Lecture 20

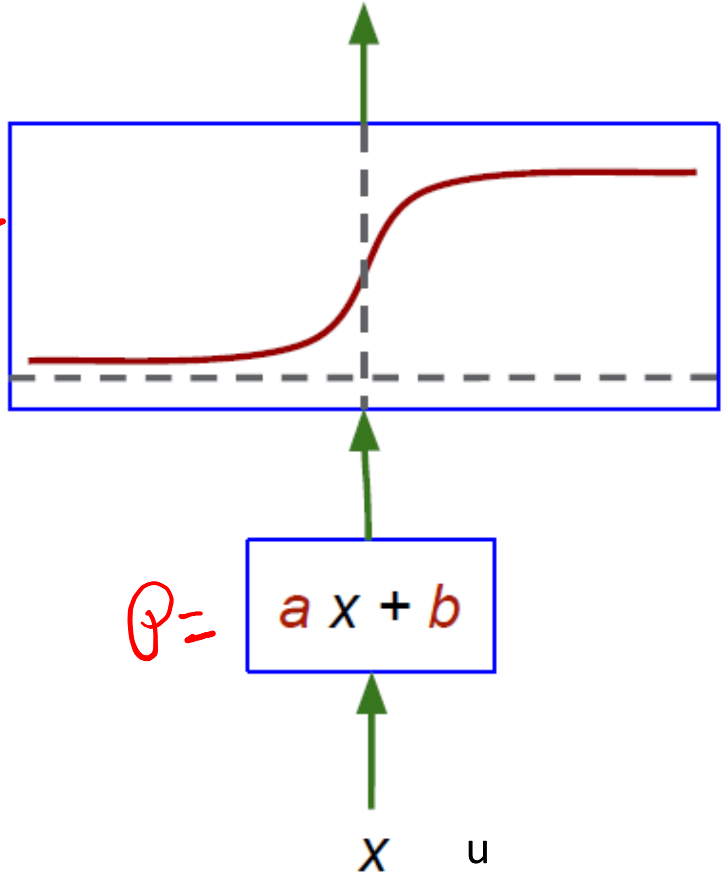
Batch Normalization

Slides based on

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, ICML 2015

Batch Normalization

activation



$\sigma(P)$

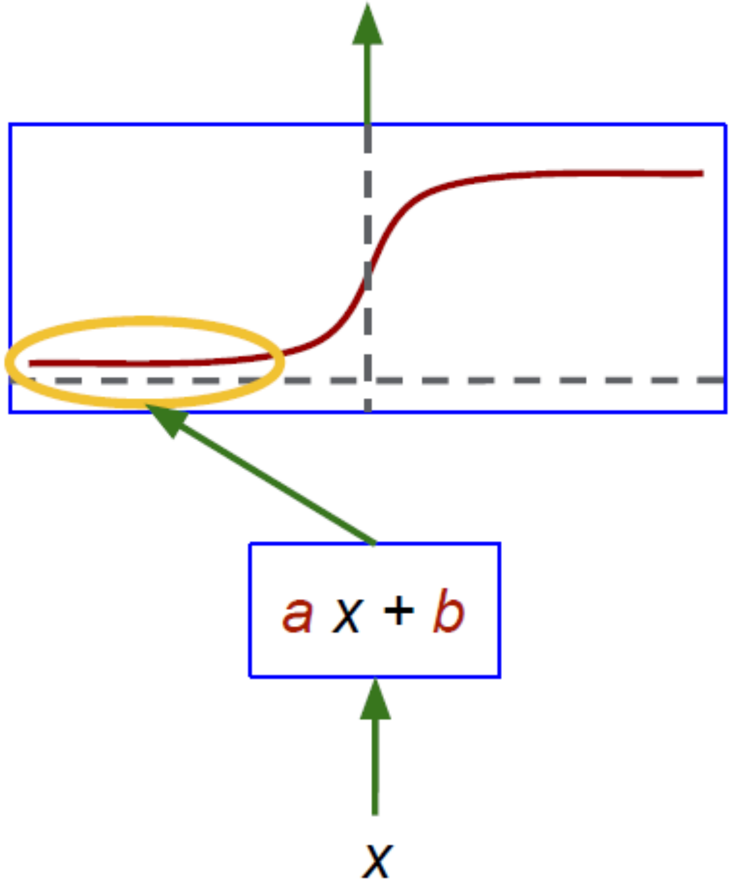
$P =$

$a \in \mathbb{R}^d$

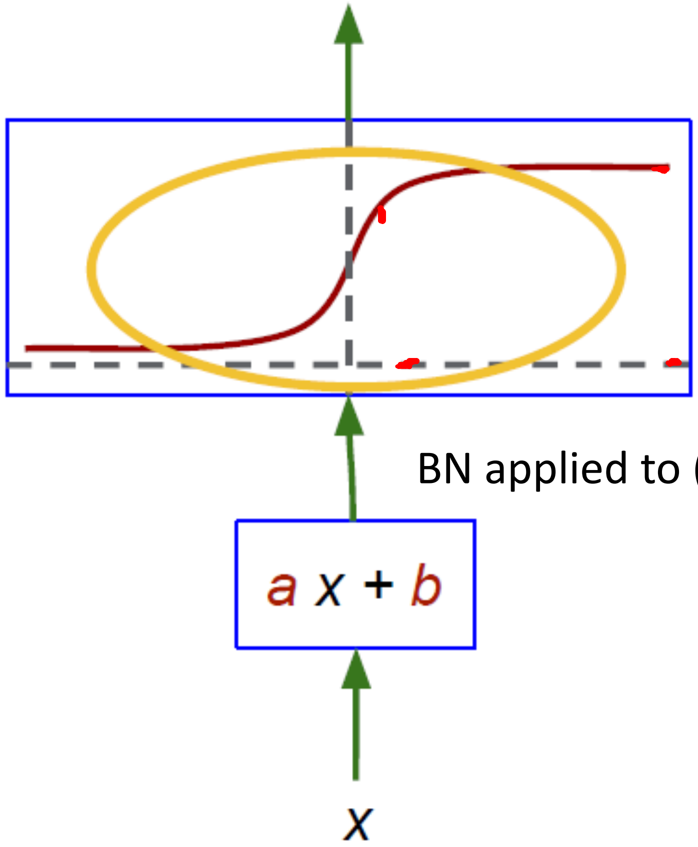
$x \in \mathbb{R}^d$

$b \to$ bias.

Batch Normalization



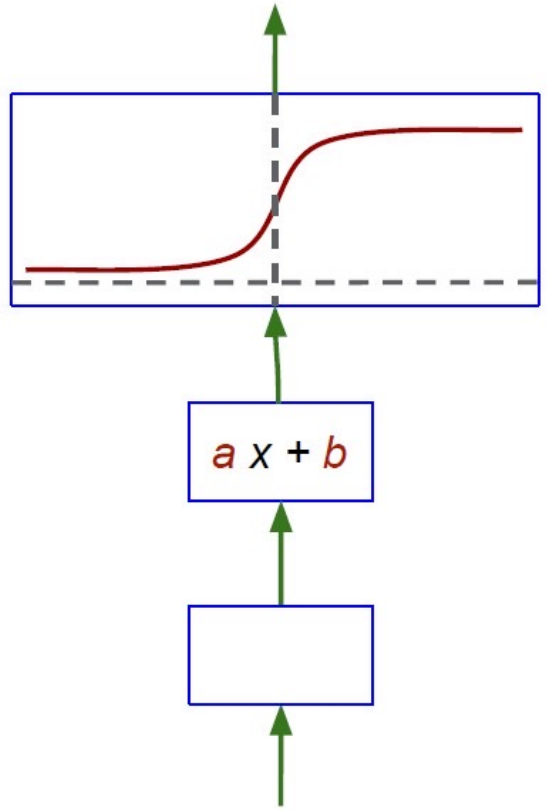
Batch Normalization



BN applied to (ax+b) ie before activation

Effect of changing input distribution

- Careful initialization
- Small learning rates
- Rectifiers



Internal covariate shift

We define *Internal Covariate Shift* as the change in the distribution of network activations due to the change in network parameters during training

- Layer input distributions change during training

$$\ell = F_2(\, F_1(\mathrm{u}, \Theta_1),\ \Theta_2)$$

- Normalize each activation:

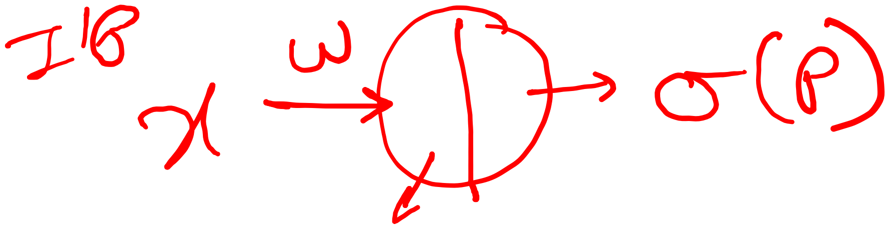
Pre-activation

$$x \mapsto \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]}}$$

$P_1 = 1, \; P_2 = 2, \; P_3 = 3$

$\mu_B = 2$

$\sigma_B^2 = 1$



$\mu_B \rightarrow$ mean of P

$\mu_B = w^T \frac{(x_1 + x_2 + x_3)}{3} + b$

$P = w^T x + b$

batch of $m = 3$ sampled.

i/p $x_1, x_2, x_3 \in \mathbb{R}^d$

$\sigma_B^2 = \frac{1}{m-1} \sum_{i=1}^{3} (P_i - \mu_B)^2$

$\gamma, \beta \rightarrow$ learnable.

Mini-batch mean: $$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

Mini-batch variance: $$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

Normalize: $$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad \sim N(0,1)$$

Scale and shift: $$y_i \leftarrow \underline{\gamma\widehat{x}_i} + \underline{\beta} \quad \sim N(\beta, \gamma^2)$$

- Replace batch statistics with population statistics

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \Rightarrow \widehat{x} \leftarrow \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}}$$

Exp. moving avg.

$$\mu_{EMA}^{t=0} = 0$$

$$\mu_{B}^{t=0}$$

$$\mu_{EMA}^{t} = \alpha \mu_B^t + (1-\alpha)\mu_{EMA}^{t-1}$$

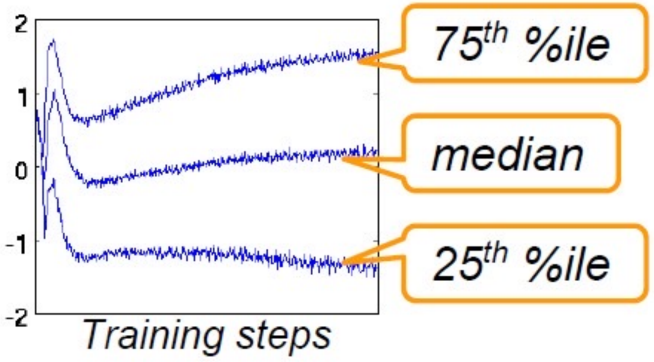
$$\sigma_{EMA}^{2,t} = \alpha \sigma_B^{2,t} + (1-\alpha)\sigma_{EMA}^{2,t-1}$$

$\mu_B^t \rightarrow$ Batch mean at iteration t

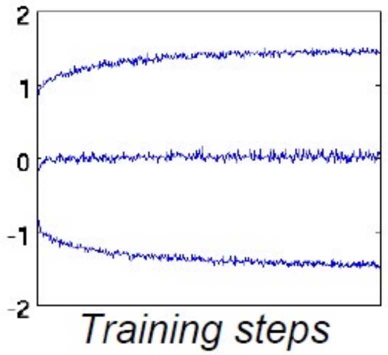
$\alpha \in [0,1]$ constant.

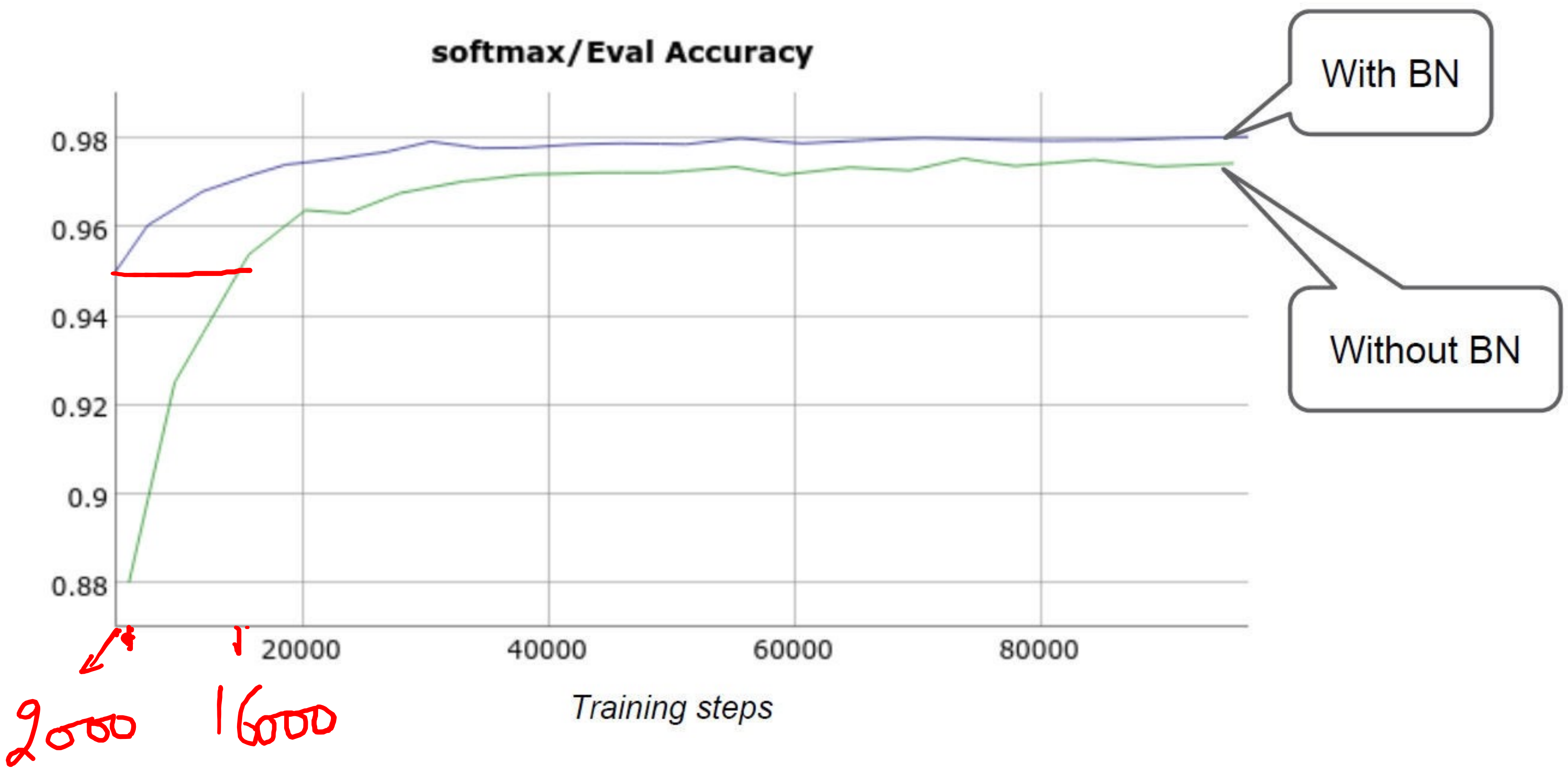
- MNIST: 3 FC layers + softmax, 100 logistic units per hidden layer
- Distribution of inputs to a typical sigmoid, evolving over 100k steps:

Without BN:

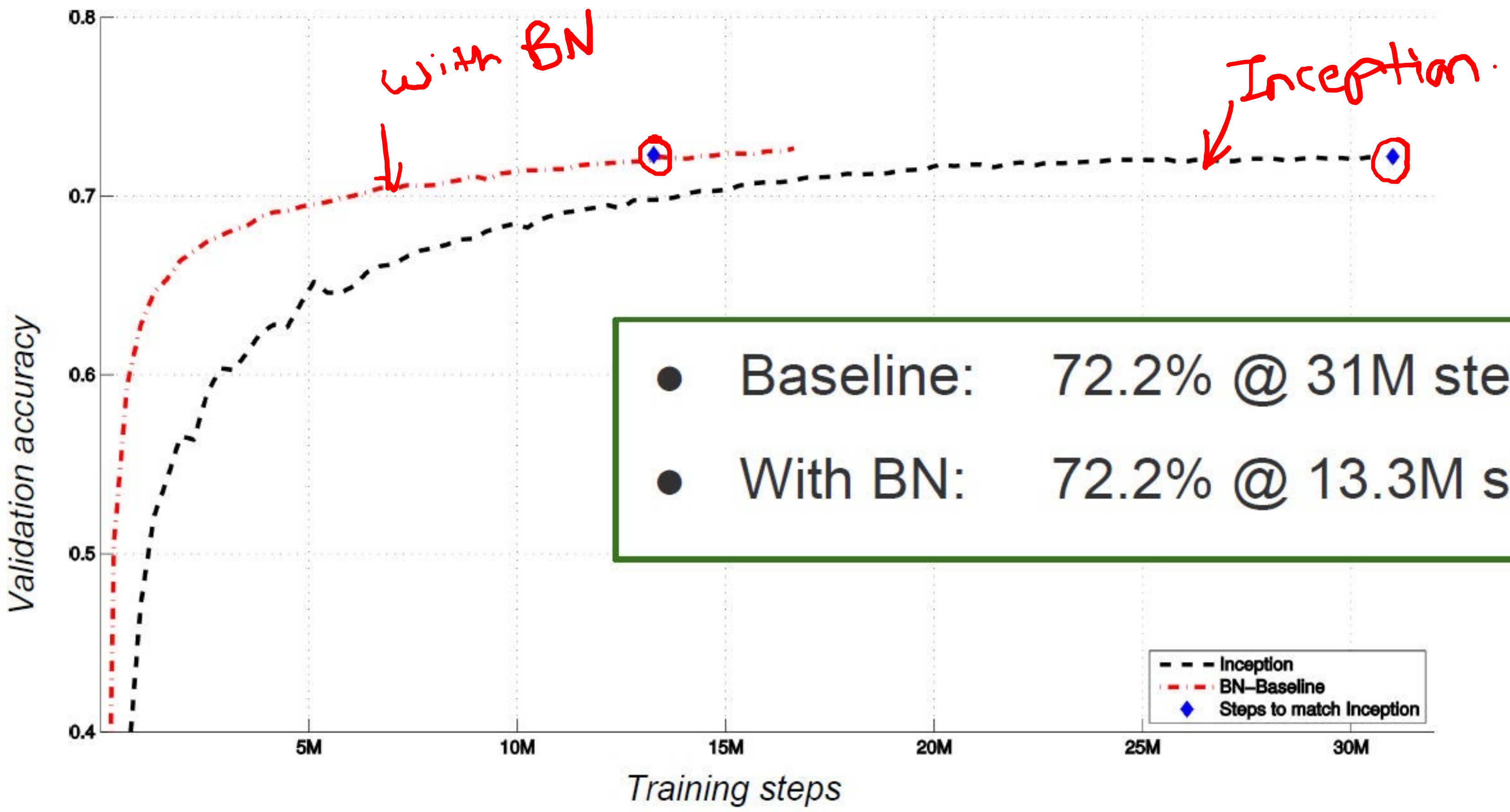


With BN:



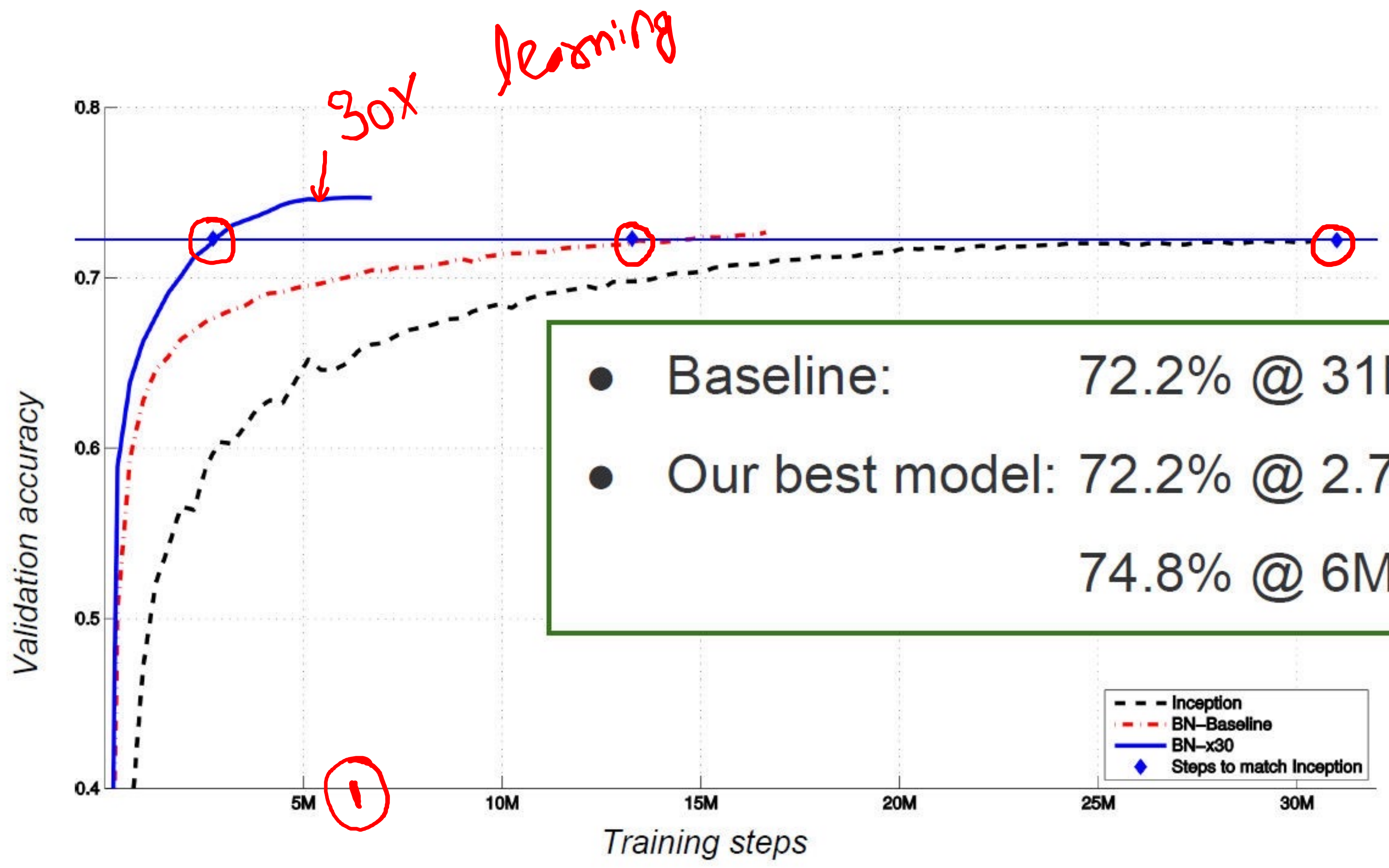

softmax/Eval Accuracy
0.98
0.96
0.94
0.92
0.9
0.88
20000
40000
60000
80000
Training steps
With BN
Without BN
2000
16000

- Inception: deep convolutional ReLU model
- Distributed SGD with momentum
- Batch Normalization applied at every convolutional layer



- Baseline: 72.2% @ 31M steps
- With BN: 72.2% @ 13.3M steps

- Batch Normalization enables higher learning rate
    - Increased 30x
- Removing dropout improves validation accuracy
    - Batch Normalization as a regularizer?



- Baseline: 72.2% @ 31M steps
- Our best model: 72.2% @ 2.7M steps
  74.8% @ 6M steps

Step 1) Import Libraries

```
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, Input
from keras.utils import np_utils

#Other types of layers
from keras.layers import LSTM
from keras.layers import Conv1D, Conv2D, Conv3D, MaxPooling2D

from keras.layers.normalization import BatchNormalization

import matplotlib.pyplot as plt
%matplotlib inline

np.random.seed(2017)
```

Step 3) Define model architecture

Form 1)

```
In [11]: model = Sequential()
         model.add(Dense(512, activation='relu', use_bias=True,  input_shape=(784,)))
         model.add(Dense(128, activation='relu', use_bias=True))
         model.add(Dense(10, activation='softmax', use_bias=True))
```

Form 2)

```
In [91]: from keras.models import Model

         X_inp = Input(shape=(784,))
         h1 = Dense(512, activation='relu', use_bias=True)(X_inp)
         h2 = Dense(128, activation='relu', use_bias=True)(h1)
         h3 = Dense(10, activation='softmax', use_bias=True)(h2)

         model = Model(inputs=X_inp, outputs=h3)
```

Step 3) Define model architecture
(Alternatives for activation)

```
model.add(Dense(128, activation='relu', use_bias=True))
```

Step 3) Define model architecture

(Other attributes of Dense layer)

```
keras.layers.core.Dense(units, activation=None, use_bias=True,
                        kernel_initializer='glorot_uniform',
                        bias_initializer='zeros',
                        kernel_regularizer=None,
                        bias_regularizer=None,
                        activity_regularizer=None,
                        kernel_constraint=None,
                        bias_constraint=None)
```

Example:

```
from keras.constraints import maxnorm
model.add(Dense(64, kernel_constraint=max_norm(2.)))
```

Available constraints

max_norm(max_value=2, axis=0): maximum-norm constraint

non_neg(): non-negativity constraint

unit_norm(): unit-norm constraint, enforces the matrix to have unit norm along the last axis

Step 3) Define model architecture

```
keras.layers.core.Dropout(rate,
                          noise_shape=None,
                          seed=None)
```

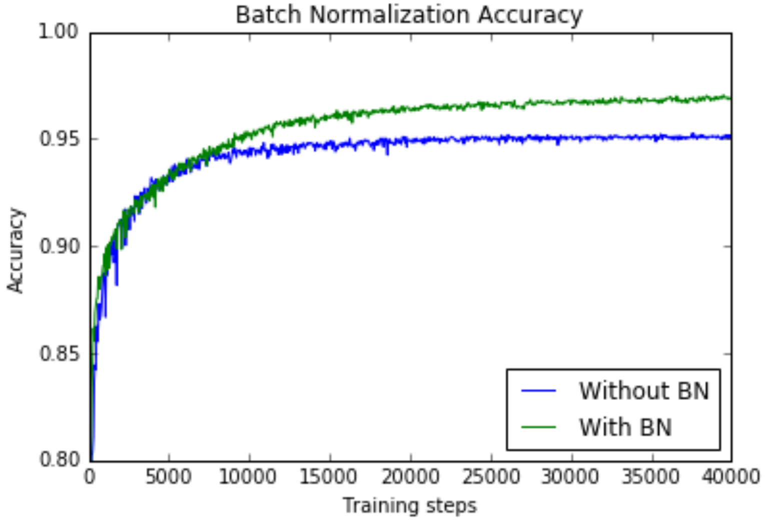
Example:

```
model.add(Dense(128, activation='relu', use_bias=True))
model.add(Dropout(0.2))
```

Step 3) Define model architecture (Batch Normalization Layers)

Example:

```
model = Sequential()
model.add(Dense(64, input_dim=14))
model.add(BatchNormalization())
model.add(Activation('tanh'))
model.add(Dropout(0.5))
```



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Step 4) Compile model (Loss functions)

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

Custom loss function

```
import theano.tensor as T

def myLoss(y_true, y_pred):
    cce = T.mean(T.sqr(y_true-y_pred))
    return cce
```

```
model.compile(optimizer='adadelta', loss=myLoss)
```

Available loss functions:

- mean_squared_error
- mean_absolute_error
- mean_absolute_percentage_error
- mean_squared_logarithmic_error
- squared_hinge
- hinge
- categorical_hinge
- logcosh
- categorical_crossentropy
- sparse_categorical_crossentropy
- binary_crossentropy
- kullback_leibler_divergence
- poisson
- cosine_proximity

Step 4) Compile model (Optimizers)

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

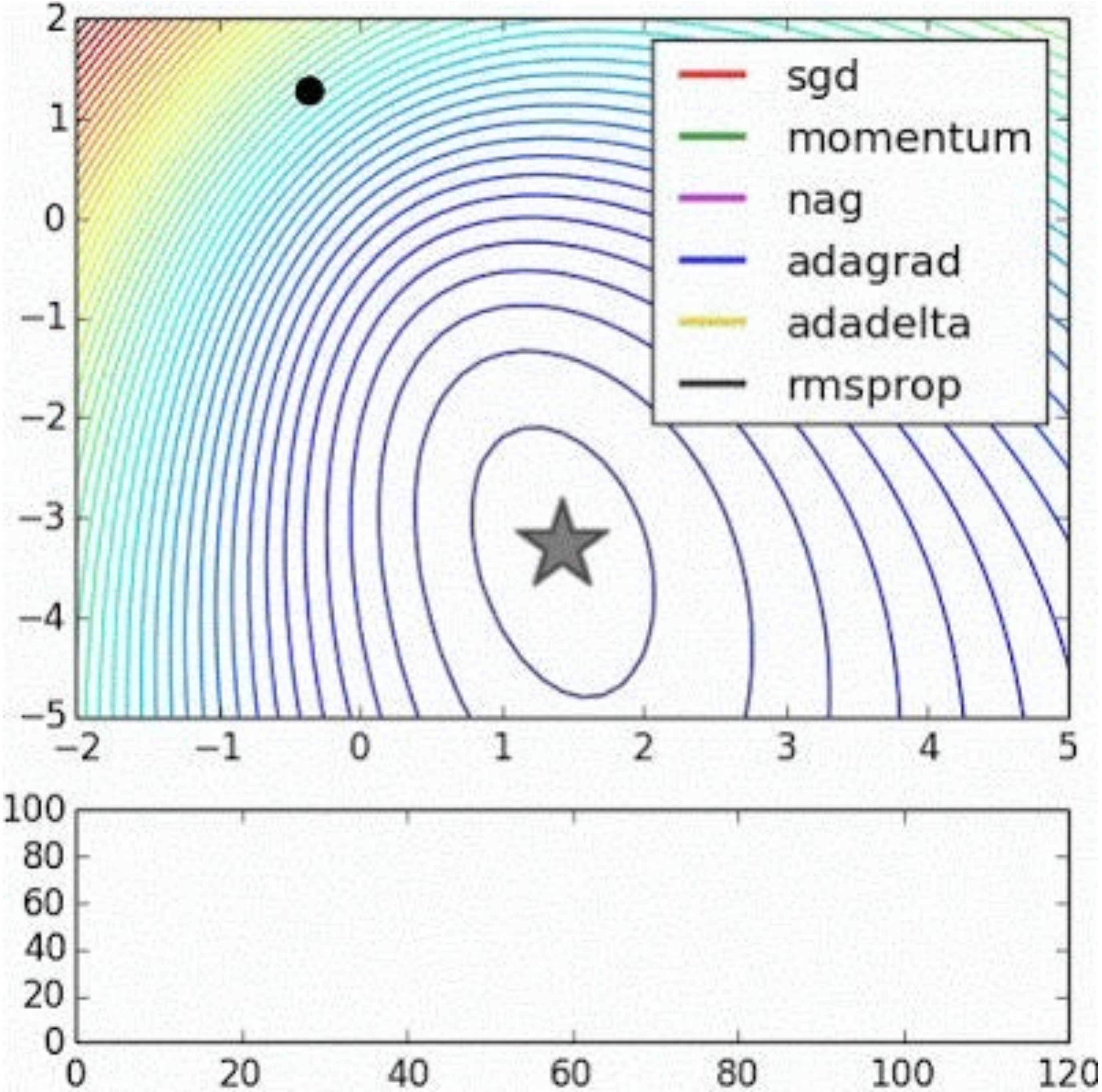
Available loss functions:

- **SGD**
- **RMSprop**
- **Adagrad**
- **Adadelta**
- **Adam**
- **Adamax**
- **Nadam**
- **TFOptimizer**

Adagrad

```
adagrad = keras.optimizers.Adagrad(lr=0.01, epsilon=1e-08, decay=0.0)
```

```
model.compile(optimizer=adagrad, loss=myLoss)
```


sgd
momentum
nag
adagrad
adadelta
rmsprop
2
1
0
−1
−2
−3
−4
−5
−2
−1
0
1
2
3
4
5
100
80
60
40
20
0
0
20
40
60
80
100
120

Deep Learning

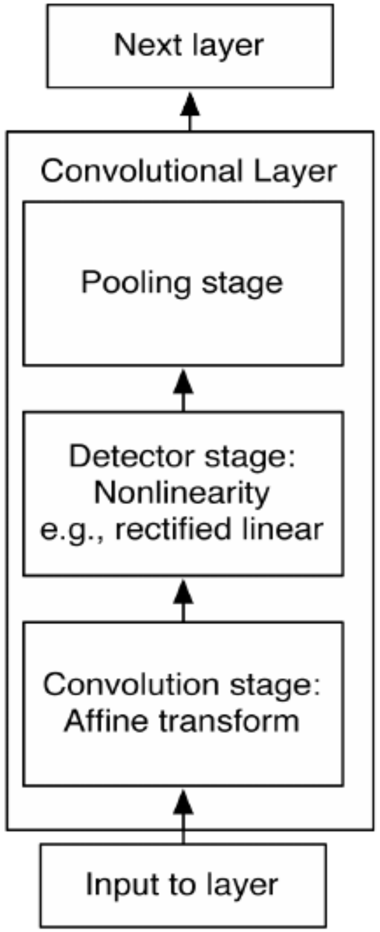
Convolutional Neural Network (CNNs)

Slides are partially based on Book, Deep Learning

by Bengio, Goodfellow, and Aaron Courville, 2015

Convolutional Networks

Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.



Convolution

This operation is called convolution.

$$s(t) = \int x(a)w(t-a)da$$

The convolution operation is typically denoted with an asterisk:

$$s(t) = (x * w)(t)$$

Discrete convolution

If we now assume that *x* and *w* are defined only on integer *t*, we can define the discrete convolution:

$$s[t] = (x * w)(t) = \sum_{a=-\infty}^{\infty} x[a]w[t-a]$$

In practice

we often use convolutions over more than one axis at a time.

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[m,n] K[i-m, j-n]$$

$I(m,n)$ $\quad$ $K(-m,-n)$

The input is usually a multidimensional array of data.

The kernel is usually a multidimensional array of parameters that should be learned.

we assume that these functions are zero everywhere but the finite set of points for which we store the values.

we can implement the infinite summation as a summation over a finite number of array elements.

Outside input, is zero padded

I

1	2	3
4	5	6
7	8	9

Input

n \ m	-1	0	1
-1	-1	-2	-1
0	0	0	0
1	1	2	1

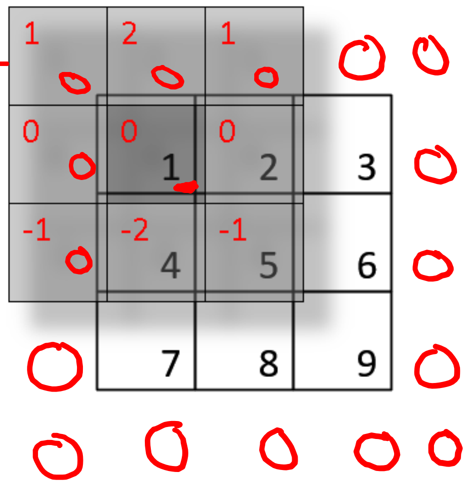
Kernel

-13	-20	-17
-18	-24	-18
13	20	17

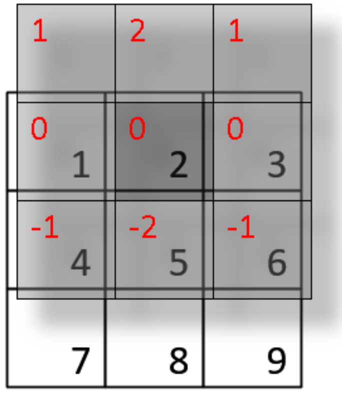
Output

K

Output dimension (3+3-1) x (3+3-1): we can take 3x3 from center of this 5x5 output flipped by 180°



$$
\begin{aligned}
y[0,0] &= \sum_j \sum_i x[i,j] \cdot h[0-i, 0-j] \\
&= \quad x[-1,-1] \cdot h[1,1] + x[0,-1] \cdot h[0,1] + x[1,-1] \cdot h[-1,1] \\
&\quad + x[-1,0] \cdot h[1,0] \quad + x[0,0] \cdot h[0,0] \quad + x[1,0] \cdot h[-1,0] \\
&\quad + x[-1,1] \cdot h[1,-1] + x[0,1] \cdot h[0,-1] + x[1,1] \cdot h[-1,-1] \\
&= \quad 0 \cdot 1 + 0 \cdot 2 + 0 \cdot 1 \\
&\quad + 0 \cdot 0 + 1 \cdot 0 + 2 \cdot 0 \\
&\quad + 0 \cdot (-1) + 4 \cdot (-2) + 5 \cdot (-1) \\
&= -13
\end{aligned}
$$



$$
\begin{aligned}
y[1,0] &= \sum_j \sum_i x[i,j] \cdot h[1-i, 0-j] \\
&= \quad x[0,-1] \cdot h[1,1] + x[1,-1] \cdot h[0,1] + x[2,-1] \cdot h[-1,1] \\
&\quad + x[0,0] \cdot h[1,0] \quad + x[1,0] \cdot h[0,0] \quad + x[2,0] \cdot h[-1,0] \\
&\quad + x[0,1] \cdot h[1,-1] + x[1,1] \cdot h[0,-1] + x[2,1] \cdot h[-1,-1] \\
&= \quad 0 \cdot 1 + 0 \cdot 2 + 0 \cdot 1 \\
&\quad + 1 \cdot 0 + 2 \cdot 0 + 3 \cdot 0 \\
&\quad + 4 \cdot (-1) + 5 \cdot (-2) + 6 \cdot (-1) \\
&= -20
\end{aligned}
$$

convolution and cross-correlation

convolution is commutative

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i-m, j-n]K[m,n]$$

Cross-correlation,

i+m,j+n

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i+m, j+n]K[m,n]$$

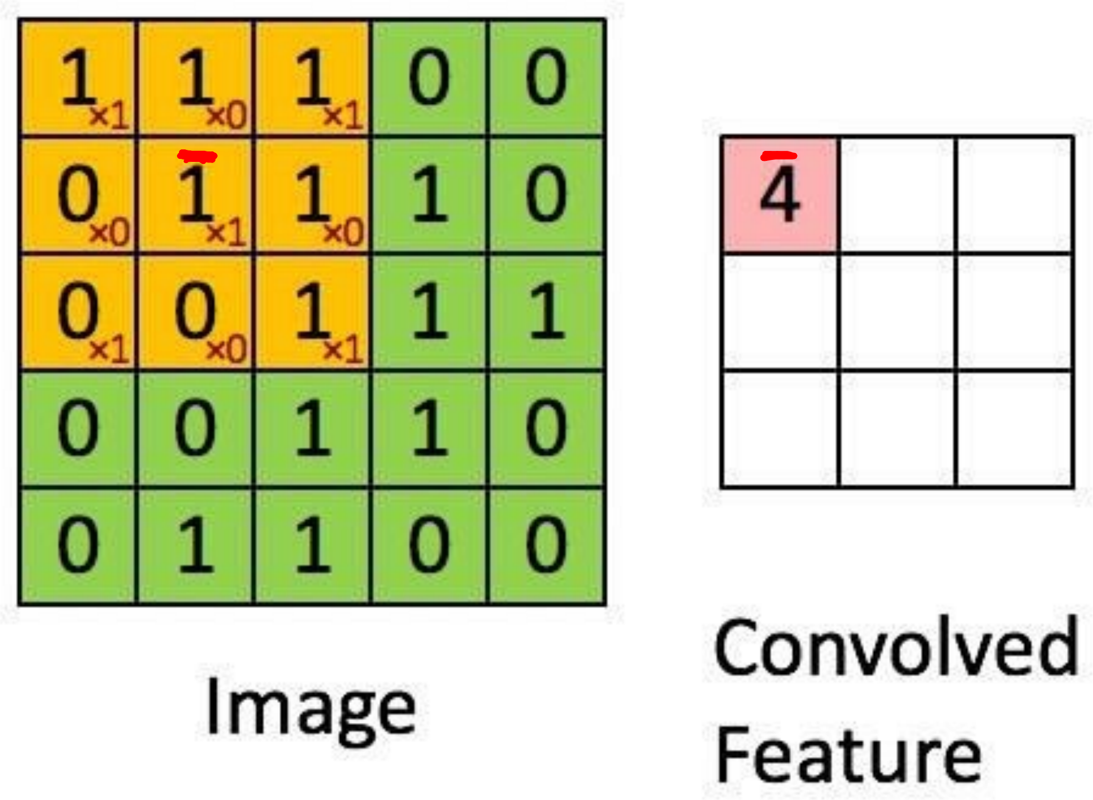
Many machine learning libraries implement cross-correlation but call it convolution.

https://www.youtube.com/watch?v=Ma0YONjMZLI

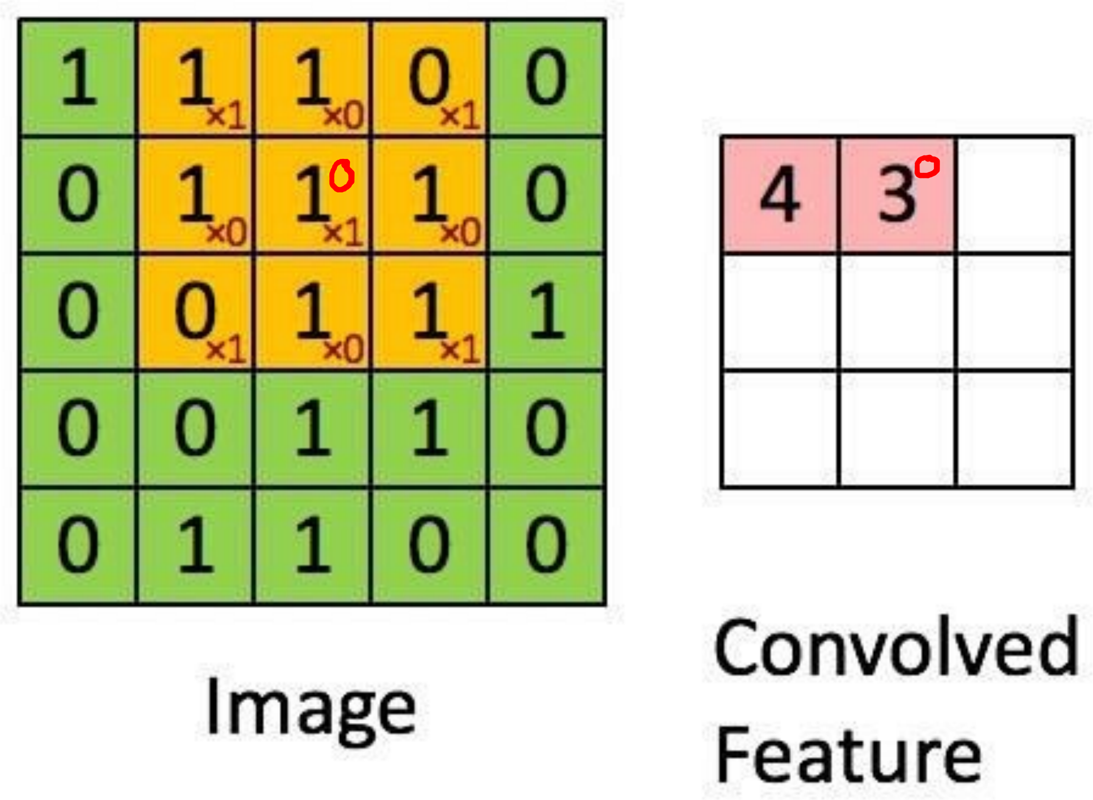
Fig 9.1

Discrete convolution can be viewed as multiplication by a matrix.

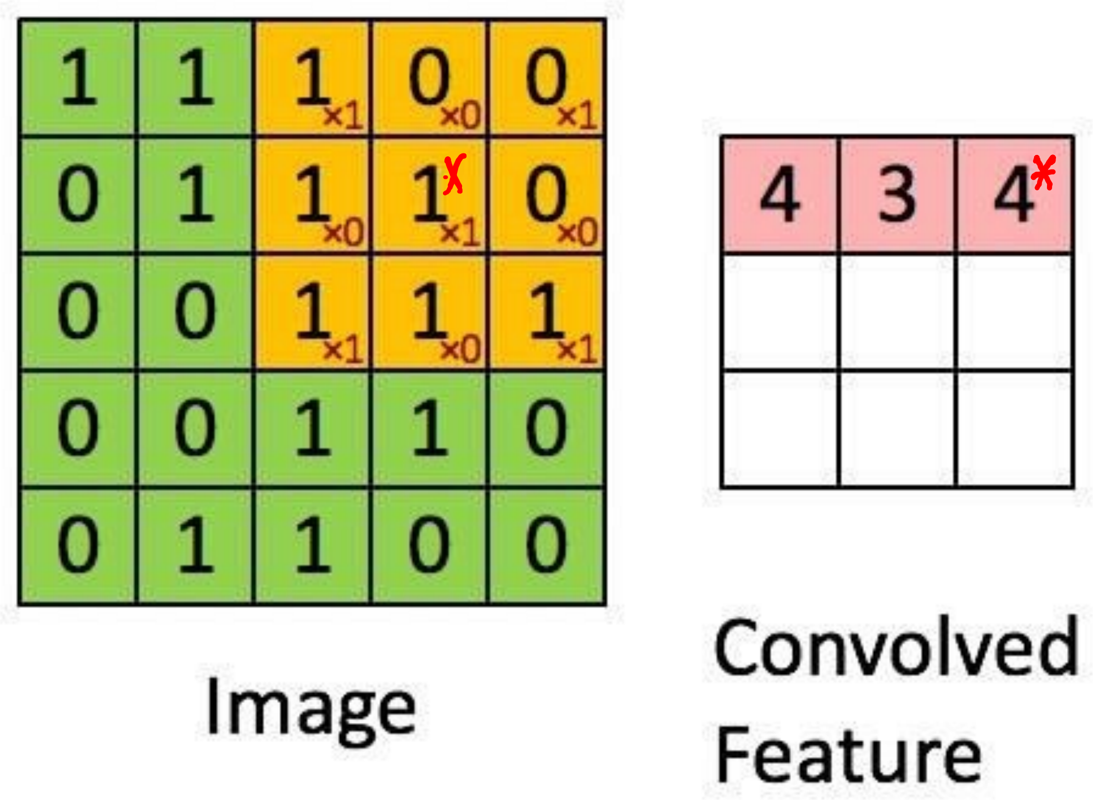
Convolutions



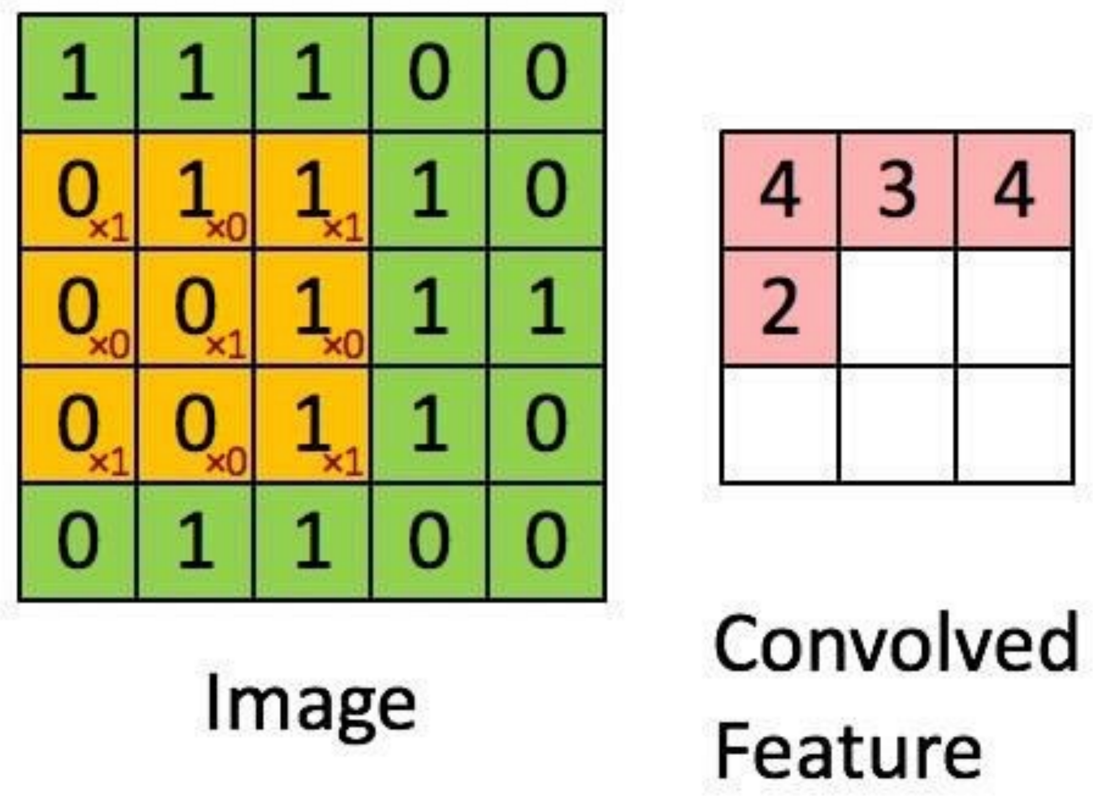
Convolutions



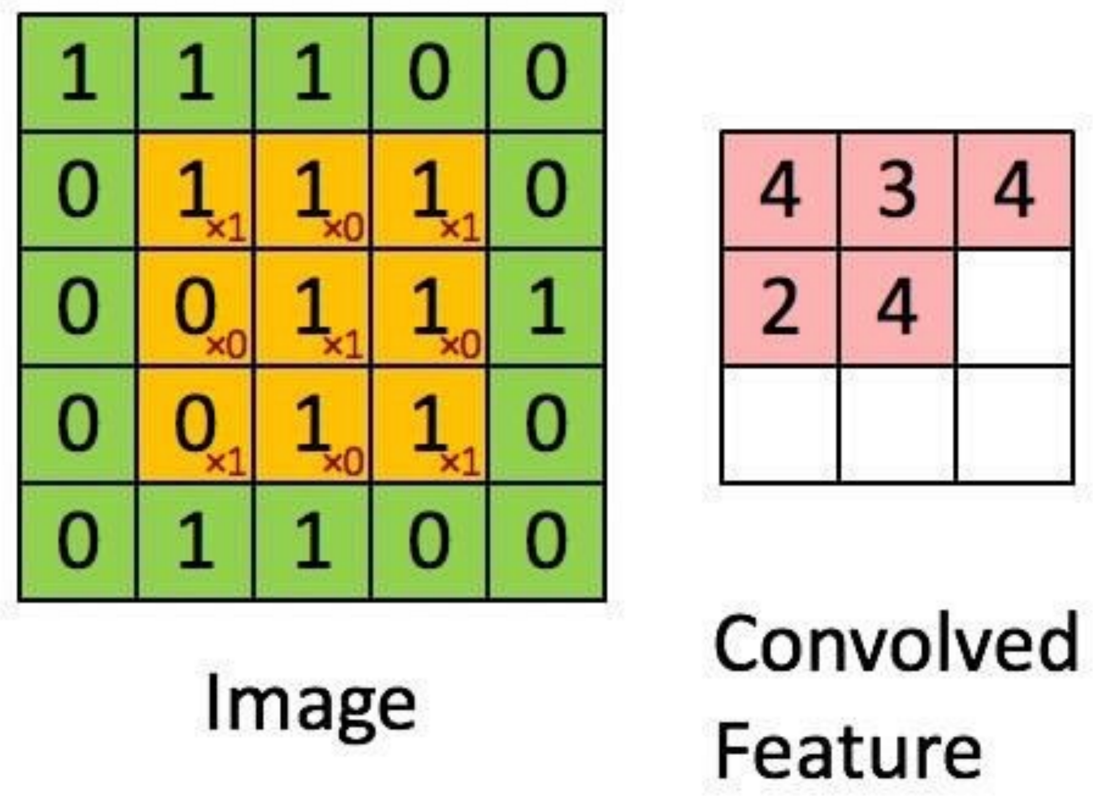
Convolutions



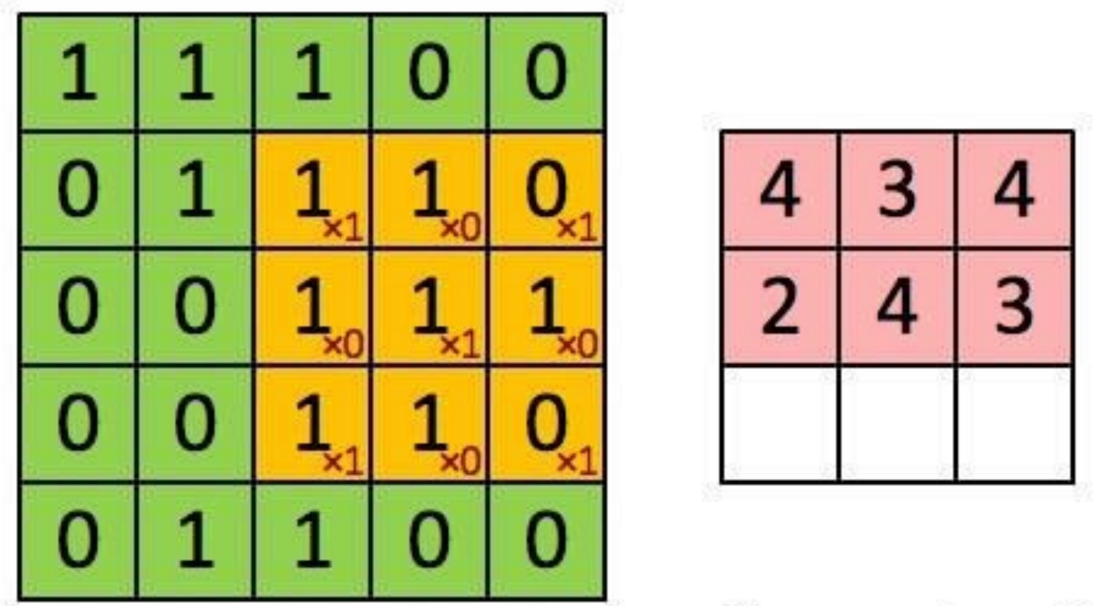
Convolutions



Convolutions



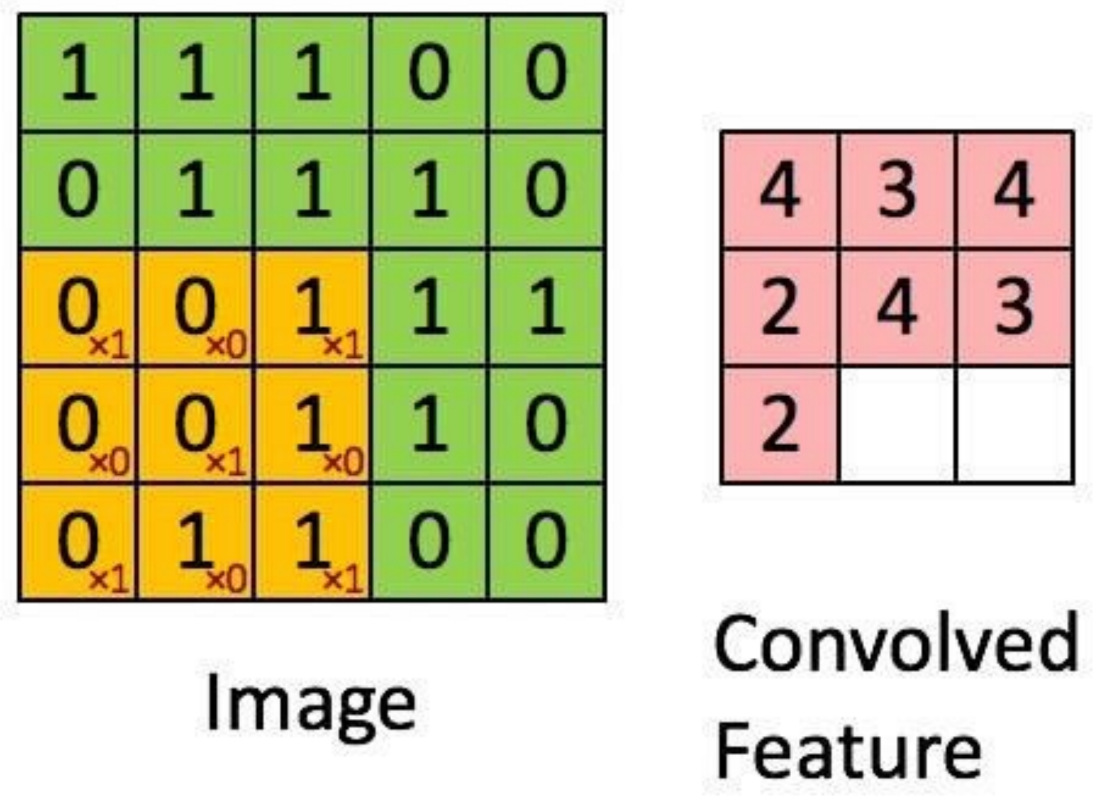
Convolutions



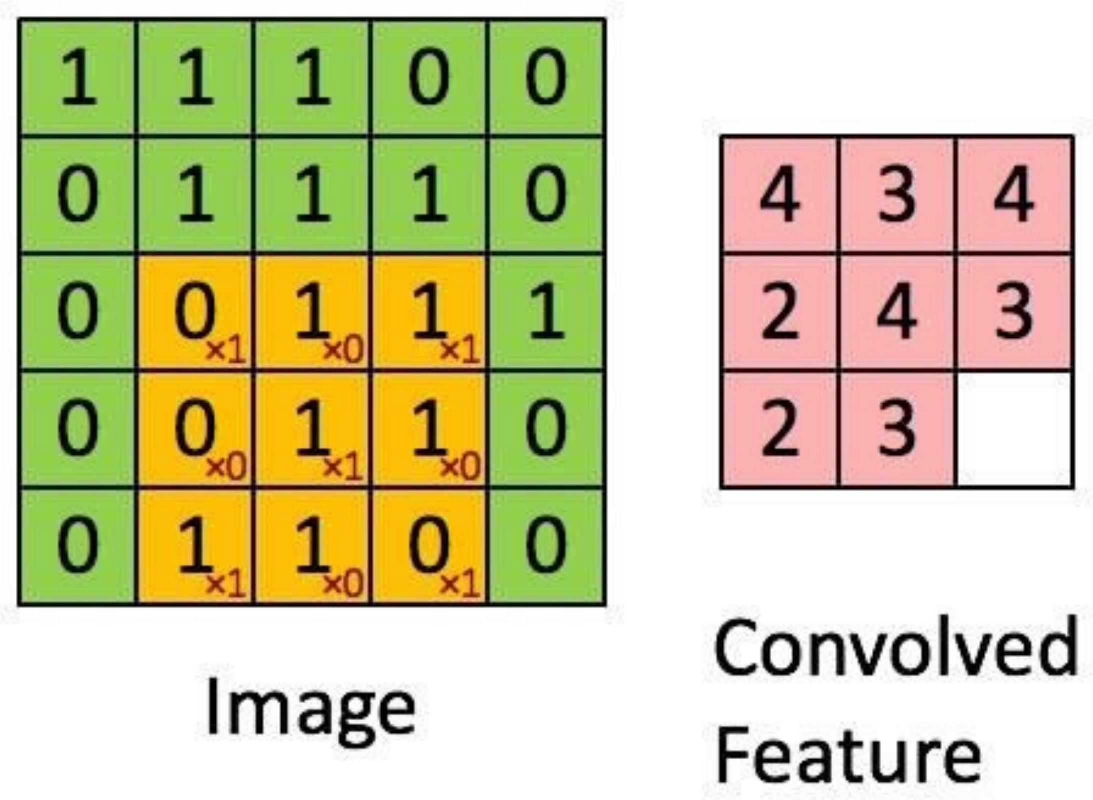
Image

Convolved Feature

Convolutions

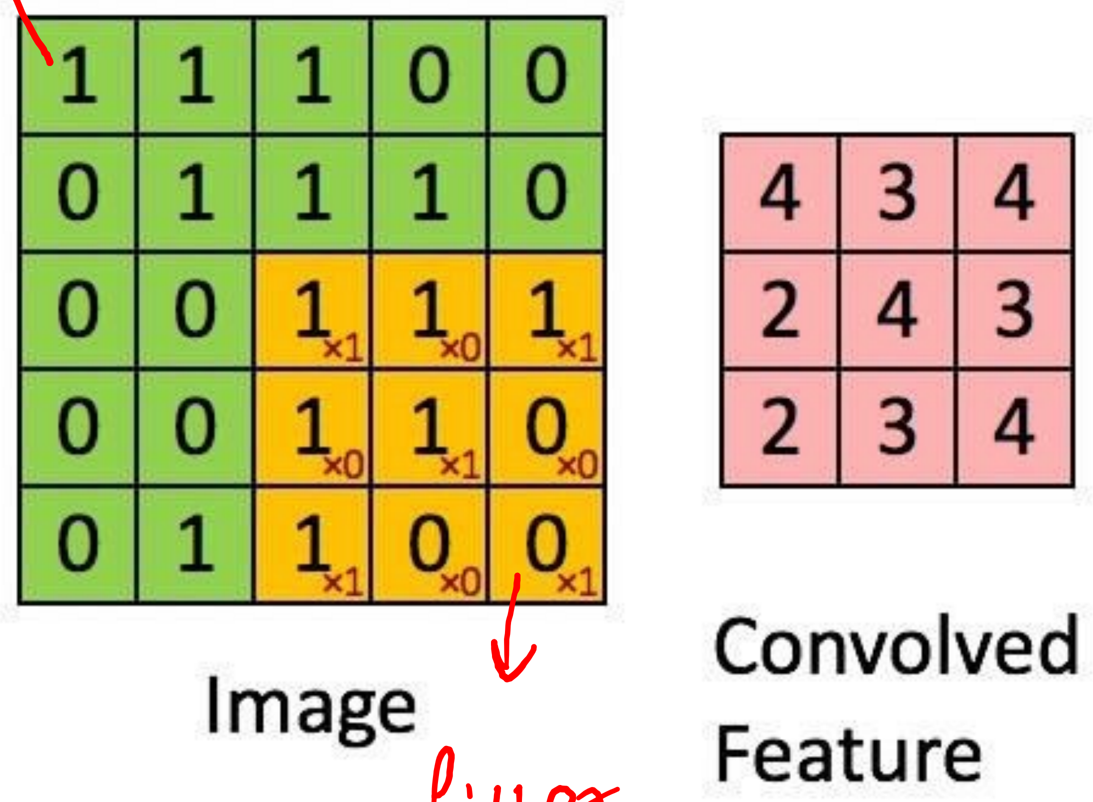


Convolutions



Convolutions

I/P



activation. σ

max
Pool.

filter

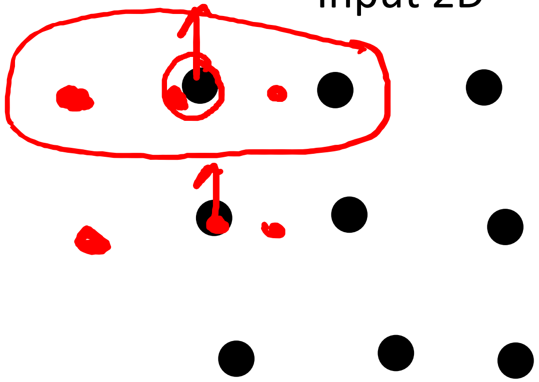
Sparse interactions

In feed forward neural network every output unit interacts with every input unit.

Convolutional networks, typically have sparse connectivity (sparse weights)

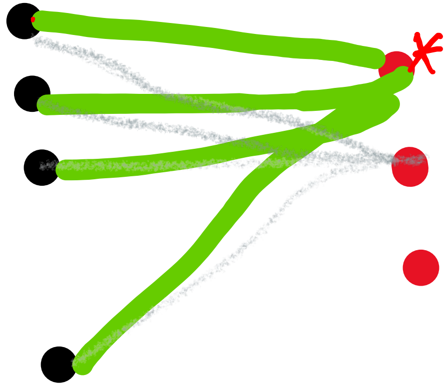
This is accomplished by making the kernel smaller than the input

Input 2D

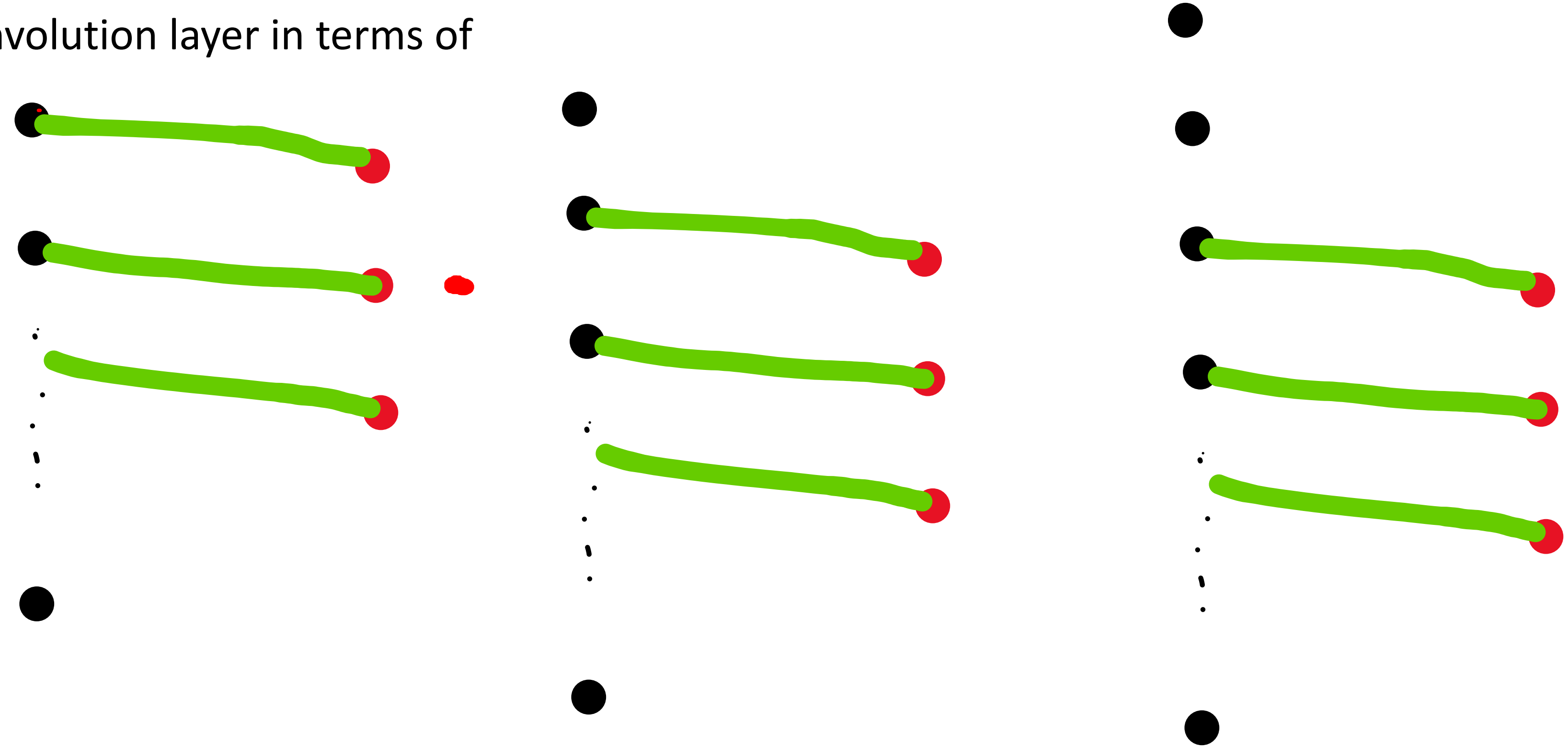


mask/kernel/filter/weights

Dense layer



Depiction of convolution layer in terms of flattened layers



Sparse interactions

When we have m inputs and n outputs, then matrix multiplication requires $m \times n$ parameters. and the algorithms used in practice have $O(m \times n)$ runtime (per example).

limit the number of connections each output may have to k, then requires only $k \times n$ parameters and $O(k \times n)$ runtime.

Parameter sharing

In a traditional neural net, each element of the weight matrix is multiplied by one element of the input. i.e. It is used once when computing the output of a layer.

In CNNs each member of the kernel is used at every position of the input

Instead of learning a separate set of parameters for every location, we learn only one set.

Convolutional Networks

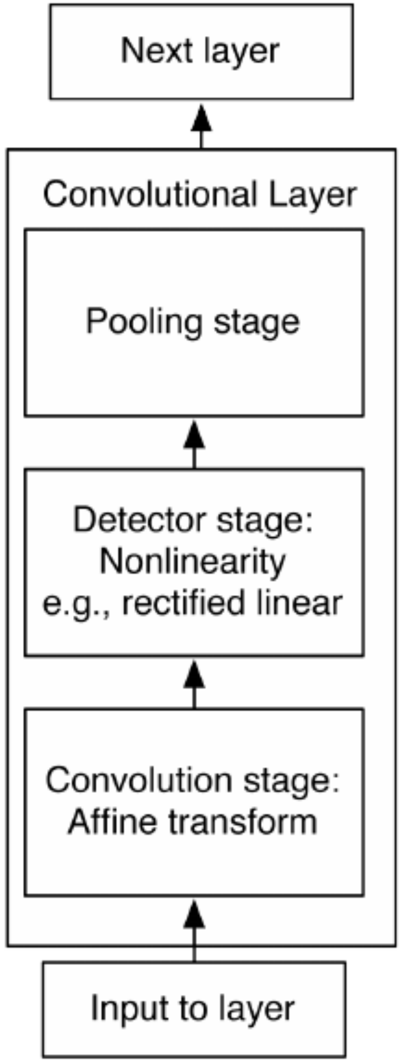
The first stage (Convolution):

The layer performs several convolutions in parallel to produce a set of preactivations.

The second stage (Detector):

Each preactivation is run through a nonlinear activation function (e.g. rectified linear).

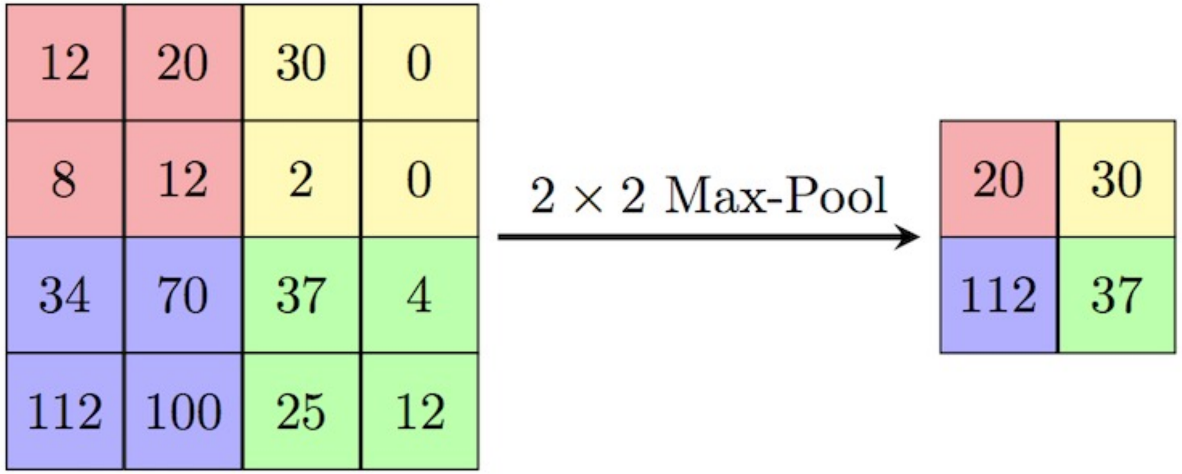
The third stage (Pooling)



Popular Pooling functions

The maximum of a rectangular neighborhood (Max pooling operation)

Feature maps/channels/Pre-activation, there could be several of them



2x2

20	20	30	30
20	20	30	30
112	112	37	37
112	112	37	37

The average of a rectangular neighborhood.

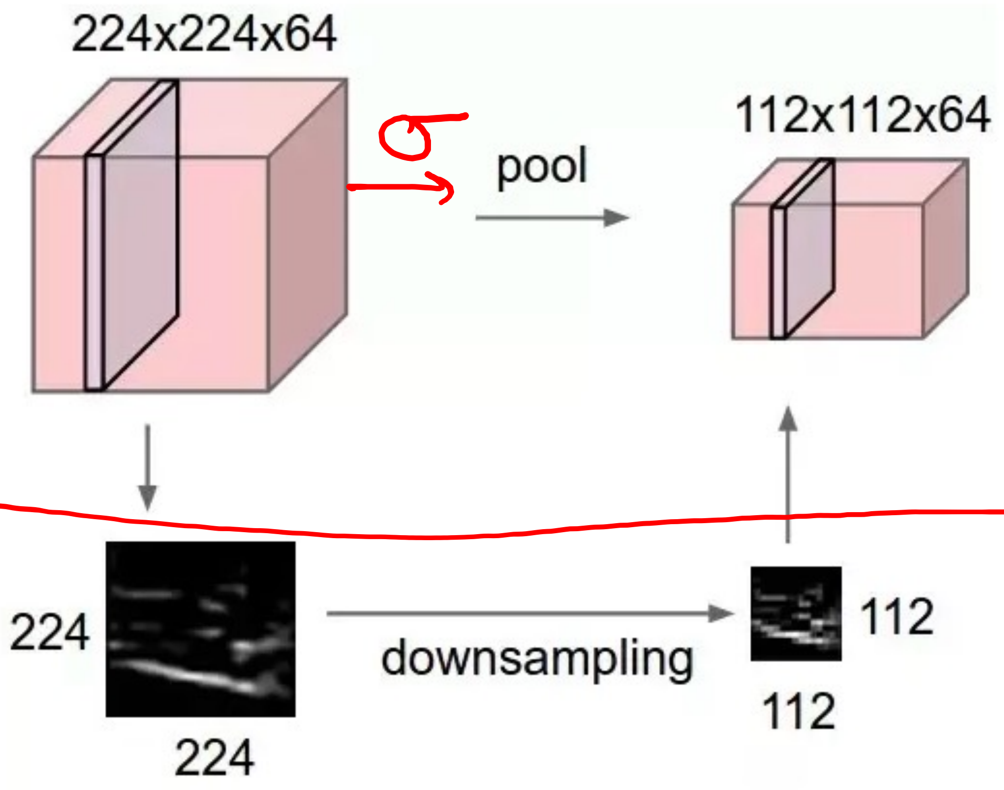
The L2 norm of a rectangular neighborhood.

Source: wiki

Pooling with downsampling

224x224
input
image

64 different
filters of
size mxn



σ

128

112x112
x128 σ
Pool.

64x64x128