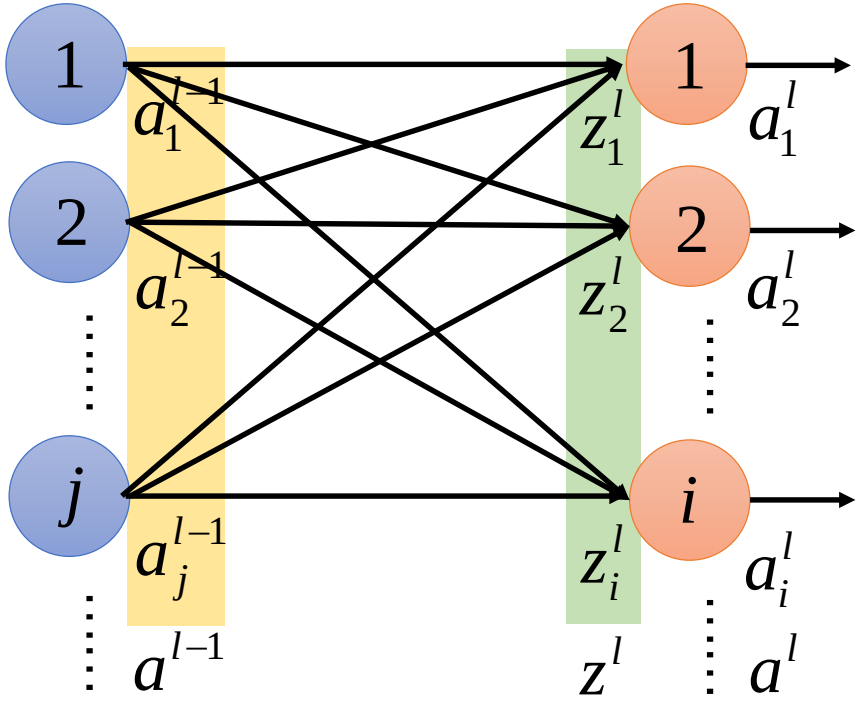
$$z_i^l = w_{i1}^l a_1^{l-1} + w_{i2}^l a_2^{l-1} + \cdots + b_i^l$$

$$\vdots$$

$$\begin{bmatrix} z_1^l \\ z_2^l \\ \vdots \\ z_i^l \\ \vdots \end{bmatrix} = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \end{bmatrix} \begin{bmatrix} a_1^{l-1} \\ a_2^{l-1} \\ \vdots \\ a_i^{l-1} \\ \vdots \end{bmatrix} + \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

$$z^l = W^l a^{l-1} + b^l$$

Inputs-Outputs Relations (Activation)



$$a_i^l = \sigma\left(z_i^l\right)$$

$$\begin{bmatrix} a_1^l \\ a_2^l \\ \vdots \\ a_i^l \\ \vdots \end{bmatrix} = \begin{bmatrix} \sigma\left(z_1^l\right) \\ \sigma\left(z_2^l\right) \\ \vdots \\ \sigma\left(z_i^l\right) \\ \vdots \end{bmatrix}$$

Layer $l-1$

N_{l-1} nodes

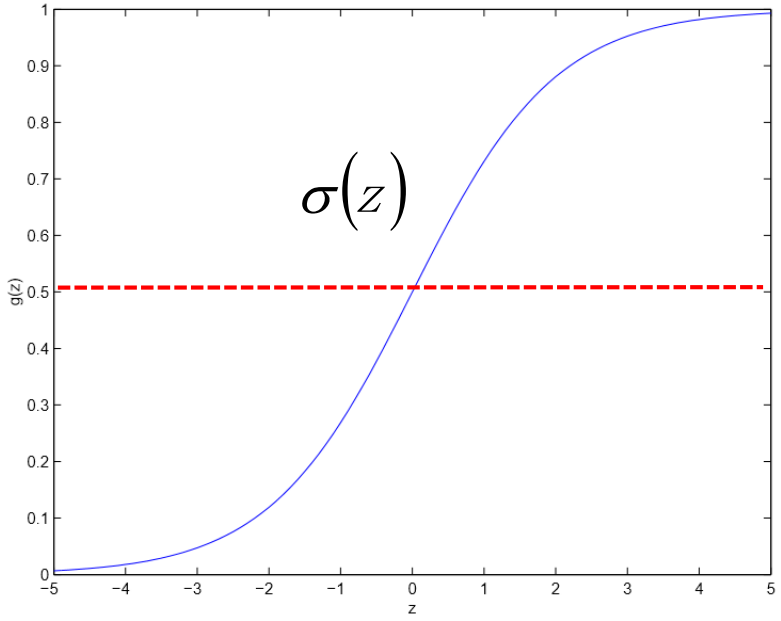
Layer l

N_l nodes

$$a^l = \sigma\left(z^l\right)$$

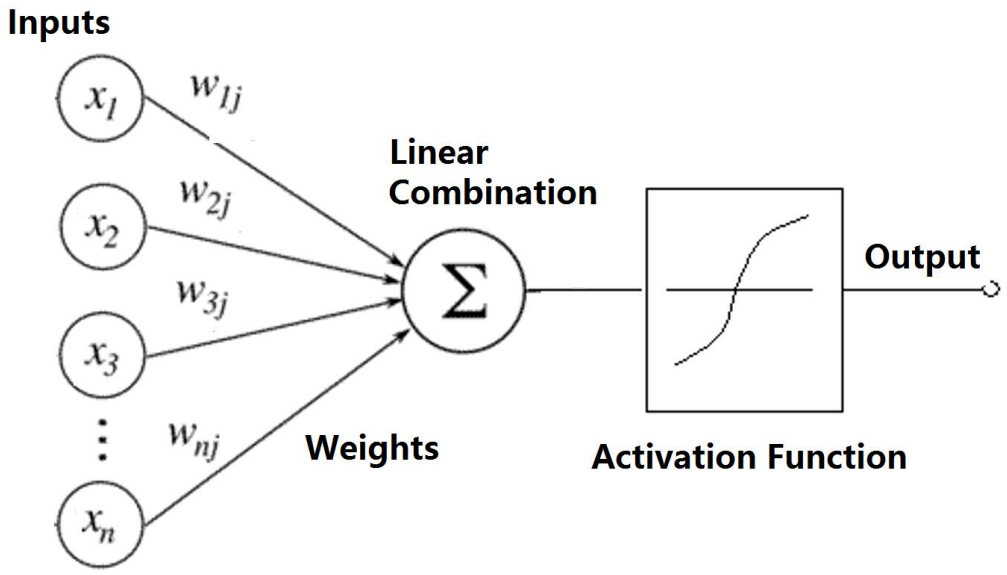
$$a^l = \sigma\left(W^l a^{l-1} + b^l\right)$$

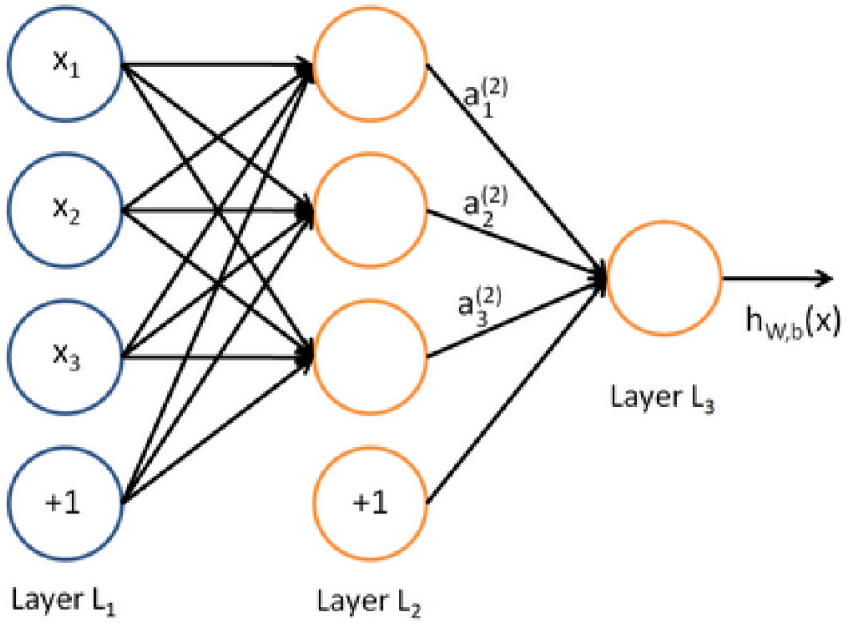
Sigmoid Function



NN and LR

A neural network = running several logistic regressions at the same time.

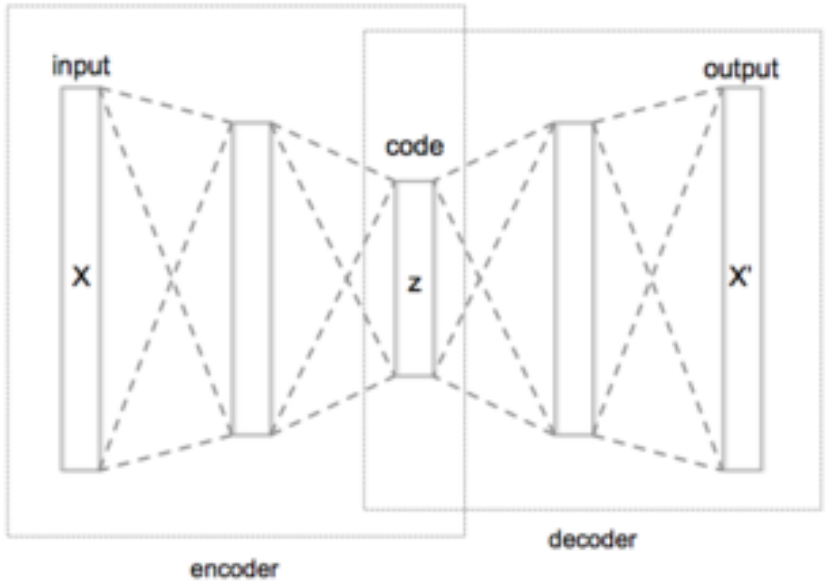


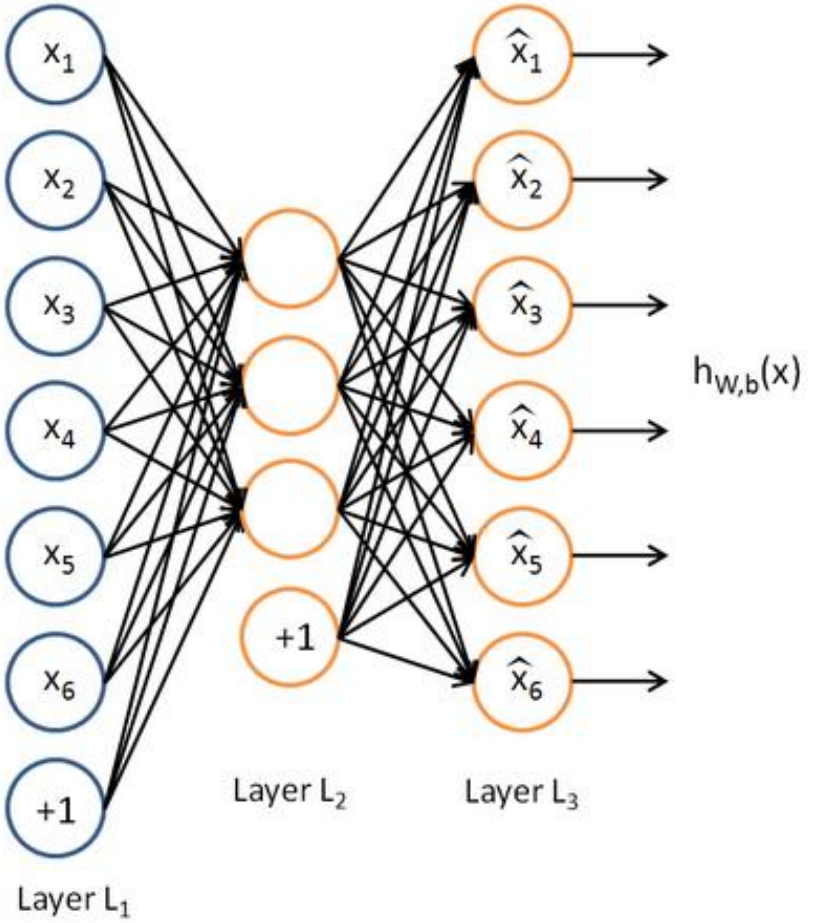


The LRs are then feed into another logistic regression function.

Auto-Encoder

Auto-Encoder is a neural network used to learn efficient coding. Architecturally, the simplest form of an autoencoder is a feedforward, non-recurrent neural network





Back Propagation

A Simple Example of BP

Backpropagation: a simple example

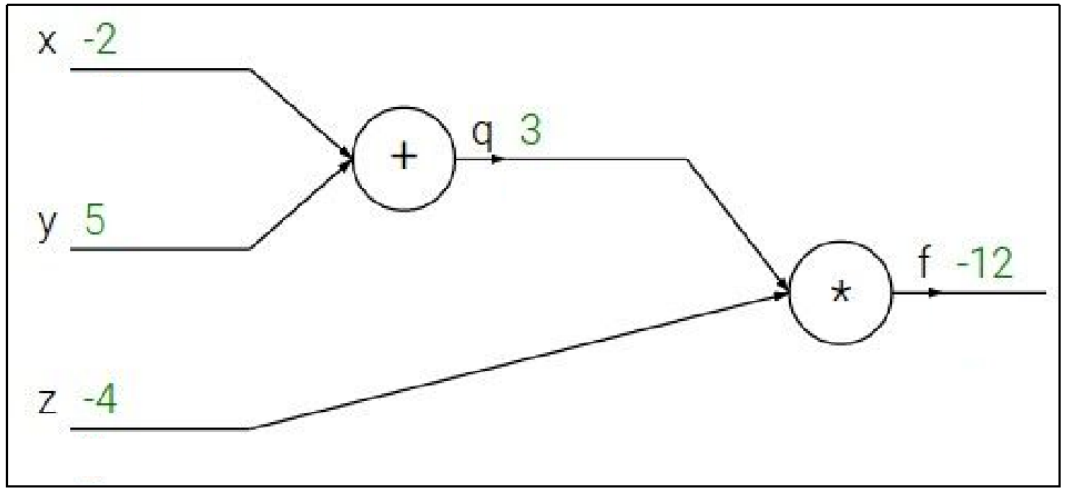
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



A Simple Example of BP

Backpropagation: a simple example

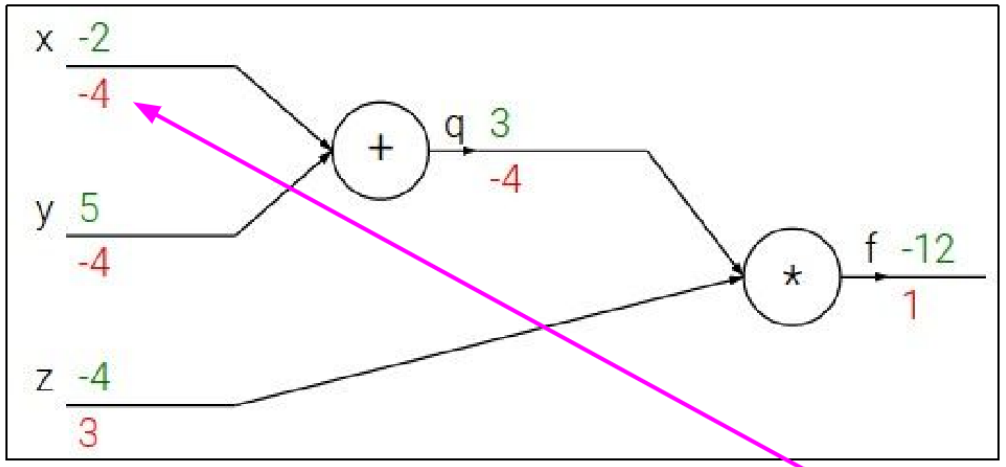
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

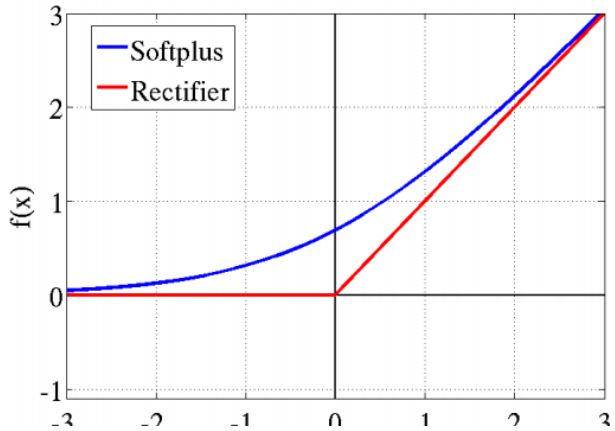
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

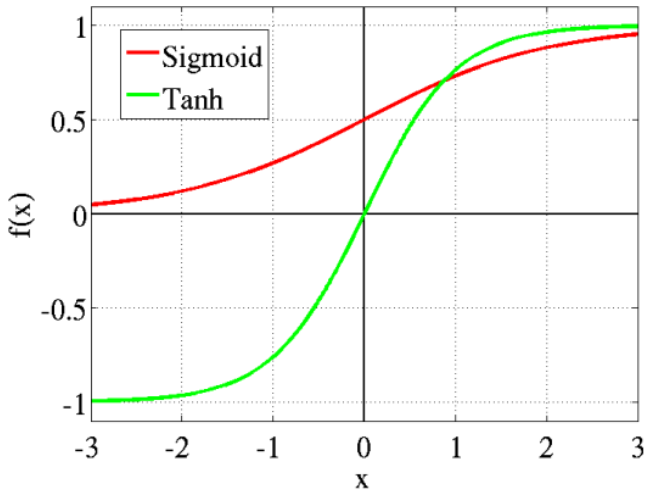


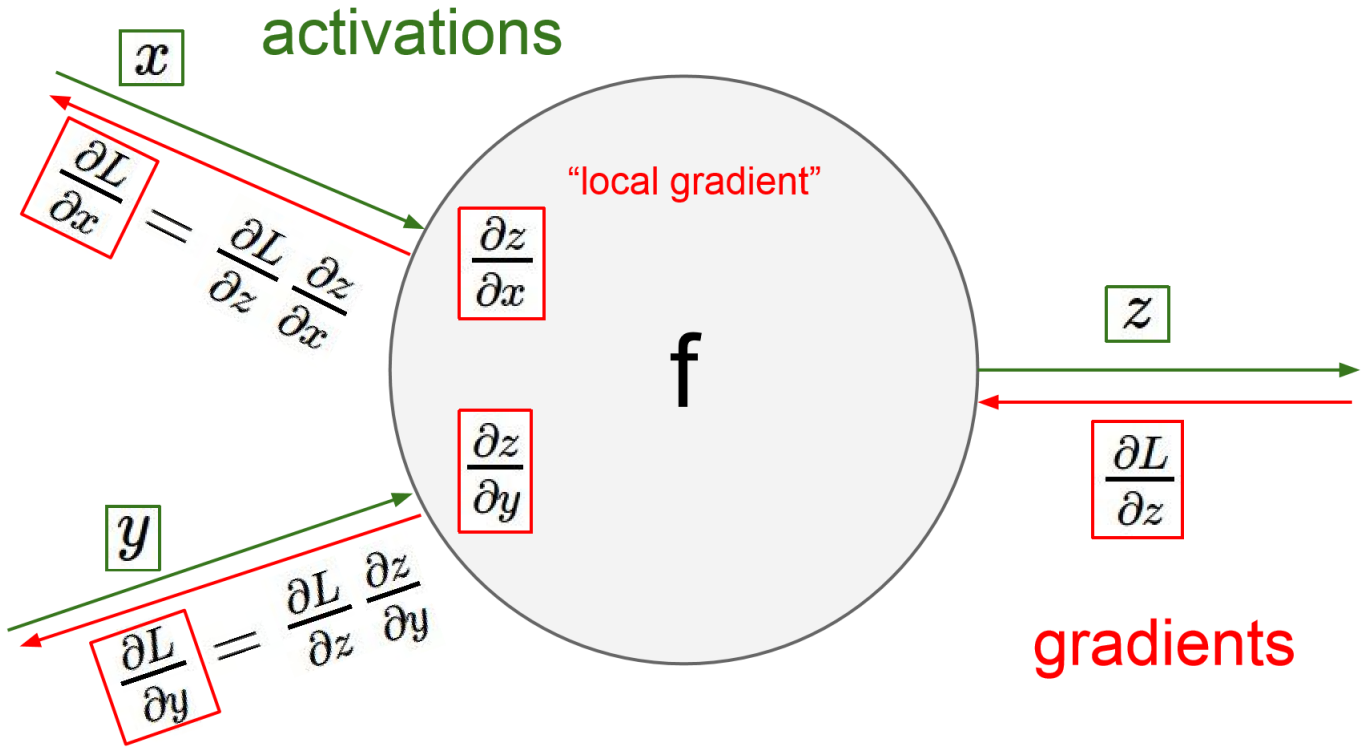
Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Illustration of Back-Propagation



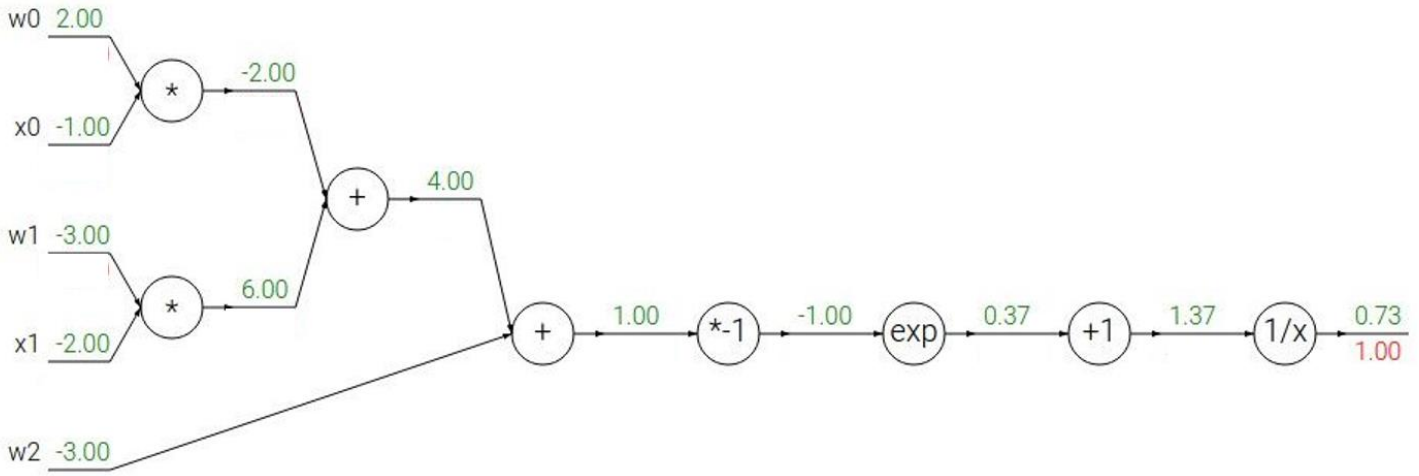




Back Propagation through layers.

A Simple Example of BP

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

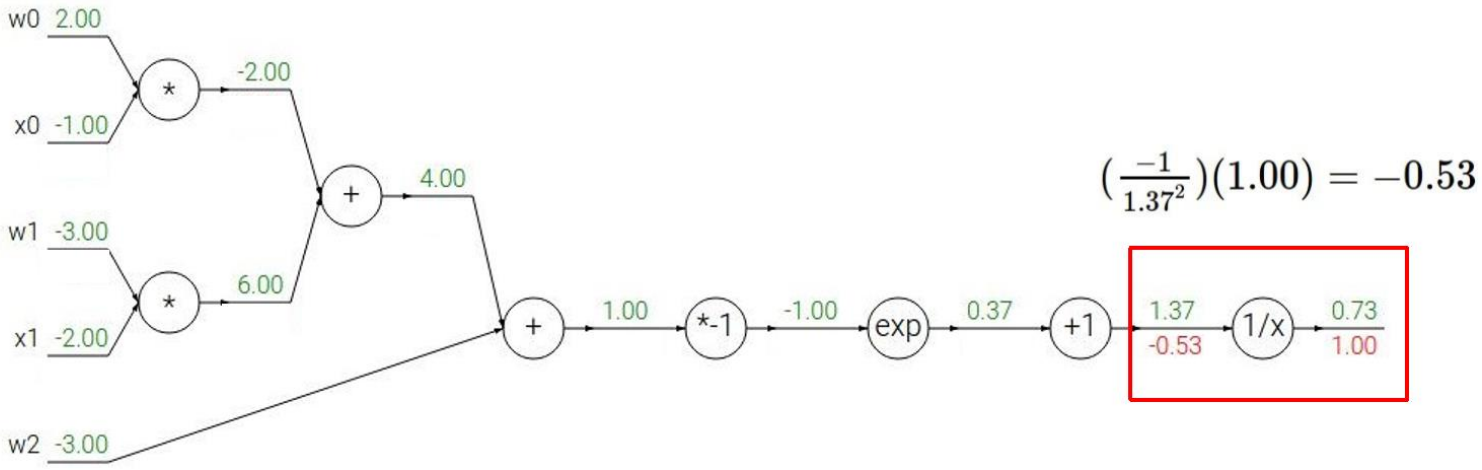
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

A Simple Example of BP

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



$(\frac{-1}{1.37^2})(1.00) = -0.53$

$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

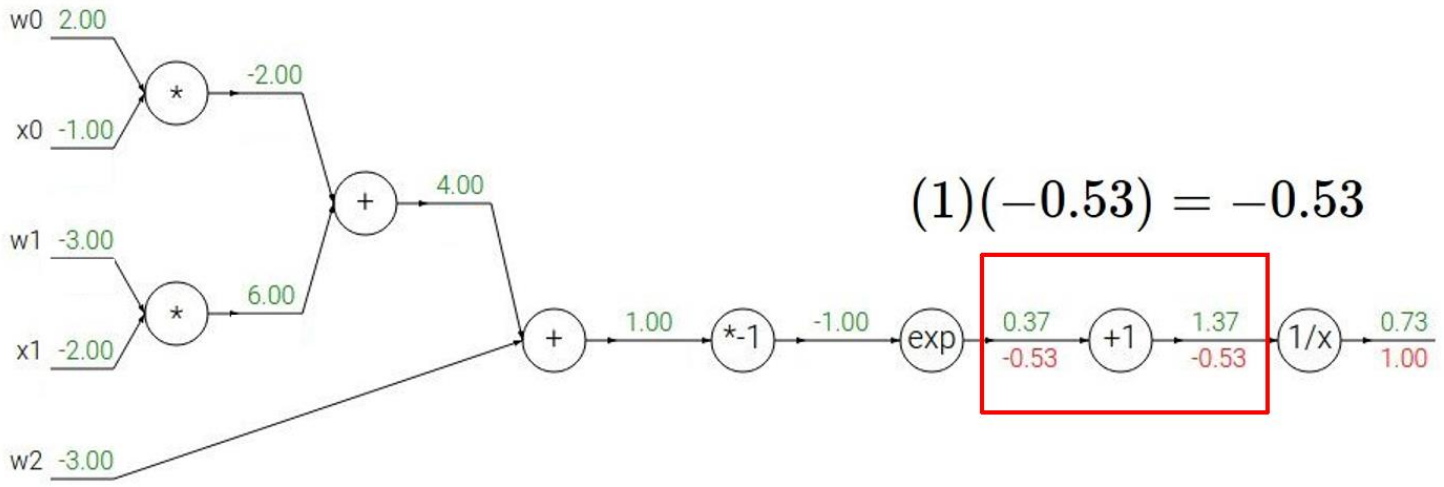
$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

A Simple Example of BP

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$(1)(-0.53) = -0.53$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

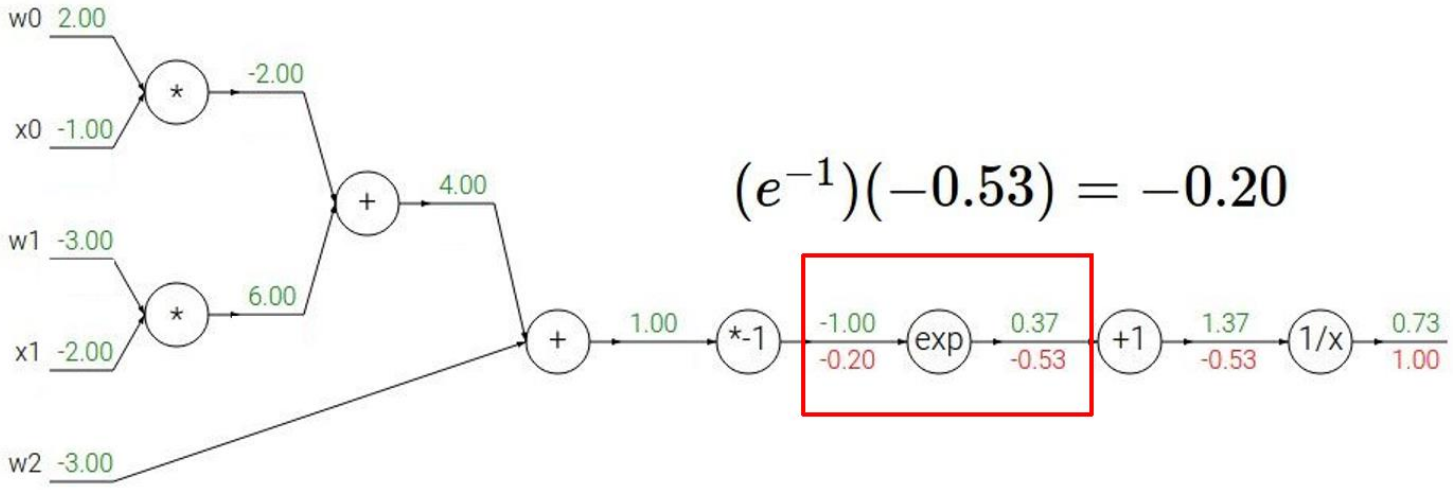
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

A Simple Example of BP

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



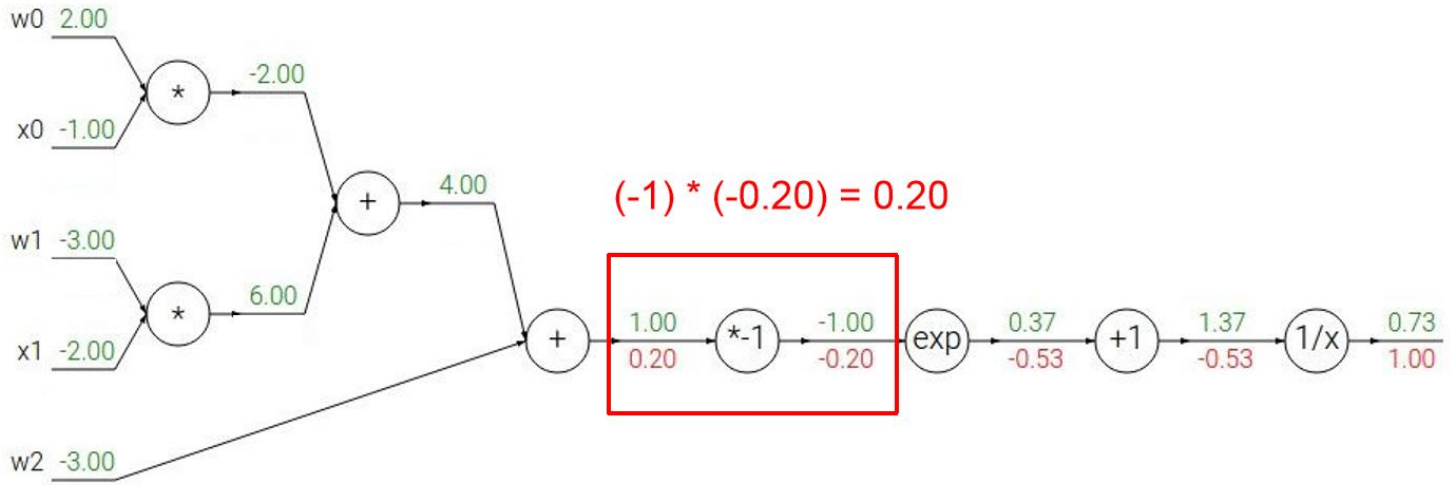
$$(e^{-1})(-0.53) = -0.20$$

$f(x) = e^x$	$\rightarrow$	$\frac{df}{dx} = e^x$
$f_a(x) = ax$	$\rightarrow$	$\frac{df}{dx} = a$

$f(x) = \frac{1}{x}$	$\rightarrow$	$\frac{df}{dx} = -1/x^2$
$f_c(x) = c + x$	$\rightarrow$	$\frac{df}{dx} = 1$

A Simple Example of BP

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

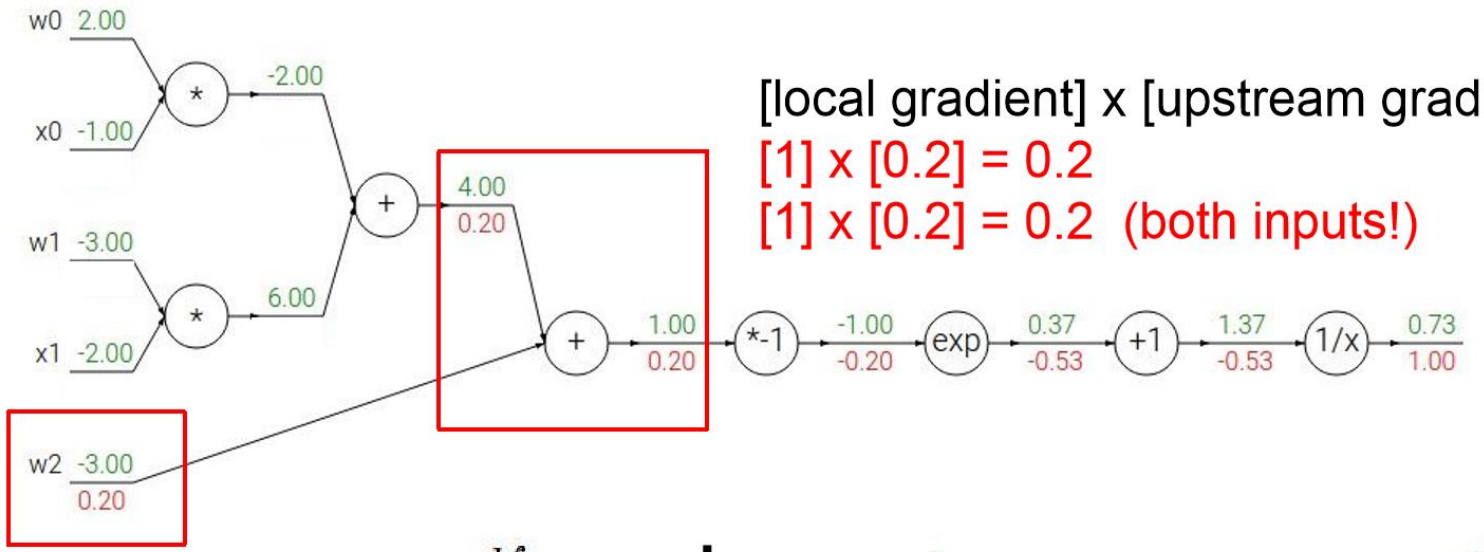
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

A Simple Example of BP

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[local gradient] x [upstream gradient]

[1] x [0.2] = 0.2

[1] x [0.2] = 0.2 (both inputs!)

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

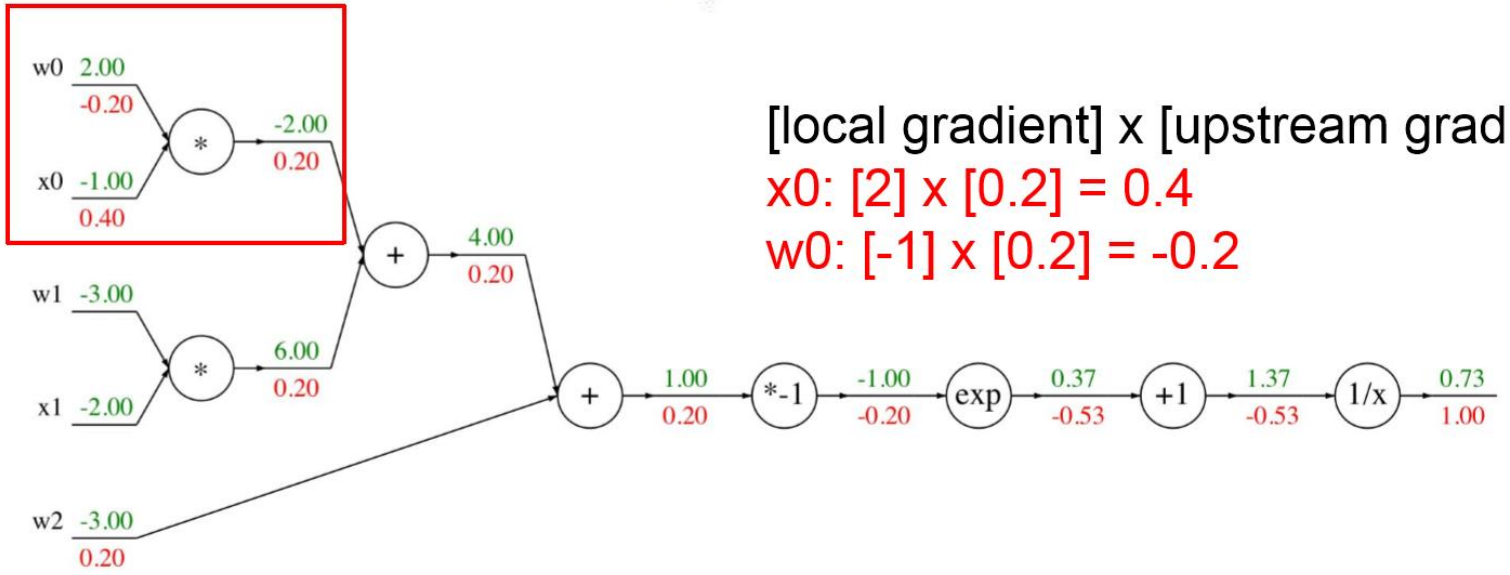
$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

A Simple Example of BP

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[local gradient] x [upstream gradient]

x0: [2] x [0.2] = 0.4

w0: [-1] x [0.2] = -0.2

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

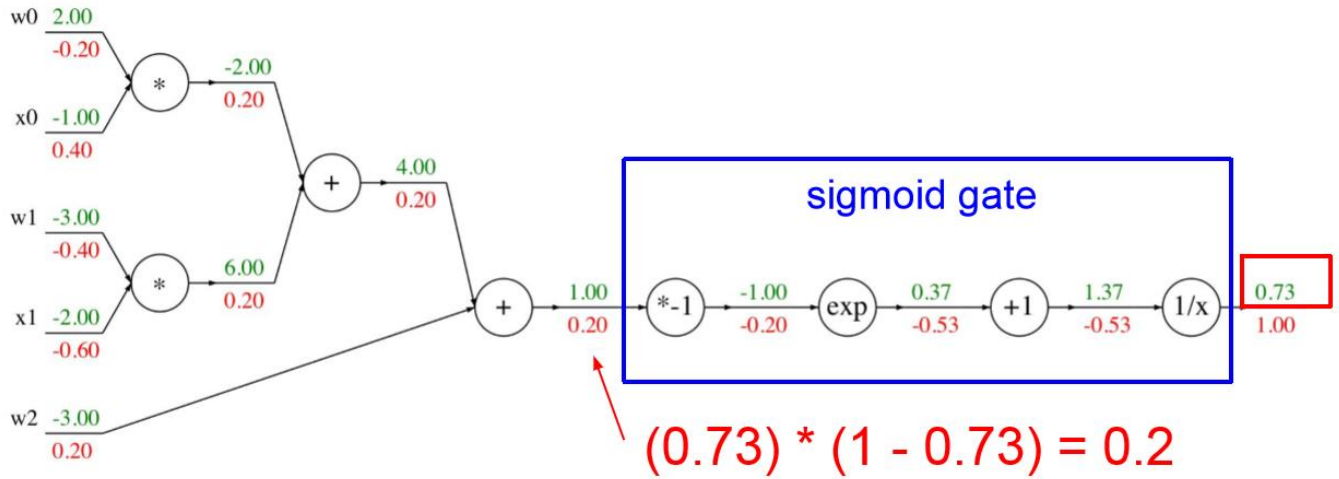
$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

A Simple Example of BP

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

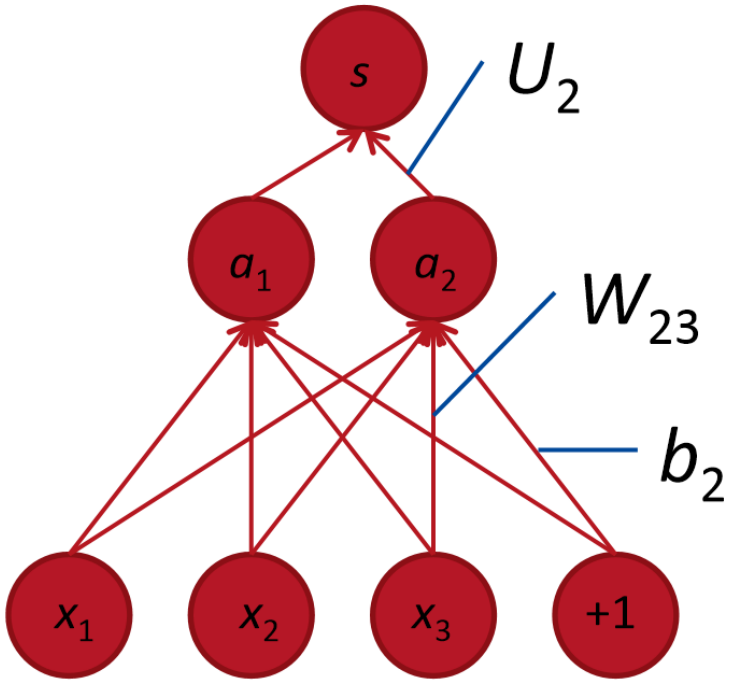
$$\sigma(x) = \frac{1}{1+e^{-x}}$$ sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$



Back-Propagation

W_{23} is only used to compute a_2, the error is back-propagated through a_2.



$$a_i = f(z_i) \quad z_i = W_i.x + b_i = \sum_{j=1}^{3} W_{ij}x_j + b_i$$

$$\frac{\partial s}{\partial W} = \frac{\partial}{\partial W}U^T a = \frac{\partial}{\partial W}U^T f(z) = \frac{\partial}{\partial W}U^T f(Wx+b)$$

$$\frac{\partial}{\partial W_{ij}}U^T a \rightarrow \frac{\partial}{\partial W_{ij}}U_i a_i$$

$$U_i\frac{\partial}{\partial W_{ij}}a_i = U_i\frac{\partial a_i}{\partial z_i}\frac{\partial z_i}{\partial W_{ij}} = U_i\frac{\partial f(z_i)}{\partial z_i}\frac{\partial z_i}{\partial W_{ij}}$$

$$= U_i f'(z_i)\frac{\partial z_i}{\partial W_{ij}} = U_i f'(z_i)\frac{\partial W_i.x + b_i}{\partial W_{ij}}$$

$$= U_i f'(z_i)\frac{\partial}{\partial W_{ij}}\sum_k W_{ik}x_k = \underbrace{U_i f'(z_i)}_{\delta_i} x_j$$

Local error signal — Local input signal

Convolutional Neural Network

Visual Receptive Field

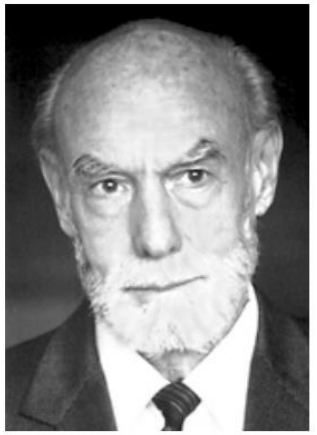
to David H. Hubel and Torsten N. Wiesel "*for their discoveries concerning information processing in the visual system*".

Cells that are sensitive to small sub-regions of the visual field, called a receptive field

Simple cells respond maximally to specific edge-like patterns within their receptive field.

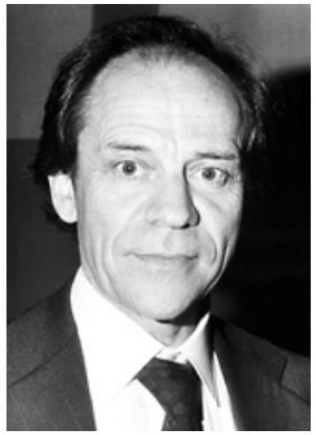
Complex cells have larger receptive fields and are locally invariant to the exact position of the pattern.

The Nobel Prize in Physiology or Medicine 1981



Roger W. Sperry
Prize share: 1/2

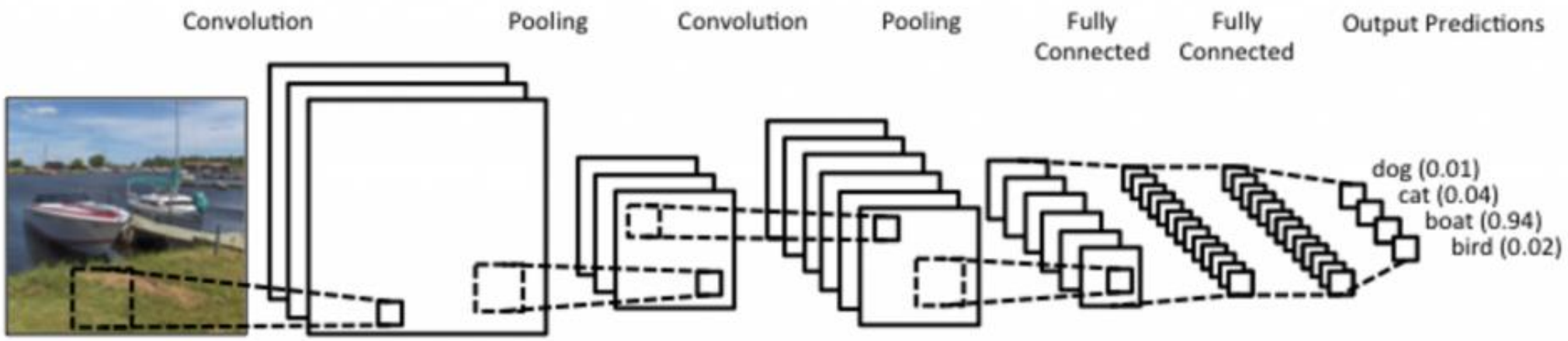
David H. Hubel
Prize share: 1/4



Torsten N. Wiesel
Prize share: 1/4

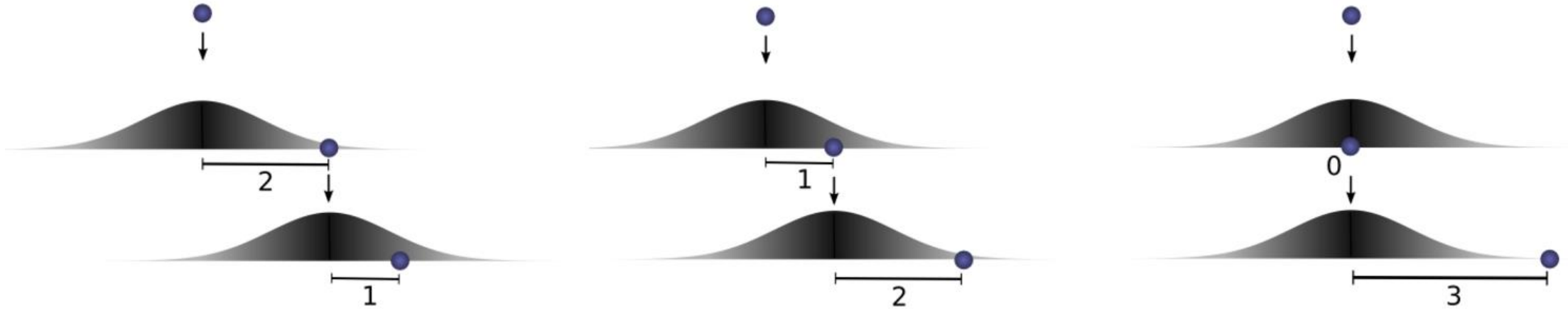
Convolutional Neural Networks

In Image Classification a CNN may learn to detect edges from raw pixels in the first layer, then use the edges to detect simple shapes in the second layer, and then use these shapes to deter higher-level features, such as facial shapes in higher layers. The last layer is then a classifier that uses these high-level features.



I admire the elegance of your method of computation; it must be nice to ride through these fields upon the horse of true mathematics while the like of us have to make our way laboriously on foot. — Albert Einstein

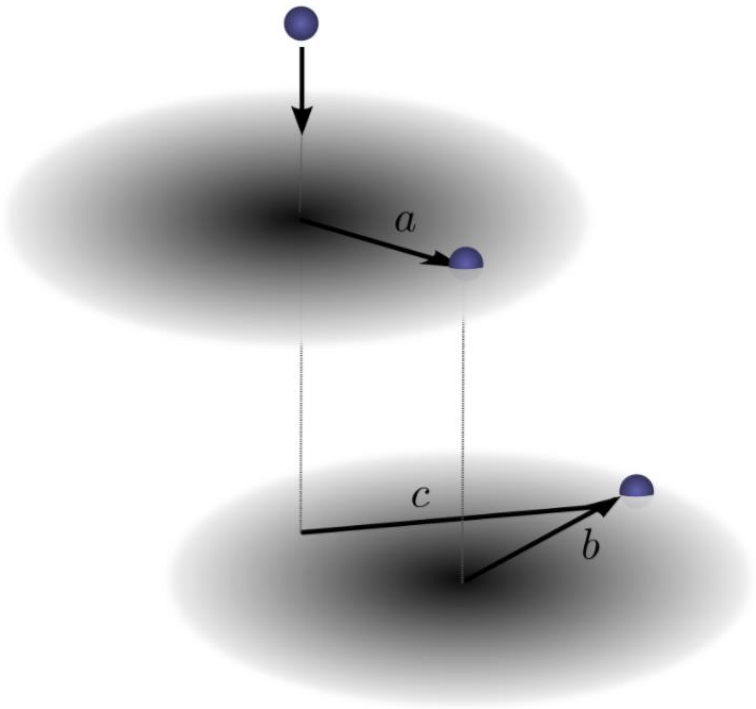
Definition of Convolution



$$\ldots \quad f(0)\cdot g(3) \;+\; f(1)\cdot g(2) \;+\; f(2)\cdot g(1) \quad \ldots$$

$$\sum_{a+b=c} f(a)\cdot g(b) \qquad \Rightarrow \qquad (f*g)(c) = \sum_{a+b=c} f(a)\cdot g(b) = \sum_{a} f(a)\cdot g(c-a)$$

Definition of Convolution



$f(a)$

$g(b)$

$$(f * g)(c) = \sum_{a+b=c} f(a) \cdot g(b)$$

$$(f * g)(c_1, c_2) = \sum_{\substack{a_1+b_1=c_1 \\ a_2+b_2=c_2}} f(a_1, a_2) \cdot g(b_1, b_2)$$

$$= \sum_{a_1,a_2} f(a_1, a_2) \cdot g(c_1 - a_1,\ c_2 - a_2)$$

For example, convolution is a commutative operation. That is, $f * g = g * f$.

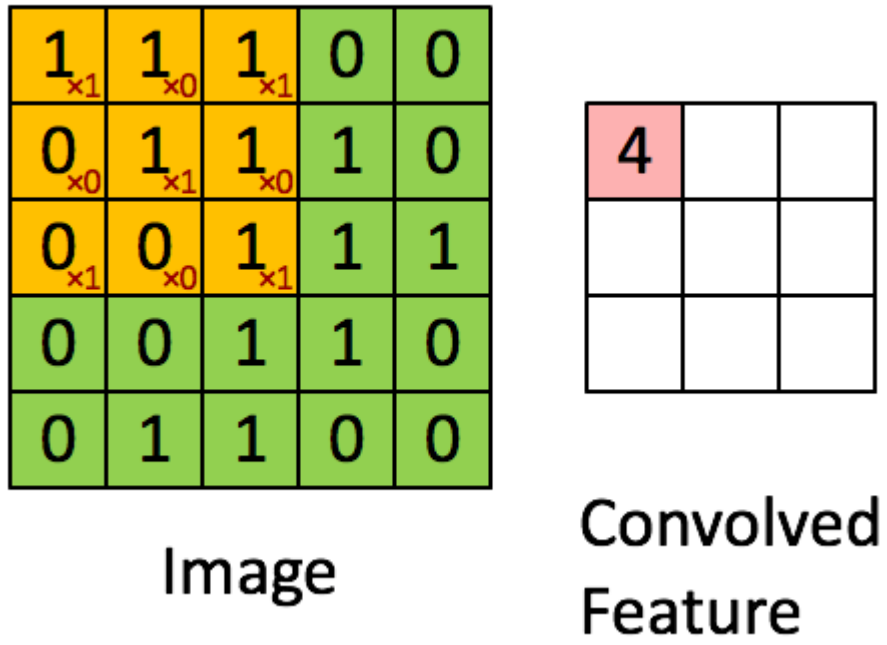
$$\sum_{a+b=c} f(a) \cdot g(b) \quad = \quad \sum_{b+a=c} g(b) \cdot f(a)$$

Convolution is also associative. That is, $(f * g) * h = f * (g * h)$. Why?

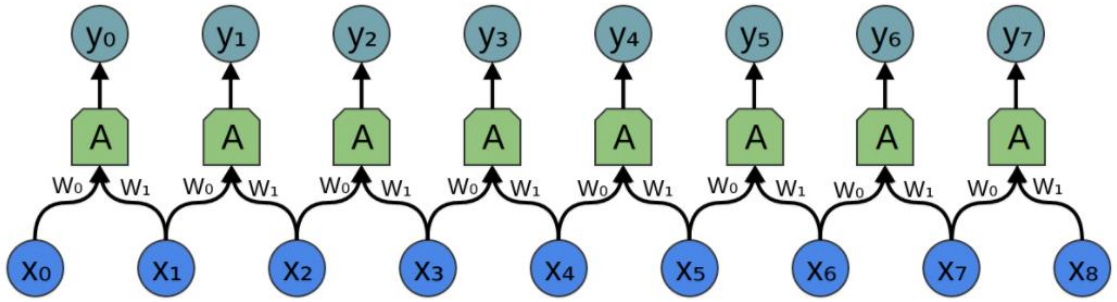
$$\sum_{(a+b)+c=d} (f(a) \cdot g(b)) \cdot h(c) \quad = \quad \sum_{a+(b+c)=d} f(a) \cdot (g(b) \cdot h(c))$$

Convolution with a Kernel

Imagine that the matrix on the left represents an black and white image. Each entry corresponds to one pixel, 0 for black and 1 for white (typically it's between 0 and 255 for grayscale images). The sliding window is called a **kernel**, **filter**, or feature detector. Here we use a 3×3 filter, multiply its values element-wise with the original matrix, then sum them up. To get the full convolution we do this for each element by sliding the filter over the whole matrix.



Partial Connections



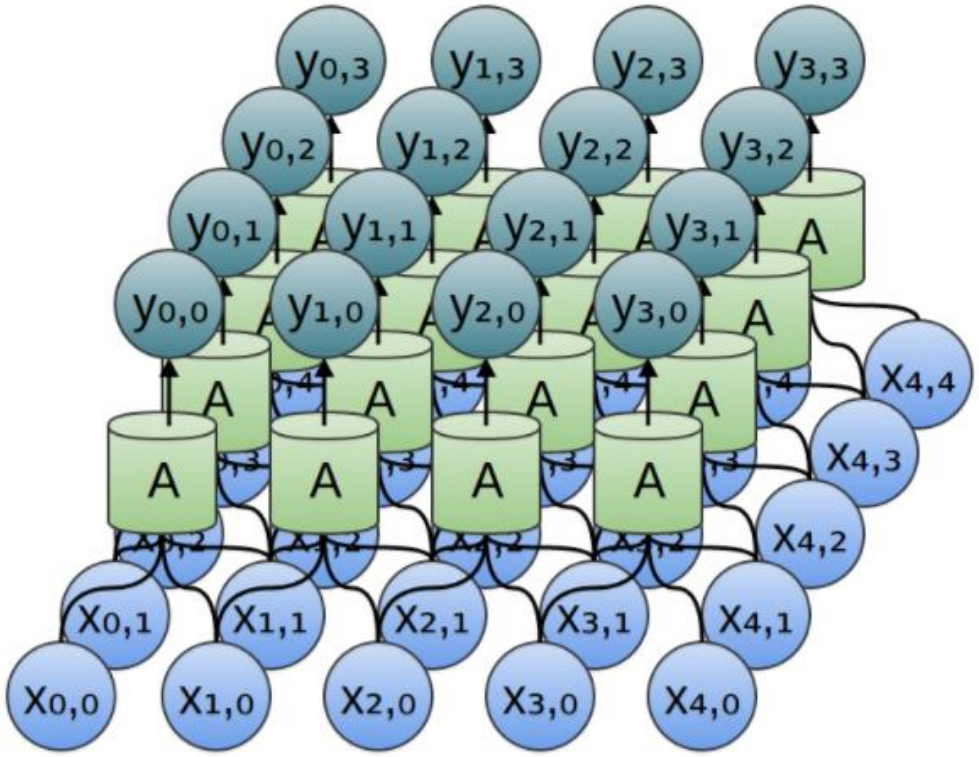
$$y_0 = \sigma(W_{0,0}x_0 + W_{0,1}x_1 + W_{0,2}x_2 \dots)$$

$$y_1 = \sigma(W_{1,0}x_0 + W_{1,1}x_1 + W_{1,2}x_2 \dots)$$

$$W = \begin{bmatrix} W_{0,0} & W_{0,1} & W_{0,2} & W_{0,3} & \dots \\ W_{1,0} & W_{1,1} & W_{1,2} & W_{1,3} & \dots \\ W_{2,0} & W_{2,1} & W_{2,2} & W_{2,3} & \dots \\ W_{3,0} & W_{3,1} & W_{3,2} & W_{3,3} & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

$$W = \begin{bmatrix} w_0 & w_1 & 0 & 0 & \dots \\ 0 & w_0 & w_1 & 0 & \dots \\ 0 & 0 & w_0 & w_1 & \dots \\ 0 & 0 & 0 & w_0 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

Shared Weights

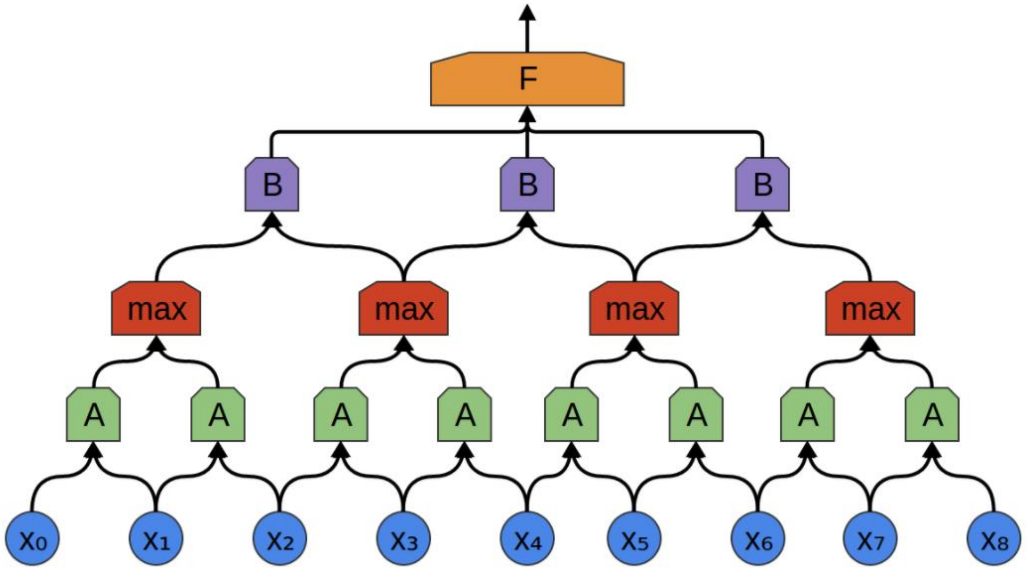


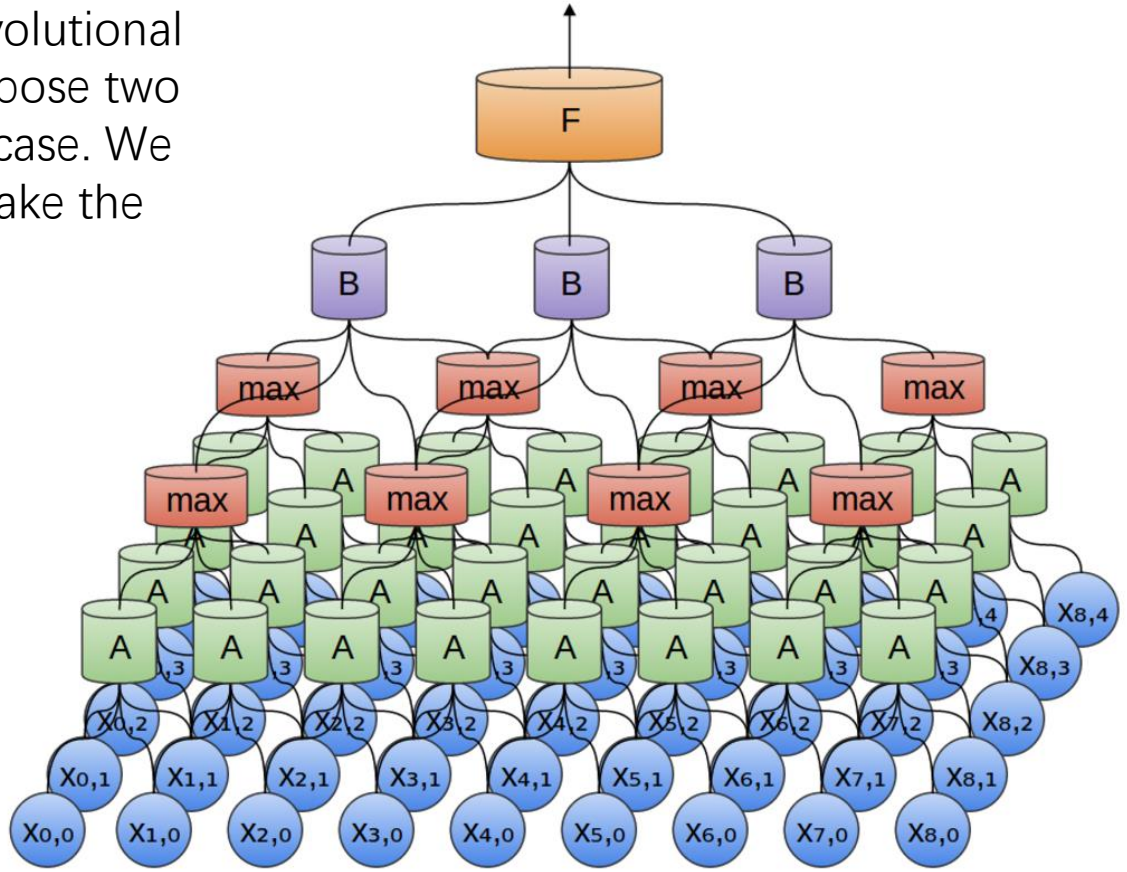
The wiring of a two dimensional convolutional layer corresponds to a two-dimensional convolution.
Consider our example of using a convolution to detect edges in an image, above, by sliding a kernel around and applying it to every patch.

0	0	0	0	0
0	0	0	0	0
0	-1	1	0	0
0	0	0	0	0
0	0	0	0	0

Convolutional Neural Networks

In the previous example, we fed the output of our convolutional layer into a fully-connected layer. But we can also compose two convolutional layers, as we did in the one dimensional case. We can also do max pooling in two dimensions. Here, we take the maximum of features over a small patch.





A Full CNN

Next: Convolutional Neural Networks

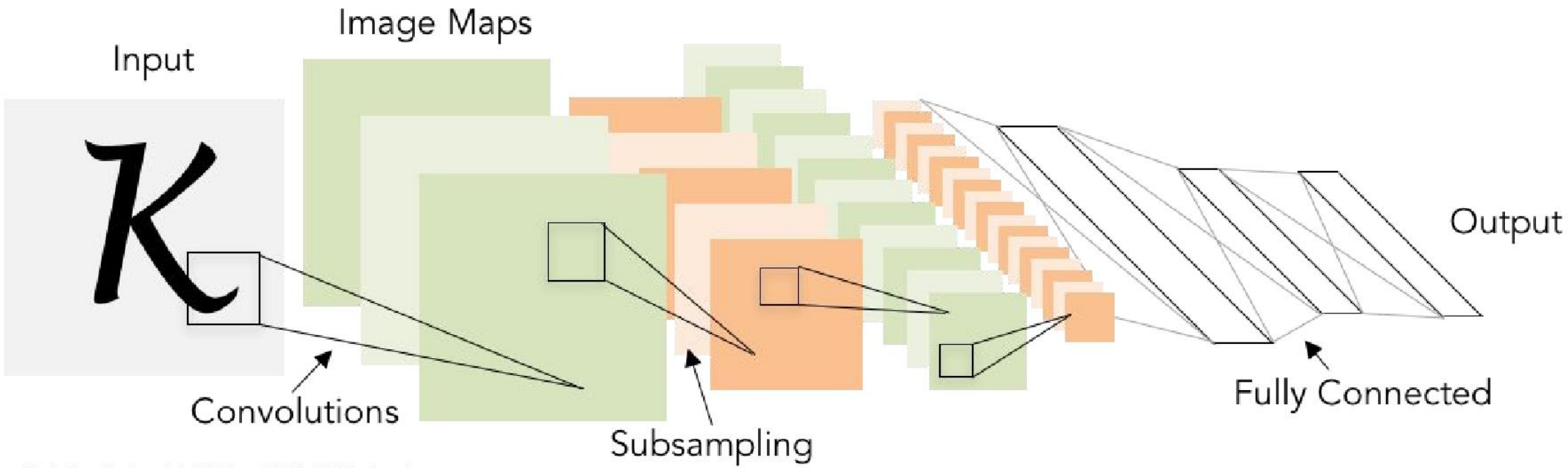
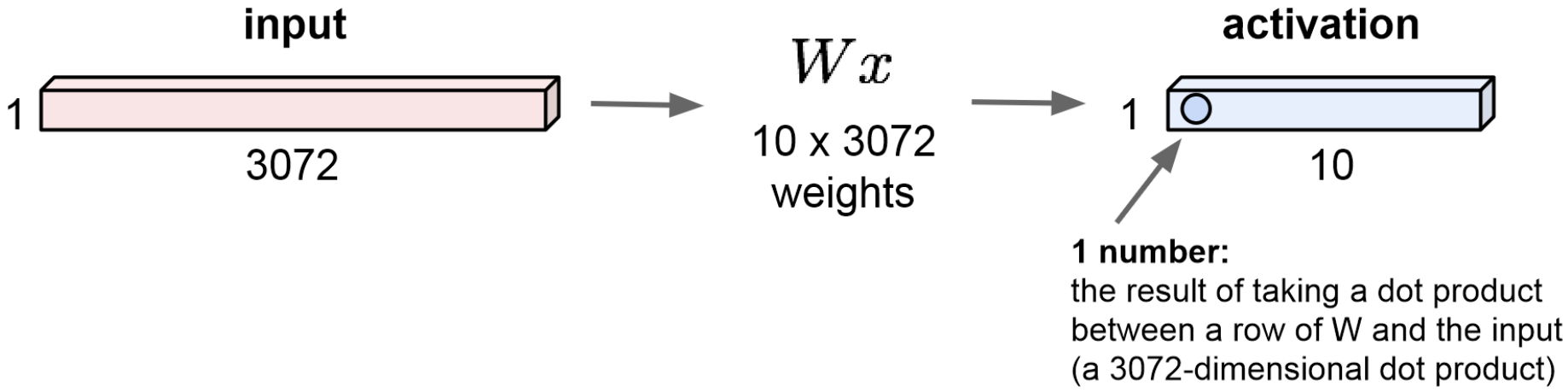


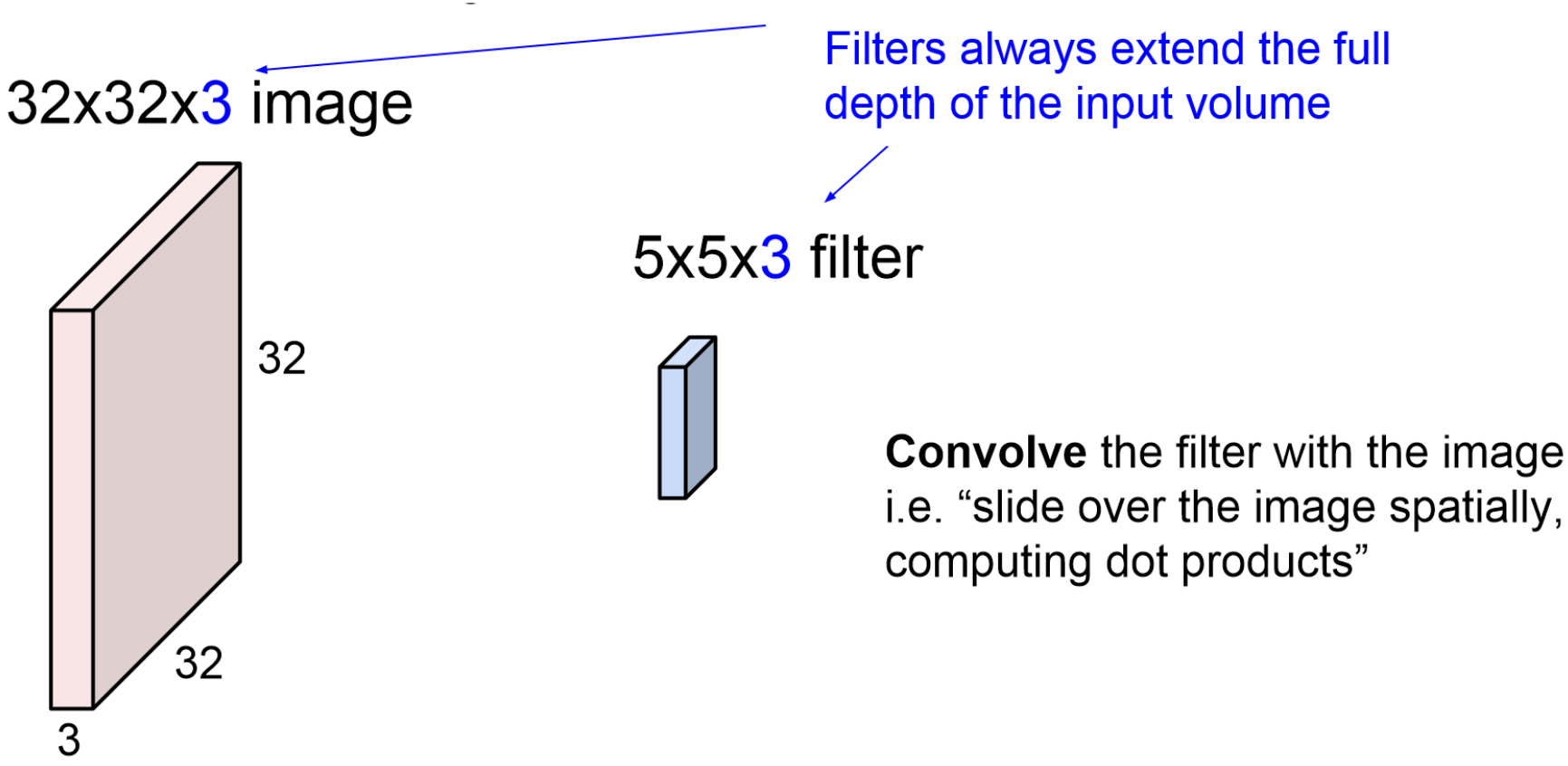
Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

From Zero to One

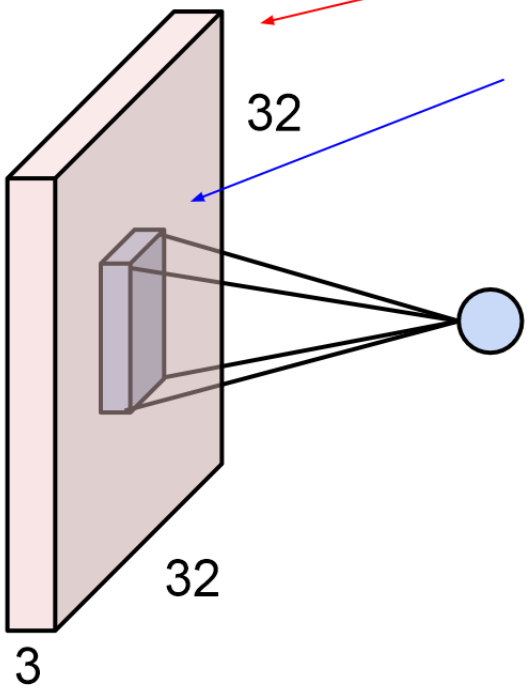
32x32x3 image -> stretch to 3072 x 1



Convolutional Neural Networks



Convolutional Neural Networks



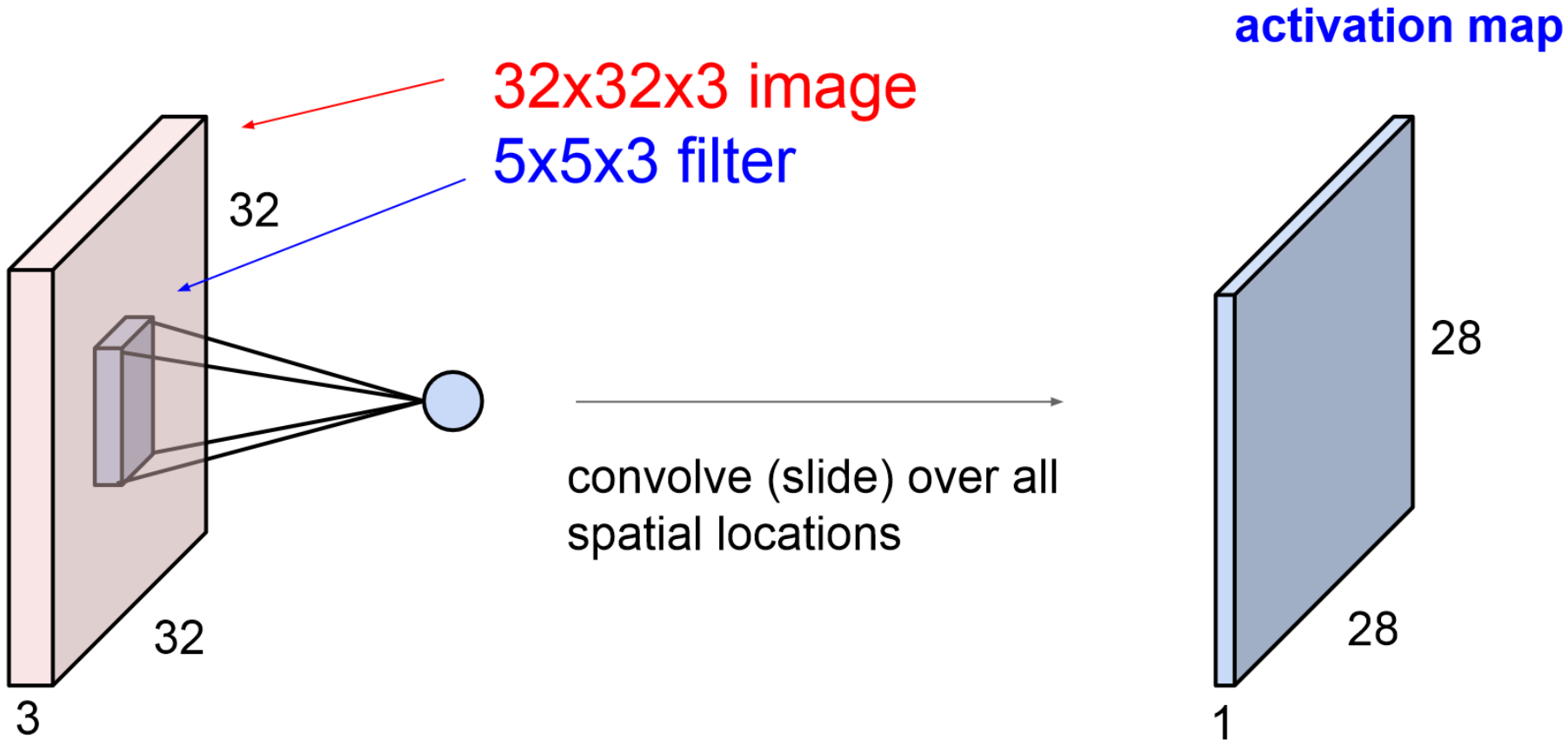
32x32x3 image

5x5x3 filter w

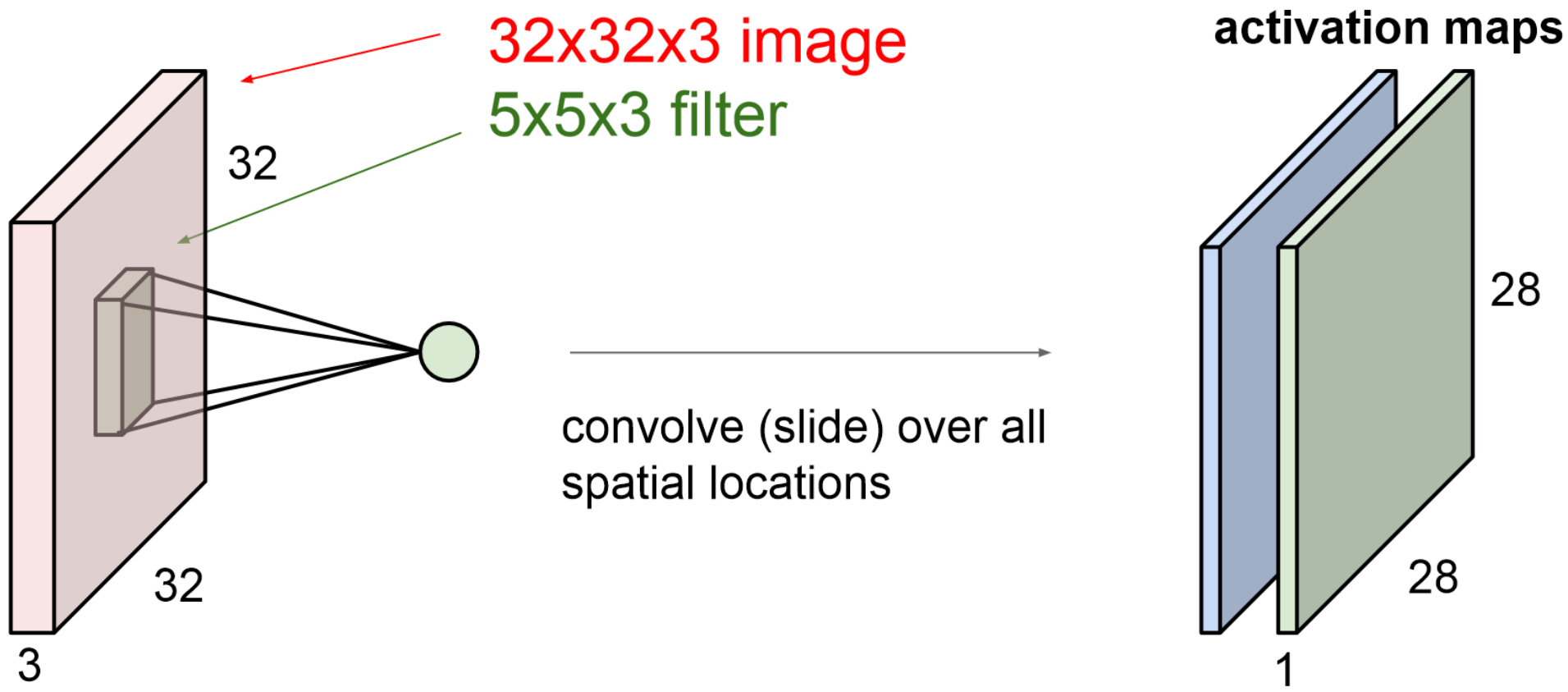
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolutional Neural Networks

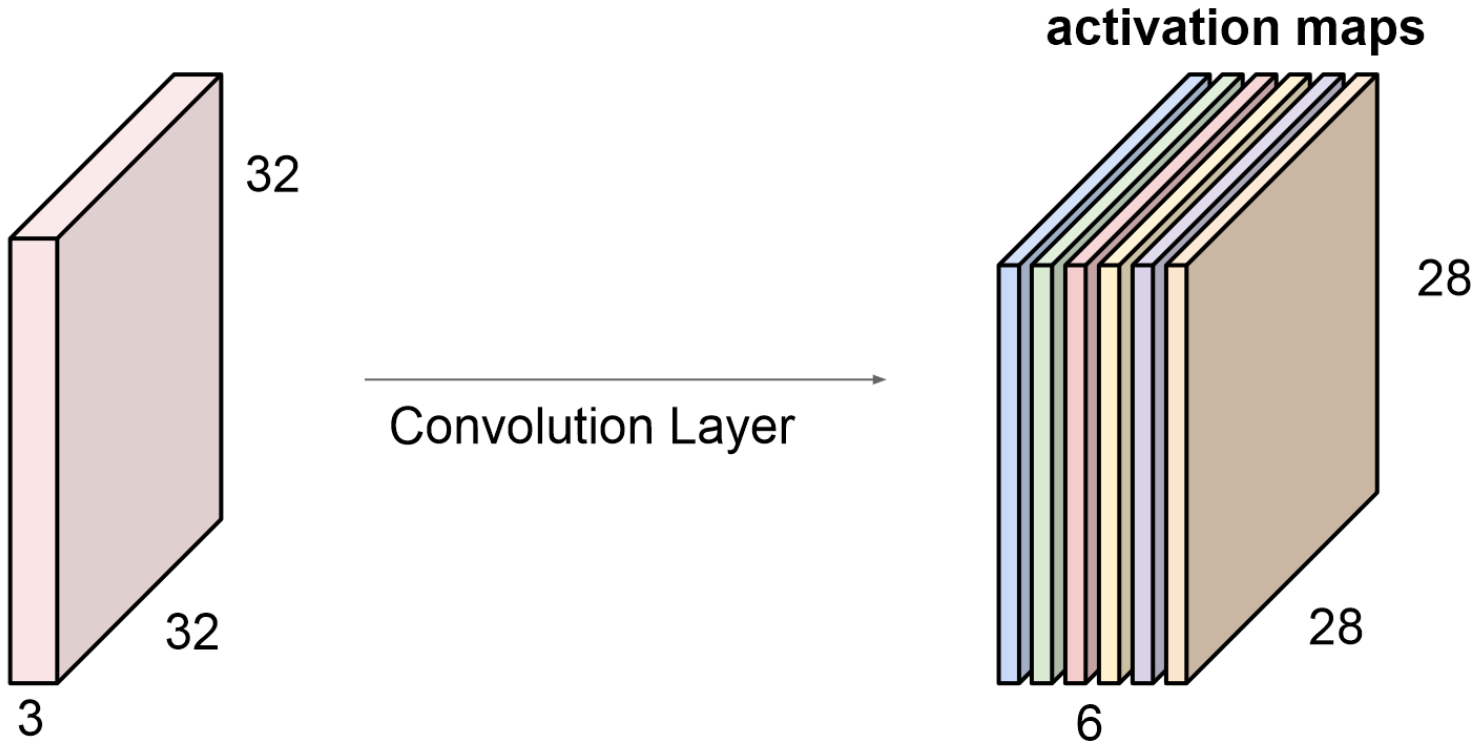


Convolutional Neural Networks



Convolutional Neural Networks

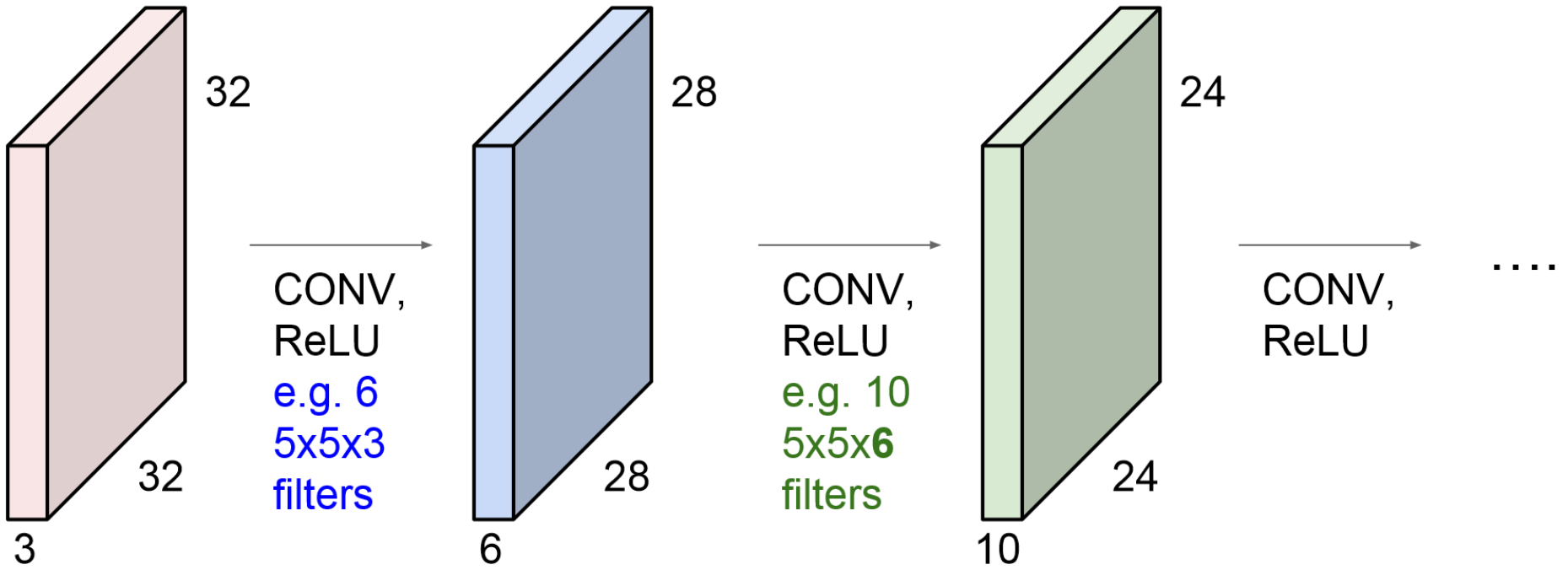
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



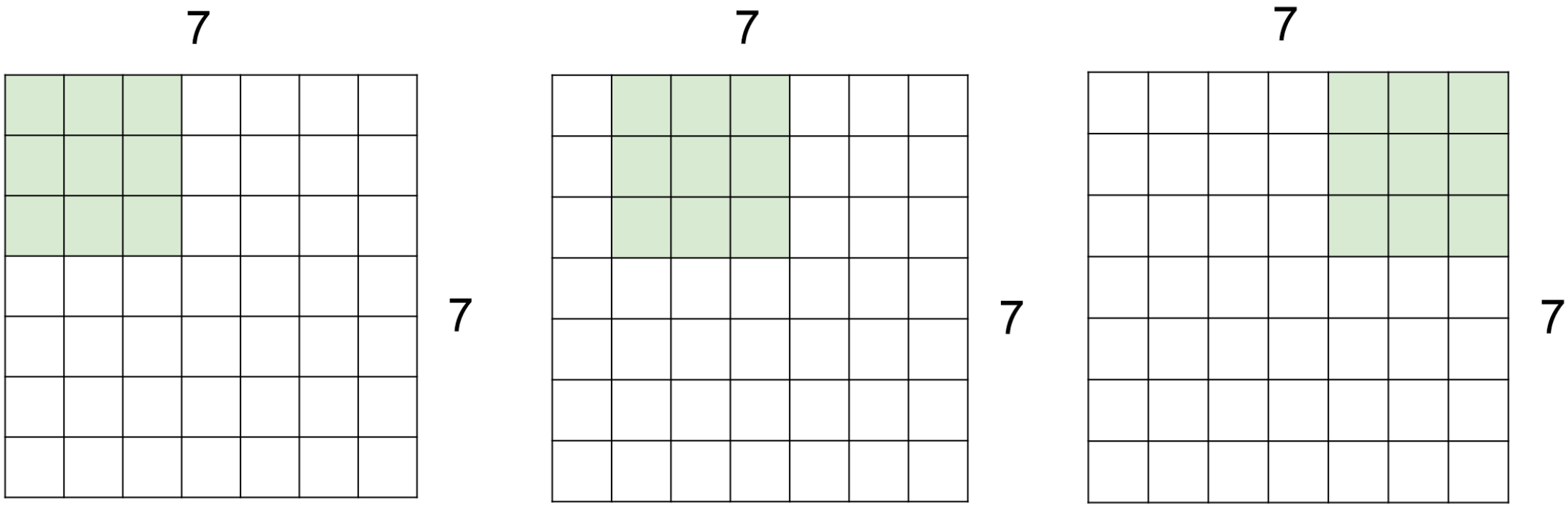
We stack these up to get a "new image" of size 28x28x6!

Convolutional Neural Networks

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



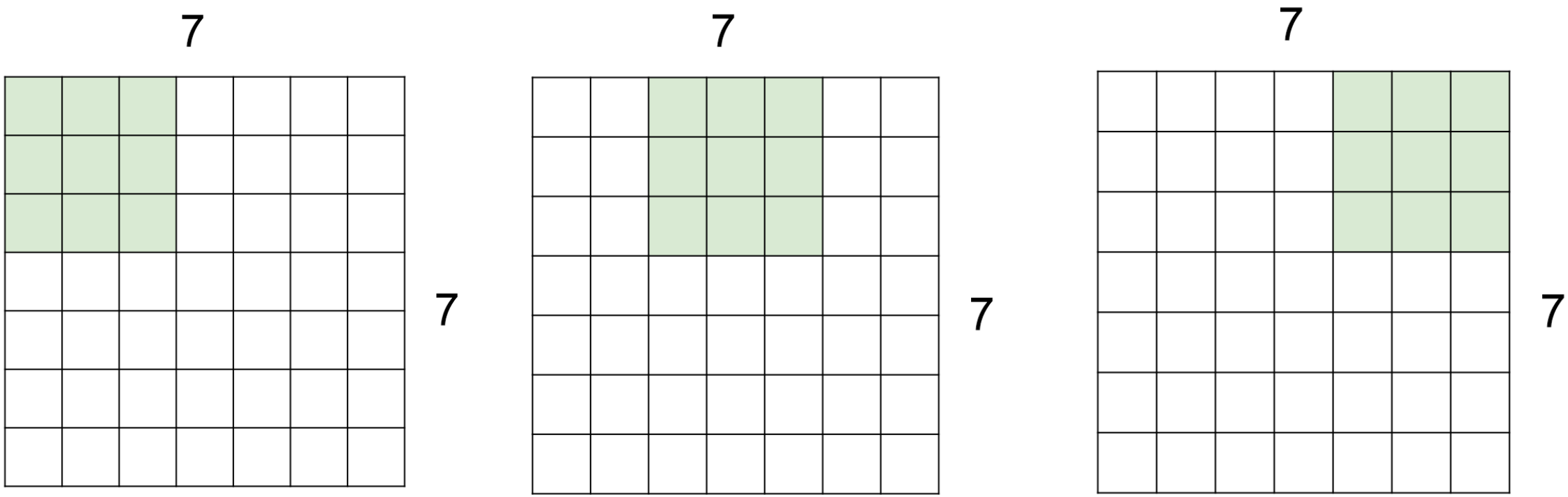
Convolving



7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

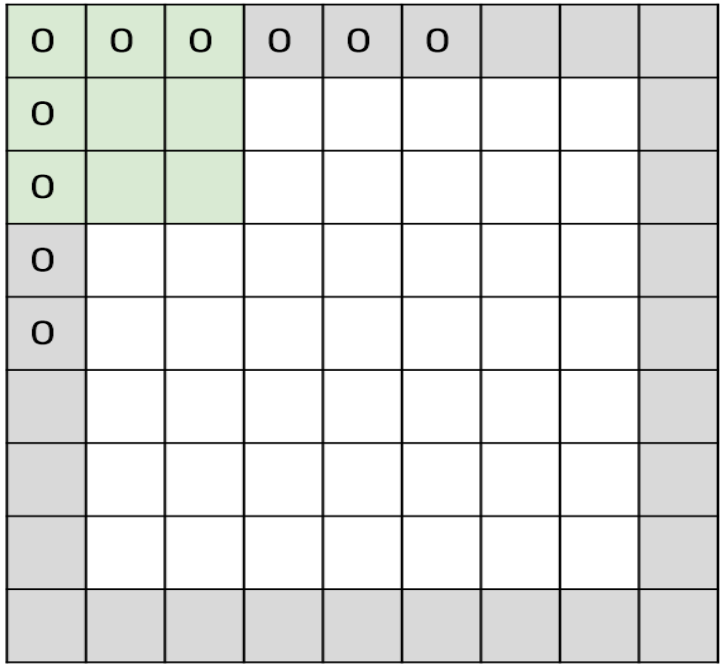
Stride



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

=> 3x3 output!

Paddling

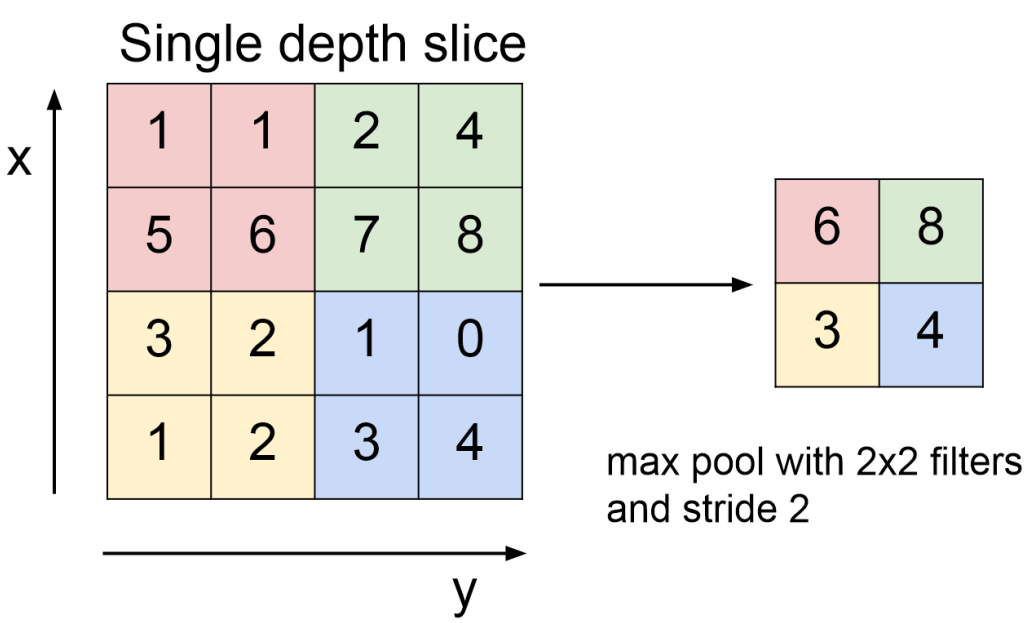


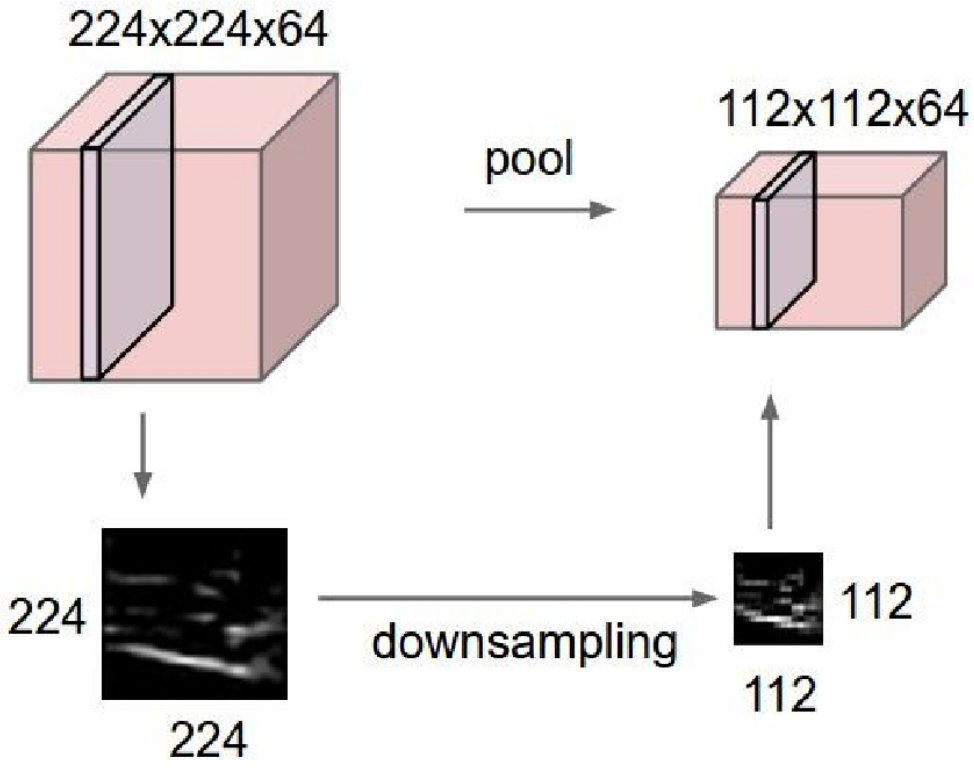
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

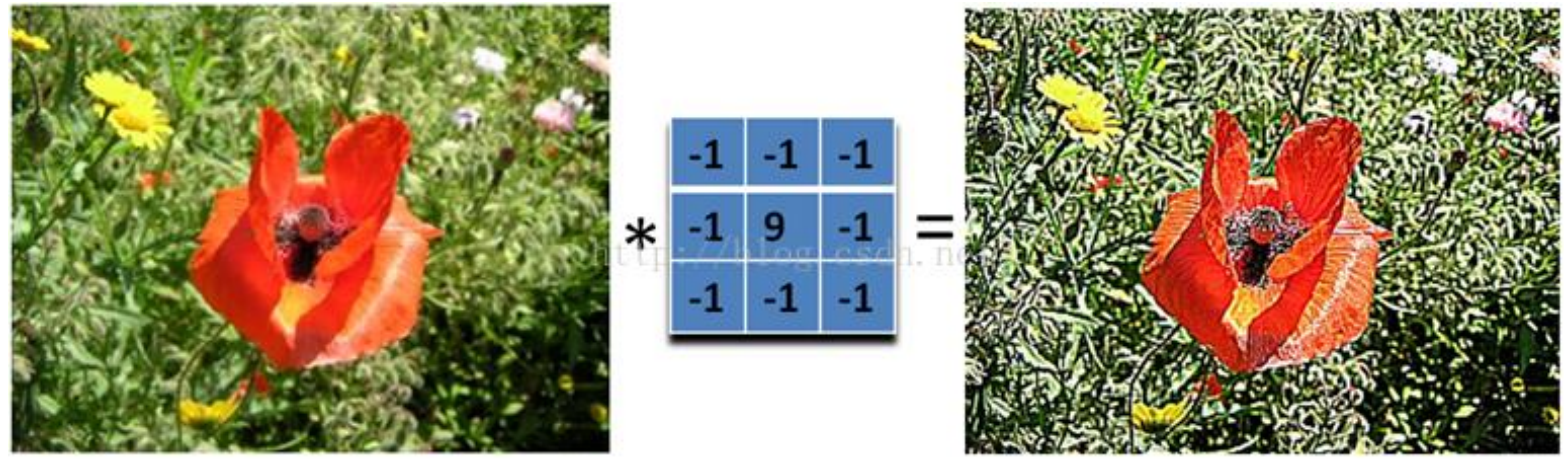
Pooling/Subsampling

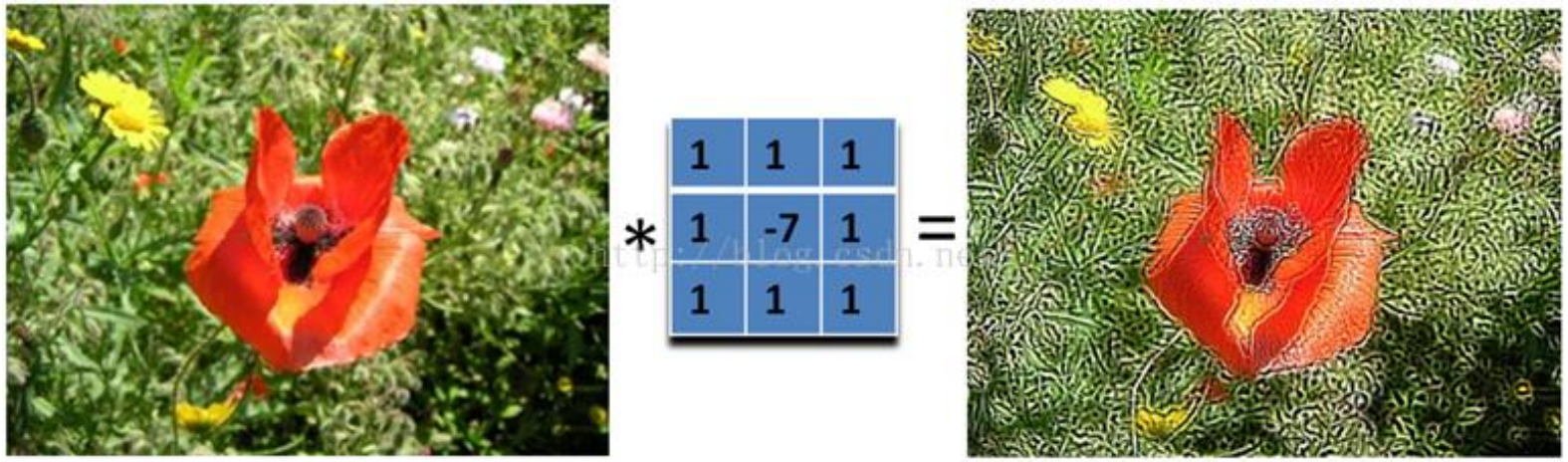
- makes the representations smaller and more manageable
- operates over each activation map independently:



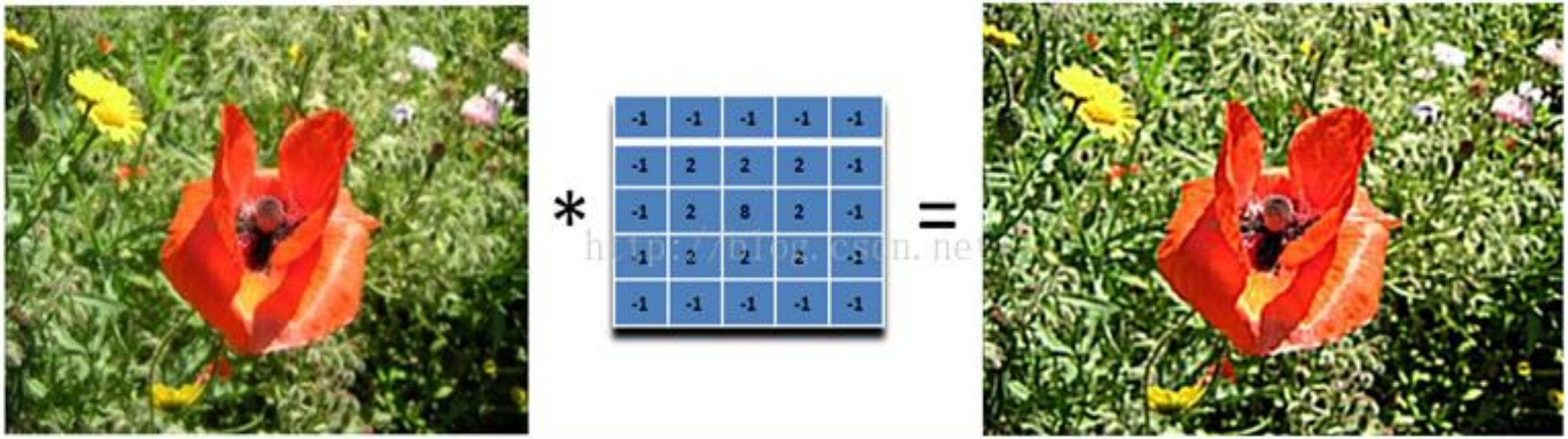


The Convolution Operator

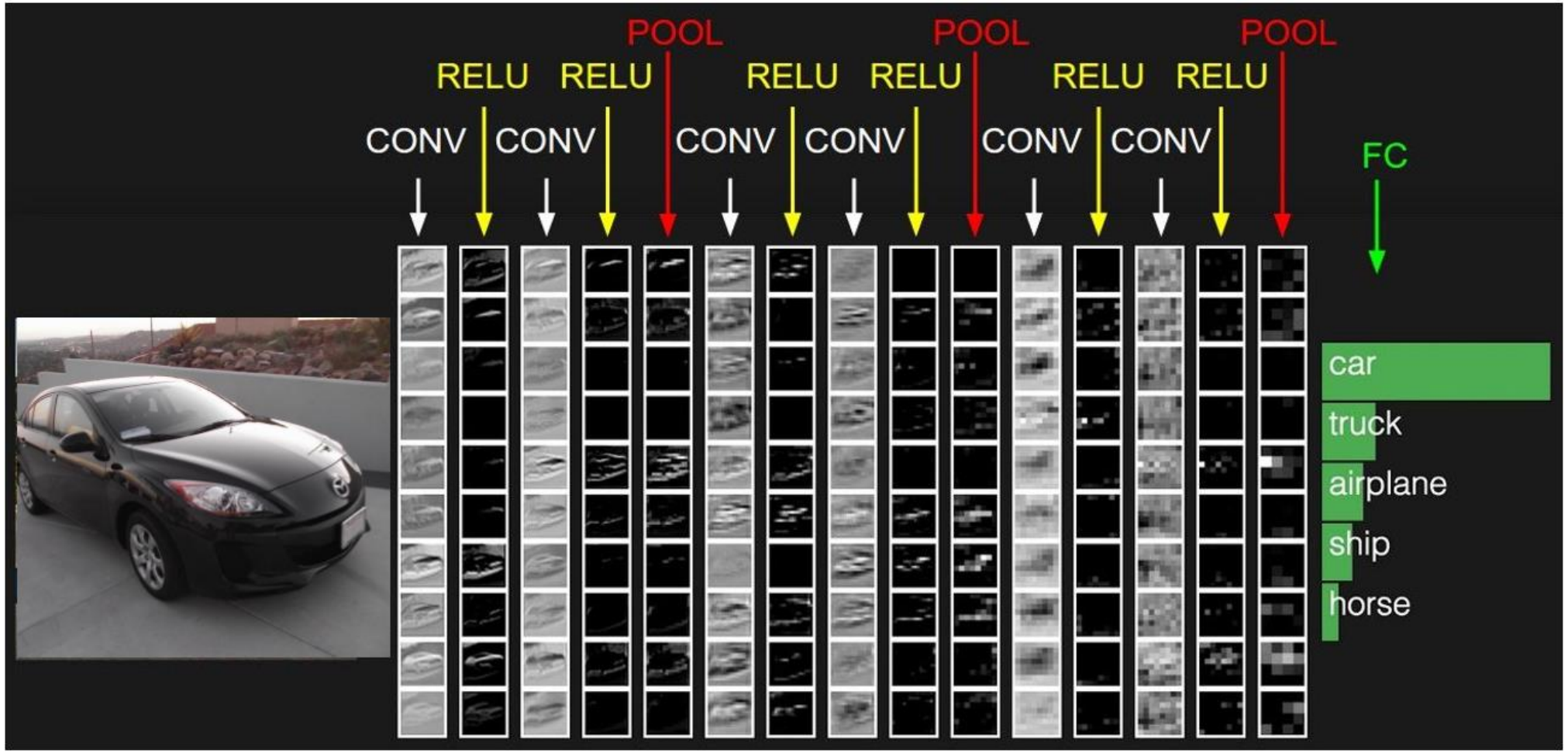




The Convolution Operator

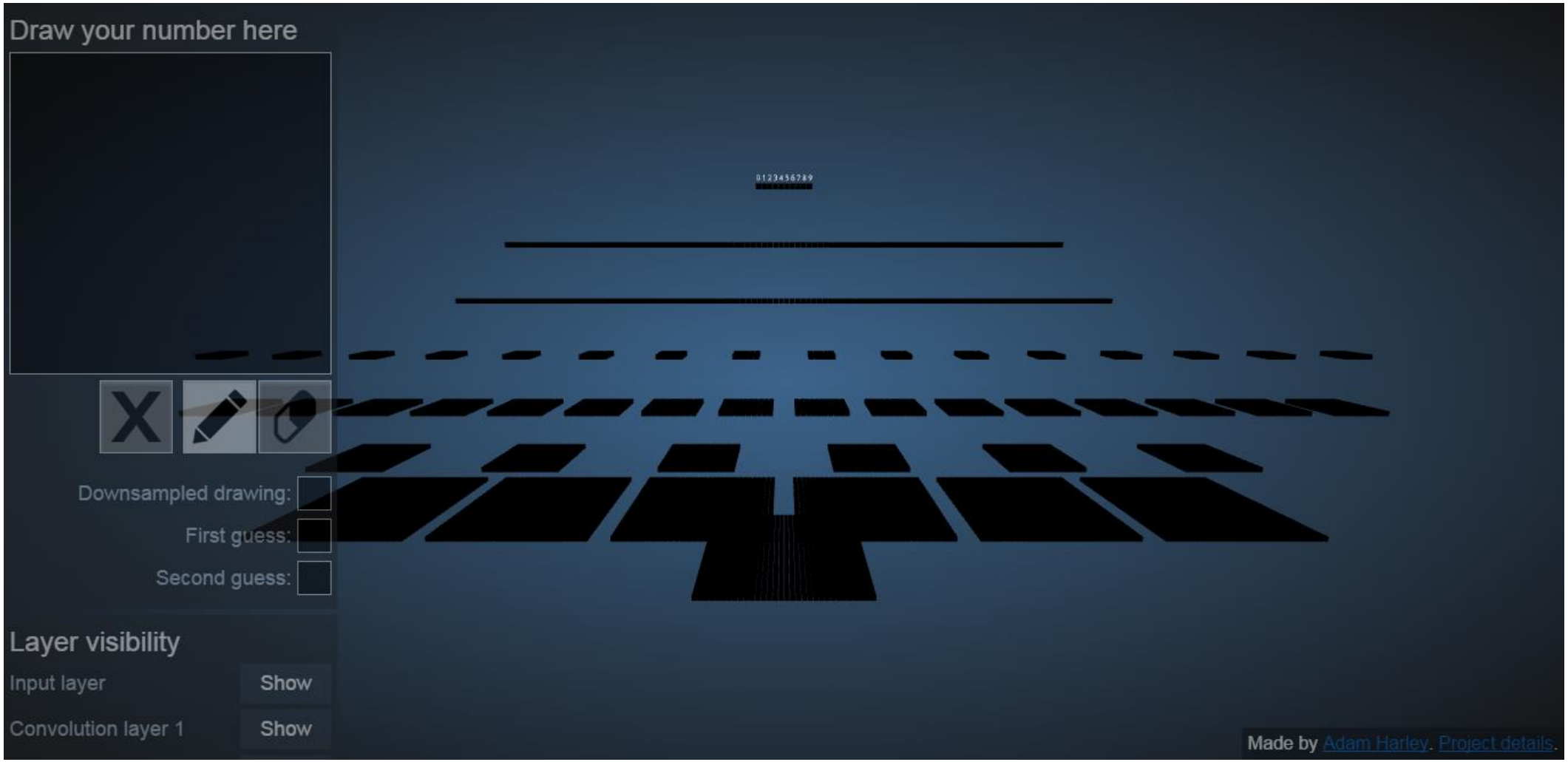


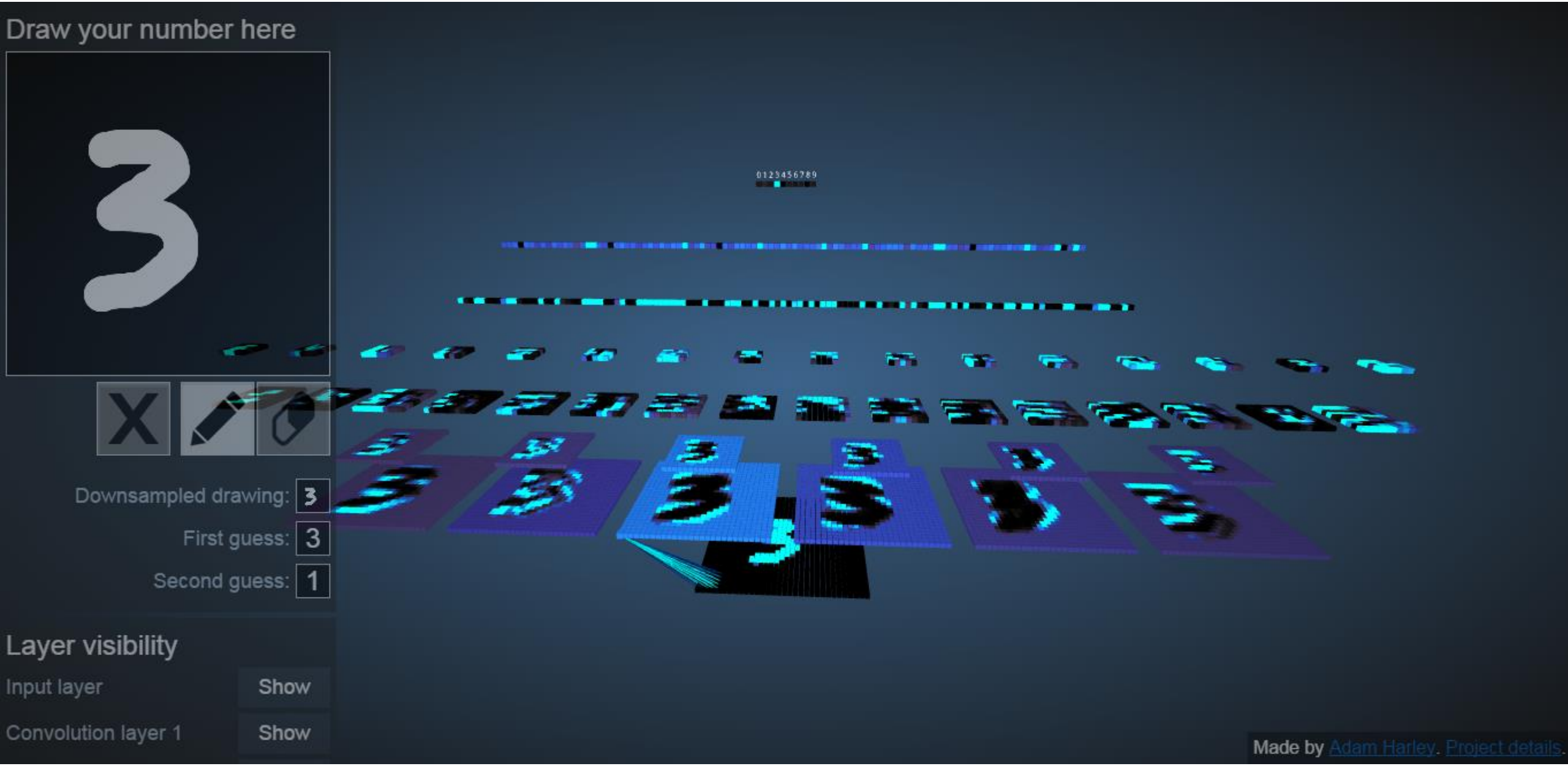
Full Connection



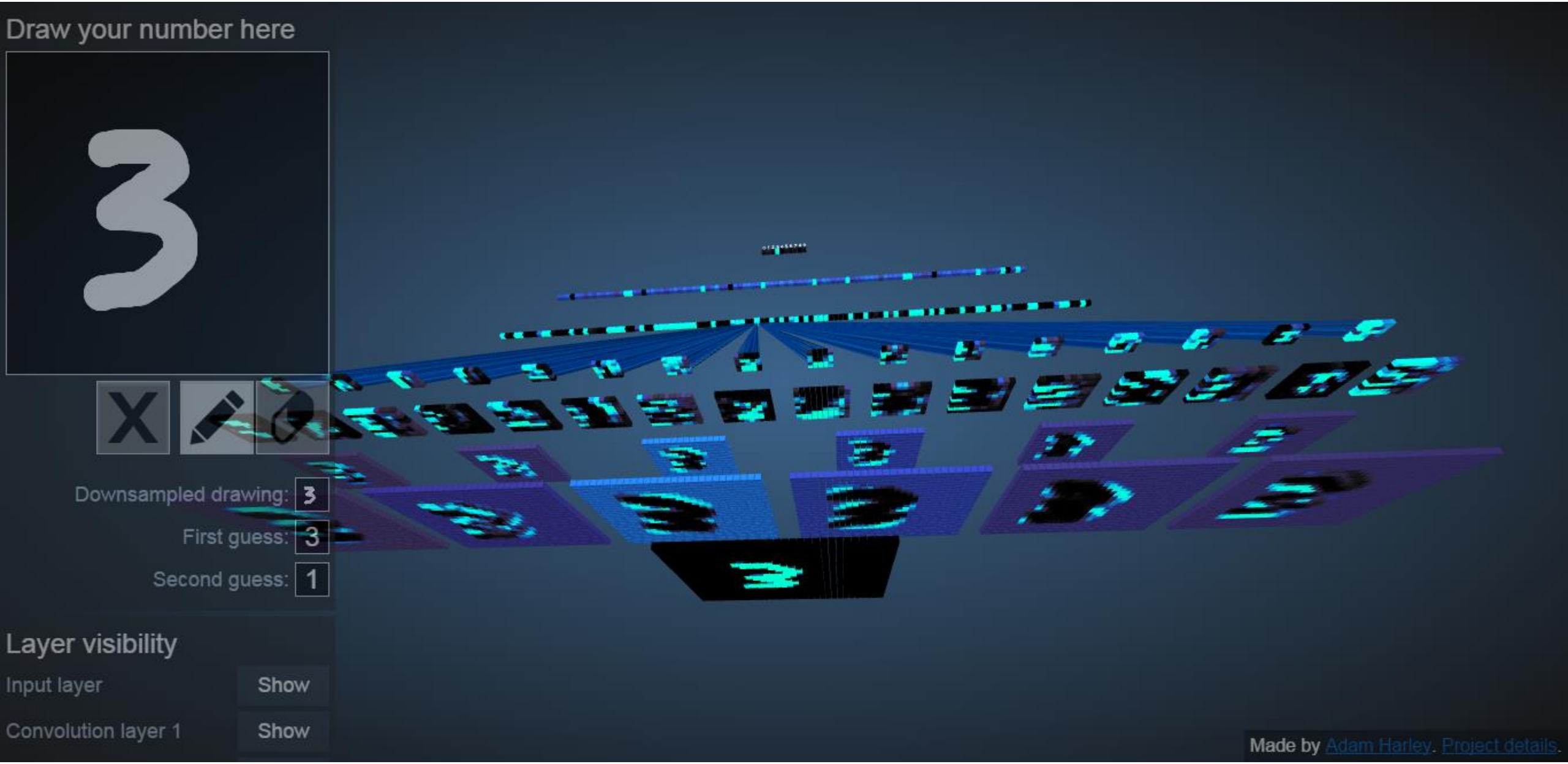
The Demo

http://scs.ryerson.ca/~aharley/vis/conv/

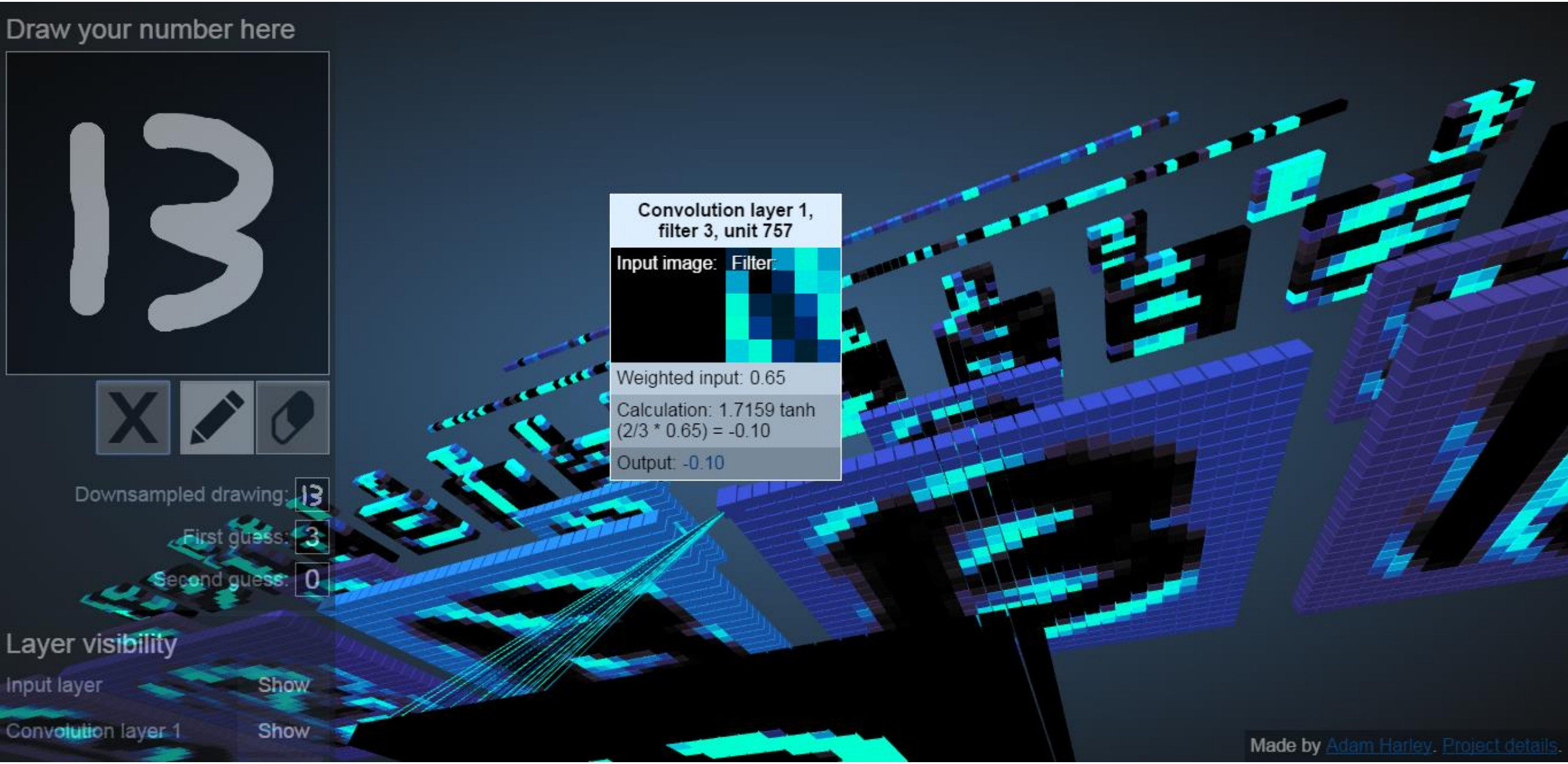




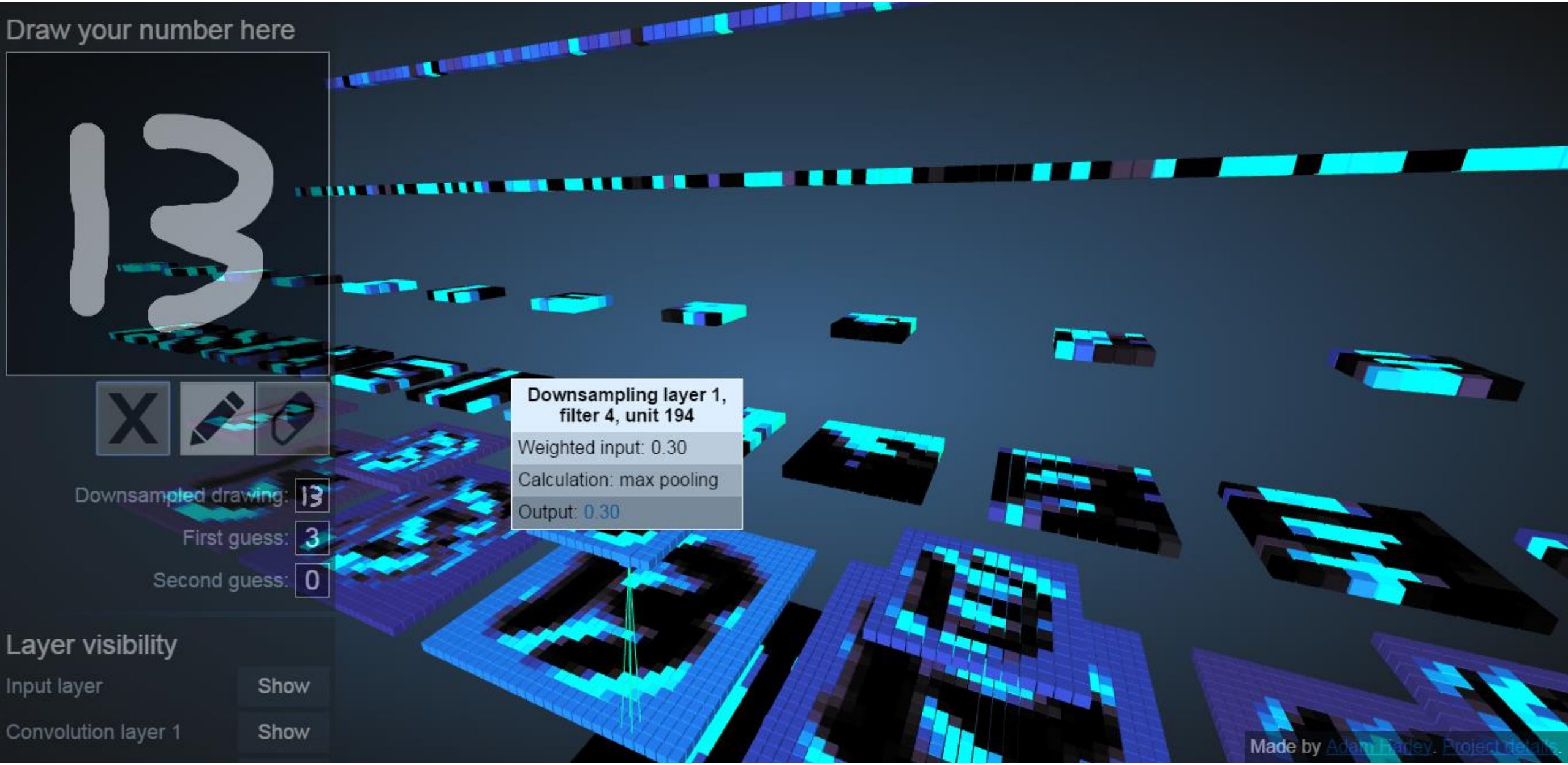
The Convolution Operator



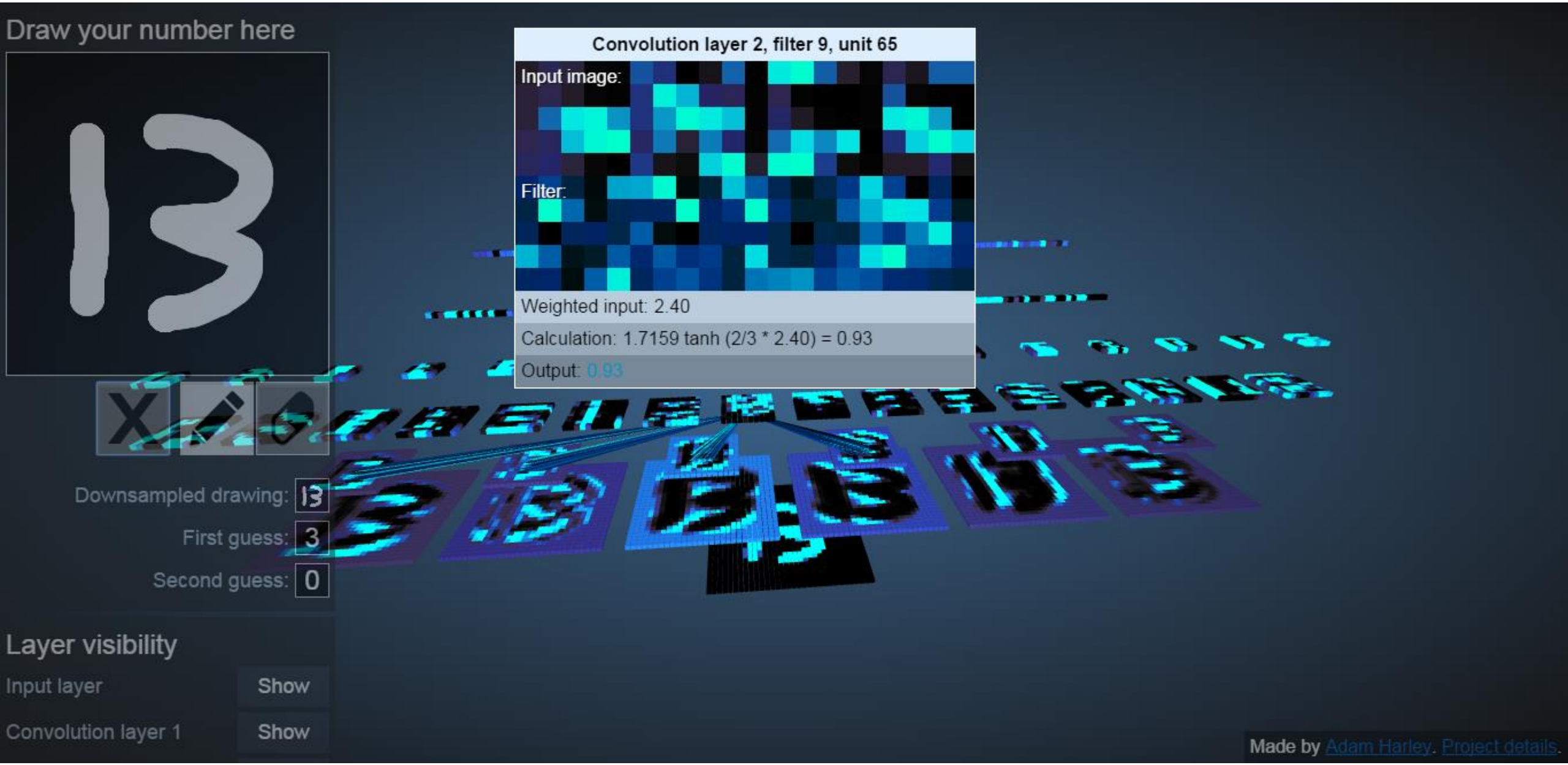
The Fully connect Operator



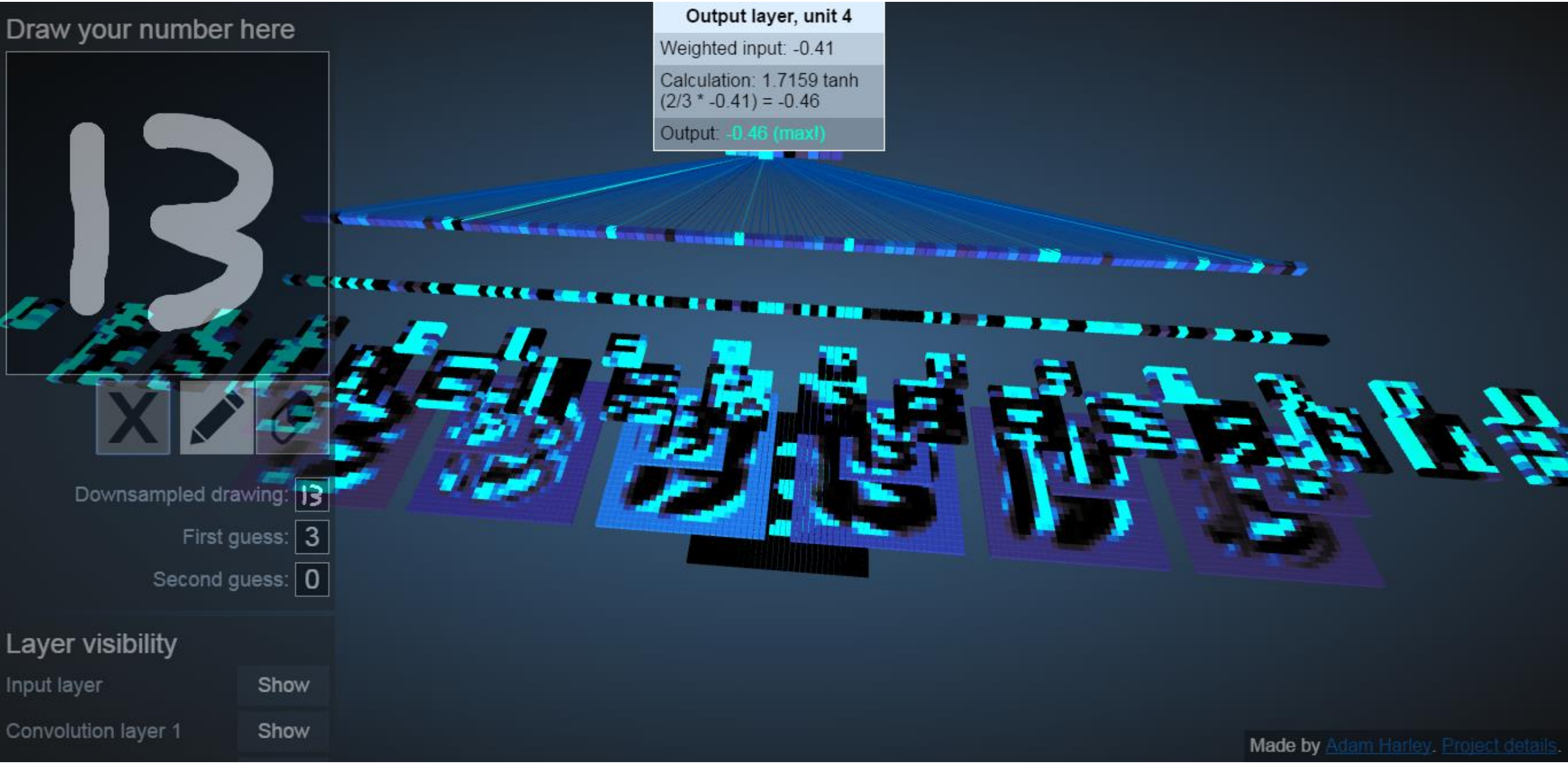
Two-digit situation



Two-digit situation



Two-digit situation



Two-digit situation

IMAGENET Large Scale Visual Recognition Challenge

Year 2010

NEC-UIUC

Dense descriptor grid: HOG, LBP

Coding: local coordinate, super-vector

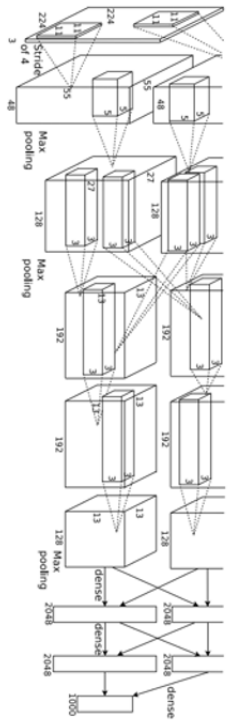
Pooling, SPM

Linear SVM

[Lin CVPR 2011]

Year 2012

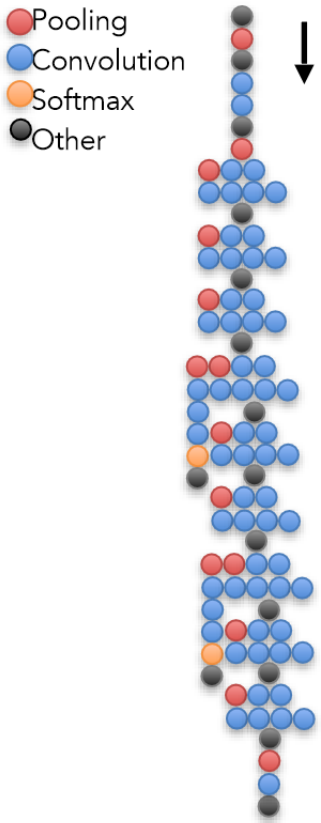
SuperVision



[Krizhevsky NIPS 2012]

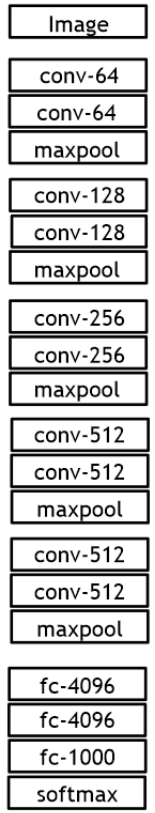
Year 2014

GoogLeNet



[Szegedy arxiv 2014]

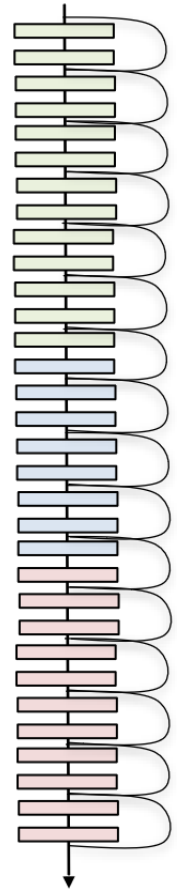
VGG



[Simonyan arxiv 2014]

Year 2015

MSRA

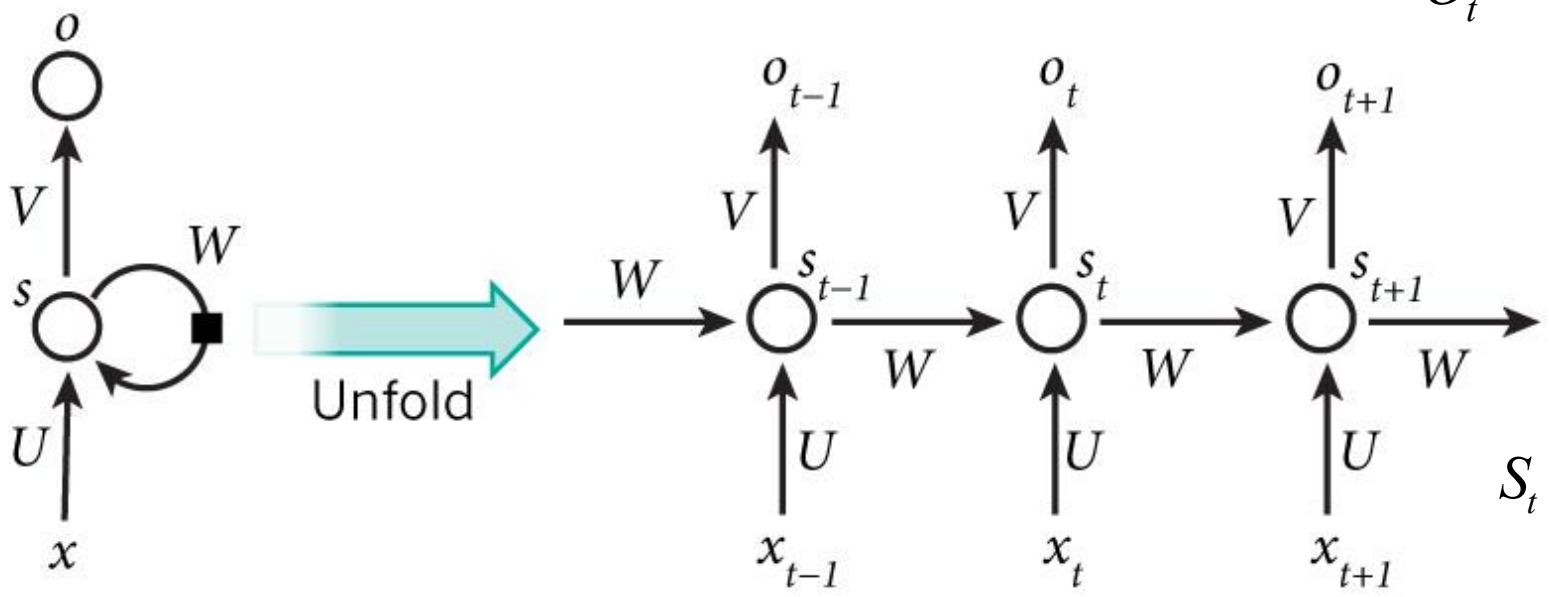


[He ICCV 2015]

Recurrent Neural Network

Recurrent Neural Network

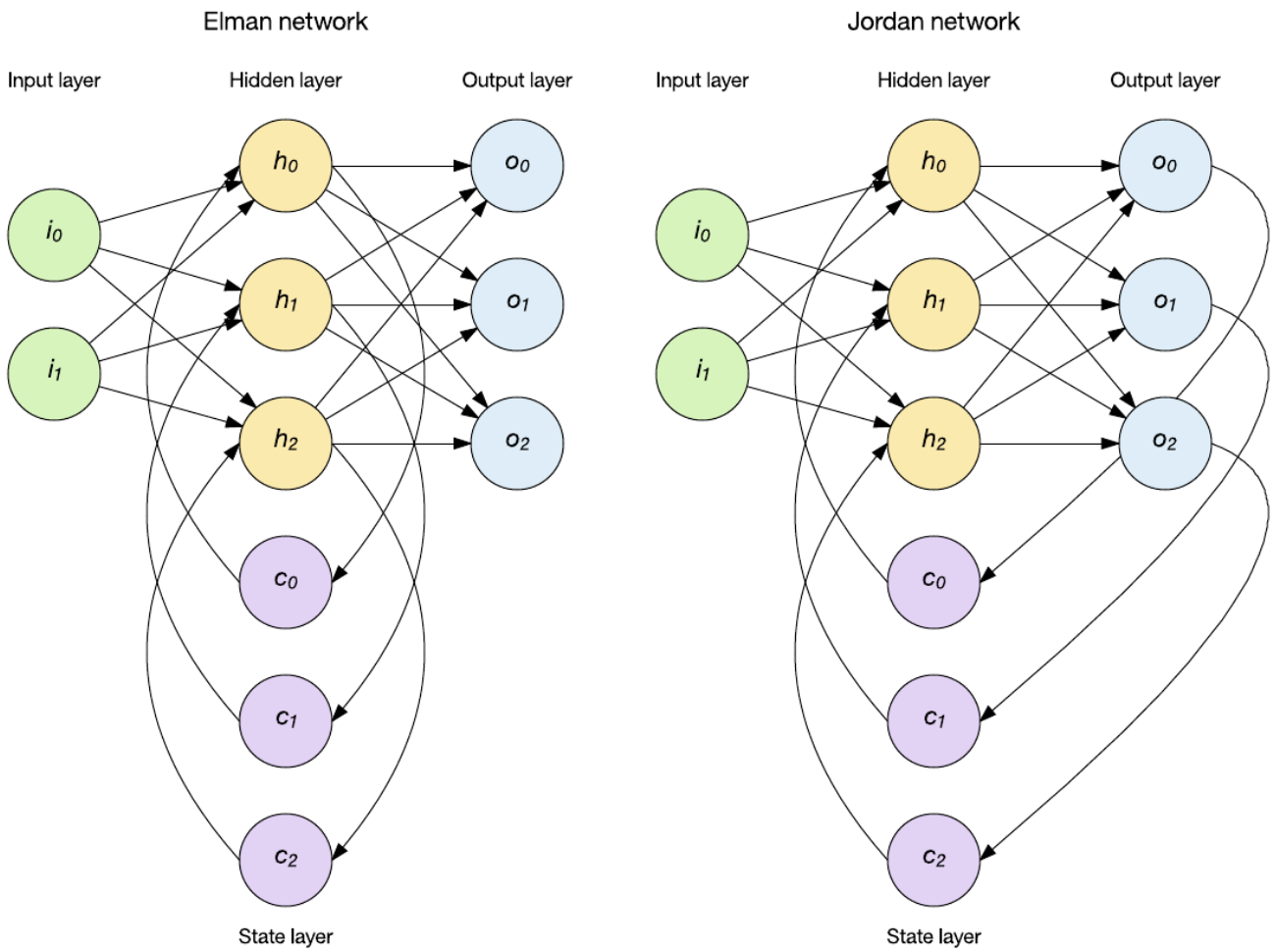
$$O_t = soft\max(V * S_t)$$



$$S_t = \tanh(U * x_t + W * S_{t-1})$$

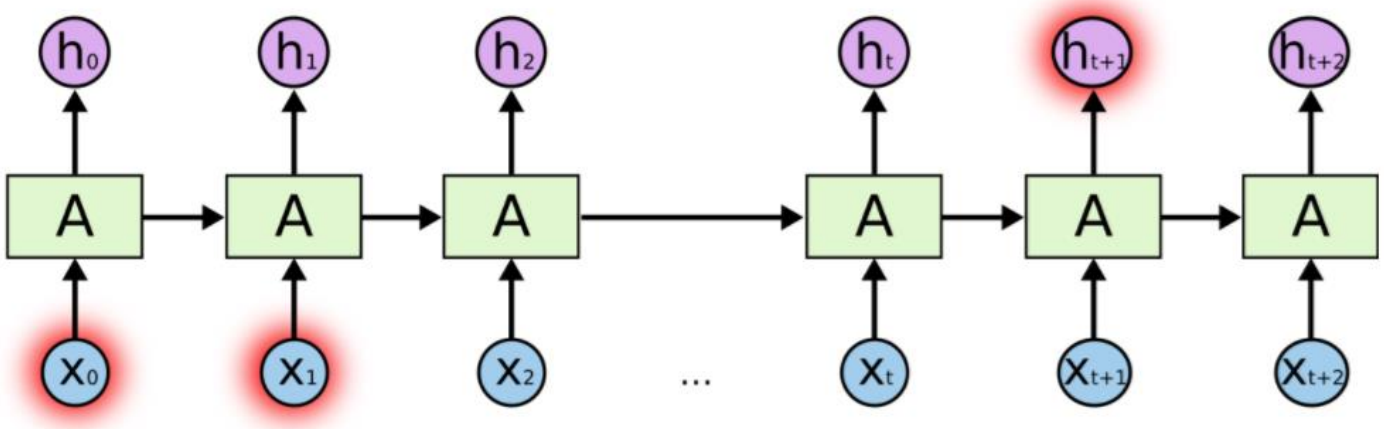
A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop

Different RNN Structures



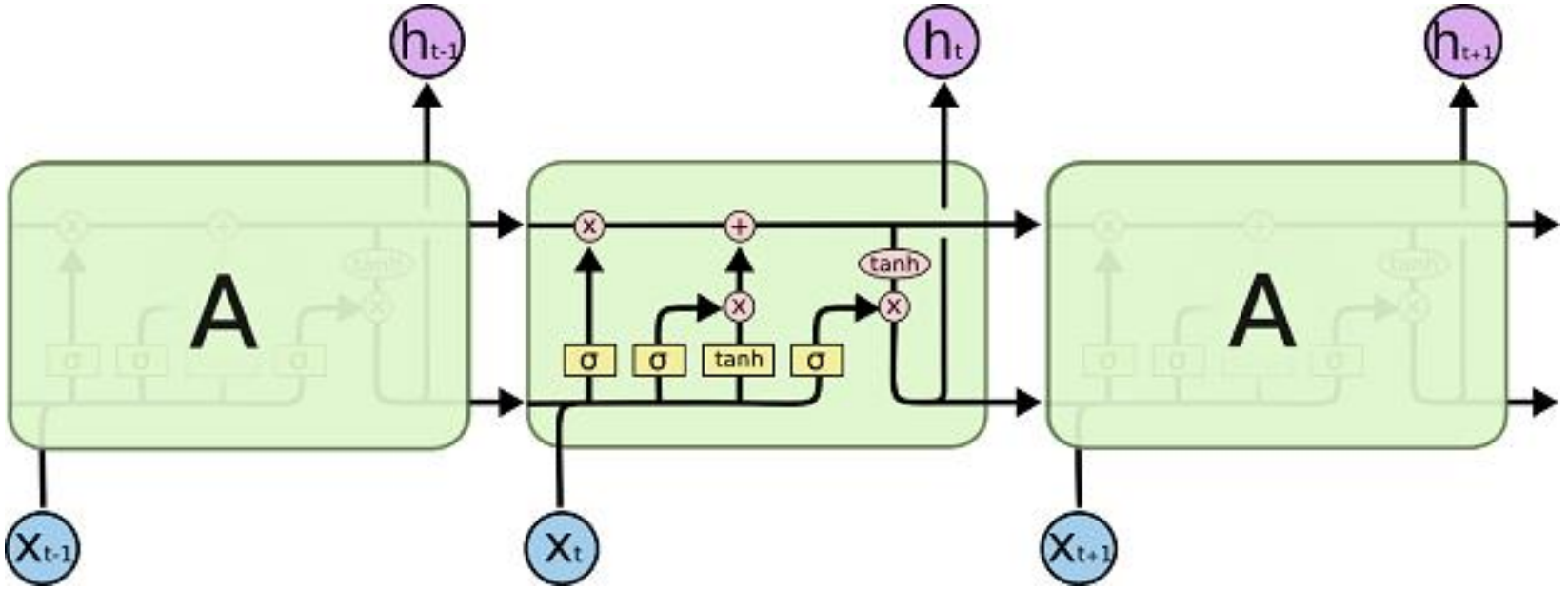
Problem of Long Dependency

Consider trying to predict the last word in the text “I grew up in France… I speak fluent *French*.” Recent information suggests that the next word is probably the name of a language

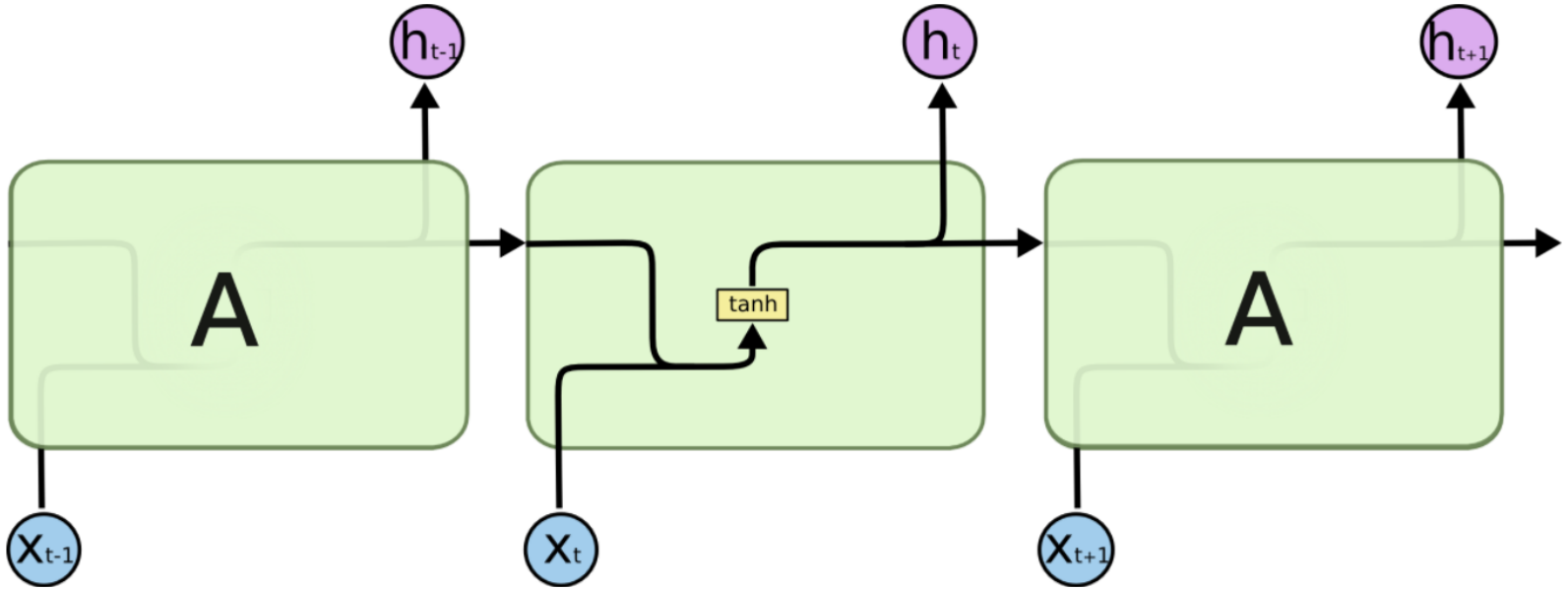


Long Short-Term Memory

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work.



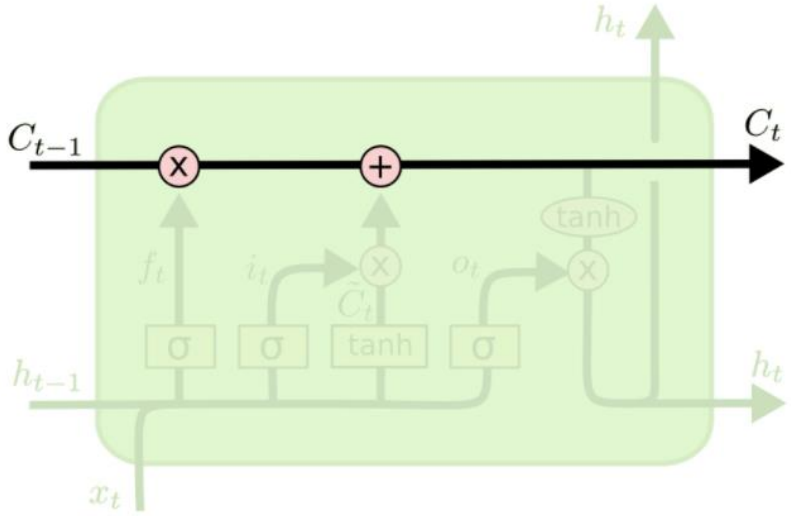
A Special RNN

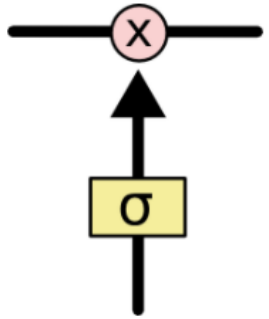


All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

The Core Idea

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.

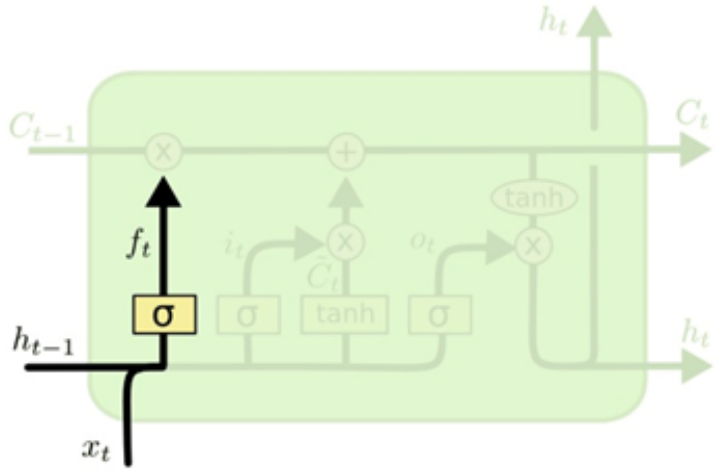




The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"

Step-by-Step LSTM Walk Through

The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer."

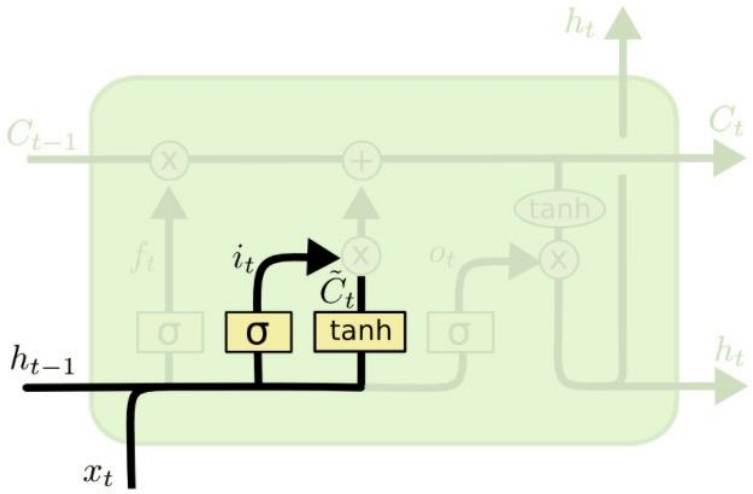


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

It looks at ht−1ht−1and xtxt, and outputs a number between 00 and 11 for each number in the cell state.
A 1 represents "completely keep this" while a 0 represents "completely get rid of this."

Input Gate

The next step is to decide what new information we're going to store in the cell state.
This has two parts. First, a sigmoid layer called the "input gate layer" decides which
values we'll update. Next, a tanh layer creates a vector of new candidate values,
that could be added to the state. In the next step, we'll combine these two to create an update to the state.

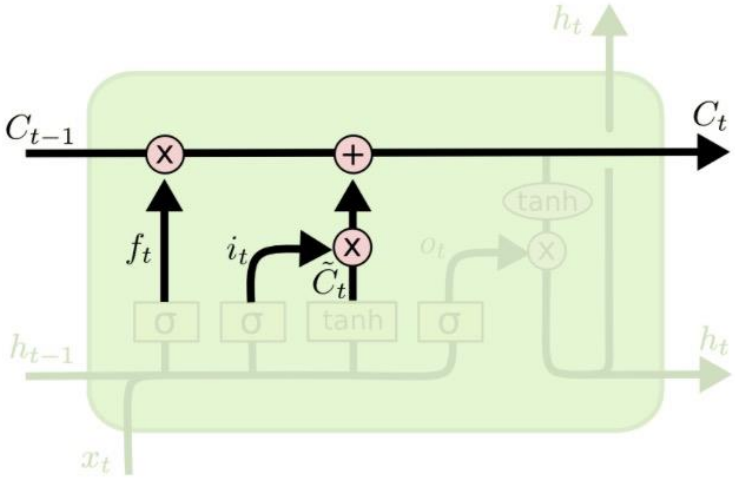


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Cell State Update

It’s now time to update the old cell state, Ct−1, into the new cell state Ct.

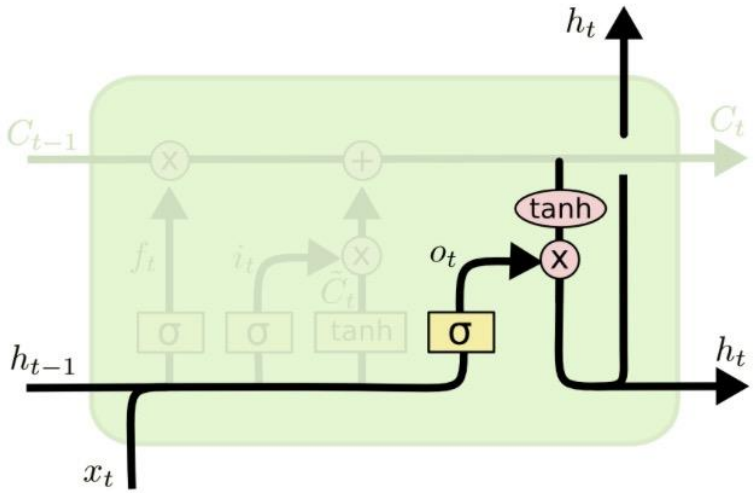


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

In the case of the language model, this is where we’d actually drop the information about the old subject’s gender and add the new information, as we decided in the previous steps.

Output Gate

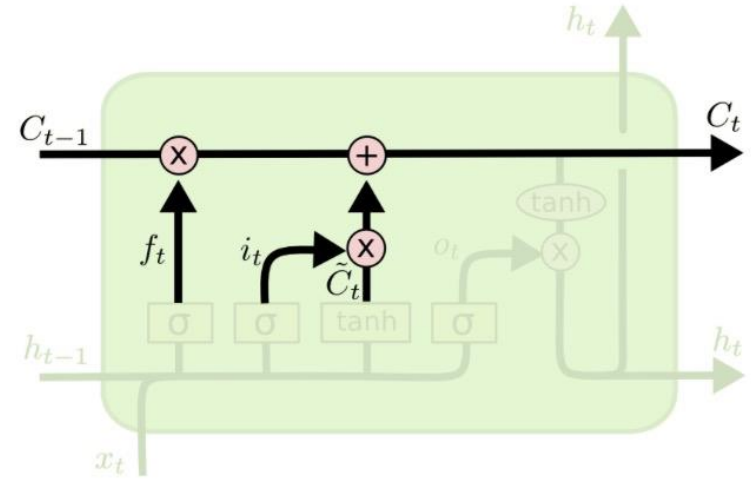
Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through **tanh** (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

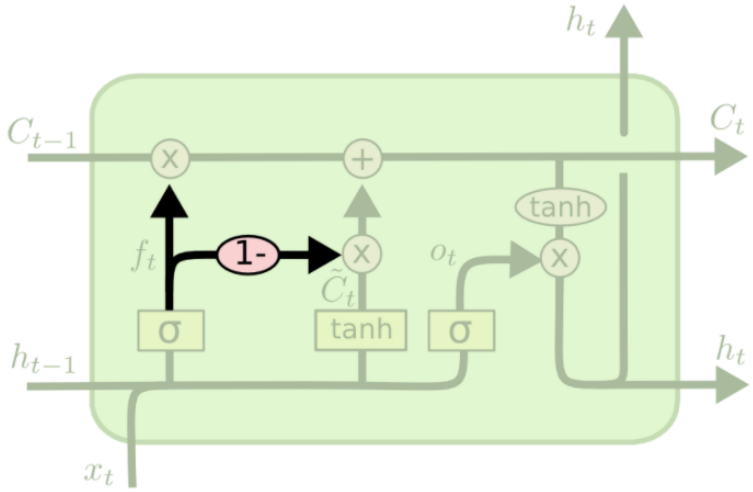
$$h_t = o_t * \tanh\left(C_t\right)$$

Other Variants



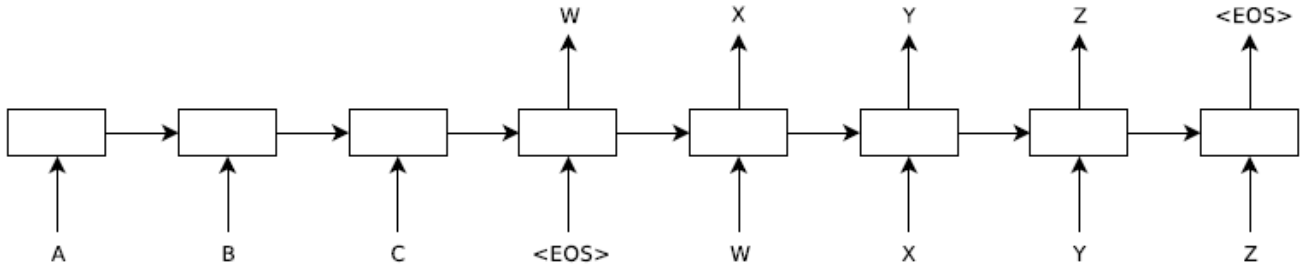
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

We only forget when we're going to input something in its place. We only input new values to the state when we forget something older.

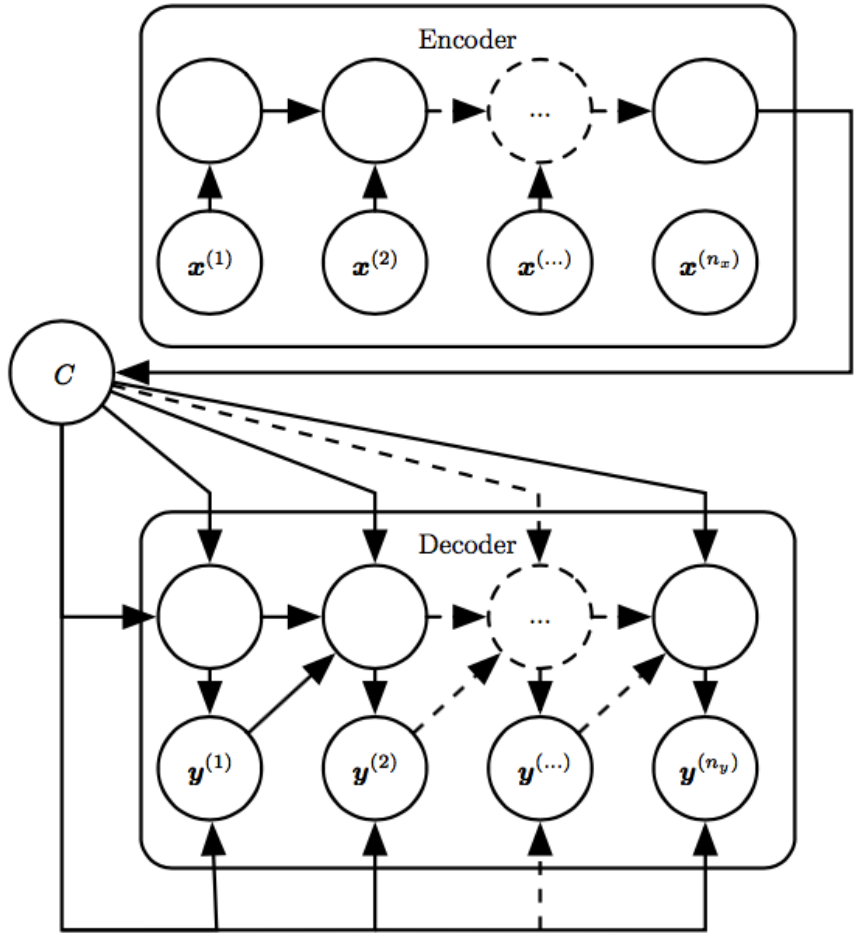


$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

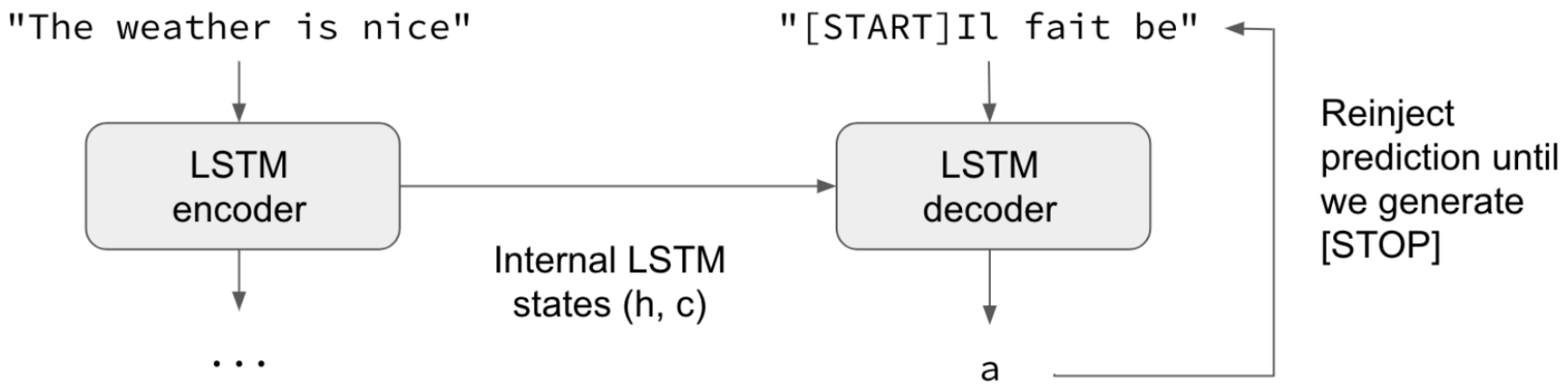
Seq2Seq



$$
\begin{aligned}
h_t &= \text{sigm}\left(W^{\text{hx}}x_t + W^{\text{hh}}h_{t-1}\right) \\
y_t &= W^{\text{yh}}h_t
\end{aligned}
$$

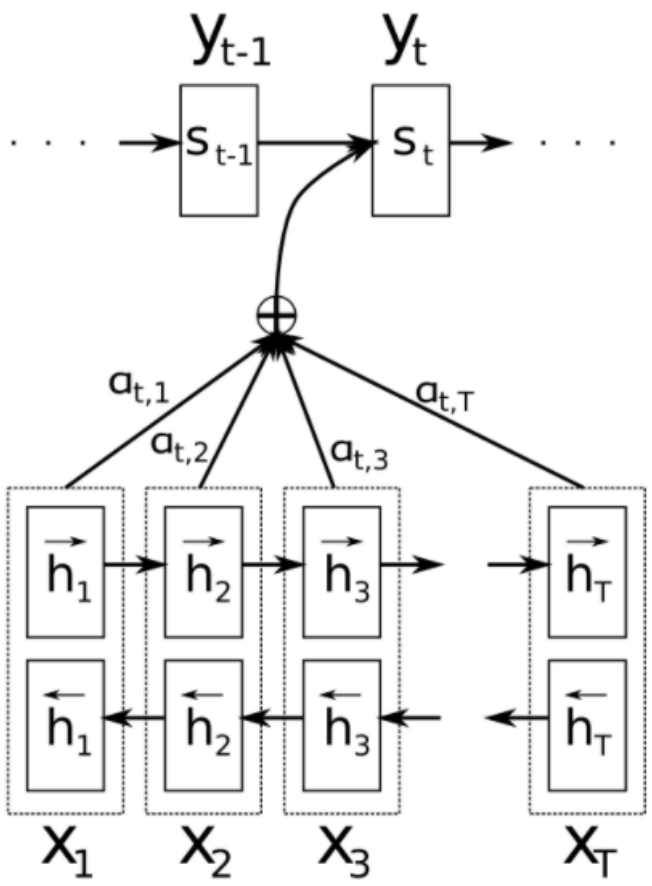


Seq2Seq Example



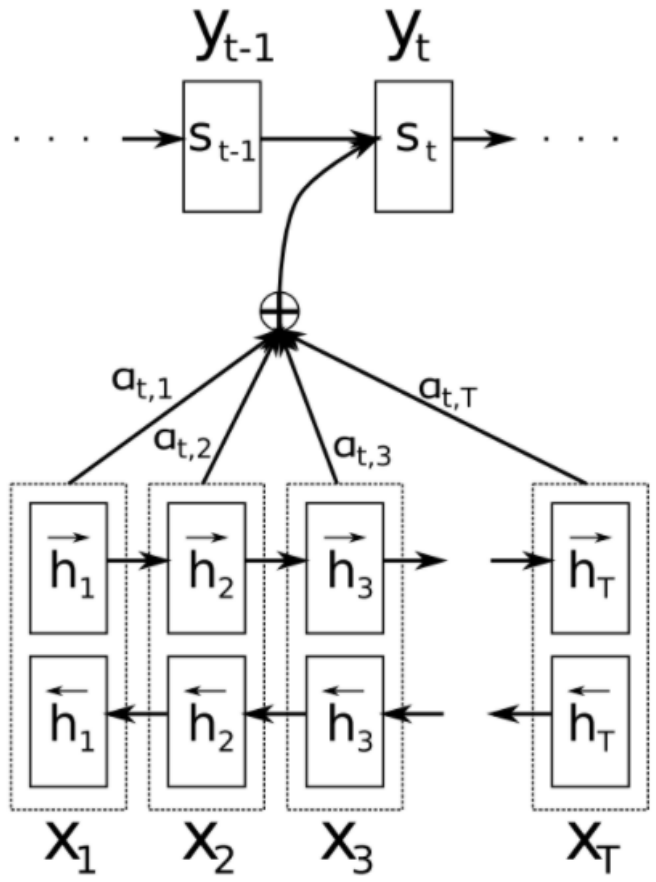
- 1) Encode the input sequence into state vectors.
- 2) Start with a target sequence of size 1 (just the start-of-sequence character).
- 3) Feed the state vectors and 1-char target sequence to the decoder to produce predictions for the next character.
- 4) Sample the next character using these predictions (we simply use argmax).
- 5) Append the sampled character to the target sequence
- 6) Repeat until we generate the end-of-sequence character or we hit the character limit.

Attention Mechanism



- Input sequence -> fixed-sized vector -> target sequence.
- The important part is that each decoder output word y now depends on **a weighted combination** of all the input states, not just the last state.
- The attention distribution is usually generated with content-based attention. The attending RNN generates a query describing what it wants to focus on.

Attention Mechanism



$$p(y_i|y_1,\ldots,y_{i-1},x) = g(y_{i-1},s_i,\boldsymbol{c_i})$$

$$s_i = f(s_{i-1},y_{i-1},\boldsymbol{c_i})$$

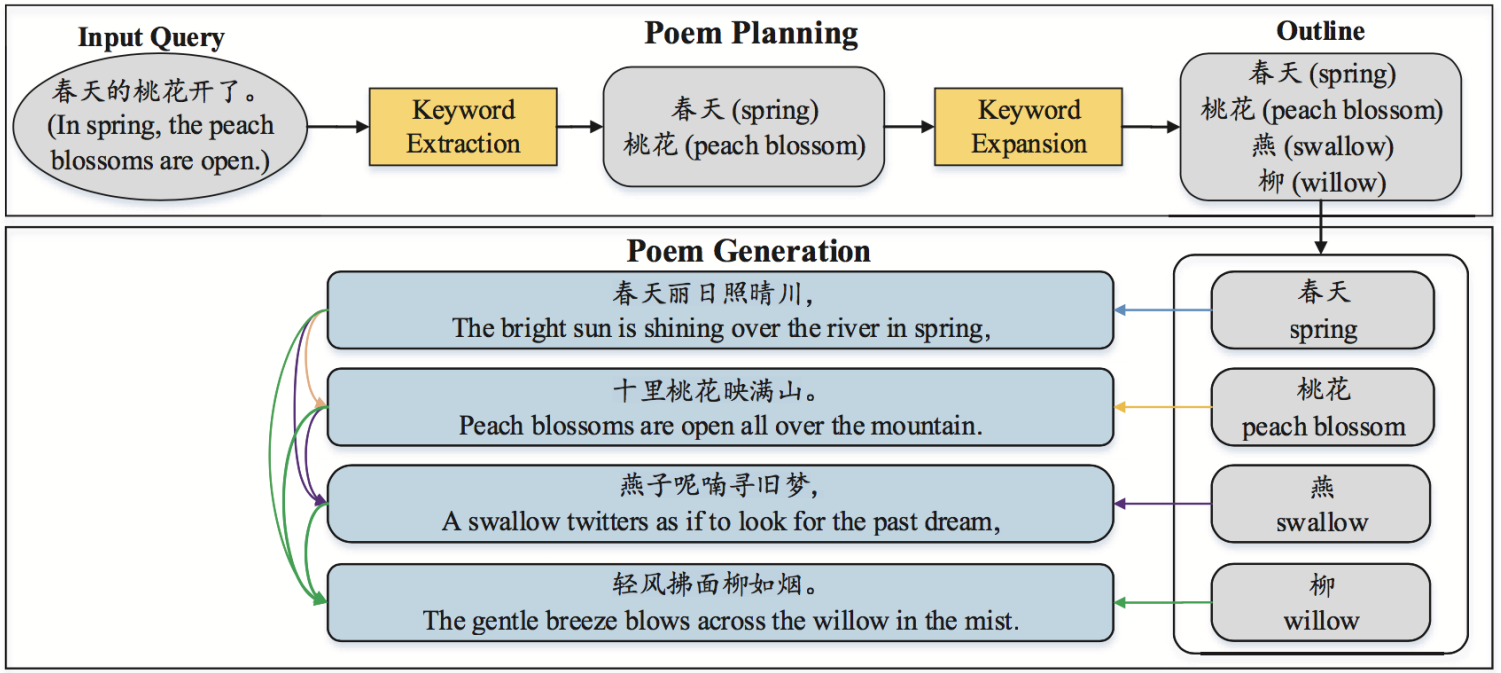
The context vector $\boldsymbol{c_i}$ is computed as a weighted sum of h_j annotations

$$e_{ij} = v^T \tanh(W_1 h_j + W_2 s_{i-1})$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x}\exp(e_{ik})}$$

$$\boldsymbol{c_i} = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

Seq2Seq Example

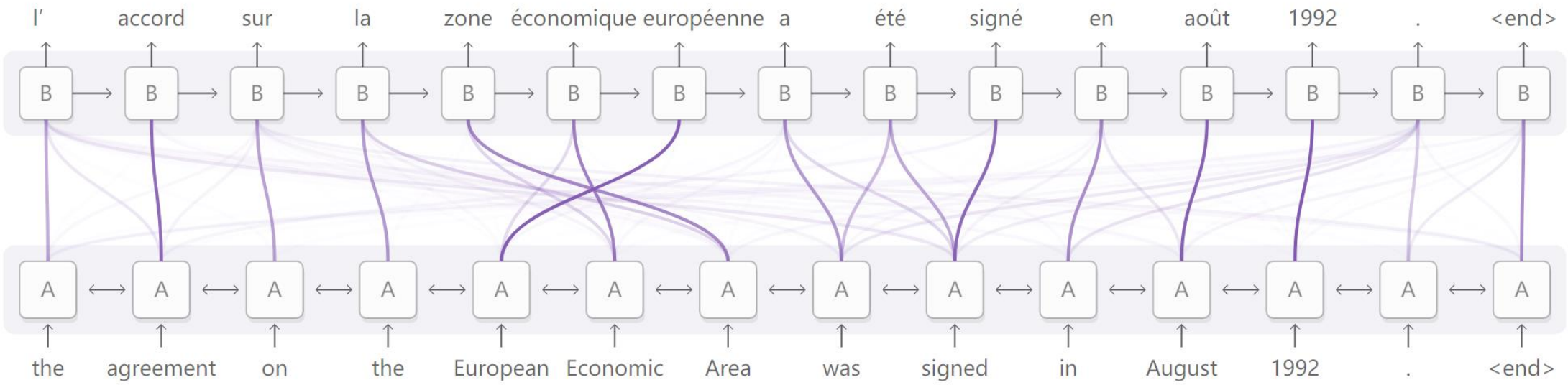


白鹭窥鱼立,
Egrets stood, peeping fishes.
青山照水开.
Water was still, reflecting mountains.
夜来风不动,
The wind went down by nightfall,
明月见楼台.
as the moon came up by the tower.

满怀风月一枝春,
Budding branches are full of romance.
未见梅花亦可人.
Plum blossoms are invisible but adorable.
不为东风无此客,
With the east wind comes Spring.
世间何处是前身.
Where on earth do I come from?

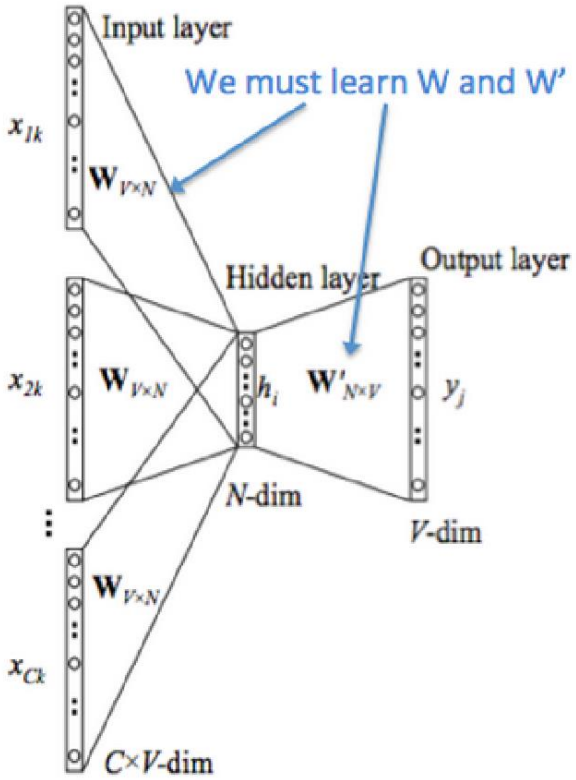
Seq2Seq Example

Attention avoids this by allowing the RNN processing the input to pass along information about each word it sees, and then for the RNN generating the output to focus on words as they become relevant.



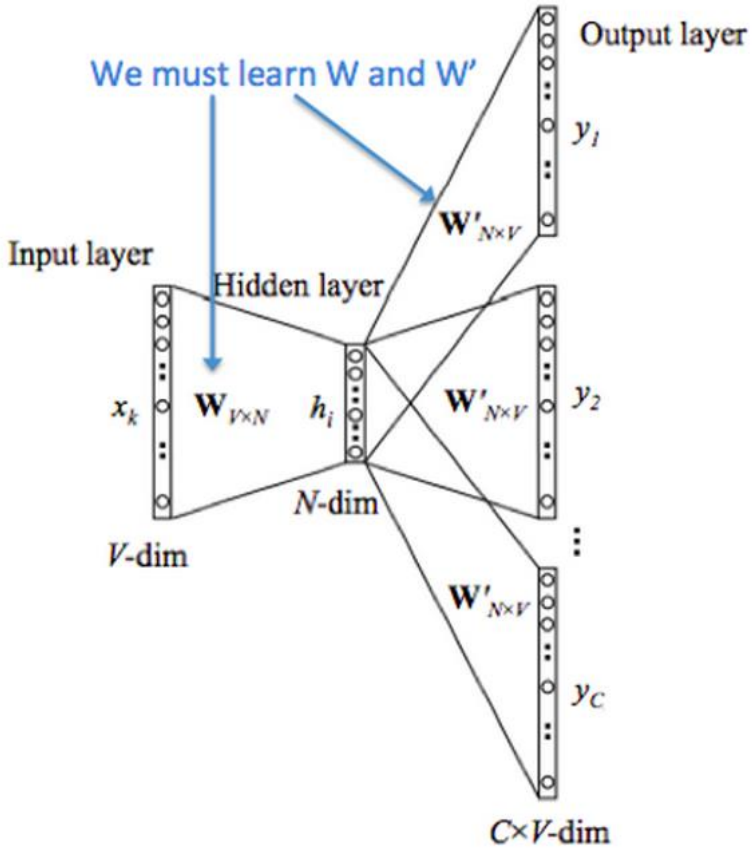
Neural Language Model - CBOW

1. We generate our one hot word vectors for the input context of size $m : (x^{(c-m)}, \ldots, x^{(c-1)}, x^{(c+1)}, \ldots, x^{(c+m)} \in \mathbb{R}^{|V|})$.
2. We get our embedded word vectors for the context ($v_{c-m} = \mathcal{V}x^{(c-m)}$, $v_{c-m+1} = \mathcal{V}x^{(c-m+1)}$, $\ldots$, $v_{c+m} = \mathcal{V}x^{(c+m)} \in \mathbb{R}^n$)
3. Average these vectors to get $\hat{v} = \frac{v_{c-m} + v_{c-m+1} + \ldots + v_{c+m}}{2m} \in \mathbb{R}^n$
4. Generate a score vector $z = \mathcal{U}\hat{v} \in \mathbb{R}^{|V|}$. As the dot product of similar vectors is higher, it will push similar words close to each other in order to achieve a high score.
5. Turn the scores into probabilities $\hat{y} = \text{softmax}(z) \in \mathbb{R}^{|V|}$.
6. We desire our probabilities generated, $\hat{y} \in \mathbb{R}^{|V|}$, to match the true probabilities, $y \in \mathbb{R}^{|V|}$, which also happens to be the one hot vector of the actual word.



Word2Vec Skip-Gram

1. We generate our one hot input vector $x \in \mathbb{R}^{|V|}$ of the center word.
2. We get our embedded word vector for the center word $v_c = \mathcal{V}x \in \mathbb{R}^n$
3. Generate a score vector $z = \mathcal{U}v_c$.
4. Turn the score vector into probabilities, $\hat{y} = \text{softmax}(z)$. Note that $\hat{y}_{c-m}, \ldots, \hat{y}_{c-1}, \hat{y}_{c+1}, \ldots, \hat{y}_{c+m}$ are the probabilities of observing each context word.
5. We desire our probability vector generated to match the true probabilities which is $y^{(c-m)}, \ldots, y^{(c-1)}, y^{(c+1)}, \ldots, y^{(c+m)}$, the one hot vectors of the actual output.



References

Andew Ng, Lectures notes of Machine Learning, CS229 at Stanford.
T. Hastie, R. Tibshirani and J. Friedman, The Elements of Statistical Learning, 2001, Springer.
http://cs231n.github.io/ (Stanford CS231n, Fei-Fei Li)
http://colah.github.io/posts/2015-08-Understanding-LSTMs/