Module 11
Deep Learning

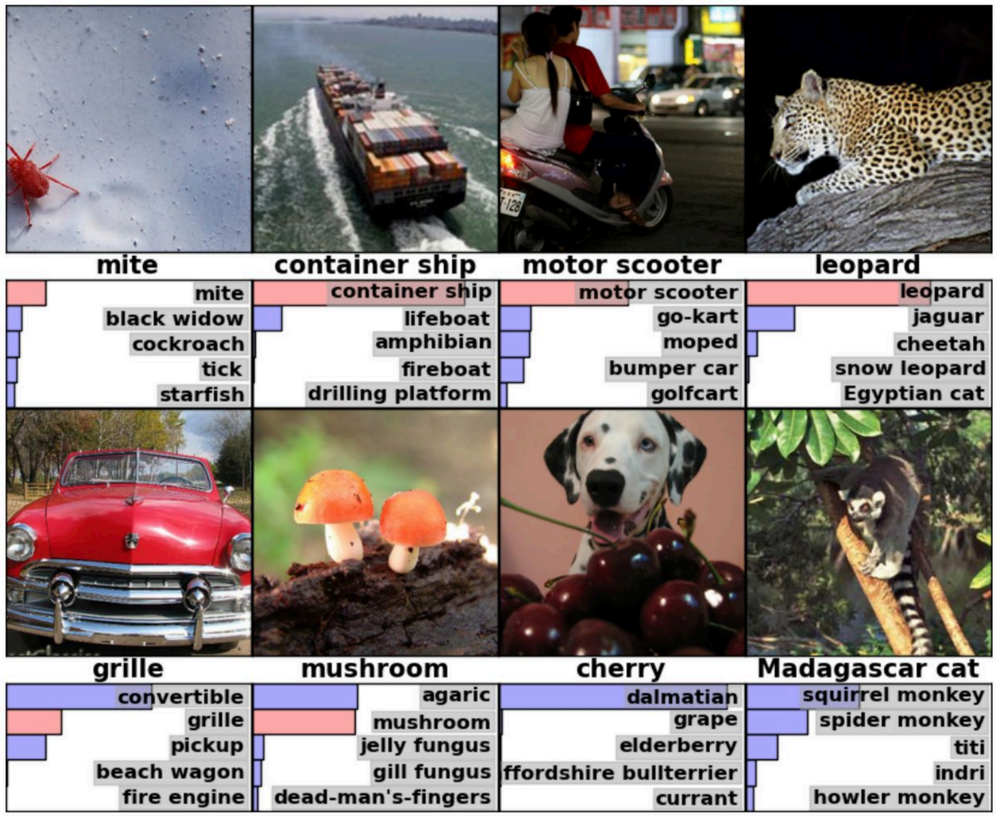
Why?

Why?

ImageNet Challenge

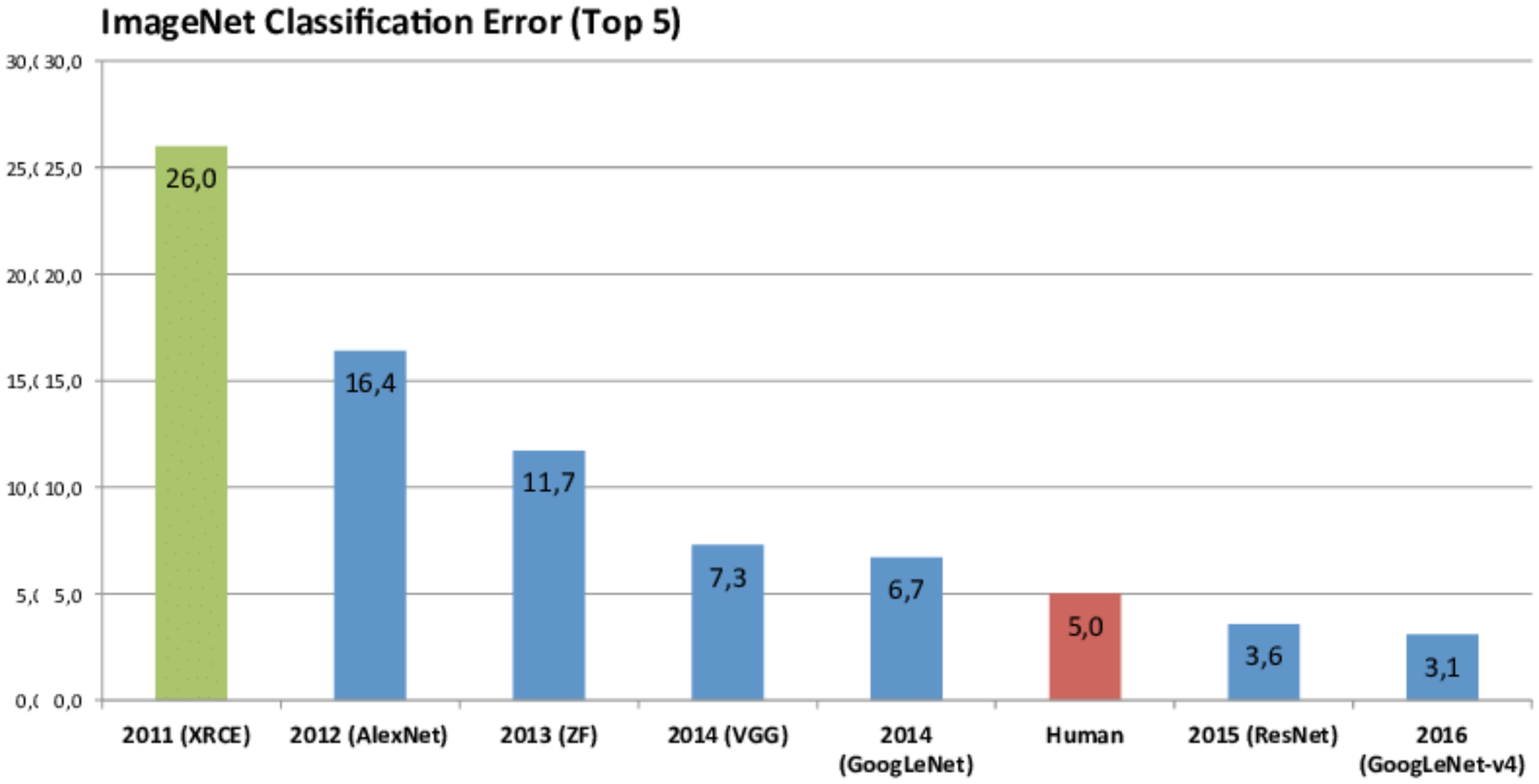
IMAGENET

- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.



(Source: Xavier Giro-o-Nieto)

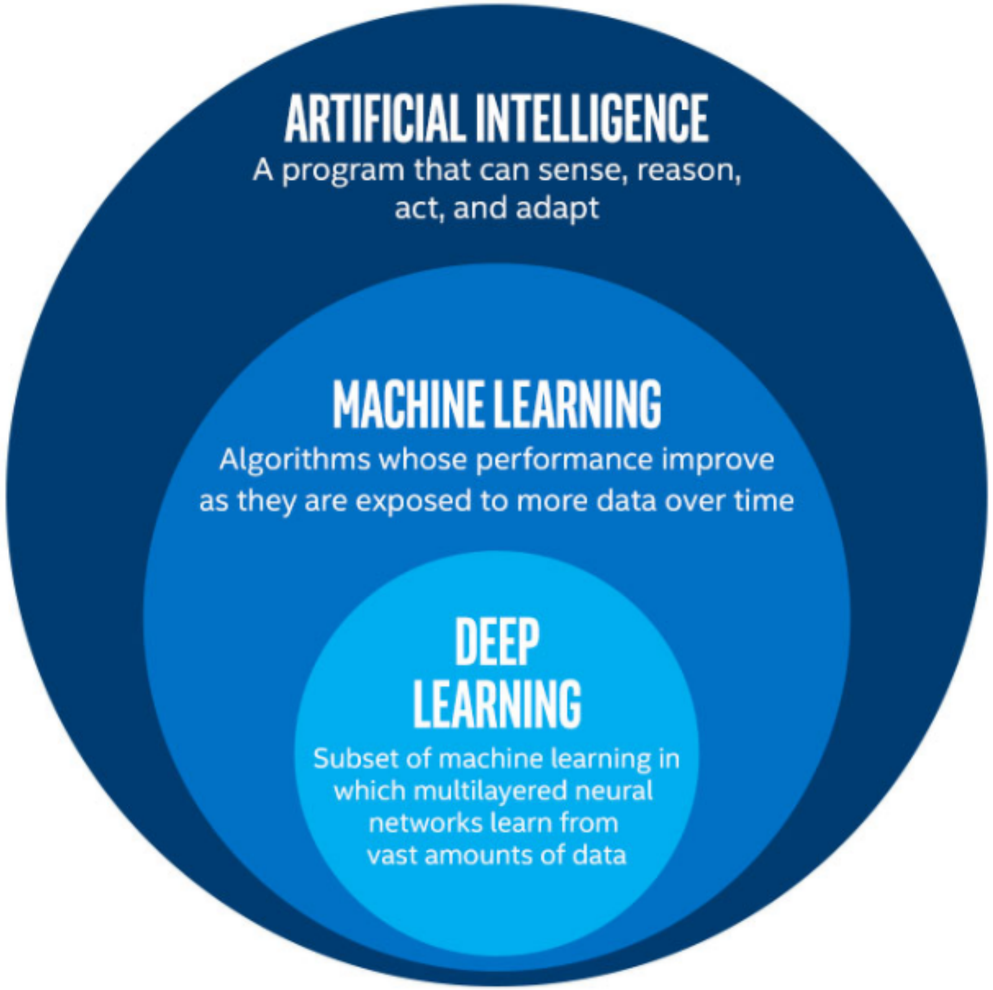
Why?



https://www.researchgate

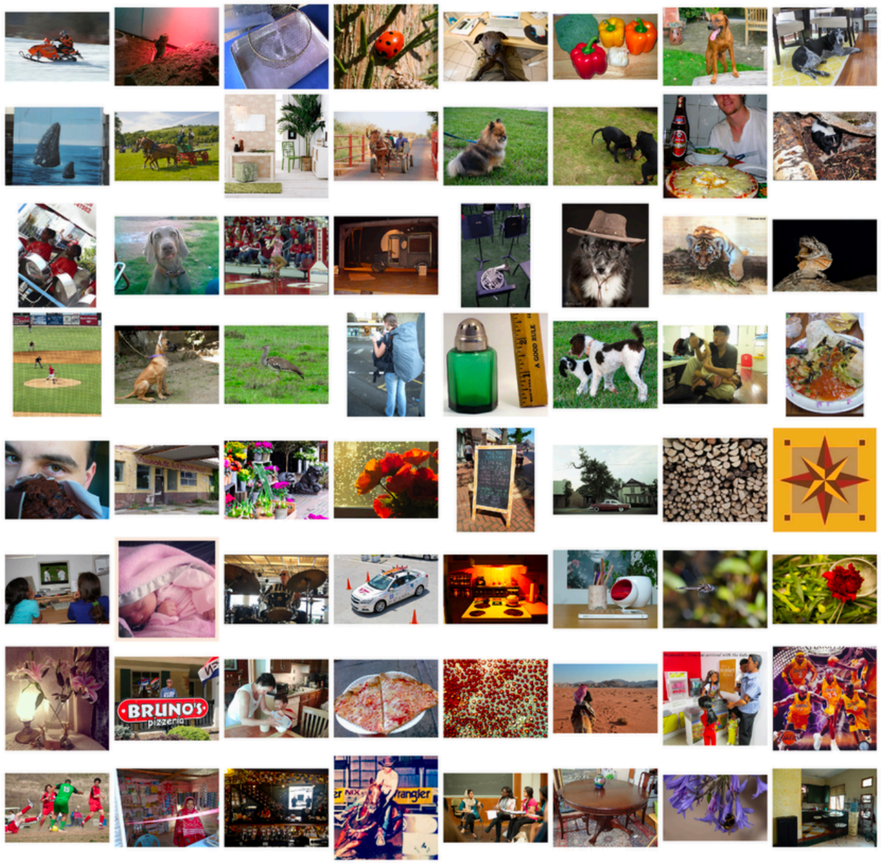
What is Deep Learning?

What is Deep Learning?



What is Deep Learning?

ImageNet



Traditional Computer Vision

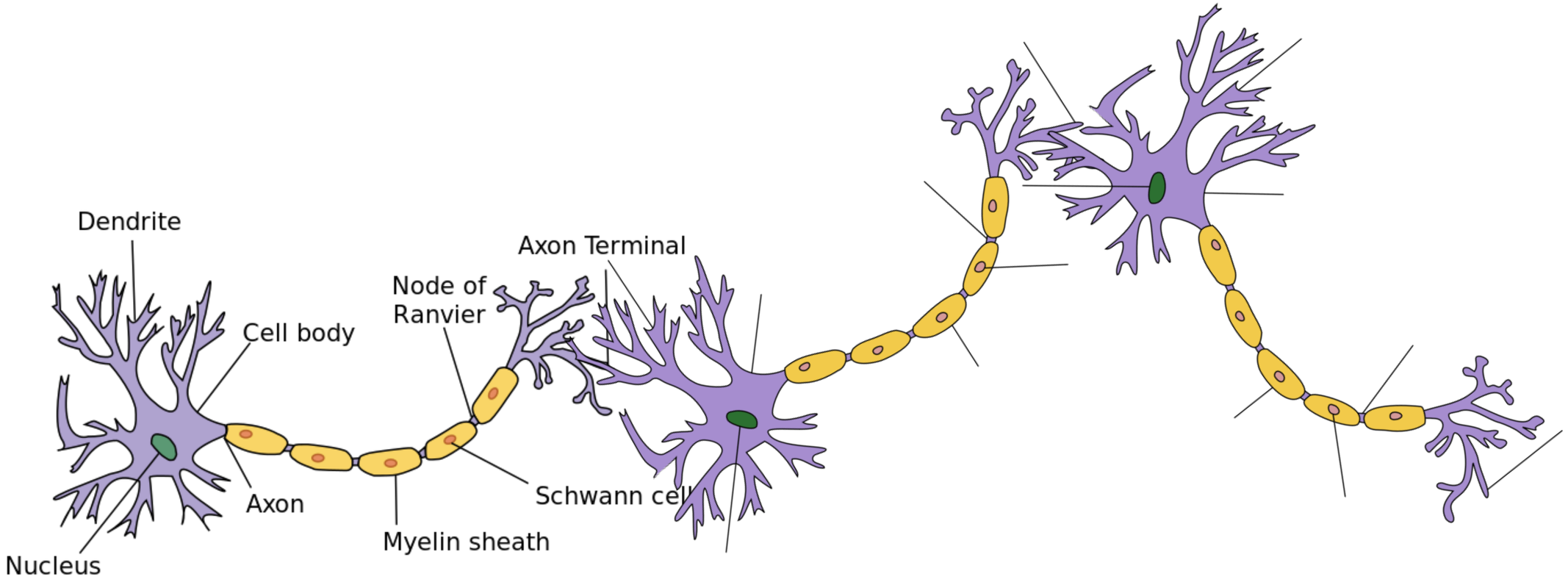
Hand-crafted Feature Extractors → Trainable Classifier

Deep Learning

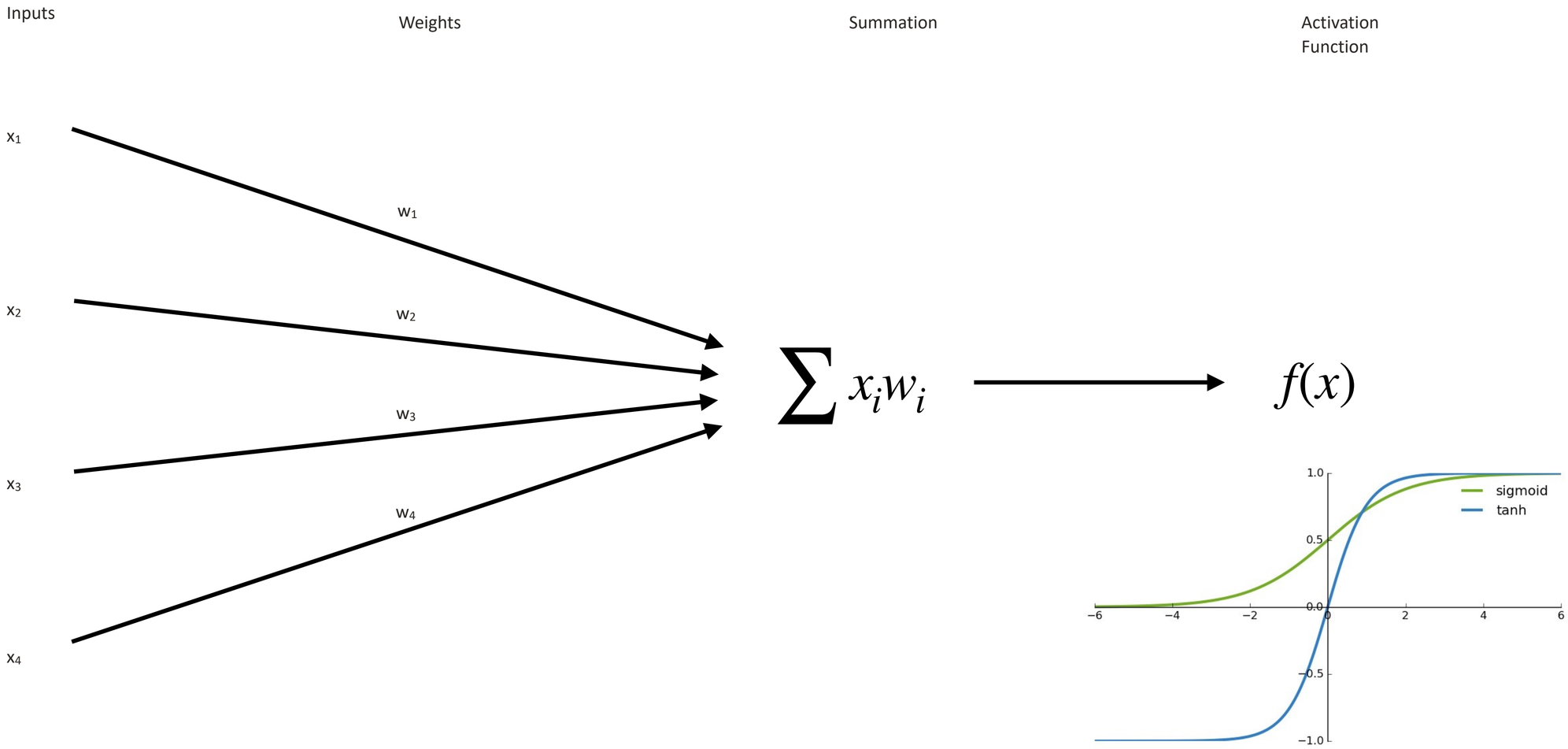
Low-Level Features → Mid-Level Features → High-Level Features → Trainable Classifier

Neural Networks

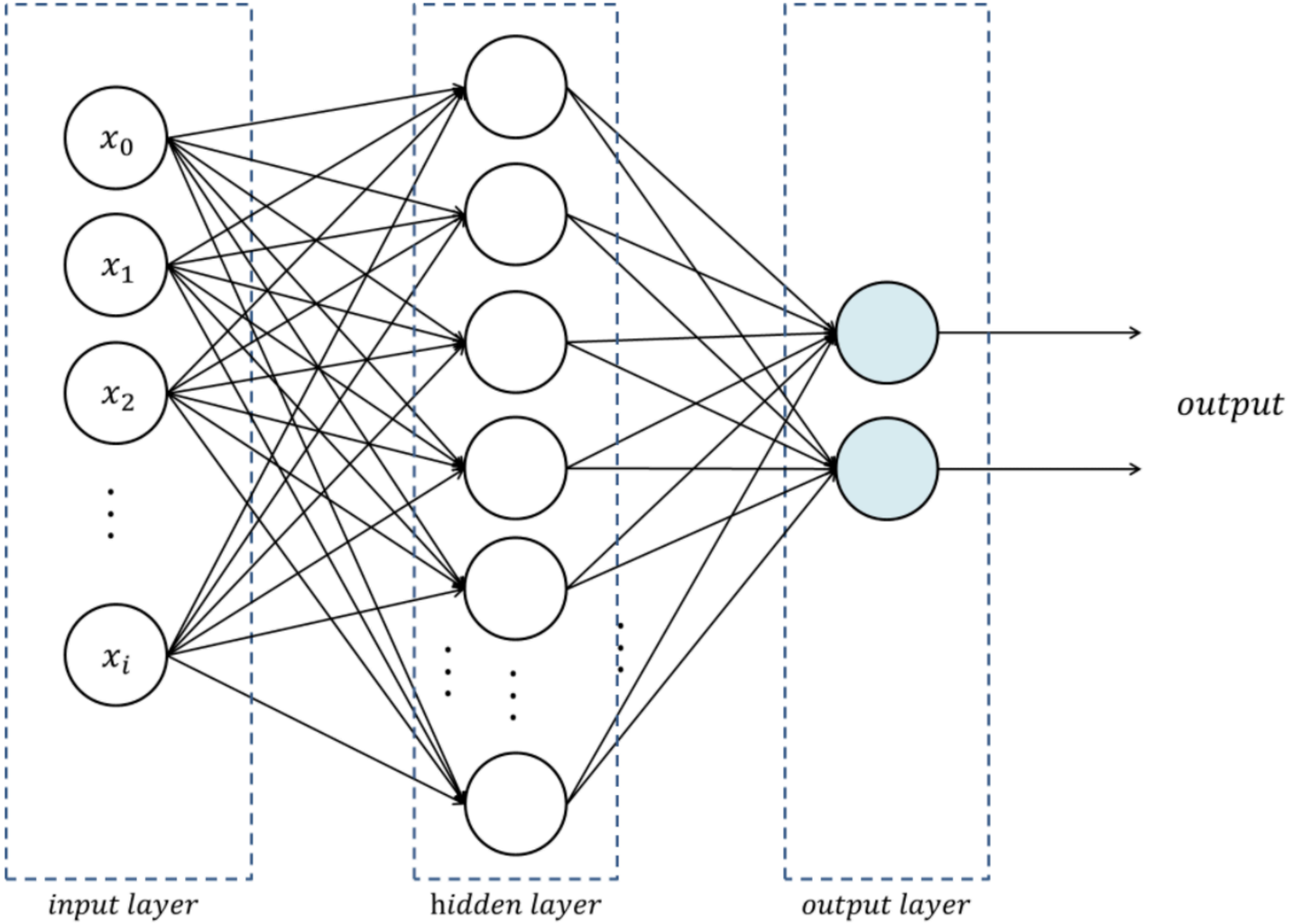
Neural Networks



Neural Networks



Multi-layer Perceptron



Neural Network Demo

https://playground.tensorflow.org/

Deep Neural Networks

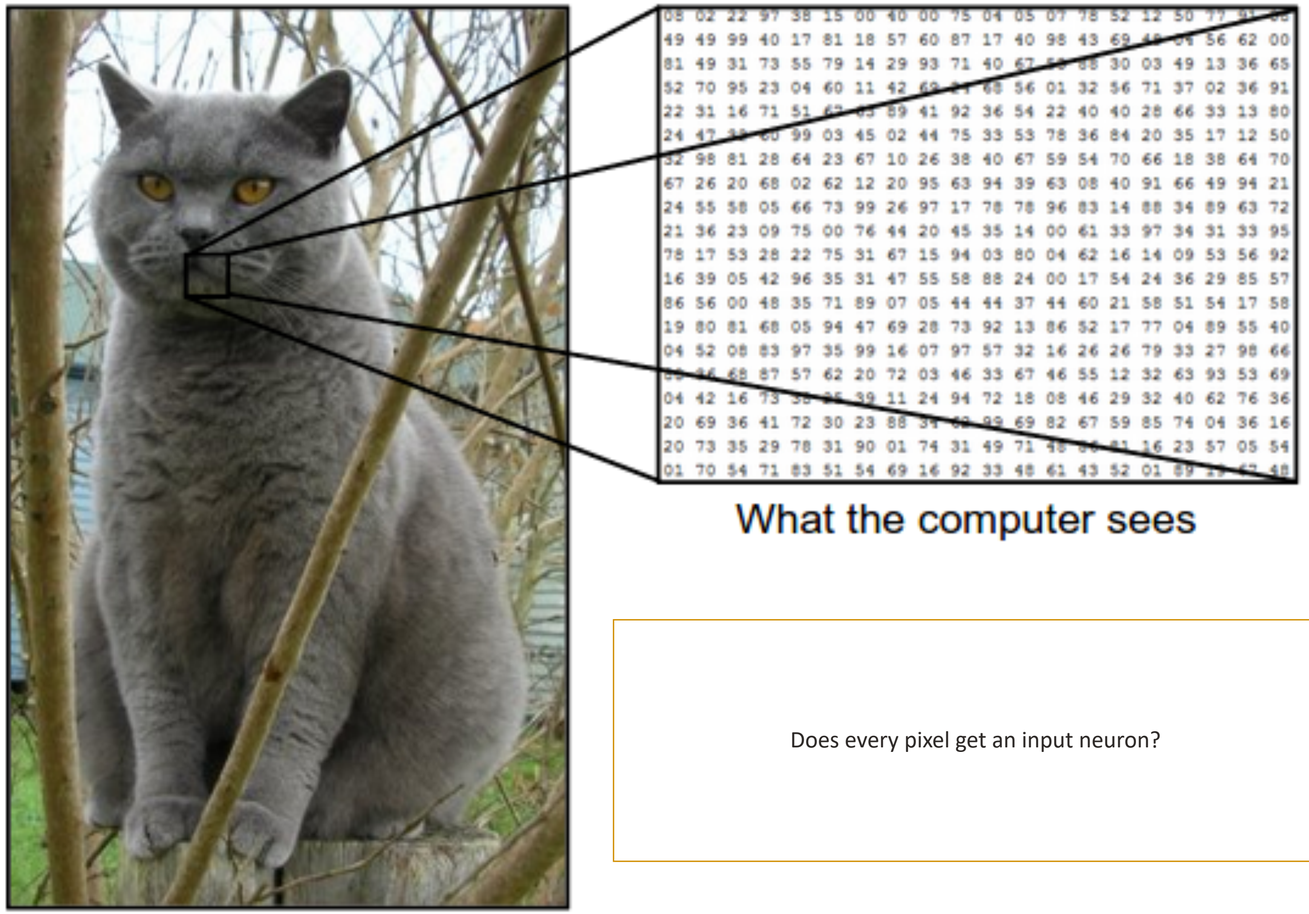
Two types

- Convolutional
- Recurrent

Convolutional Neural Networks

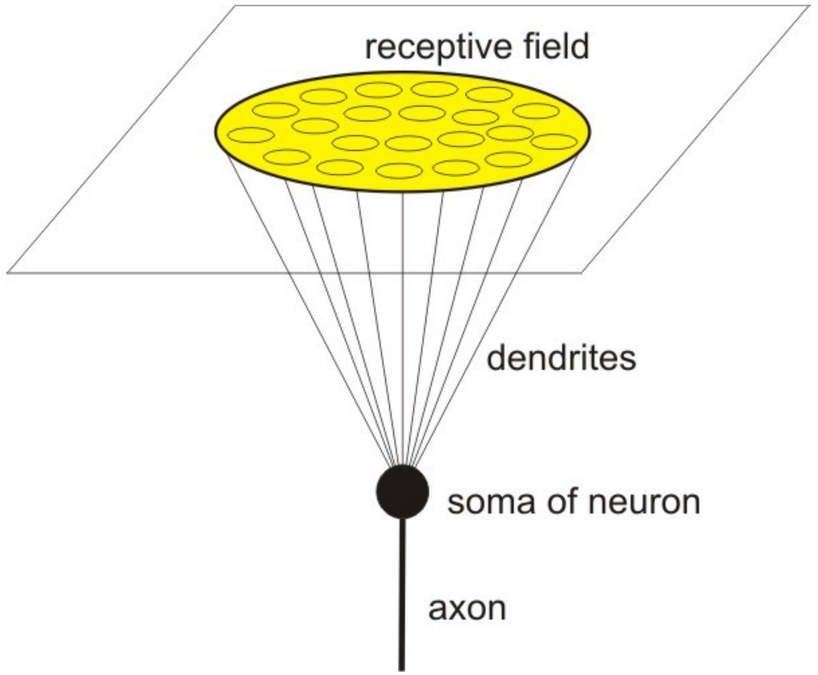
The Real Deep Neural Networks

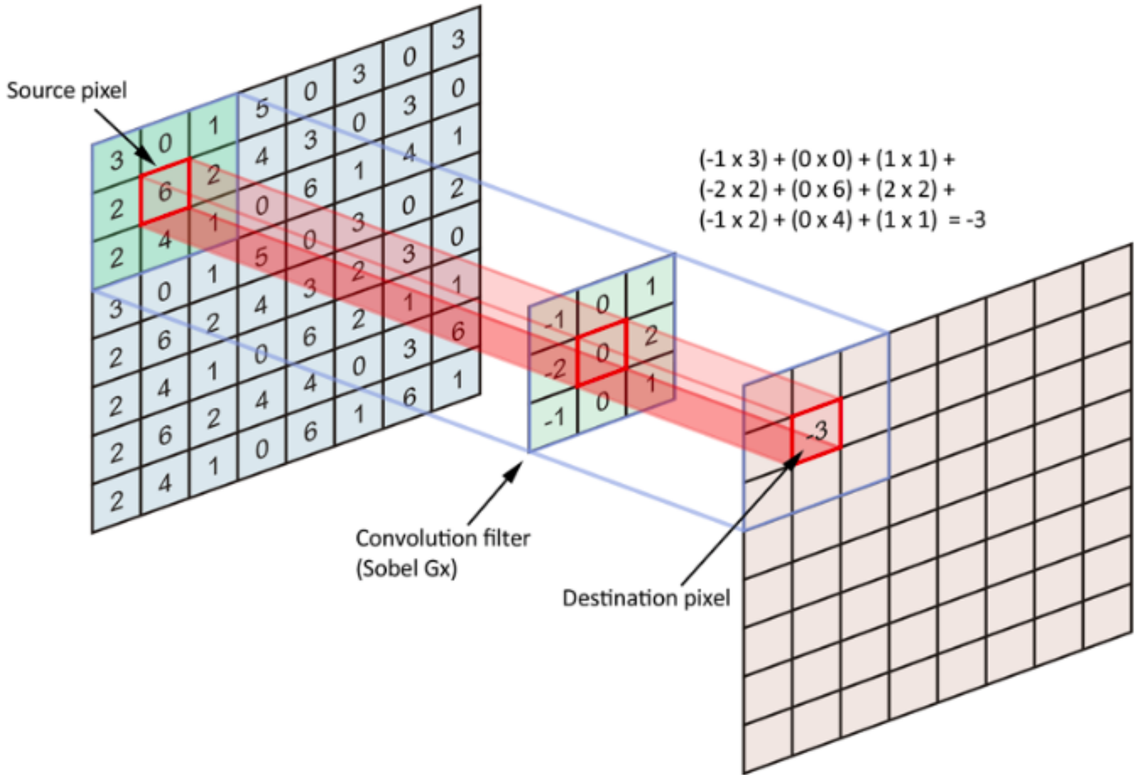
Convolution



Does every pixel get an input neuron?

Convolution

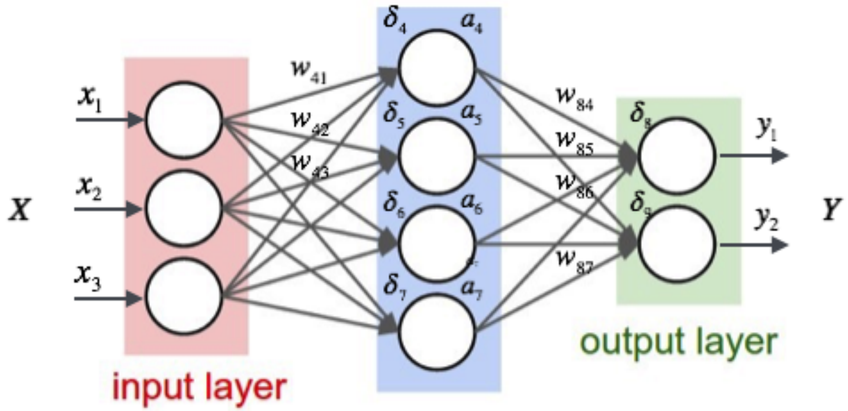


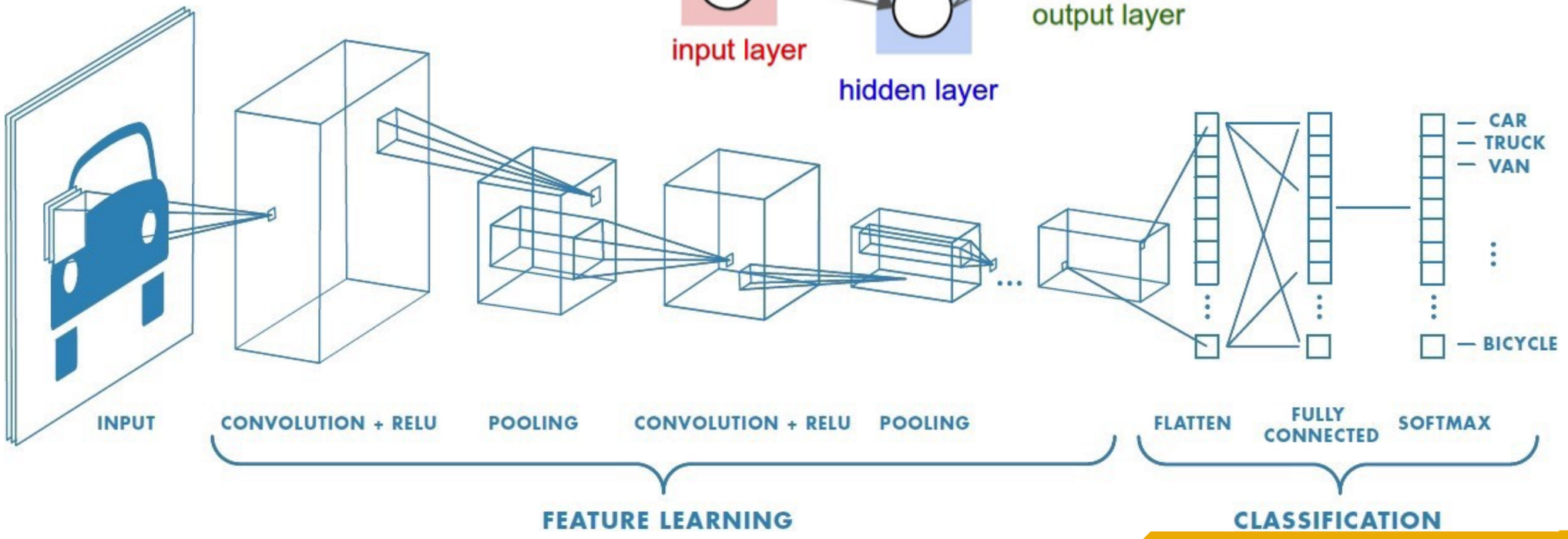


http://neuroclusterbrain.com/neuron_model.html

https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

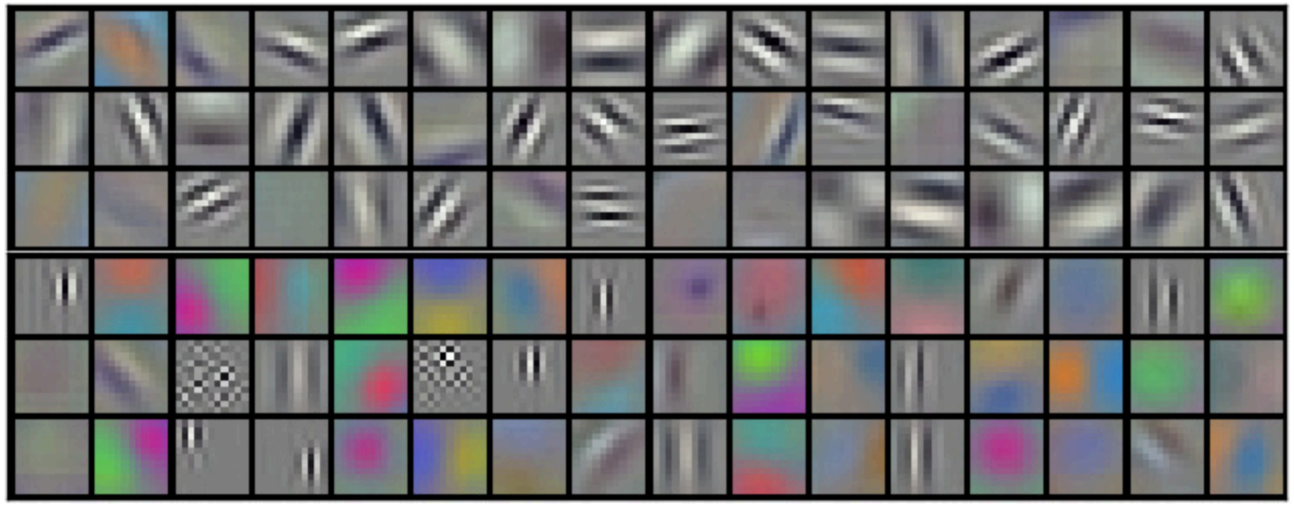
Convolution





https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Convolution - Learned Filters



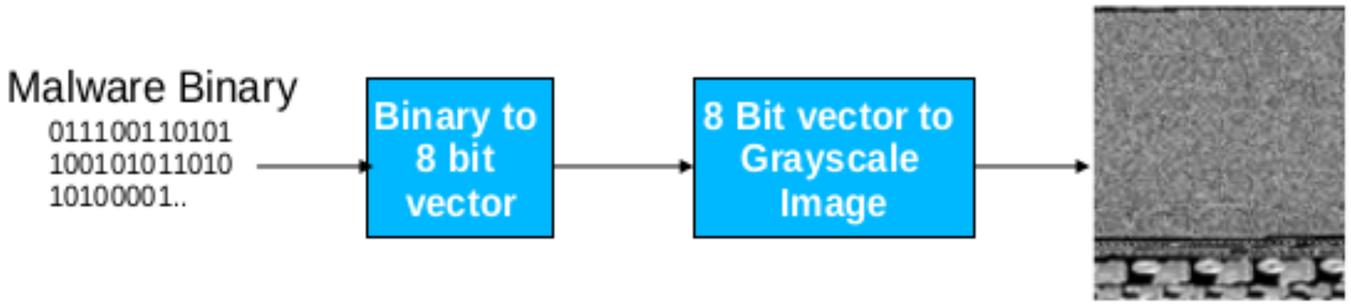
Cool CNN Applications

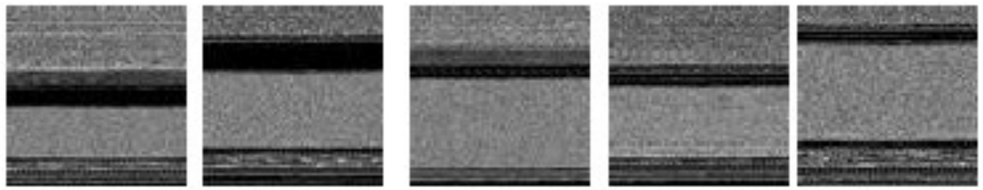
- Revolutionized any field having to do with images
  - Object detection/segmentation
  - Video analysis
  - Robotics
  - Self-driving cars
- Hand-writing recognition
- Heathcare
  - Breast and skin-cancer diagnosis

Convolution in Cybersecurity

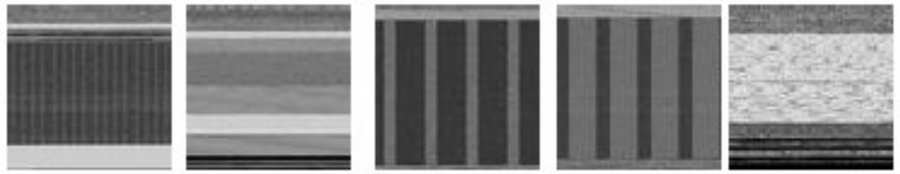
Gibert, Daniel. "Convolutional neural networks for malware classification."

***University Rovira i Virgili, Tarragona, Spain* (2016).**





Rammit Samples

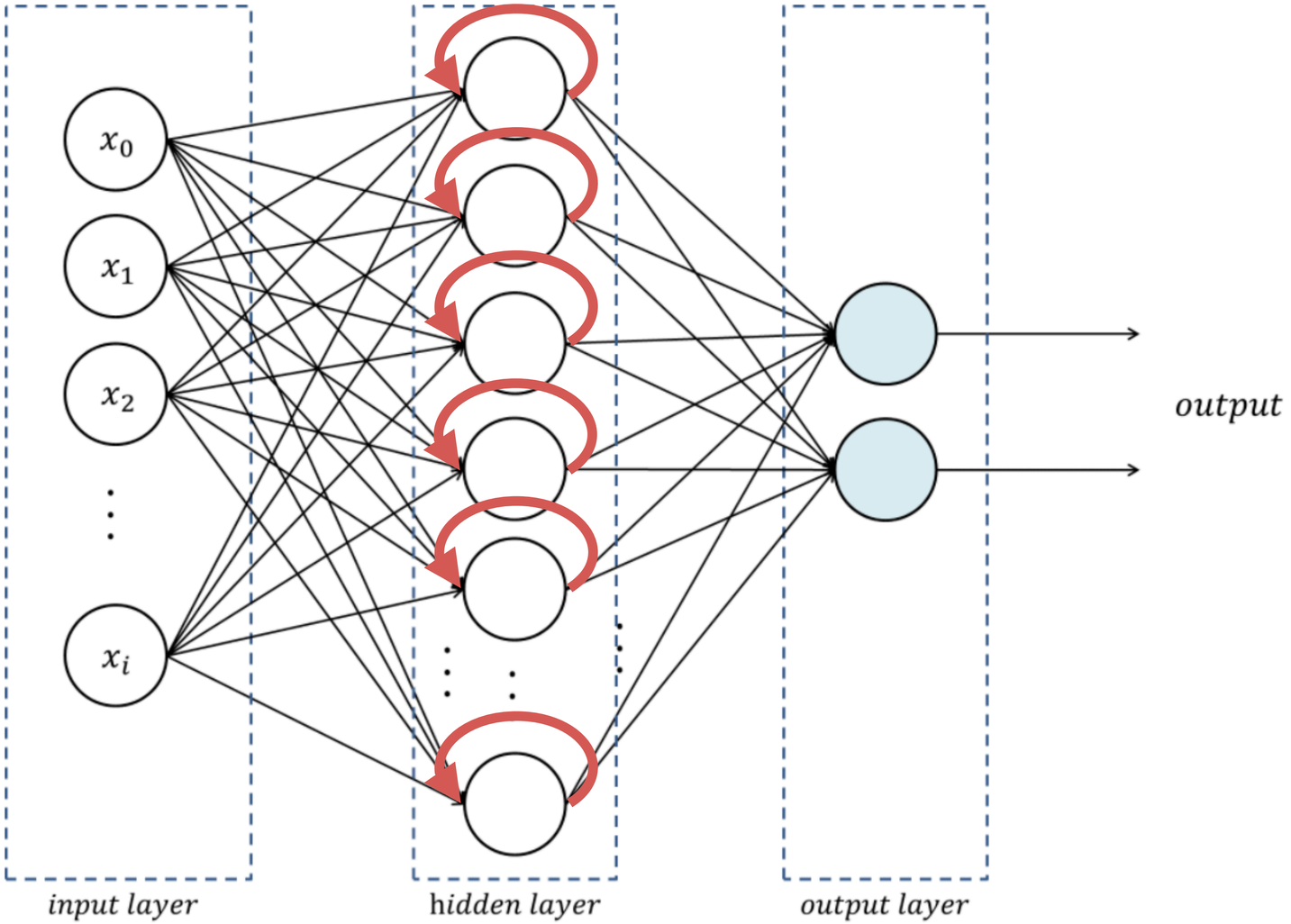


Lollipop Samples

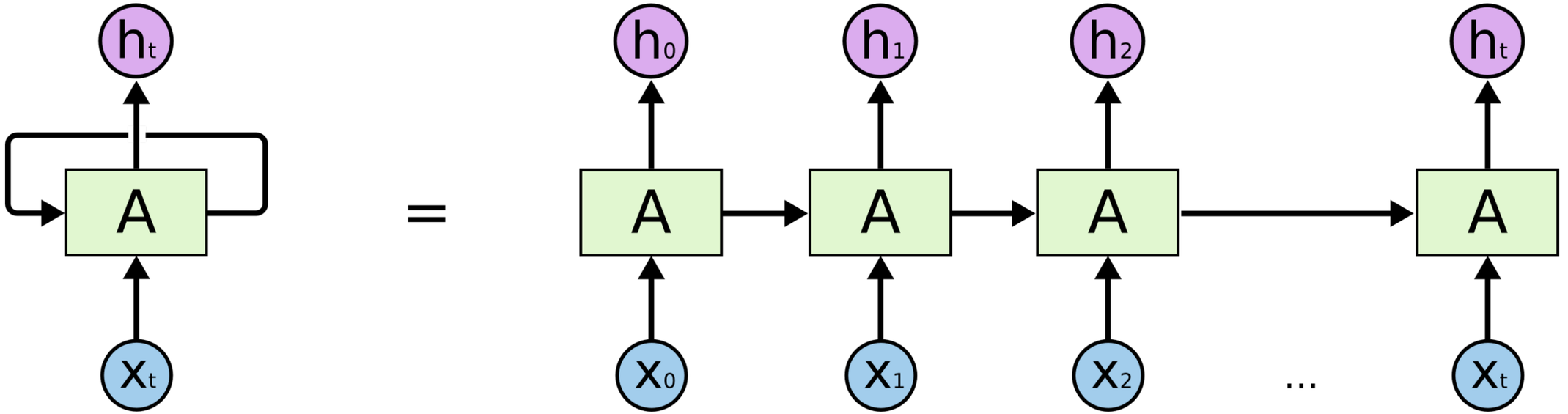
Recurrent Neural Networks

Time-series Neural Networks

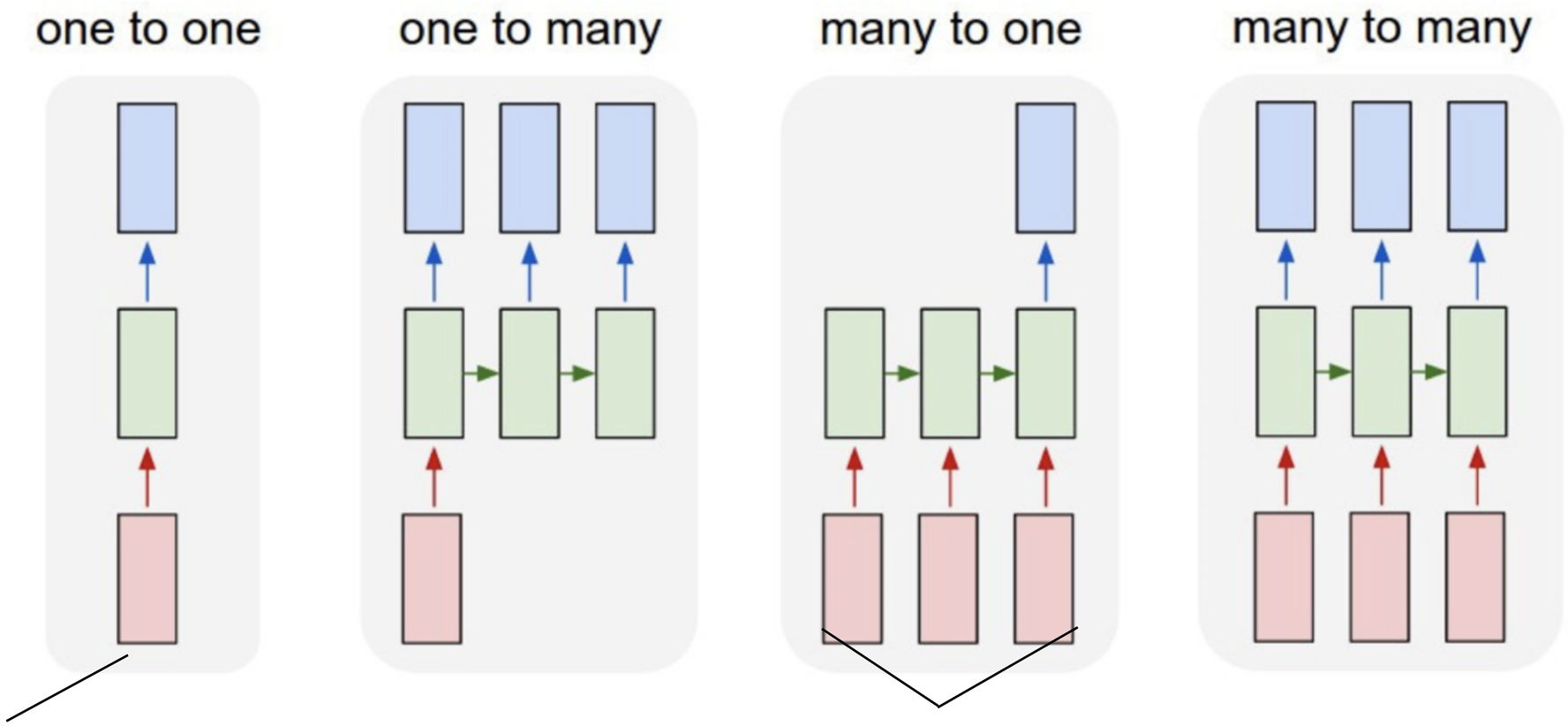
Recurrence



Recurrence - unwrapped



Recurrent Neural Networks (RNNs)



- 1 data observation
- Can be multidimensional

- Time series or sequence of multiple data observations
- Each can be multidimensional

RNN Variants

- Vanilla RNN ✗
- Long Short-term Memory (LSTM) ✓
- Gated Recurrent Unit (GRU) ✓

Cool RNN Applications

- Natural Language Processing (NLP)
  - Machine Translation (Google’s seq2seq)
- Image Captioning
- Webpage Classification
- Domain Generation Algorithm (DGA) Classification

RNNs in Cybersecurity

Torres, Pablo, et al. "An analysis of Recurrent Neural Networks for Botnet detection behavior." *2016 IEEE biennial congress of Argentina (ARGENCON)*. IEEE, 2016.

TABLE I
SYMBOL ASSIGNMENT STRATEGY FOR BUILDING BEHAVIORAL MODEL ACCORDING TO THE STRATOSPHERE PROJECT.

	Size Small			Size Medium			Size Large		
	Dur. Short	Dur. Med	Dur. Long	Dur. Short	Dur. Med	Dur. Long	Dur. Short	Dur. Med	Dur. Long
Strong Per	a	b	c	d	e	f	g	h	i
Weak Per.	A	B	C	D	E	F	G	H	I
Weak Non-Per.	r	s	t	u	v	w	x	y	z
Strong Non-Per	R	S	T	U	V	W	X	Y	Z
No Data	1	2	3	4	5	6	7	8	9

Symbols for time difference	
Between 0 and 5 seconds:	.
Between 5 and 60 seconds:	,
Between 60 and 5 mins:	+
Between 5 mins and 1 hour	*
Timeout of 1 hour:	0

2.4.R*R.R.R*a*b*a*a*b*b*a*R.R*R.R*a*a*b*a*a*a*a*

Fig. 1. An example of the behavioral model of connection from IP address 10.0.2.103 to destination port 53 at IP address 8.8.8.8 port 53 using protocol UDP.

Deep Learning Tools

- Tensorflow (Google)
  - The most math-heavy but most versatile
- PyTorch (Facebook)
- Caffe (UC Berkeley)
- Keras
  - Abstraction to Tensorflow to make it easier
  - Probably use this

Deep Learning Takeaways

- Deep learning isn’t magic
  - The size of our datasets is what’s really magic
- Certain problems and problem areas are perfect for deep learning
  - Others are not
- Deep learning is hard to implement
  - But the tools are getting easier every day
- RNNs probably have the most application in Cybersecurity

Use Case: Featureless Deep Learning

Deep Learning for Everyone!

+

+

Nice to have!
Lots of GPUs!!!

Backend C - Frontend Python
(Released 2015 by Google)

Keras - Python
(High-level wrapper API)

Alternative backend (theano, torch) and high-level APIs (Lasagne, Caffe, Pylearn2, Chainer)

Terminology

Deep Learning is performed in an iterative fashion! This means only part of all training data is used at a time.

- **Batch Size:** number of training examples in one forward/backward pass aka how many URLs are we training on here at a time (limited by RAM).
- **Iterations:** number of passes, where for each pass “batch size” number of training examples are used.
- **Epoch**: one forward and backward pass of “all” training data.

Example: For 1000 training examples (here URLs) and batch size of 500 it will take 2 iterations to complete 1 epoch. For Keras you have to provide batch size and number of epochs!!! Batch size of 32 is commonly used!

Keras Intro

High-level API for building neural networks:

- As simple as chaining layers together! Define your architecture (provide input, output, fully connected hidden layers, convolutional, recurrent or other layer types if applicable)!
- Define your optimizer method!
- Define suitable loss function and compile model
- Fit model on training set
- Evaluate model on test set

Keras Input and Output Layers

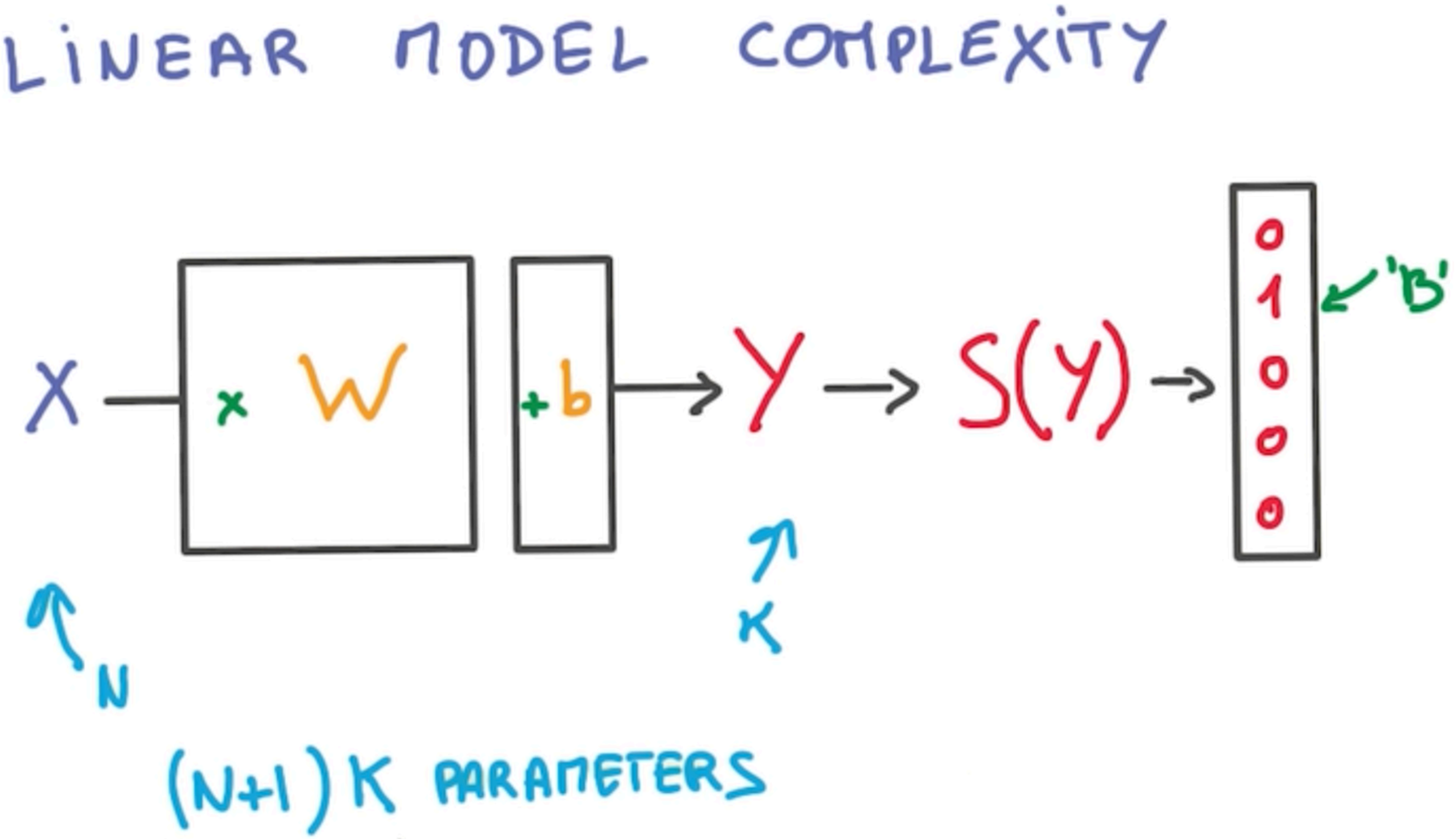
```
# Input
max_len = 75

# we will explain in more detail later, for now accept URL will be
# cropped at 75 characters or padded with zeros in case the URL is
# shorter, therefore input size is constant!)
main_input = Input(shape=(max_len,), dtype='int32', name='main_input')
first_layer = <LayerType>(<parameters>)(main_input)
<next_layer> = <LayerType>(<parameters>)(first_layer)
<penultimate_layer> = <LayerType>(<parameters>)(<next_layer>)

output = Dense(1, activation='sigmoid',name='output')(<penultimate_layer>) #
Note for binary cases output layer is 1, however, for multi-class cases
# it's of size n_classes!!!
```

Keras Compile Model, Fit & Evaluate

```
# Define model
model = Model(input=[main_input], output=[output])

# Define optimizer
adam = Adam(lr=1e-4, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)

# Compile mode
model.compile(optimizer=adam, loss='binary_crossentropy',
metrics=['accuracy'])

# Fit model and Cross-Validation
model.fit(X_train, target_train, nb_epoch=nb_epoch,
batch_size=batch_size) loss, accuracy = model.evaluate(X_test,
target_test, verbose=1)
```

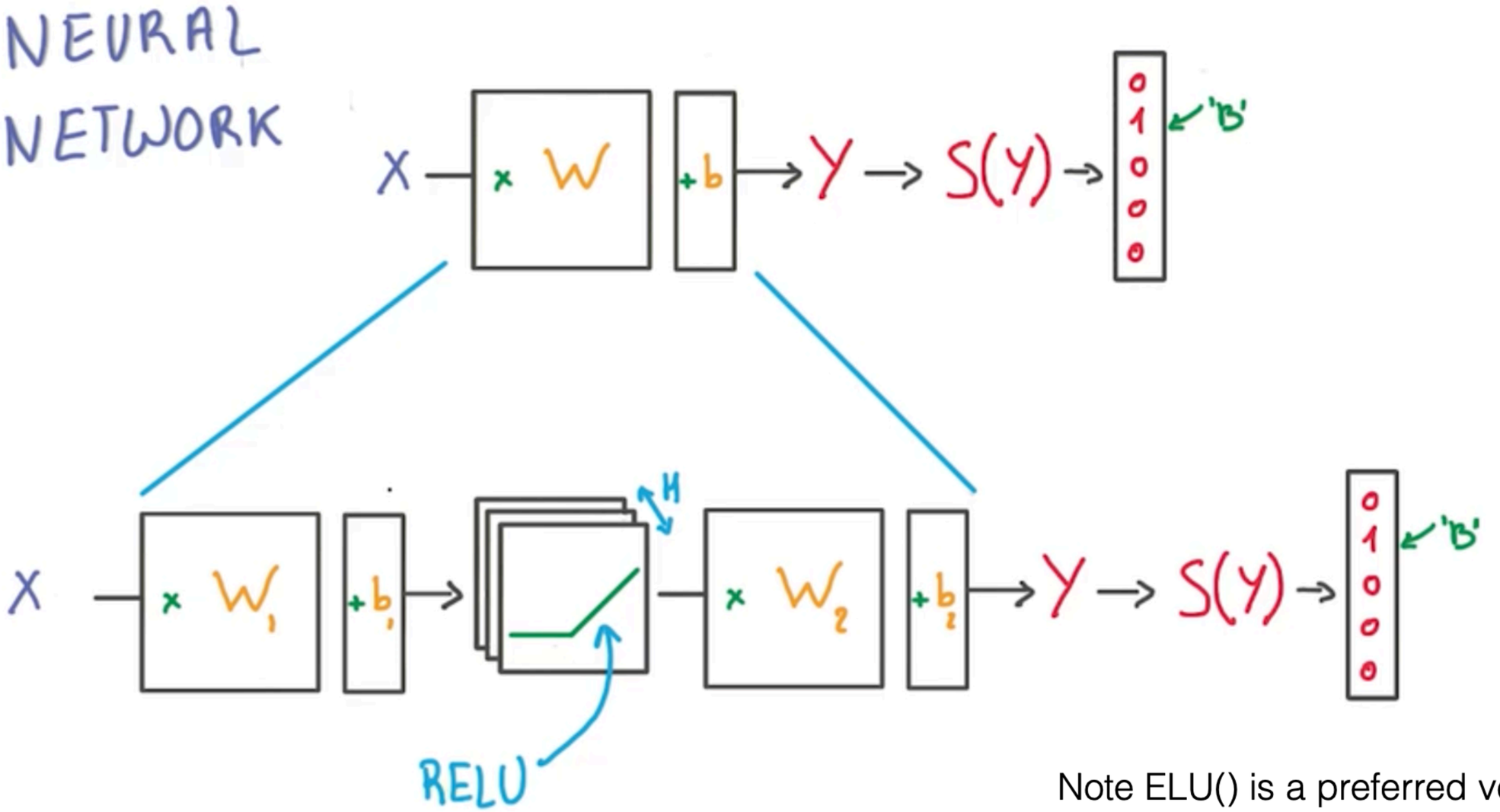
Note: For example for images you would use a more efficient model.fit_generator() method that loads images on the fly ("yield" method) in a more memory efficient way!

Simple Neural Network Linear Model



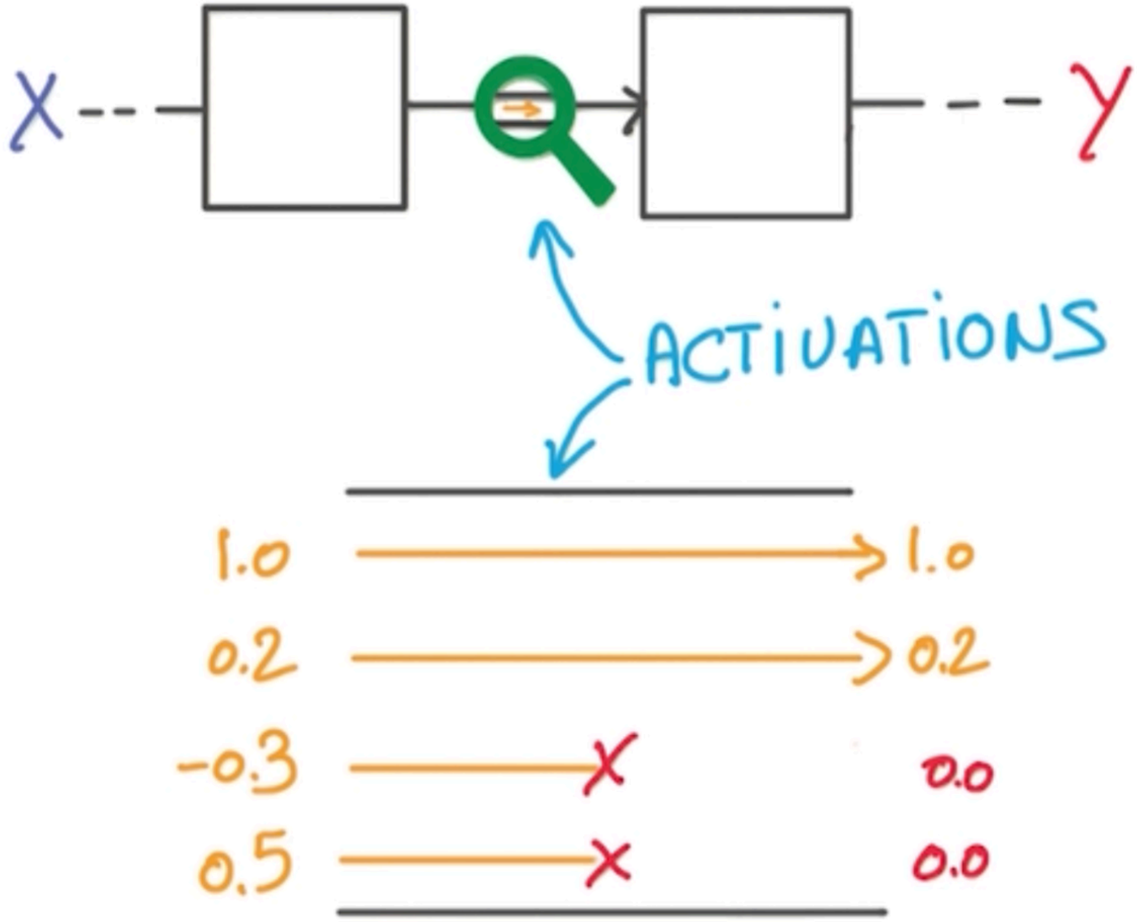
https://www.udacity.com/course/deep-learning-ud730

RELU Activation



Note ELU() is a preferred version!

Dropout



Matrix Multiplications

- Matrix Multiplication is important to understanding deep learning layers

Formula

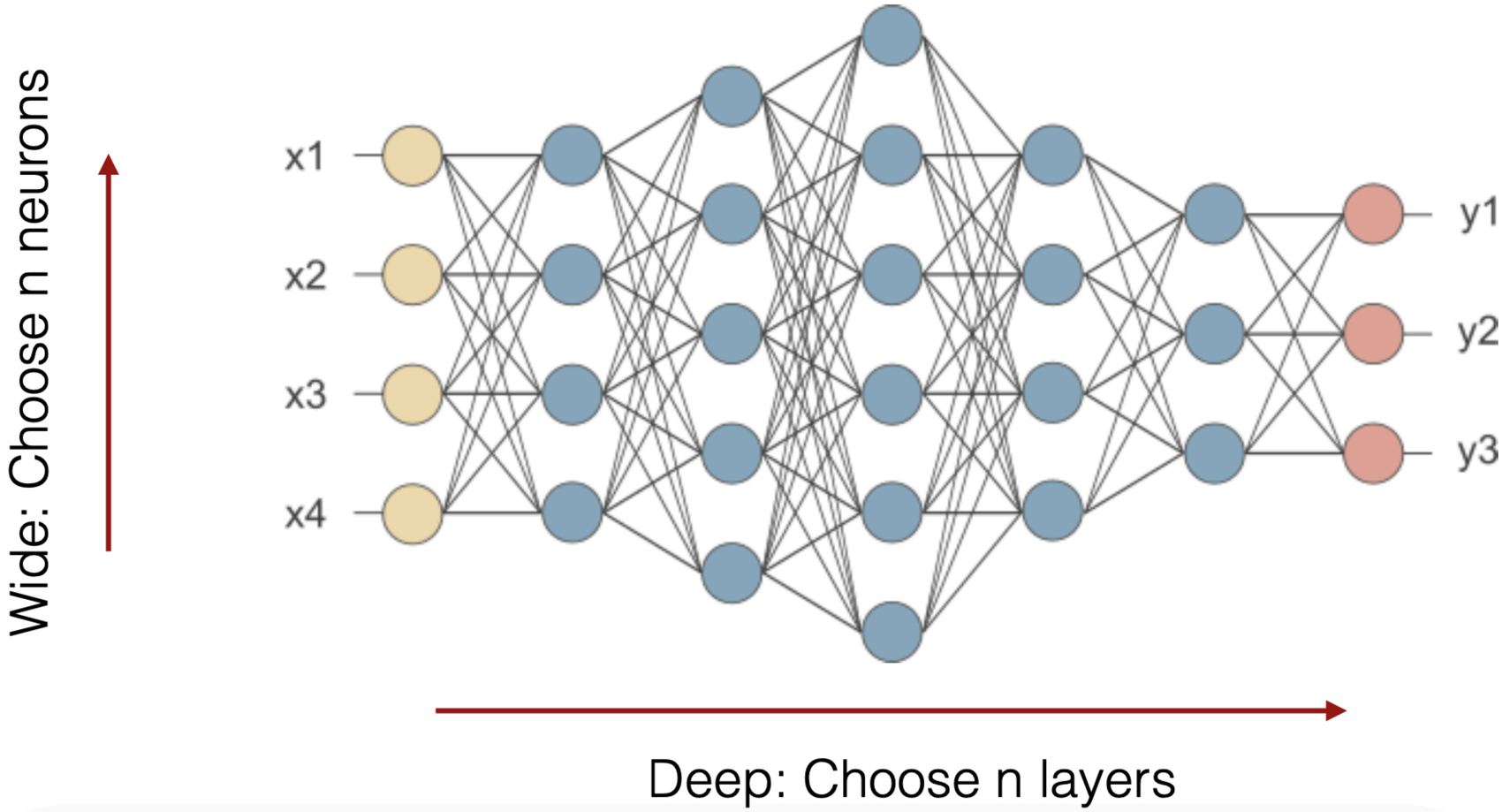
$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ax + by + cz \\ dx + ey + fz \\ gx + hy + iz \end{bmatrix}$$

Example

$$1*1 + 3*2 + 5*3 + 7*4 = 50$$

$$\begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \end{bmatrix} \times \begin{bmatrix} 1 & 8 & 9 \\ 2 & 7 & 10 \\ 3 & 6 & 11 \\ 4 & 5 & 12 \end{bmatrix} = \begin{bmatrix} 50 & 94 & 178 \\ 60 & 120 & 220 \end{bmatrix}$$

Deep vs. Wide Neural Network



word2vec embedding

Prepare Raw Data

https://www.google.com/search?q=URL

...

[18, 30, 30, 26, 29, 78, 77, 77, 33, 33, 33, 76, 17, 25, 25, 17, 22, 15, 76, 13, 25, 23, 77, 29, 15, 11, 28, 13, 18, 83, 27, 81, 57, 54, 48]

Prepare Raw Data

Summary steps data preparation

- **Step 1**: Create hash table of characters and encode chars as int. (Easy workaround use Python's 100 printable characters and their indices)

```
url_int_tokens = [[printable.index(x) for x in url if x in printable] for url in df.url]
```

- **Step 2:** Define max URL length (your choice, let's say 75 characters) and crop or pad with zeros.

```
X = sequence.pad_sequences(url_int_tokens, maxlen=75)
```

crop or pad with zeros

Step 3: Extract target vector with labels. Note in our binary case a 1D vector is fine, however for multiple categories the target vector needs to be one-hot encoded! This is different from sklearn!!!

```
target = np.array(df.isMalicious)
```

https://www.tensorflow.org/tutorials/word2vec

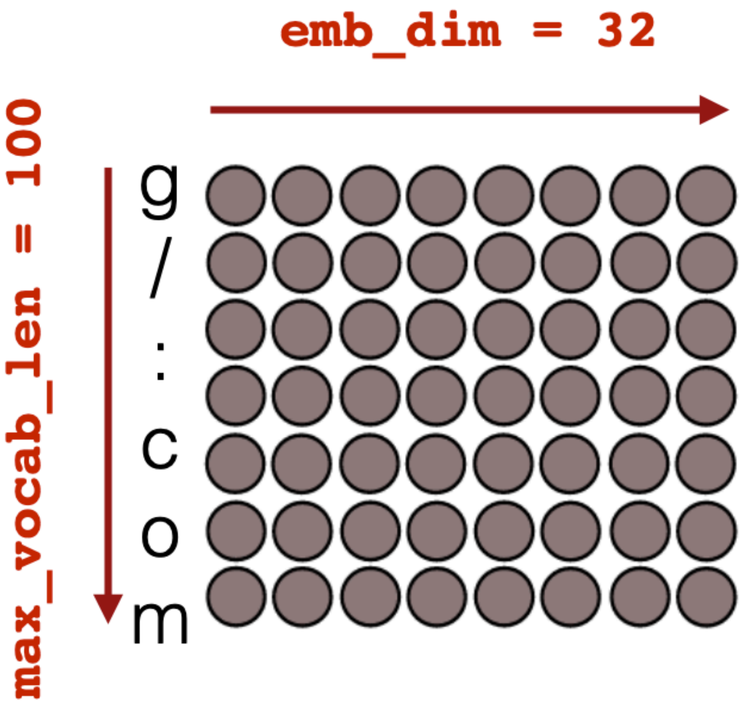
word2vec: Train Model

- **Step 4:** Train a mini neural network word2vec! Use the context of each character to embed each character into an n-dimensional space. Here: CBOW method
- You can choose the embedding dimension as well as the maximum vocabulary size.

word2vec Model Weight Matrix

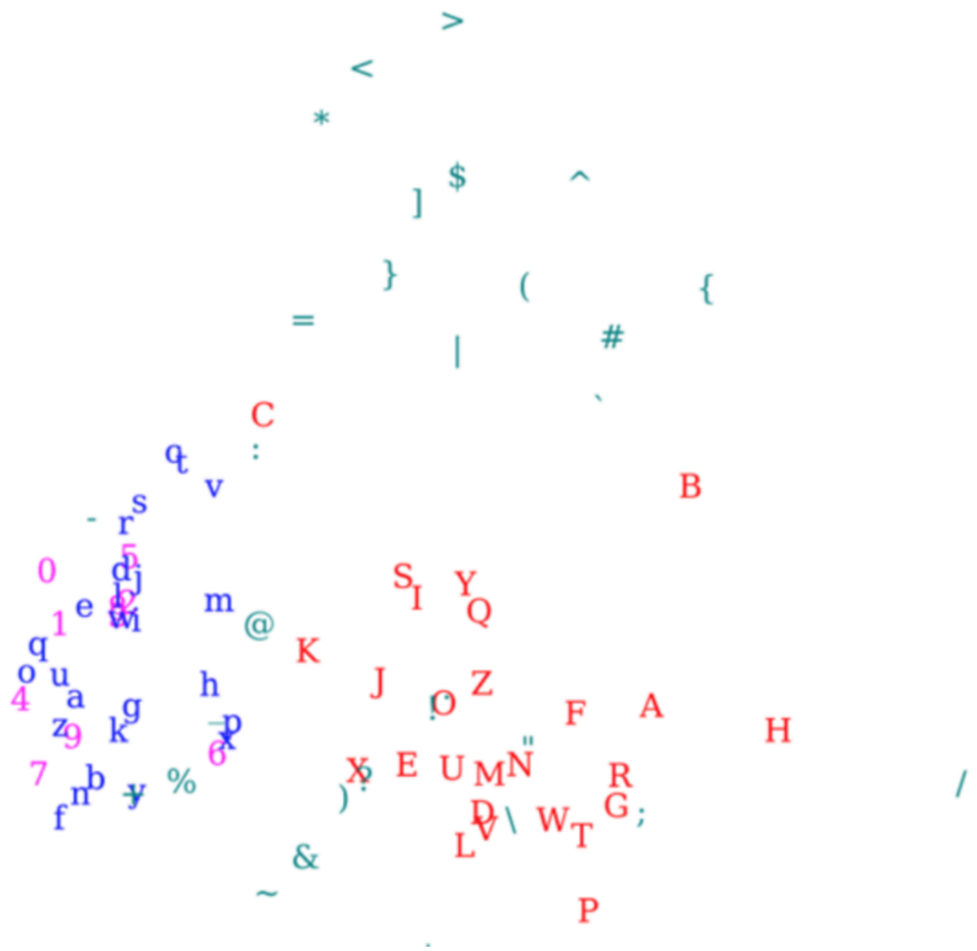
- For example here **`max_vocab_len = 100`** since we use only Python printable characters and let's arbitrarily say **`emb_dim = 32`**.

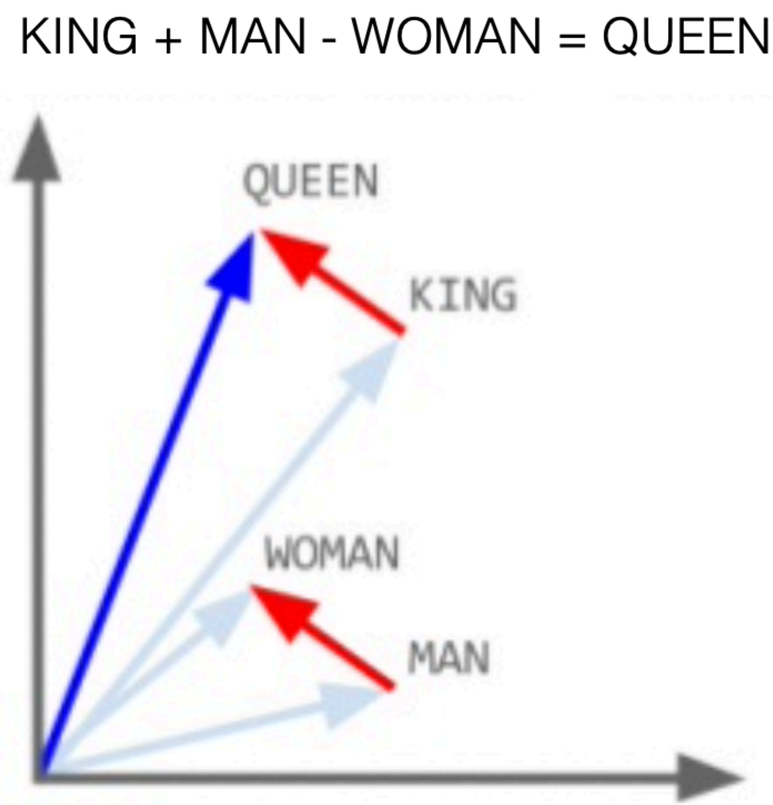
Every char in vocabulary has a vector of length embed_dim



Trained model word2vec weight matrix

Plot in 2D - Another Embedding: t-SNE



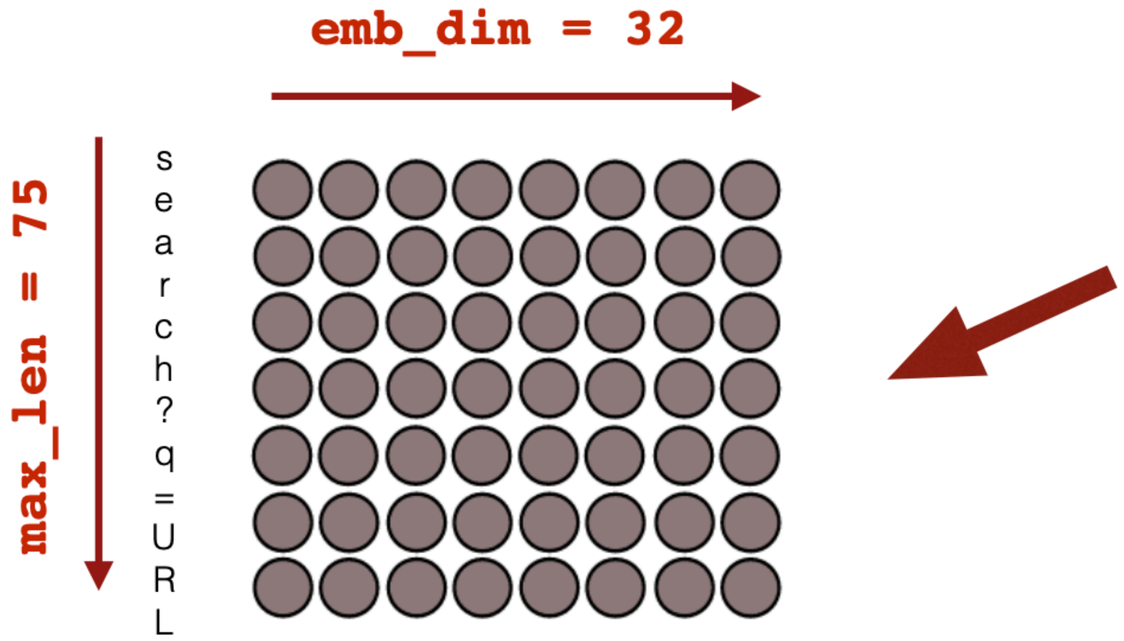


https://www.invincea.com/2017/02/look-ma-no-features-deep-learning-methods-in-intrusion-detection/

https://www.tensorflow.org/tutorials/word2vec

Transform New URLs

- Now apply your trained word2vec model of (weight matrix of `max_vocab_len = 100` x `emb_dim = 32`) to a new URL!
- The resulting embedded matrix is of shape (`max_len = 75` x `emb_dim = 32`). Note if the character is not in vocabulary the vector will be full of zeros as well as when URL is shorter than 75 chars the padded vectors will be zeros!



Matrix for one embedded URL using the previously trained word2vec model

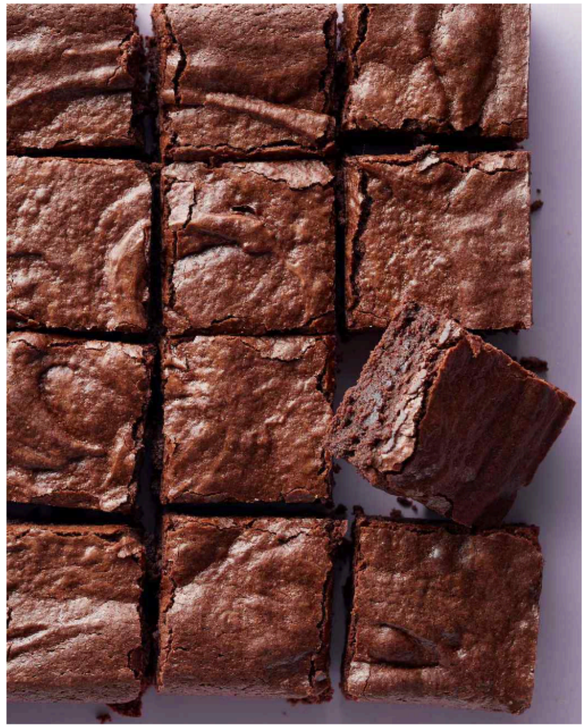
Word2Vec in Keras

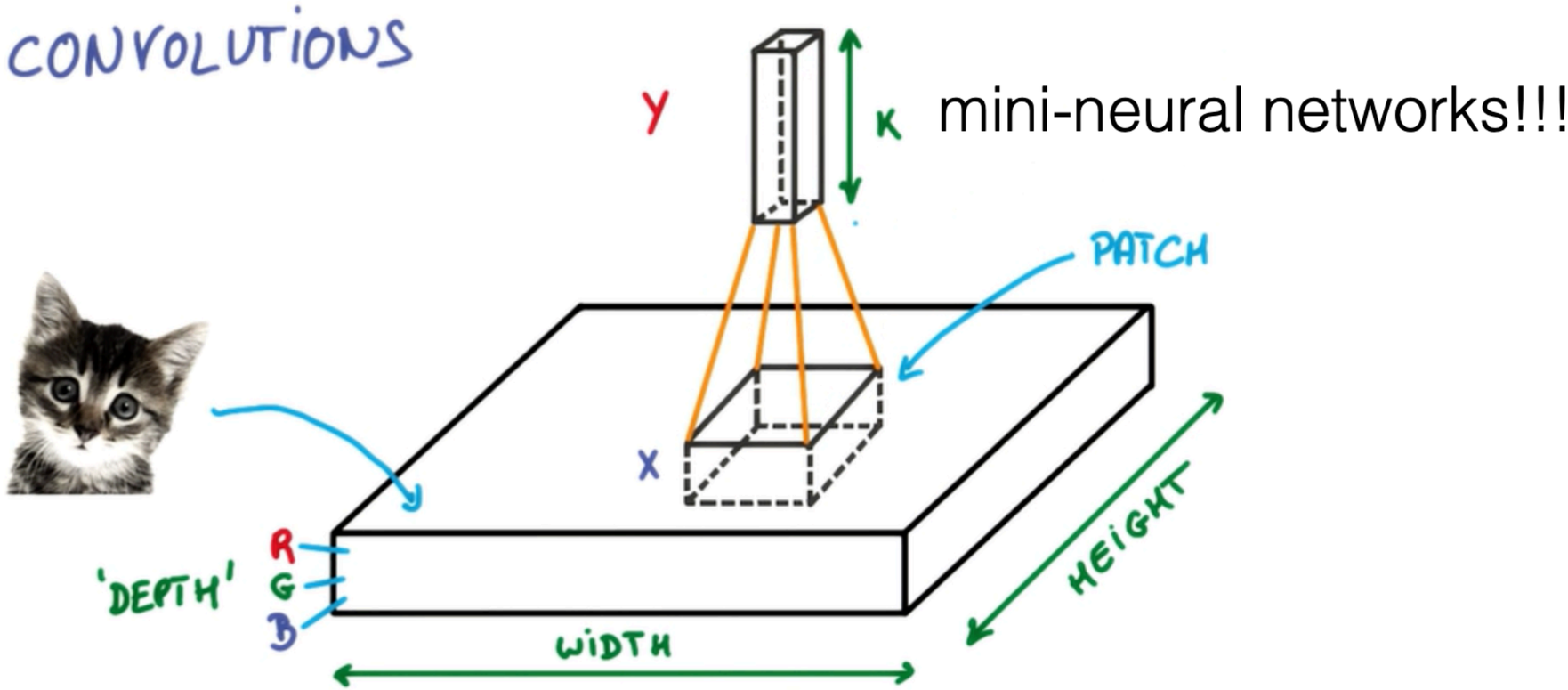
```
# Input
main_input = Input(shape=(max_len,), dtype='int32', name='main_input')
# Embedding layer
emb = Embedding(input_dim=max_vocab_len, output_dim=emb_dim, input_length=max_len,
                dropout=0.2, W_regularizer=W_reg)(main_input)
emb = Dropout(0.25)(emb)
```

```
# Track dimensions!
<keras.engine.topology.InputLayer object at 0x7fea614d10b8>
Input Shape:  (None, 75) Output Shape:  (None, 75)
<keras.layers.embeddings.Embedding object at 0x7fea614d10f0>
Input Shape:  (None, 75) Output Shape:  (None, 75, 32)
<keras.layers.core.Dropout object at 0x7fea63cb2e80>
Input Shape:  (None, 75, 32) Output Shape:  (None, 75, 32)
```

“None” is always batch size! Compared to sklearn and regular 2D feature matrices we have a 3D array/tensor at first!

2D Convolutions with Images



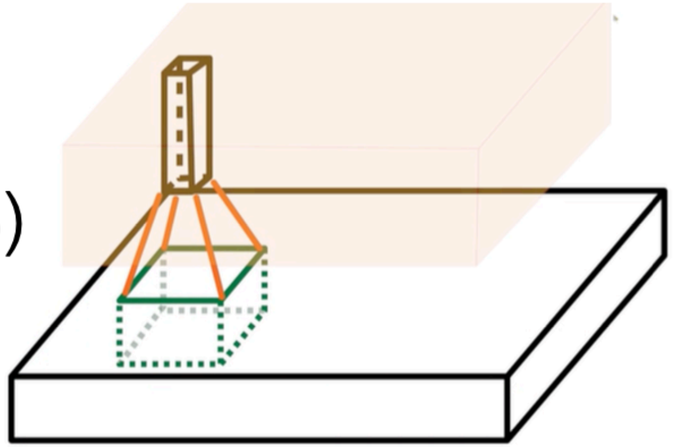


You choose K!!!
Here it's the new k color channels/
depth of picture (`nb_filter` or `filters` in keras)

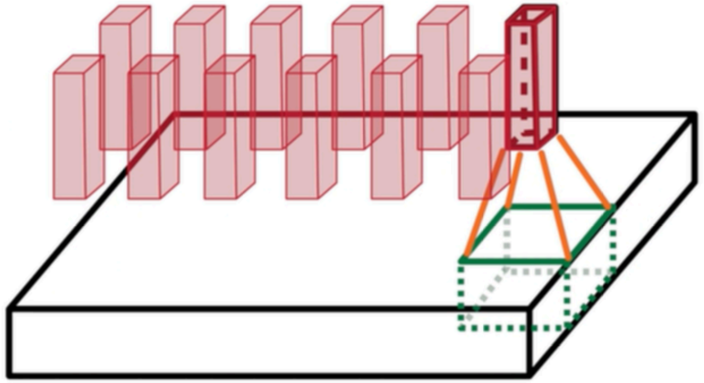
https://www.udacity.com/course/deep-learning--ud730

2D Convolution with Images

For example:
Filter Size (3,3)



Slide with stride 1

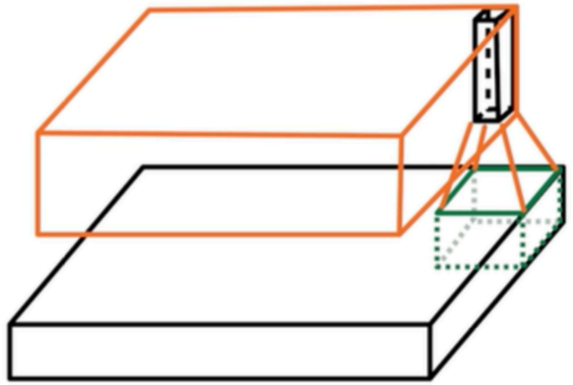


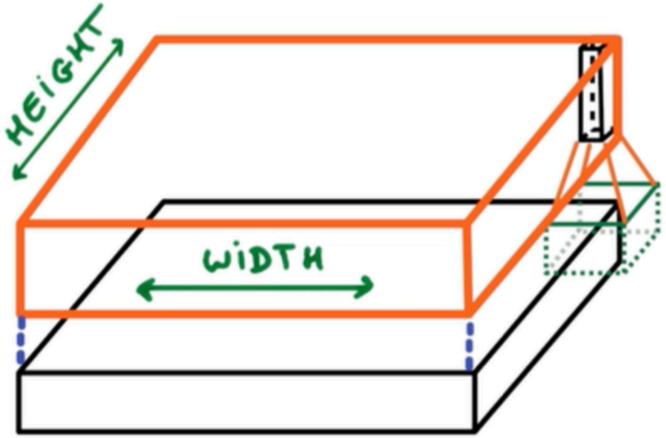
Slide with stride 2

CONVOLUTIONAL LINGO

'VALID' PADDING

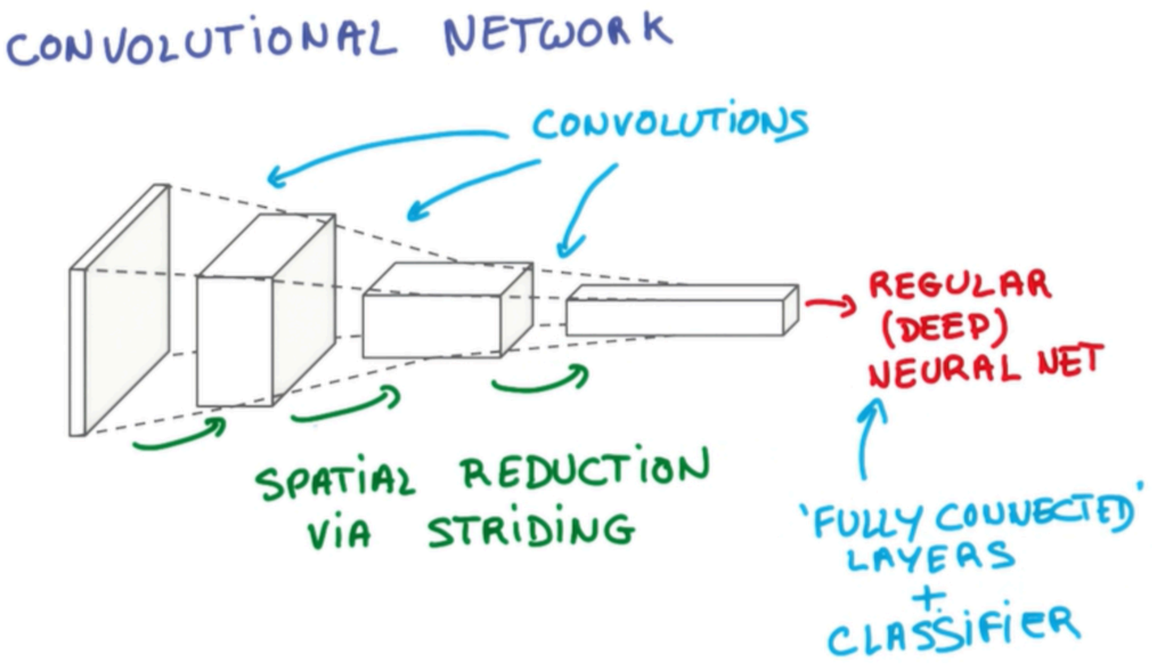
'SAME' PADDING



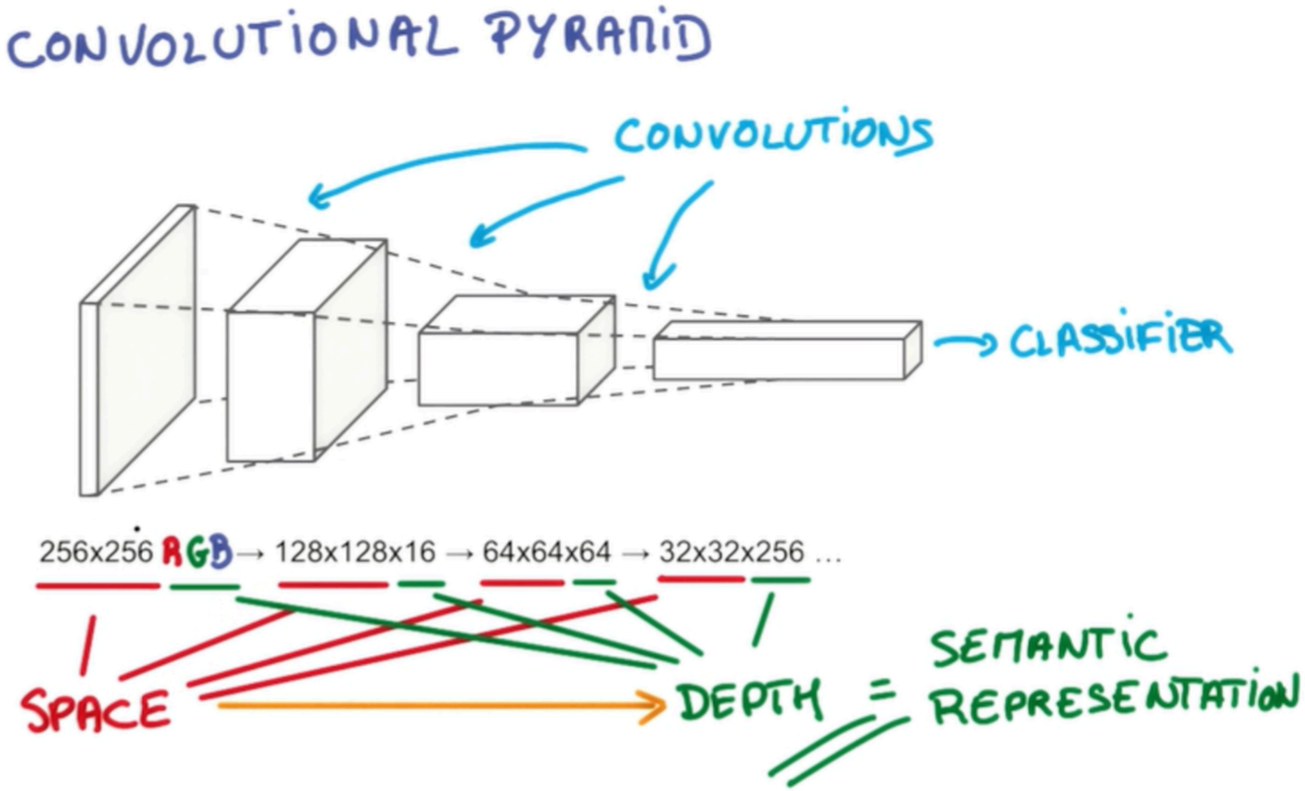


Define Padding Type

2D Convolution with Images

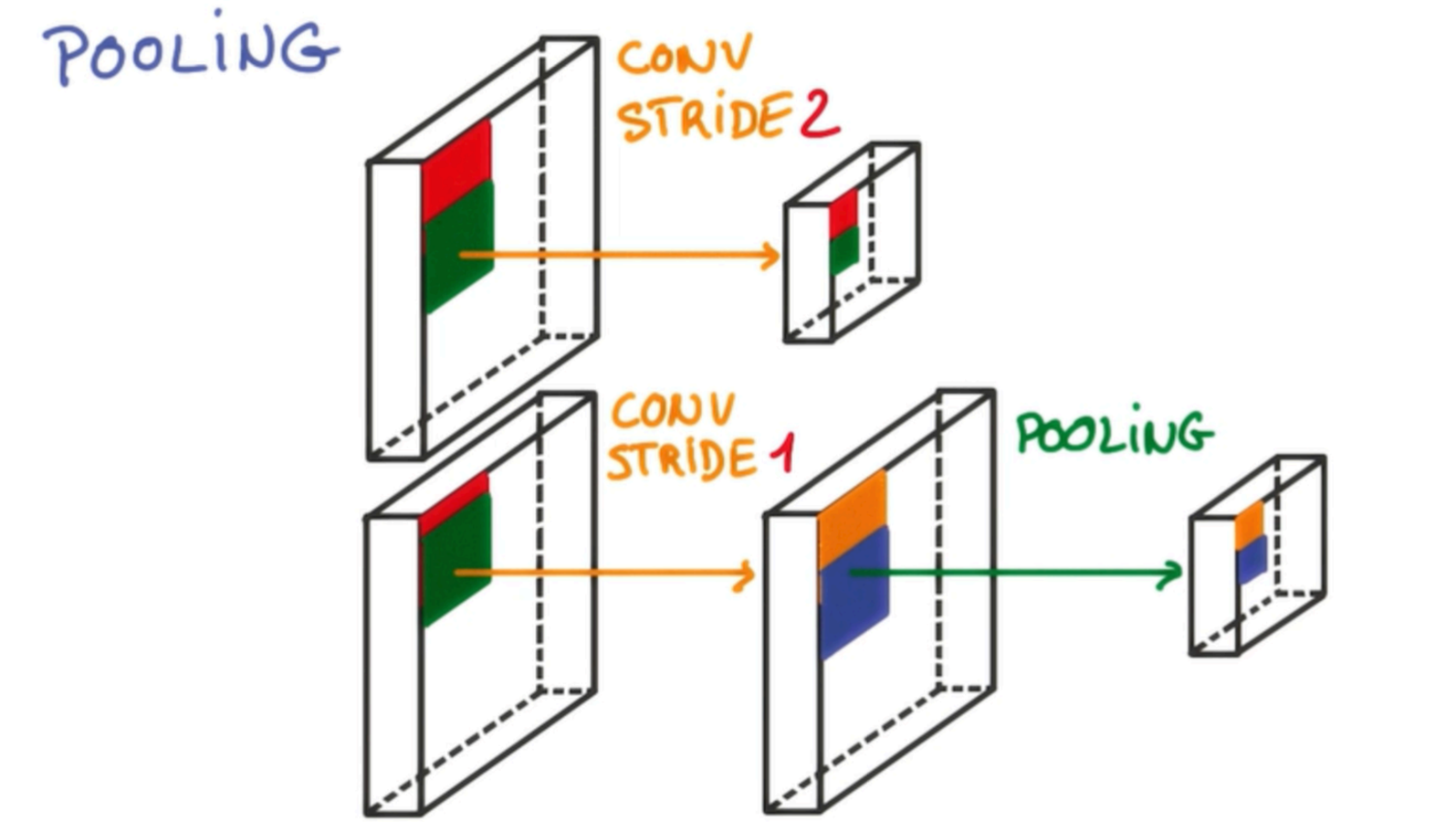


Simple Case



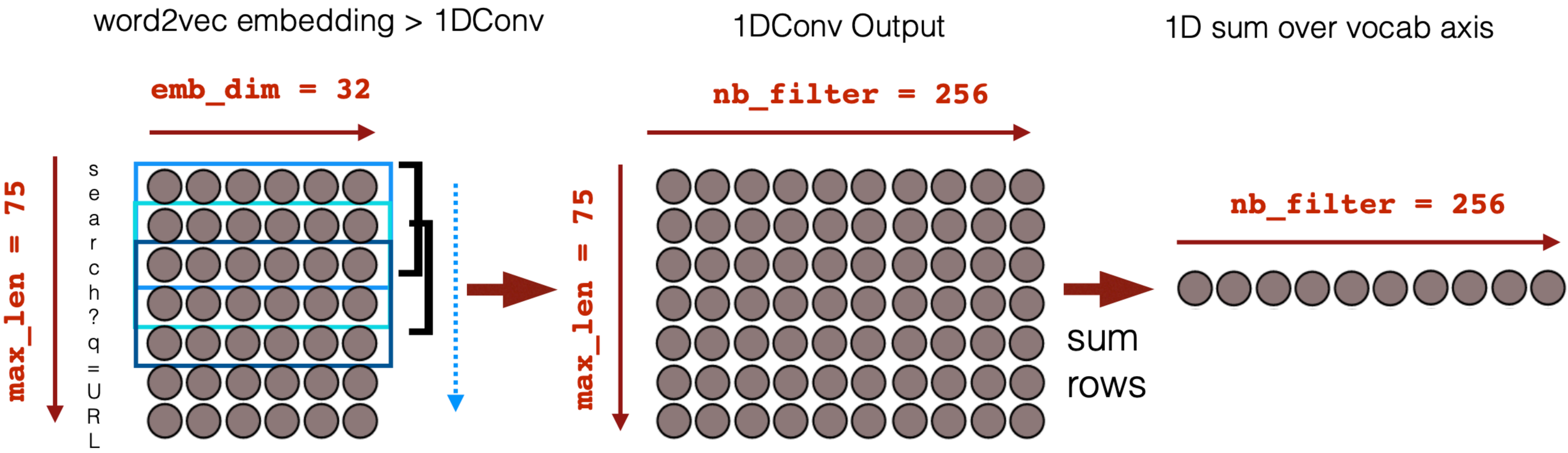
More progressive via additional methods

2D Convolution with Images



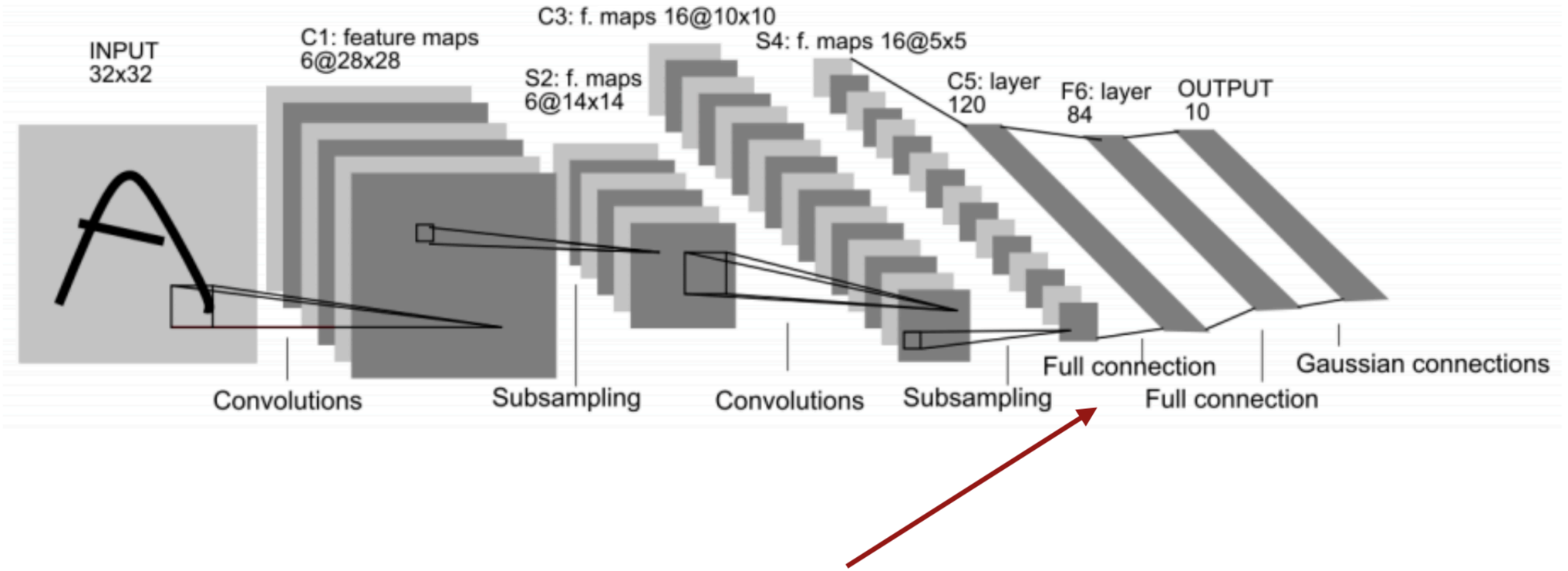
- Max Pooling
- Mean Pooling
- Sum

1D Convolution with Text



filter_length = 3
nb_filter = 256
border_mode = ‘same’
strides=1

Fully Connected Layers



Think of “regular feature matrix” (batch_size x n_neurons) at the end where you usually reduce the neurons each step before fitting the output layer

Keras Architecture - Con + Fully Connected

```
# Input
main_input = Input(shape=(max_len,), dtype='int32', name='main_input')
# Embedding layer
emb = Embedding(input_dim=max_vocab_len, output_dim=emb_dim, input_length=max_len,
                dropout=0.2, W_regularizer=W_reg)(main_input)
emb = Dropout(0.25)(emb)

# Multiple Conv Layers

# calling custom conv function from above
conv1 = get_conv_layer(emb, filter_length=2, nb_filter=256)
conv2 = get_conv_layer(emb, filter_length=3, nb_filter=256)
conv3 = get_conv_layer(emb, filter_length=4, nb_filter=256)
conv4 = get_conv_layer(emb, filter_length=5, nb_filter=256)

# Fully Connected Layers
merged = merge([conv1,conv2,conv3,conv4],mode="concat")

hidden1 = Dense(1024,activation=ELU())(merged)
hidden1 = Dropout(0.5)(hidden1)

hidden2 = Dense(1024,activation=ELU())(hidden1)
hidden2 = Dropout(0.5)(hidden2)

# Output layer (last fully connected layer)
output = Dense(1, activation='sigmoid', name='output')(hidden2)
```

```
def sum_1d(X):
    return K.sum(X, axis=1)


def get_conv_layer(emb, filter_length=5, nb_filter=256):

    # Conv layer
    conv = Convolution1D(filter_length=filter_length,
    nb_filter=nb_filter, border_mode='same',
    activation=ELU())(emb)

    conv = Lambda(sum_1d, output_shape=(nb_filter,))(conv)
    #conv = BatchNormalization(mode=0)(conv)
    conv = Dropout(0.5)(conv)
    return conv
```

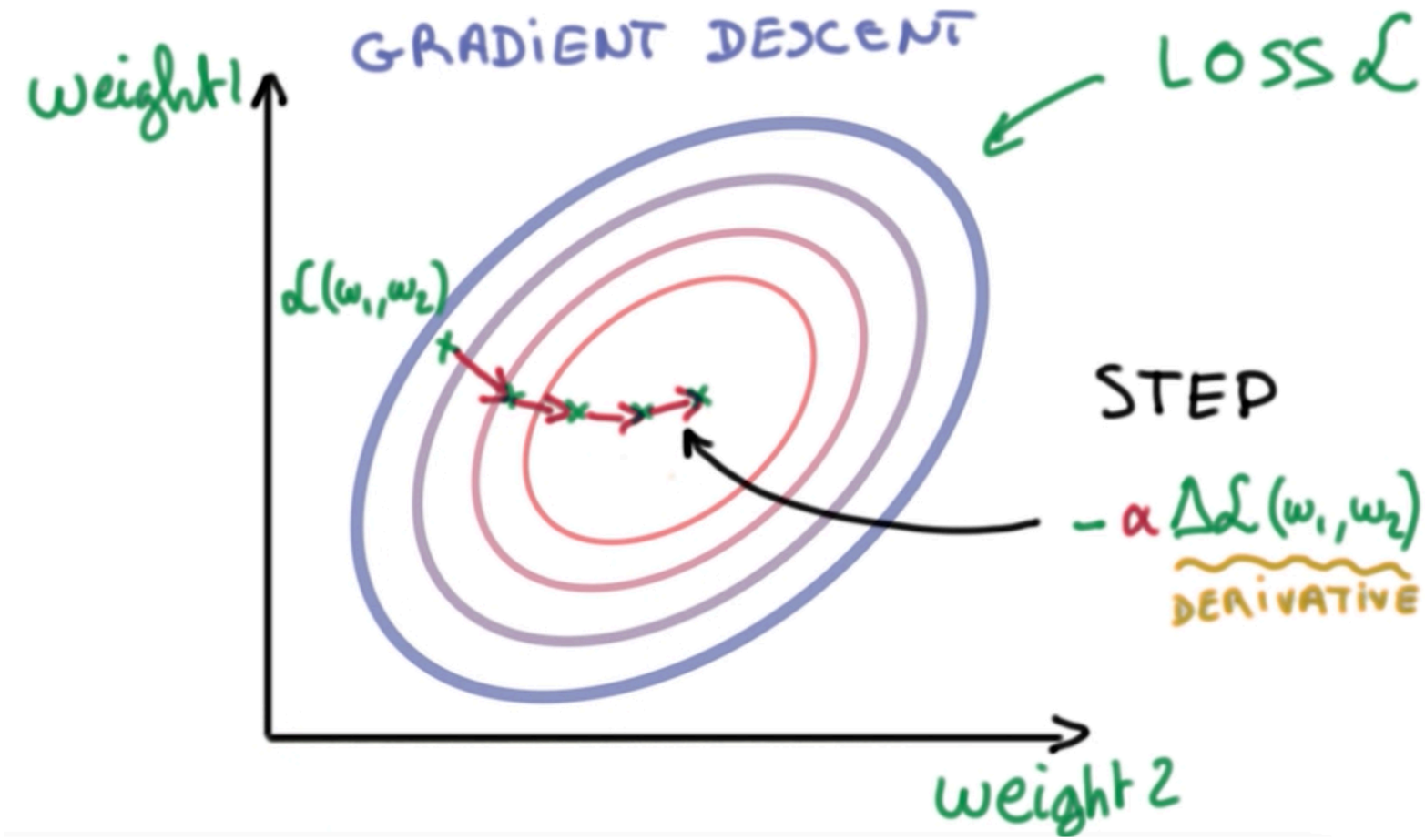
Keras Architecture - Con + LSTM

```
# Input
main_input = Input(shape=(max_len,), dtype='int32', name='main_input')
# Embedding layer
emb = Embedding(input_dim=max_vocab_len, output_dim=emb_dim,
input_length=max_len, dropout=0.2, W_regularizer=W_reg)(main_input)
emb = Dropout(0.25)(emb)
# Conv layer
conv = Convolution1D(filter_length=5, nb_filter=256, \
                     border_mode='same', activation=ELU())(emb)
conv = MaxPooling1D(pool_length=4)(conv)
#conv = BatchNormalization(mode=0)(conv)
conv = Dropout(0.5)(conv)
# LSTM layer
lstm = LSTM(lstm_output_size)(conv)
lstm = Dropout(0.5)(lstm)
# Output layer (last fully connected layer)
output = Dense(1, activation='sigmoid', name='output')(lstm)
```

Keras Architecture - Simple LSTM

```
# Input
main_input = Input(shape=(max_len,), dtype='int32',
name='main_input')

# Embedding layer
emb = Embedding(input_dim=max_vocab_len, output_dim=emb_dim,
input_length=max_len, dropout=0.2, W_regularizer=W_reg)
(main_input)

# LSTM layer
lstm = LSTM(lstm_output_size)(emb)
lstm = Dropout(0.5)(lstm)

# Output layer (last fully connected layer)
output = Dense(1, activation='sigmoid', name='output')(lstm)
```

Neural Network Optimization



Questions?