Linear Algebra Preparation for Emergent Neural Network Architectures (LAPENNA)

Unit 7

Convolutional neural Network II

Kwai Wong, Stan Tomov,
Julian Halloy, Stephen Qiu, Eric Zhao

March 17, 2022

University of Tennessee, Knoxville

Acknowledgements:

- Support from NSF, UTK, JICS, ICL, NICS
- LAPENNA, www.jics.utk.edu/lapenna, NSF award #202409
- www.icl.utk.edu, cfdlab.utk.edu, www.xsede.org, www.jics.utk.edu/recsem-reu,
- MagmaDNN is a project grown out from the RECSEM REU Summer program supported under NSF award #1659502
- Source code: www.bitbucket.org/icl/magmadnn
- www.bitbucket.org/cfdl/opendnnwheel

INNOVATIVE COMPUTING LABORATORY

Overview of LAPENNA

The major goal of The Linear Algebra Preparation for Emergent Neural Network Architectures (LAPENNA) program is to provide essential knowledge to advance literacy in AI to sustain the growth and development of the workforce in the cyberinfrastructure (CI) ecosystem. This program aims to prepare college faculty, researchers, and industrial practitioners to design, enable and direct their own course curricula, collaborative projects, and training programs for in-house data-driven sciences programs. The LAPENNA program focuses on delivering computational techniques, numerical algorithms and libraries, and implementation of AI software on emergent CPU and GPU platforms.

Ecosystem Linear Algebra Preparation for Emergent Neural Network Architectures (LAPENNA)

Modeling, Numerical Linear Algebra, Data Analytics, Machine Learning, DNN, GPU, HPC

Session 1	Session 2	Session 3	Session 4
7/2020 - 12/2020	1/2021- 6/2021	7/2021 - 1/2022	1/2022 - 6/2022
16 participants	16 participants	16 participants	16 participants
Faculty/Students	Faculty/Students	Faculty/Students	Faculty/Students
10 webinars	10 webinars	10 webinars	10 webinars
4 Q & A	4 Q & A	4 Q & A	4 Q & A

Colleges courses, continuous integration, online courses, projects, software support

Web-based resources, tutorials, webinars, training, outreach

Teams and Participants of LAPENNA

- ✓ **PIs : Kwai Wong (JICS), Stan Tomov (ICL), University of Tennessee, Knoxville**
  - **Stephen Qiu, Julian Halloy, Eric Zhao (Students)**

- ✓ **Team : Clemson University**
- ✓ **Teams : University of Arkansas**
- ✓ **Team : University of Houston, Clear Lake**
- ✓ **Team : Miami University, Ohio**
- ✓ **Team : Boston University**
- ✓ **Team : West Virginia University**
- ✓ **Team : Louisiana State University, Alexandria**
- ✓ **Teams : Jackson Laboratory**
- ✓ **Team : Georgia State University**
- ✓ **Teams : University of Tennessee, Knoxville**
- ✓ **Teams : Morehouse College, Atlanta**
- ✓ **Team : North Carolina A & T University**
- ✓ **Team : Clark Atlanta University, Atlanta**
- ✓ **Team : Alabama A & M University**
- ✓ **Team : Slippery Rock University**
- ✓ **Team : University of Maryland, Baltimore County**

Meeting Time

- ✓ **Webinar Meeting time. Thursday 8:00 – 10:00 pm ET,**
- ✓ **Tentative schedule, www.jics.utk.edu/lapenna --> Spring 2022**

Topic: LAPENNA Spring 2022 Webinar
Time: Feb 3, 2022 08:00 PM Eastern Time (US and Canada)
Every week on Thu, 12 occurrence(s)
Feb 3, 2022 07:30 PM
Feb 10, 2022 07:30 PM
Feb 17, 2022 07:30 PM
Feb 24, 2022 07:30 PM
Mar 3, 2022 07:30 PM
Mar 10, 2022 07:30 PM
Mar 17, 2022 07:30 PM
Mar 24, 2022 07:30 PM
Mar 31, 2022 07:30 PM
Apr 7, 2022 07:30 PM
Apr 14, 2022 07:30 PM
Apr 21, 2022 07:30 PM

Join from PC, Mac, Linux, iOS or Android: https://tennessee.zoom.us/j/94140469394
Password: 708069

Schedule of LAPENNA Spring 2022
Thursday 8:00pm -10:00pm Eastern Time

The goal of The Linear Algebra Preparation for Emergent Neural Network Architectures (LAPENNA) program is to provide essential knowledge to advance literacy in AI to sustain the growth and development of the workforce in the cyberinfrastructure (CI) ecosystem for data-driven applications.

Month	Week	Date	Topics
February	Week 01	3	Logistics, High Performance Computing
	Week 02	10	Computational Ecosystem, Linear Algebra
	Week 03	17	Introduction to DNN, Forward Path (MLP)
	Week 04	24	Backward Path (MLP), Math, Example
March	Week 05	3	Backpro (MLP), CNN Computation
	Week 06	10	CNN Backpropagation, Example
	Week 07	17	CNN Network, Linear Algebra
	Week 08	24	Segmentation, Unet,
	Week 09	31	Object Detection, RC vehicle
April	Week 10	7	RNN, LSTEM, Transformers
	Week 11	14	DNN Computing on GPU
	Week 12	21	Overview, Closing
June or July	Workshop	To be arranged	4 Days In Person at UTK

Topics of Unit 6 - CNN

- ✓ **UNIT 7 : Convolutional Neural Network (CNN)**
  - **Overview Forward and Backpropagation of MLP**
  - **CNN computation**
  - **CNN forward path calculation**
  - **MNIST TensorFlow code**
  - **CIFAT 10 example**
  - **AlexNet, number of parameters**
  - **CNN Backward Calculations**
  - **Homework exercises**

DNN Model

Applications

+

Data Ensemble + Input

+

It's all about

linear algebra calculations

DNN in particular

+

Algorithms, Software

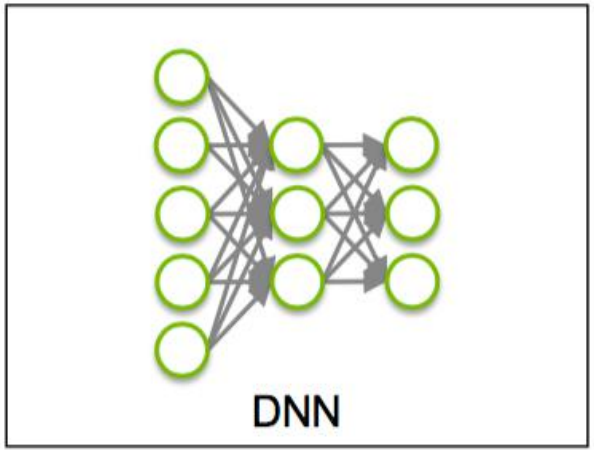
+

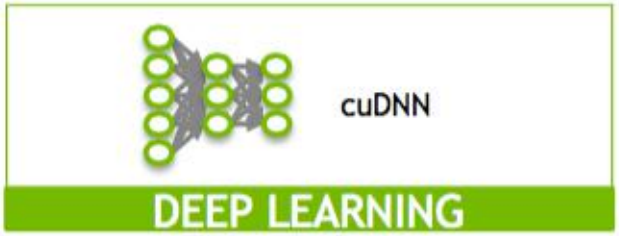
Output + Data Analysis

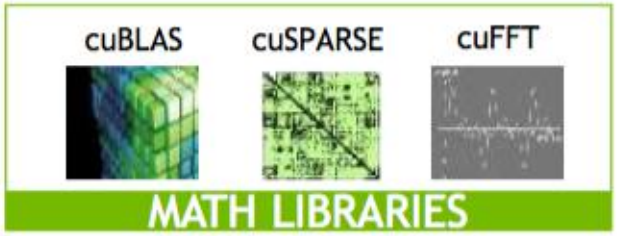
+

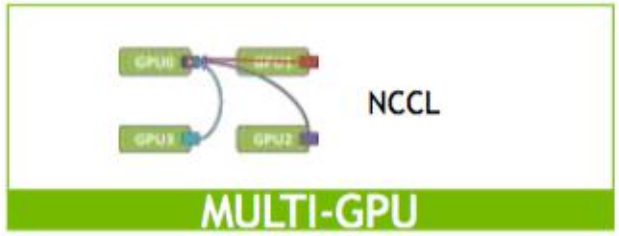
Hardware

Machine Learning – GPU acceleration









Source captured frin - Julie Bernauer – HPC Advisory Council Stanford Tutorial – 2017/02/07

Supervised learning is a type of machine learning which automate decision-making processes by generalizing from known examples. That is, the users provides the algorithm with pairs of inputs and desired outputs. The algorithm finds a way to produce the desired output given an input. In other words, we construct a model for the problem. The model is governed by some unknown parameters which we will learn from the data.

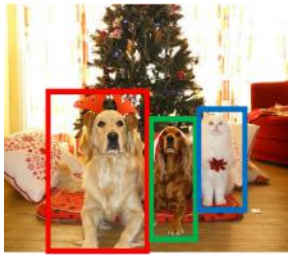
Supervised Learning Data:
(x, y) x is data, y is label

Goal: Learn a function to map x -> y

Examples:
Classification (discrete values),
regression (numerical values),

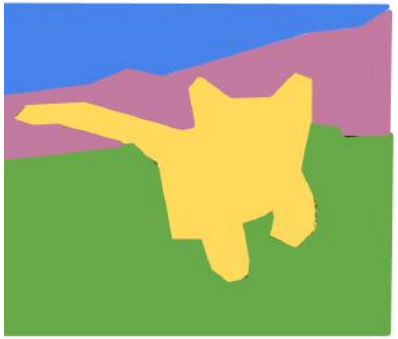
CAT

Classification



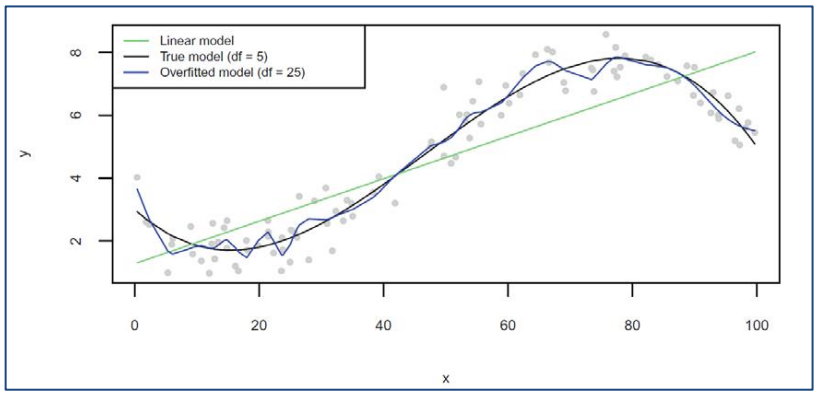
DOG, DOG, CAT

Object Detection



GRASS, CAT, TREE, SKY

Semantic Segmentation



Basic Ideas

Typical Neural Network – MLP

Convolutional Neural Network

STEP 1 : Model Definition

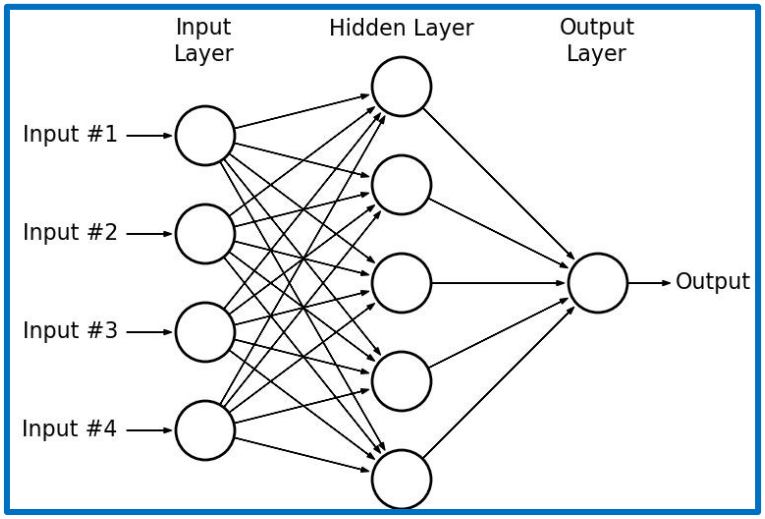
STEP 2 : Cost Function

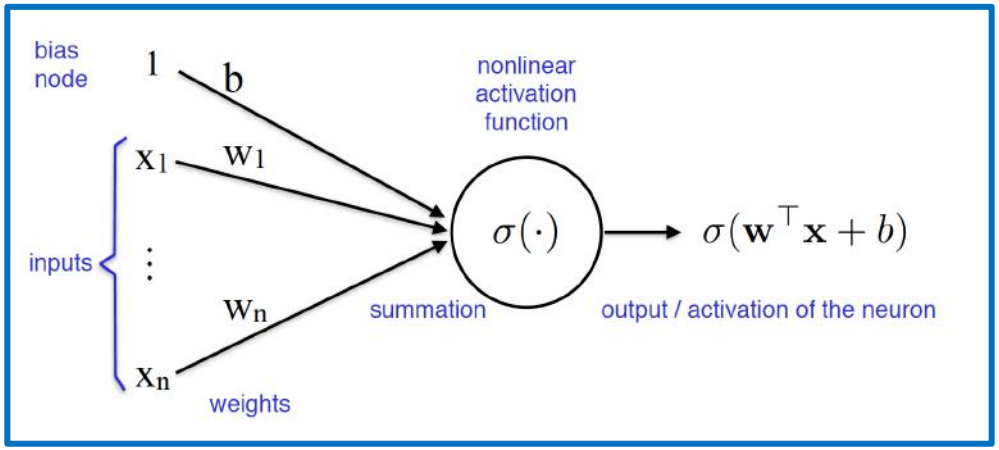
Step 3 : Optimization Scheme

Step 4 : Numerical Implementation (fitting)

Step 5 : Evaluation

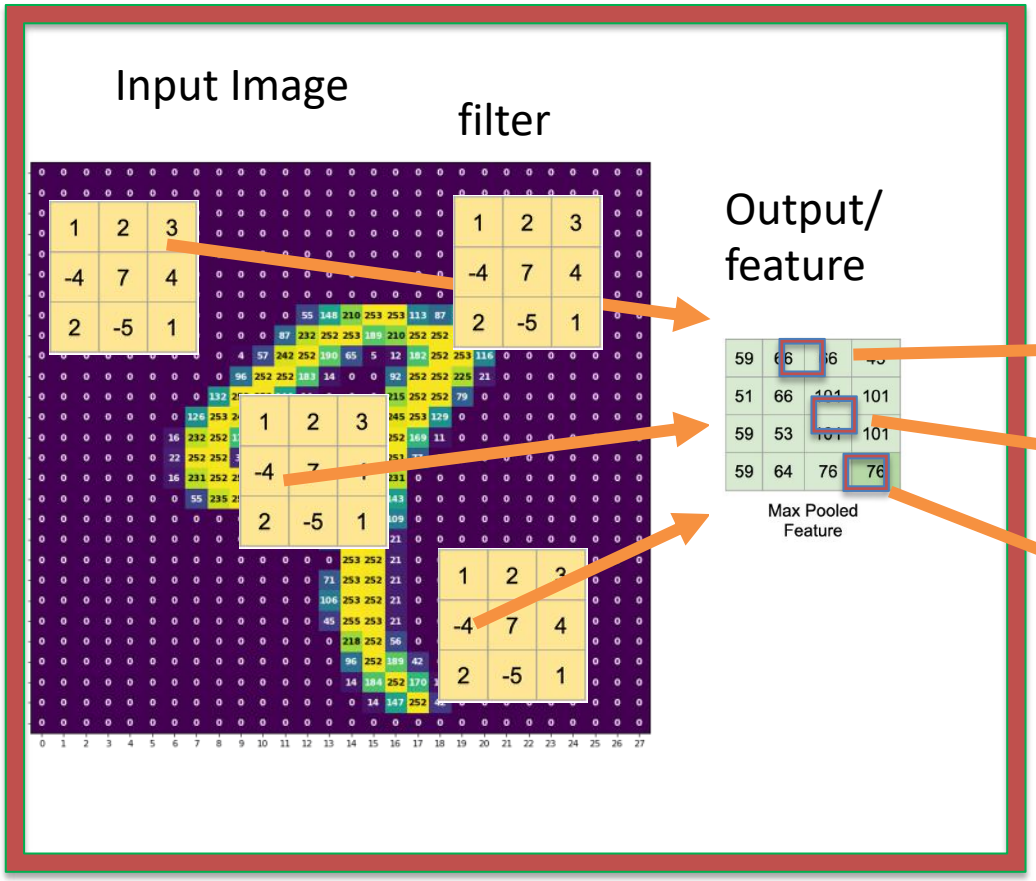
Parametric Model : NN Modeling

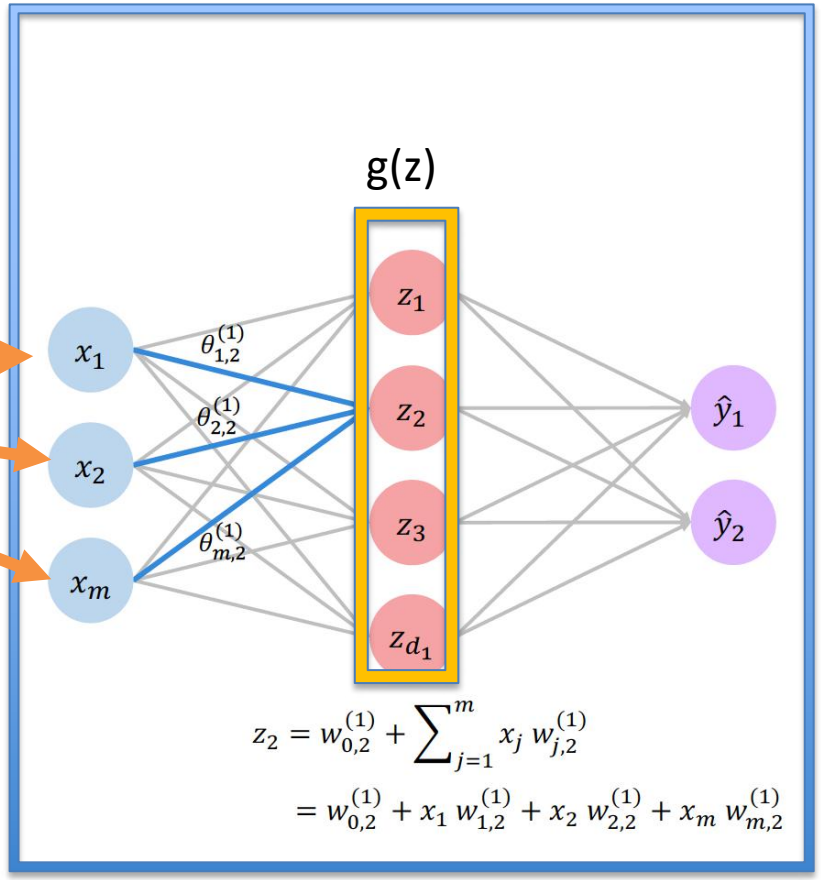




- ✓ A node in the neural network is a mathematical function or activation function which maps input to output values.
- ✓ Inputs represent a set of vectors containing weights (w) and bias (b). They are the sets of parameters to be determined.
- ✓ Many nodes form a **neural layer**, **links** connect layers together, defining a NN model.
- ✓ Activation function (f or σ), is generally a nonlinear data operator which facilitates identification of complex features.

Step 1: Parametric Model : Convolutional Neural Network (CNN)
Convolutional filter + Connected Neural Network





Convolutional filtering

Multilayer Perceptron

MLP : Neural Network

CNN : Convolutional NN

STEP 1 : Model Definition

STEP 2 : Cost Function

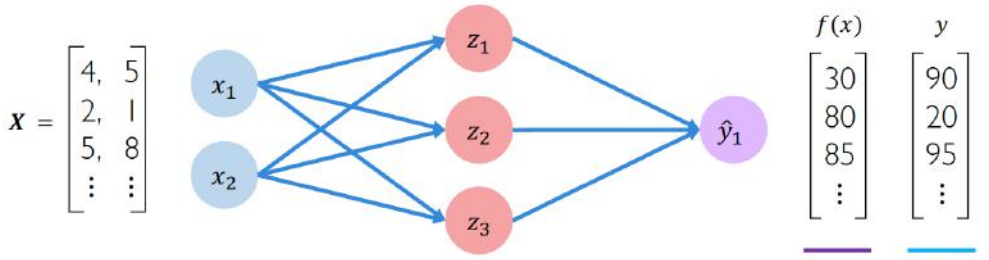
Step 3 : Optimization Scheme

Step 4 : Numerical Implementation

Step 5 : Evaluation

NN Cost Function (regression) : Mean Squared Error Loss

Mean squared error loss can be used with regression models that output continuous real numbers



$$J(W) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - f\left(x^{(i)}; W\right)\right)^2$$

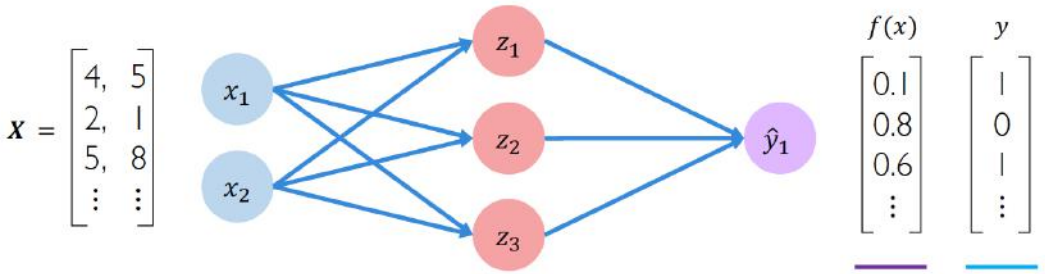
Actual Predicted

```
loss = tf.reduce_mean( tf.square(tf.subtract(model.y, model.pred) )
```

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

NN Cost Function (Classification) : Cross Entropy Loss

Cross entropy loss can be used with models that output a probability between 0 and 1



$$J(W) = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} \log\left(f\left(x^{(i)}; W\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - f\left(x^{(i)}; W\right)\right)$$

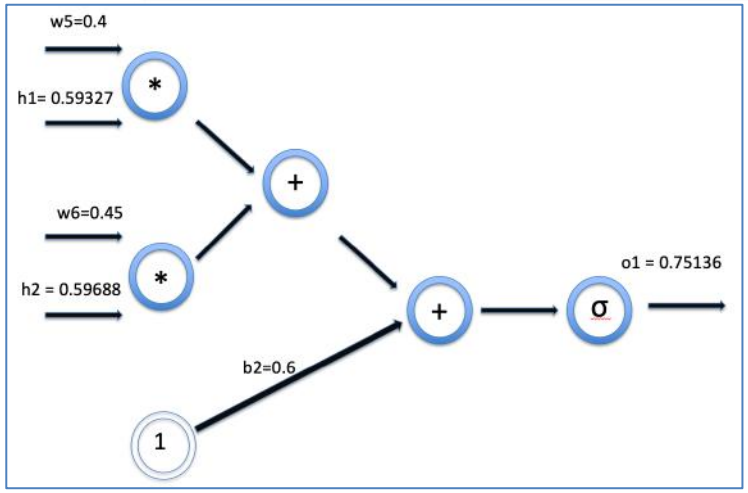
Actual Predicted Actual Predicted

```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(model.y, model.pred) )
```

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

target=0.01 | o1=0.75136 | Eo1=0.27481



Forward Path to Objective Function

target1 = 0.01

o1 = 0.75136

Cost f

Eo1 = 0.27481

+

ET = 0.298371

grad ET = 1.0

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Etotal = Eo1 + Eo2 = 0.298371109

Eo2 = 0.02356

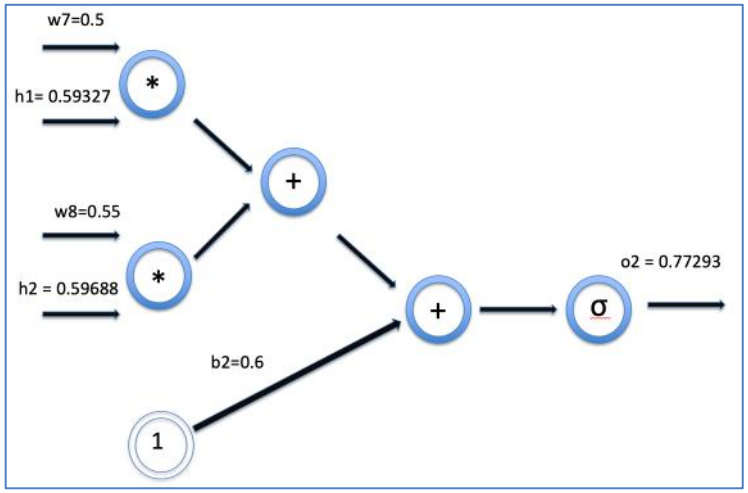
o2 = 0.77293

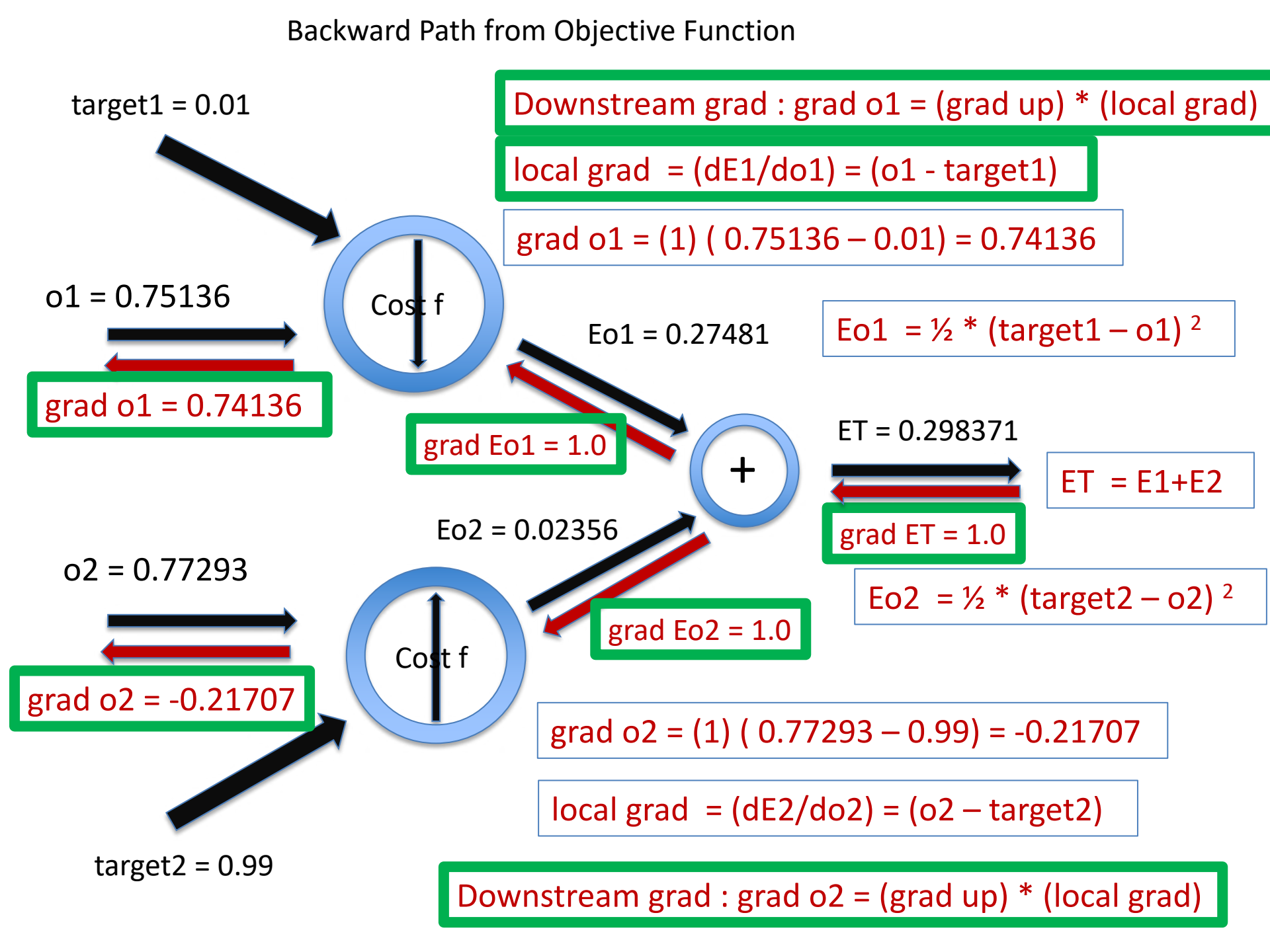
Cost f

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

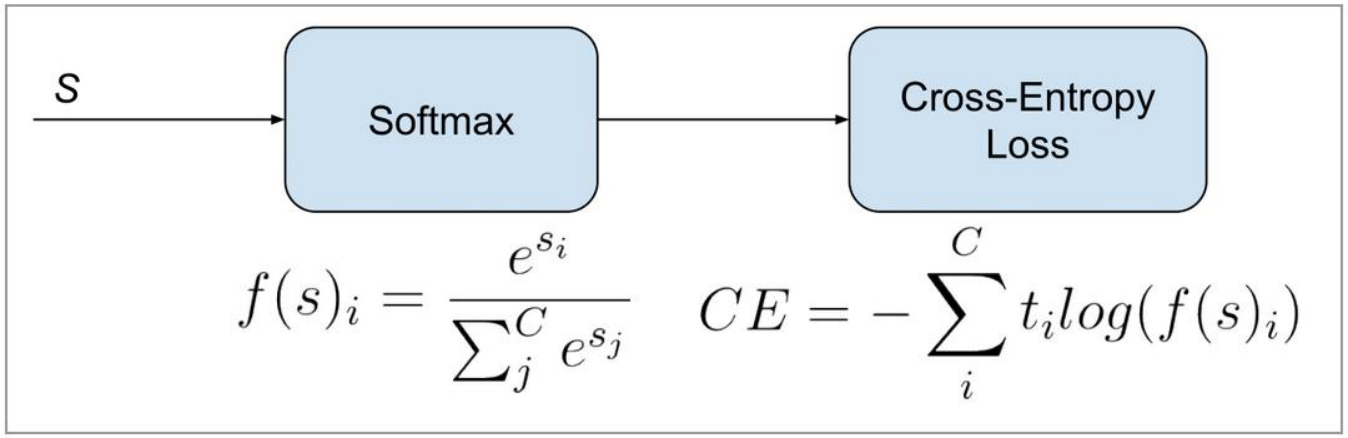
target2 = 0.99

target=0.99 | o2=0.77293 | Eo2=0.02356



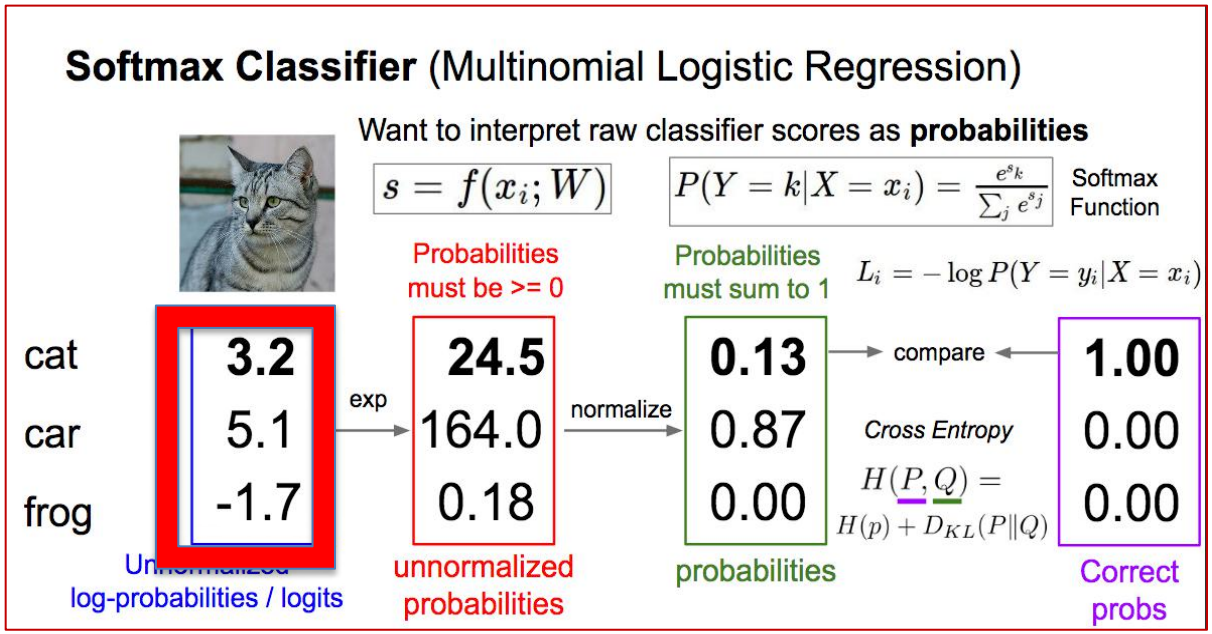

Backward Path from Objective Function
target1 = 0.01
Downstream grad : grad o1 = (grad up) * (local grad)
local grad = (dE1/do1) = (o1 - target1)
grad o1 = (1) (0.75136 – 0.01) = 0.74136
o1 = 0.75136
Cost f
Eo1 = 0.27481
Eo1 = ½ * (target1 – o1) 2
grad o1 = 0.74136
grad Eo1 = 1.0
+
ET = 0.298371
ET = E1+E2
grad ET = 1.0
Eo2 = 0.02356
Eo2 = ½ * (target2 – o2) 2
o2 = 0.77293
Cost f
grad Eo2 = 1.0
grad o2 = -0.21707
grad o2 = (1) (0.77293 – 0.99) = -0.21707
local grad = (dE2/do2) = (o2 – target2)
target2 = 0.99
Downstream grad : grad o2 = (grad up) * (local grad)

Categorical (Softmax) Cross Entropy Loss (Statistical Learning)



T = 1

ti and si are the groundtruth (label) and the computed score for each class i in C.



e^3.2 = 24.5

e^5.1 = 164.0

e^-1.7 = 0.18

188.68

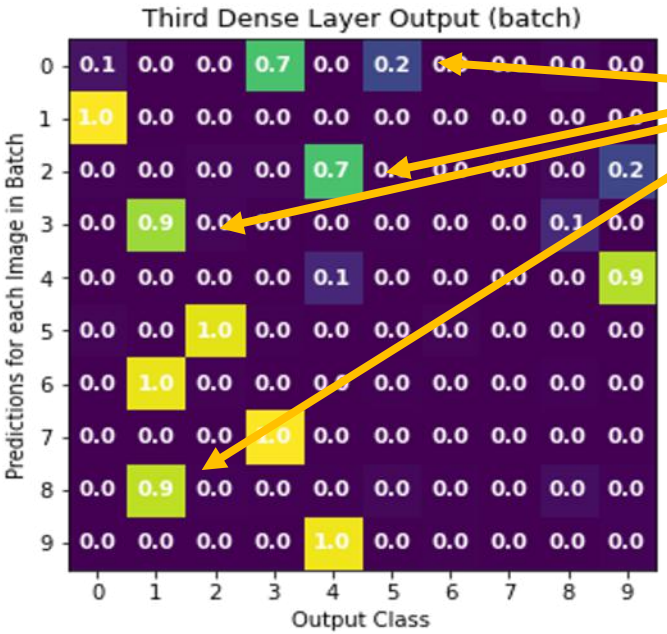
Software = 24.5 / 188.68 = 0.13

Software = 164 / 188.68 = 0.87

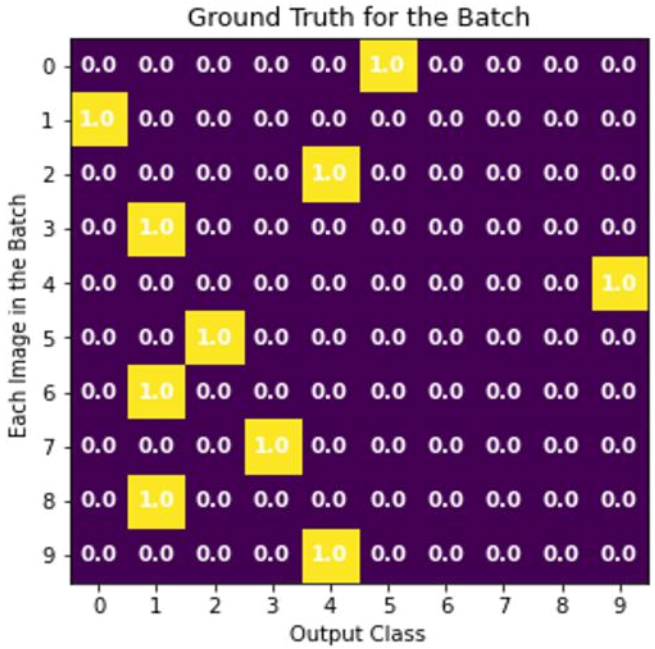
Software = -1.7 / 188.68 = 0.00

https://gombru.github.io/2018/05/23/cross_entropy_loss/

Evaluating the Error



Z

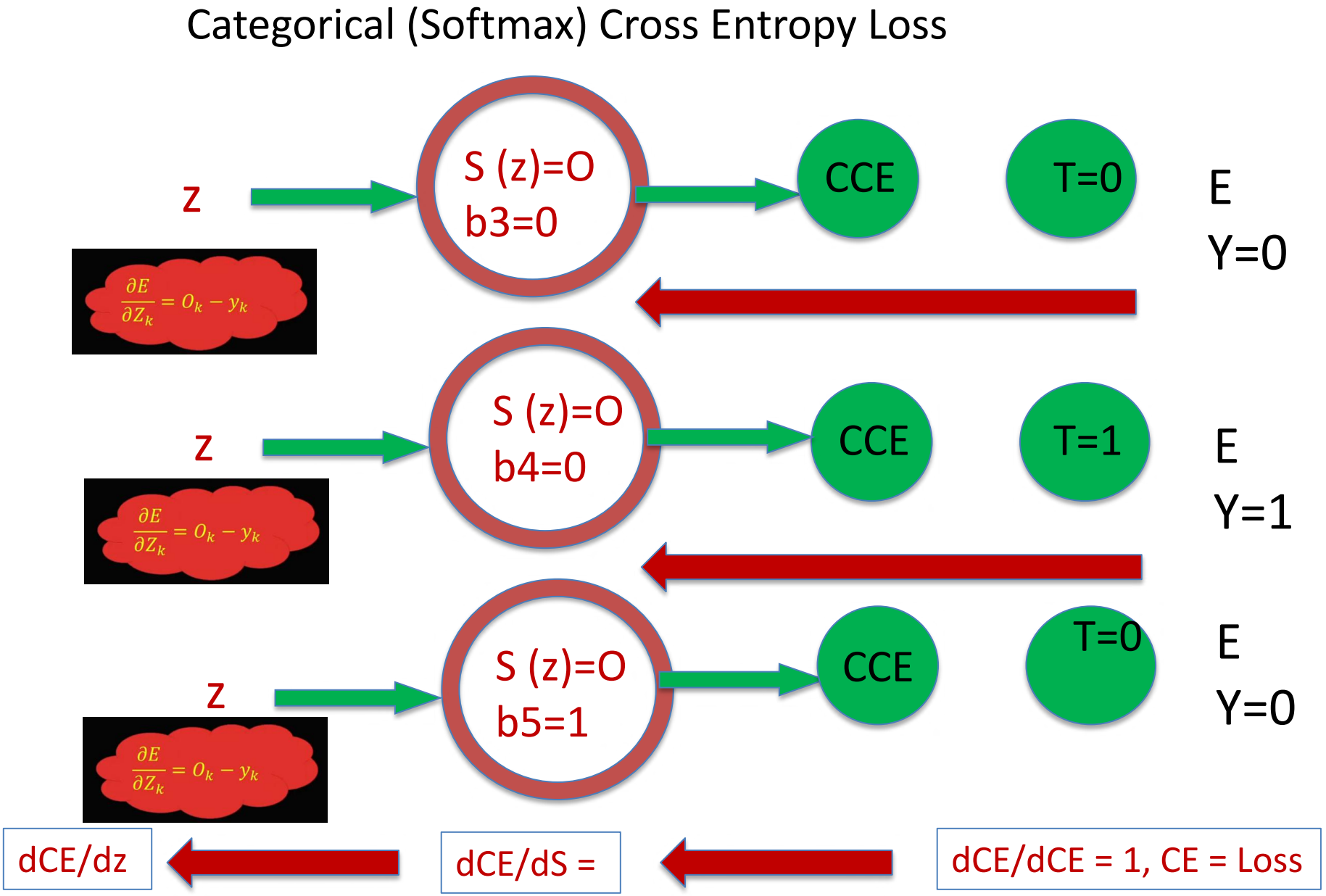


Categorical Cross Entropy Loss

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

L (i) = - log0.2 = 1.61	Image 0 = 5
- log1.0 = 0	Image 1 = 0
- log0.7 = 0.356	Image 2 = 4
- log0.9 = 0.105	Image 3 = 1
- log0.9 = 0.105	Image 4 = 9
- log1.0 = 0	Image 5 = 2
- log1.0 = 0	Image 6 = 1
- log1.0 = 0	Image 7 = 3
- log0.9 = 0.105	Image 8 = 2
- log1.0 = 0	Image 9 = 4

Exercise : Represent the following NN model as a computational graph and build a python code to update w and b



MLP : Neural Network

CNN : Convolutional NN

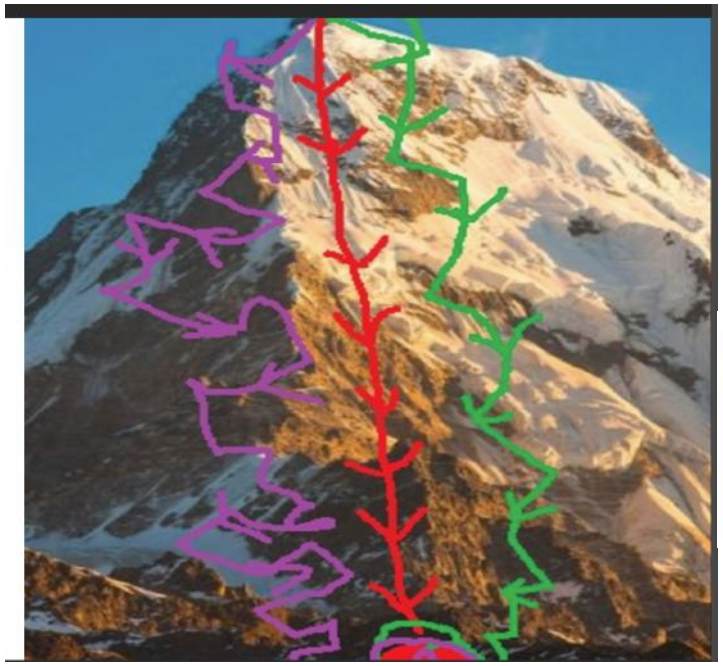
STEP 1 : Model Definition

STEP 2 : Cost Function

Step 3 : Optimization Scheme

Step 4 : Numerical Implementation

Step 5 : Evaluation



Optimization Schemes: Gradient Descent : Simple Story

Batch Gradient Descent
Mini-batch Gradient Descent
Stochastic Gradient Descent

Derivative is the slope of the tangent line

$\frac{d}{dx}x^2 = 2x$

$x = 1$

The slope of the tangent line is 2

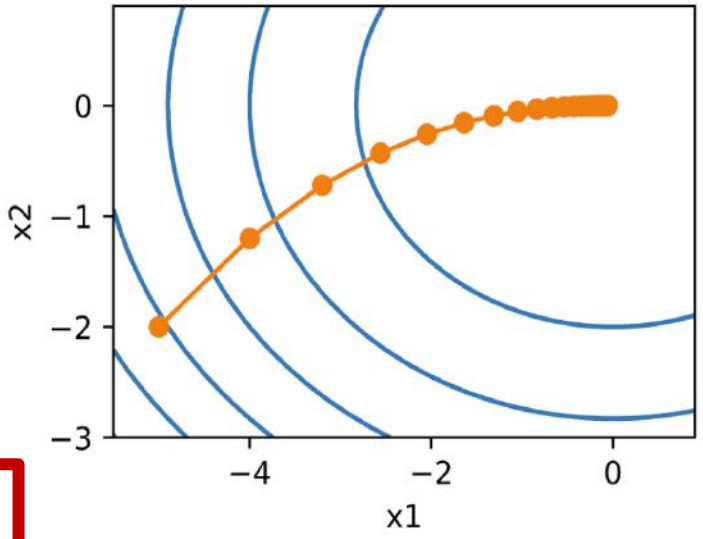
Algorithm

- Choose initial $\mathbf{x}_0$ (X = W, weight)
- At time $t = 1, \ldots, T$

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \eta \nabla f(\mathbf{x}_{t-1})$$

- η is called learning rate

W+ = W – (learning rate) * dJ/dw



DNN Forward Backward Calculation : Weights and bias

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$

```
weights = tf.random_normal(shape, stddev=sigma)
```

2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$

```
grads = tf.gradients(ys=loss, xs=weights)
```

4. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$

```
weights_new = weights.assign(weights - lr * grads)
```

5. Return weights

In one epoch

1. Pick a mini-batch
2. Feed it to Neural Network
3. Forward path calculation
4. Calculate the mean gradient of the mini-batch
5. Use the mean gradient calculated in step 4 to update the weights
6. Repeat steps 1–5 for the mini-batches we created

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C,$$

$$w_k \to w_k' = w_k - \eta \frac{\partial C}{\partial w_k}$$

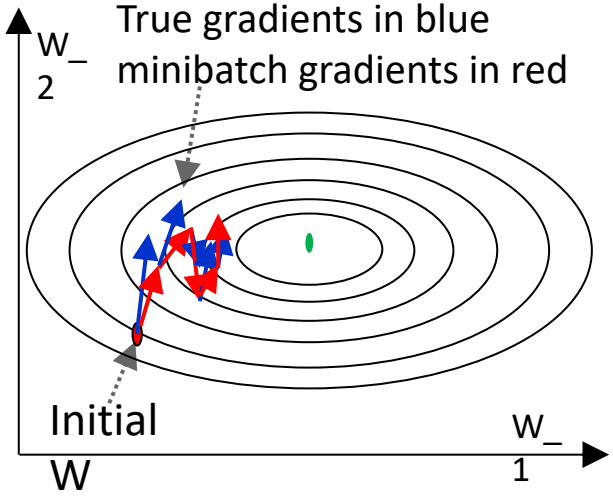
$$b_l \to b_l' = b_l - \eta \frac{\partial C}{\partial b_l}.$$

Optimization: Batch SGD

Instead of computing a gradient across the entire dataset with size N, divides the dataset into a fixed-size subset ("minibatch") with size m, and computes the gradient for each update on this smaller batch:

(N is the dataset size, m is the minibatch size)

$$\mathbf{g} = \frac{1}{m'}\sum_{i=1}^{m'} \nabla_{\boldsymbol{\theta}} E_i(\mathbf{x}^{(n_i)}, y^{(n_i)}, \boldsymbol{\theta}),$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\mathbf{g},$$



Epoch step

Given: training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$
Initialize all $\Theta^{(l)}$ randomly (NOT to 0!)
Loop // each iteration is called an epoch
  Loop // each iteration is a mini-batch
    Set $\Delta^{(l)}_{i,j} = 0 \quad \forall l, i, j$ (Used to accumulate gradient)
    Sample m training instances $\mathcal{X} = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_m, y'_m)\}$ without replacement
    For each instance in $\mathcal{X}$, $(\mathbf{x}_k, y_k)$:
      Set $\mathbf{a}^{(1)} = \mathbf{x}_k$
      Compute $\{\mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(L)}\}$ via forward propagation
      Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_k$
      Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(2)}\}$
      Compute gradients $\Delta^{(l)}_{ij} = \Delta^{(l)}_{ij} + a^{(l)}_j \delta^{(l+1)}_i$
    Compute mini-batch regularized gradient $D^{(l)}_{ij} = \begin{cases} \frac{1}{m}\Delta^{(l)}_{ij} + \lambda\Theta^{(l)}_{ij} & \text{if } j \neq 0 \\ \frac{1}{m}\Delta^{(l)}_{ij} & \text{otherwise} \end{cases}$
    Update weights via gradient step $\Theta^{(l)}_{ij} = \Theta^{(l)}_{ij} - \alpha D^{(l)}_{ij}$
  Until all training instances are seen
Until weights converge or max #epochs is reached

Batch iteration

MLP : Neural Network

CNN : Convolutional NN

STEP 1 : Model Definition

STEP 2 : Cost Function

Step 3 : Optimization Scheme

Step 4 : Numerical Implementation

Step 5 : Evaluation

Implementation : Training

"Training a neural network"

What does it mean?

What does the network train?

minimize error of the computed values against the labeled values (loss function)

What are the training parameters

W and b

How does it work?

optimization based on gradient descent algorithm

go back and forth in the NN model to adjust for w and b => need derivatives w.r.t. w and b

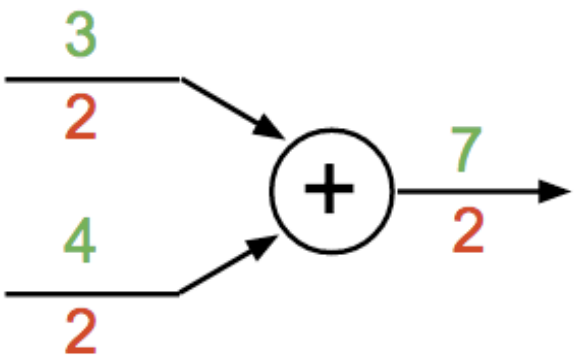
Go through forward calculation

training with backpropagation

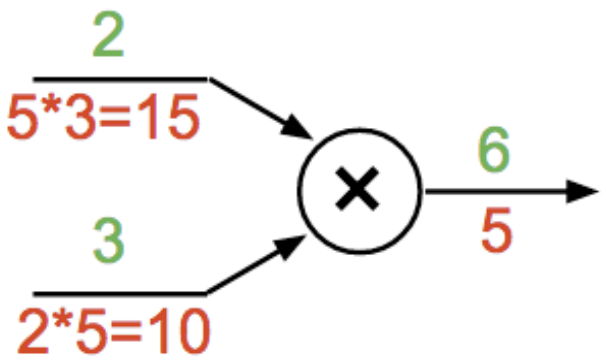
Forward computational Operators

Forward path (Green) : backward path

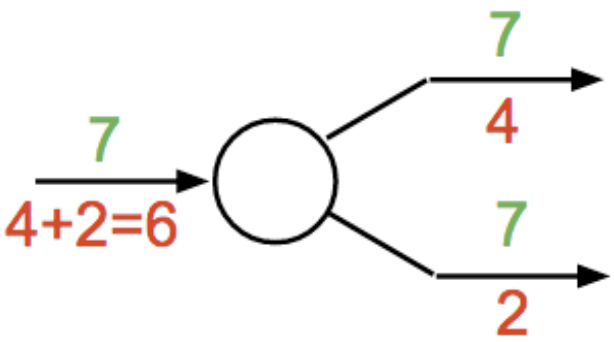
add gate: gradient distributor



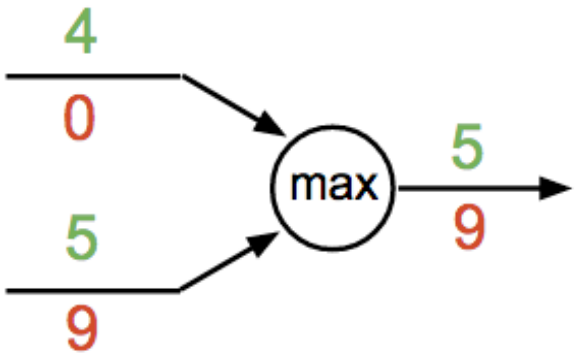
mul gate: "swap multiplier"



copy gate: gradient adder



max gate: gradient router

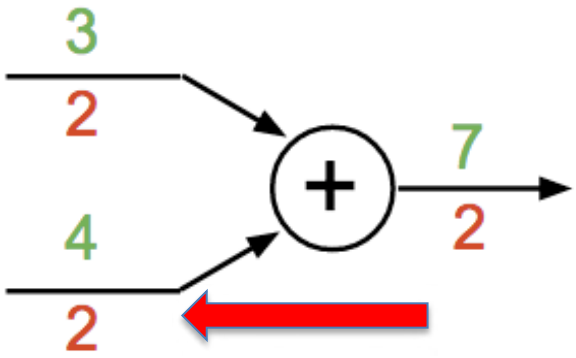


http://cs231n.stanford.edu/slides/2020/lecture_4.pdf

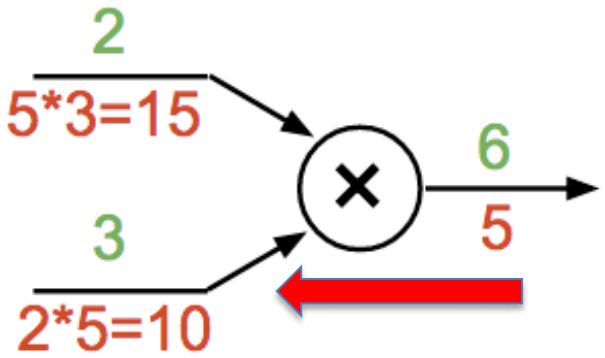
Forward and Backpropagation Operators

Forward path : backward path

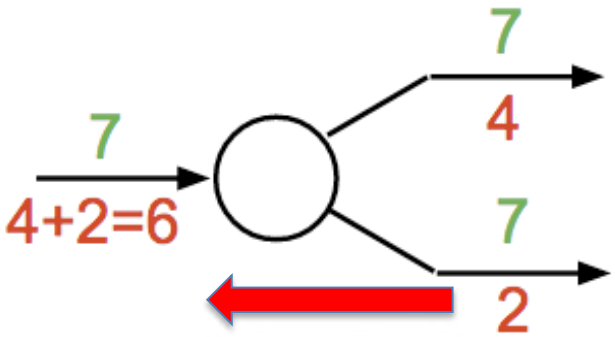
add gate: gradient distributor



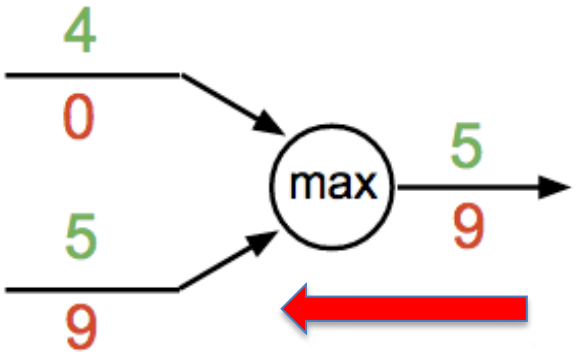
mul gate: "swap multiplier"



copy gate: gradient adder



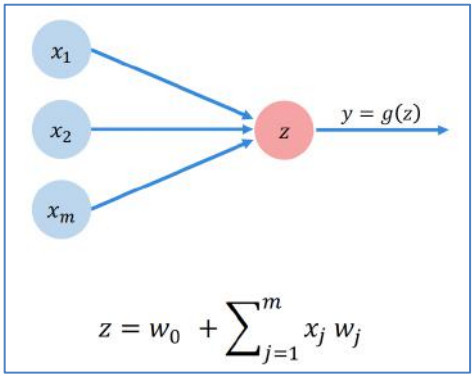
max gate: gradient router



http://cs231n.stanford.edu/slides/2020/lecture_4.pdf

Forward and backward computation operators

Z = (x1*w1 + x2*w2) + b = (-1.0*-2.0 + -2.0*3.0) + -3.0 = 4.0 – 3.0 = 1.0



Sigmoid function

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

Y = σ (z) = 1 / (1 + e $^{-Z}$)
= 1 / (1 + e $^{-1}$) = 0.731

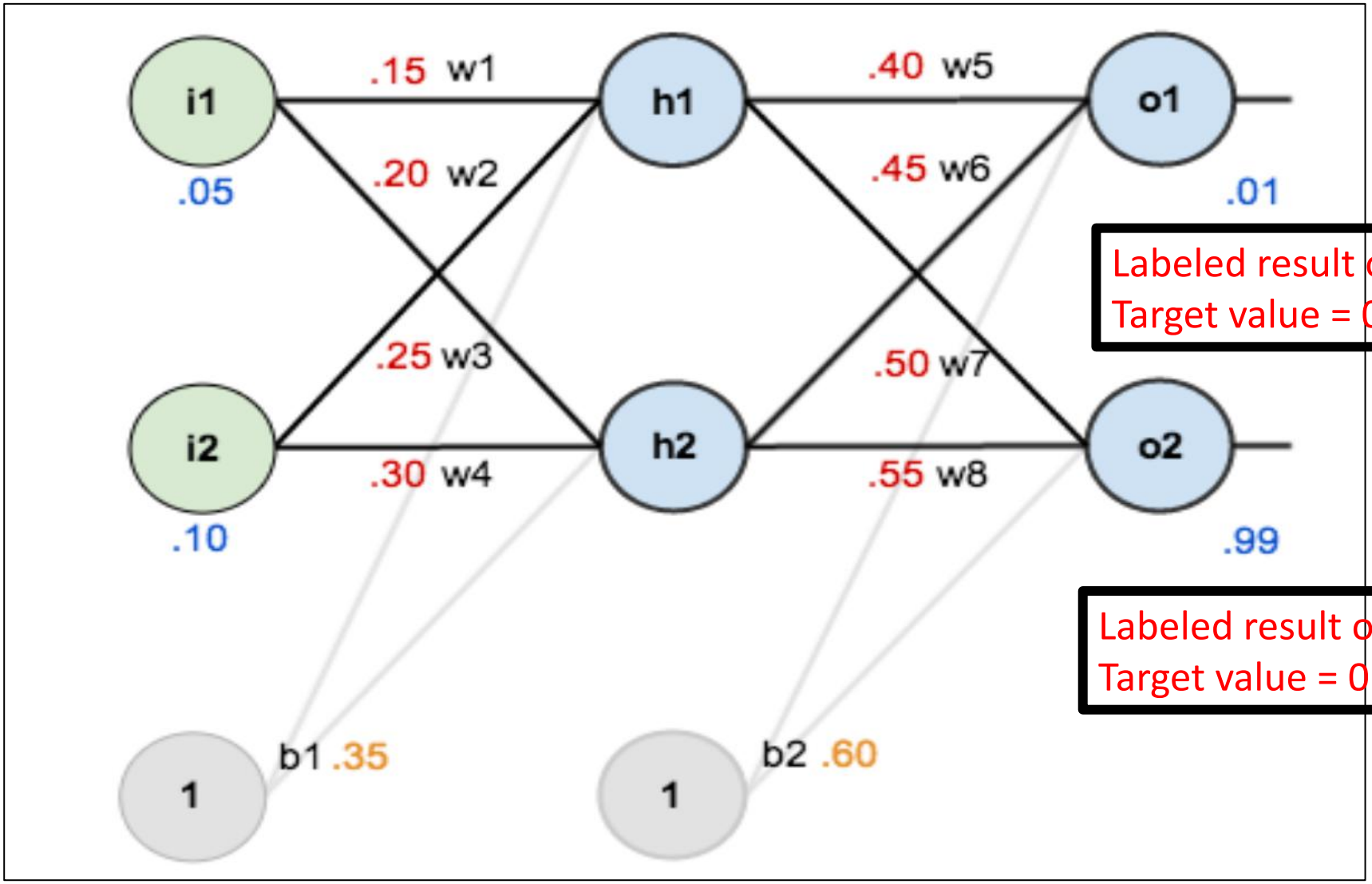
Z = (x1*w1 + x2*w2) + b

multiple operator :
downstream gradient =
upstream gradient * input value

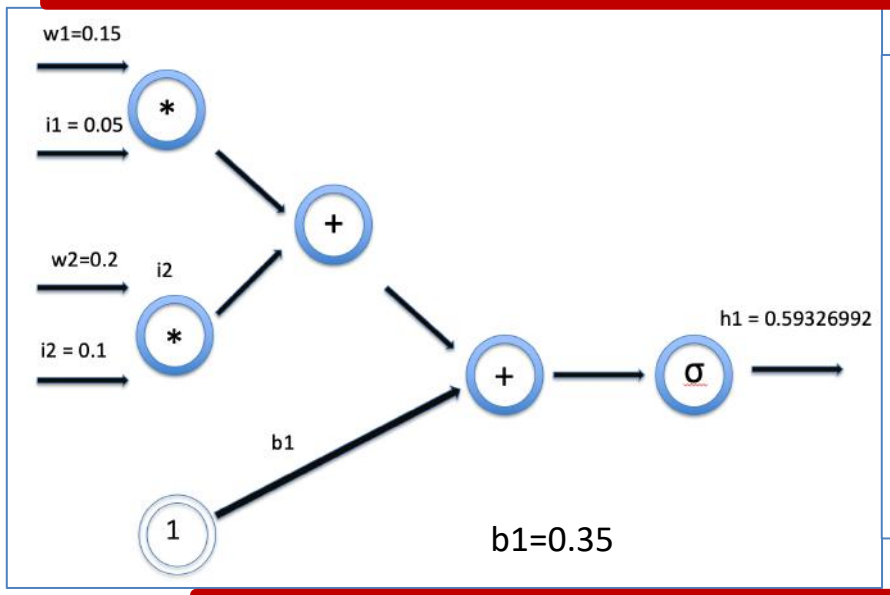
add operator :
Downstream gradient =
upstream gradient

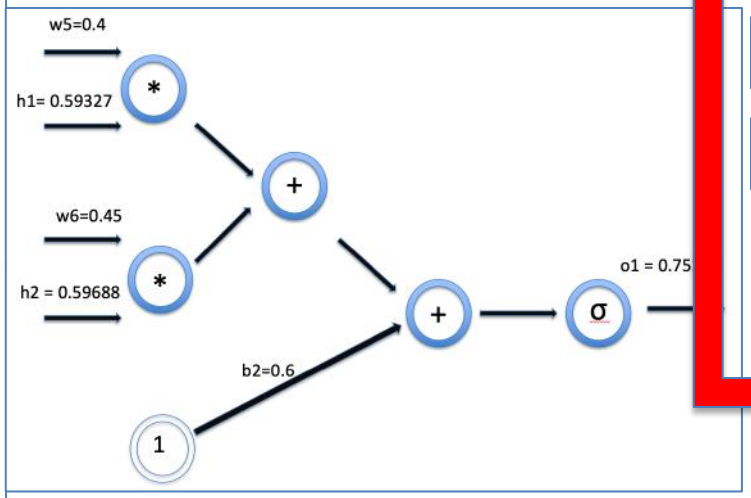
function operator :
downstream gradient =
Upstream gradient * local function gradient

DNN Example


i1
i2
h1
h2
o1
o2
1
1
.15 w1
.20 w2
.25 w3
.30 w4
.40 w5
.45 w6
.50 w7
.55 w8
b1 .35
b2 .60
.05
.10
.01
.99
Labeled result of out1
Target value = 0.01
Labeled result of out2
Target value = 0.99

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$





target=0.01

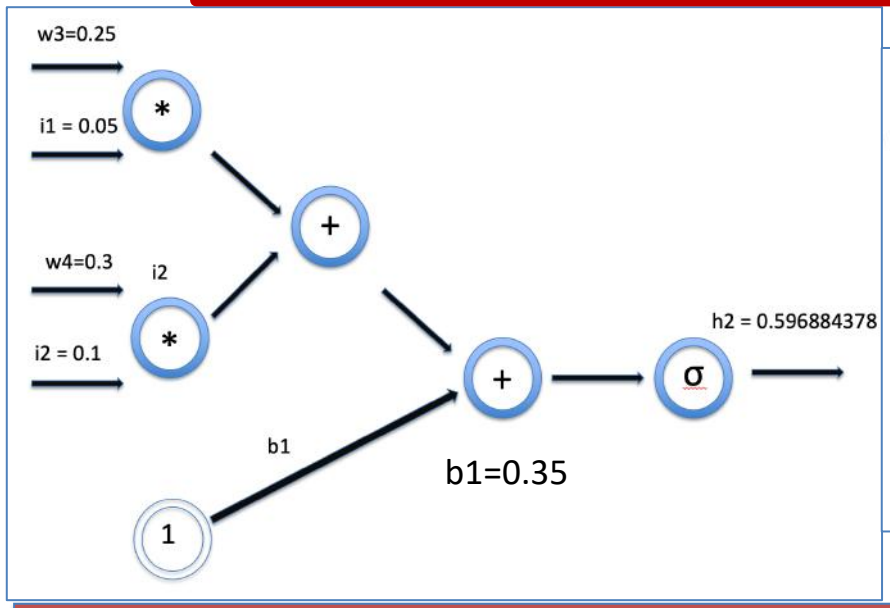
o1=0.75136

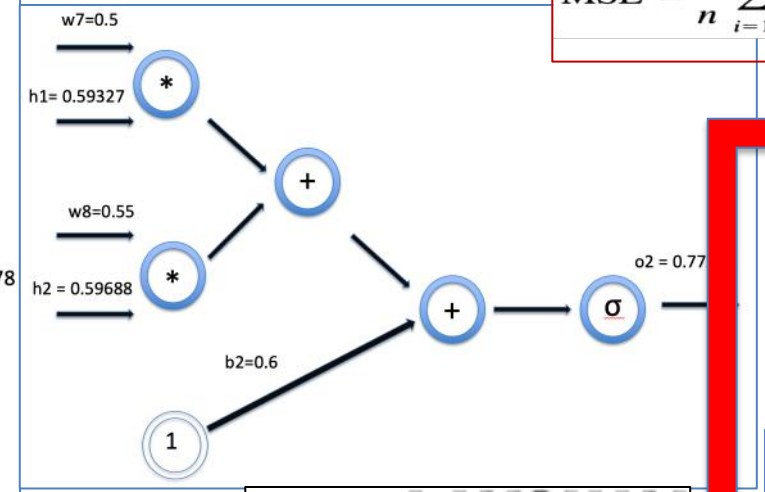
Eo1=0.27481

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Etotal = Eo1 + Eo2 = 0.298371109

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$





$$E_{o2} = 0.023560026$$

Eo2=0.02356

o2=0.77293

target=0.99

```
# e constant
e = 2.7182818284
# initial values
i1 = 0.05
i2 = 0.10
# initial weights
w1 = 0.15
w2 = 0.20
w3 = 0.25
w4 = 0.30
w5 = 0.40
w6 = 0.45
w7 = 0.50
w8 = 0.55
# bias
b1 = 0.35
b2 = 0.60
# targets
To1 = 0.01
To2 = 0.99
```

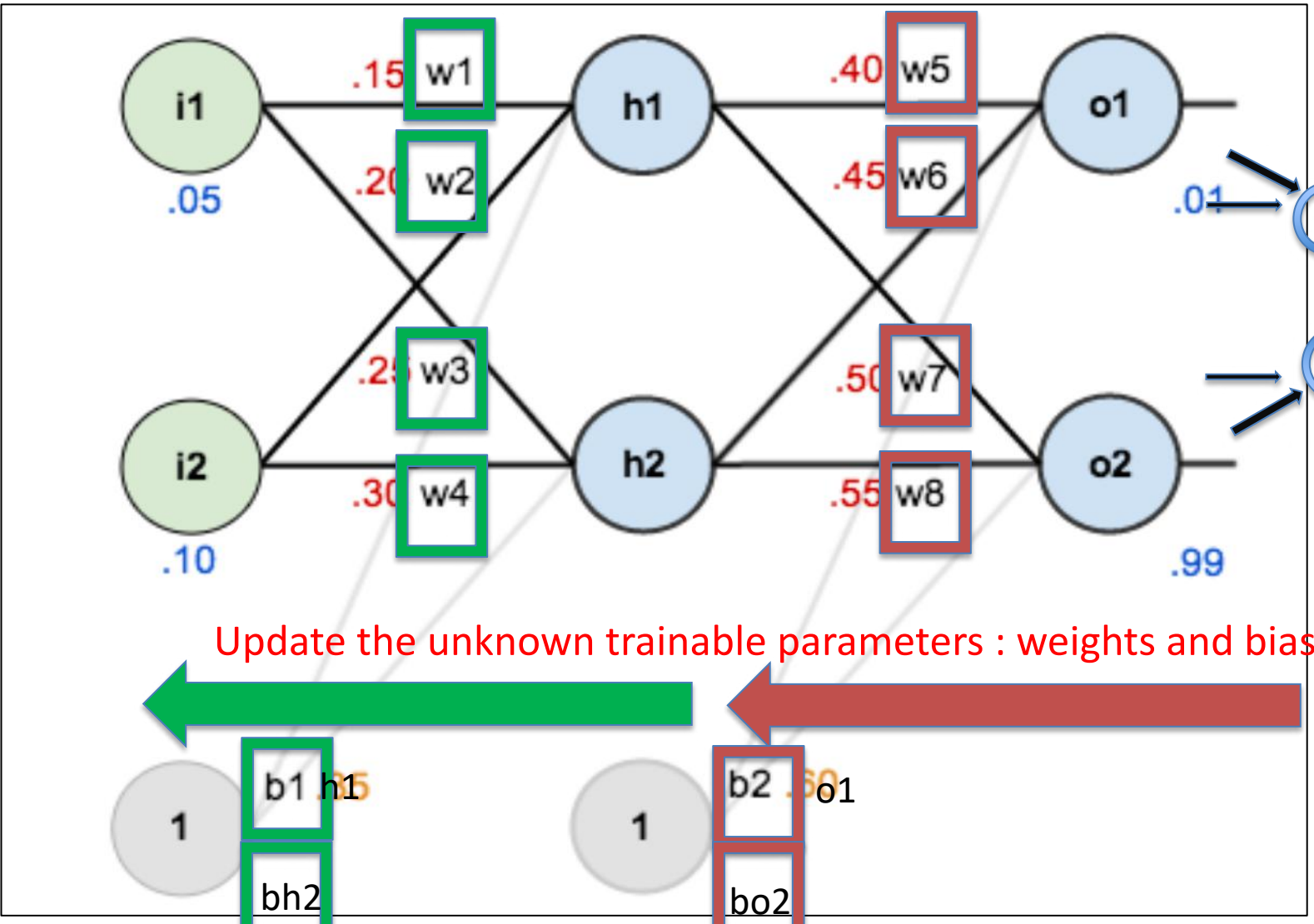
```
# forward propagation
h1 = 1/(1+e**(-(w1*i1 + w2*i2+b1)))
print("h1: " + str(h1))

h2 = 1/(1+e**(-(w3*i1 + w4*i2+b1)))
print("h2: " + str(h2))

o1 = 1/(1+e**(-(w5*h1 + w6*h2+b2)))
print("o1: " + str(o1))

o2 = 1/(1+e**(-(w7*h1 + w8*h2+b2)))
print("o2: " + str(o2))
```

```
h1: 0.5932699921052087 h2: 0.5968843782577157
o1: 0.7513650695475076 o2: 0.772928465316421
```

```
# Error
Eo1 = 0.5*(To1-o1)**2
print("Error o1: " + str(Eo1))
Eo2 = 0.5*(To2-o2)**2
print("Error o2: " + str(Eo2))

E = Eo1 + Eo2
print("Total Error: " + str(E))
```

```
Error o1: 0.2748110831725904
Error o2: 0.023560025584942117
Total Error: 0.29837110875753253
```

Multilayer Perceptron : Backward Path Calculation



```
# Backpropagation
n = 0.5

grad_o1 = o1 - To1
grad_o2 = o2 - To2

print("grad_o1: "+str(grad_o1))
print("grad_o2: "+str(grad_o2))

grad_d1 = (grad_o1)*o1*(1-o1)
grad_d2 = (grad_o2)*o2*(1-o2)

print("grad_d1: "+str(grad_d1))
print("grad_d2: "+str(grad_d2))

grad_w5 = grad_d1*h1
print("grad_w5: "+str(grad_w5))

w5_final = w5 - n*grad_w5

grad_w6 = grad_d1*h2
print("grad_w6: "+str(grad_w6))
w6_final = w6 - n*grad_w6

grad_w7 = grad_d2*h1
print("grad_w7: "+str(grad_w7))
w7_final = w7 - n*grad_w7

grad_w8 = grad_d2*h2
print("grad_w8: "+str(grad_w8))
w8_final = w8 - n*grad_w8
```

```
grad_h1 = w5*grad_d1 + w7*grad_d2
print("grad_h1: "+str(grad_h1))
grad_d11 = grad_h1*h1*(1-h1)
print("grad_d11: "+str(grad_d11))

grad_w1 = grad_d11*i1
print("grad_w1: "+str(grad_w1))
w1_final = w1 - n*grad_w1

grad_w2 = grad_d11*i1
print("grad_w2: "+str(grad_w2))
w2_final = w2 - n*grad_w2

grad_h2 = w6*grad_d1 + w8*grad_d2
print("grad_h2: "+str(grad_h2))
grad_d22 = grad_h2*h2*(1-h2)
print("grad_d22: "+str(grad_d22))

grad_w3 = grad_d22*i1
print("grad_w3: "+str(grad_w3))
w3_final = w3 - n*grad_w3

grad_w4 = grad_d22*i2
print("grad_w4: "+str(grad_w4))
w4_final = w4 - n*grad_w4
```

Exercise: Write the backpropagation code using simply python

```
print("w1+: "+str(w1_final))
print("w2+: "+str(w2_final))
print("w3+: "+str(w3_final))
print("w4+: "+str(w4_final))
print("w5+: "+str(w5_final))
print("w6+: "+str(w6_final))
print("w7+: "+str(w7_final))
print("w8+: "+str(w8_final))
```

w1+: 0.1497807161327648
w2+: 0.19978071613276482
w3+: 0.249751143632371 64
w4+: 0.29950228726474326
w5+: 0.35891647971775653
w6+: 0.4086661860761087
w7+: 0.5113012702391395
w8+: 0.5613701211083925

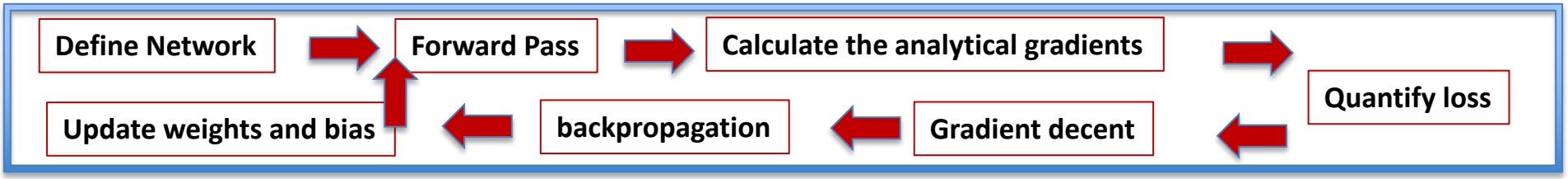
grad_o1: 0.7413650695475076
grad_o2: -0.2170715346835789 8
grad_d1: 0.1384985616294507 6
grad_d2: -0.0380982365180384 4
grad_w5: 0.0821670405644870 1
grad_w6: 0.0826676278477826 3
grad_w7: -0.0226025404782790 4
grad_w8: -0.0227402422167847 7
grad_h1: 0.0363530639276108 6
grad_d11: 0.008771354689407 79
grad_w1: 0.0004385677344703895
grad_w2: 0.0004385677344703895
grad_h2: 0.041370322648331 71
grad_d22: 0.009954254705134271
grad_w3: 0.0004977127352567136
grad_w4: 0.009954254705134271

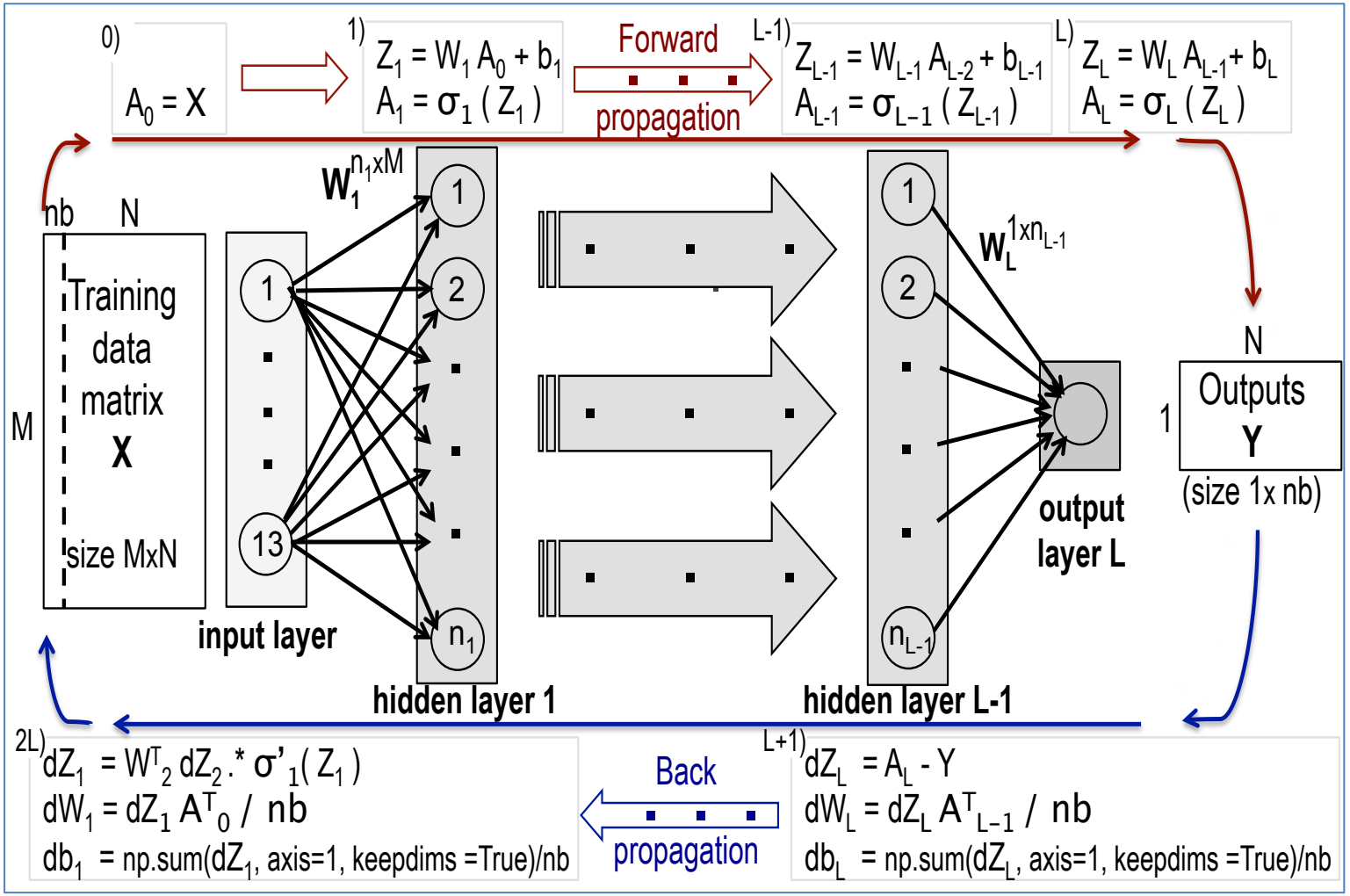
```
from sklearn.neural_network import MLPRegressor
```

```
from sklearn.neural_network import MLPClassifier
```

Summary : Flow of NN

```
model.fit(x_train, y_train, epochs=3, batch_size=10)
```





Training

"Training a neural network"

What does it mean?

What does the network train?

minimize error of the computed values against the labeled values (loss function)

What are the training parameters

W, b ,**and values of the filters (Wf)**

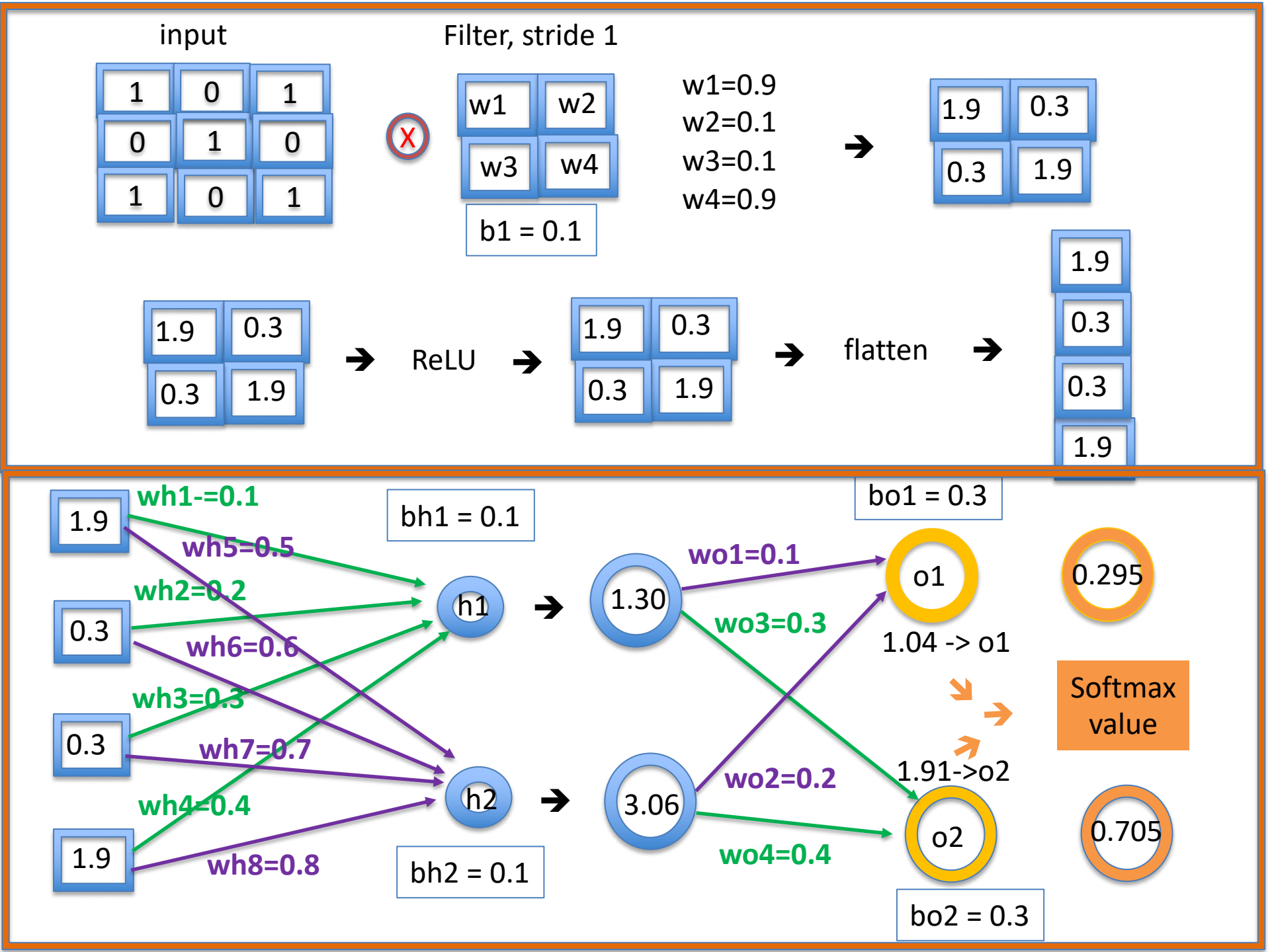
How does it work?

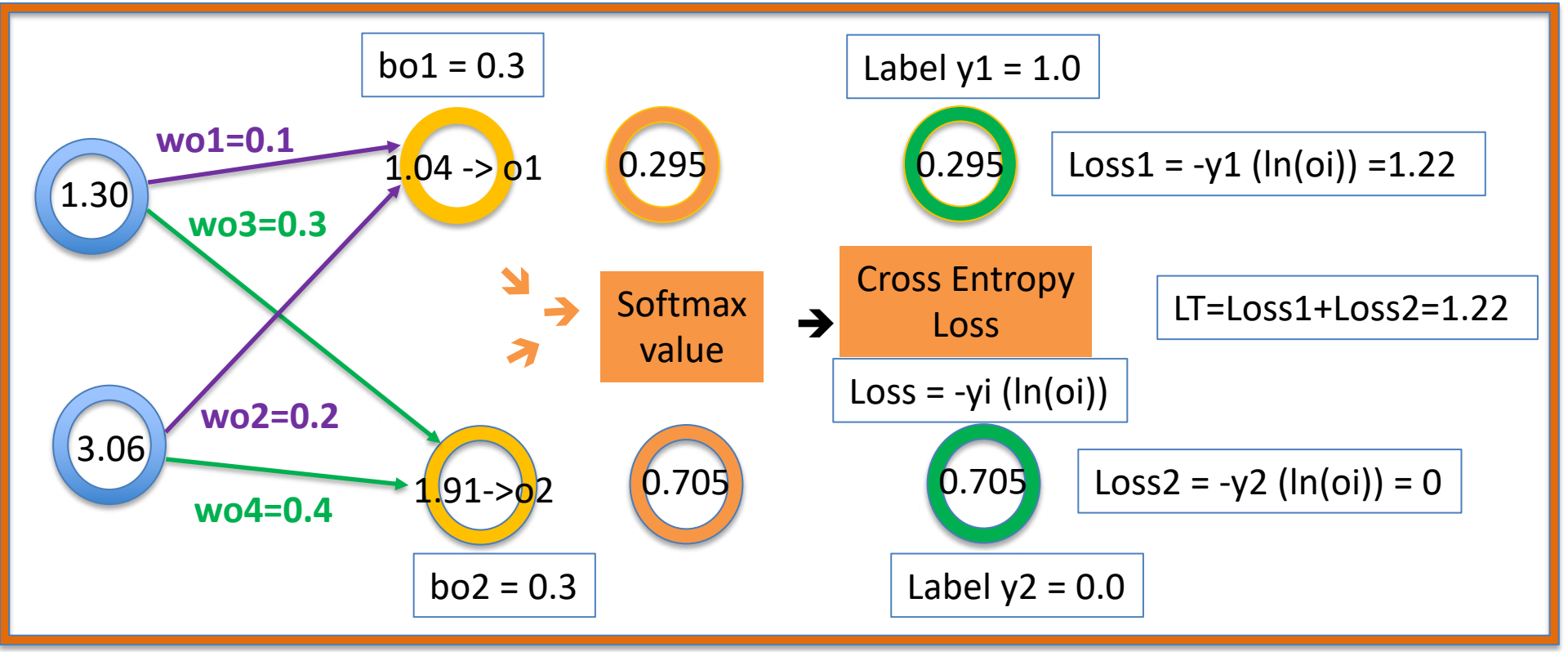
optimization based on gradient descent algorithm

go back and forth in the NN model to adjust for w and b => need derivatives w.r.t. parameters

Go through forward calculation

training with backpropagation


input
Filter, stride 1
1 0 1
0 1 0
1 0 1
X
w1 w2
w3 w4
b1 = 0.1
w1=0.9
w2=0.1
w3=0.1
w4=0.9
1.9 0.3
0.3 1.9
1.9 0.3
0.3 1.9
ReLU
1.9 0.3
0.3 1.9
flatten
1.9
0.3
0.3
1.9
1.9
wh1-=0.1
wh5=0.5
wh2=0.2
wh6=0.6
wh3=0.3
wh7=0.7
wh4=0.4
wh8=0.8
0.3
0.3
1.9
bh1 = 0.1
h1
h2
bh2 = 0.1
1.30
3.06
wo1=0.1
wo3=0.3
wo2=0.2
wo4=0.4
bo1 = 0.3
o1
o2
bo2 = 0.3
1.04 -> o1
1.91->o2
Softmax value
0.295
0.705



CNN : Backward Path (derivatives)

Learning rate = n = 0.5

$$\frac{\partial S(z_i)}{\partial z_j} = \begin{cases} S(z_i) \times (1 - S(z_i)) & \text{if } i = j \\ -S(z_i) \times S(z_j) & \text{if } i \neq j \end{cases}$$

$$\frac{\partial E}{\partial Z_k} = O_k - y_k$$

weights bias

w+ = w – n * (dE/dw)
dE/dw = (dE/dS) * (dS/dw)

Softmax value

dL/dS

Cross Entropy Loss (L)

dL/dL = 1

MLP : Neural Network

CNN : Convolutional NN

STEP 1 : Model Definition

STEP 2 : Cost Function

Step 3 : Optimization Scheme

Step 4 : Numerical Implementation

TensorFlow MNIST MLP & CNN Examples

MNIST Example (28x28 pixels)

Handwritten digits in the MNIST dataset are 28x28 pixel grayscale images. The simplest approach for classifying them is to use the 28x28=784 pixels as inputs for a 1-layer neural network.

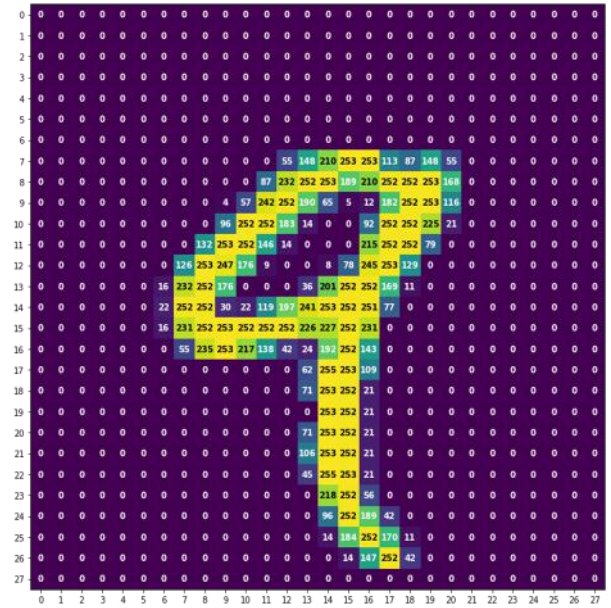
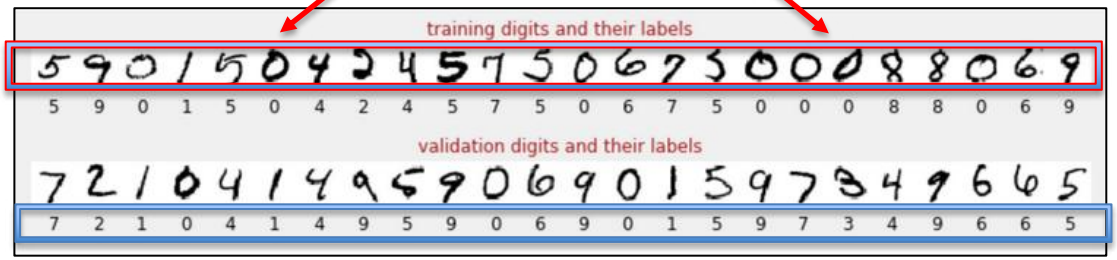
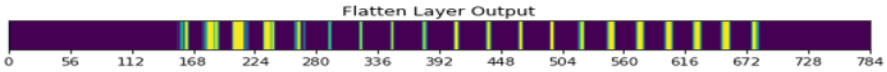


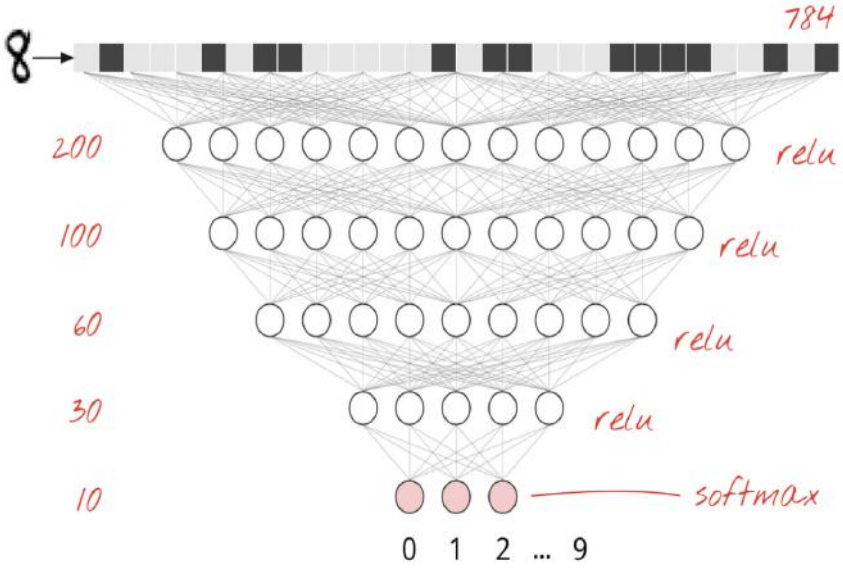
Image : input

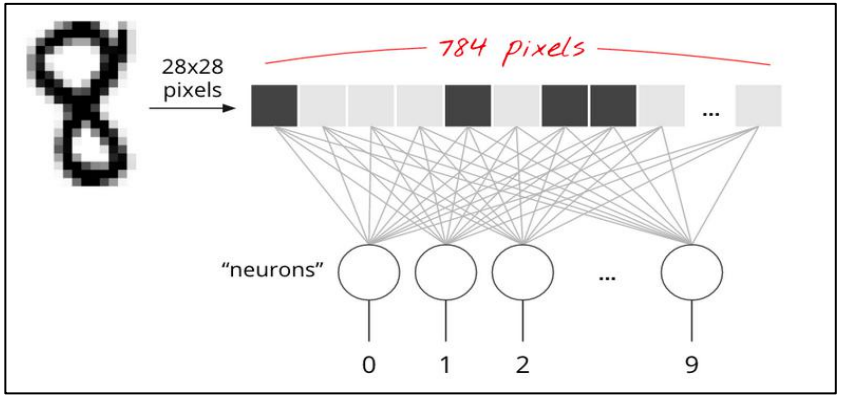


Labels : target

Flatten the 28 x 28 matrix
to a vector of 28 x 28 elements 784 elements







TensorFlow, Keras and deep learning, without a PhD
https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist#0

Simple MNIST MLP Neural Network

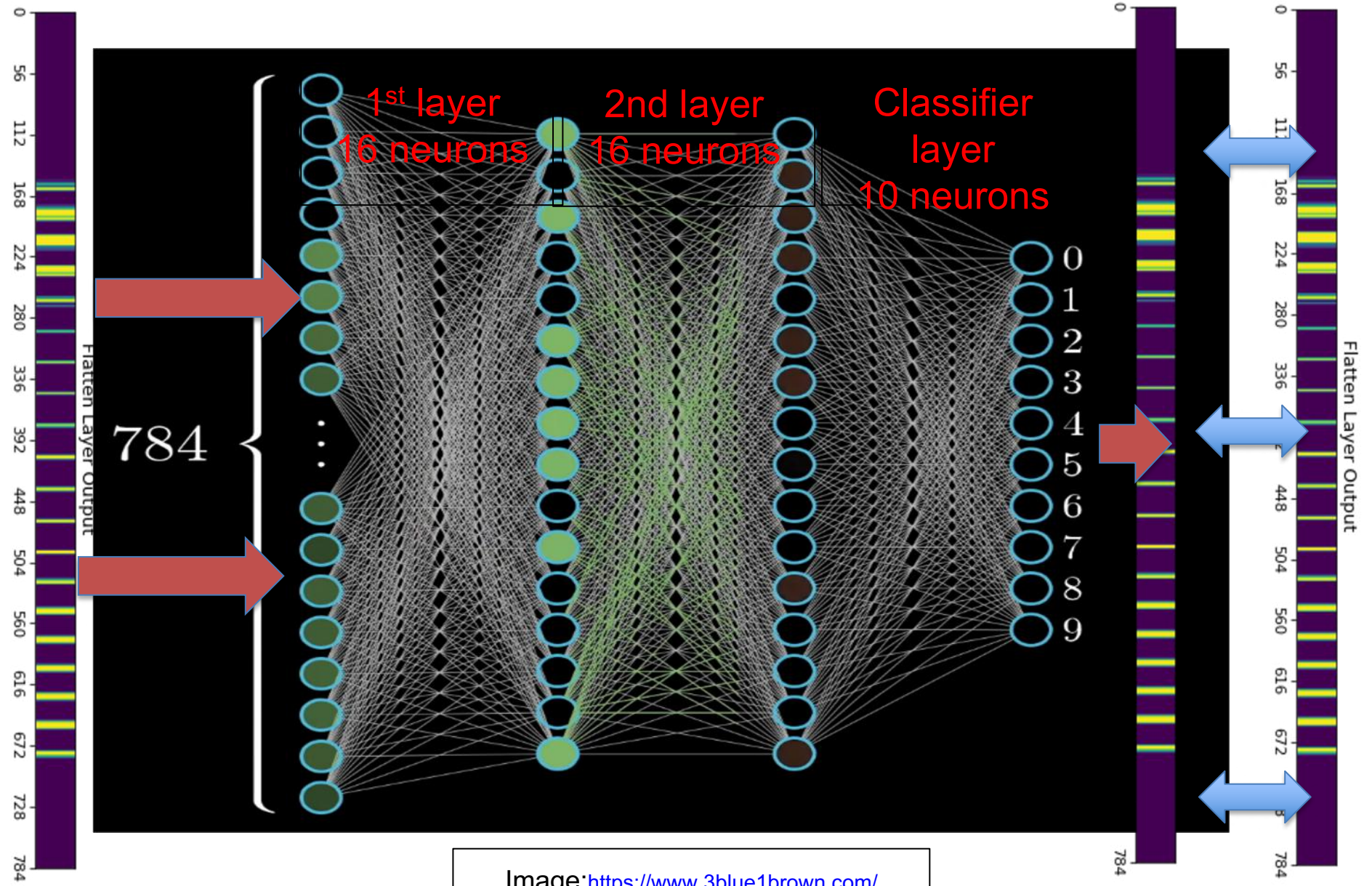
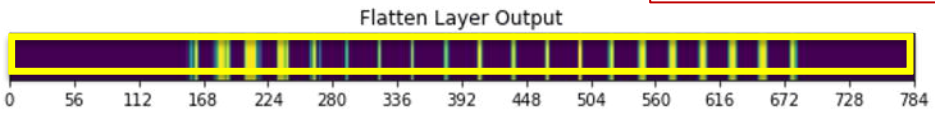
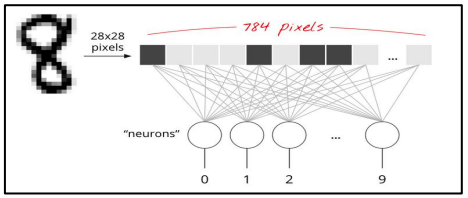


Image:https://www.3blue1brown.com/

Summary : Flow of MLP Dense layer NN

1 image = a vector of 784





An image of 28x28 pixels

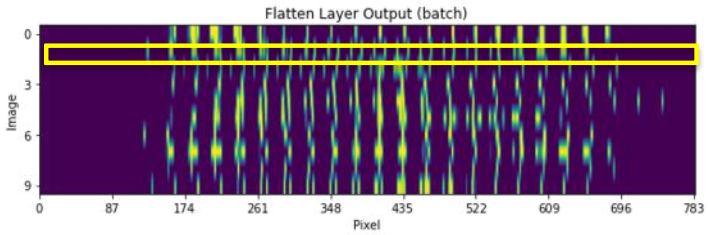
Input to 1st layer

10 images : batch of 10
10 x 784 matrix

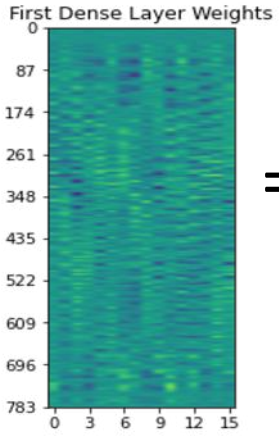
16 neurons
784 x 16 W matrix

10 x 16 output matrix

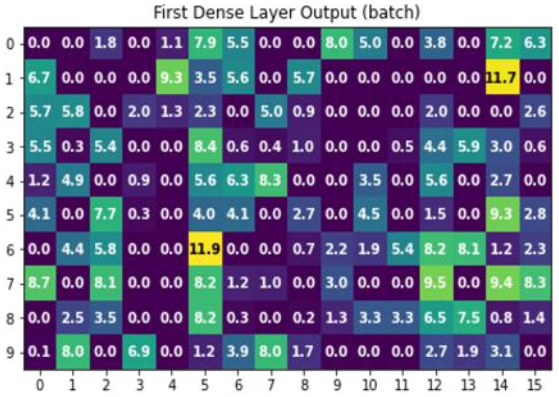
Input to 2nd layer



*

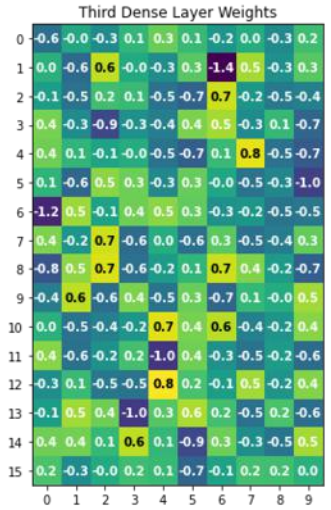


=



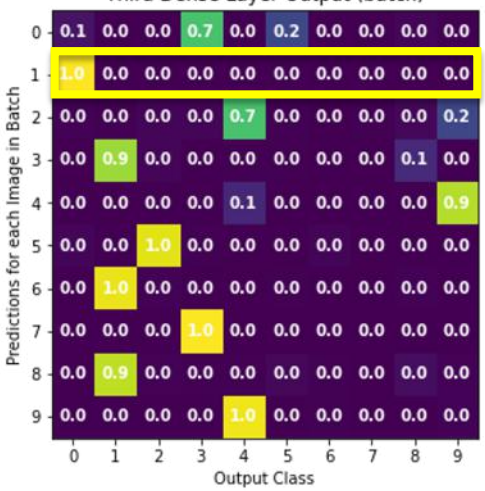
10 x 16 matrix

*

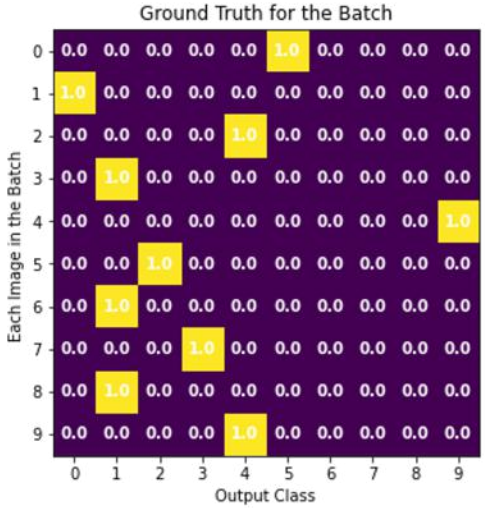


16 neurons
16 x 10
W matrix

=



10 x 10 output matrix



10 x 10 labeled matrix

Tensorflow Example

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(16, activation='relu'),
 tf.keras.layers.Dense(16, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

Tensorflow is imported

Sets the mnist dataset to variable "mnist"

Loads the mnist dataset

Builds the layers of the model
4 layers in this model

```
model.compile(optimizer='sdg', loss='SparseCategoricalCrossentropy',
metrics=['accuracy'])

model.summary()

logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-
%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
```

Loss Function

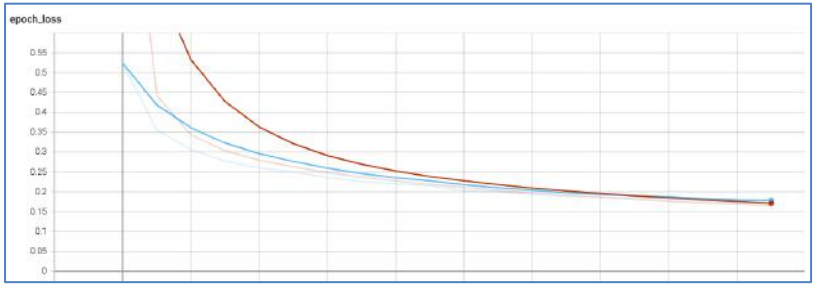
Compiles the model with the SGD optimizer

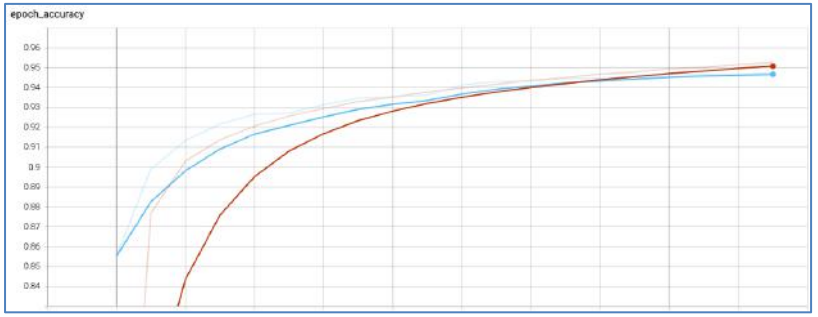
Print summary

Use tensorborad

```
model.compile(optimizer='sgd',
              loss='SparseCategoricalCrossentropy',
              metrics=['accuracy'])

model.summary()

logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)

model.fit(x_train, y_train, epochs=20, batch_size=50,validation_data=(x_test, y_test),
          callbacks=[tensorboard_callback])

model.evaluate(x_test,  y_test, verbose=2)

%tensorboard --logdir logs
```

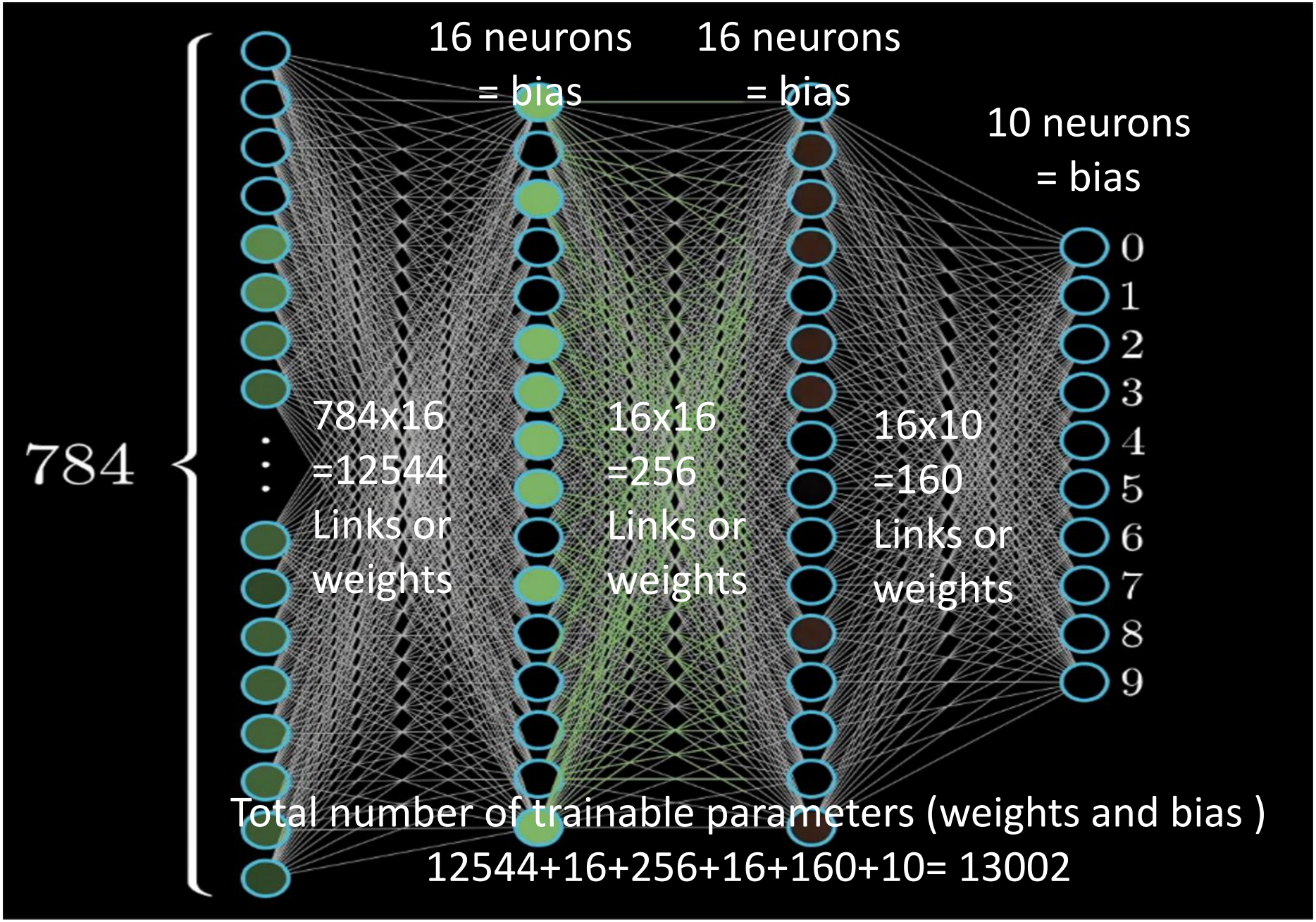
```
2.4.1
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_3 (Dense)              (None, 16)                12560
_________________________________________________________________
dense_4 (Dense)              (None, 16)                272
_________________________________________________________________
dense_5 (Dense)              (None, 10)                170
=================================================================
Total params: 13,002
Trainable params: 13,002
Non-trainable params: 0
```





Simple MLP Network

Image:https://www.3blue1brown.com/



Tensorflow Example

The model is trained in about 10 seconds completing 5 epochs with a Nvidia GTX 1080 GPU and has an accuracy of around 97-98%

```
2020-07-29 19:53:40.592860: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1108]      0
2020-07-29 19:53:40.592871: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1121] 0:   N
2020-07-29 19:53:40.593073: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from
SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2020-07-29 19:53:40.593535: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from
SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2020-07-29 19:53:40.593973: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] successful NUMA node read from
SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2020-07-29 19:53:40.594387: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1247] Created TensorFlow device (/job:lo
calhost/replica:0/task:0/device:GPU:0 with 7219 MB memory) -> physical GPU (device: 0, name: GeForce GTX 1080, pci bus
id: 0000:26:00.0, compute capability: 6.1)
2020-07-29 19:53:40.623075: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x55d981198b80 initialized fo
r platform CUDA (this does not guarantee that XLA will be used). Devices:
2020-07-29 19:53:40.623096: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecutor device (0): GeForce GTX
1080, Compute Capability 6.1
2020-07-29 19:53:52.215159: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
 library libcublas.so.10
Epoch 1/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2982 - accuracy: 0.9127
Epoch 2/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1433 - accuracy: 0.9571
Epoch 3/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1097 - accuracy: 0.9663
Epoch 4/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0893 - accuracy: 0.9730
Epoch 5/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0777 - accuracy: 0.9750
313/313 - 0s - loss: 0.0727 - accuracy: 0.9776
```

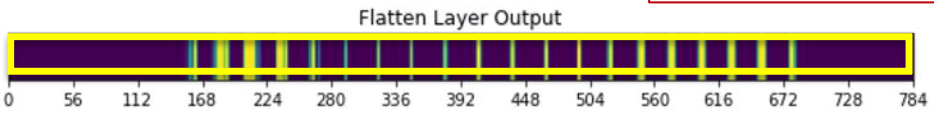
CE : Categorical Cross Entropy Loss
Classification
0, 1, 2, …., 9 MNIST Example
Image Recognition

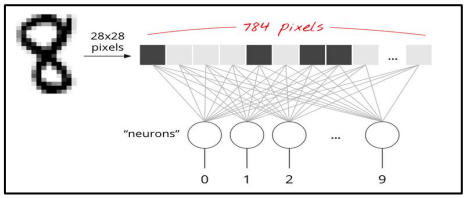
Cross Entropy Loss = CL
Need d(CL)/dwi for backpropagation

For Images
Convolutional Neural Network

Summary : Flow of MLP Dense layer NN

1 image = a vector of 784

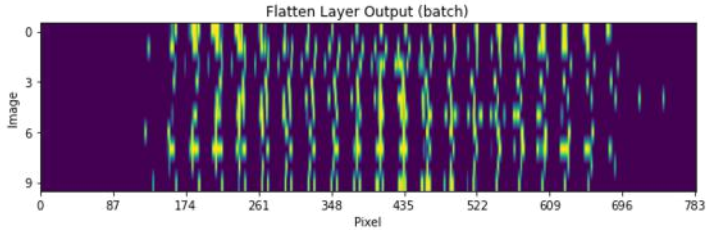




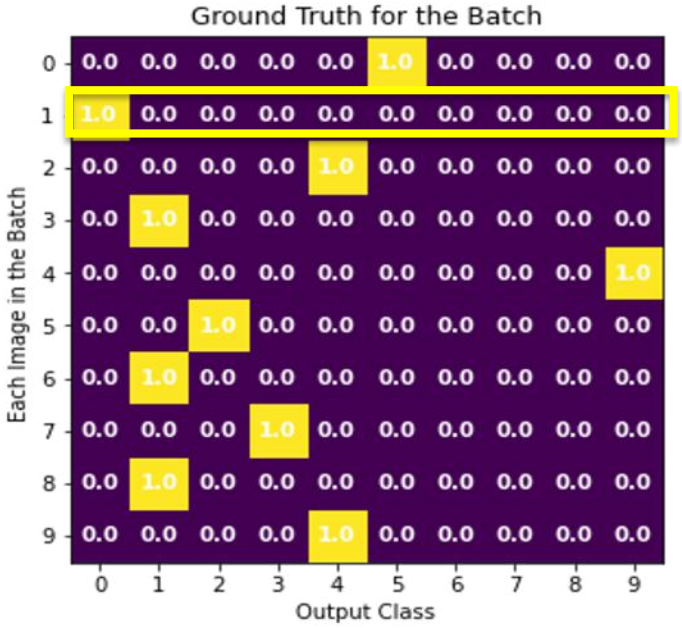
An image of 28x28 pixels

Input to 1st layer

10 images : batch of 10
10 x 784 matrix

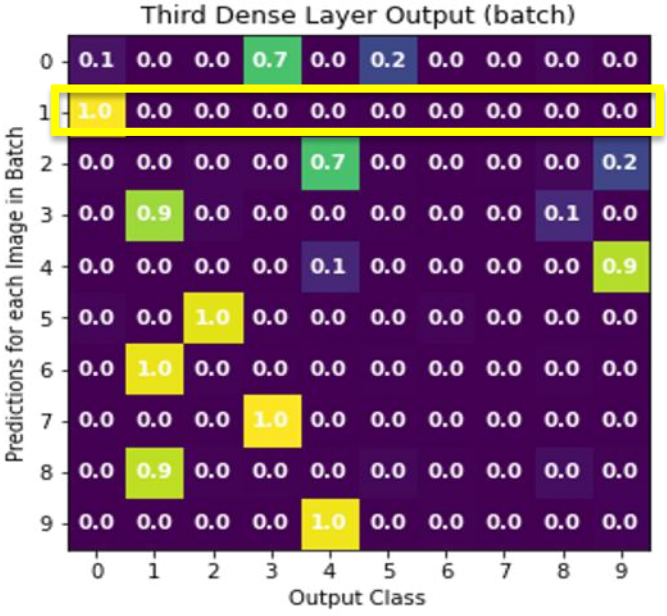


10 x 10 labeled matrix

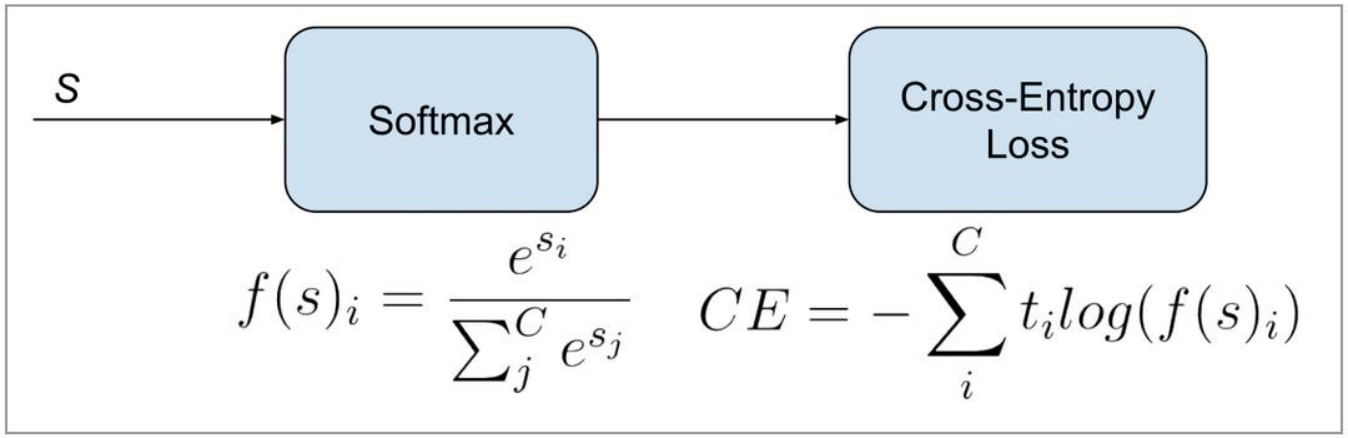


Comparing Computed NN results To Labeled results (target)

10 x 10 output matrix

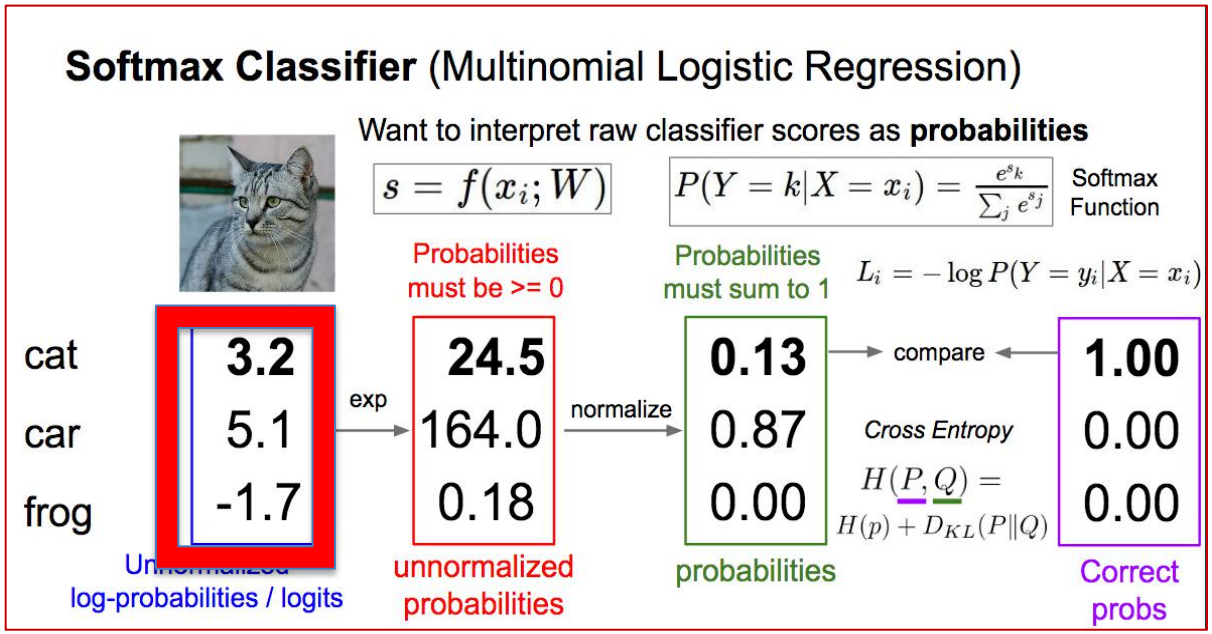


Categorical (Softmax) Cross Entropy Loss (Statistical Learning)



T = 1

ti and si are the groundtruth (label) and the computed score for each class i in C.



e^3.2 = 24.5

e^5.1 = 164.0

e^-1.7 = 0.18

188.68

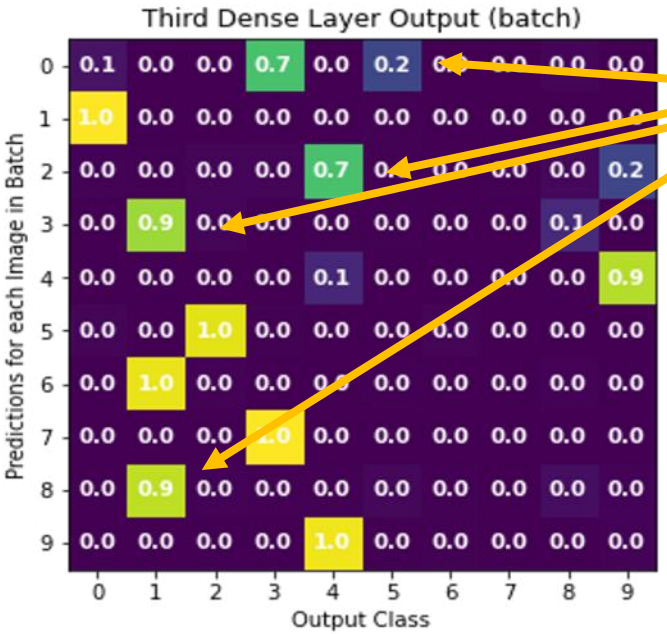
Software = 24.5 / 188.68 = 0.13

Software = 164 / 188.68 = 0.87

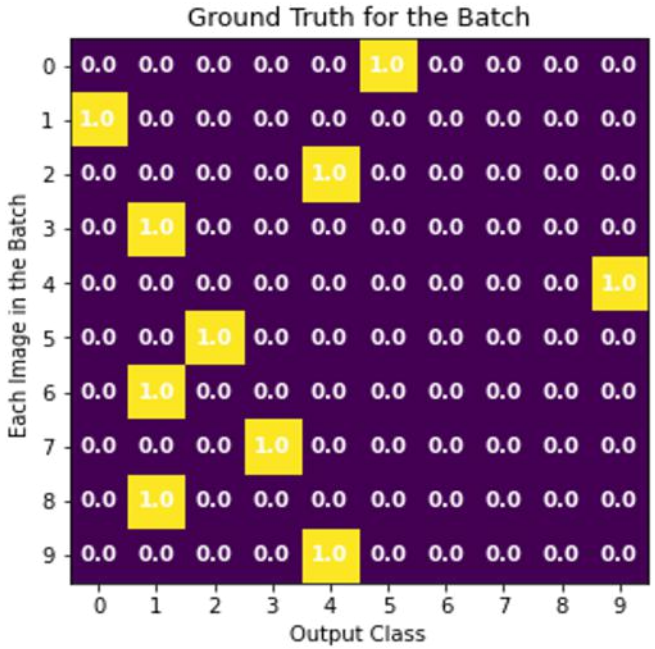
Software = -1.7 / 188.68 = 0.00

https://gombru.github.io/2018/05/23/cross_entropy_loss/

Evaluating the Error



Z



Categorical Cross Entropy Loss

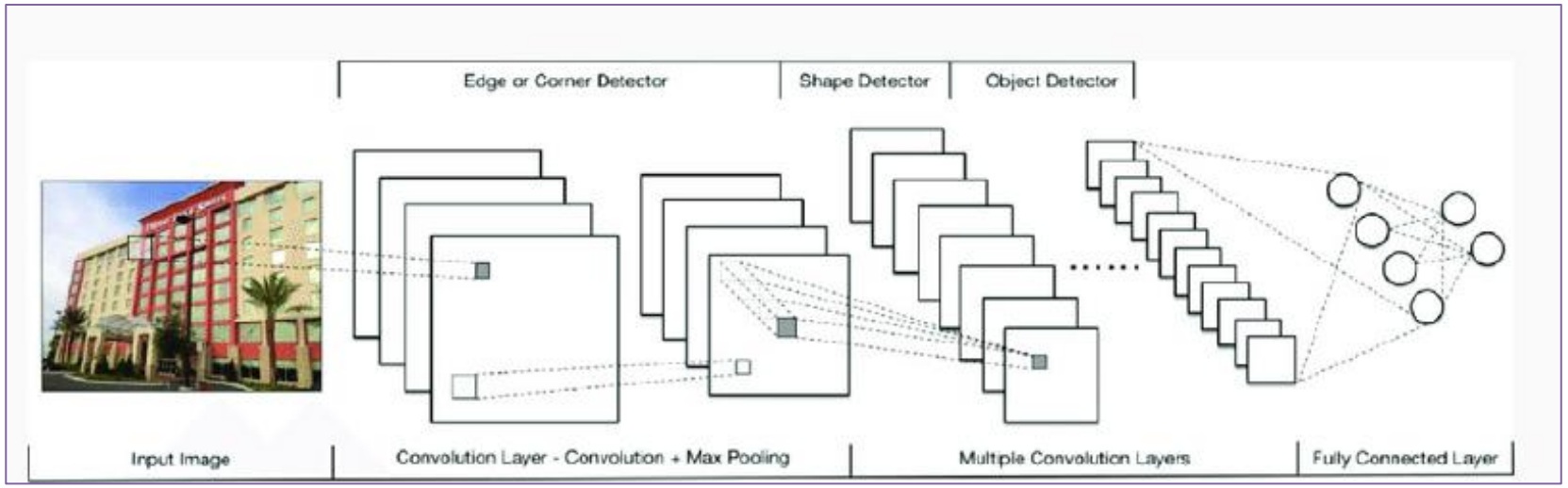
$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

L (i) = - log0.2 = 1.61	Image 0 = 5
- log1.0 = 0	Image 1 = 0
- log0.7 = 0.356	Image 2 = 4
- log0.9 = 0.105	Image 3 = 1
- log0.9 = 0.105	Image 4 = 9
- log1.0 = 0	Image 5 = 2
- log1.0 = 0	Image 6 = 1
- log1.0 = 0	Image 7 = 3
- log0.9 = 0.105	Image 8 = 2
- log1.0 = 0	Image 9 = 4

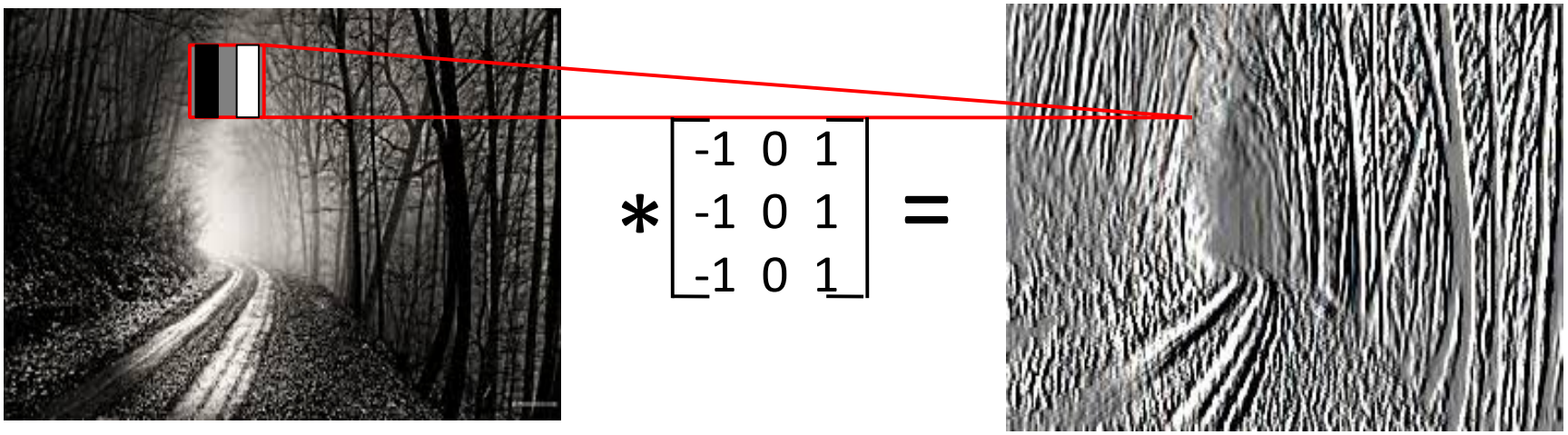
Convolutional Neural Network

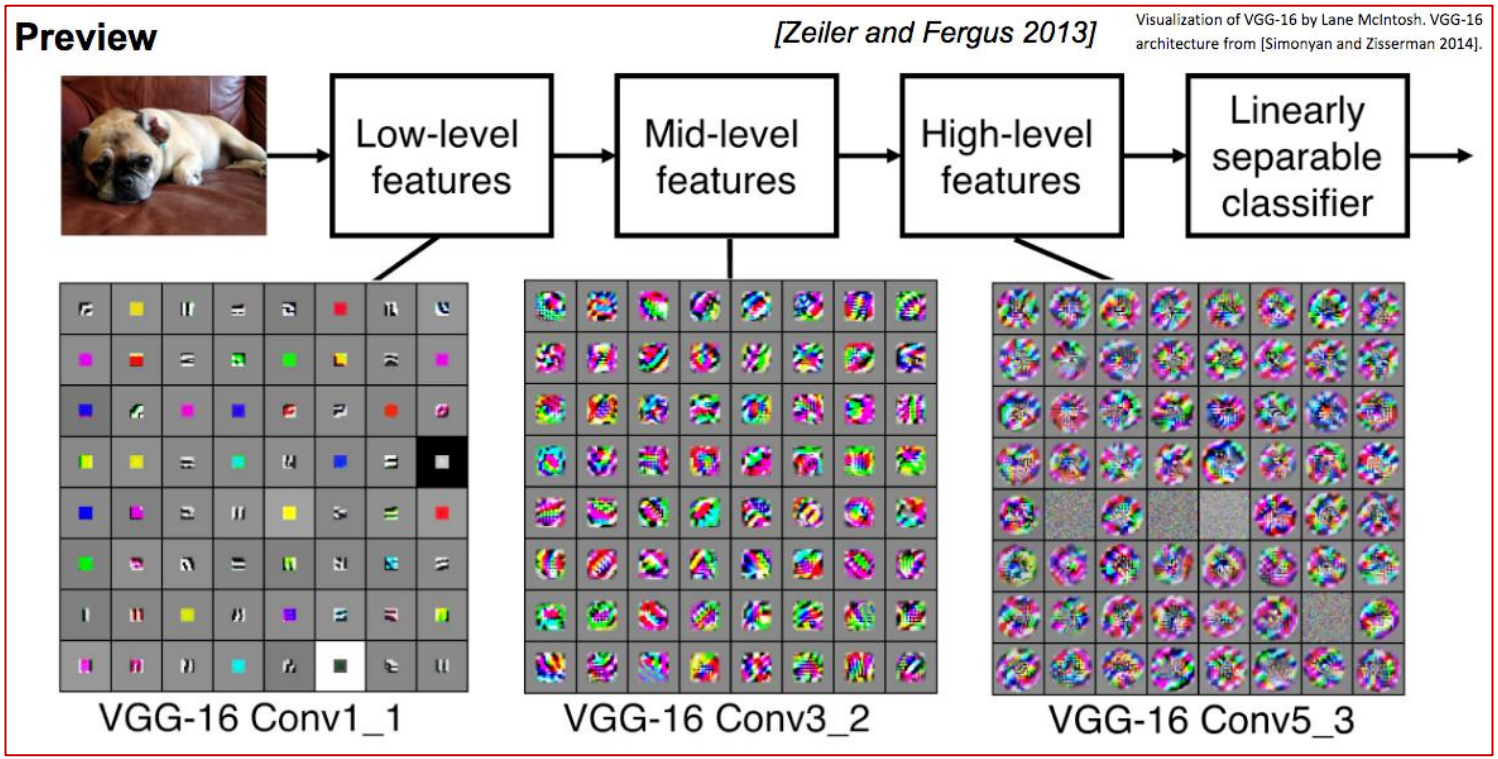
- ✓ **Convolutional Neural Network Introduction**
- ✓ **CNN example, Convolution operations**
- ✓ **MM in convolution operations**

- ✓ **CNN framework :**a class of neural network where the input is modified by a filter
- ✓ Multiple filters in convolutional networks are applied to different slices of an image, mapping them one by one, and identifying different features of an input image

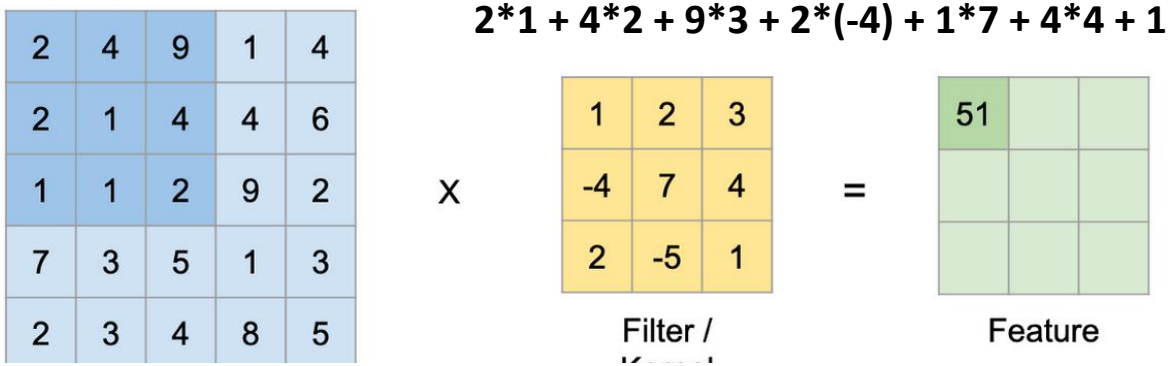


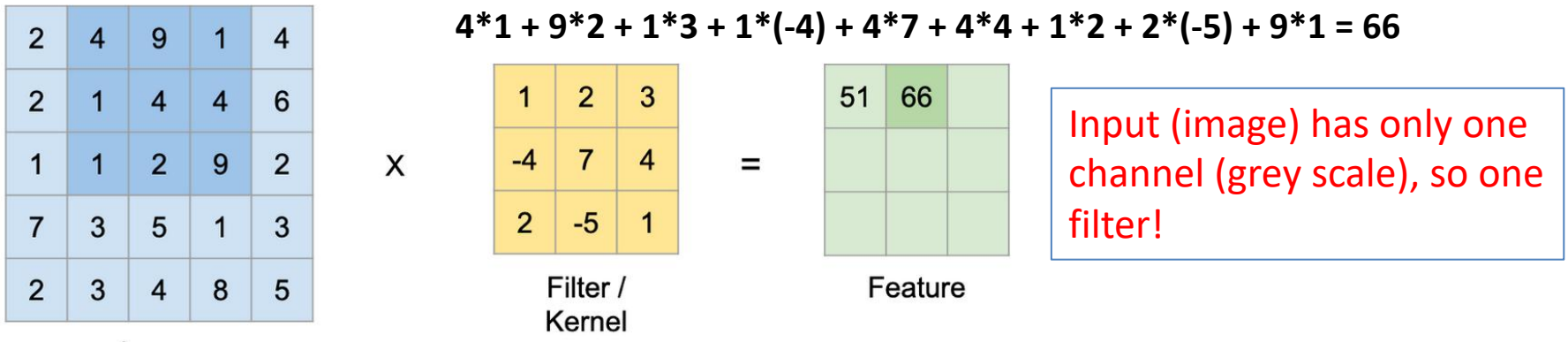
CNN filters


-1 0 1
-1 0 1
-1 0 1
*
=


Preview
[Zeiler and Fergus 2013]
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].
Low-level features
Mid-level features
High-level features
Linearly separable classifier
VGG-16 Conv1_1
VGG-16 Conv3_2
VGG-16 Conv5_3

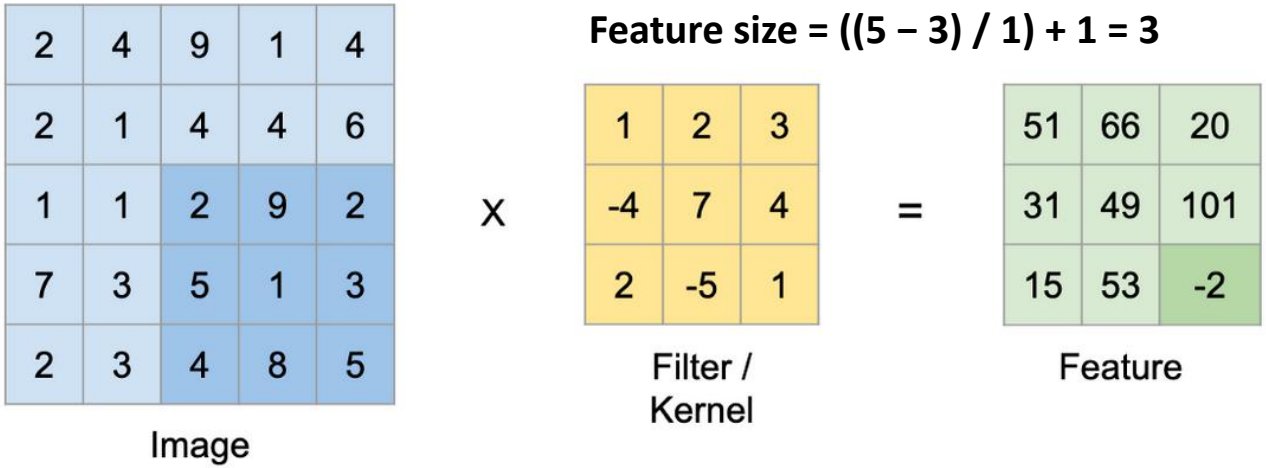
https://towardsdatascience.com/convolution-neural-networks-a-beginners-guide-implementing-a-mnist-hand-written-digit-8aa60330d022





Feature size = ((Image size − Kernel size) / Stride) + 1

Feature size = ((5 − 3) / 1) + 1 = 3



Feature size = ((Image size + 2 * Padding size − Kernel size) / Stride)+1

Feature size = ((5 + 2 * 1 − 3) / 1) + 1= 5

0	0	0	0	0	0	0
0	2	4	9	1	4	0
0	2	1	4	4	6	0
0	1	1	2	9	2	0
0	7	3	5	1	3	0
0	2	3	4	8	5	0
0	0	0	0	0	0	0

Image

X

1	2	3
-4	7	4
2	-5	1

Filter / Kernel

=

21	59	37	-19	2
30	51	66	20	43
-14	31	49	101	-19
59	15	53	-2	21
49	57	64	76	10

Feature

21	59	37	-19	2
30	51	66	20	43
-14	31	49	101	-19
59	15	53	-2	21
49	57	64	76	10

Convolved Feature

21	59	37	-19	2
30	51	66	20	43
-14	31	49	101	-19
59	15	53	-2	21
49	57	64	76	10

Convolved Feature

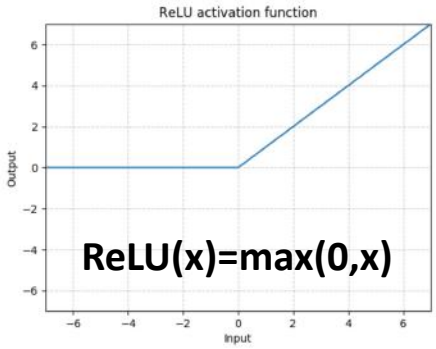
59	66		

Max Pooled Feature

59	66	66	43
51	66	101	101
59	53	101	101
59	64	76	76

Max Pooled Feature

MaxPool(Relu(x)) = Relu(MaxPool(x))



The procedure satisfies the communicative property and can be used either way. In practice ReLU activation function is applied right after a convolution layer and then that output is max pooled. Change negative numbers to zero, positive numbers stay the same

-?

0

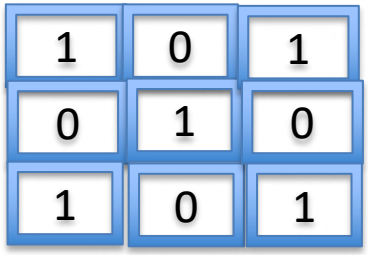
CNN : Forward Path
A Simple Example

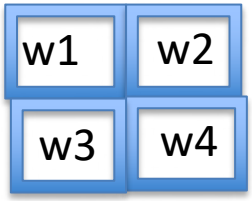
Example exercise

Compute the updated values of filter weights after backpropagation of the following problem -- 3x3 input matrix, 2x2 filter, stride 1, no padding, flatten it and use it as input connecting to a two-neuron layer, then pass the results to a softmax binary classification of two neurons.

3a) (0.5%) What is the values of the output at the end of the forward path

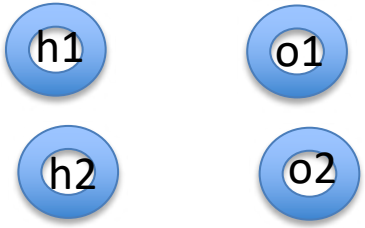
3b) (1.5%) What are the updated values of the weight and bias after backpropagation





b1 = 0.1

w1=0.9
w2=0.1
w3=0.1
w4=0.9



Apply ReLU activation function after the convolution step

Flatten the value after applying the ReLU activation function

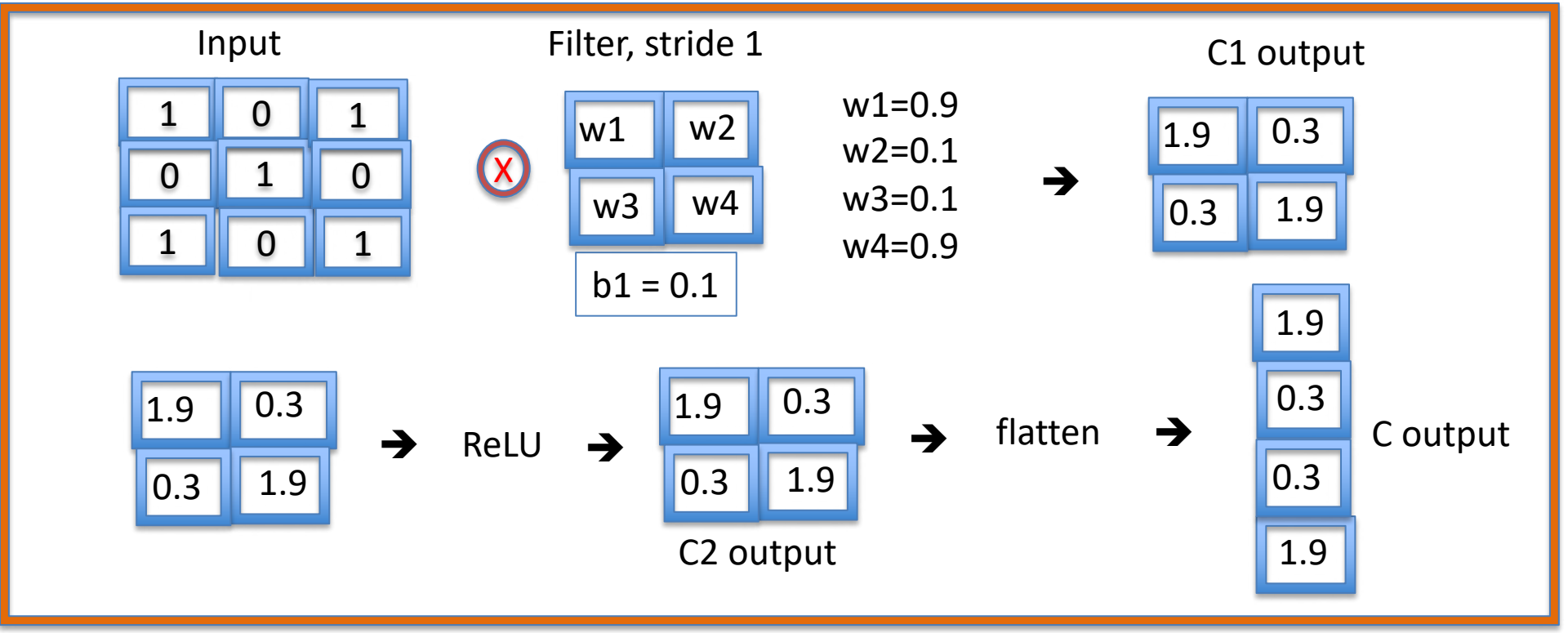
Assuming the weight of each link to this layer is w = 0.1 to 0.8, b2=0.2

Fully connecting to a layer of two neurons

Assuming the weight of each link is w = 0.1 to 0.4, b3=0.3

Fully Connected layer of 2 softmax neurons

Labeled output of two classes are 1.0 and 0.00



ReLU (1xw1+0xw2+0xw3+1xw4+b1 =1x0.9+0x0.1+0x0.1+1x0.9+0.1=1.9) = C1

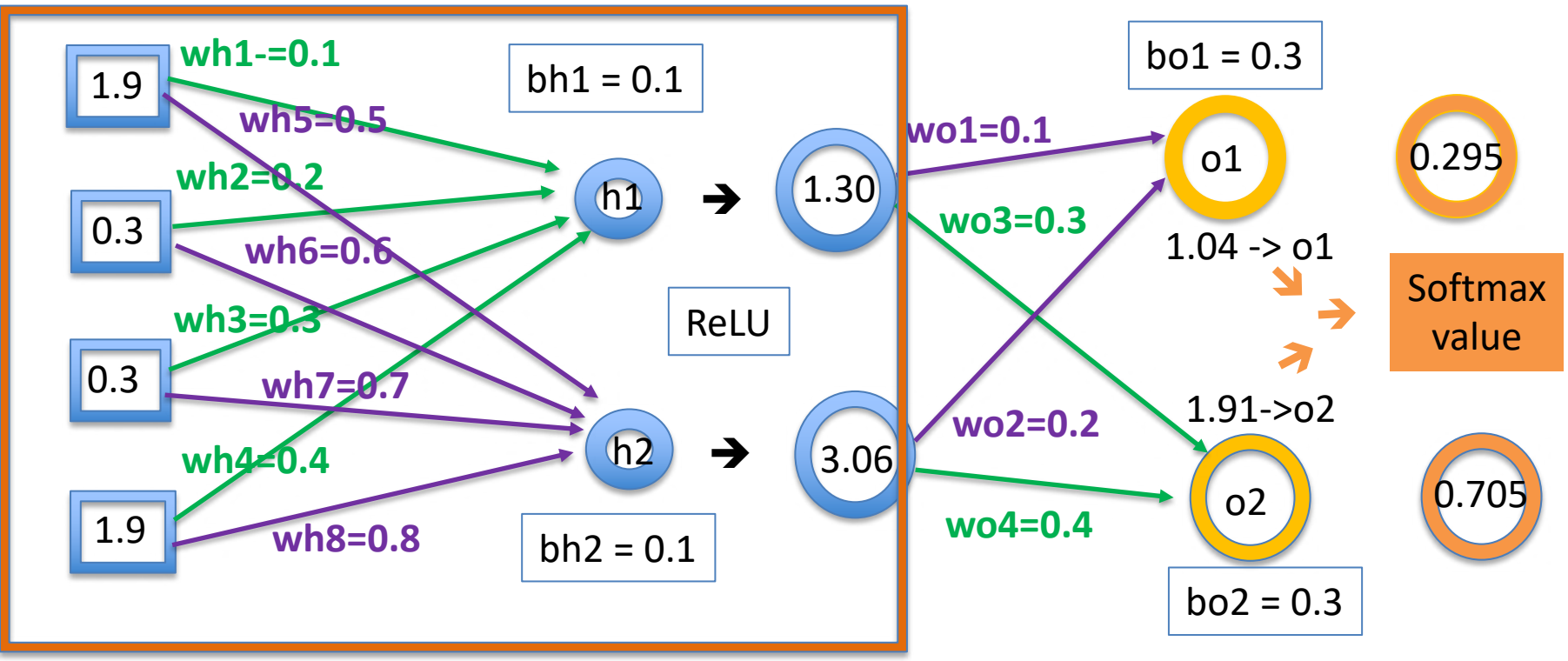
ReLU (0xw1+1xw2+1xw3+0xw4+b1=0x0.9+1x0.1+1x0.1+0x0.9+0.1=0.3) = C2

ReLU (0xw1+1xw2+1xw3+0xw4+b1=0x0.9+1x0.1+1x0.1+0x0.9+0.1=0.3) = C3

ReLU (1xw1+0xw2+0xw3+1xw4+b1 =1x0.9+0x0.1+0x0.1+1x0.9+0.1=1.9) = C4

1.9
0.3
0.3
1.9

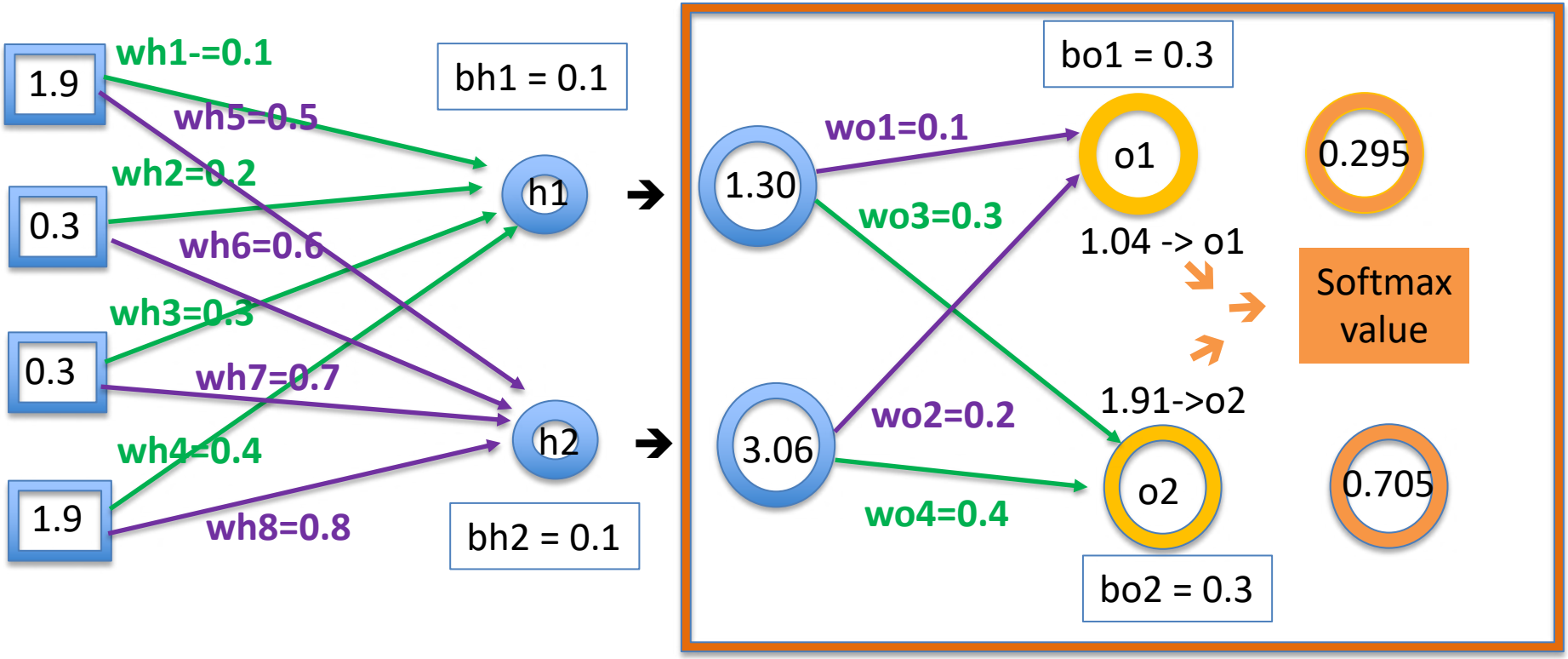
C output



C1xwh1+ C2xwh2 + C3xwh3 x C4xwh4 + bh1 = 1.9x0.1+0.3x0.2+0.3x0.3+1.9x0.4+0.1= 1.3 = h1

C1xwh5+ C2xwh6 + C3xwh7 x C4xwh8 + bh2 = 1.9x0.5+0.3x0.6+0.3x0.7+1.9x0.8+0.1= 3.06 = h2

ReLU { h1, h2 } = {H1, H2} = {1.3, 3.06}



Softmax (H1 x wo1+ H2 x wo2 + bo1 = 1.3 x 0.1+ 3.06 x 0.2 +0.3 = 1.042) = O1

Softmax (H1 x wo3+ H2 x wo4 + bo2 = 1.3 x 0.3+ 3.06 x 0.4 +0.3 = 1.914) = O2

Softmax function

$$\sigma(j) = \frac{\exp(\mathbf{w}_j^\top \mathbf{x})}{\sum_{k=1}^{K} \exp(\mathbf{w}_k^\top \mathbf{x})} = \frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)}$$

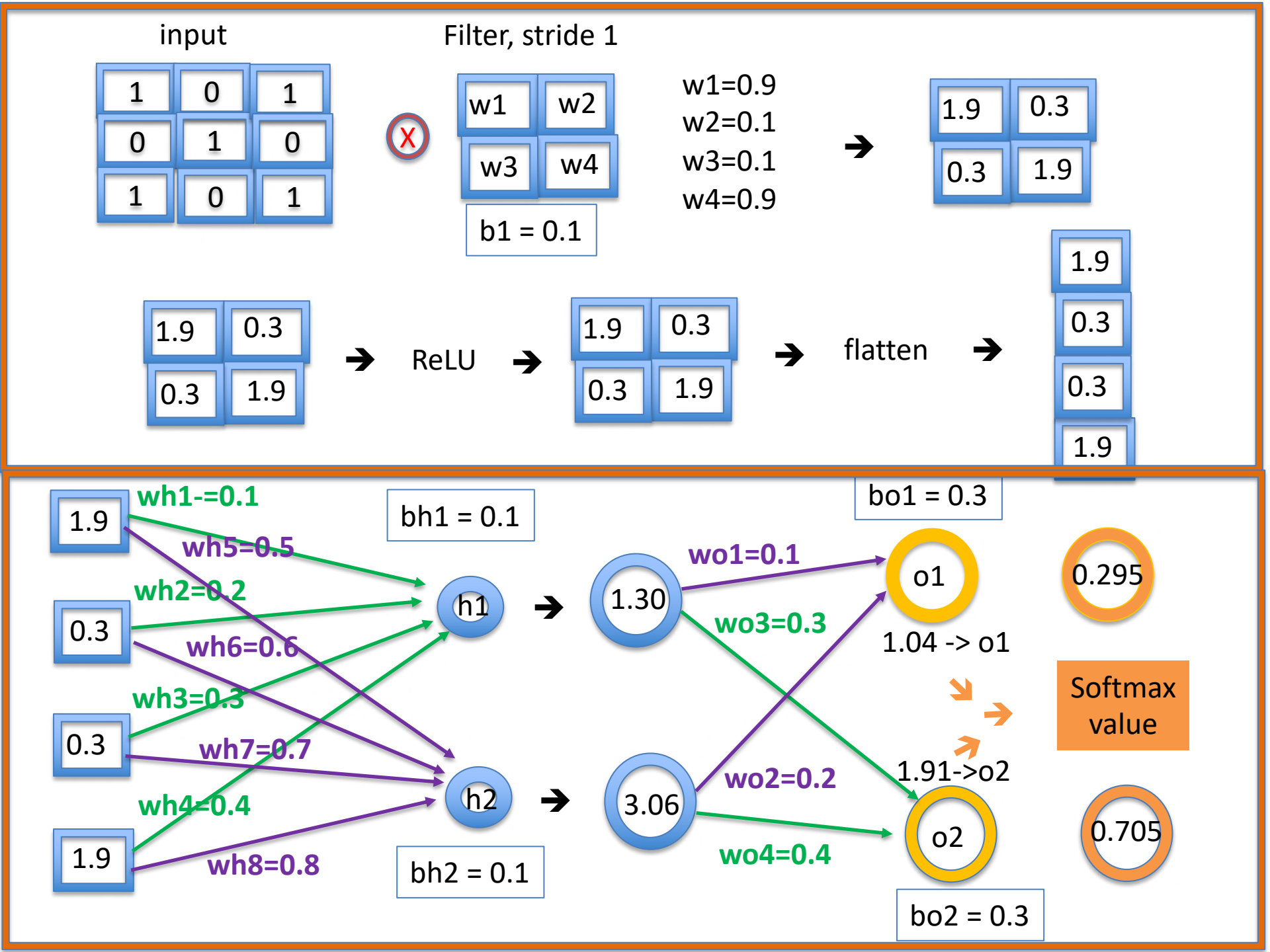
exp(1.042) = 2.8348

exp(1.914) = 6.7802

exp(1.042) + exp(1.914) = 9.615

O1 = 0.295

O1 = 0.705


input
Filter, stride 1
1 0 1
0 1 0
1 0 1
X
w1 w2
w3 w4
b1 = 0.1
w1=0.9
w2=0.1
w3=0.1
w4=0.9
1.9 0.3
0.3 1.9
1.9 0.3
0.3 1.9
ReLU
1.9 0.3
0.3 1.9
flatten
1.9
0.3
0.3
1.9
1.9
wh1-=0.1
wh5=0.5
wh2=0.2
wh6=0.6
wh3=0.3
wh7=0.7
wh4=0.4
wh8=0.8
0.3
0.3
1.9
bh1 = 0.1
h1
h2
bh2 = 0.1
1.30
3.06
wo1=0.1
wo3=0.3
wo2=0.2
wo4=0.4
bo1 = 0.3
o1
o2
bo2 = 0.3
1.04 -> o1
1.91->o2
Softmax value
0.295
0.705

CNN Arithmetic, multiple input channels (RGB image)

Input : W=5, H=5, D=3 (Channel)

Filter : K=2 (number), F=3 x3 (size), S (Stride)=2, P (Zero Padding) = 1

(W-F+S)/3 + P = output volume size
Output : W=3, H=3, D=2 (Channel)

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	1	0	1	0	1	0
0	1	2	0	1	0	0
0	1	0	1	1	2	0
0	0	0	2	1	2	0
0	1	2	2	1	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	1	0	1	2	0
0	0	0	1	0	1	0
0	1	0	1	1	1	0
0	2	2	2	1	1	0
0	1	0	2	2	1	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	0	2	0	1	1	0
0	2	1	2	1	1	0
0	2	2	0	1	2	0
0	0	2	1	2	1	0
0	2	2	2	1	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

0	-1	1
0	1	1
-1	0	1

w0[:,:,1]

0	0	1
0	1	1
1	-1	0

w0[:,:,2]

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

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

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

w1[:,:,1]

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

w1[:,:,2]

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

Bias b1 (1x1x1)

b1[:,:,0]

0

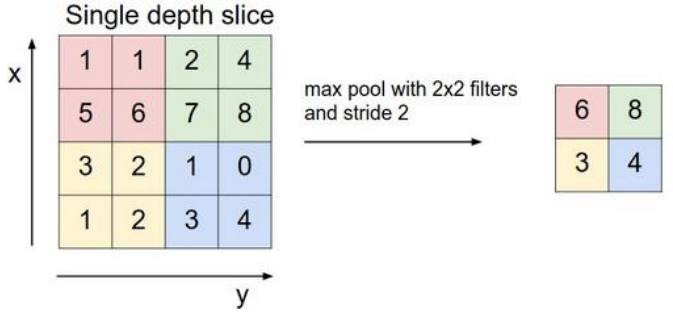
Output Volume (3x3x2)

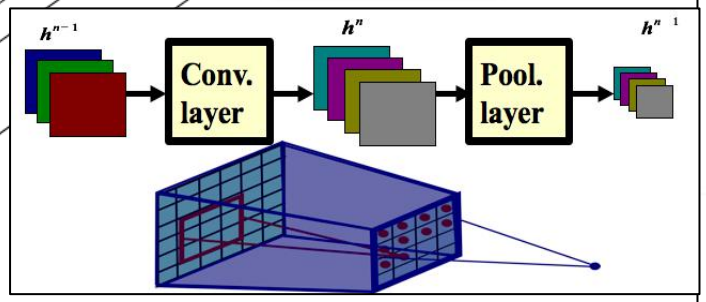
o[:,:,0]

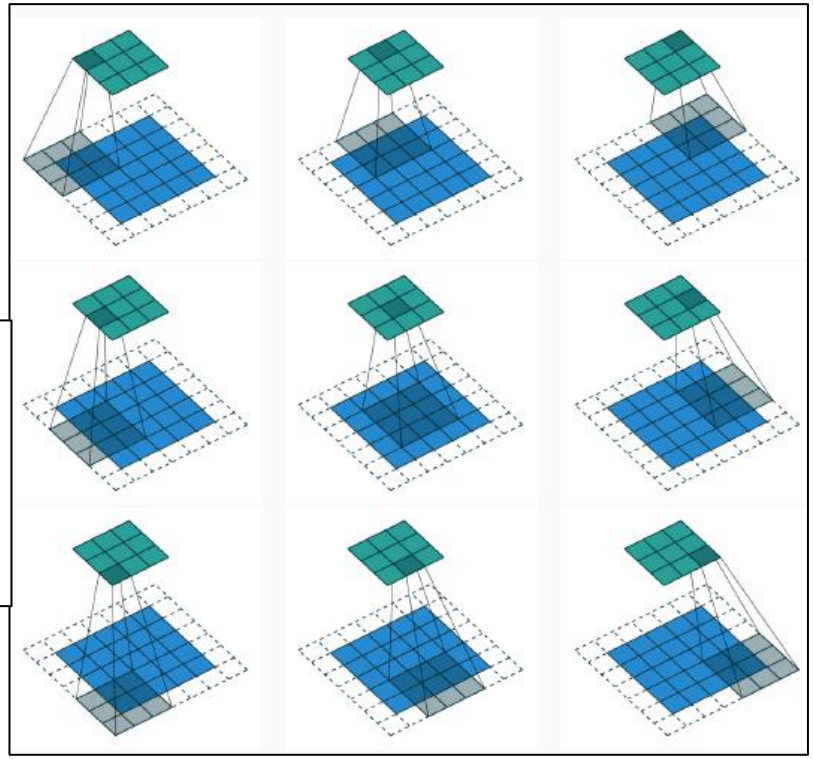
7	0	0
2	6	0
5	5	0

o[:,:,1]

-1	1	-2
0	-5	-5
-4	-10	-9

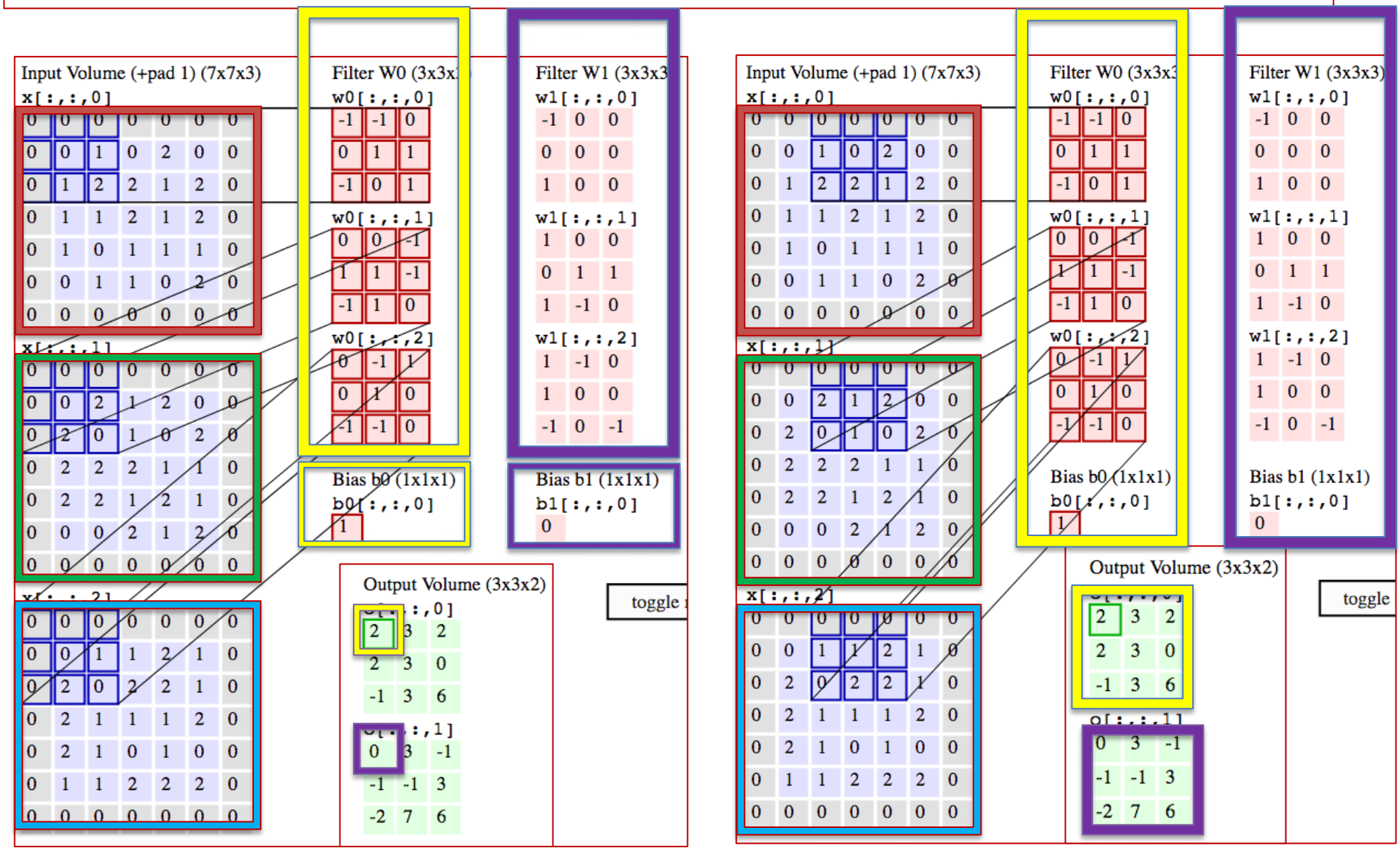






https://cs231n.github.io/convolutional-networks/

Input volume : $W1=5, H1=5, D1=3$, CONV layer parameters = $K=2, F=3, S=2, P=1$.



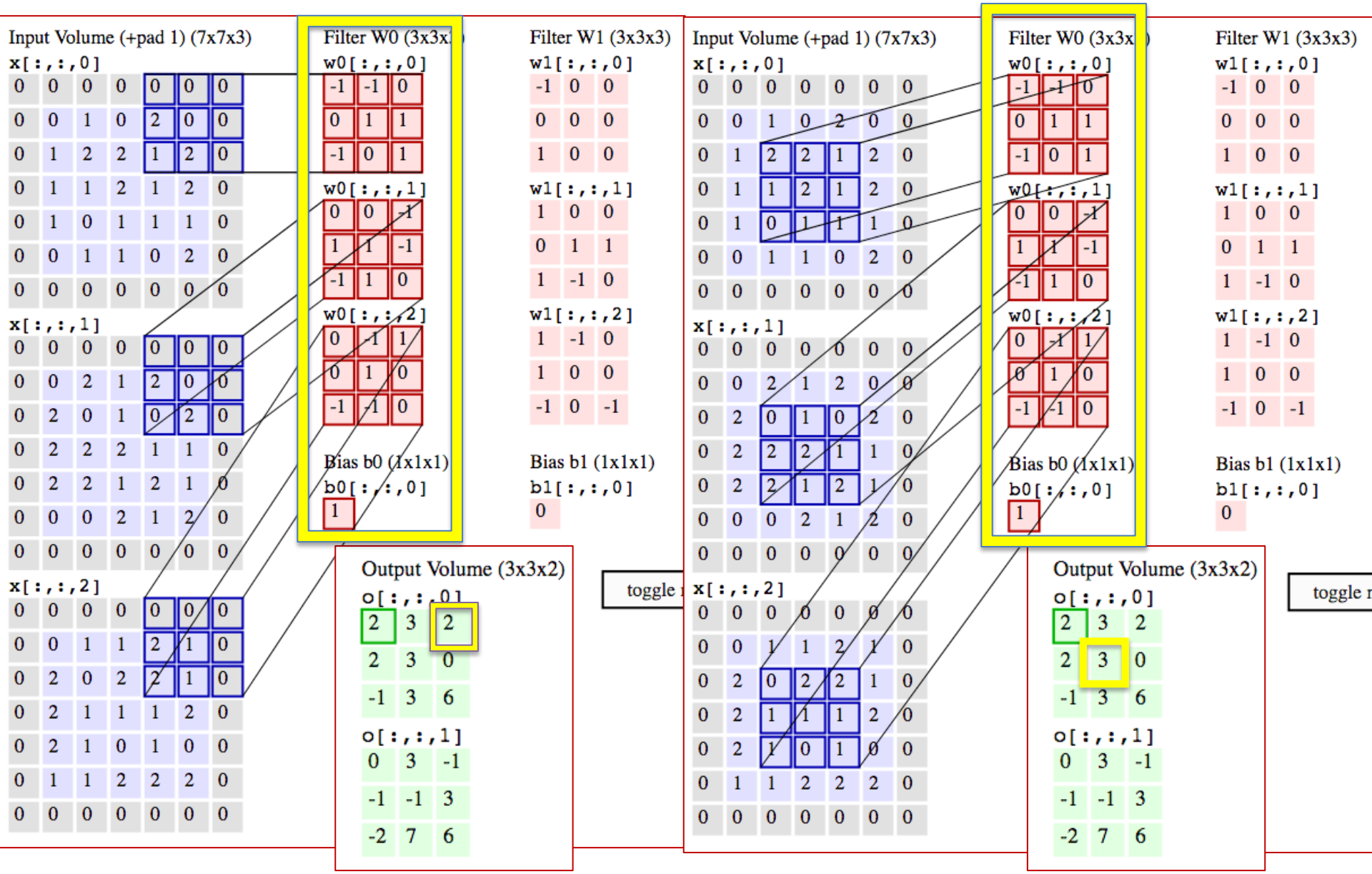
(0 x -1 + 0 x -1 + 0 x 0) + (0 x 0 + 0 x 1 + 1 x 1) + (0 x -1 + 1 x 0 + 2 x 1) = 3

(0 x 0 + 0 x 0 + 0 x -1) + (0 x 1 + 0 x 1 + 2 x -1)+ (0 x -1 + 2 x 1 + 0 x 0) = 0

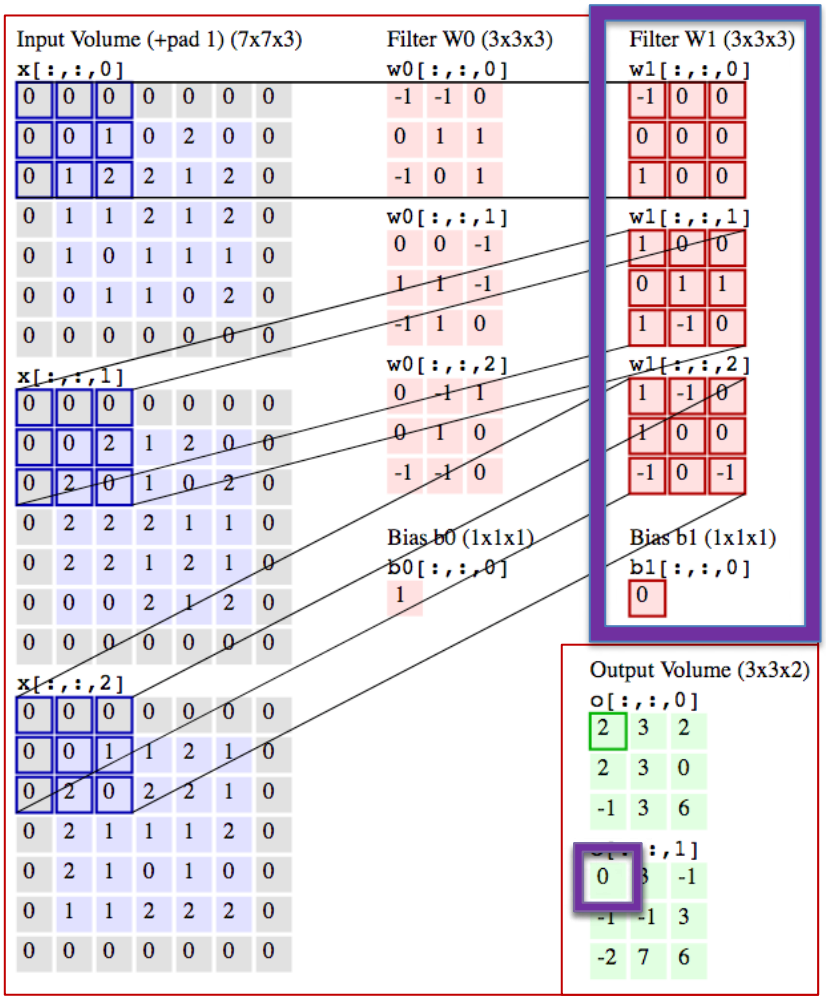
(0 x 0 + 0 x -1 + 0 x -1) + (0 x 0 + 0 x 1 + 1 x 0)+ (0 x -1 + 2 x -1 + 0 x 0) = -2

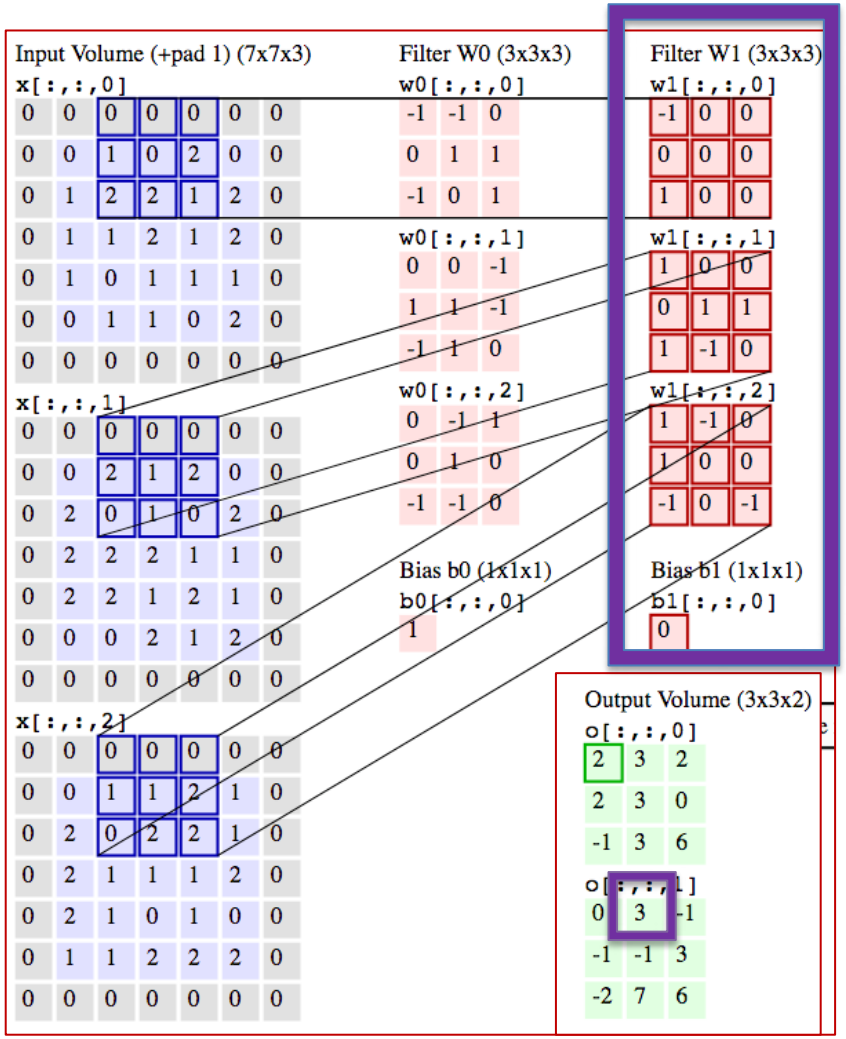
1 + b0 (1) = 2

CNN Filter Operations - RGB



CNN Filter Operations - RGB





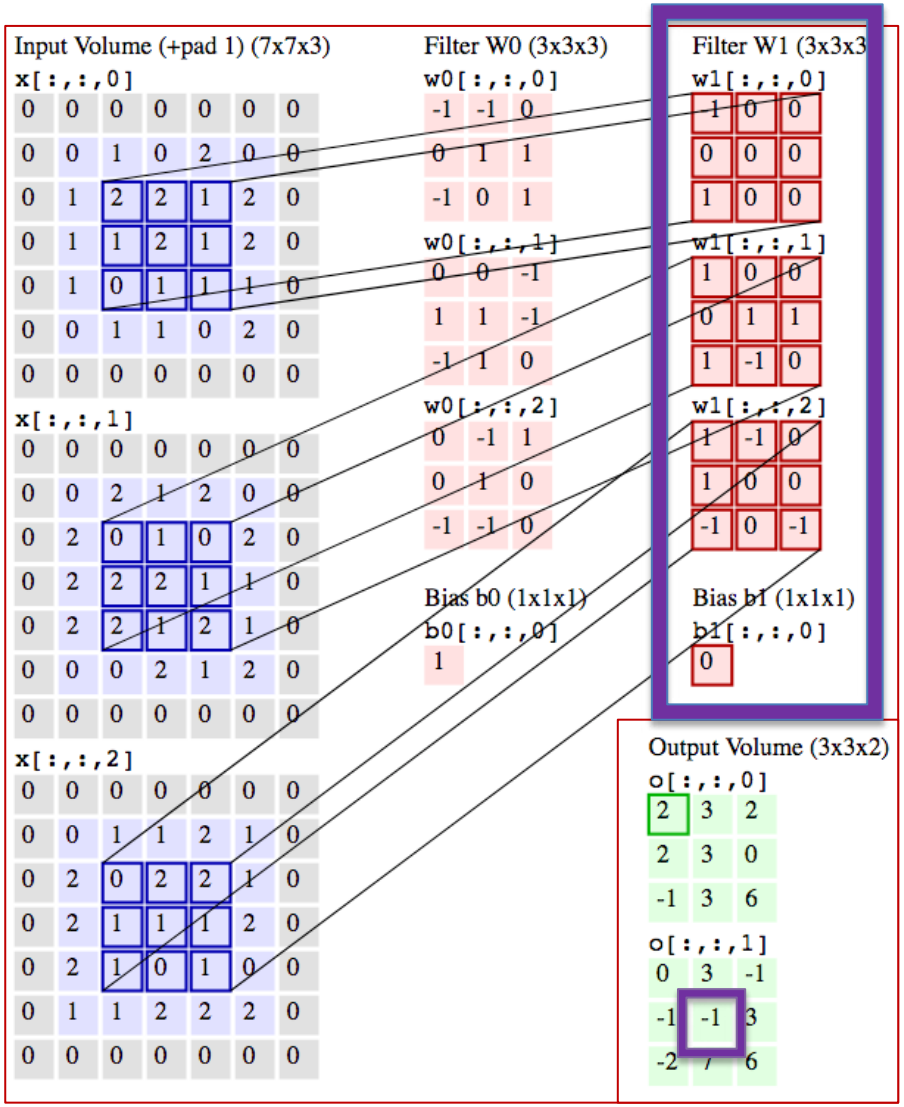
(0 x -1 + 0 x 0 + 0 x 0) + (0 x 0 + 0 x 0 + 1 x 0) + (0 x 1 + 1 x 0 + 2 x 0) = 0

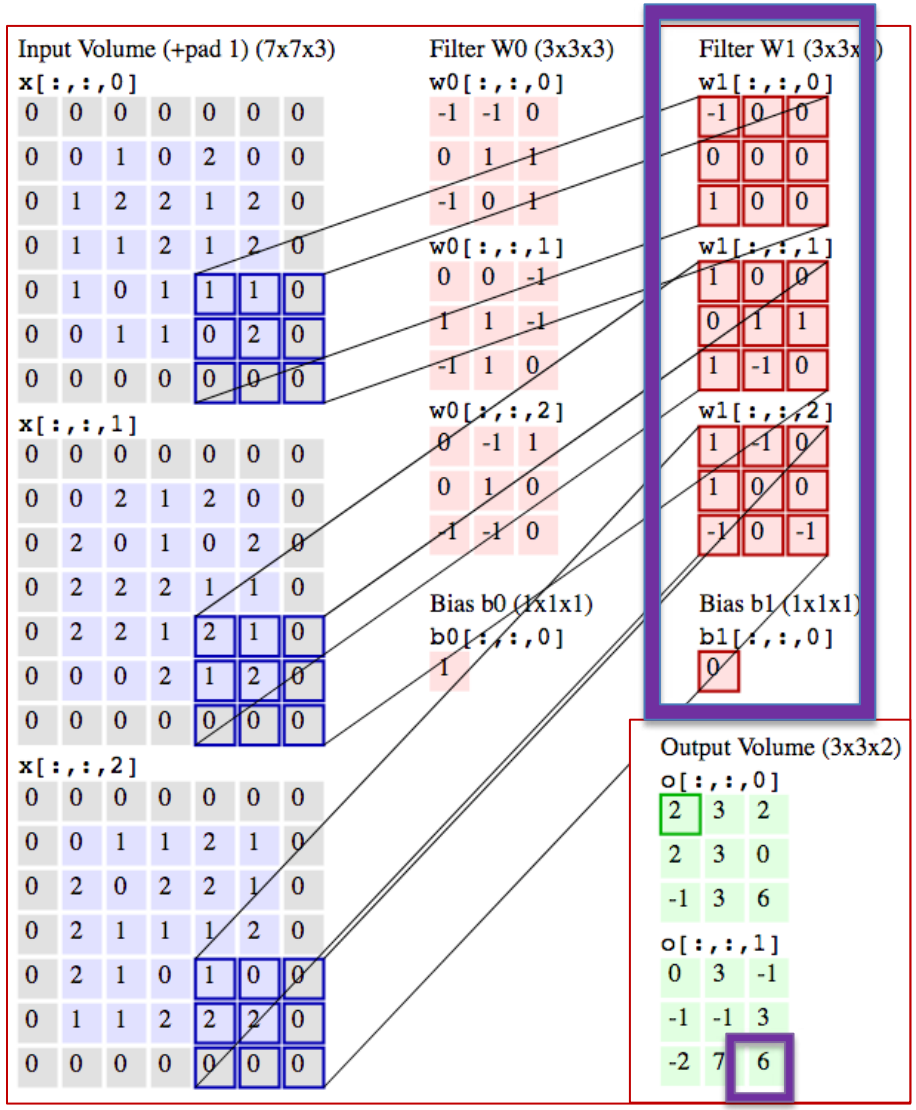
(0 x 1 + 0 x 0 + 0 x 0) + (0 x 0 + 0 x 1 + 2 x 1)+ (0 x 1 + 2 x -1 + 0 x 0) = 0

(0 x 1 + 0 x -1 + 0 x 0) + (0 x 1 + 0 x 0 + 1 x 0)+ (0 x -1 + 2 x 0 + 0 x -1) = 0

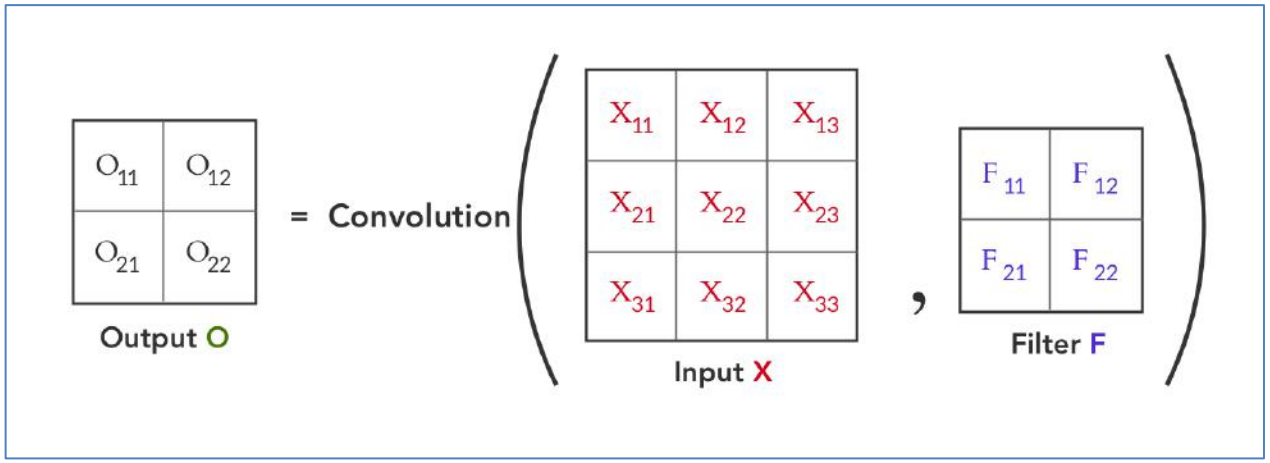
0 + b1 (0) = 0

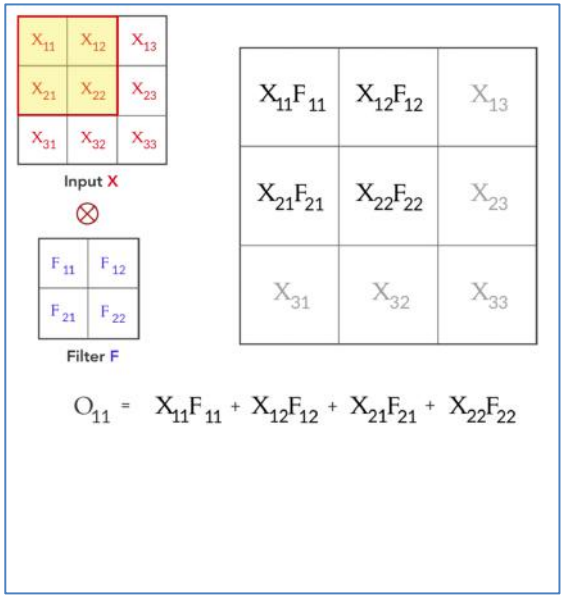
CNN Filter Operations - RGB

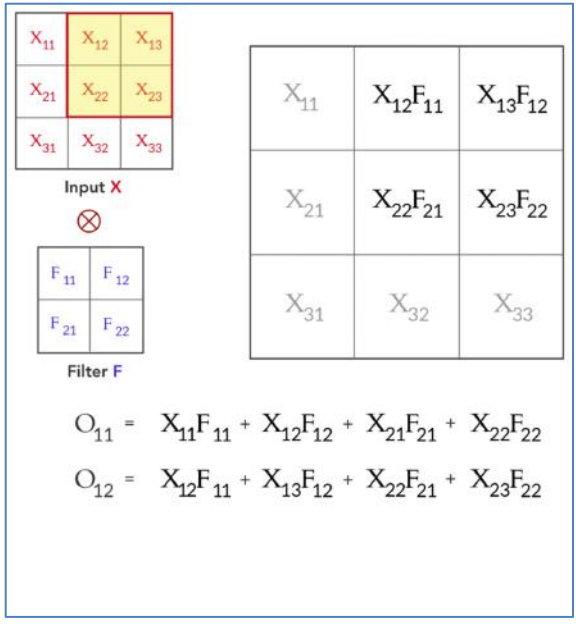


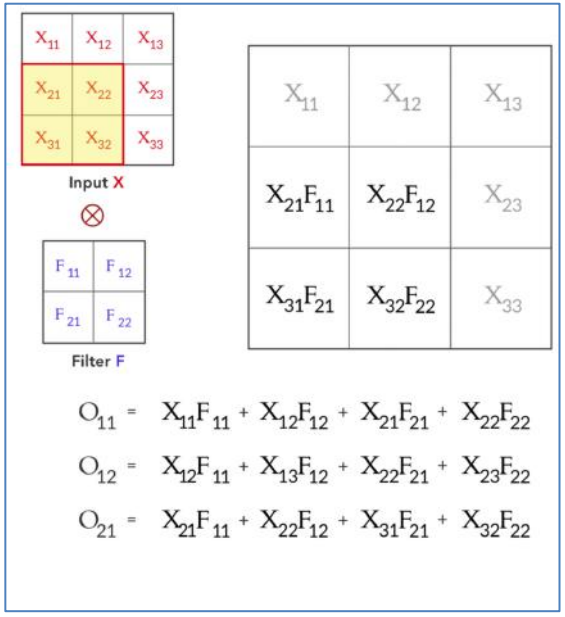


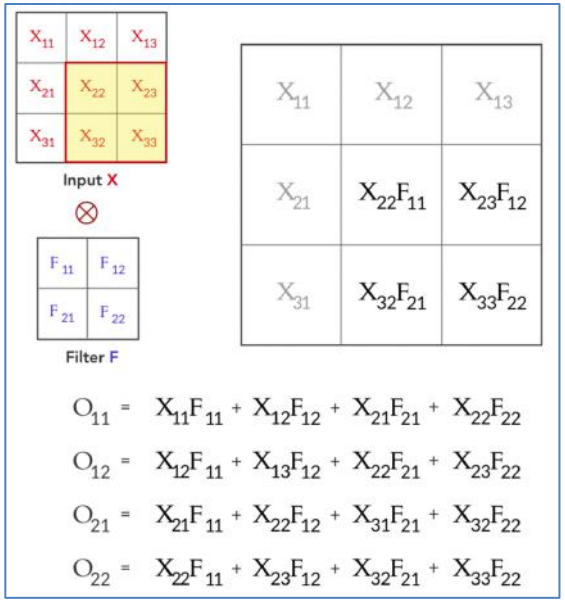
Forward path





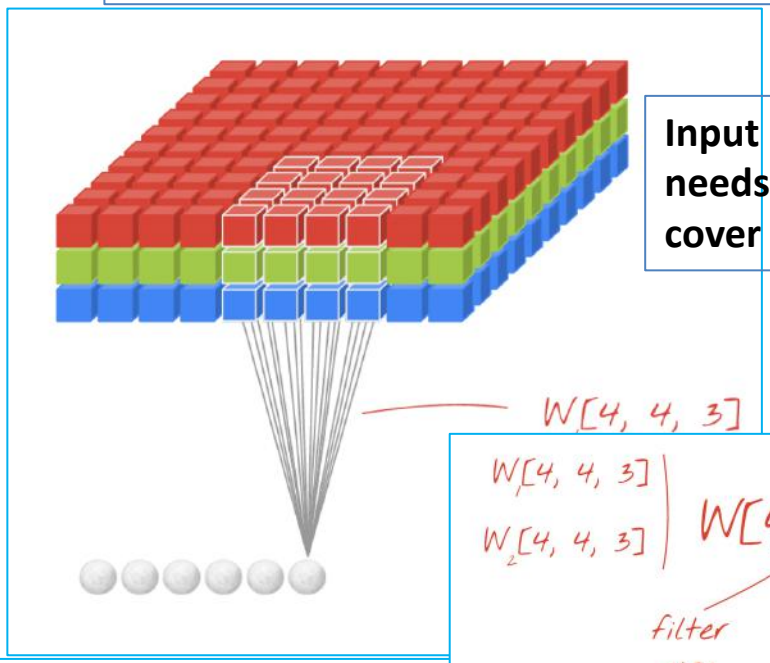




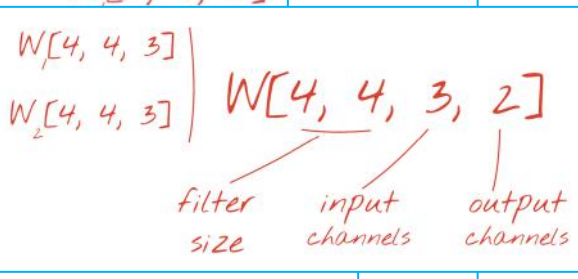


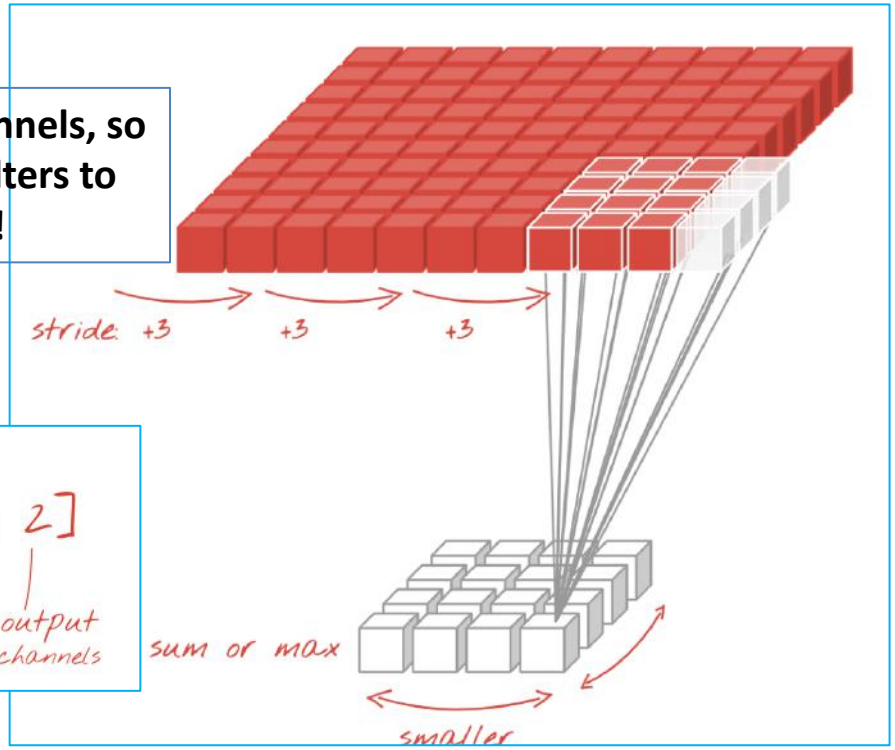
https://medium.com/@pavisj/convolutions-and-backpropagations-46026a8f5d2c

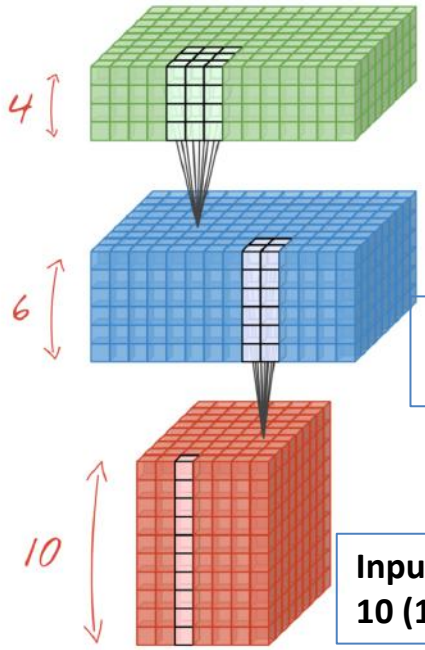
Multiple channels in inputs (outputs from previous convolution steps)



Input has 3 channels, so needs 3 (4x4) filters to cover the input!







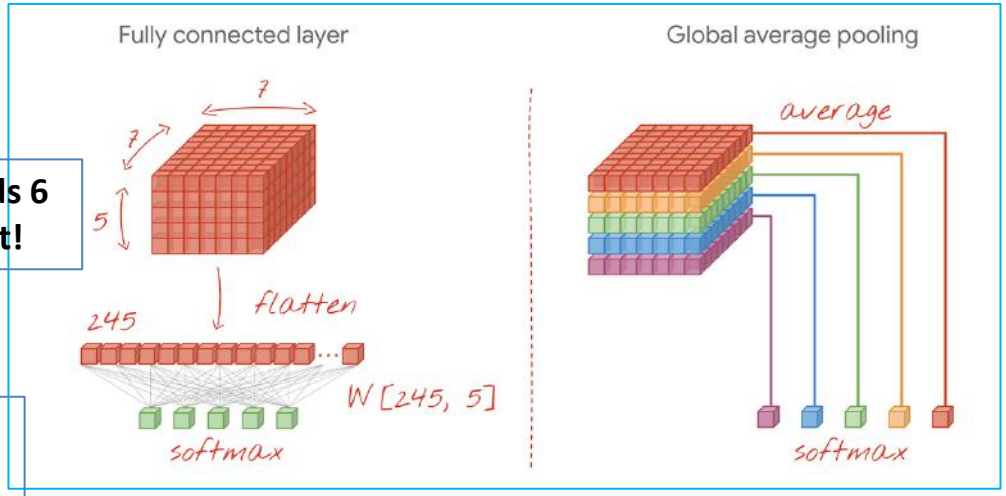
W1[3, 3, 4, 6]

W2[2, 2, 6, 10]

Input has 6 channels, so needs 6 (2x2) filters to cover the input!

W2[1, 1, 10, ...]
stride 2

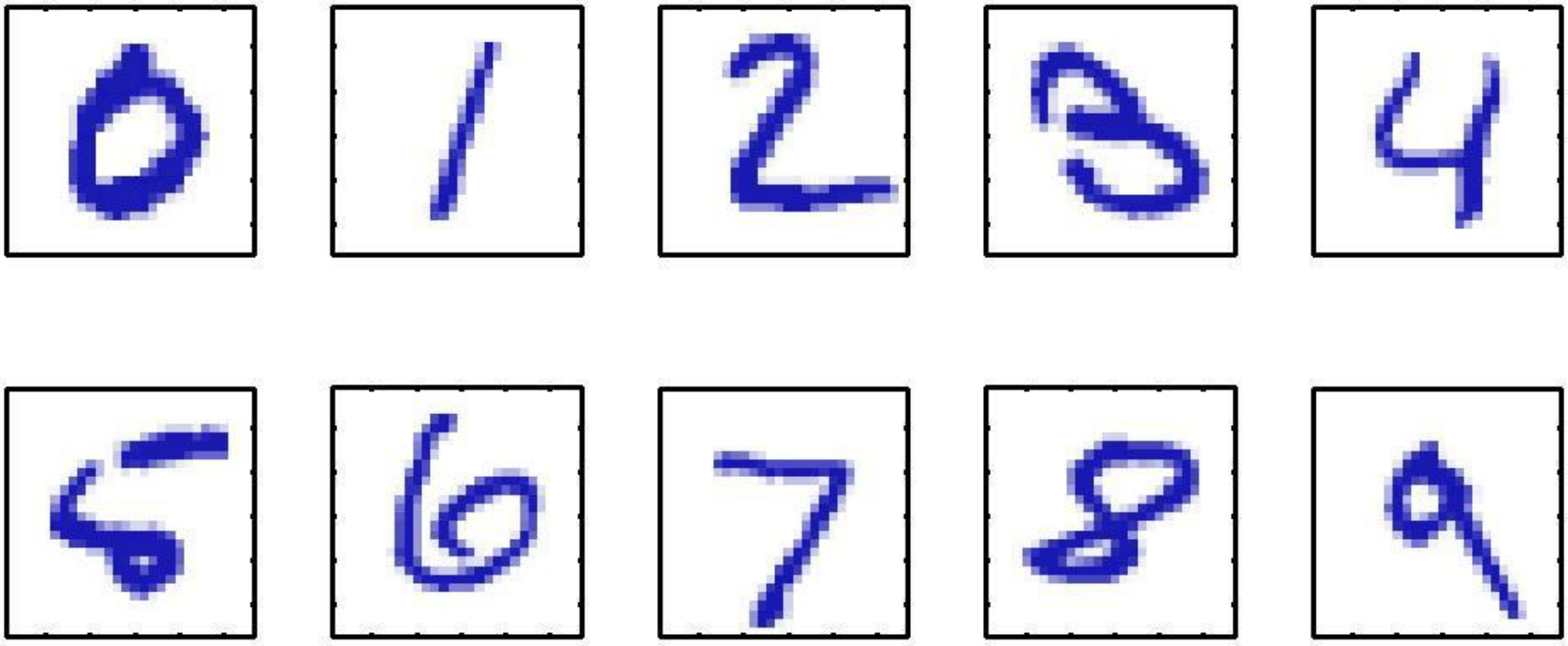
Input has 10 channels, so needs 10 (1x1) filters to cover the input!



Example: how can computer see images?

Handwritten Digit Recognition (MNIST data set)

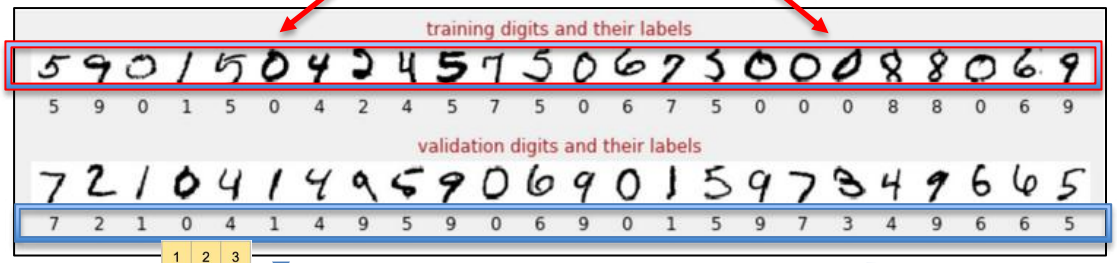
The **MNIST database** (*Modified National Institute of Standards and Technology database*) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning.[4][5] It was created by "re-mixing" the samples from NIST's original datasets. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels. The MNIST database contains 60,000 training images and 10,000 testing images. – from Wikipedia



MNIST Example (28x28 pixels image)

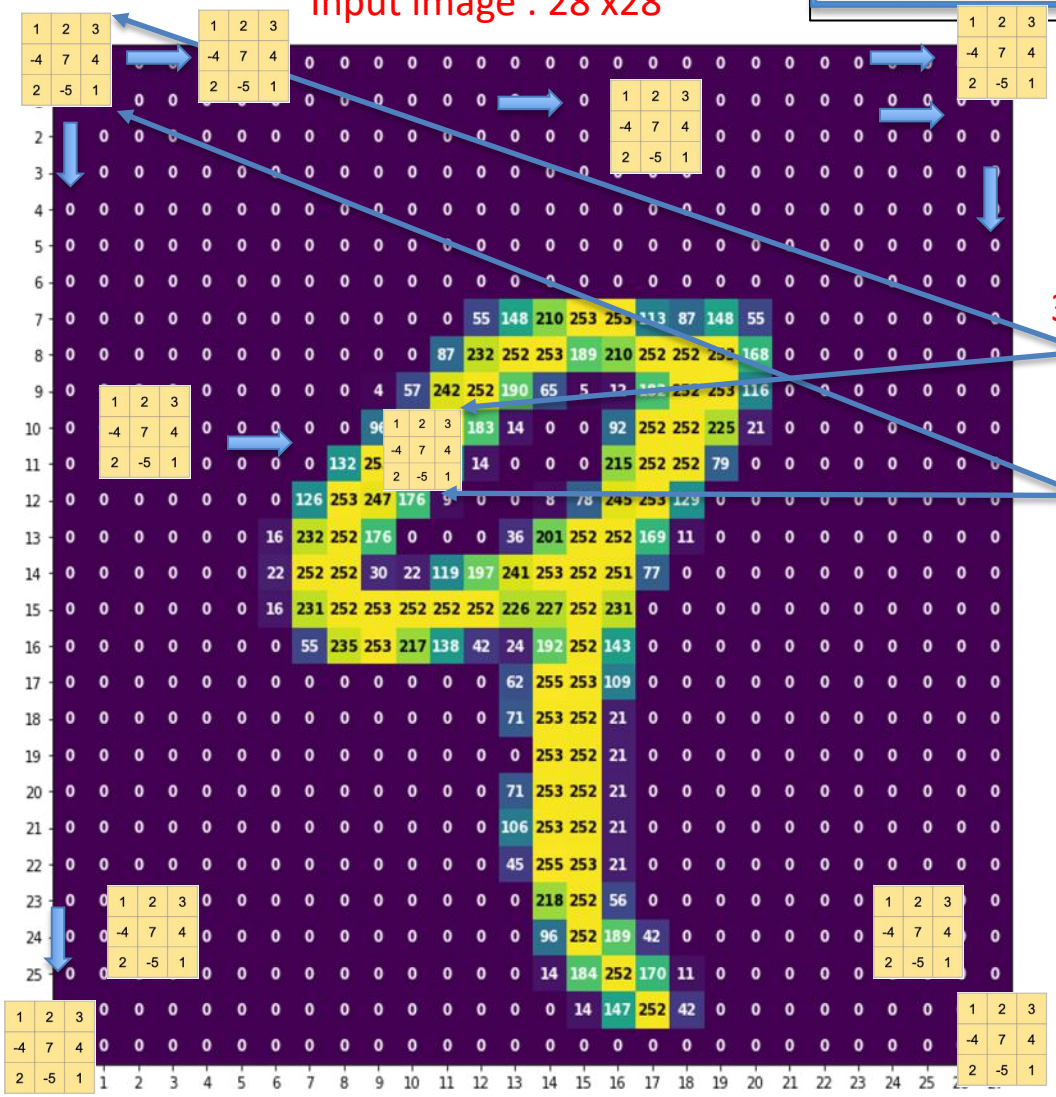
Handwritten digits in the MNIST dataset are 28x28 pixel grayscale images, use the original image (28x28 matrix).

Image : input



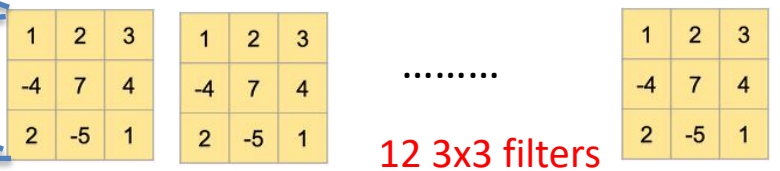
Labels : target

Input image : 28 x28

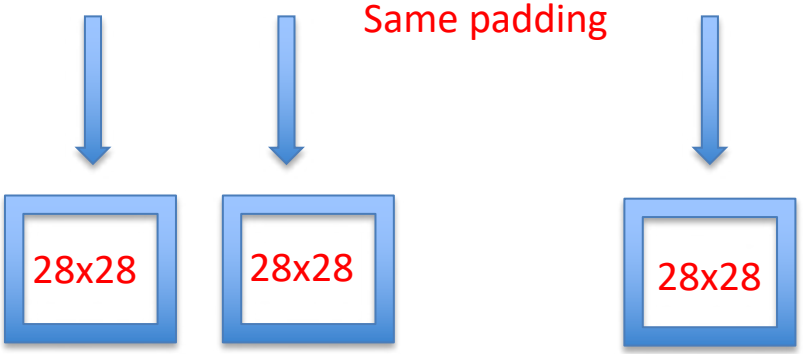


12 filters, each one acts on an image until it is fully covered.

3 x 3 filter (kernel)



12 3x3 filters
Same padding

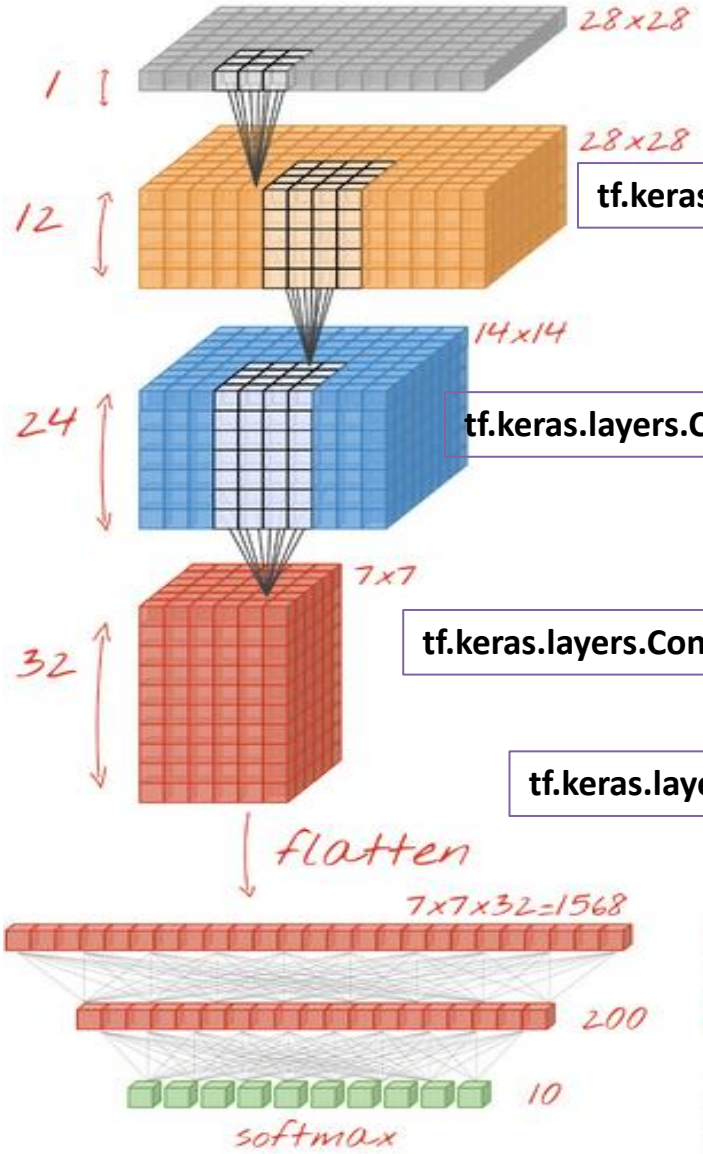


12 Output features : each 28 x28 matrix

Convolution NN : MNIST 28x28 Pixels

[FW, FH, C, H] = [filter size ? X ?, input channel (Depth), output channel (no. of filter)]

tf.keras.layers.Reshape(input_shape=(28*28,), target_shape=(28, 28, 1))



Convolutional 3x3 filters=12

$W_1[3, 3, 1, 12]$

tf.keras.layers.Conv2D(kernel_size=3, filters=12, padding='same', activation='relu')

Convolutional 6x6 filters=24

$W_2[6, 6, 12, 24]$ stride 2

tf.keras.layers.Conv2D(kernel_size=6, filters=24, padding='same', activation='relu',strides=2)

Convolutional 6x6 filters=32

$W_3[6, 6, 24, 32]$ stride 2

tf.keras.layers.Conv2D(kernel_size=6, filters=32, padding='same', activation='relu', strides=2)

tf.keras.layers.Flatten()

https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#0

Dense layer

$W_4[1568, 200]$

tf.keras.layers.Dense(200, activation='relu')

Softmax dense layer

$W_5[200, 10]$

tf.keras.layers.Dense(10, activation='softmax')

CNN : example – MNIST

There are three main types of layers in a CNN: Convolutional Layer, Pooling Layer, and Fully-Connected Layer.

- ✓ INPUT [28x28x1] will hold the raw pixel values of the image, in this case an image of width 28, height 28 in grey scale (0-255)
- ✓ Apply twelve (12) 3x3 filters to one layer (channel) - W[3,3,Channel=1,number of filters=12]
- ✓ CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume of [28x28x12] when we use 12 filters. Volume size = [28 width x height = 28 , 12 channels (depth)]
- ✓ RELU layer will apply an elementwise activation function, such as the $max(0,x)$
- ✓ Apply 24 6x6 filters to 12 layers (channel) with stride 2 - W[6,6,12,24] stride 2
- ✓ This may result in volume of [14x14x24] when we use 24 filters, stride 2.
- ✓ RELU layer will apply an elementwise activation function, such as the $max(0,x)$
- ✓ Apply 32 6x6 filters to 24 layers (channel) with stride 2 - W[6,6,24,32] stride 2
- ✓ This may result in volume of [7x7x32] when we use 32 filters, stride 2.
- ✓ RELU layer will apply an elementwise activation function, such as the $max(0,x)$
- ✓ Flatten the volume (tensor) to be a vector of 7x6x32=1568 elements
- ✓ Use 200 neurons with the input vector, Fully connected layer, W [1558, 200], output=1 x 200
- ✓ FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score.
- ✓ In this way, ConvNets transform the original image layer by layer from the original pixel values to the final class scores.

```
import tensorflow as tf
print(tf.__version__)

import datetime, os

%load_ext tensorboard

import numpy as np
from tensorflow import keras
from tensorflow.keras import layers

# Model / data parameters
num_classes = 10
input_shape = (28, 28, 1)

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")

# Make sure images have shape (28, 28, 1)
x_train = x_train.reshape([-1,28,28,1])
x_test = x_test.reshape([-1,28,28,1])
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

https://github.com/keras-team/keras-io/blob/master/examples/vision/mnist_convnet.py

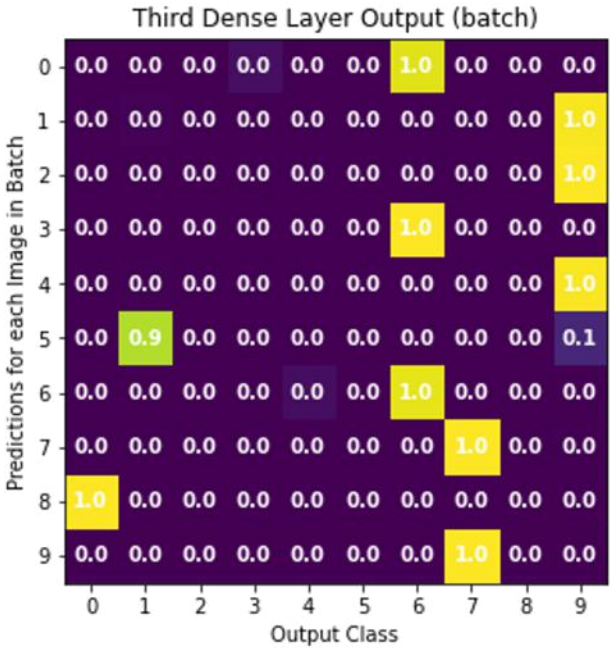
https://github.com/keras-team/keras-io/blob/master/examples/vision/mnist_convnet.py

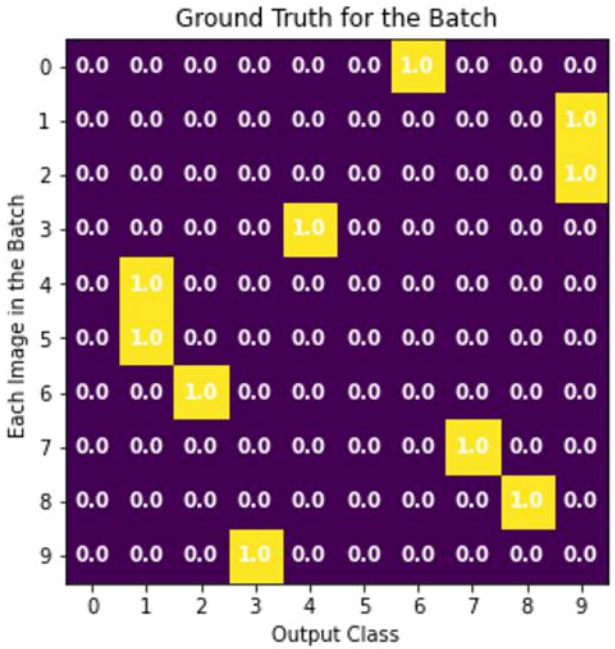
```
## Build the model
model = keras.Sequential(
  [
     keras.Input(shape=input_shape),
     layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
     layers.MaxPooling2D(pool_size=(2, 2)),
     layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
     layers.MaxPooling2D(pool_size=(2, 2)),
     layers.Flatten(),
     layers.Dropout(0.5),
     layers.Dense(num_classes, activation="softmax"),
  ]
)

model.summary()

## Train the model
batch_size = 128
epochs = 5

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1,
callbacks=[tensorboard_callback])

## Evaluate the trained model
%tensorboard --logdir logs
```

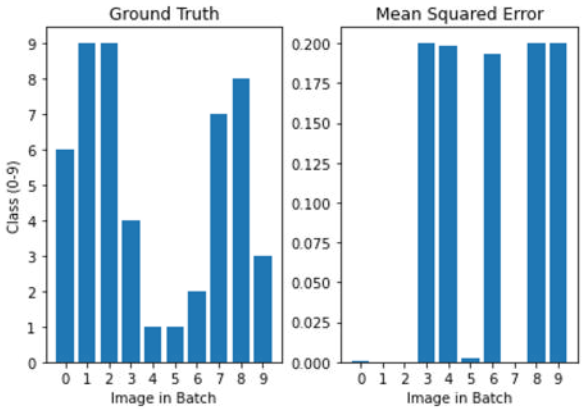
TensorFlow 2.0 : CNN for CIFAR10

Evaluation

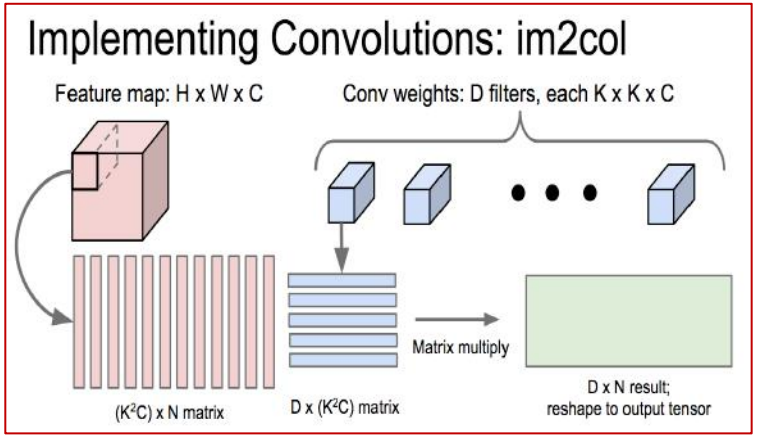
```
tf.losses.mean_squared_error(y_true_tf, y_pred_tf)
```



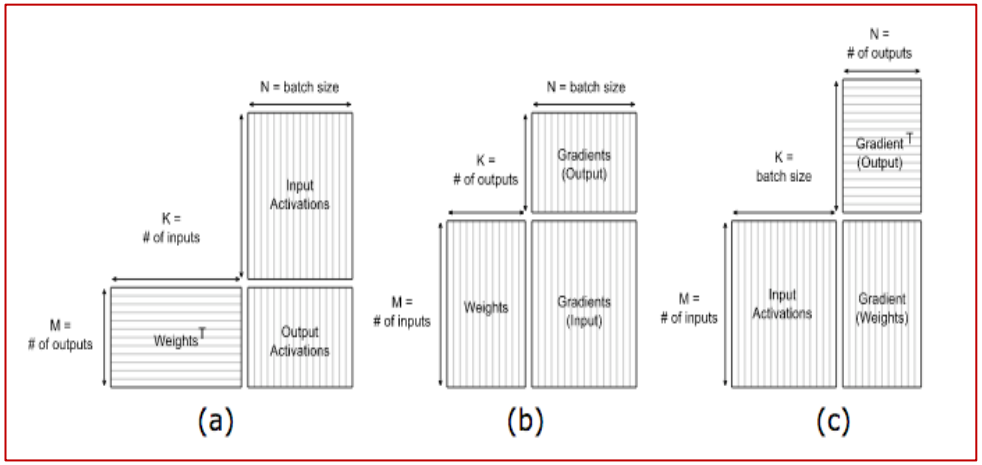




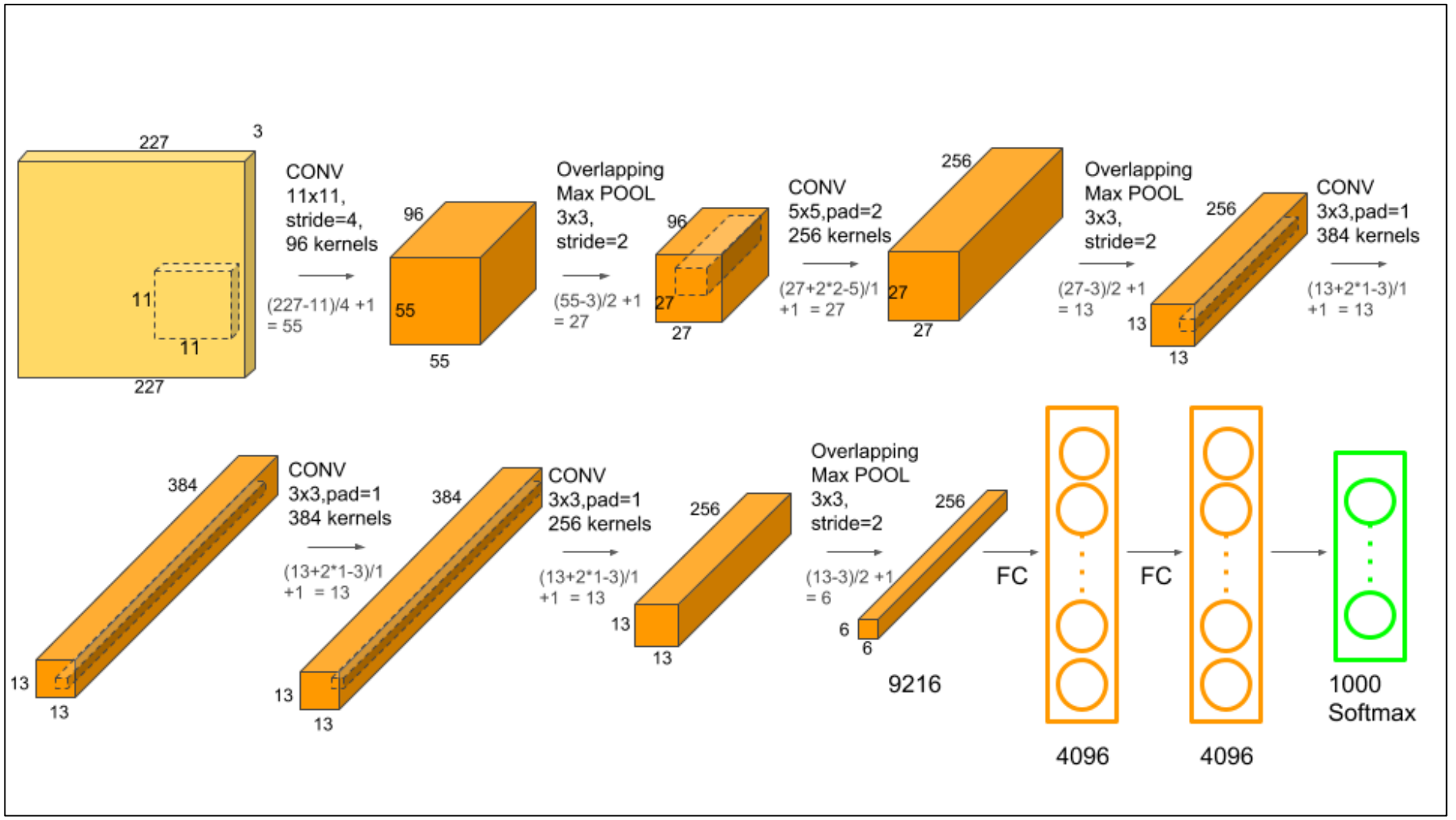
$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$



Matrix Multiplication : MM it is !!



AlexNet :Number of Parameters



https://www.learnopencv.com/number-of-parameters-and-tensor-sizes-in-convolutional-neural-network/

Number of Parameters

- ✓ **Input**: Color images of size 227x227x3. The AlexNet paper mentions the input size of 224×224 but that is a typo in the paper.
- ✓ **Conv-1**: The first convolutional layer consists of 96 kernels of size 11×11 applied with a stride of 4 and padding of 0.
- ✓ **MaxPool-1**: The maxpool layer following Conv-1 consists of pooling size of 3×3 and stride 2.
- ✓ **Conv-2**: The second conv layer consists of 256 kernels of size 5×5 applied with a stride of 1 and padding of 2.
- ✓ **MaxPool-2**: The maxpool layer following Conv-2 consists of pooling size of 3×3 and a stride of 2.
- ✓ **Conv-3**: The third conv layer consists of 384 kernels of size 3×3 applied with a stride of 1 and padding of 1.
- ✓ **Conv-4**: The fourth conv layer has the same structure as the third conv layer. It consists of 384 kernels of size 3×3 applied with a stride of 1 and padding of 1.
- ✓ **Conv-5**: The fifth conv layer consists of 256 kernels of size 3×3 applied with a stride of 1 and padding of 1.
- ✓ **MaxPool-3**: The maxpool layer following Conv-5 consists of pooling size of 3×3 and a stride of 2.
- ✓ **FC-1**: The first fully connected layer has 4096 neurons.
- ✓ **FC-2**: The second fully connected layer has 4096 neurons.
- ✓ **FC-3**: The third fully connected layer has 1000 neurons

https://www.learnopencv.com/number-of-parameters-and-tensor-sizes-in-convolutional-neural-network/

Number of Parameters

Size of the Output Tensor (Image) of a Conv Layer

O = Size (width) of output image.

I = Size (width) of input image.

K = Size (width) of kernels used in the Conv Layer

N = Number of kernels.

S = Stride of the convolution operation.

P = Padding.

The size (O) of the output image is given by

$$O = \frac{227 - 11 + 2 \times 0}{4} + 1 = 55$$

$$O = \frac{I - K + 2P}{S} + 1$$

Number of Parameters of a Conv Layer

W_c = Number of weights of the Conv Layer.

B_c = Number of biases of the Conv Layer.

P_c = Number of parameters of the Conv Layer.

K = Size (width) of kernels used in the Conv Layer.

N = Number of kernels.

C = Number of channels of the input image.

$$W_c = K^2 \times C \times N$$
$$B_c = N$$
$$P_c = W_c + B_c$$

$$W_c = 11^2 \times 3 \times 96 = 34,848$$
$$B_c = 96$$
$$P_c = 34,848 + 96 = 34,944$$

Number of Parameters of a MaxPool Layer = 0

Size of Output Tensor (Image) of a MaxPool Layer

O = Size (width) of output image.

I = Size (width) of input image.

S = Stride of the convolution operation.

P_s = Pool size.

The size (O) of the output image is given by

$$O = \frac{55 - 3}{2} + 1 = 27$$

$$O = \frac{I - P_s}{S} + 1$$

https://www.learnopencv.com/number-of-parameters-and-tensor-sizes-in-convolutional-neural-network/

Number of Parameters

The total number of parameters in AlexNet is the sum of all parameters in the 5 Conv Layers + 3 FC Layers. It comes out to a whopping **62,378,344**! The table below provides a summary.

Layer Name	Tensor Size	Weights	Biases	Parameters
Input Image	227x227x3	0	0	0
Conv-1	55x55x96	34,848	96	34,944
MaxPool-1	27x27x96	0	0	0
Conv-2	27x27x256	614,400	256	614,656
MaxPool-2	13x13x256	0	0	0
Conv-3	13x13x384	884,736	384	885,120
Conv-4	13x13x384	1,327,104	384	1,327,488
Conv-5	13x13x256	884,736	256	884,992
MaxPool-3	6x6x256	0	0	0
FC-1	4096x1	37,748,736	4,096	37,752,832
FC-2	4096x1	16,777,216	4,096	16,781,312
FC-3	1000x1	4,096,000	1,000	4,097,000
Output	1000x1	0	0	0
Total				**62,378,344**

W_{ff} = Number of weights of a FC Layer which is connected to an FC Layer.
B_{ff} = Number of biases of a FC Layer which is connected to an FC Layer.
P_{ff} = Number of parameters of a FC Layer which is connected to an FC Layer.
F = Number of neurons in the FC Layer.
F_{-1} = Number of neurons in the previous FC Layer.

$$W_{ff} = F_{-1} \times F$$
$$B_{ff} = F$$
$$P_{ff} = W_{ff} + B_{ff}$$

In the above equation, $F_{-1} \times F$ is the total number of connection weights from neurons of the previous FC Layer the neurons of the current FC Layer. The total number of biases is the same as the number of neurons (F).

Example: The last fully connected layer of AlexNet is connected to an FC Layer. For this layer, $F_{-1} = 4096$ and $F = 1000$. Therefore,

$$W_{ff} = 4096 \times 1000 = 4,096,000$$
$$B_{ff} = 1,000$$
$$P_{ff} = W_{ff} + B_{ff} = 4,097,000$$

W_{cf} = Number of weights of a FC Layer which is connected to a Conv Layer.
B_{cf} = Number of biases of a FC Layer which is connected to a Conv Layer.
O = Size (width) of the output image of the previous Conv Layer.
N = Number of kernels in the previous Conv Layer.
F = Number of neurons in the FC Layer.

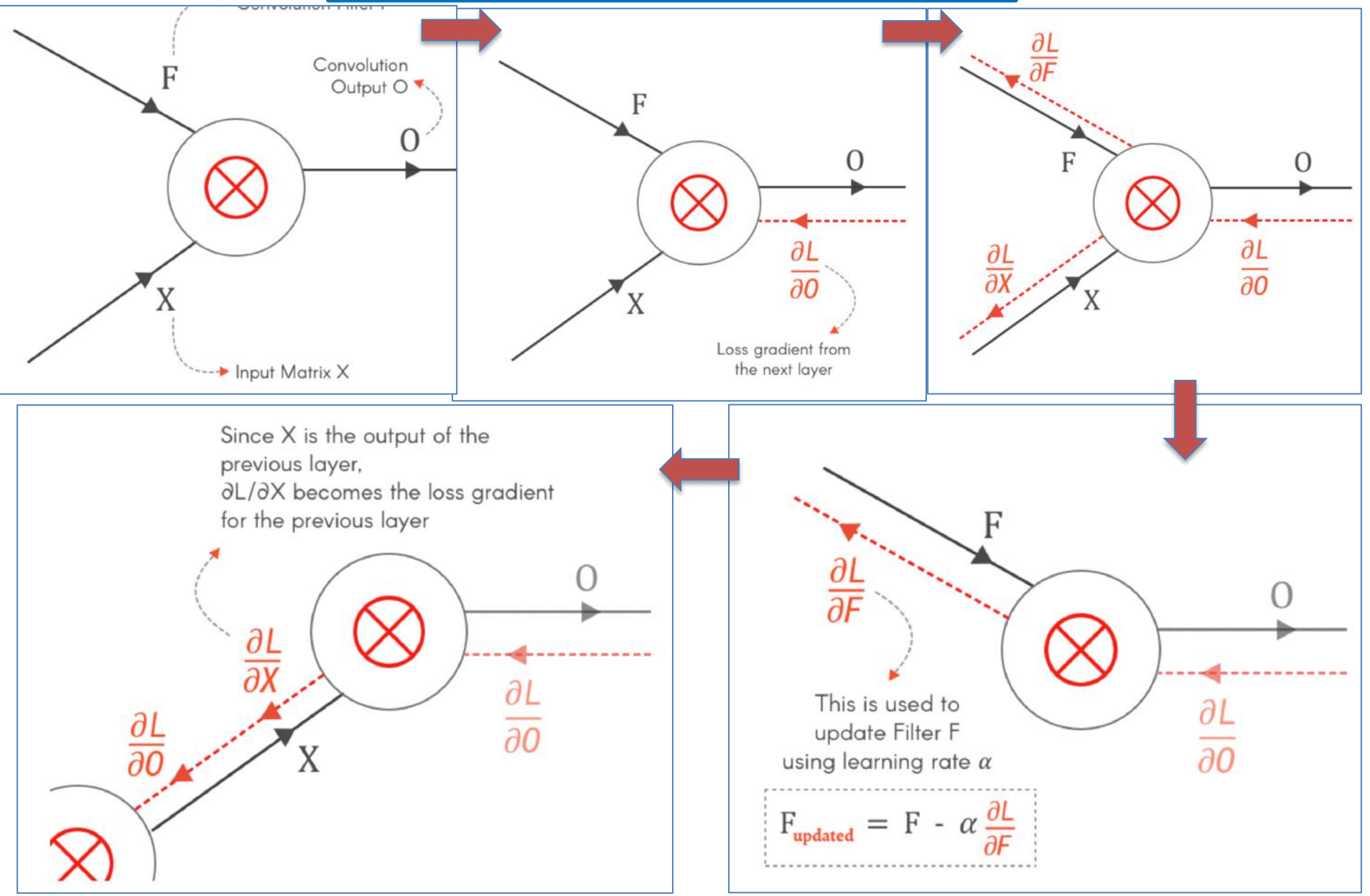
$$W_{cf} = O^2 \times N \times F$$
$$B_{cf} = F$$
$$P_{cf} = W_{cf} + B_{cf}$$

Example: The first fully connected layer of AlexNet is connected to a Conv Layer. For this layer, $O = 6$, $N = 256$ and $F = 4096$. Therefore,

$$W_{cf} = 6^2 \times 256 \times 4096 = 37,748,736$$
$$B_{cf} = 4096$$
$$P_{cf} = W_{cf} + B_{cf} = 37,752,832$$

https://www.learnopencv.com/number-of-parameters-and-tensor-sizes-in-convolutional-neural-network/

Forward path : backward path



https://medium.com/@pavisj/convolutions-and-backpropagations-46026a8f5d2c

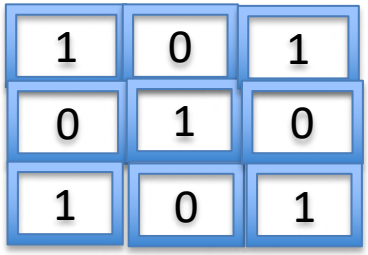
CNN : Backward Path Calculation
A Simple Example

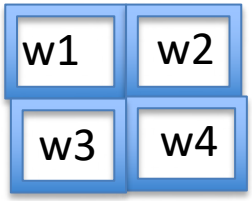
Example exercise

Compute the updated values of filter weights after backpropagation of the following problem -- 3x3 input matrix, 2x2 filter, stride 1, no padding, flatten it and use it as input connecting to a two-neuron layer, then pass the results to a softmax binary classification of two neurons.

3a) (0.5%) What is the values of the output at the end of the forward path

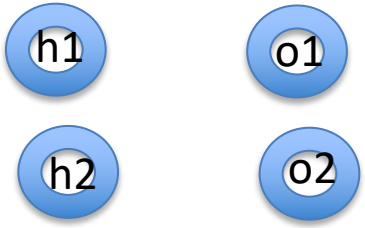
3b) (1.5%) What are the updated values of the weight and bias after backpropagation





b1 = 0.1

w1=0.9
w2=0.1
w3=0.1
w4=0.9



Apply ReLU activation function after the convolution step

Flatten the value after applying the ReLU activation function

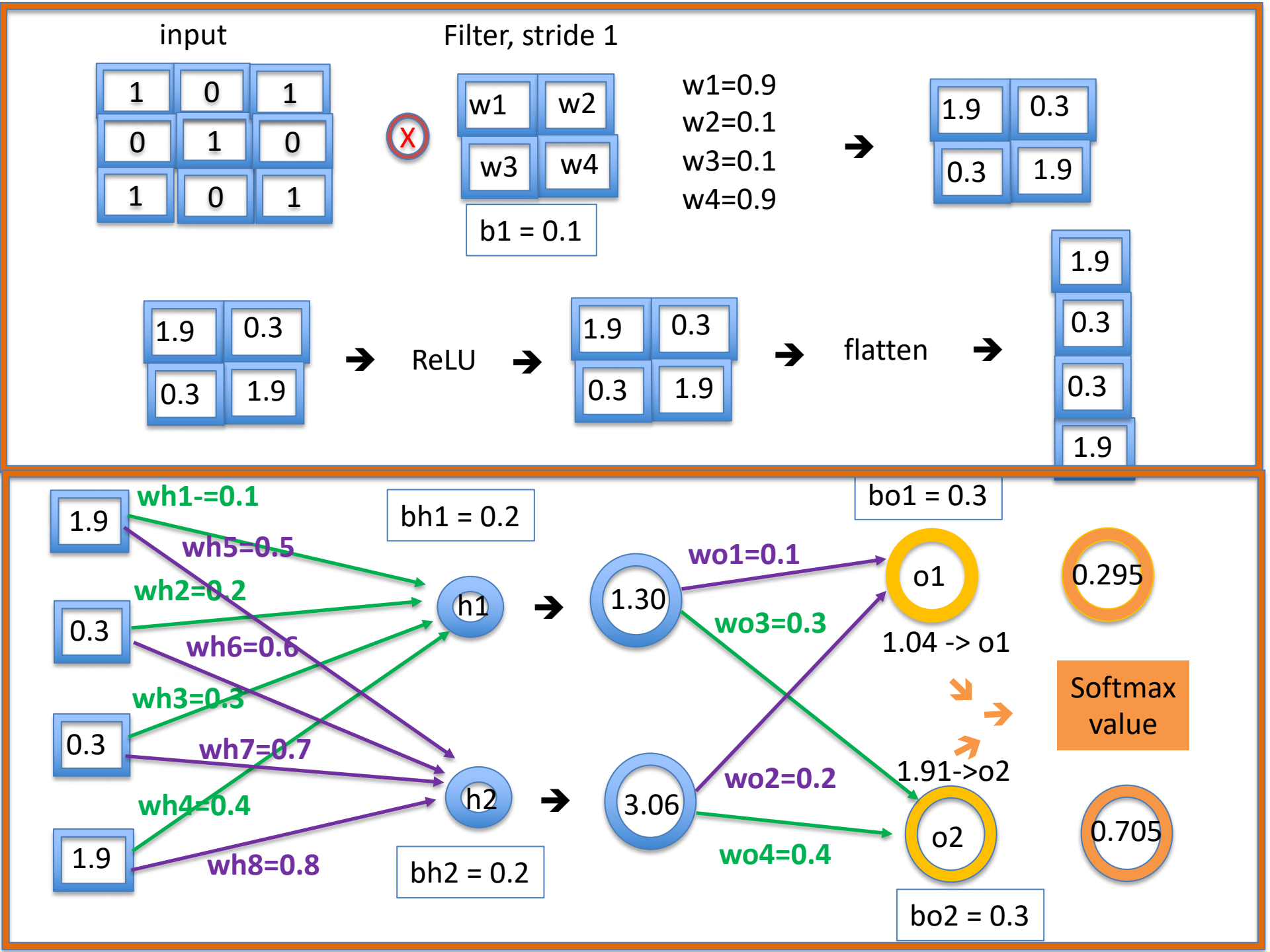
Assuming the weight of each link to this layer is w = 0.1 to 0.8, b2=0.2

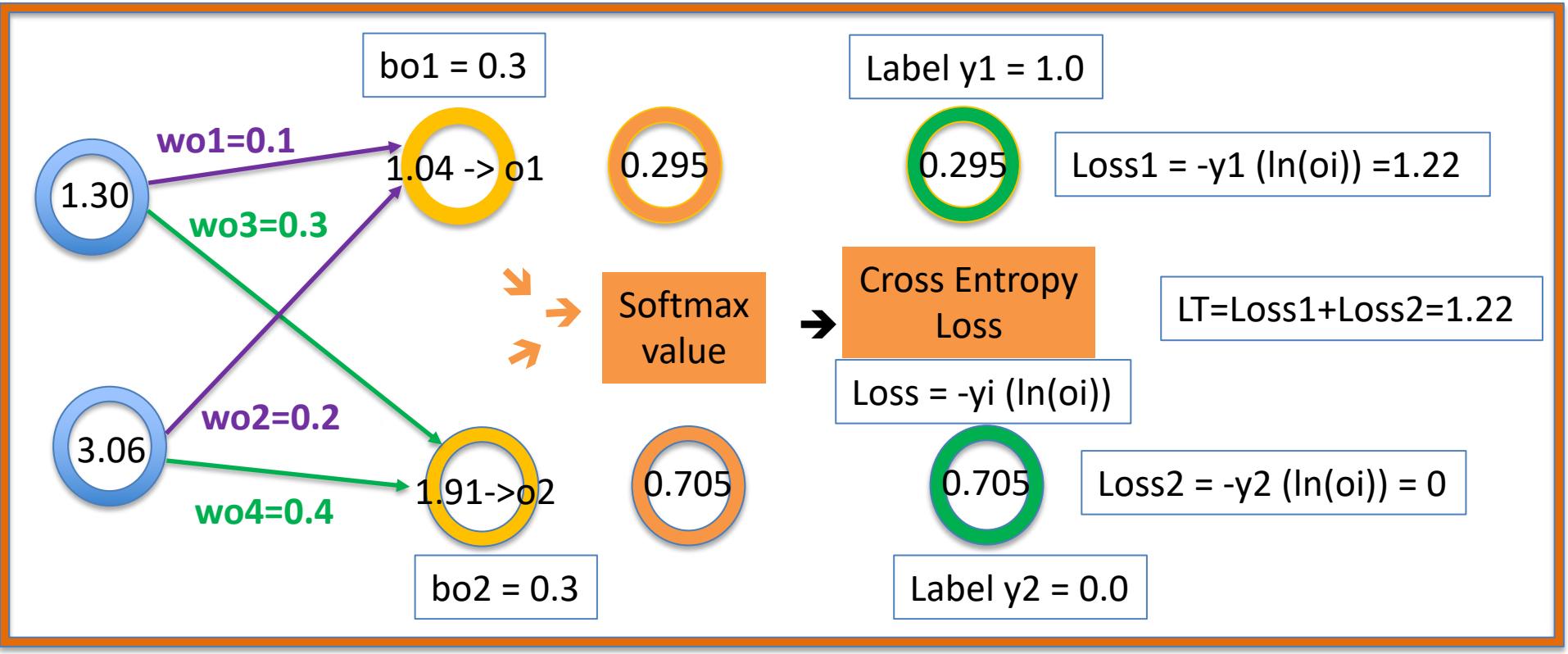
Fully connecting to a layer of two neurons

Assuming the weight of each link is w = 0.1 to 0.4, b3=0.3

Fully Connected layer of 2 softmax neurons

Labeled output of two classes are 1.0 and 0.00


input
Filter, stride 1
1 0 1
0 1 0
1 0 1
X
w1 w2
w3 w4
b1 = 0.1
w1=0.9
w2=0.1
w3=0.1
w4=0.9
1.9 0.3
0.3 1.9
1.9 0.3
0.3 1.9
ReLU
1.9 0.3
0.3 1.9
flatten
1.9
0.3
0.3
1.9
wh1-=0.1
wh5=0.5
wh2=0.2
wh6=0.6
wh3=0.3
wh7=0.7
wh4=0.4
wh8=0.8
1.9
0.3
0.3
1.9
bh1 = 0.2
h1
h2
bh2 = 0.2
1.30
3.06
wo1=0.1
wo3=0.3
wo2=0.2
wo4=0.4
bo1 = 0.3
o1
o2
bo2 = 0.3
1.04 -> o1
1.91->o2
0.295
0.705
Softmax value



CNN : Backward Path

Learning rate = n = 0.5

$$\frac{\partial S(z_i)}{\partial z_j} = \begin{cases} S(z_i) \times (1 - S(z_i)) & \text{if } i = j \\ -S(z_i) \times S(z_j) & \text{if } i \neq j \end{cases}$$

$$\frac{\partial E}{\partial Z_k} = O_k - y_k$$

weights bias

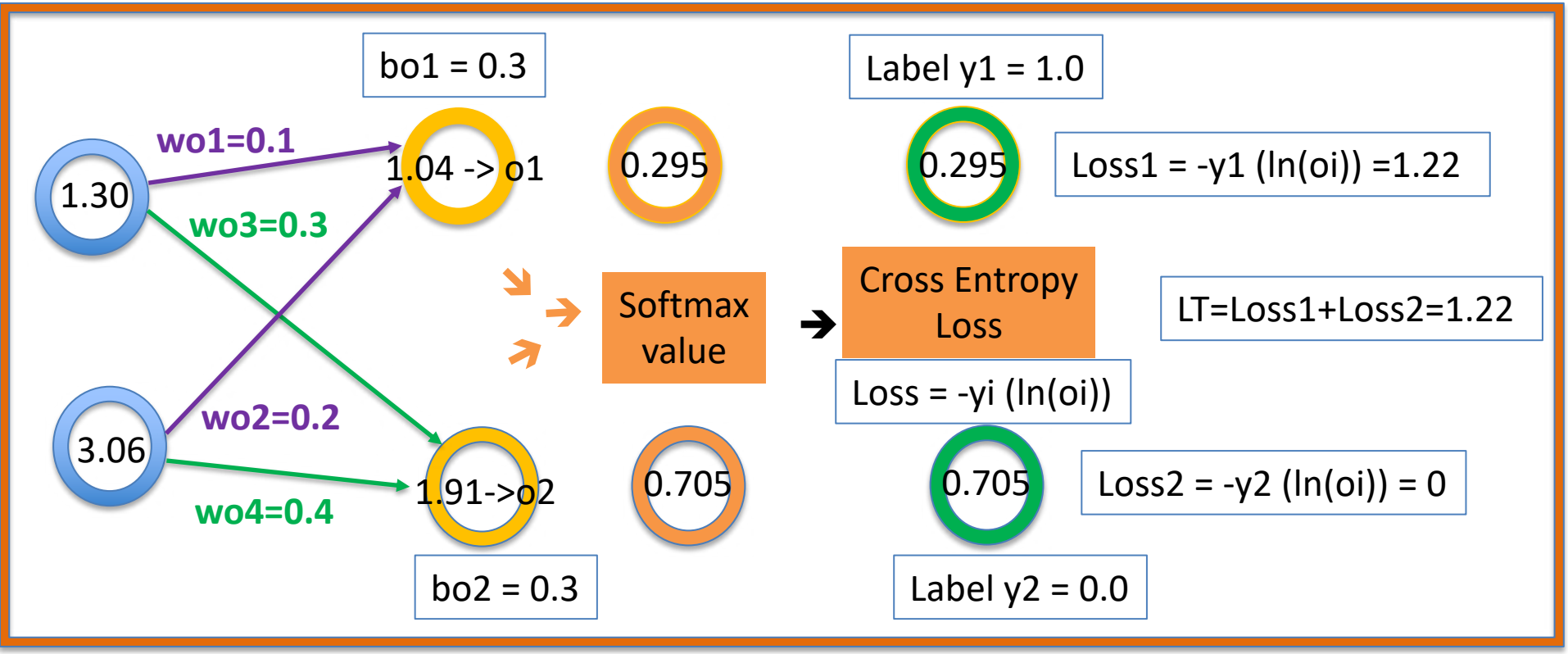
w+ = w – n * (dE/dw)
dE/dw = (dE/dS) * (dS/dw)

Softmax value

dL/dS

Cross Entropy Loss (L)

dL/dL = 1



CNN : Backward Path – jumps over Cross Entropy and Softmax all together

Learning rate = n = 0.5

$$\frac{\partial E}{\partial Z_k} = O_k - y_k$$

Z = Z(w, b)

S = S(z)

weights bias

w+ = w – n * (dE/dw)
dE/dw = (dE/dS) * (dS/dw)

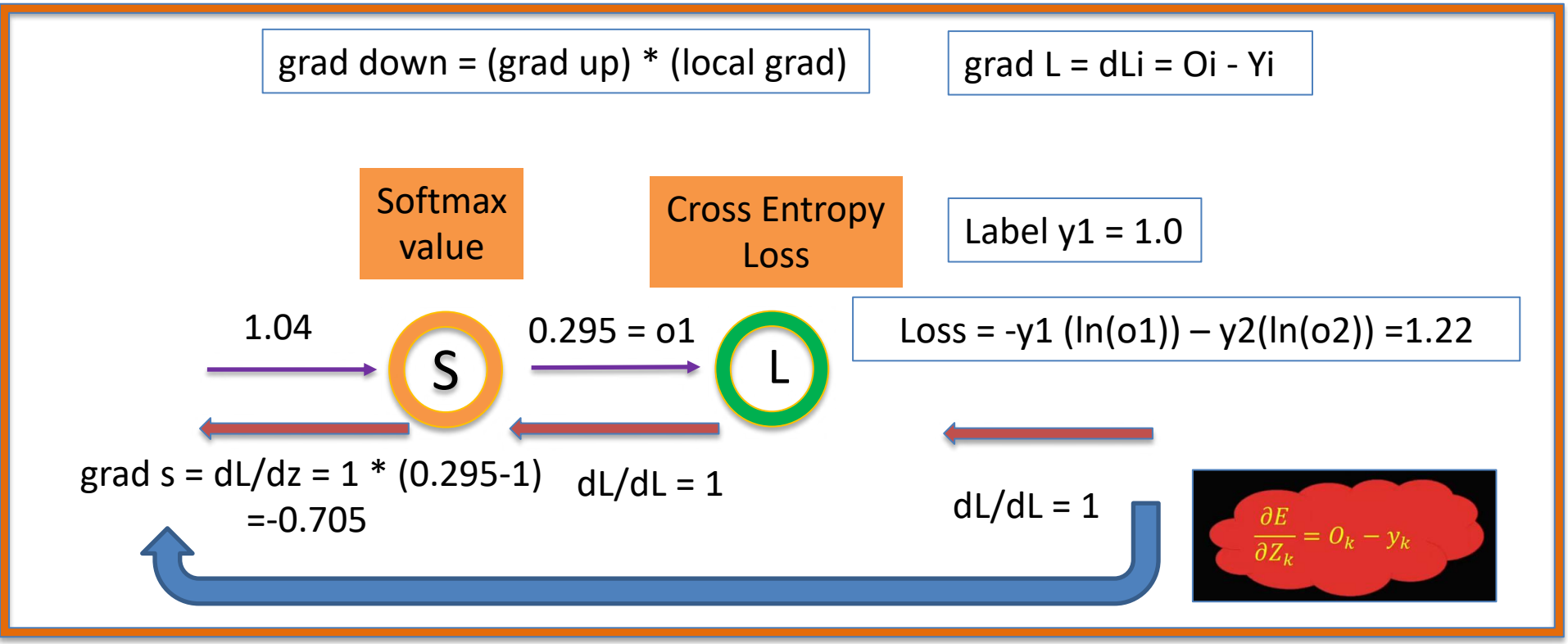
Softmax value

dL/dS

Cross Entropy Loss (L)

dL/dL = 1

Flow Back from the Total Loss to Softmax to Z (S1)



Jump from cross entropy and softmax (E , S) to (weight and bias) Z (o1), Y1 =1

E=L ; dE/dz = dL/dz = (dL/ds * ds/dz)

dE/dZ = local grad = O1 – Y1

grad S = grad down = (grad up * local grad)

= 1 * (0.295-1) = -0.705

$$\frac{\partial E}{\partial Z_k} = O_k - y_k$$

Flow Back from CCE Loss to Softmax to W and b (S1)

grad down = (grad up) * (local grad)

wo1=0.1

1.30

wo2=0.2

-0.705

3.06

-0.705

bo1 = 0.3

-0.705

+

1.04

S

Softmax value

Cross Entropy Loss

L

dL/dw ; dL/db

grad wo1=gradS * 1.30
grad wo2=gradS * 3.06
grad bo1 =-0.705

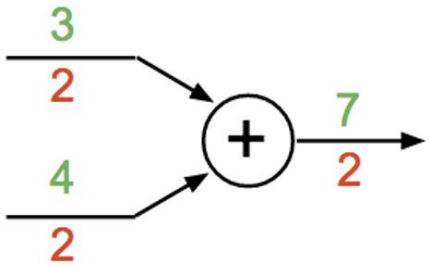
gradS = 1 * (0.295-1)
=-0.705

$$\frac{\partial E}{\partial Z_k} = O_k - y_k$$

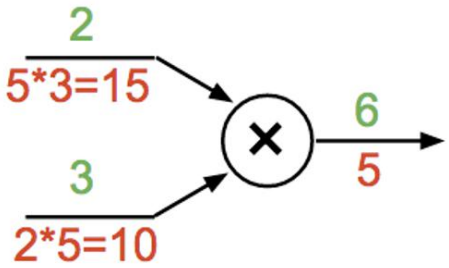
Jump from S(o1) to wo1, wo2, bo1

Same as MLP example. use multiplication operators to propagate gradient back

add gate: gradient distributor

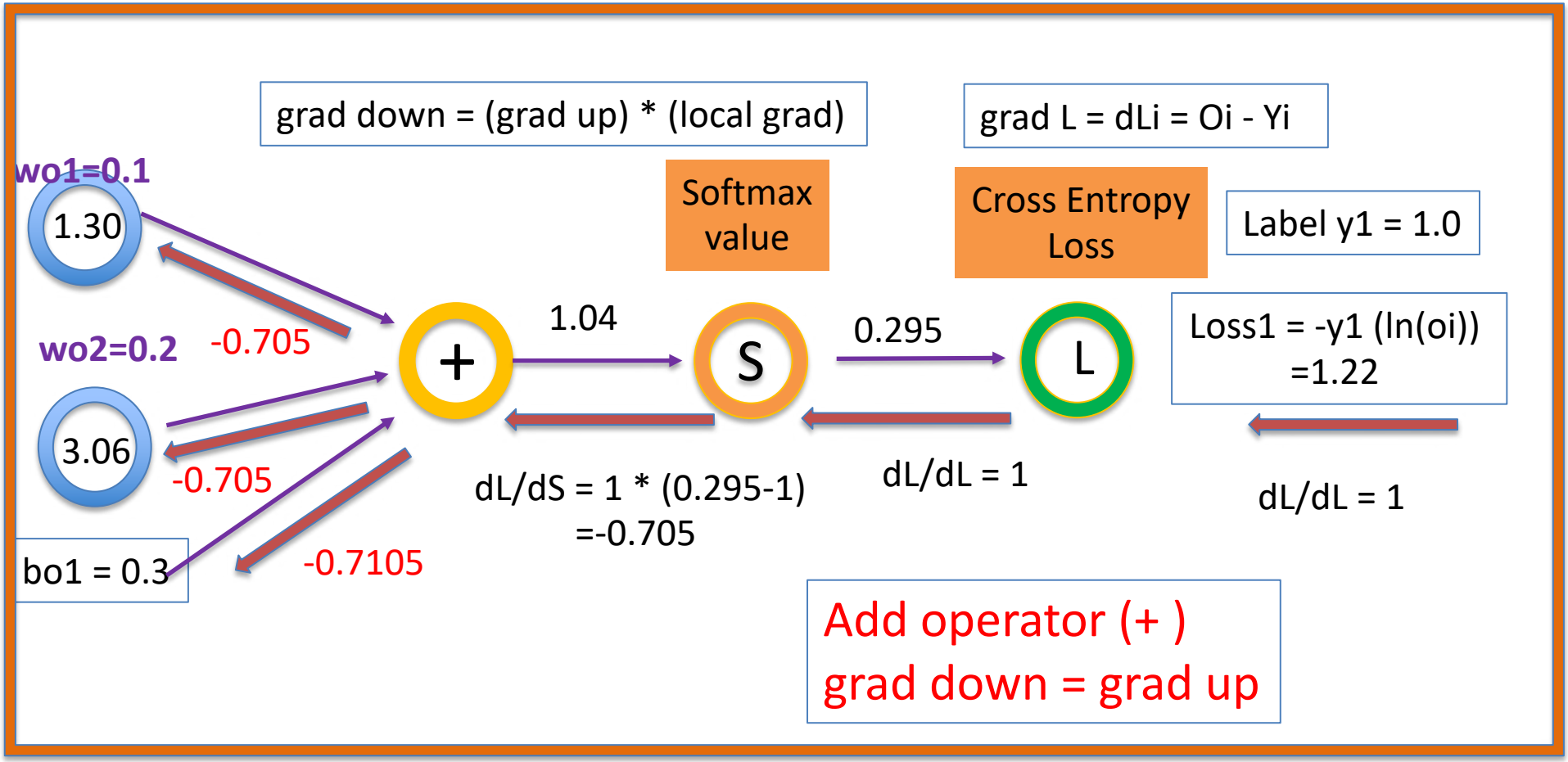


mul gate: "swap multiplier"



Flow Back from Cross Entropy and Softmax to W and b

L -> L1 -> S1 (o1) -> wo1, wo2, bo1



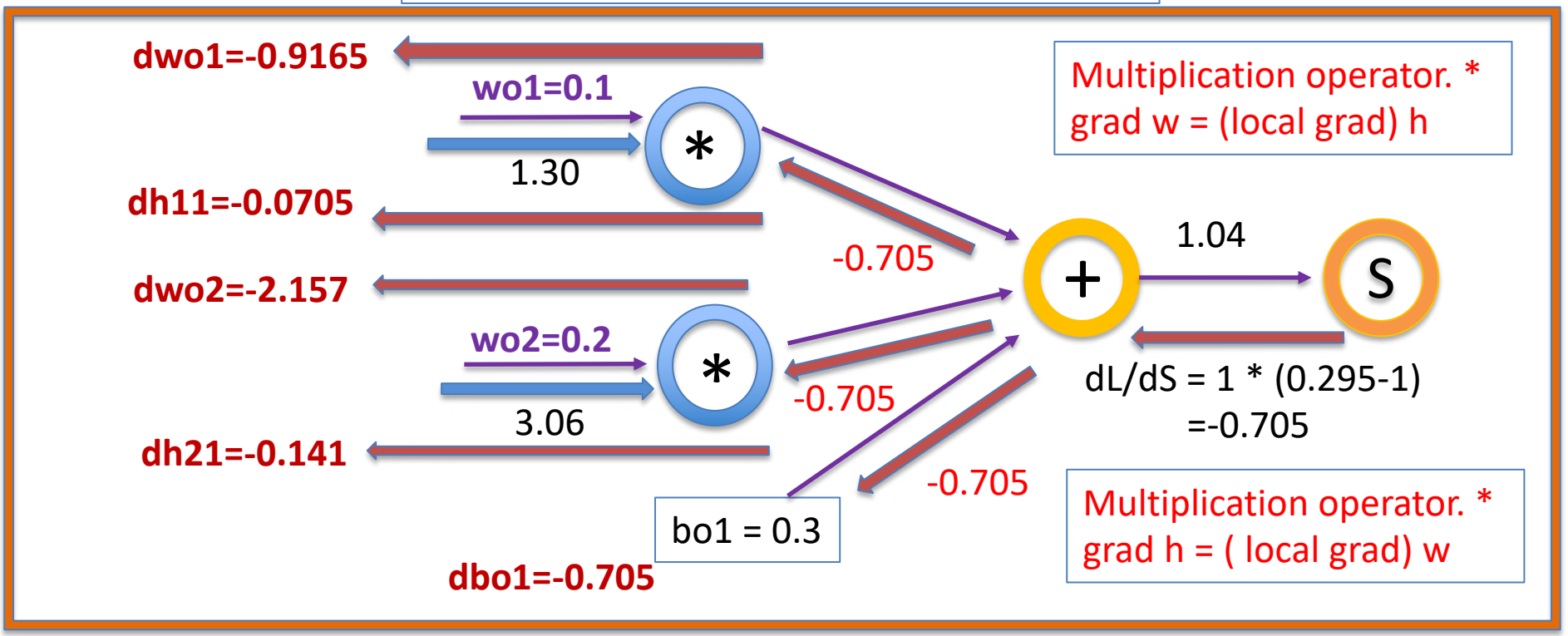
Multiplication operator. (*)

grad w = (local grad) h

Multiplication operator. (*)

grad h = (local grad) w

Update values of wo1, wo2, bo1



In here, i =1 (o1),
j = 1,2: h1, h2

d (L/dwo1) = dL/d(w11)
= -0.705*1.30 = -0.9165

d (L/dwo2) = dL/d(w12)
= -0.705*3.06= -2.1573

d (L/dbo1) = dL/d(b1)
= dL/dS = Oi-Yi=-0.705

wo1+= 0.1– (0.5)*(-0.9165) =0.558

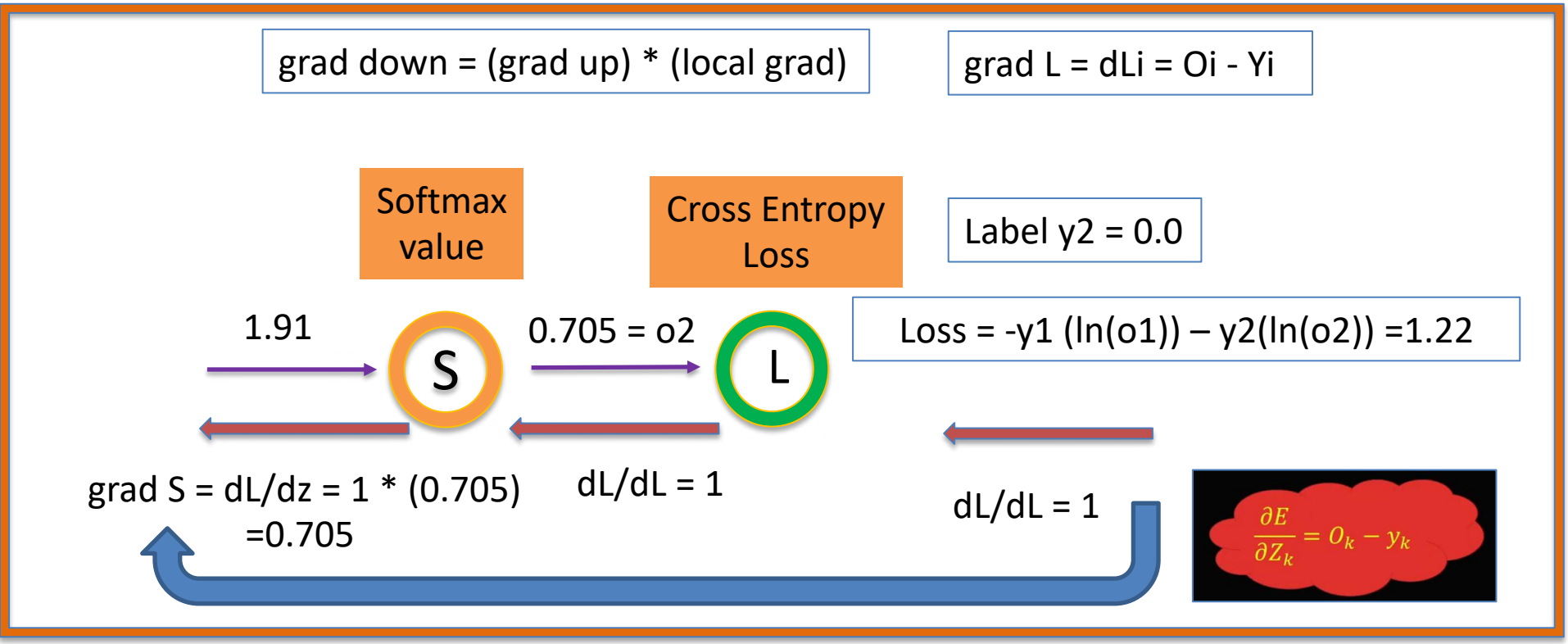
wo2+= 0.2– (0.5)*(-2.1573) =1.278

bo1+= 0.3– (0.5)*(-0.705) =0.652

dh11= (-0.0705)

dh21= (-0.141)

Flow Back from the CEE Loss to Softmax to Z : S2



Jump from cross entropy and softmax (E, S) to weight and bias Z (o2), Y2=0

E=L ; dE/dz = dL/dz = (dL/ds * ds/dz)

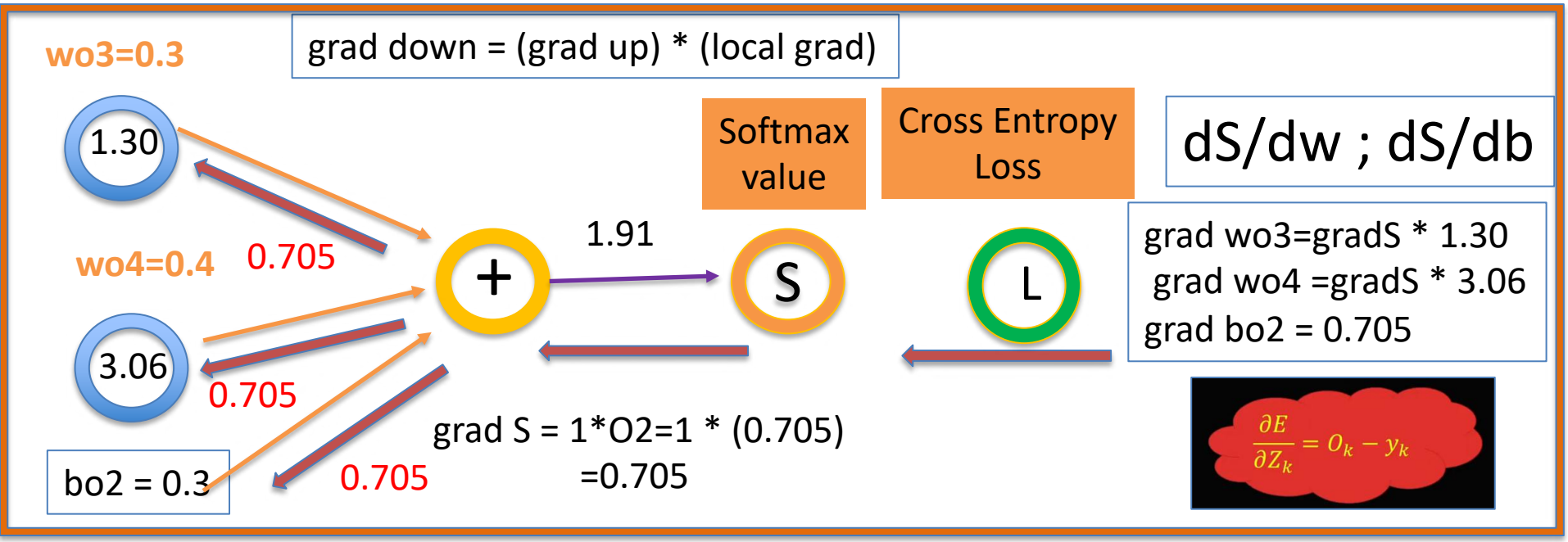
dL/dz = local grad = O2 – Y2 = O2

grad S = grad down = (grad up * local grad)

= 1 * (0.705) = 0.705

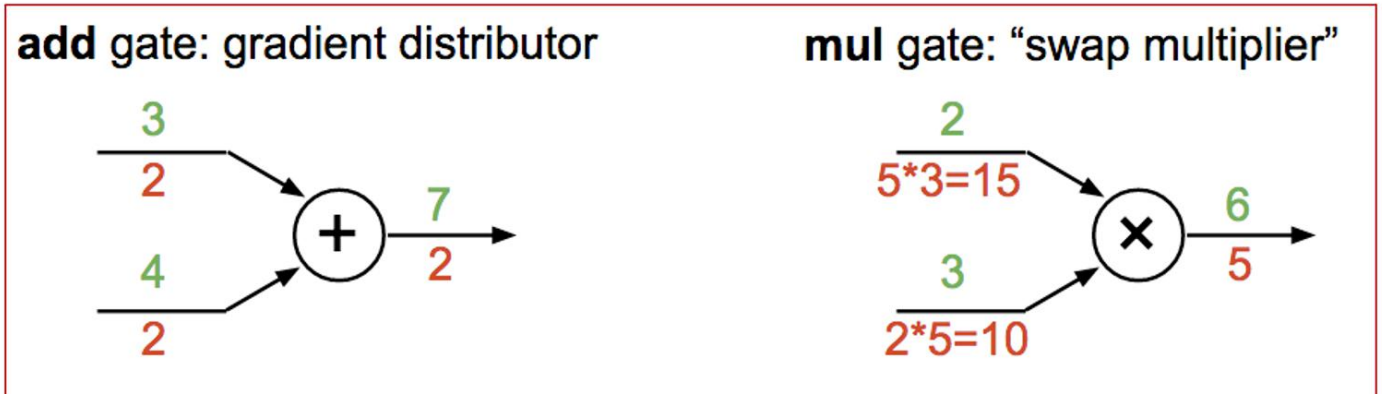
$$\frac{\partial E}{\partial Z_k} = O_k - y_k$$

Flow Back from Softmax to W and b (S2)



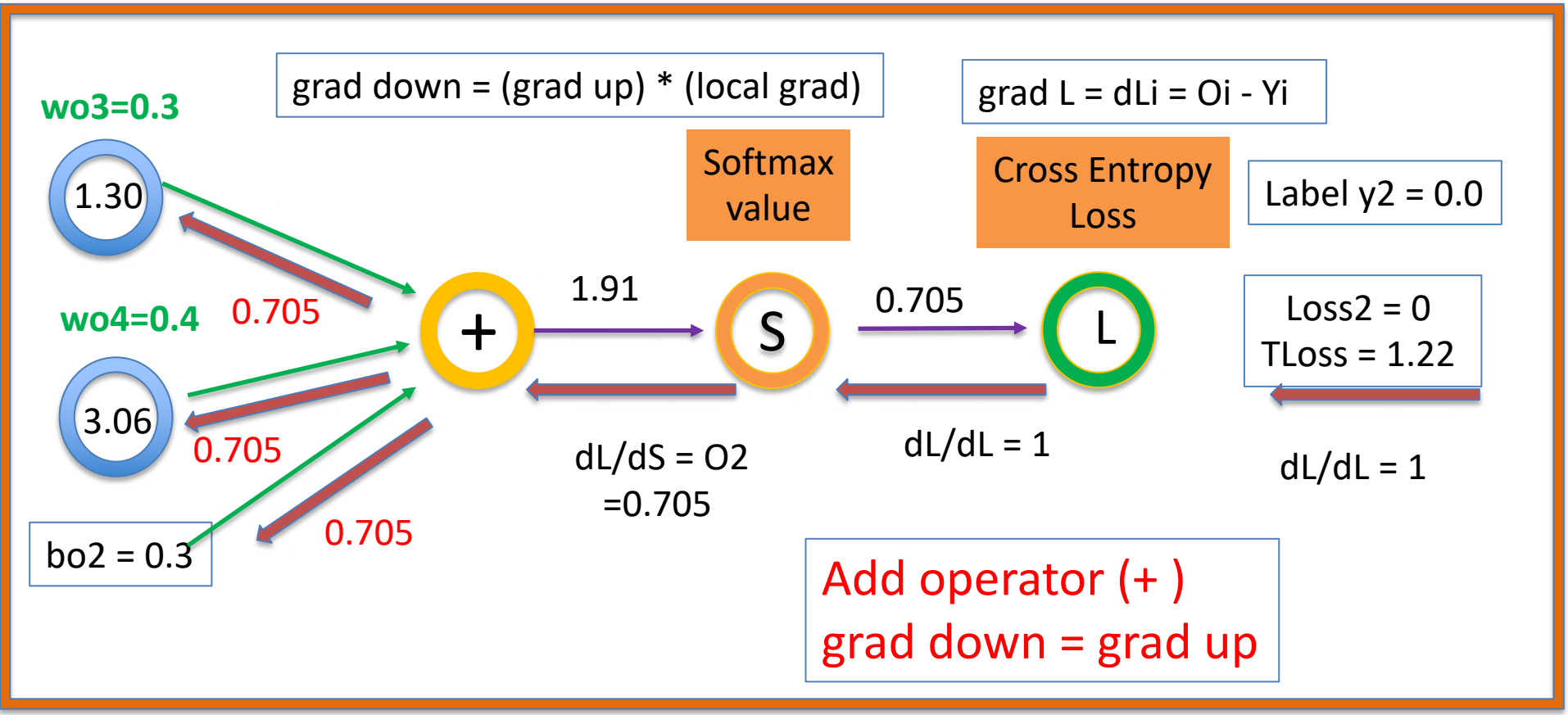
Jump from S(o2) to wo3, wo4, bo2

Same as MLP example. use multiplication operators to propagate gradient back



Flow Back from Cross Entropy and Softmax to W and b

L -> L2 -> S2 (o2) -> wo3, wo4, bo2



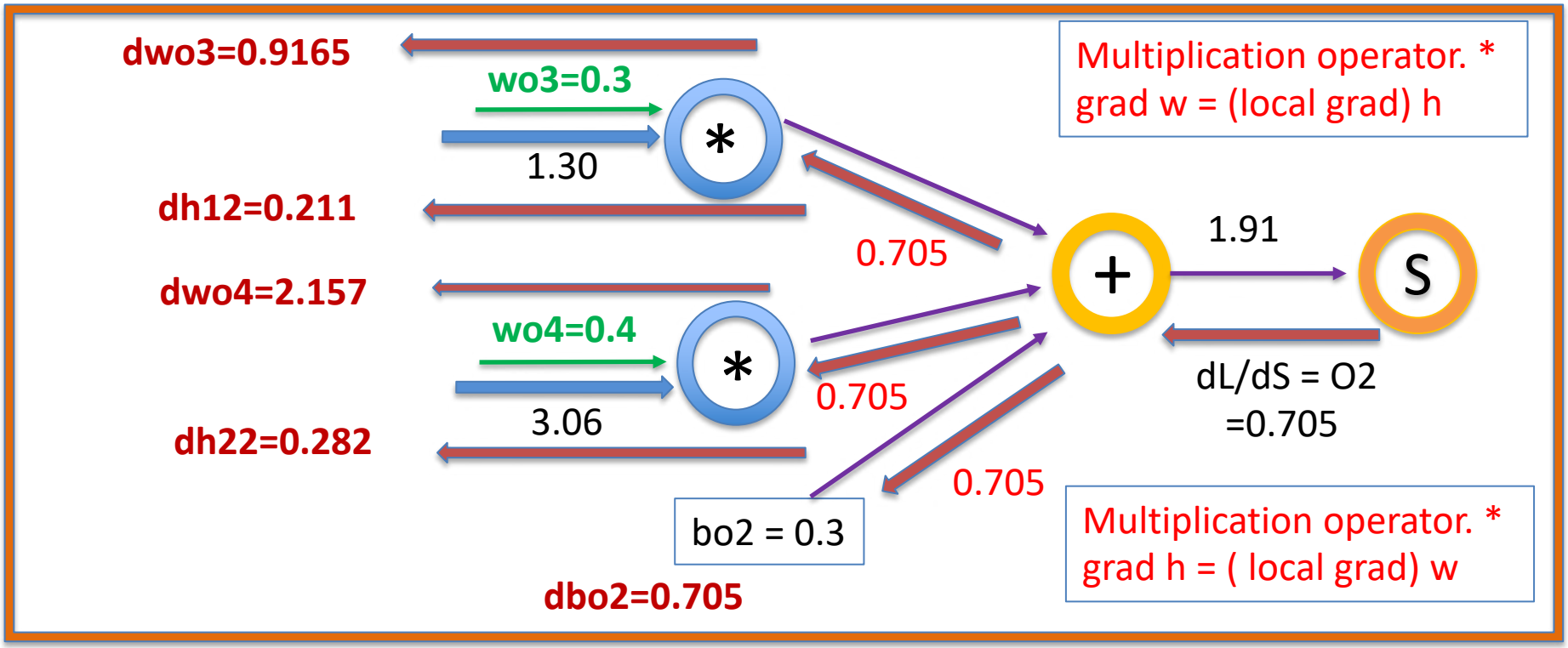
Multiplication operator. (*)

grad w = (local grad) h

Multiplication operator. (*)

grad h = (local grad) w

Update values of wo3, wo4, bo2



In here , i =2 (o2)
j = 1,2: h1, h2

d (L/dwo3) = dL/d(w21)
= 0.705*1.30 = 0.9165

d (L/dwo4) = dL/d(w22)
= 0.705*3.06 = 2.157

d (L/dbo1) = dL/d(b1)
= dL/dS = Oi-Yi=0.705

wo3+= 0.3– (0.5)*(0.9165) =-0.158

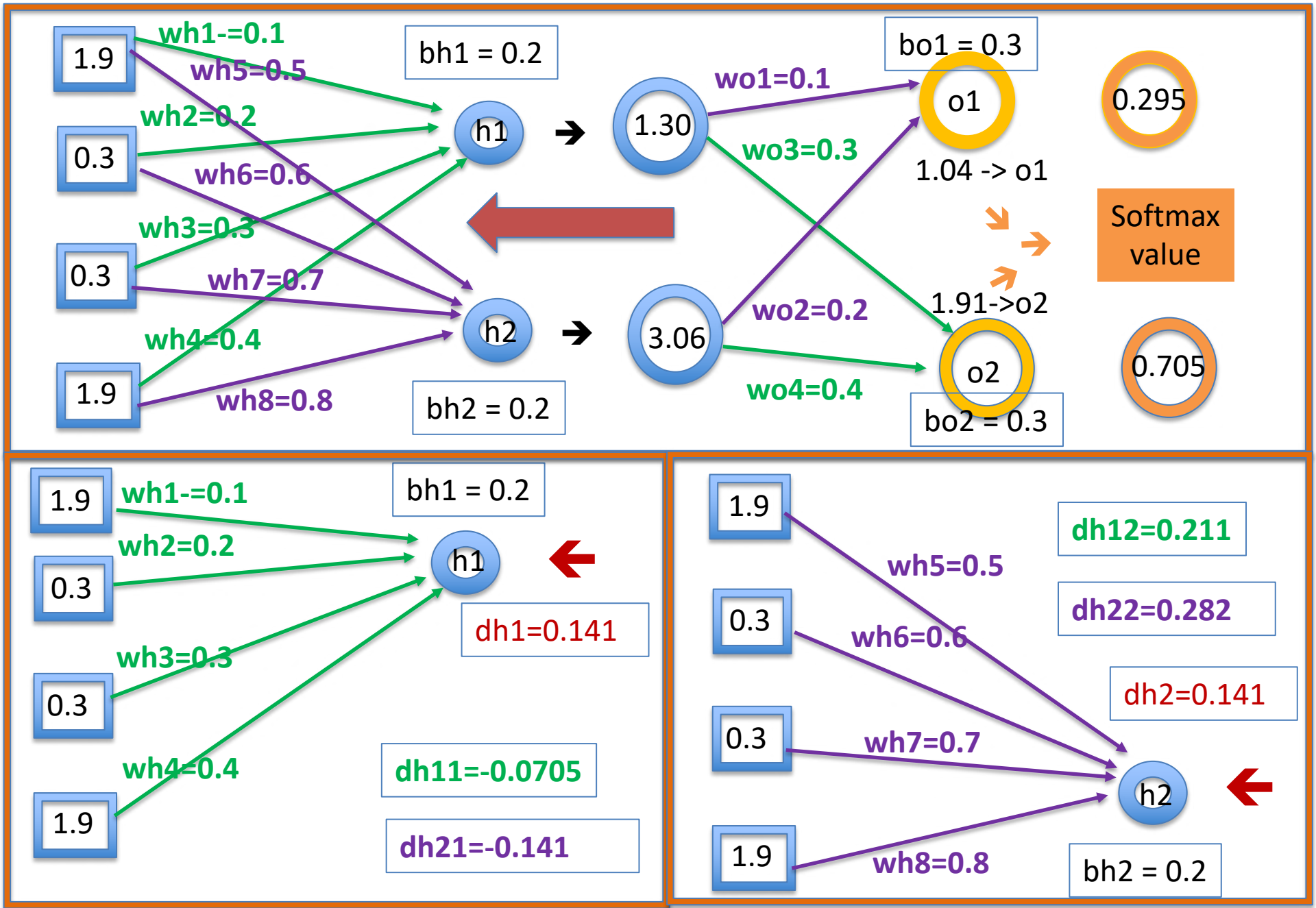
wo4+= 0.4– (0.5)*(2.157) =-0.678

bo2+= 0.3– (0.5)*(0.705) =-0.352

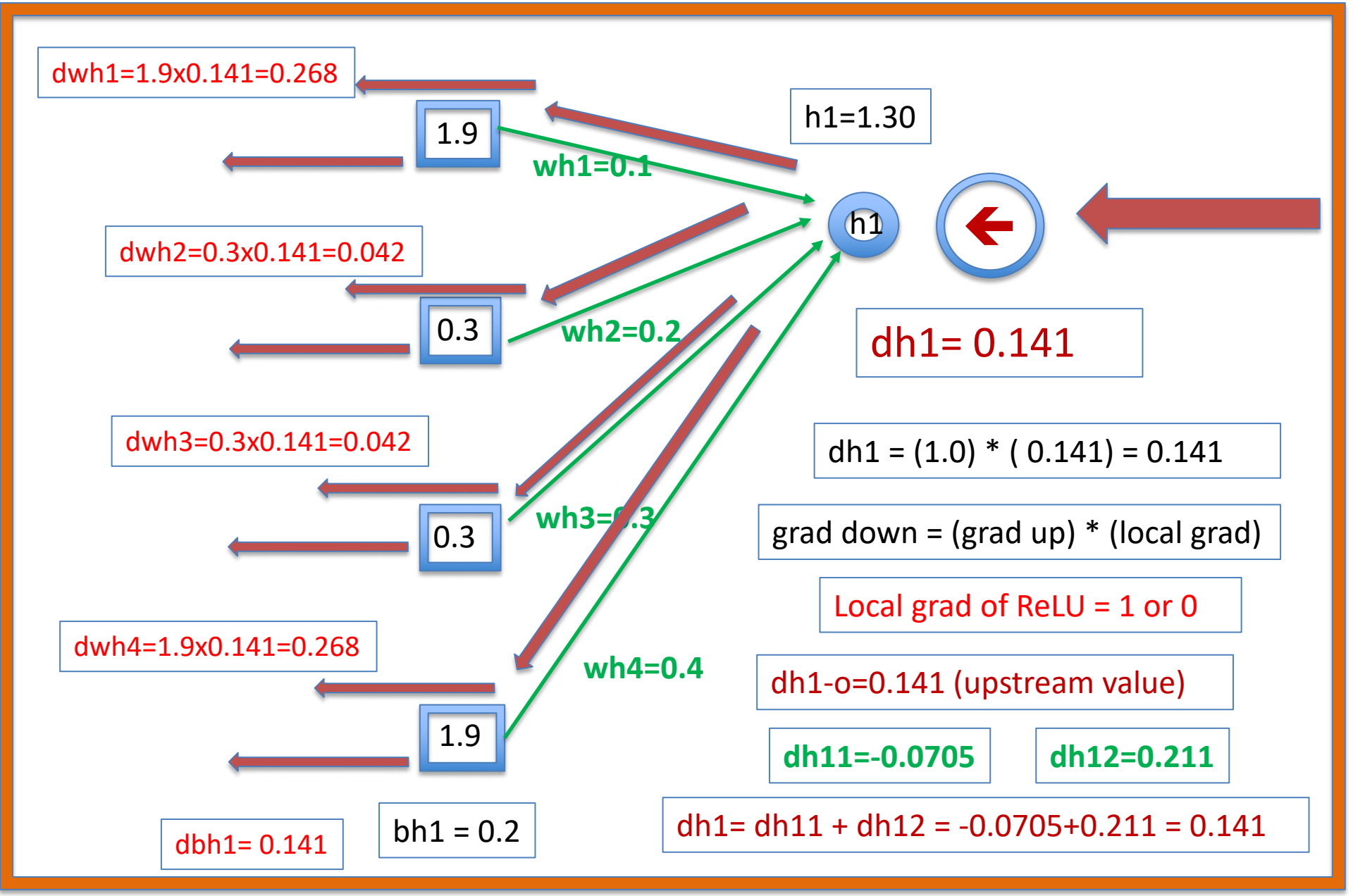
dh12= (0.211)

dh22= (0.282)

Flow back from o1 to the hidden NN layer (h) , MLP calculation



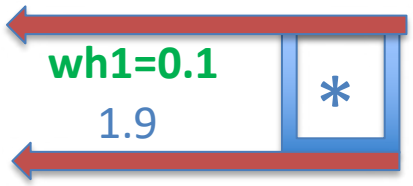
Gradients after h1,ReLU function (h1 neuron)



Update values of wh1, wh2, wh3, wh4, bh1

-0.034 = wh1+=0.1-0.5x0.268

dwh1=1.9x0.141=0.268

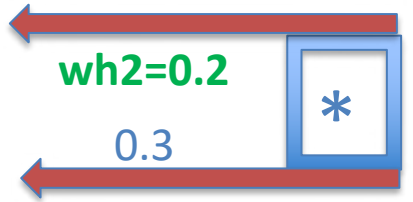


dC1h1=0.1x0.141=0.0141

dh1= 0.141

0.179 = wh2+=0.2-0.5x0.042

dwh2=0.3x0.141=0.042

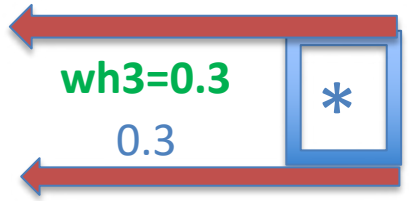


dC2h1=0.2x0.141=0.0282

dh1= 0.141

0.279 = wh3+=0.3-0.5x0.042

dwh3=0.3x0.141=0.042

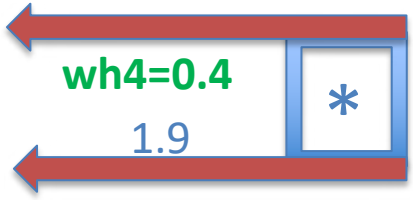


dC3h1=0.3x0.141=0.0423

dh1= 0.141

0.266 = wh4+=0.4-0.5x0.268

dwh4=1.9x0.141=0.268



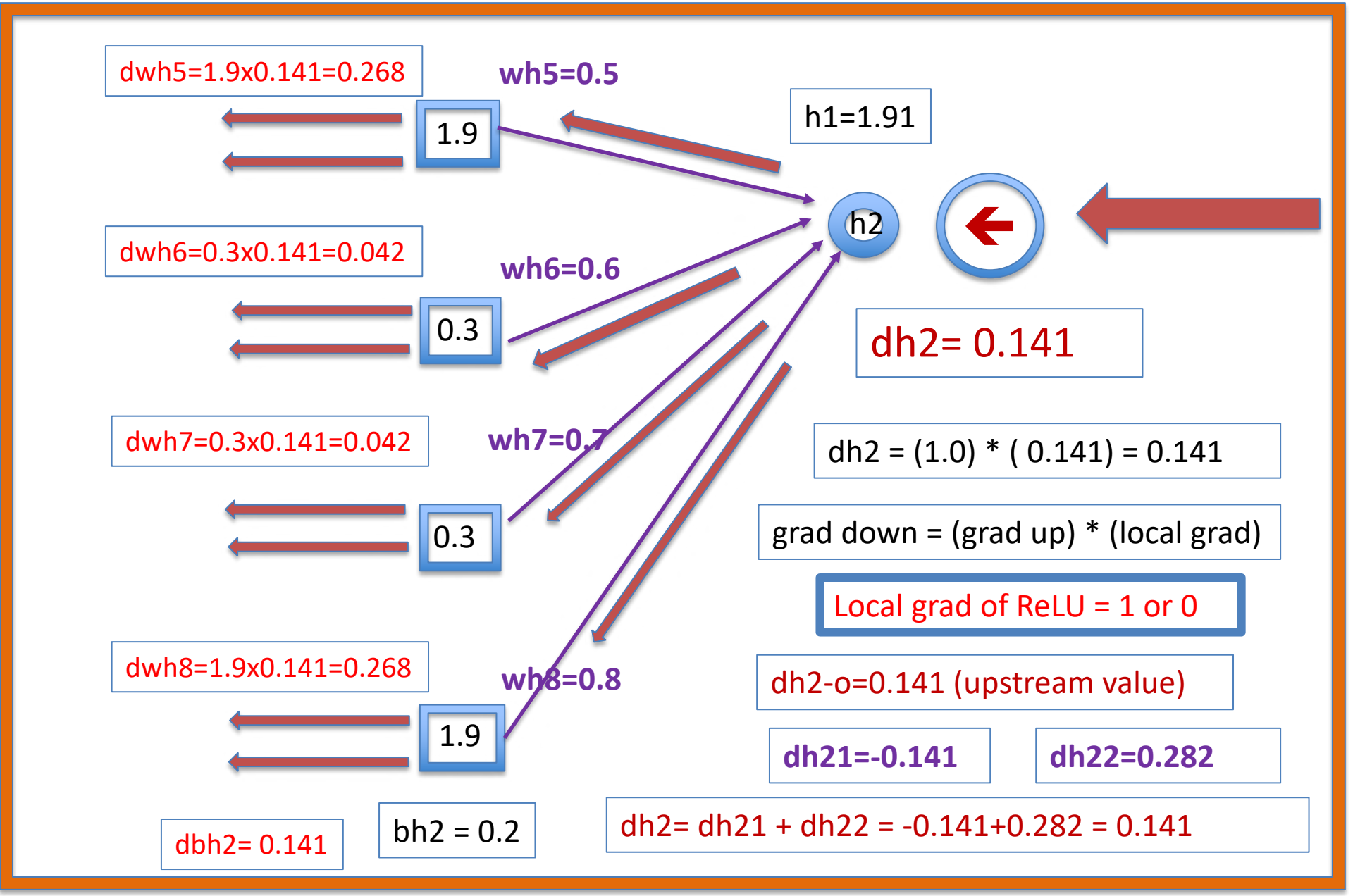
dC4h1=0.4x0.141=0.0564

dh1= 0.141

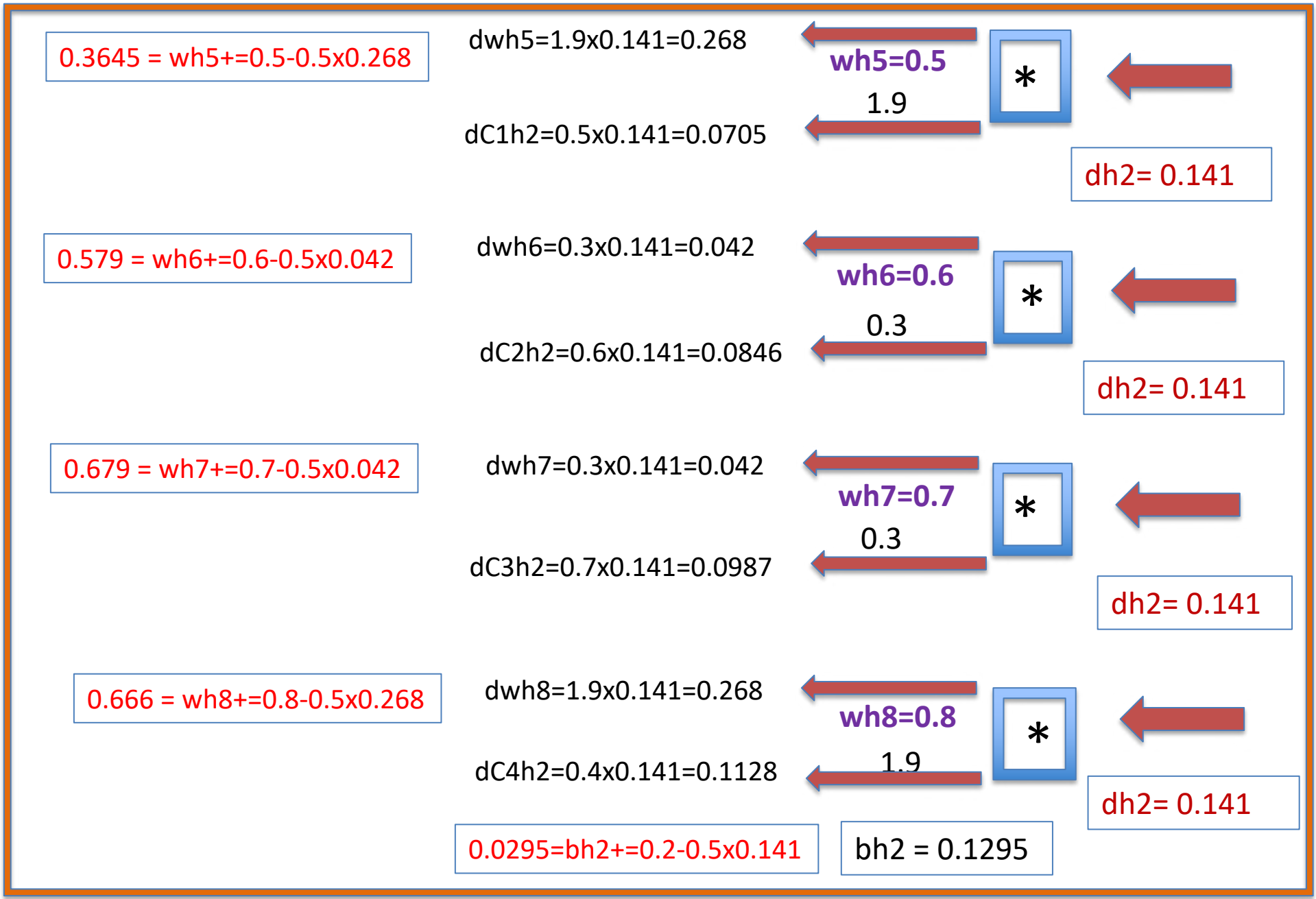
0.0295=bh1+=0.2-0.5x0.141

bh1 = 0.1295

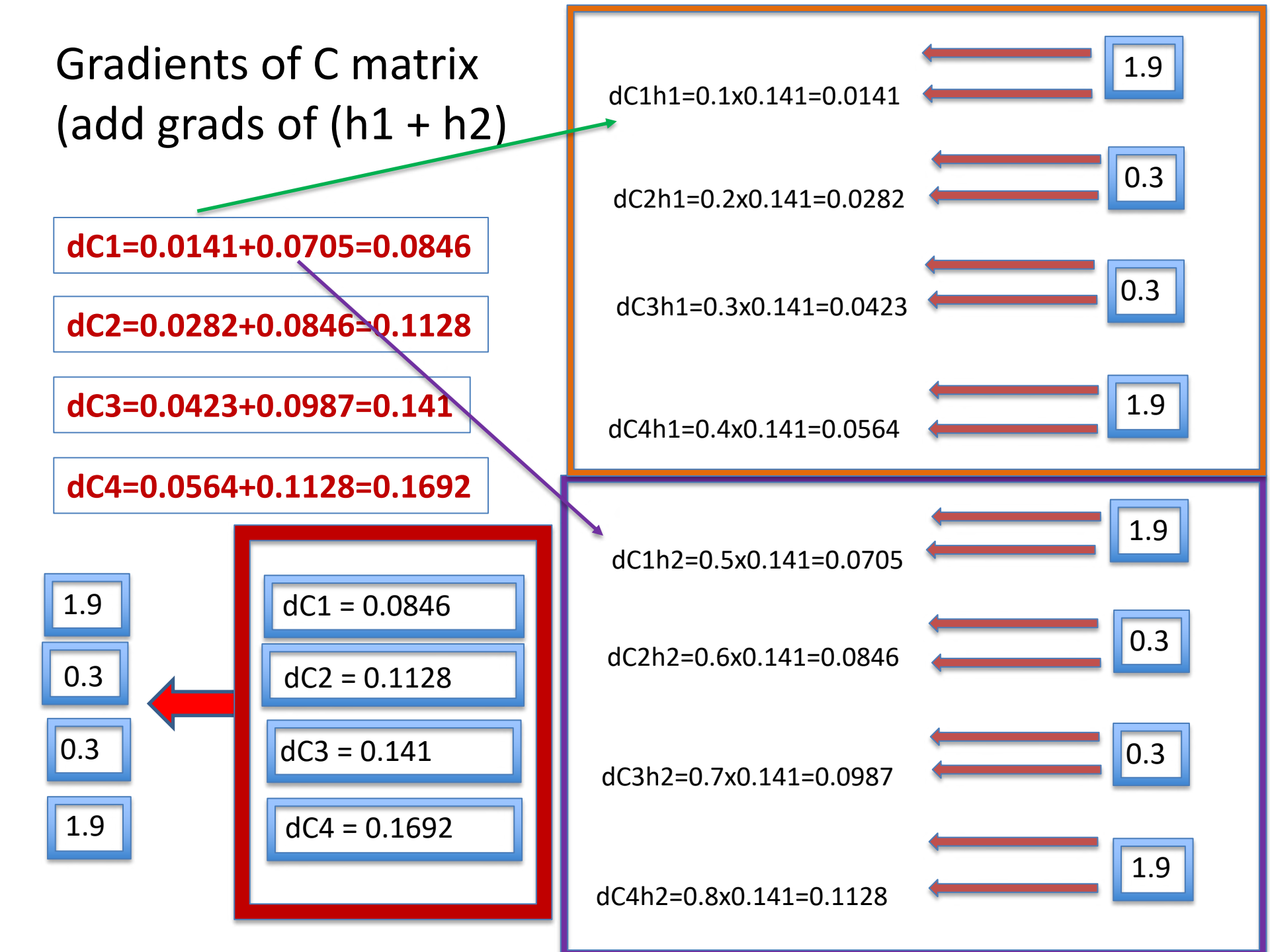
Gradients after h2, ReLU function (h2 neuron)

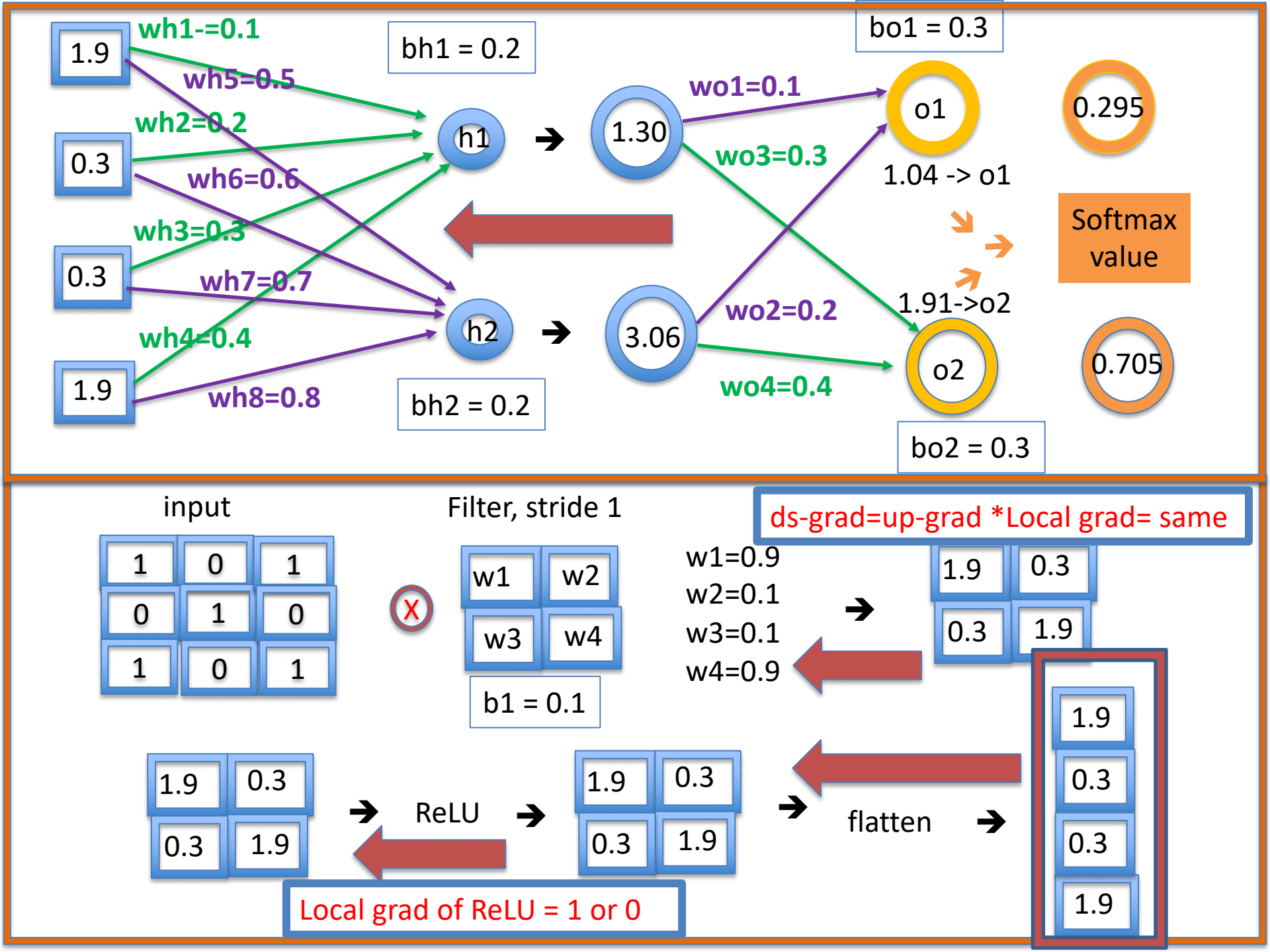


Update values of wh5, wh6, wh7, wh8, bh2

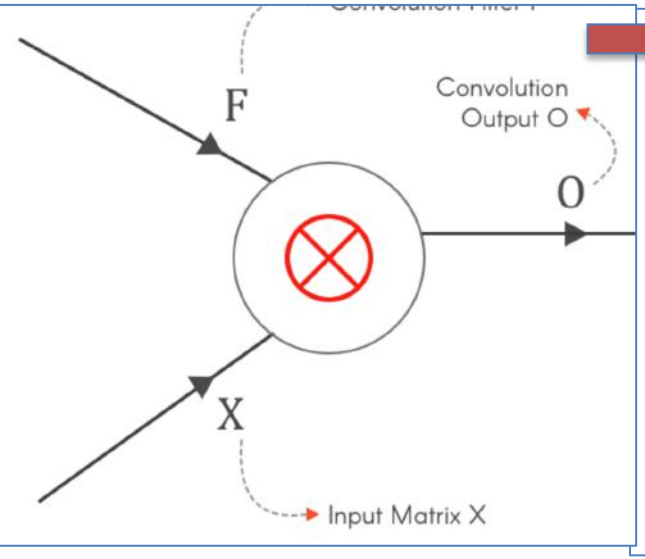


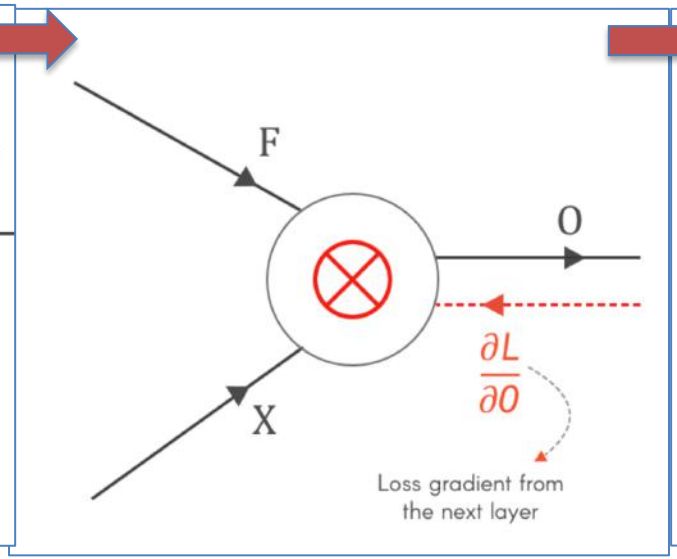
Gradients of C matrix (add grads of (h1 + h2)

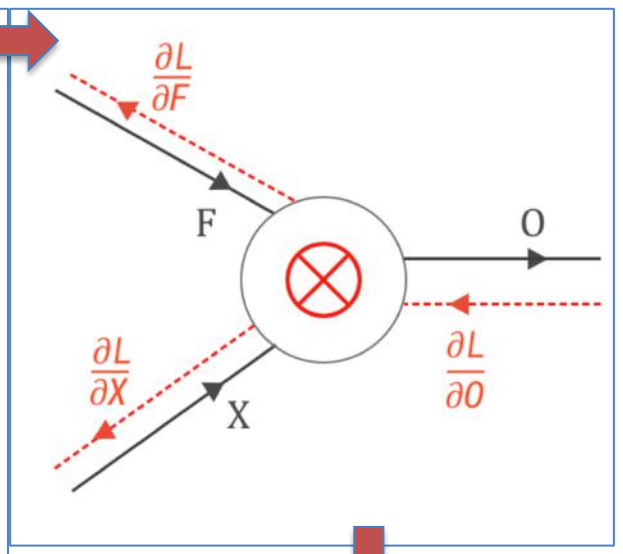


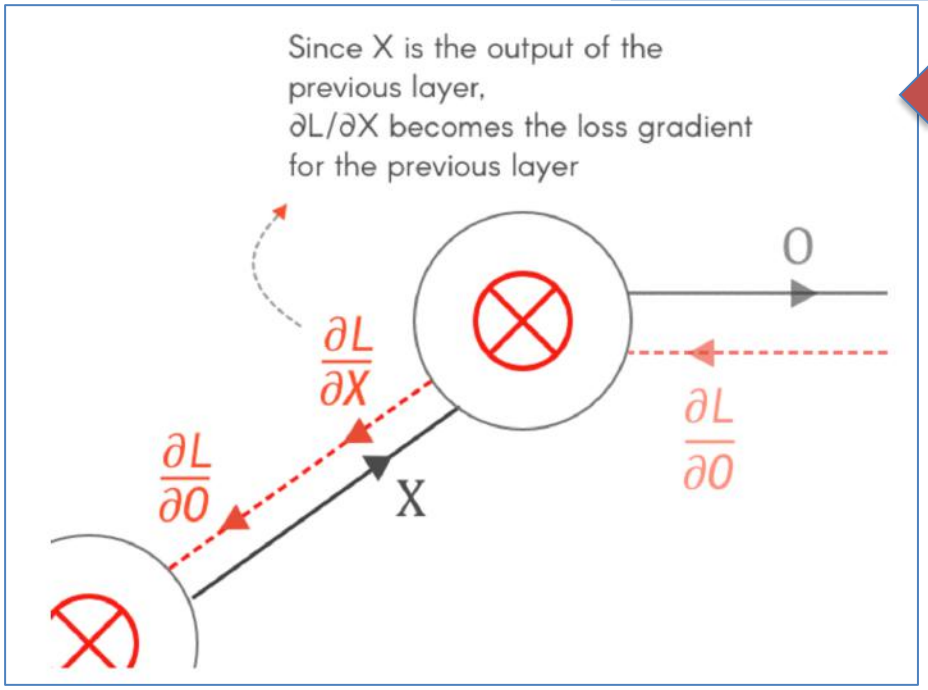

1.9
0.3
0.3
1.9
wh1-=0.1
wh5=0.5
wh2=0.2
wh6=0.6
wh3=0.3
wh7=0.7
wh4=0.4
wh8=0.8
bh1 = 0.2
h1
h2
bh2 = 0.2
1.30
3.06
wo1=0.1
wo3=0.3
wo2=0.2
wo4=0.4
bo1 = 0.3
o1
o2
bo2 = 0.3
0.295
1.04 -> o1
1.91->o2
0.705
Softmax value
input
1 0 1
0 1 0
1 0 1
X
Filter, stride 1
w1 w2
w3 w4
b1 = 0.1
ds-grad=up-grad *Local grad= same
w1=0.9
w2=0.1
w3=0.1
w4=0.9
1.9 0.3
0.3 1.9
1.9
0.3
0.3
1.9
flatten
1.9 0.3
0.3 1.9
ReLU
1.9 0.3
0.3 1.9
Local grad of ReLU = 1 or 0

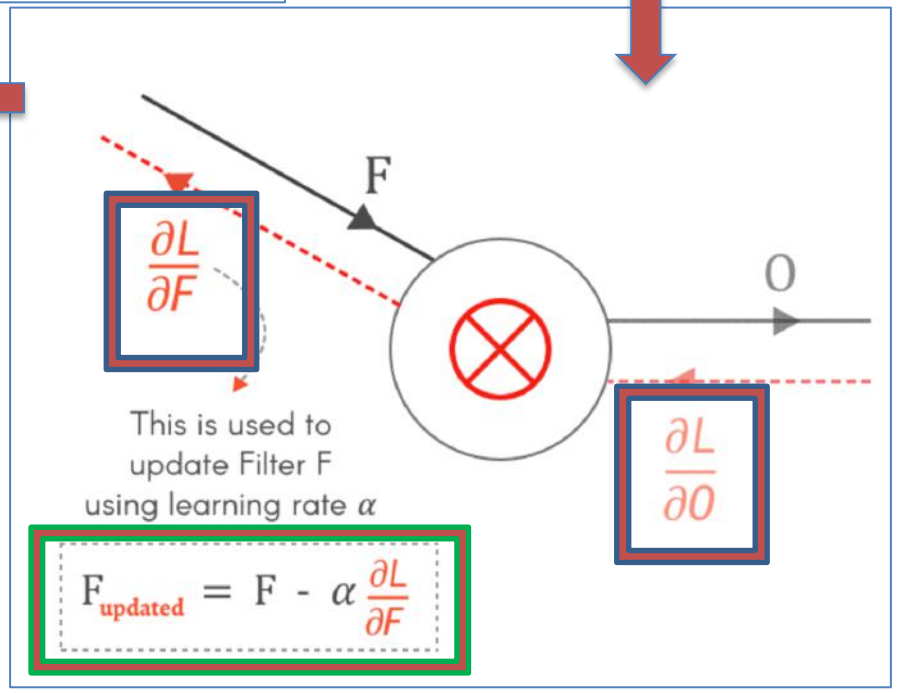
Convolution : Forward path : backward path











https://medium.com/@pavisj/convolutions-and-backpropagations-46026a8f5d2c

Backpropagation in a Convolutional Layer of a CNN

Finding the gradients:

$$\frac{\partial L}{\partial F} = \text{Convolution}\left(\text{Input } X, \text{ Loss gradient } \frac{\partial L}{\partial O}\right)$$

$$\frac{\partial L}{\partial X} = \text{Full Convolution}\left(180^\circ \text{rotated Filter } F, \text{ Loss Gradient } \frac{\partial L}{\partial O}\right)$$

Loss gradient = dL/dC

dC1=0.0846 | dC2=0.1128

dC3=0.141 | dC4=0.1692

1	0	1
0	1	0
1	0	1

X

dC1=0.0846 | dC2=0.1128

dC3=0.141 | dC4=0.1692

=

dL/dF

dF1=0.2538 | dF2=0.2538

dF3=0.2538 | dF4=0.238

dF1 = 1 x dC1 + 0 x dC2 + 0 x dC3 + 1 x dC4 = 0.5238

dF2 = 0 x dC1 + 1 x dC2 + 1 x dC3 + 0 x dC4 = 0.2538

dF3 = 0 x dC1 + 1 x dC2 + 1 x dC3 + 0 x dC4 = 0.2538

dF4 = 1 x dC1 + 0 x dC2 + 0 x dC3 + 1 x dC4 = 0.2538

F1+=0.7731 | F2+=-0.027

F3+=-0.027 | F4+=0.7731

=

0.9	0.1
0.1	0.9

- (0.5) *

dF1=0.2538 | dF2=0.2538

dF3=0.2538 | dF4=0.2538

2x2 filter

https://medium.com/@pavisj/convolutions-and-backpropagations-46026a8f5d2c

Local Gradients → (A)

$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$

Finding derivatives with respect to F_{11}, F_{12}, F_{21} and F_{22}

$$\frac{\partial O_{11}}{\partial F_{11}} = X_{11} \quad \frac{\partial O_{11}}{\partial F_{12}} = X_{12} \quad \frac{\partial O_{11}}{\partial F_{21}} = X_{21} \quad \frac{\partial O_{11}}{\partial F_{22}} = X_{22}$$

Similarly, we can find the local gradients for O_{12}, O_{21} and O_{22}

$$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{11}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{11}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{11}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{11}}$$

$$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{12}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{12}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{12}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{12}}$$

$$\frac{\partial L}{\partial F_{21}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{21}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{21}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{21}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{21}}$$

$$\frac{\partial L}{\partial F_{22}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{22}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{22}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{22}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{22}}$$

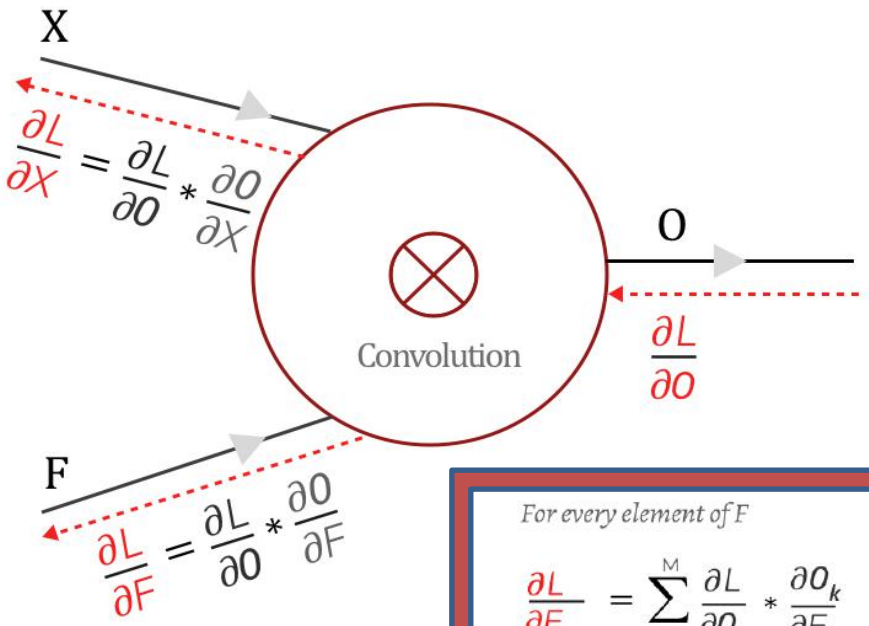
a convolution operation between input X and loss gradient ∂L/∂O

$$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O_{11}} * X_{11} + \frac{\partial L}{\partial O_{12}} * X_{12} + \frac{\partial L}{\partial O_{21}} * X_{21} + \frac{\partial L}{\partial O_{22}} * X_{22}$$

$$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}} * X_{12} + \frac{\partial L}{\partial O_{12}} * X_{13} + \frac{\partial L}{\partial O_{21}} * X_{22} + \frac{\partial L}{\partial O_{22}} * X_{23}$$

$$\frac{\partial L}{\partial F_{21}} = \frac{\partial L}{\partial O_{11}} * X_{21} + \frac{\partial L}{\partial O_{12}} * X_{22} + \frac{\partial L}{\partial O_{21}} * X_{31} + \frac{\partial L}{\partial O_{22}} * X_{32}$$

$$\frac{\partial L}{\partial F_{22}} = \frac{\partial L}{\partial O_{11}} * X_{22} + \frac{\partial L}{\partial O_{12}} * X_{23} + \frac{\partial L}{\partial O_{21}} * X_{32} + \frac{\partial L}{\partial O_{22}} * X_{33}$$



For every element of F

$$\frac{\partial L}{\partial F_i} = \sum_{k=1}^{M} \frac{\partial L}{\partial O_k} * \frac{\partial O_k}{\partial F_i}$$

$\frac{\partial O}{\partial X}$ & $\frac{\partial O}{\partial F}$ *are local gradients*

$\frac{\partial L}{\partial z}$ *is the loss from the previous layer which has to be backpropagated to other layers*

$$\frac{\partial L}{\partial F} = \frac{\partial L}{\partial O} * \frac{\partial O}{\partial F}$$

- Gradient to update Filter F
- Loss Gradient from previous layer
- Local Gradients

$$\begin{bmatrix} \frac{\partial L}{\partial F_{11}} & \frac{\partial L}{\partial F_{12}} \\ \frac{\partial L}{\partial F_{21}} & \frac{\partial L}{\partial F_{22}} \end{bmatrix} = \text{Convolution} \left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} \frac{\partial L}{\partial O_{11}} & \frac{\partial L}{\partial O_{12}} \\ \frac{\partial L}{\partial O_{21}} & \frac{\partial L}{\partial O_{22}} \end{bmatrix} \right)$$

∂L/∂F is nothing but the convolution between Input X and Loss Gradient from the next layer ∂L/∂O

where

$$\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix} = \text{Input X} \qquad \begin{bmatrix} \frac{\partial L}{\partial O_{11}} & \frac{\partial L}{\partial O_{12}} \\ \frac{\partial L}{\partial O_{21}} & \frac{\partial L}{\partial O_{22}} \end{bmatrix} = \frac{\partial L}{\partial O}$$ Loss gradient from previous layer

Local Gradients: B

2x2 filter

$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$

Differentiating with respect to X_{11}, X_{12}, X_{21} *and* X_{22}

$$\frac{\partial O_{11}}{\partial X_{11}} = F_{11} \quad \frac{\partial O_{11}}{\partial X_{12}} = F_{12} \quad \frac{\partial O_{11}}{\partial X_{21}} = F_{21} \quad \frac{\partial O_{11}}{\partial X_{22}} = F_{22}$$

Similarly, we can find local gradients for O_{12}, O_{21} *and* O_{22}

For every element of X_i

$$\frac{\partial L}{\partial X_i} = \sum_{k=1}^{M} \frac{\partial L}{\partial O_k} * \frac{\partial O_k}{\partial X_i}$$

$$\frac{\partial L}{\partial X_{11}} = \frac{\partial L}{\partial O_{11}} * F_{11}$$

$$\frac{\partial L}{\partial X_{12}} = \frac{\partial L}{\partial O_{11}} * F_{12} + \frac{\partial L}{\partial O_{12}} * F_{11}$$

$$\frac{\partial L}{\partial X_{13}} = \frac{\partial L}{\partial O_{12}} * F_{12}$$

$$\frac{\partial L}{\partial X_{21}} = \frac{\partial L}{\partial O_{11}} * F_{21} + \frac{\partial L}{\partial O_{21}} * F_{11}$$

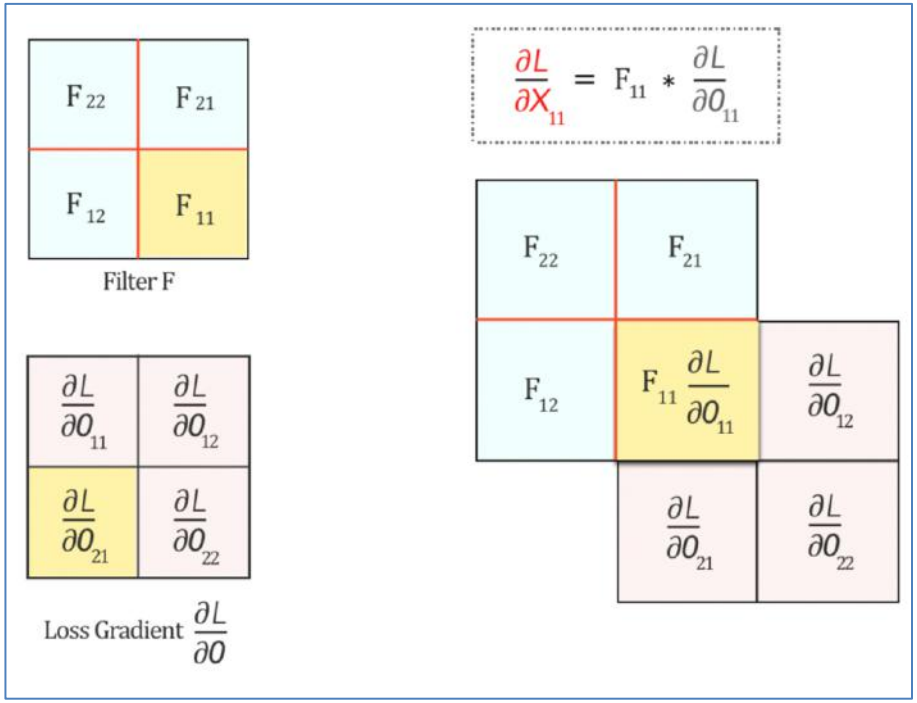
$$\frac{\partial L}{\partial X_{22}} = \frac{\partial L}{\partial O_{11}} * F_{22} + \frac{\partial L}{\partial O_{12}} * F_{21} + \frac{\partial L}{\partial O_{21}} * F_{12} + \frac{\partial L}{\partial O_{22}} * F_{11}$$

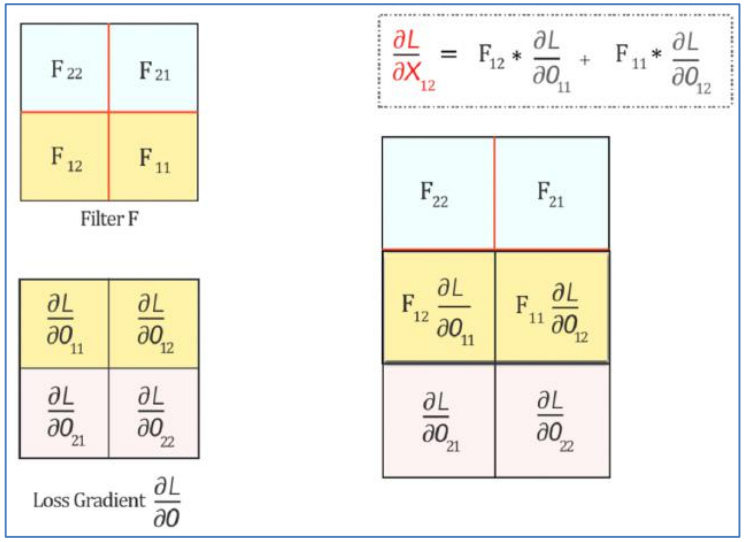
$$\frac{\partial L}{\partial X_{23}} = \frac{\partial L}{\partial O_{12}} * F_{22} + \frac{\partial L}{\partial O_{22}} * F_{12}$$

$$\frac{\partial L}{\partial X_{31}} = \frac{\partial L}{\partial O_{21}} * F_{21}$$

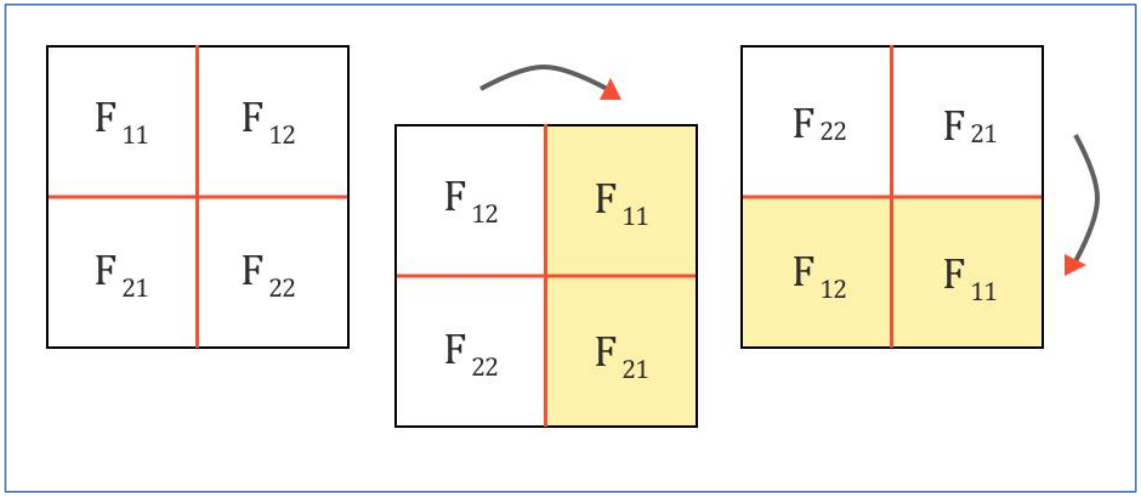
$$\frac{\partial L}{\partial X_{32}} = \frac{\partial L}{\partial O_{21}} * F_{22} + \frac{\partial L}{\partial O_{22}} * F_{21}$$

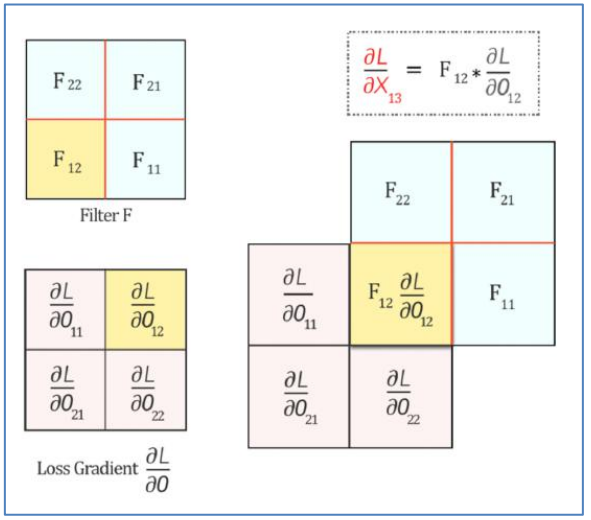
$$\frac{\partial L}{\partial X_{33}} = \frac{\partial L}{\partial O_{22}} * F_{22}$$





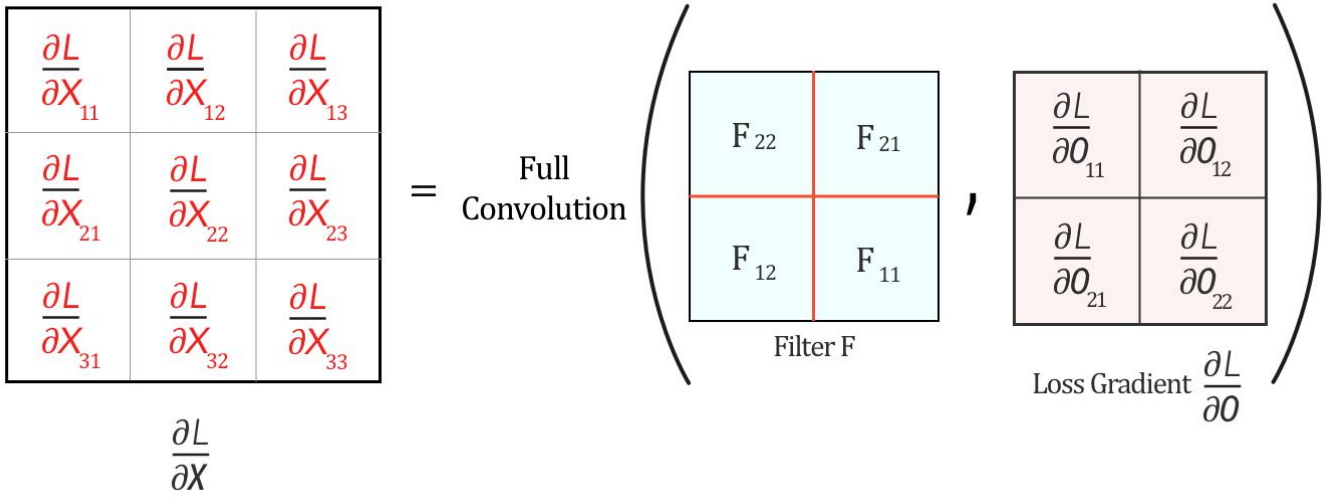
∂L/∂X can be represented as 'full' convolution between a 180-degree rotated Filter F and loss gradient ∂L/∂O





Forward path : backward path

Forward path : backward path



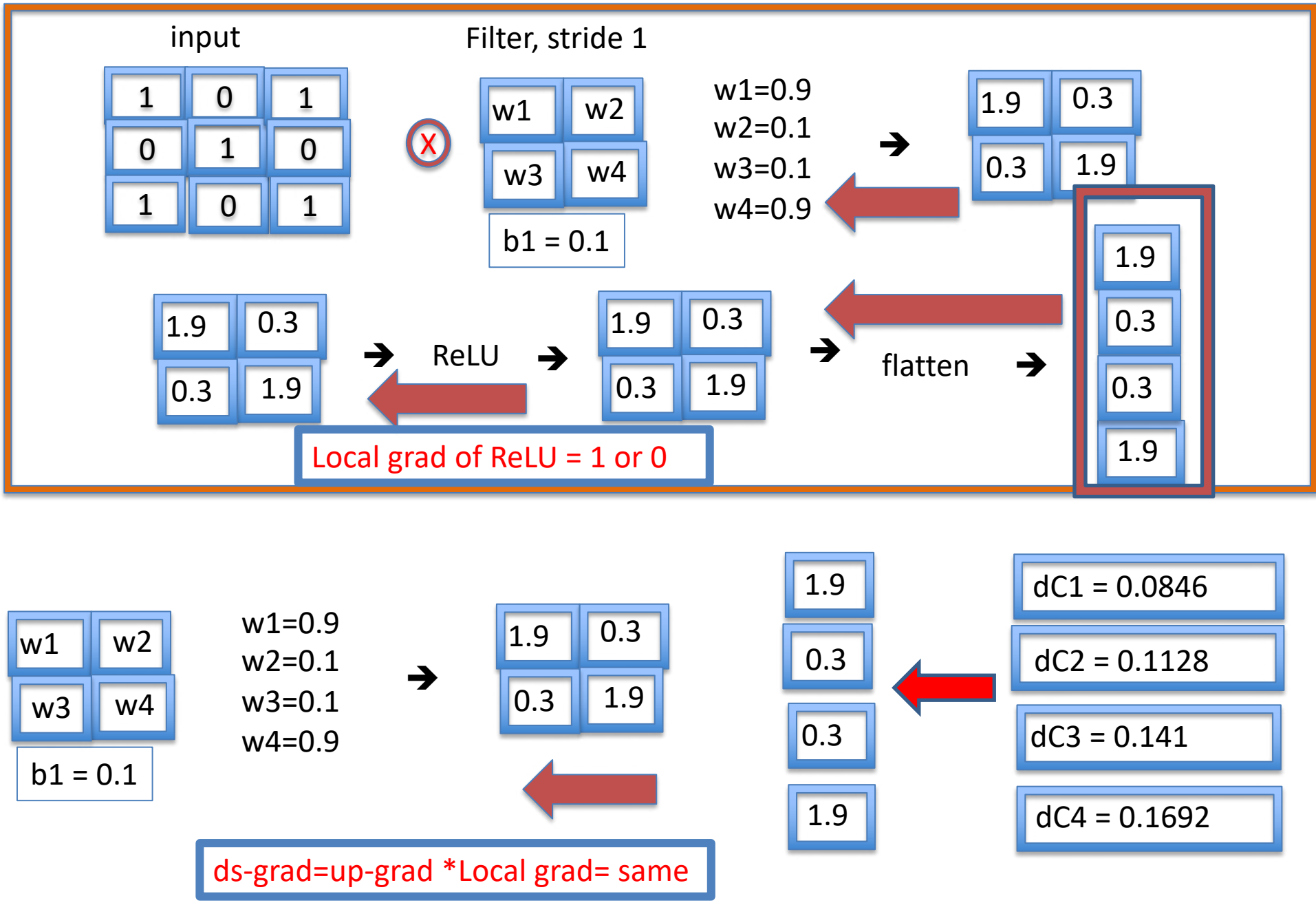
Backpropagation in a Convolutional Layer of a CNN

Finding the gradients:

$$\frac{\partial L}{\partial F} = \text{Convolution}\left(\text{Input } X, \text{ Loss gradient } \frac{\partial L}{\partial O}\right)$$

$$\frac{\partial L}{\partial X} = \text{Full Convolution}\left(180^\circ \text{rotated Filter } F, \text{ Loss Gradient } \frac{\partial L}{\partial O}\right)$$

∂L/∂X can be represented as 'full' convolution between a 180-degree rotated Filter F and loss gradient ∂L/∂O


input
Filter, stride 1
1 0 1
0 1 0
1 0 1
X
w1 w2
w3 w4
b1 = 0.1
w1=0.9
w2=0.1
w3=0.1
w4=0.9
1.9 0.3
0.3 1.9
1.9
0.3
0.3
1.9
flatten
1.9 0.3
0.3 1.9
ReLU
1.9 0.3
0.3 1.9
Local grad of ReLU = 1 or 0
w1 w2
w3 w4
b1 = 0.1
w1=0.9
w2=0.1
w3=0.1
w4=0.9
1.9 0.3
0.3 1.9
1.9
0.3
0.3
1.9
dC1 = 0.0846
dC2 = 0.1128
dC3 = 0.141
dC4 = 0.1692
ds-grad=up-grad *Local grad= same

Backpropagation in a Convolutional Layer of a CNN

Finding the gradients:

$$\frac{\partial L}{\partial F} = \text{Convolution}\left(\text{Input } X, \text{ Loss gradient } \frac{\partial L}{\partial O}\right)$$

$$\frac{\partial L}{\partial X} = \text{Full Convolution}\left(180^{\circ}\text{rotated Filter } F, \text{ Loss Gradient } \frac{\partial L}{\partial O}\right)$$

Loss gradient = dL/dC

dC1=0.0846 | dC2=0.1128

dC3=0.141 | dC4=0.1692

1	0	1
0	1	0
1	0	1

X

dC1=0.0846 | dC2=0.1128

dC3=0.141 | dC4=0.1692

=

dL/dF

dF1=0.2538 | dF2=0.2538

dF3=0.2538 | dF4=0.238

dF1 = 1 x dC1 + 0 x dC2 + 0 x dC3 + 1 x dC4 = 0.5238

dF2 = 0 x dC1 + 1 x dC2 + 1 x dC3 + 0 x dC4 = 0.2538

dF3 = 0 x dC1 + 1 x dC2 + 1 x dC3 + 0 x dC4 = 0.2538

dF4 = 1 x dC1 + 0 x dC2 + 0 x dC3 + 1 x dC4 = 0.2538

F1+=0.7731 | F2+=-0.027

F3+=-0.027 | F4+=0.7731

=

0.9	0.1
0.1	0.9

- (0.5) *

dF1=0.2538 | dF2=0.2538

dF3=0.2538 | dF4=0.2538

Update values of b1 – bias of convolution

Local Gradients → (A)

$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22} + b$$

Finding derivatives with respect to F_{11}, F_{12}, F_{21} and F_{22}

$$\frac{\partial O_{11}}{\partial F_{11}} = X_{11} \quad \frac{\partial O_{11}}{\partial F_{12}} = X_{12} \quad \frac{\partial O_{11}}{\partial F_{21}} = X_{21} \quad \frac{\partial O_{11}}{\partial F_{22}} = X_{22}$$

Similarly, we can find the local gradients for O_{12}, O_{21} and O_{22}

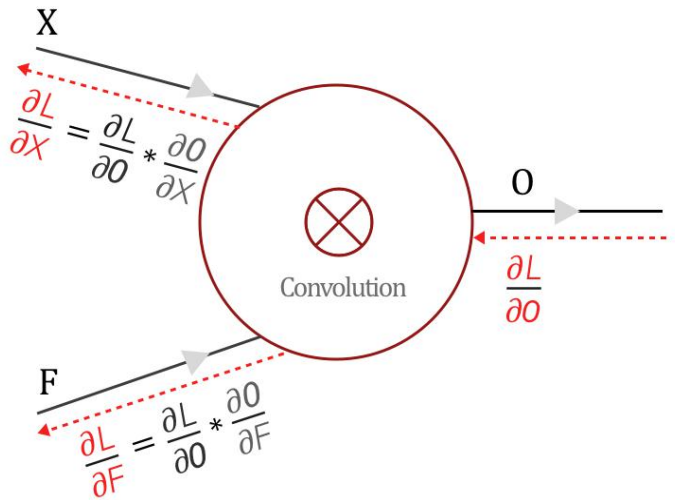
Loss gradient = dL/dC

dC1=0.0846

dC2=0.1128

dC3=0.141

dC4=0.1692



$\frac{\partial O}{\partial X}$ & $\frac{\partial O}{\partial F}$ are local gradients

$\frac{\partial L}{\partial z}$ is the loss from the previous layer which has to be backpropagated to other layers

dL/db = (dL/dO) * (dO/db)

O11 = X11F11 + X12*F12 + X21*F21 + X22*F22 +b1

So dO/db = 1 ; O = C in this case

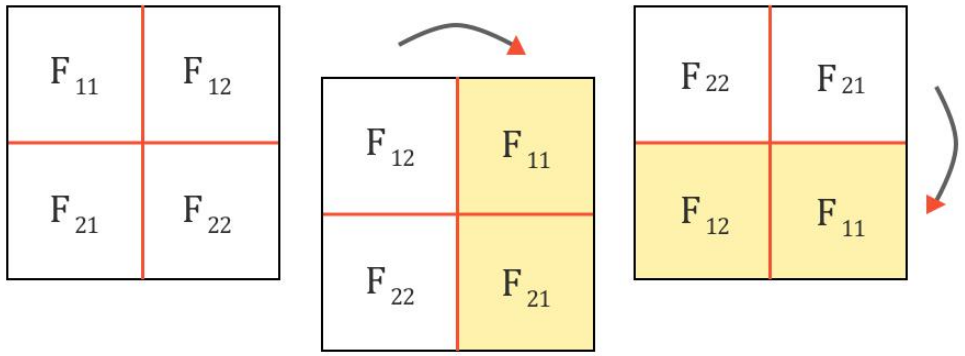
$dL/db_i = (dL/dC) * (dO/db) = (dL/dC)_i$

Add each contribution together

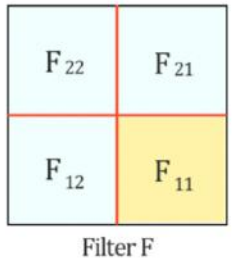
grad b = dC1 + dC2 + dC3 + dC4

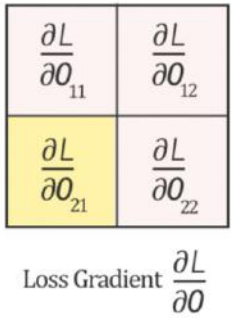
grad b = 0.0846+0.1128+0.141+0.1692 = 0.5076

b1+ = b – 0.5 * 0.5076 = 0.1 – 0.2538 = - 0.1538

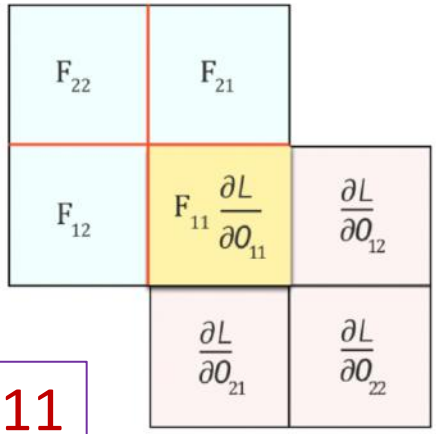


∂L/∂X can be represented as 'full' convolution between a 180-degree rotated Filter F and loss gradient ∂L/∂O

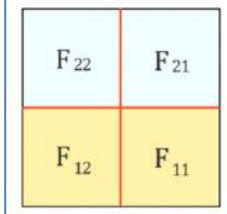


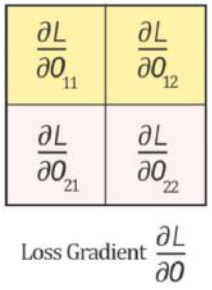


$$\frac{\partial L}{\partial X_{11}} = F_{11} * \frac{\partial L}{\partial O_{11}}$$

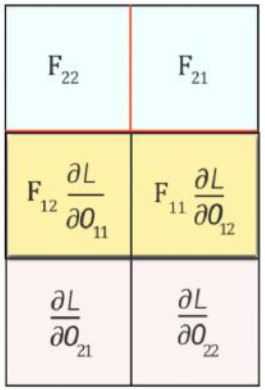


dL/dX11

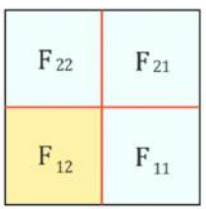


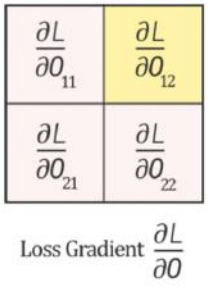


$$\frac{\partial L}{\partial X_{12}} = F_{12} * \frac{\partial L}{\partial O_{11}} + F_{11} * \frac{\partial L}{\partial O_{12}}$$

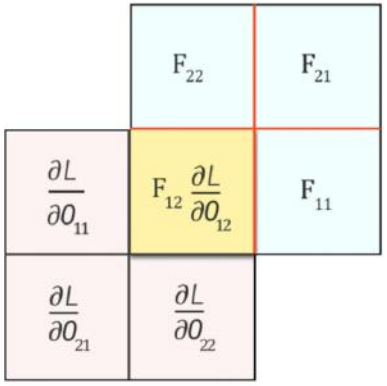


dL/dX12



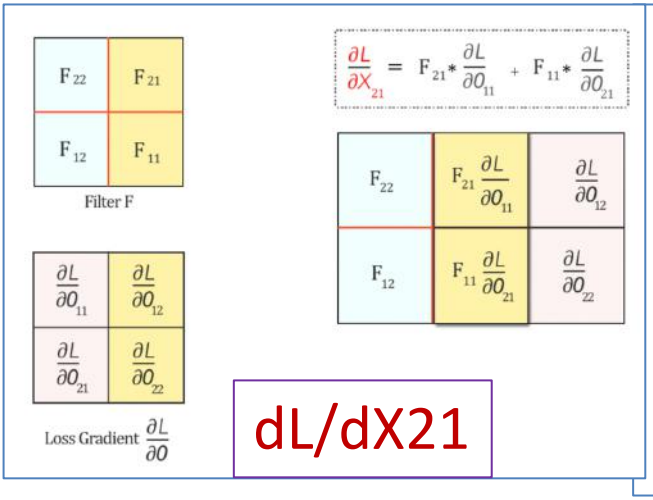


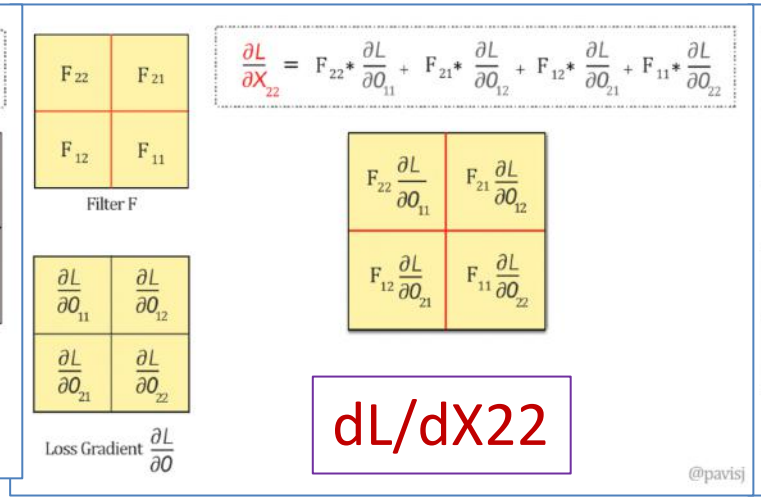
$$\frac{\partial L}{\partial X_{13}} = F_{12} * \frac{\partial L}{\partial O_{12}}$$

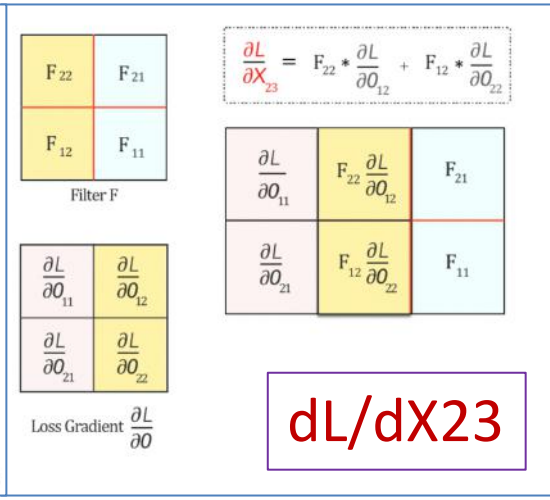


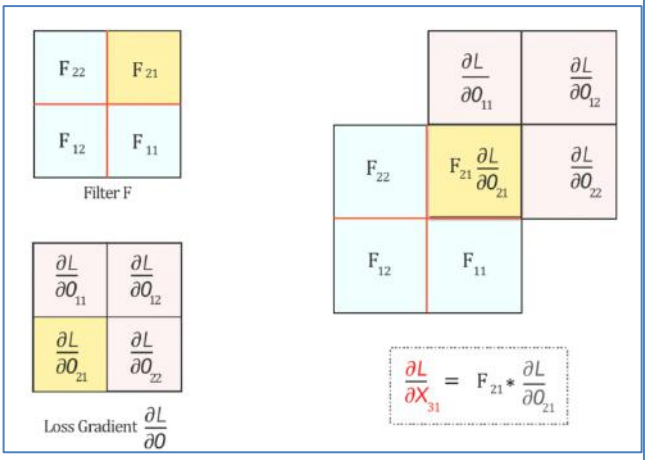
dL/dX13

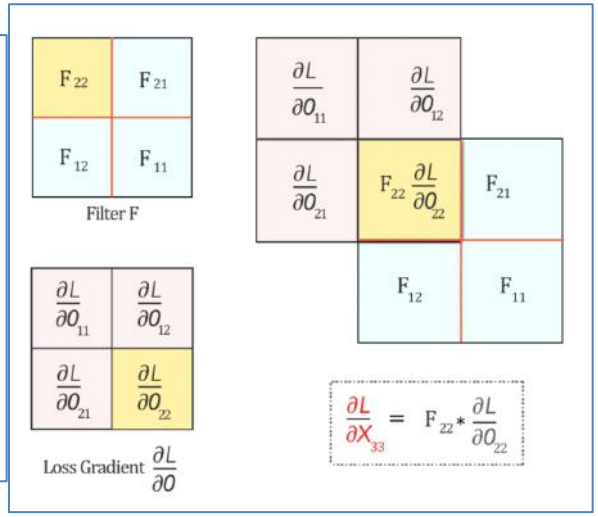
Forward path : backward path

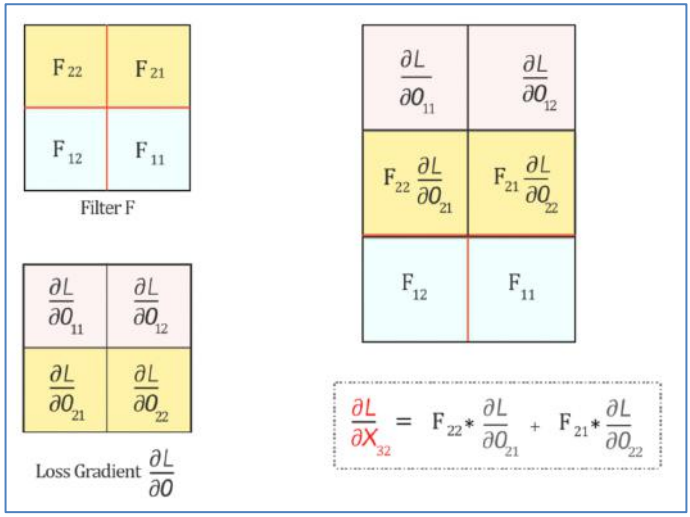












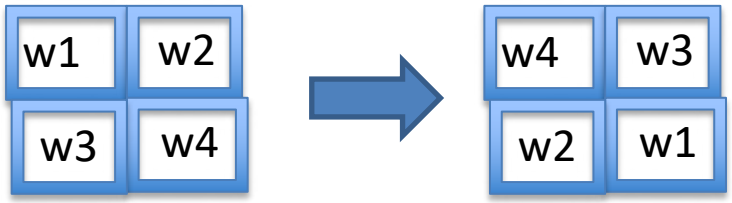
Backward path dL/dxij

Backpropagation in a Convolutional Layer of a CNN

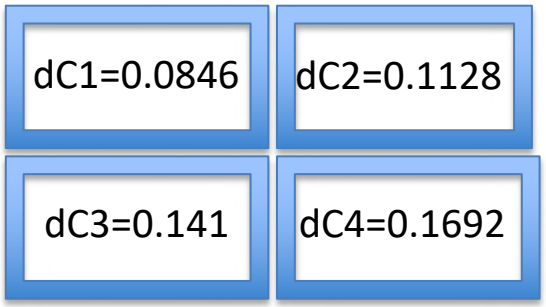
Finding the gradients:

$$\frac{\partial L}{\partial F} = \text{Convolution}\left(\text{Input } X, \text{ Loss gradient } \frac{\partial L}{\partial O}\right)$$

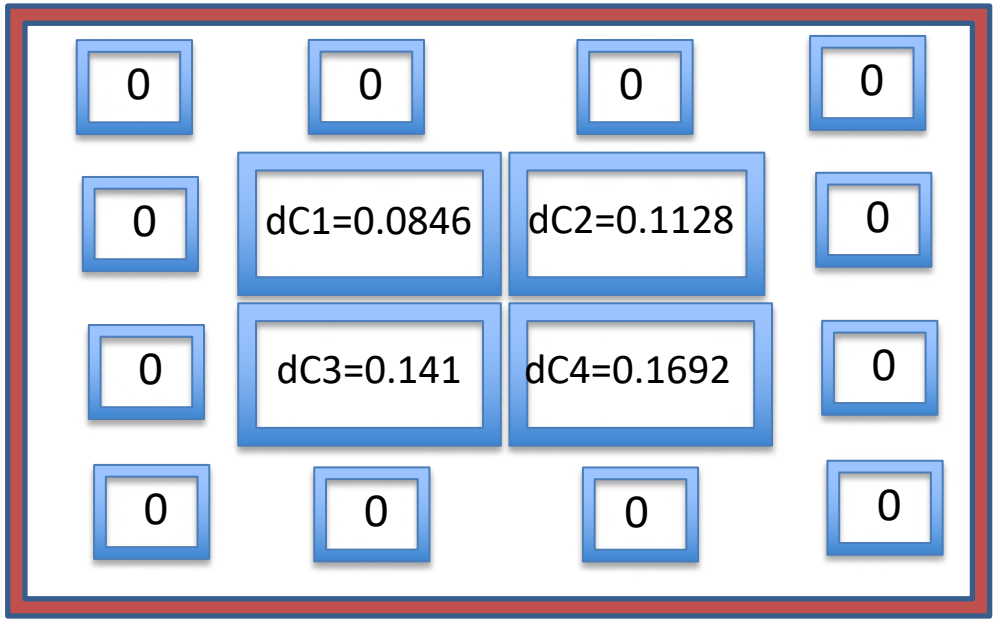
$$\frac{\partial L}{\partial X} = \text{Full Convolution}\left(180^{\circ}\text{rotated Filter } F, \text{ Loss Gradient } \frac{\partial L}{\partial O}\right)$$



Rotation 180° clockwise

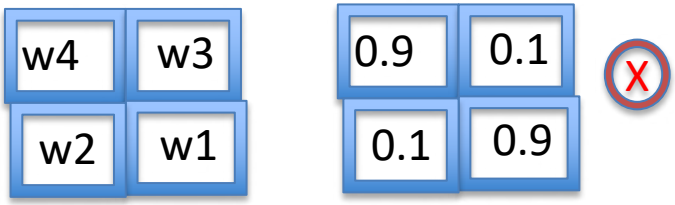


Add padding to return back the same shape of input

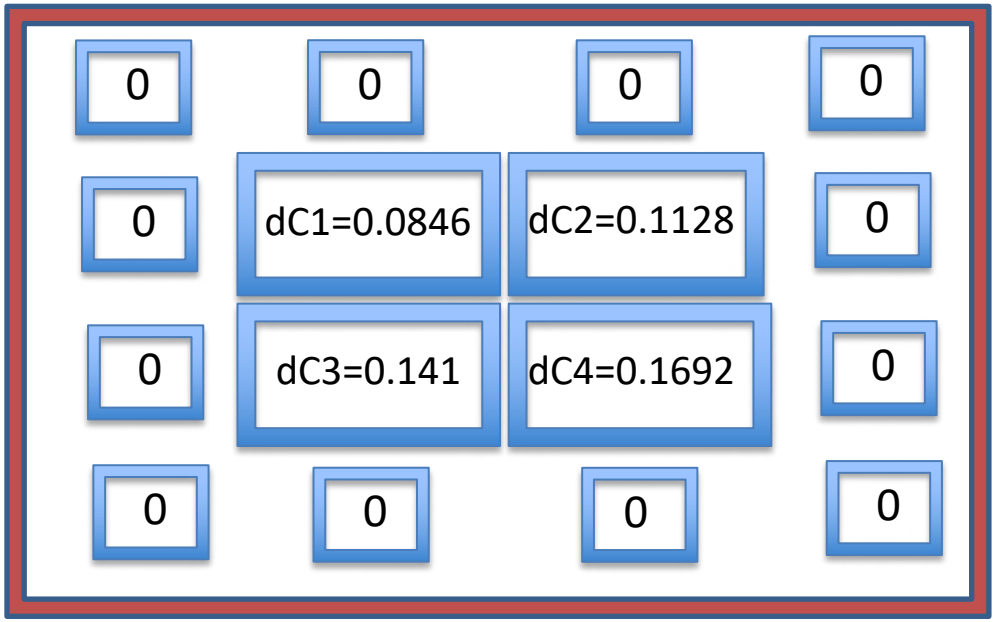


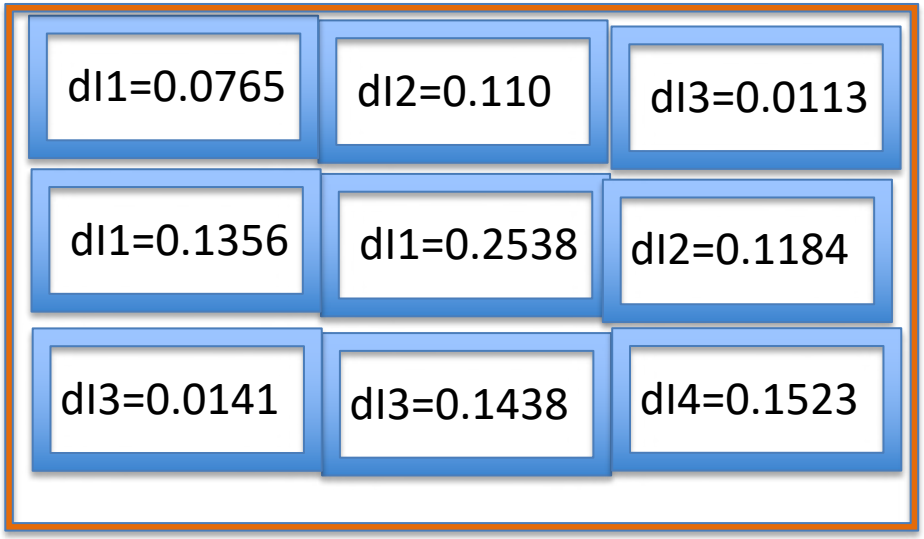
Values of dL/dI

w1=w4=0.9, w2=w3=0.1



Full convolution





ONLY Needed for intermediate CONV layers, not for input layer

i+ = i- (0.5) * grad i

I+	I+	i+
I+	I+	I+
I+	I+	I+

=

1	0	1
0	1	0
1	0	1

- (0.5) *

dI1=0.0765	dI2=0.110	dI3=0.0113
dI1=0.1356	dI1=0.2538	dI2=0.1184
dI3=0.0141	dI3=0.1438	dI4=0.1523

I1+ = xxxx I2+ = xxxx I3+ = xxxx

I4+ = xxxx I5+ = xxxxx I6+ = xxxxx

I7+ = xxxxx I8+ = xxxx I9+ = xxxxx

not needed in here for input layer

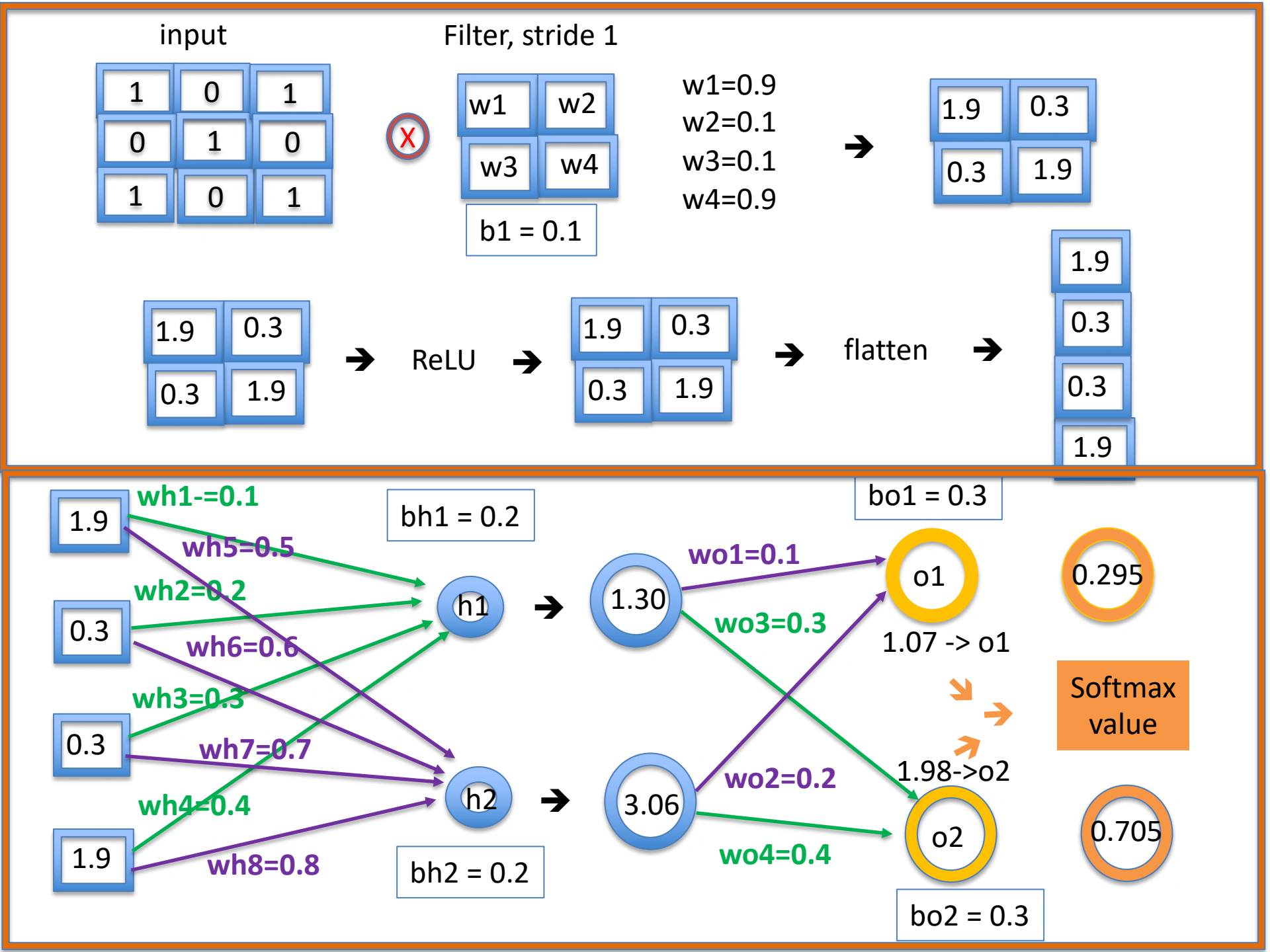
F1+=0.7731	F2+=-0.027
F3+=-0.027	F4+=0.7731

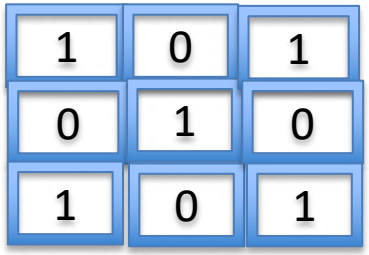
=

0.9	0.1
0.1	0.9

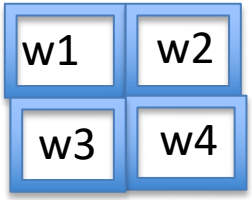
- (0.5) *

dF1=0.2538	dF2=0.2538
dF3=0.2538	dF4=0.2538


input
Filter, stride 1
1 0 1
0 1 0
1 0 1
X
w1 w2
w3 w4
b1 = 0.1
w1=0.9
w2=0.1
w3=0.1
w4=0.9
1.9 0.3
0.3 1.9
1.9 0.3
0.3 1.9
ReLU
1.9 0.3
0.3 1.9
flatten
1.9
0.3
0.3
1.9
wh1-=0.1
wh5=0.5
wh2=0.2
wh6=0.6
wh3=0.3
wh7=0.7
wh4=0.4
wh8=0.8
1.9
0.3
0.3
1.9
bh1 = 0.2
h1
h2
bh2 = 0.2
1.30
3.06
wo1=0.1
wo3=0.3
wo2=0.2
wo4=0.4
bo1 = 0.3
o1
o2
bo2 = 0.3
1.07 -> o1
1.98->o2
0.295
0.705
Softmax value



X



I1 = new data | I2 = new data | I3 = new data

I4 = new data | I5 = new data | I6 = new data

I7 = new data | I8 = new data | I9 = new data

X

F1+=0.7731 | F2+=-0.027

F3+=-0.027 | F4+=0.7731

w1+=F1+=0.7731

w2+=F2+=-0.027

w3+=F3+=-0.027

w4+=F4+=0.7731

b1+=b1-0.5 * 0.5076
= 0.1 – 0.5 * 0.5076
= -0.1538

wh1+=0.1-0.5x0.268=-0.034

wh2+=0.2-0.5x0.042=0.179

wh3+=0.3-0.5x0.042=0.279

wh4+=0.4-0.5x0.268=0.266

wh5+=0.5-0.5x0.268=0.364

wh6+=0.6-0.5x0.042=0.579

wh7+=0.7-0.5x0.042=0.679

wh8+=0.8-0.5x0=268-0.666

bh1+=0.2-0.5x0.1428=0.1295

bh2+=0.2-0.5x0.1428=0.1295

wo1+= 0.1– (0.5)*(-0.9996) =0.558

wo2+= 0.2– (0.5)*(-2.256) =1.278

wo3+= 0.3– (0.5)*(0.9995) =-0.158

wo4+= 0.4– (0.5)*(2.256) =-0.7678

bo1+= 0.3– (0.5)*(-0.705) =0.625

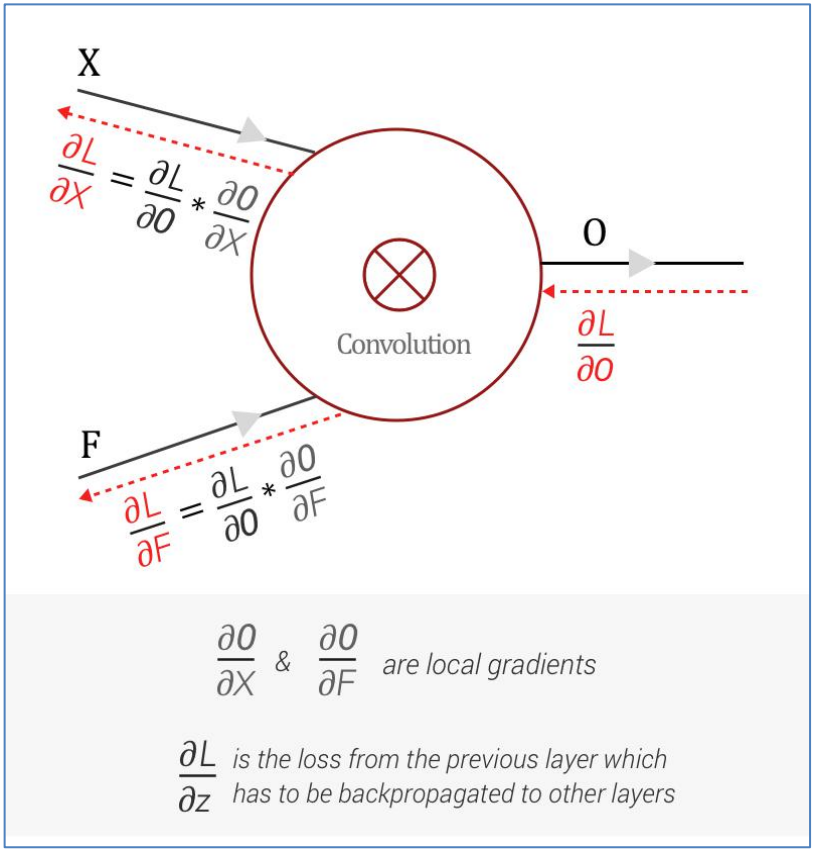
bo2+= 0.3– (0.5)*(0.705) =-0.0525

bo+= (bo1+bo2)/2 =
(-0.057 + 0.657) /. 2 = 0.3 !!

Exercise

CNN Backward Path Calculation

Given the following input, filter, and dL/dO matrices.
a) Compute the matrix dL/dF
b) Compute the matrix dL/dX



Input 3 x3

1	0	-1
1	0	1
0	1	1

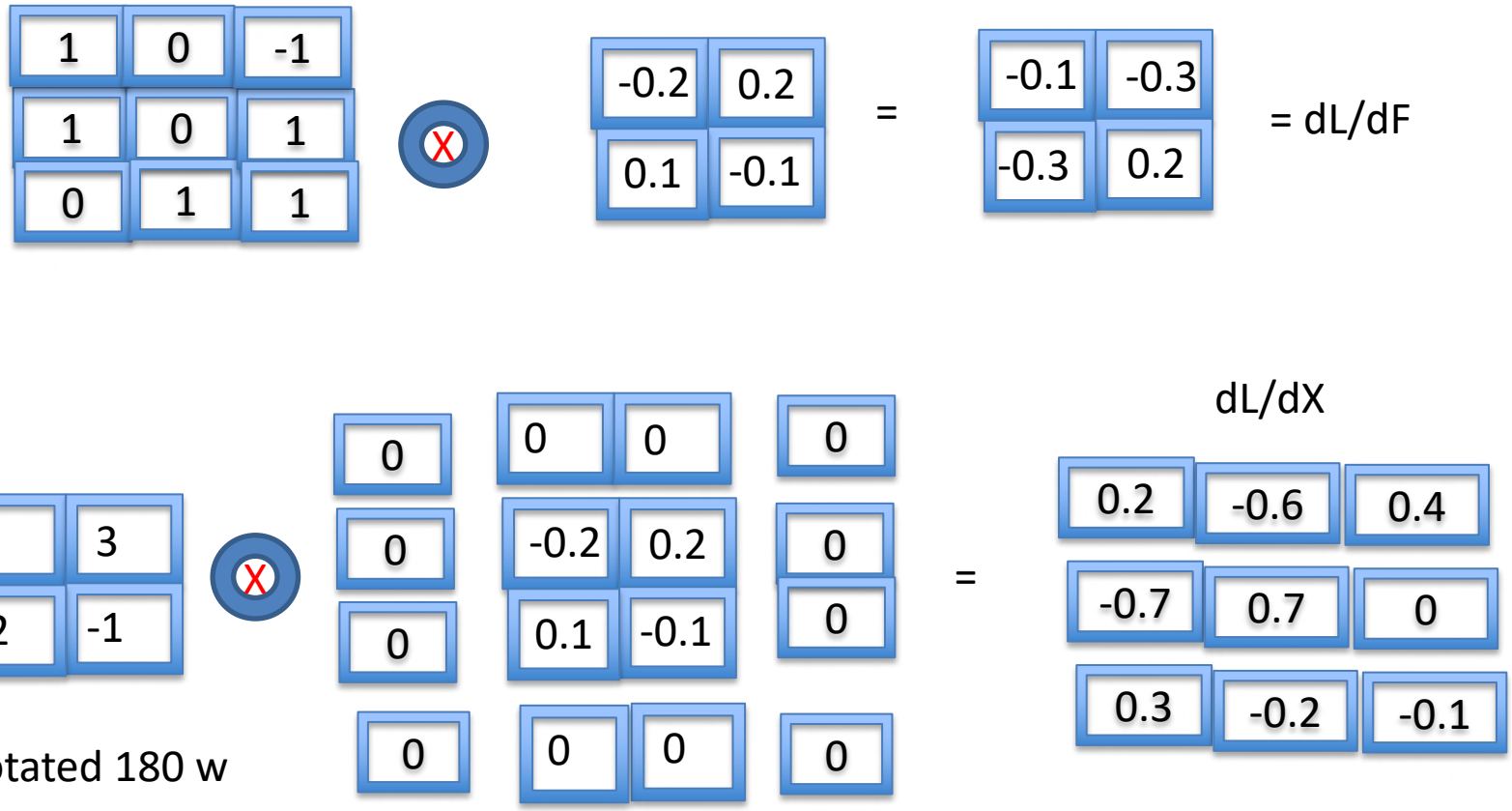
Filter = 2 x 2

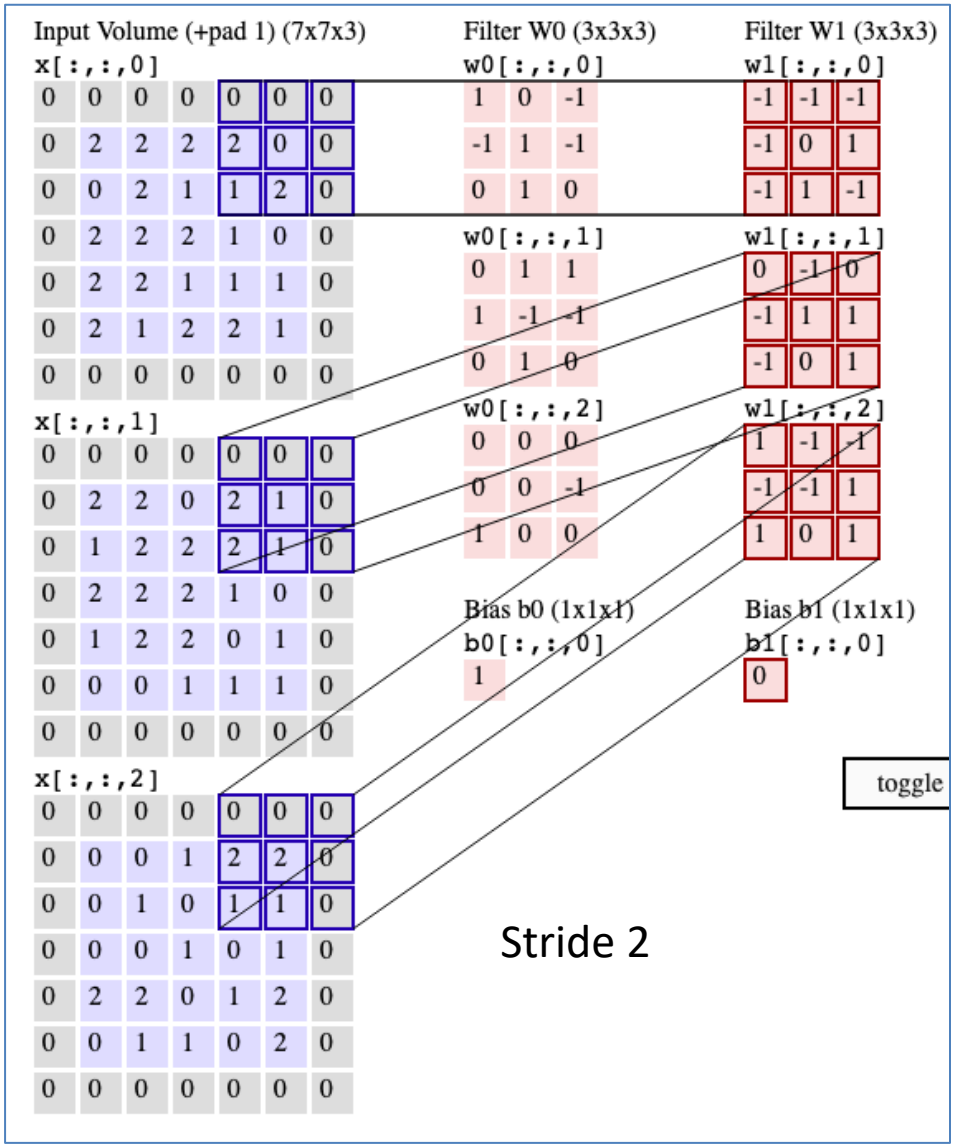
-1	2
3	1

dL/dO =

-0.2	0.2
0.1	-0.1

CNN Backward Path Calculation - Answers





Overlapping max pooling of 2x2, stride 1

Flatten Fully Connected layer of neurons

Fully Connected layer of 2 softmax neurons

Exercises

(Q1) Forward path calculation of CNN

(a) Compute the values of the output matrices after the convolution step.

(b) Compute the output values after the ReLU operation.

(c) Compute the values after the max pooling operation

(d) List the values of the vector after it is flattened based on the row major counting system.

(e) Given the NN, what is the total number of the trainable parameters?

(Q2) Build a python code to compute the values of (Q1), compute also the outputs.

Google drive : Q&A-3-CNN
TF-CNN-CONV.ipynb

Google drive : QA3 : TF-CNN-CONV.ipynb (2D CNN)

```
import tensorflow as tf
import numpy as np

input = tf.constant([
  [[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0]],
  [[0,0,0],[2,2,0],[2,2,0],[2,0,1],[2,2,2],[0,1,2],[0,0,0]],
  [[0,0,0],[0,1,0],[2,2,1],[1,2,0],[1,2,1],[2,1,1],[0,0,0]],
  [[0,0,0],[2,2,0],[2,2,0],[2,2,1],[1,1,0],[0,0,1],[0,0,0]],
  [[0,0,0],[2,1,2],[2,2,2],[1,2,0],[1,0,1],[1,1,2],[0,0,0]],
  [[0,0,0],[2,0,0],[1,0,1],[2,1,1],[2,1,0],[1,1,2],[0,0,0]],
  [[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0]] ],
  shape=[1,7,7,3],dtype=tf.float32)

print("Dimension of input = ",input.shape)
filter = tf.constant([[
  [[1,-1],[0,0],[0,1]],
  [[0,-1],[1,-1],[0,-1]],
  [[-1,-1],[1,0],[0,-1]]],
  [[[-1,-1],[1,-1],[0,-1]],
  [[1,0],[-1,1],[0,-1]],
  [[-1,1],[-1,1],[-1,1]]],
  [[[0,-1],[0,-1],[1,1]],
  [[1,1],[1,0],[0,0]],
  [[0,-1],[0,1],[0,1]]]],
  shape=[3,3,3,2],dtype=tf.float32)

print("Dimension of Filter = ",filter.shape)
bias = tf.constant([1,0],shape=[2],dtype=tf.float32)
```

Input Tensor
= N xH x W x C
= 1 x 7 x 7 x 3

Input Tensor
H(i), W(j), C(k)
For N=1,1
for i =1, 7
for j = 1, 7
for k=1, 3
imat(n,i,j,k)

Filter Tensor
= H x W x C x M
= 3 x 3 x 3 x 2

Filter Tensor
H(i), W(j), C(k), M(m)
for H =1, 3
for j = 1, 3
for k=1, 3
for m =1,2
fmat(i,j,k,m)

```
# Question(a)
print("The output values after the convolution step are: ")
qa = tf.nn.conv2d(input,filter,strides=[1,2,2,1],padding='VALID')+bias
print("Dimension of qa = ",qa.shape)
print(qa.numpy())

# Question(b)
print("The output values after the ReLU operation are")
qb = tf.nn.relu(qa)
print("Dimension of qb = ",qb.shape)
print(qb.numpy())

# Question(c)
print("The output values after the max pooling operation are")
qc = tf.nn.max_pool(qb,[1,2,2,1],[1,1,1,1],padding='VALID')
print("Dimension of qc = ",qc.shape)
print(qc.numpy())

# Question(d)
print("The values of the vector after it is flattened based on the row major counting system
are")
qd = tf.reshape(tf.constant([[[9,9],[9,9]],[[7,1],[6,0]]]),[-1,8])
print("Dimension of qd = ",qd.shape)
print(qd.numpy())

# Question(e)
print('The total number of the trainable parameters is')
parameters = 3*3*3*2+2 + 0 + 8*8 + 8 + 8*2 + 2
print(parameters)

#Softmax output
flat_float = tf.constant([9,9,9,9,7,1,6,0],dtype=tf.float32)
output = tf.nn.softmax(flat_float)
print("The outputs after the softmax function are")
print("Dimension of output = ",output.shape)
print(output.numpy())
```

Dimension of Input = (1, 7, 7, 3)
Dimension of Filter = (3, 3, 3, 2)

The output values after the convolution step are:
Dimension of qa = (1, 3, 3, 2)
[[[[-2. 7.] [1. 1.] [4. -7.]]
[[1. 6.] [9. -8.] [6. -6.]]
[[2. -7.] [1. -4.] [2. -8.]].]]

The output values after the ReLU operation are
Dimension of qb = (1, 3, 3, 2)
[[[[0. 7.] [1. 1.] [4. 0.]]
[[1. 6.] [9. 0.] [6. 0.]]
[[2. 0.] [1. 0.] [2. 0.]]]]

The output values after the max pooling operation are
Dimension of qc = (1, 2, 2, 2)
[[[[9. 7.] [9. 1.]]
[[9. 6.] [9. 0.]].]]

The values of the vector after it is flattened based on the row major counting system are
Dimension of qd = (1, 8)
[[9 9 9 9 7 1 6 0]]

The total number of the trainable parameters is 146

The outputs after the softmax function are
Dimension of output = (8,)
[2.3891544e-01 2.3891544e-01 2.3891544e-01 2.3891544e-01
3.2333687e-02 8.0147205e-05 1.1894900e-02 2.9484507e-05]

Input Tensor
= N xH x W x C
= 1 x 7 x 7 x 3

Filter Tensor
= H x W x C x M
= 3 x 3 x 3 x 2

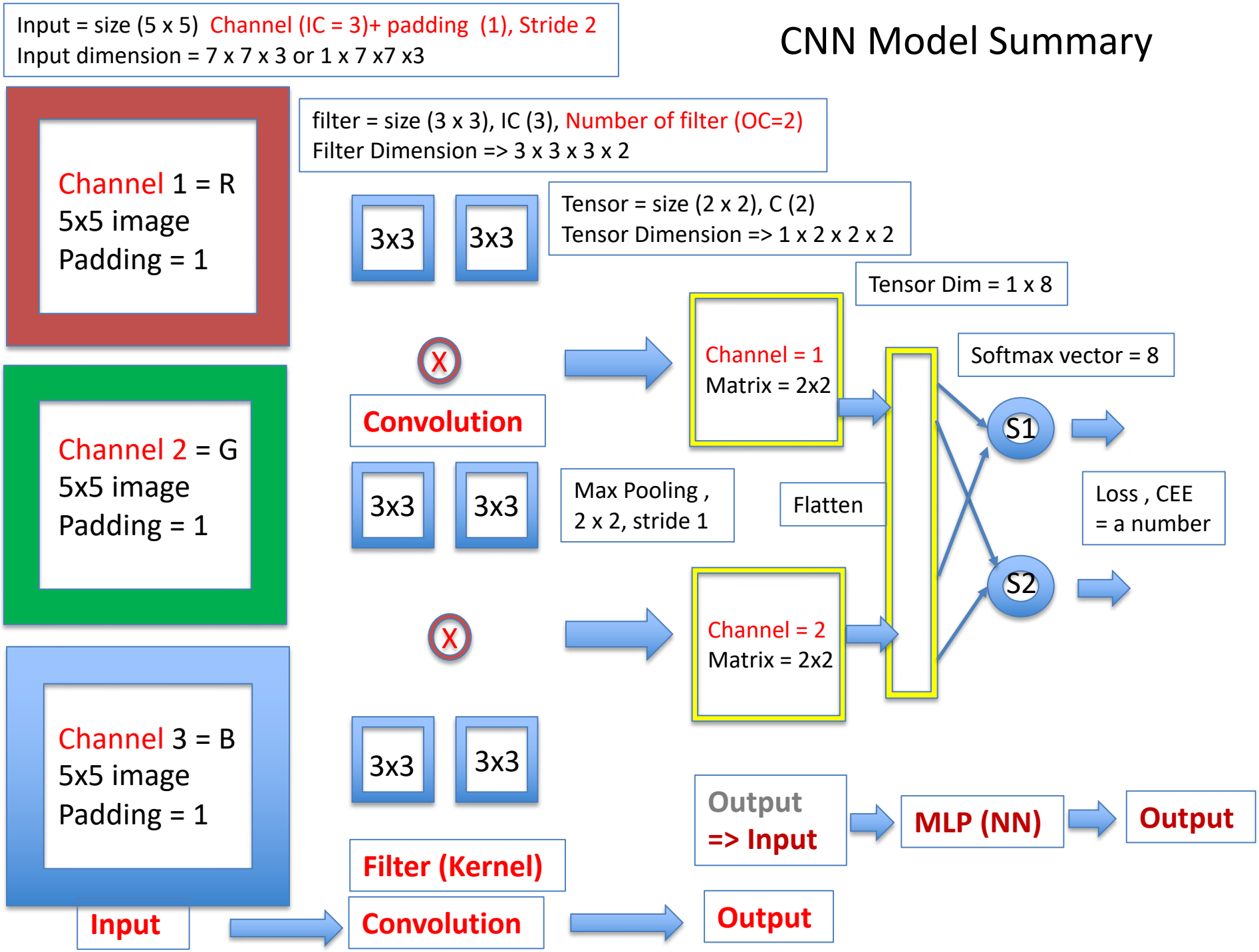
qa, qb tensor
= N x H x W x M
= 1 x 3 x 3 x 2

qc tensor
= N x H x W x M
= 1 x 2 x 2 x 2

qd matrix
= 1 x (2 x 2 x 2) = 1 x 8

softmax vector
= (2 x 2 x 2) = 8

CNN Model Summary



Homework Question

Given the CNN network shown below, an RGB image size of 5x5, filter size of 3x3, padding of 1, stride of 2:

a) Compute the single value of the output after the convolution of the highlighted regions.

b) What is the total number of unknown parameters of this network after an overlapping max pooling of 2x2 of stride 1, followed by a FC layer and a softmax output layer?

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

image = RGD
Im size = 5x5

filter = 3x3

padding = 1

stride = 2

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

Kernel Channel #1

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

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

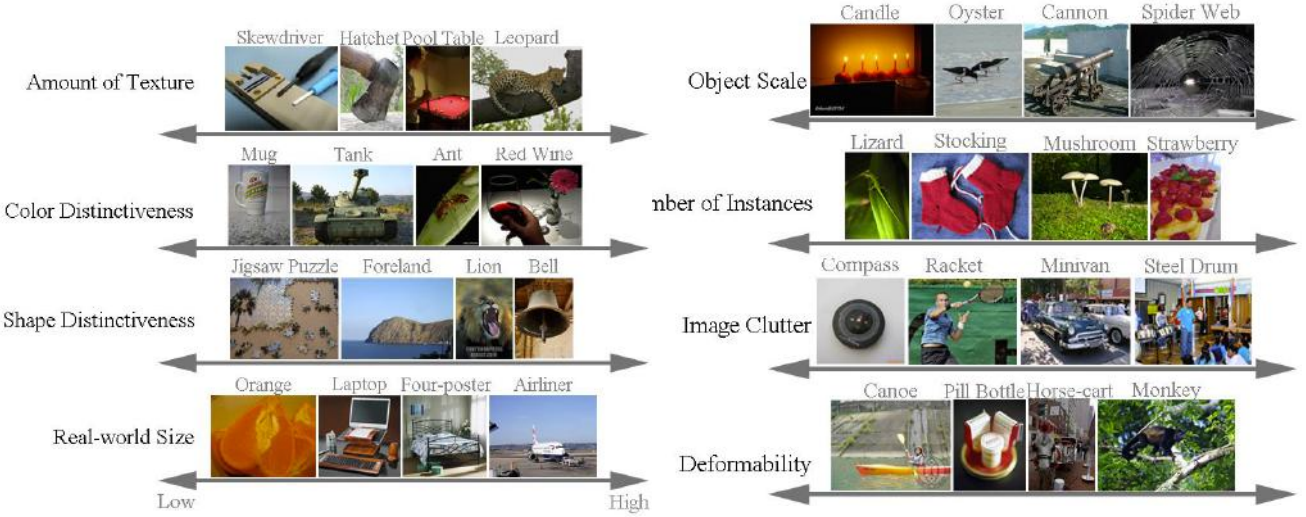
Bias = 2

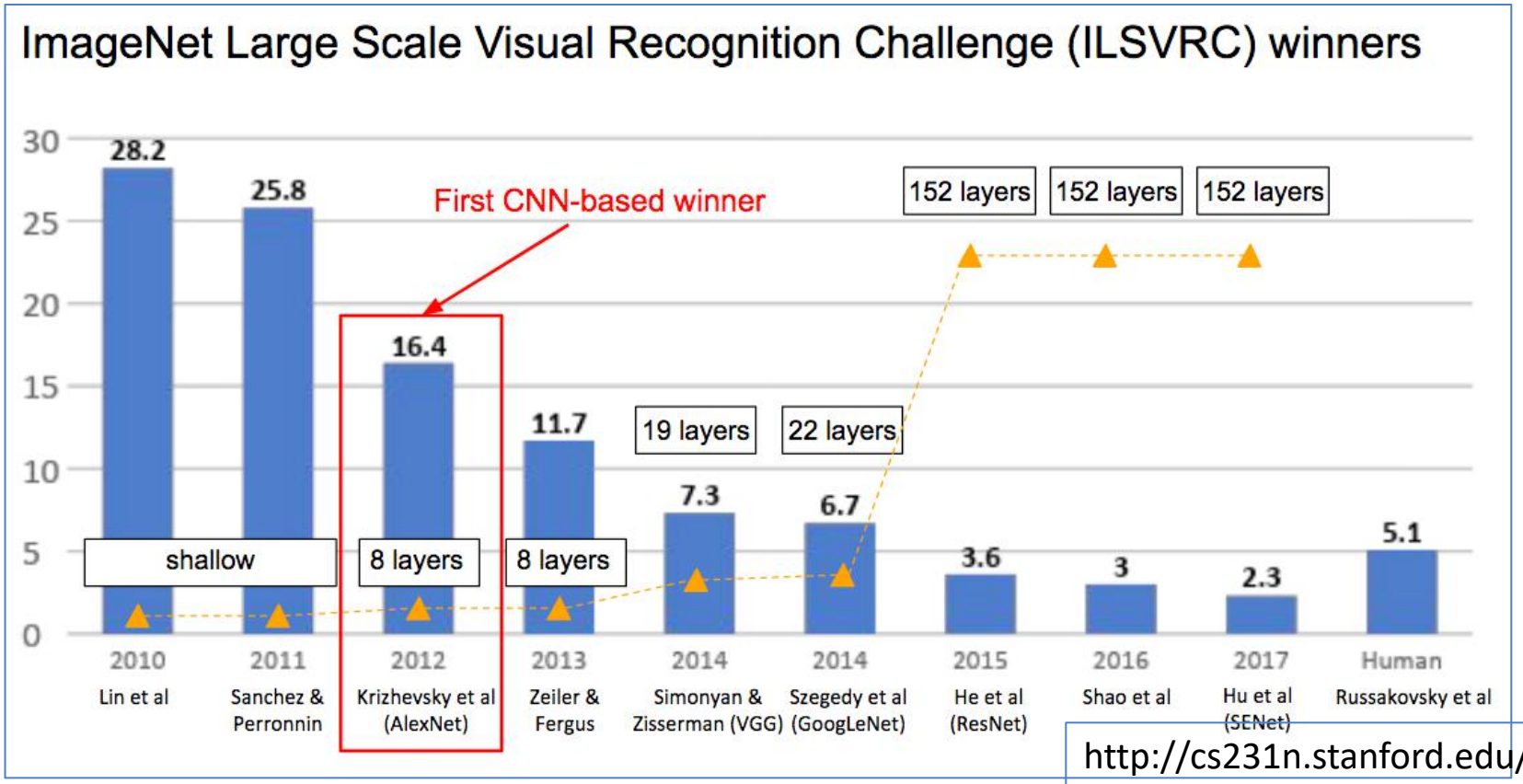
Overlapping max pooling of 2x2, stride 1

Flatten Fully Connected layer of neurons

Fully Connected layer of 2 softmax neurons

- ✓ 1000 classes
- ✓ RGB images
- ✓ Cluttered Images (one category)
- ✓ Full resolution images
- ✓ 1.2 million training images
- ✓ 100 thousand test images





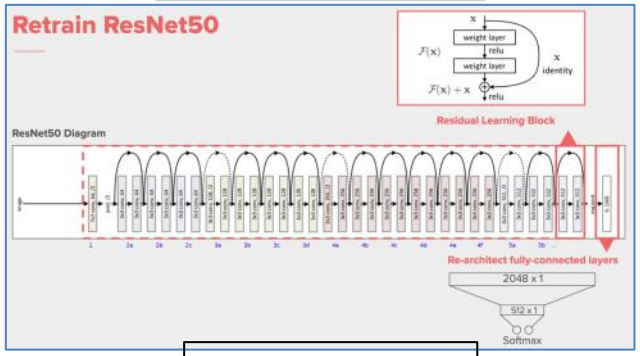
http://cs231n.stanford.edu/

Neural Network Models

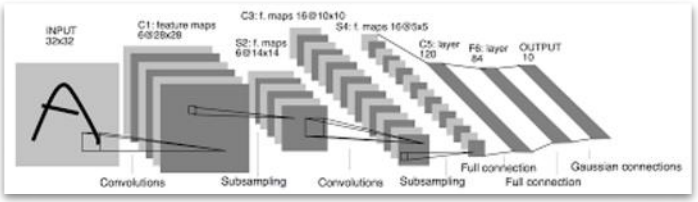
Dataset → Pre-defined Neural Network Model → Fully connected layer # neurons = # classes → Classification

Examples of Pre-defined Neural Network Models

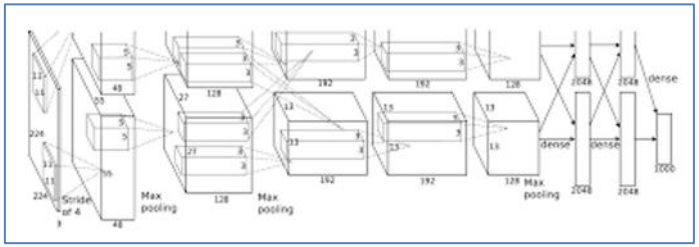
ResNet50



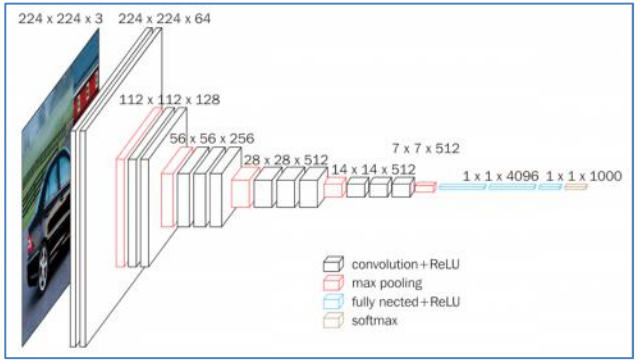
LeNet5



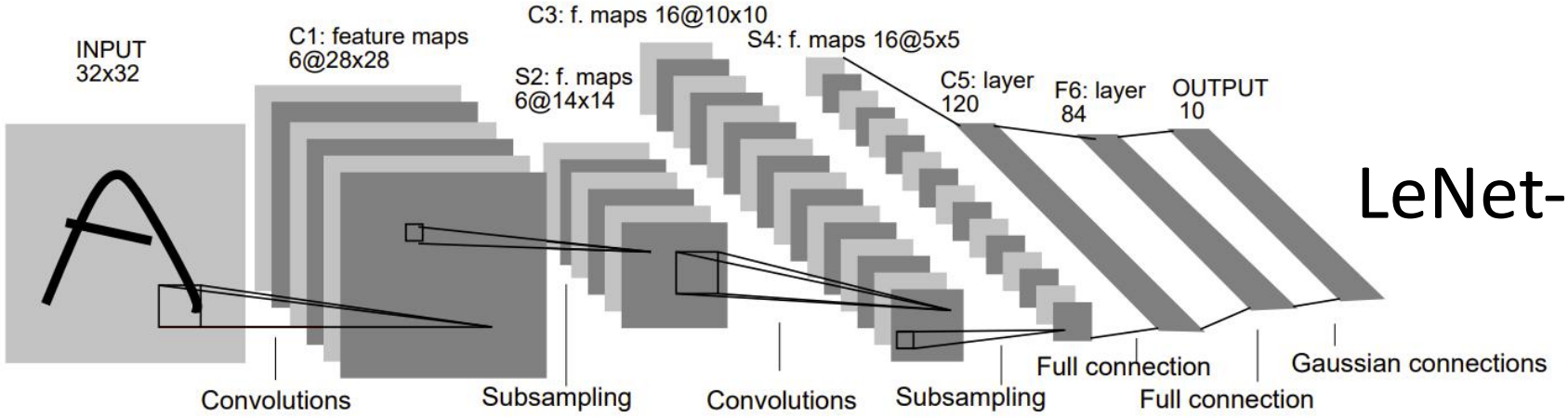
AlexNet



VGG16



Gradient Based Learning Applied To Document Recognition -Y. Lecun,L. Bottou,Y. Bengio,P. Haffner; 1998

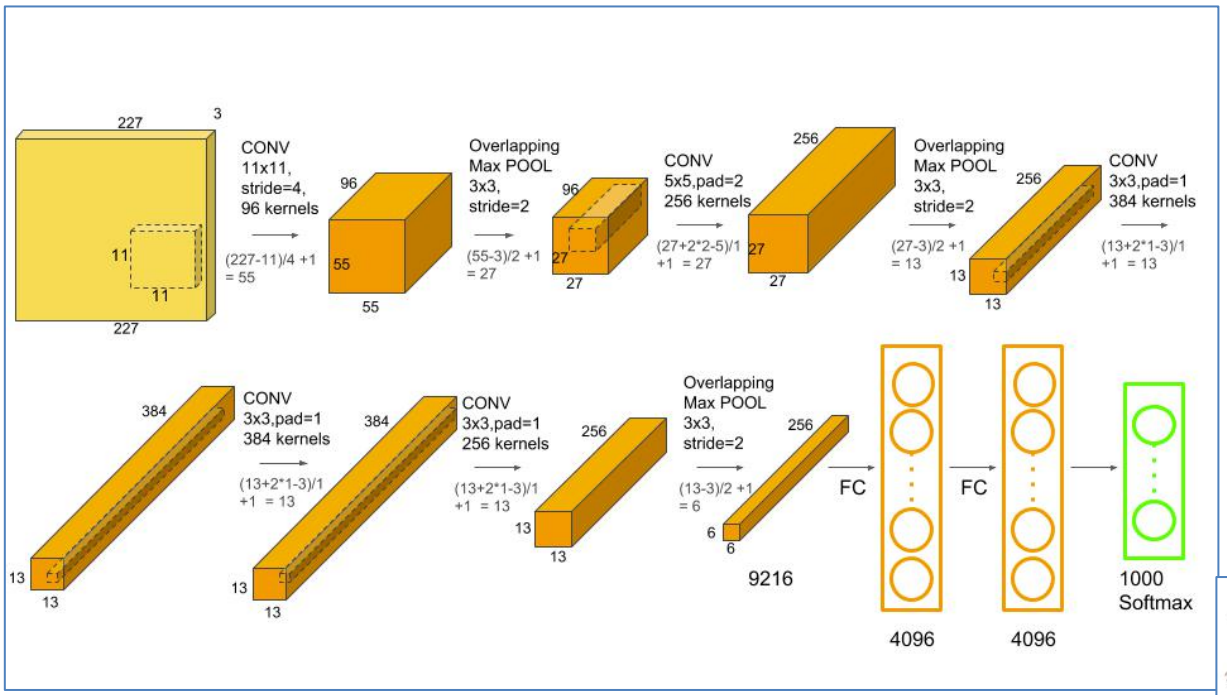


LeNet-5

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

http://cs231n.stanford.edu/

AlexNet

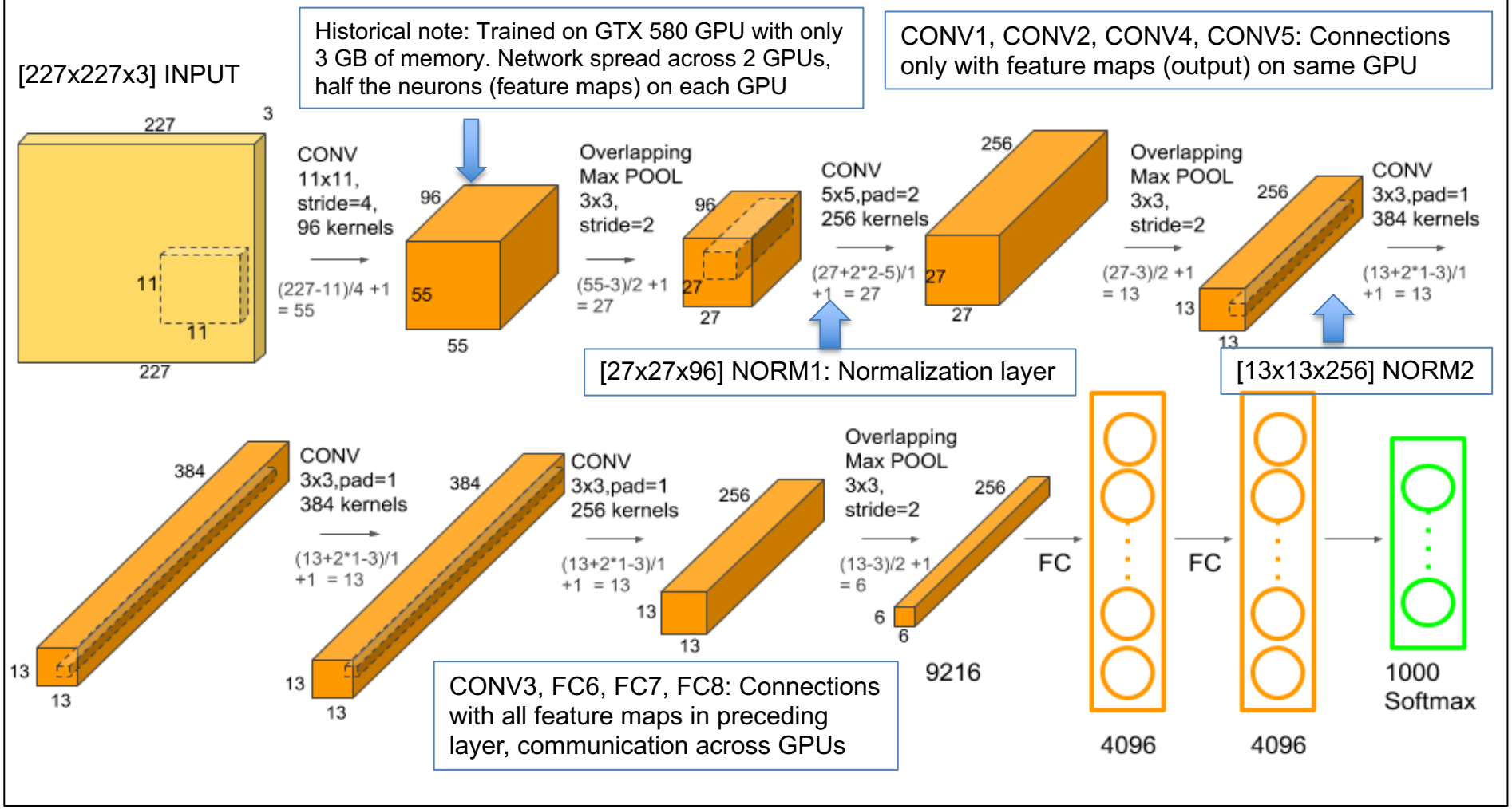


- 60 million parameters
- Use ReLU
- Use Multi-GPU
- Overlapping pooling
- Data augmentation
- Dropout – Bernoulli(p)
- SGD –batch size 128
- Momentum 0.9
- Weight decay 0.0005

$$v_{i+1} := 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i}$$

$$w_{i+1} := w_i + v_{i+1}$$

AlexNet architecture, Training



Details/Retrospectives: - (1) first use of ReLU, (2) used Norm layers (not common anymore), (3) heavy data augmentation, (4) dropout 0.5, (5) batch size 128, (6) SGD Momentum 0.9, (7) Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus, (8) L2 weight decay 5e-4, (9) 7 CNN ensemble: 18.2% -> 15.4%

https://www.learnopencv.com/number-of-parameters-and-tensor-sizes-in-convolutional-neural-network/

Number of Parameters

- ✓ **Input**: Color images of size 227x227x3. The AlexNet paper mentions the input size of 224×224 but that is a typo in the paper.
- ✓ **Conv-1**: The first convolutional layer consists of 96 kernels of size 11×11 applied with a stride of 4 and padding of 0.
- ✓ **MaxPool-1**: The maxpool layer following Conv-1 consists of pooling size of 3×3 and stride 2.
- ✓ **Conv-2**: The second conv layer consists of 256 kernels of size 5×5 applied with a stride of 1 and padding of 2.
- ✓ **MaxPool-2**: The maxpool layer following Conv-2 consists of pooling size of 3×3 and a stride of 2.
- ✓ **Conv-3**: The third conv layer consists of 384 kernels of size 3×3 applied with a stride of 1 and padding of 1.
- ✓ **Conv-4**: The fourth conv layer has the same structure as the third conv layer. It consists of 384 kernels of size 3×3 applied with a stride of 1 and padding of 1.
- ✓ **Conv-5**: The fifth conv layer consists of 256 kernels of size 3×3 applied with a stride of 1 and padding of 1.
- ✓ **MaxPool-3**: The maxpool layer following Conv-5 consists of pooling size of 3×3 and a stride of 2.
- ✓ **FC-1**: The first fully connected layer has 4096 neurons.
- ✓ **FC-2**: The second fully connected layer has 4096 neurons.
- ✓ **FC-3**: The third fully connected layer has 1000 neurons

https://www.learnopencv.com/number-of-parameters-and-tensor-sizes-in-convolutional-neural-network/

Number of Parameters

Size of the Output Tensor (Image) of a Conv Layer

O = Size (width) of output image.

I = Size (width) of input image.

K = Size (width) of kernels used in the Conv Layer

N = Number of kernels.

S = Stride of the convolution operation.

P = Padding.

The size (O) of the output image is given by

$$O = \frac{227 - 11 + 2 \times 0}{4} + 1 = 55$$

$$O = \frac{I - K + 2P}{S} + 1$$

Number of Parameters of a Conv Layer

W_c = Number of weights of the Conv Layer.

B_c = Number of biases of the Conv Layer.

P_c = Number of parameters of the Conv Layer.

K = Size (width) of kernels used in the Conv Layer.

N = Number of kernels.

C = Number of channels of the input image.

$$W_c = K^2 \times C \times N$$
$$B_c = N$$
$$P_c = W_c + B_c$$

$$W_c = 11^2 \times 3 \times 96 = 34,848$$
$$B_c = 96$$
$$P_c = 34,848 + 96 = 34,944$$

Number of Parameters of a MaxPool Layer = 0

Size of Output Tensor (Image) of a MaxPool Layer

O = Size (width) of output image.

I = Size (width) of input image.

S = Stride of the convolution operation.

P_s = Pool size.

The size (O) of the output image is given by

$$O = \frac{55 - 3}{2} + 1 = 27$$

$$O = \frac{I - P_s}{S} + 1$$

https://www.learnopencv.com/number-of-parameters-and-tensor-sizes-in-convolutional-neural-network/

Number of Parameters

The total number of parameters in AlexNet is the sum of all parameters in the 5 Conv Layers + 3 FC Layers. It comes out to a whopping **62,378,344**! The table below provides a summary.

Layer Name	Tensor Size	Weights	Biases	Parameters
Input Image	227x227x3	0	0	0
Conv-1	55x55x96	34,848	96	34,944
MaxPool-1	27x27x96	0	0	0
Conv-2	27x27x256	614,400	256	614,656
MaxPool-2	13x13x256	0	0	0
Conv-3	13x13x384	884,736	384	885,120
Conv-4	13x13x384	1,327,104	384	1,327,488
Conv-5	13x13x256	884,736	256	884,992
MaxPool-3	6x6x256	0	0	0
FC-1	4096x1	37,748,736	4,096	37,752,832
FC-2	4096x1	16,777,216	4,096	16,781,312
FC-3	1000x1	4,096,000	1,000	4,097,000
Output	1000x1	0	0	0
Total				**62,378,344**

W_{ff} = Number of weights of a FC Layer which is connected to an FC Layer.
B_{ff} = Number of biases of a FC Layer which is connected to an FC Layer.
P_{ff} = Number of parameters of a FC Layer which is connected to an FC Layer.
F = Number of neurons in the FC Layer.
F_{-1} = Number of neurons in the previous FC Layer.

$$W_{ff} = F_{-1} \times F$$
$$B_{ff} = F$$
$$P_{ff} = W_{ff} + B_{ff}$$

In the above equation, $F_{-1} \times F$ is the total number of connection weights from neurons of the previous FC Layer the neurons of the current FC Layer. The total number of biases is the same as the number of neurons (F).

Example: The last fully connected layer of AlexNet is connected to an FC Layer. For this layer, $F_{-1} = 4096$ and $F = 1000$. Therefore,

$$W_{ff} = 4096 \times 1000 = 4,096,000$$
$$B_{ff} = 1,000$$
$$P_{ff} = W_{ff} + B_{ff} = 4,097,000$$

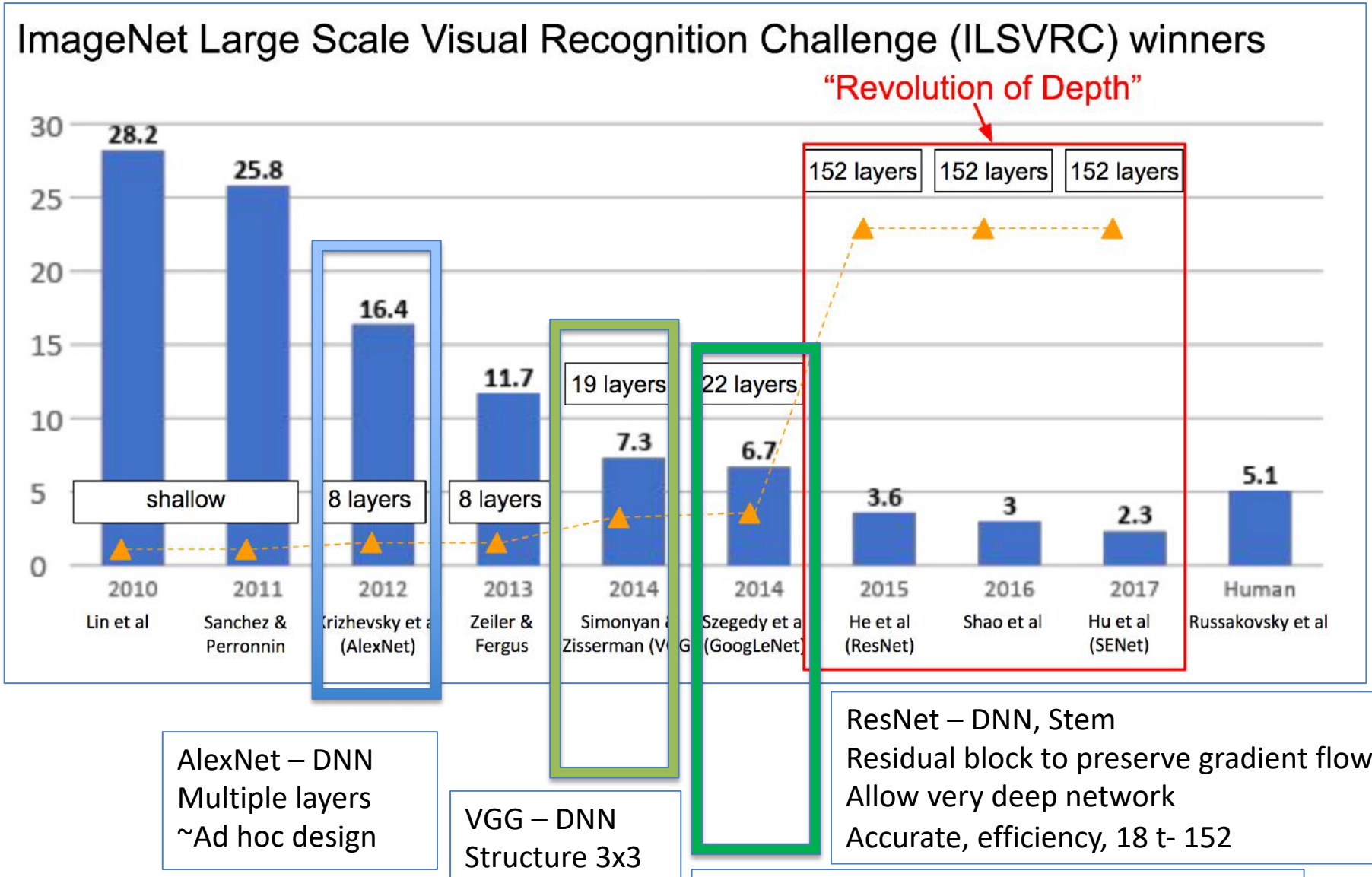
W_{cf} = Number of weights of a FC Layer which is connected to a Conv Layer.
B_{cf} = Number of biases of a FC Layer which is connected to a Conv Layer.
O = Size (width) of the output image of the previous Conv Layer.
N = Number of kernels in the previous Conv Layer.
F = Number of neurons in the FC Layer.

$$W_{cf} = O^2 \times N \times F$$
$$B_{cf} = F$$
$$P_{cf} = W_{cf} + B_{cf}$$

Example: The first fully connected layer of AlexNet is connected to a Conv Layer. For this layer, $O = 6$, $N = 256$ and $F = 4096$. Therefore,

$$W_{cf} = 6^2 \times 256 \times 4096 = 37,748,736$$
$$B_{cf} = 4096$$
$$P_{cf} = W_{cf} + B_{cf} = 37,752,832$$

https://www.learnopencv.com/number-of-parameters-and-tensor-sizes-in-convolutional-neural-network/



AlexNet – DNN
Multiple layers
~Ad hoc design

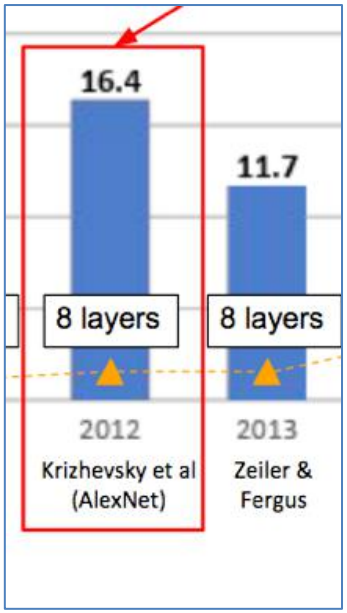
VGG – DNN
Structure 3x3
Deep NN,
Lots of parameters
Accuracy

GoogleNet – DNN
Stem to reduce feature size
Block structure for efficiency
Group pooling to reduce parameter count

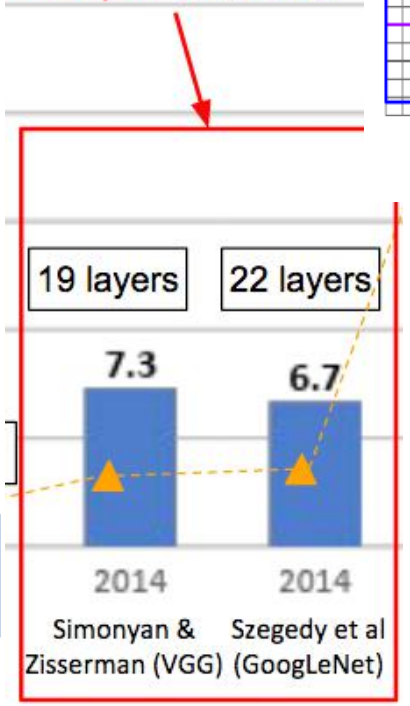
ResNet – DNN, Stem
Residual block to preserve gradient flow
Allow very deep network
Accurate, efficiency, 18 t- 152

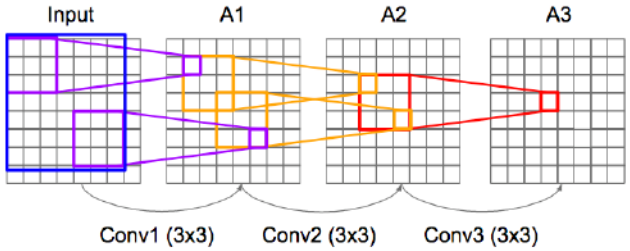
http://cs231n.stanford.edu/

VGG16, 19, Structured Network Design



11 x 11 filter
AlexNet





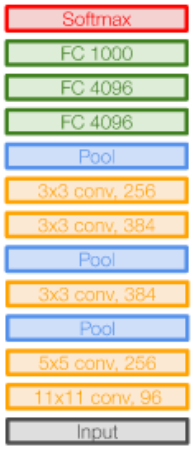
VGG Design rules:

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

- ✓ Stack of three 3x3 conv (stride 1) layers has same effective receptive field as one 7x7 (ZFNet) conv layer
- ✓ But deeper, more non-linearities
- ✓ Fewer parameters: 3 * (32C2) vs. 72C2 for C channels per layer



AlexNet

7 x 7 filter
ZFNet

8 layers (AlexNet) -> 16 - 19 layers (VGG16Net)

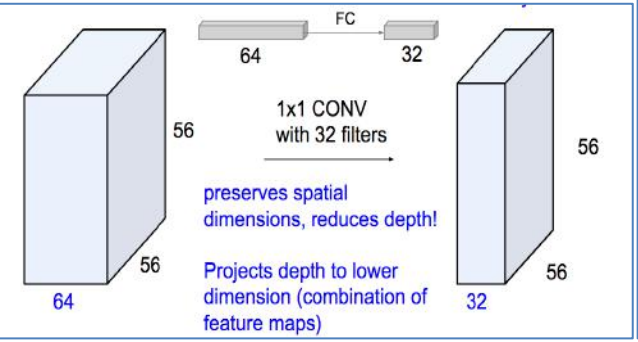
Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

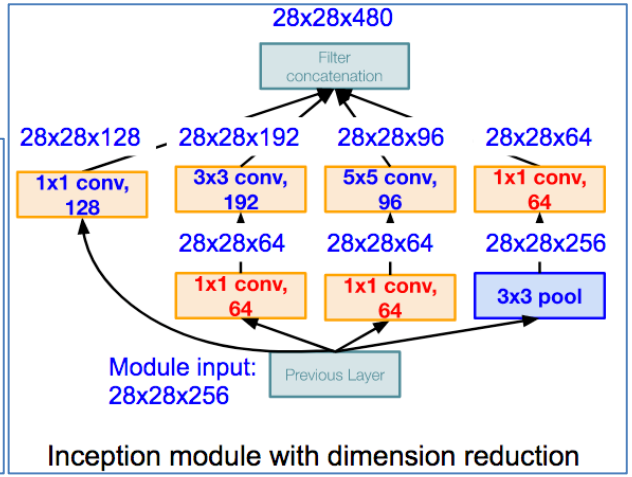
11.7% top 5 error in ILSVRC'13 (ZFNet)->
7.3% top 5 error in ILSVRC'14

Small filters, Deeper networks

http://cs231n.stanford.edu/

GoogLeNet



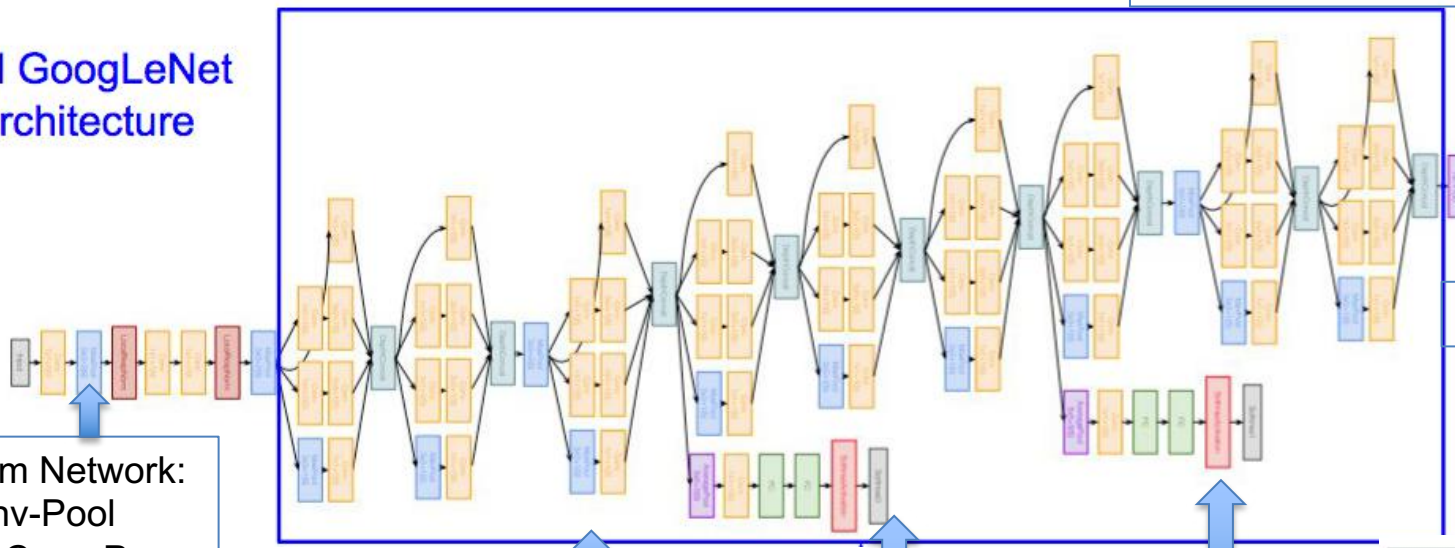


Inception module with dimension reduction

Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

Note: after the last convolutional layer, a global average pooling layer is used that spatially averages across each feature map, before final FC layer. No longer multiple expensive FC layers

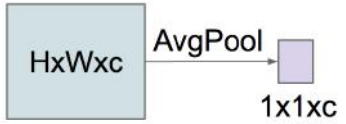
Full GoogLeNet architecture



Classifier output

Stem Network:
Conv-Pool
-2x Conv-Poo

Stacked Inception Modules

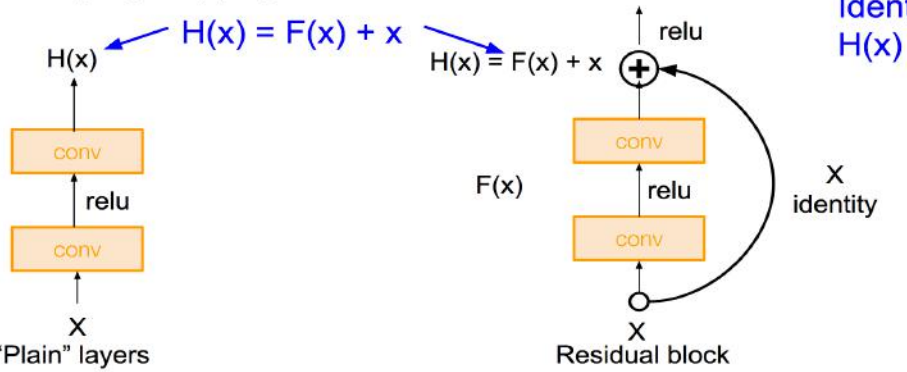


classification outputs to inject additional gradient at lower layers (AvgPool-1x1Conv-FC-FC-Softmax)

http://cs231n.stanford.edu/

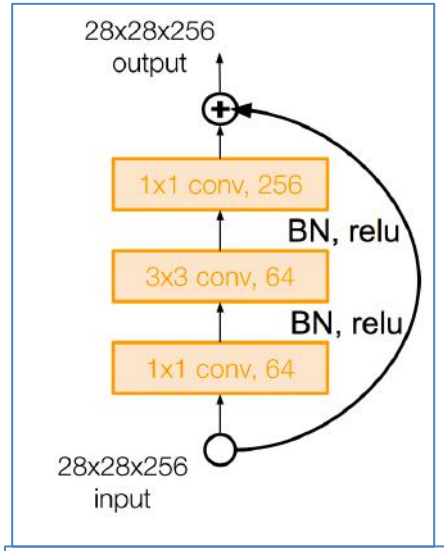
ResNet

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Identity mapping:
H(x) = x if F(x) = 0

Use layers to fit **residual** F(x) = H(x) - x instead of H(x) directly



For deeper networks (ResNet-50+), use "bottleneck" layer to improve efficiency (similar to GoogLeNet)

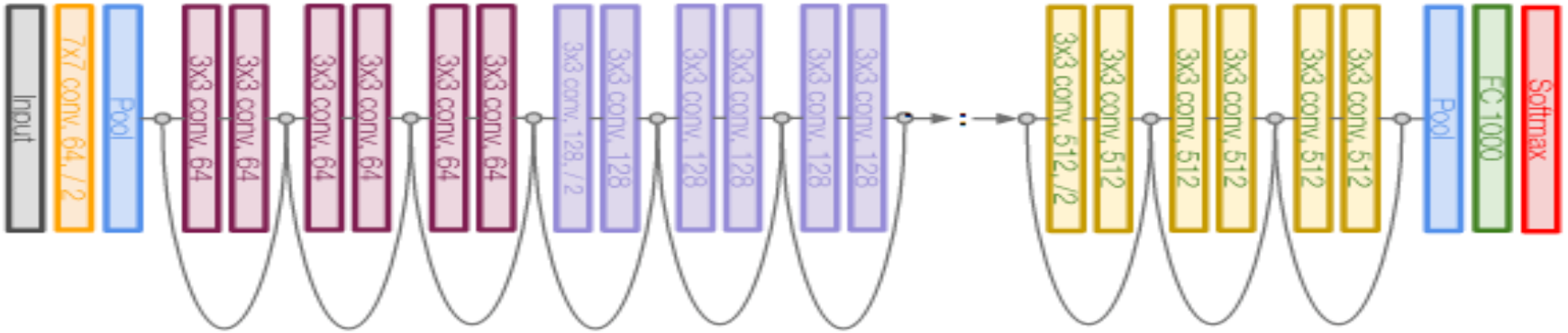
- ✓ Full ResNet architecture:-
- ✓ Stack residual blocks
- ✓ Every residual block has two 3x3 conv layers
- ✓ Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- ✓ Additional conv layer at the beginning (stem)
- ✓ No FC layers at the end (only FC 1000 to output classes)

Total depths of 18, 34, 50, 101, or 152 layers for ImageNet

1x1 conv, 256 filters projects back to 256 feature maps (28x28x256)

3x3 conv operates over only 64 feature maps

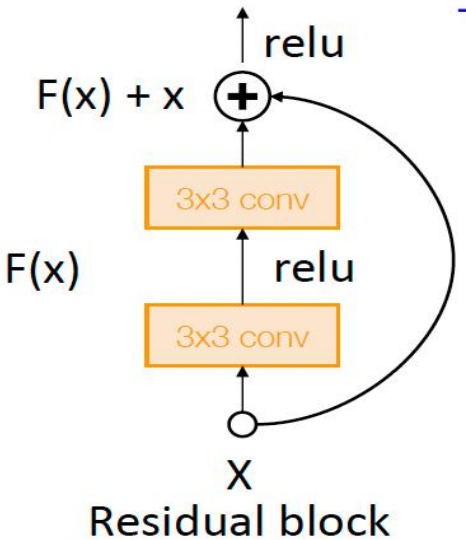
1x1 conv, 64 filters to project to 28x28x64

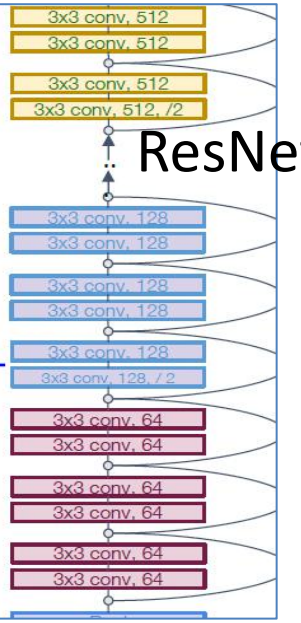


A residual network is a stack of many residual blocks

Regular design, like VGG: each residual block has two 3x3 conv

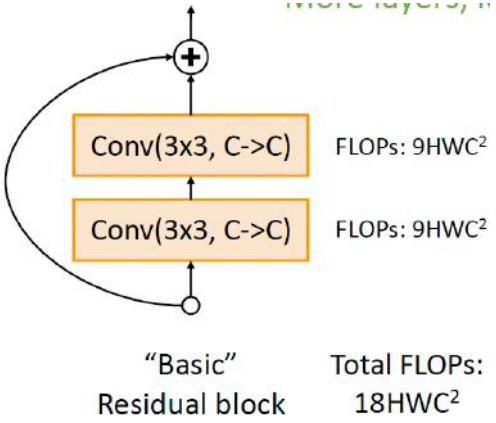
Network is divided into **stages**: the first block of each stage halves the resolution (with stride-2 conv) and doubles the number of channels





ResNet

Basic Block
ResNet18, 34



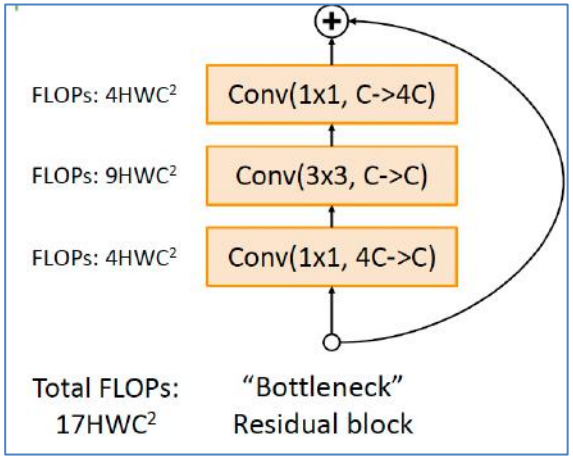
ResNet-18:
Stem: 1 conv layer
Stage 1 (C=64): 2 res. block = 4 conv
Stage 2 (C=128): 2 res. block = 4 conv
Stage 3 (C=256): 2 res. block = 4 conv
Stage 4 (C=512): 2 res. block = 4 conv
Linear

ImageNet top-5 error: 10.92
GFLOP: 1.8

ResNet-34:
Stem: 1 conv layer
Stage 1: 3 res. block = 6 conv
Stage 2: 4 res. block = 8 conv
Stage 3: 6 res. block = 12 conv
Stage 4: 3 res. block = 6 conv
Linear

ImageNet top-5 error: 8.58
GFLOP: 3.6

Bottlenect Block
ResNet50 …



https://nnabla.org/paper/imagenet_in_224sec.pdf
https://paperswithcode.com/sota/image-classification-on-imagenet

ILSVRC Complexity

GoogLeNet: most efficient

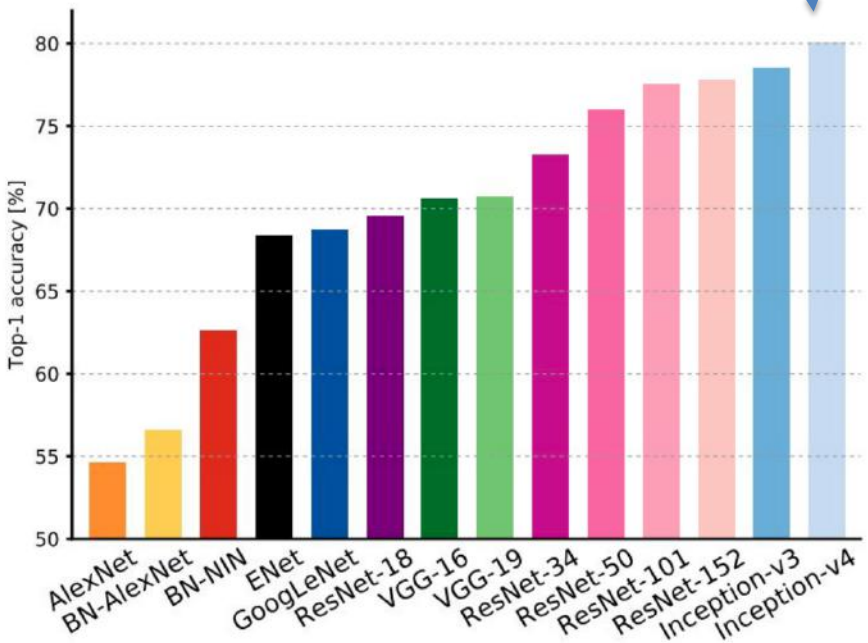
VGG: most parameters, most operations

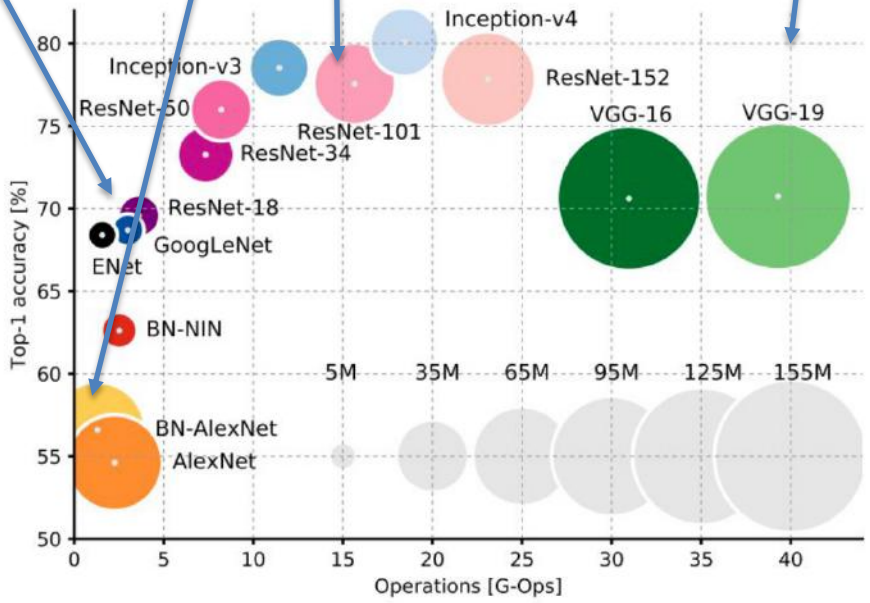
ResNet:Moderate efficiency depending on model, highest accuracy

Inception-v4: Resnet + Inception!

AlexNet:Smaller compute, still memory heavy, lower accuracy

Comparing complexity...





An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Acknowledgements and References

This portion of the tutorial contains many extracted materials from many online websites and courses. Listed below are the major sites. This portion of the materials is not intended for public distribution. Please visit the sites websites for detail contents. If you are beginner users of DNN, I suggest to read the following list of websites in its order.

1) http://neuralnetworksanddeeplearning.com

2) https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist

3) https://www.deeplearning.ai/deep-learning-specialization/

4) http://cs231n.stanford.edu/

5) MIT 6.S191, https://www.youtube.com/watch?v=njKP3FqW3Sk

6) https://livebook.manning.com/book/deep-learning-with-python/about-this-book/

7) https://www.fast.ai/

8) https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/

9) https://www.nersc.gov/users/training/gpus-for-science/gpus-for-science-2020/

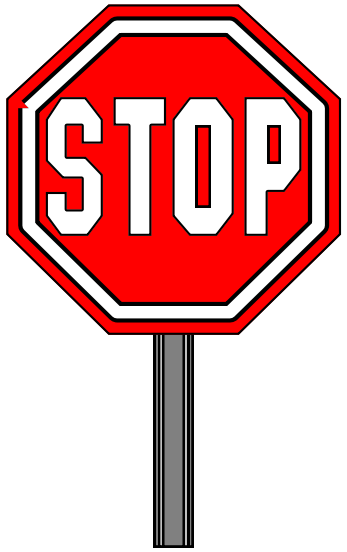
10) https://oneapi-src.github.io/oneDNN/

11) http://www.cs.cornell.edu/courses/cs4787/2020sp/

12) More sites and free books are listed in www.jics.utk.edu/actia → ML

13) https://machinelearningmastery.com

The End



- The End!