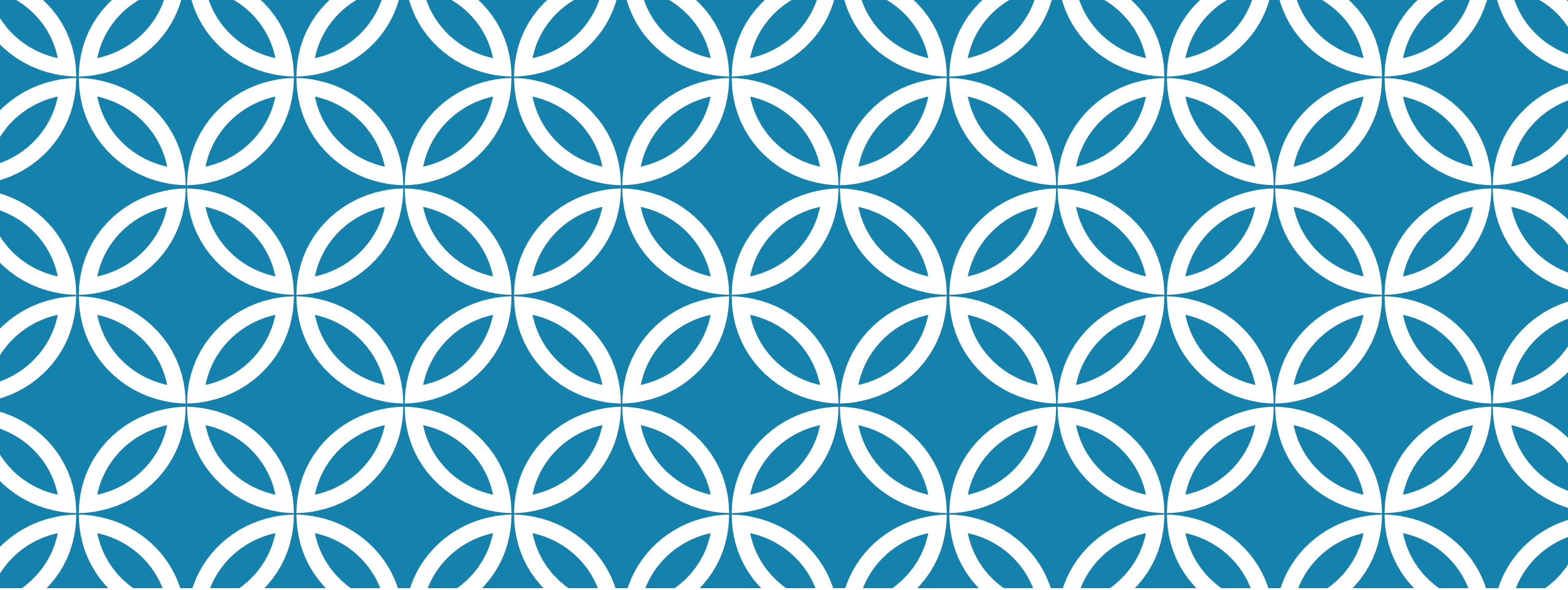


DEEPLEARNING

M.Lukac

Deep Learning

What society thinks I do

What my friends think I do

What other computer scientists think I do

What mathematicians think I do

What I think I do

```
from theano import *
```

What I actually do

WHY NOW?

GPU!

Good data, good annotated data(ImageNet).

Some great simple ideas.

Most of the techniques are old 20-30 years.

99% is matrix multiplications.

NEURAL NETWORKS

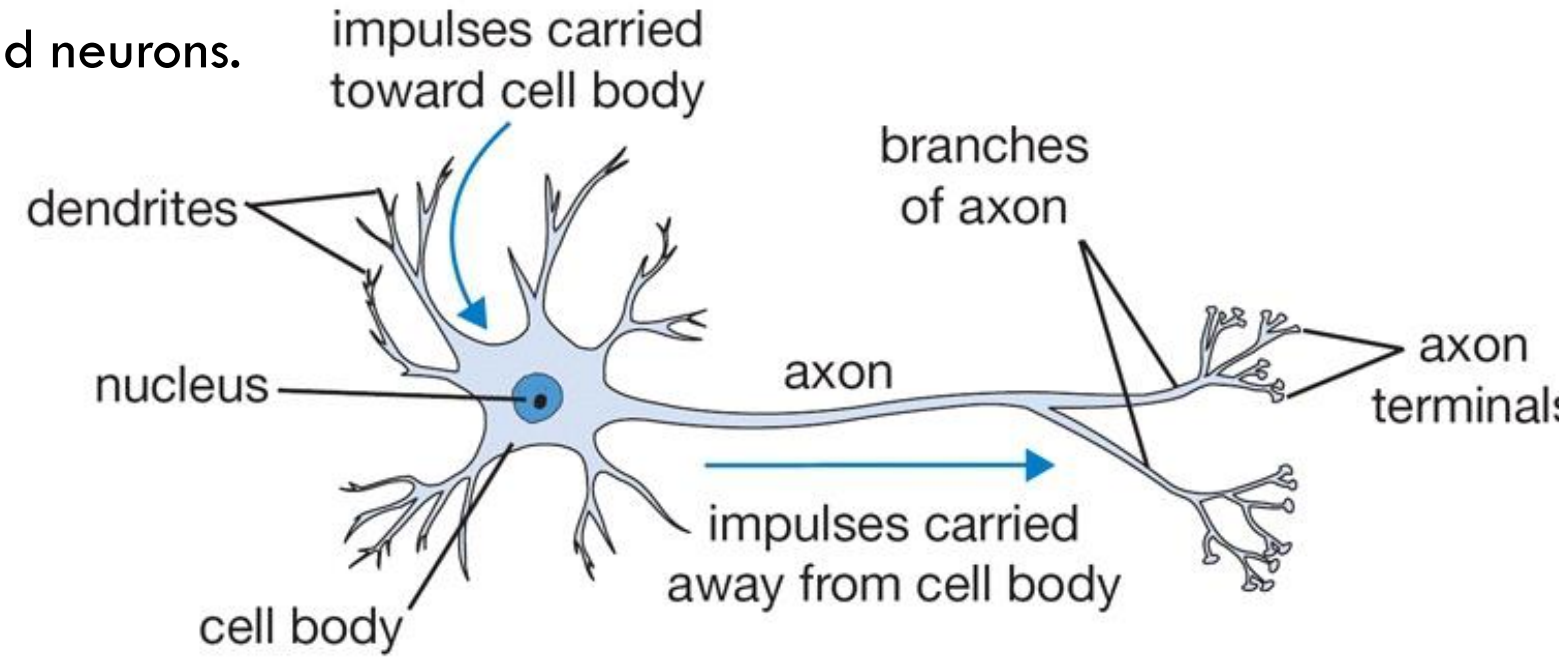
Beginning in 50s and 60s.

Biologically inspired by brain and neurons.

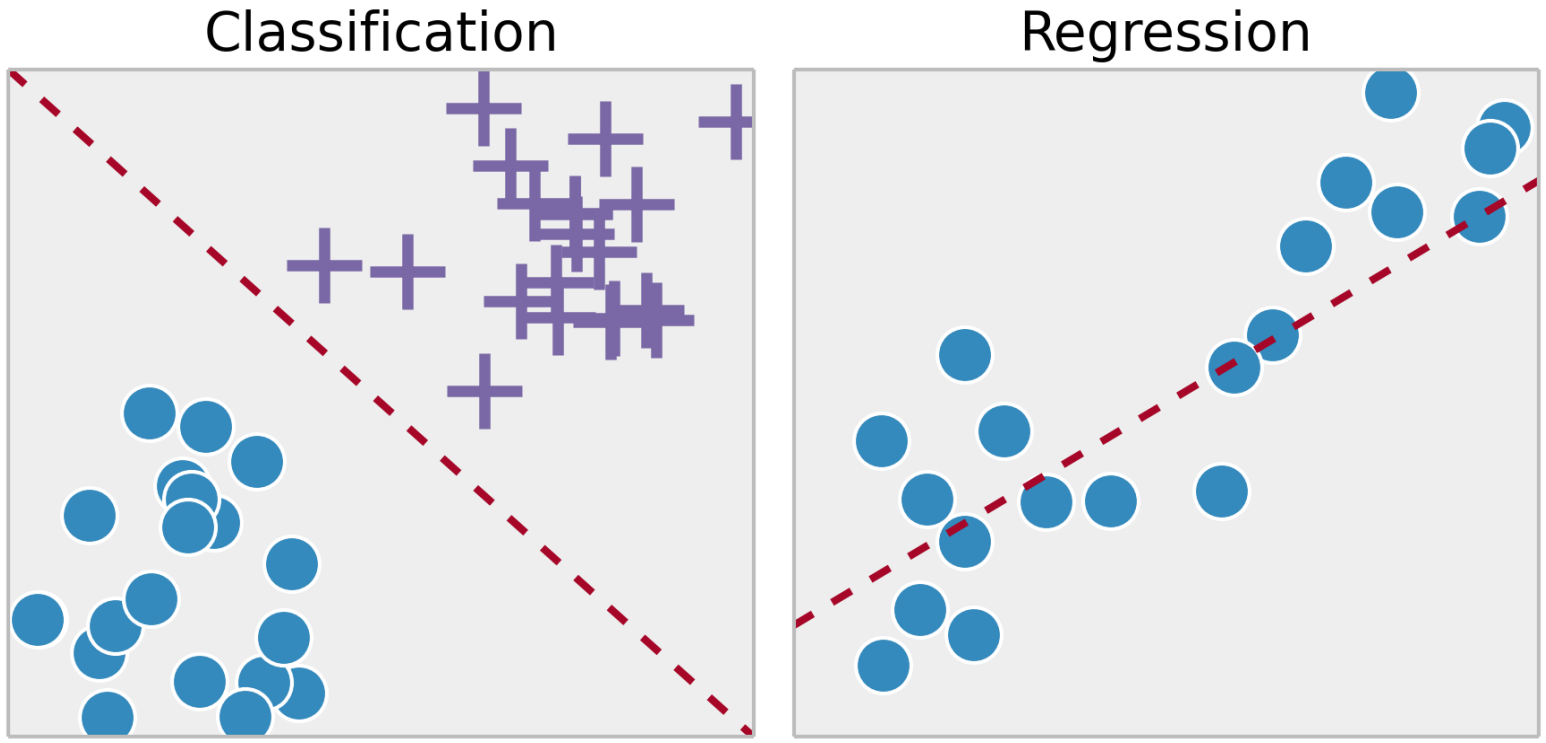
Boom every ten years...

State of the art for multimedia

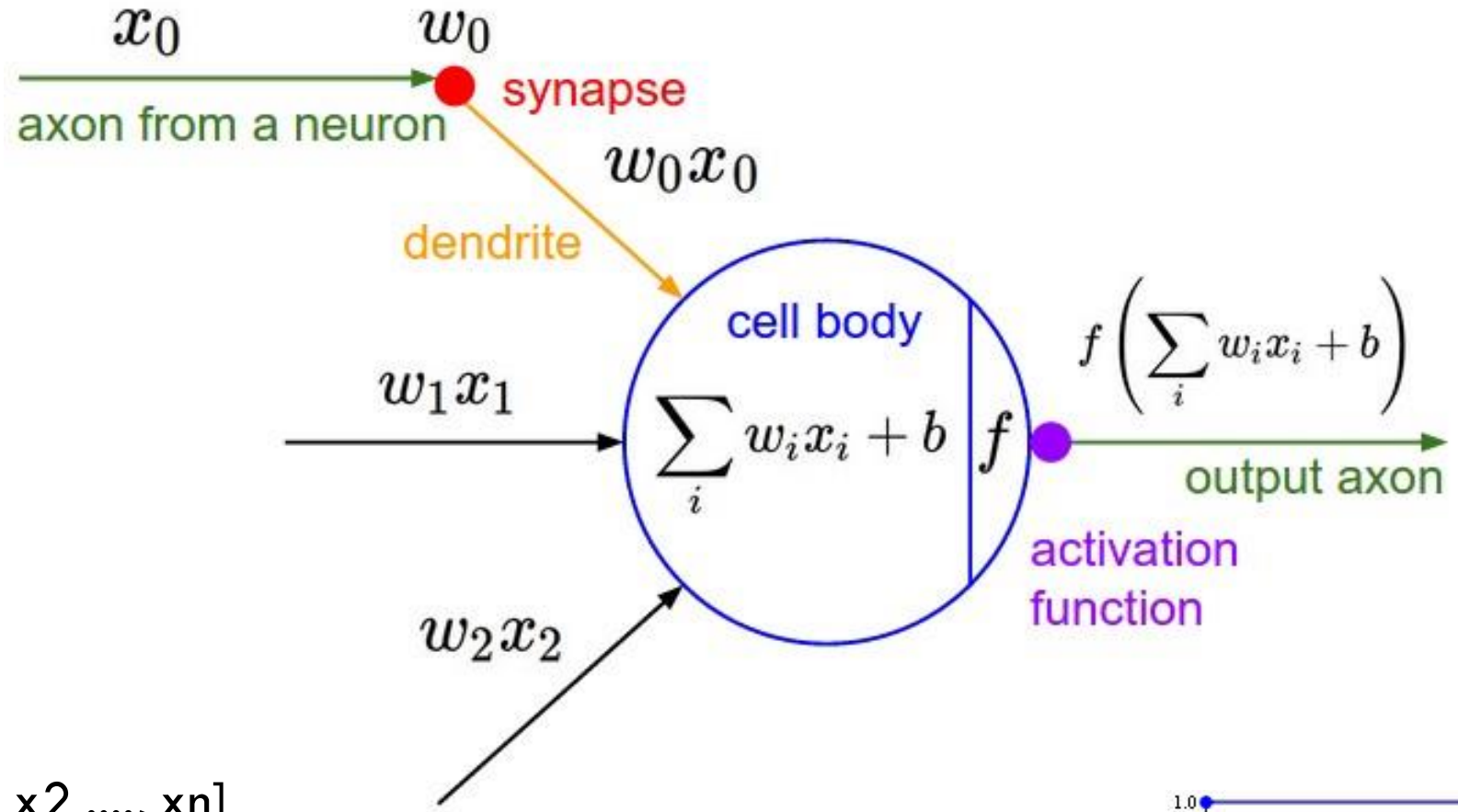
data processing!



PROBLEMS?



NEURON



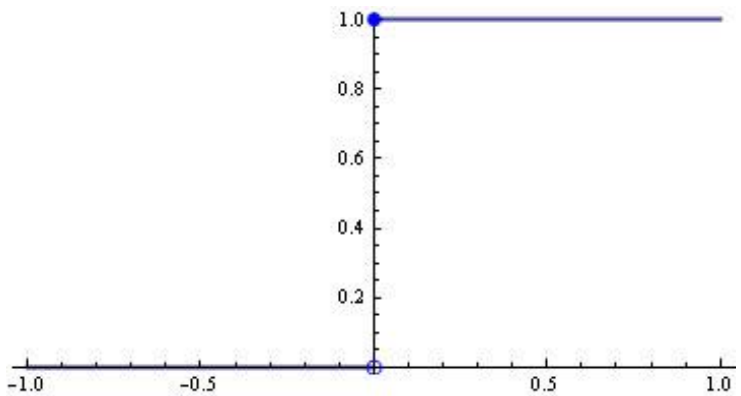
Perceptron

Output is y=f (Wx+b)

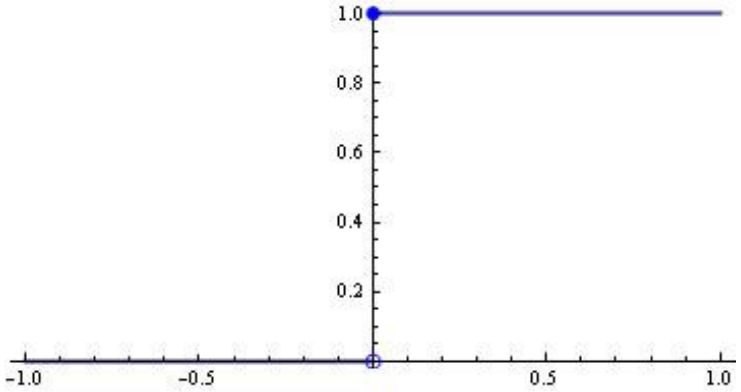
f is activation function

Input x = [x0=1.0, x1, x2 …., xn]

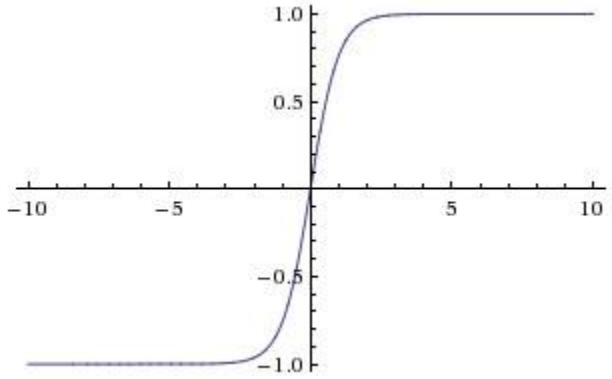
Weights W = [w0, w1, w1, ….., wn]



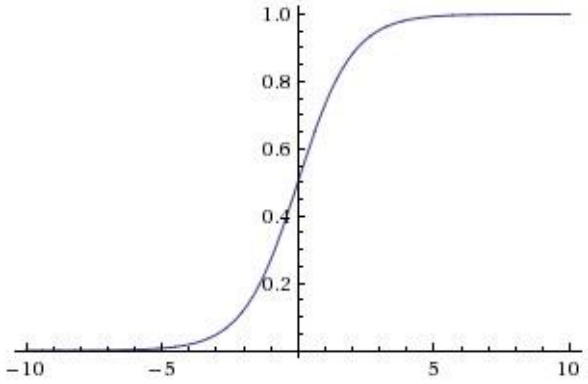
ACTIVATION FUNCTION



Step Function



TanH

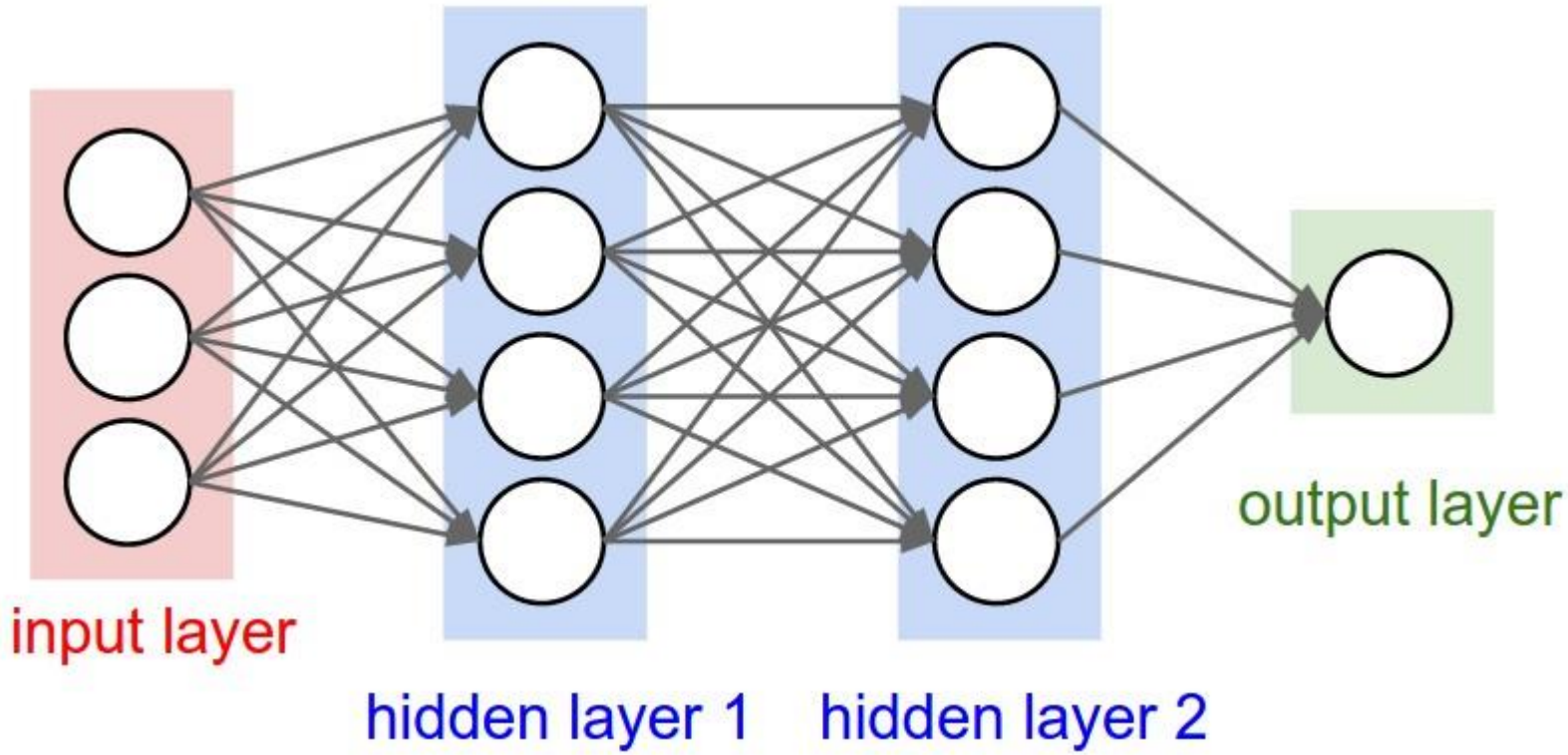


Sigmoid

SIMPLE IMPLEMENTATION

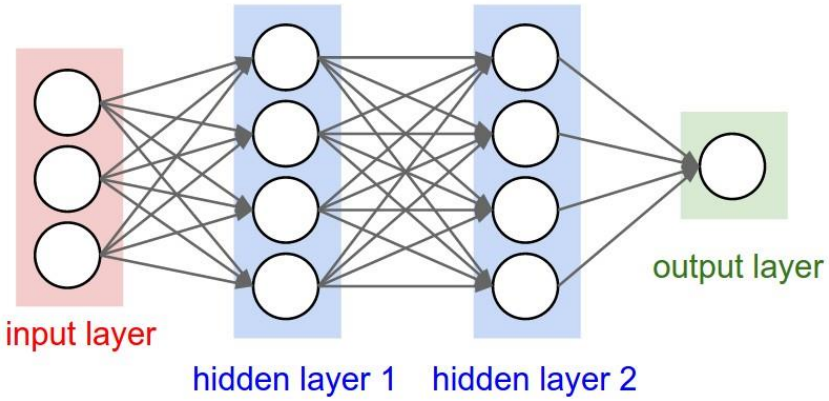
```
class Neuron(object):
  def output(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    # sigmoid activation function
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum))
    return firing_rate
```

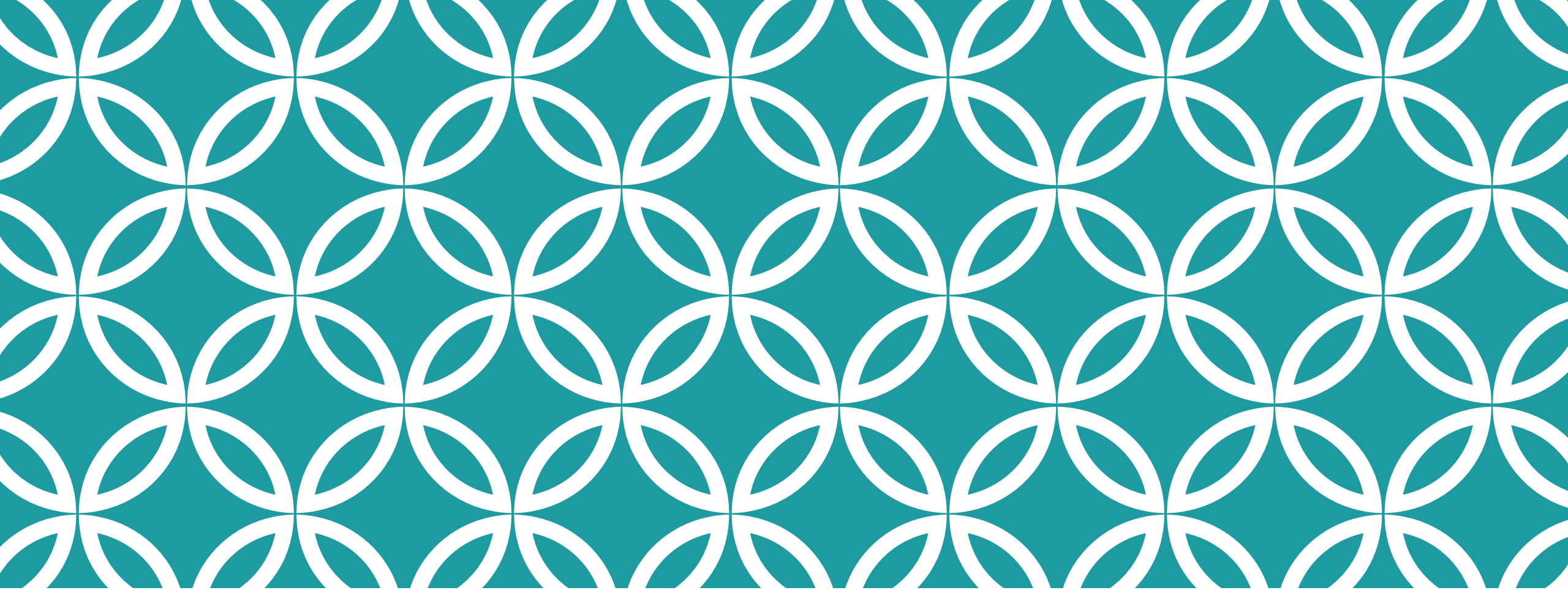
NEURAL NETWORK



SIMPLE IMPLEMENTATION

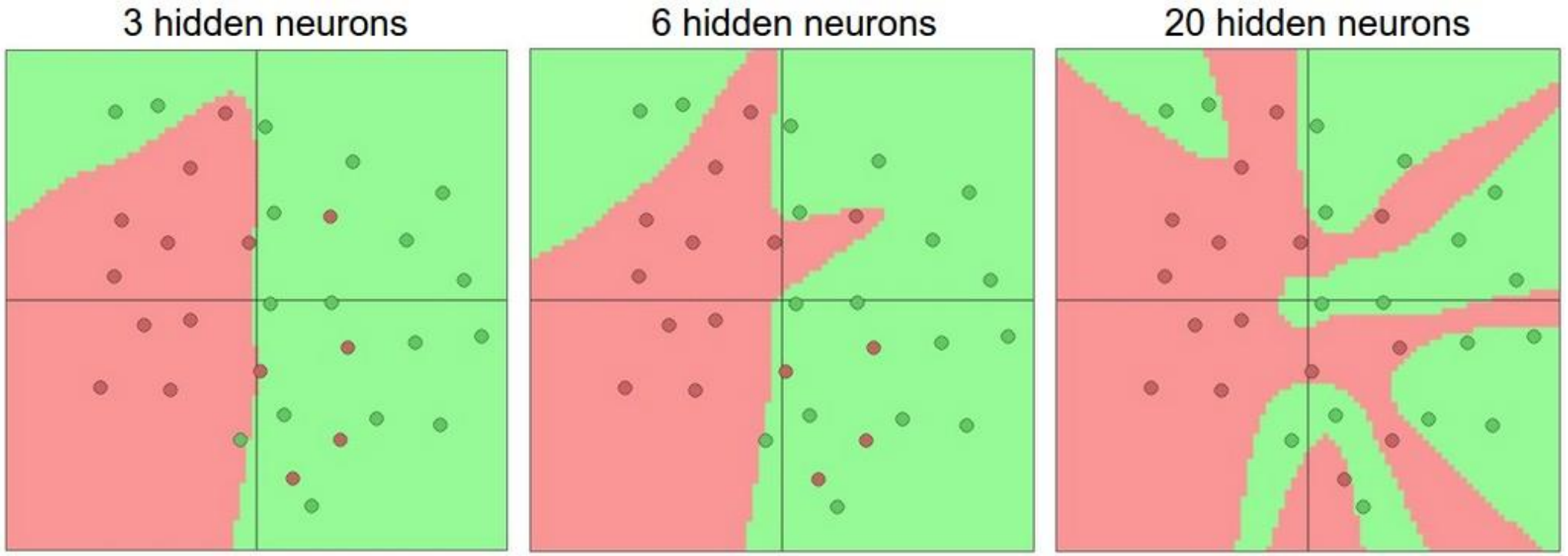
```
class THREELayerNetwork(object):
  def output(inputs):
    # forward-pass of a 3-layer neural network:
    f = lambda x: 1.0/(1.0 + np.exp(-x)) # sigmoid
    h1 = f(np.dot(W1, inputs) + b1)
    h2 = f(np.dot(W2, h1) + b2)
    out = np.dot(W3, h2) + b3
    return out
```





PLAYGROUND.TENSORFLOW.ORG

OVERFITTING?



HOW TO LEARN WEIGHTS? #VERY SIMPLIFIED

Weights are learned with Backpropagation algorithm(gradient learning).

$$\nabla G = (\frac{\partial G}{x_1}, \dots \frac{\partial G}{x_n})$$

"gradient points in the direction of the greatest rate of increase of the function, components are p. derivatives"

Objective function for bin. Classification[0,1]: Binary Cross Entropy

$$J(W) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) log(1 - \hat{y}^{(i)})]$$

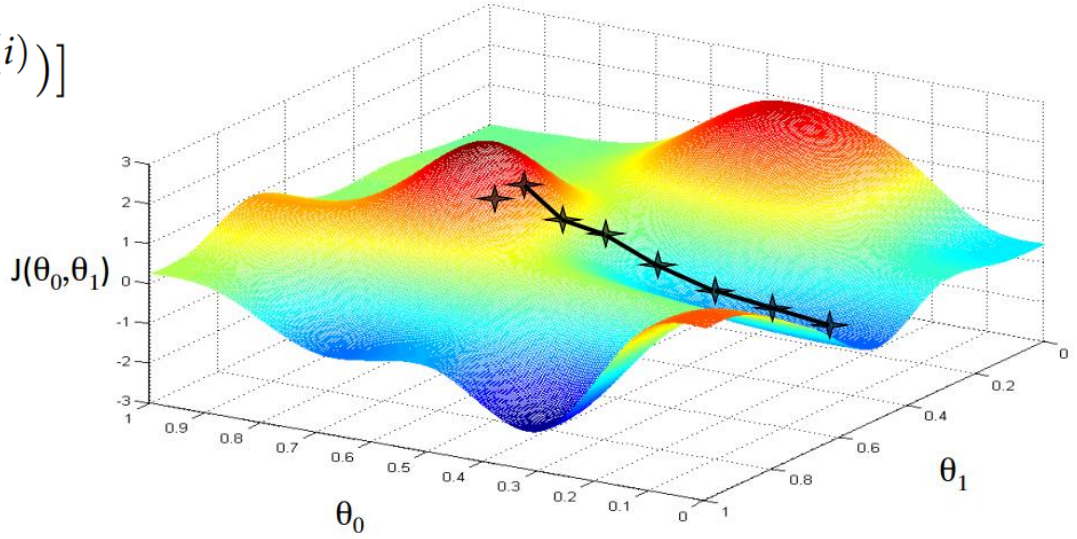
Gradient for last layer:

$$\nabla J = (\hat{y}^{(i)} - y^{(i)})x$$

You need to propagate the error through the network.

Update weights(alpha is learning rate like 0.01):

$$W_{j+1} = W_j - \alpha \frac{1}{M} \nabla J(W)$$



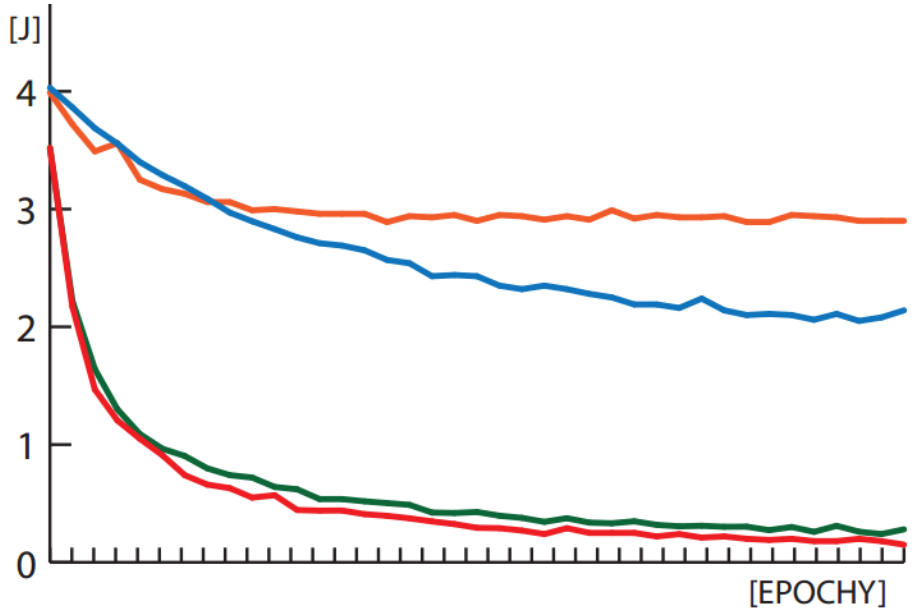
TRAINING

Monitor the objective function: It should decrease over time.

Play with learning rate alpha=[0.1, 0.001, 0.05, ...]

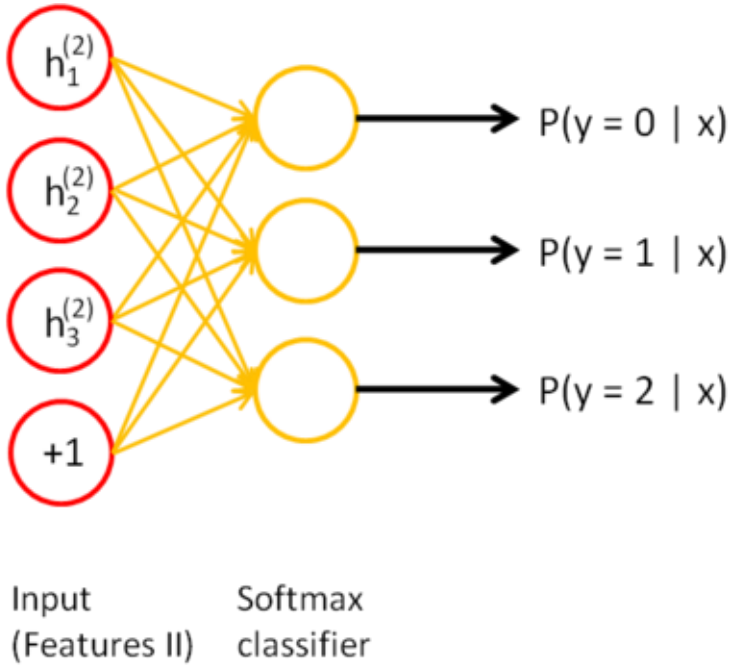
Train with mini batch of samples.

Normalize your data to <0,1>, ...



SOFTMAX LAYER

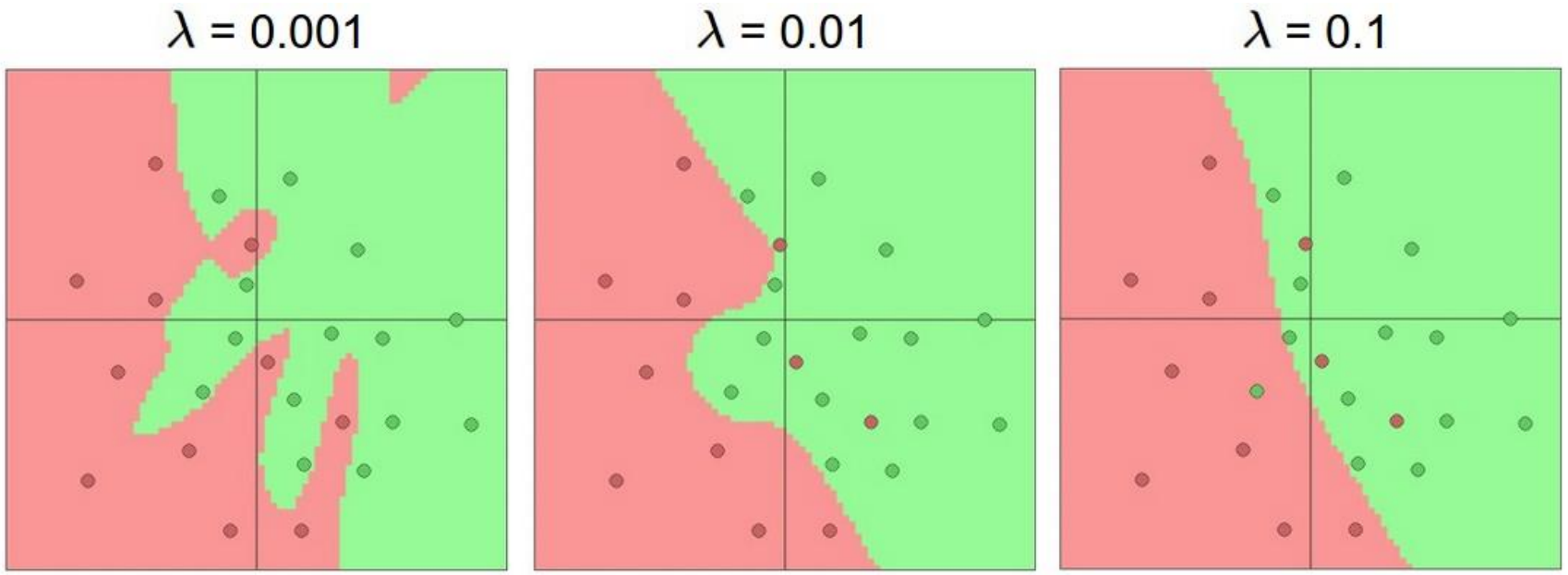
A softmax layer takes the activations and divides each of them by the sum of all activations, thereby forcing the outputs of the layer to take the form of probability distribution (sum to 1).



$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

OVERFITTING

$$R(W) = \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (W_{ji}^{l})^2$$

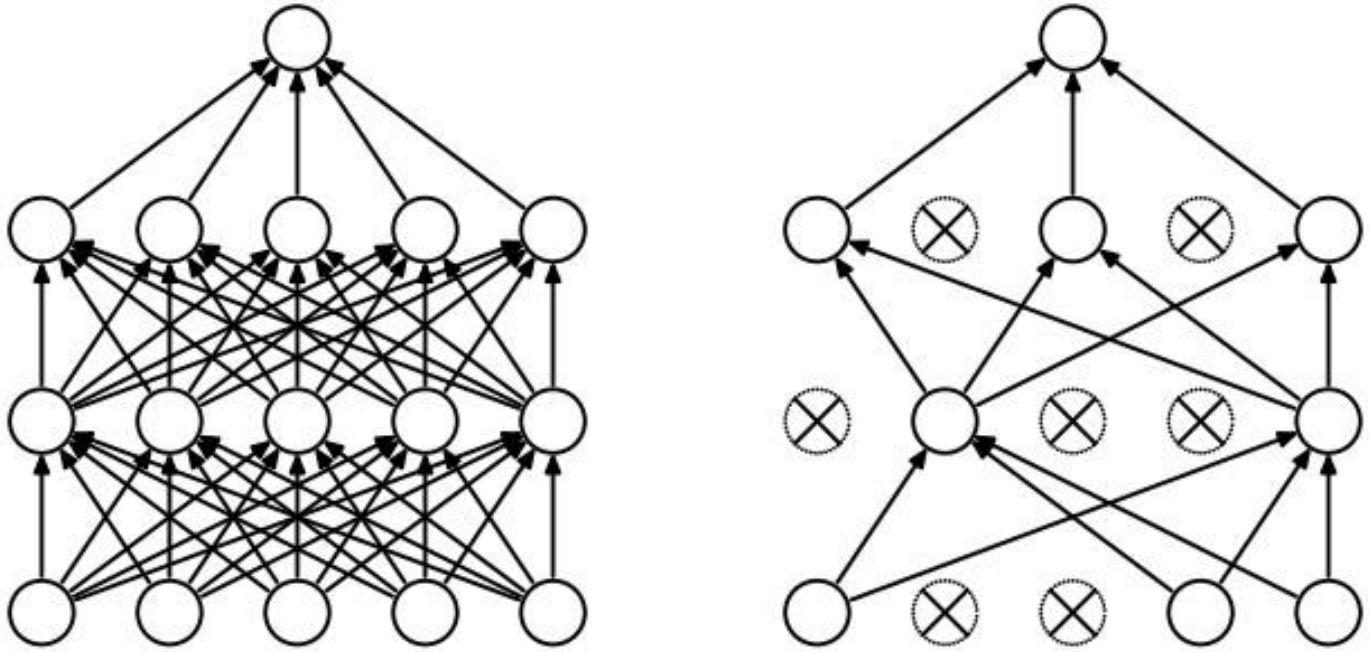


DROPOUT LAYER

Regularization technique.

Active only during training.

With some probability set output of unit to zero.



(a) Standard Neural Net

(b) After applying dropout.

GRADIENT PROBLEM

Vanishing gradient : gradients are smaller in every next layer

Exploding gradient : gradients are larger in every next layer

when backpropagating error

Result:

Unable to learn deeper model(lower layers)

Why?

Weights and activation functions squeeze gradients.

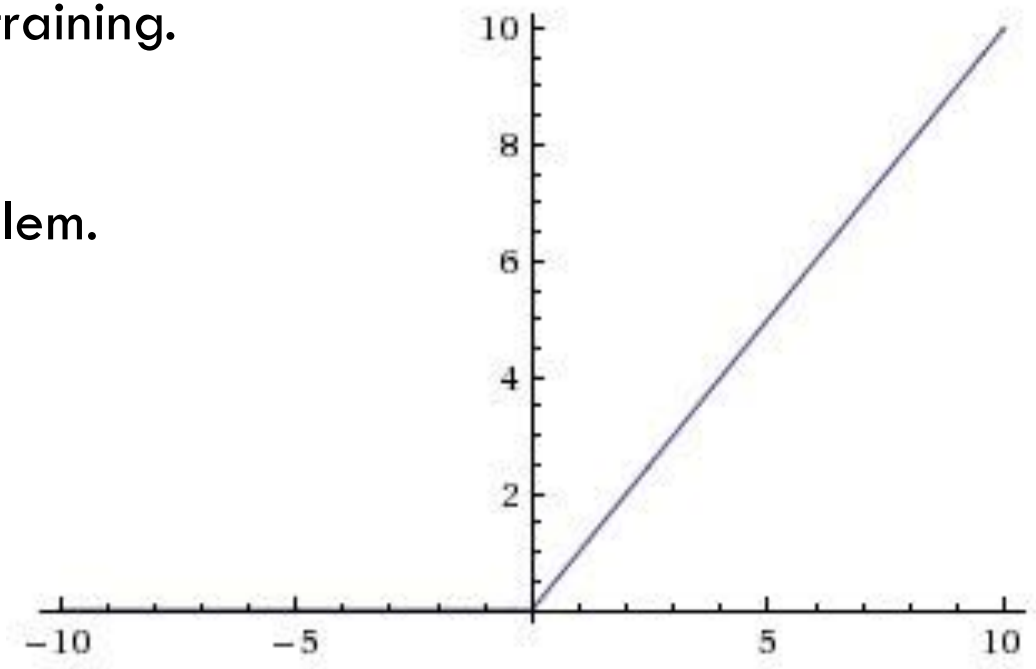
Understanding the difficulty of training deep feedforward neural networks [X.Glorot, 2010]

WHY NOW? SOLVING GP #2

Rectified Linear Unit(ReLU) as activation function.

Intelligent Initialization of Weights at beginning of training.

It doesn't solve the problem, It just minified the problem.



ReLU: f(x) = max(0,x)

HOW TO INITIALIZE WEIGHTS?

1. Random uniform from [-e, e]
2. Gaussian distribution
3. Xavier initialization
4. Pretraining with RBM models

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}} \right]$$

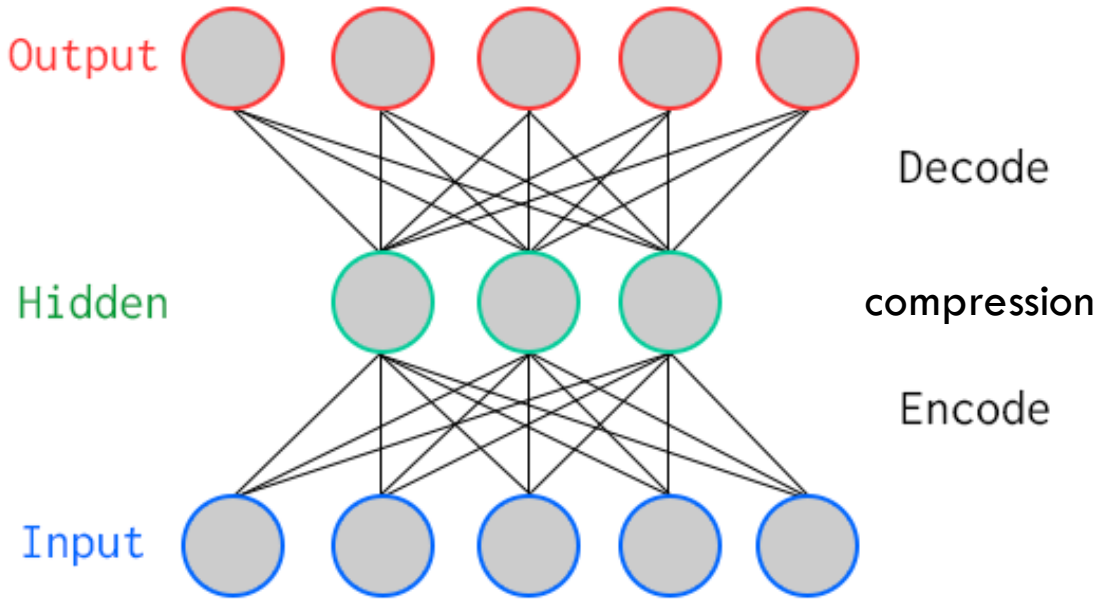
AUTOENCODER

Non-Linear dimensionality reduction.

Encoder and Decoder part.

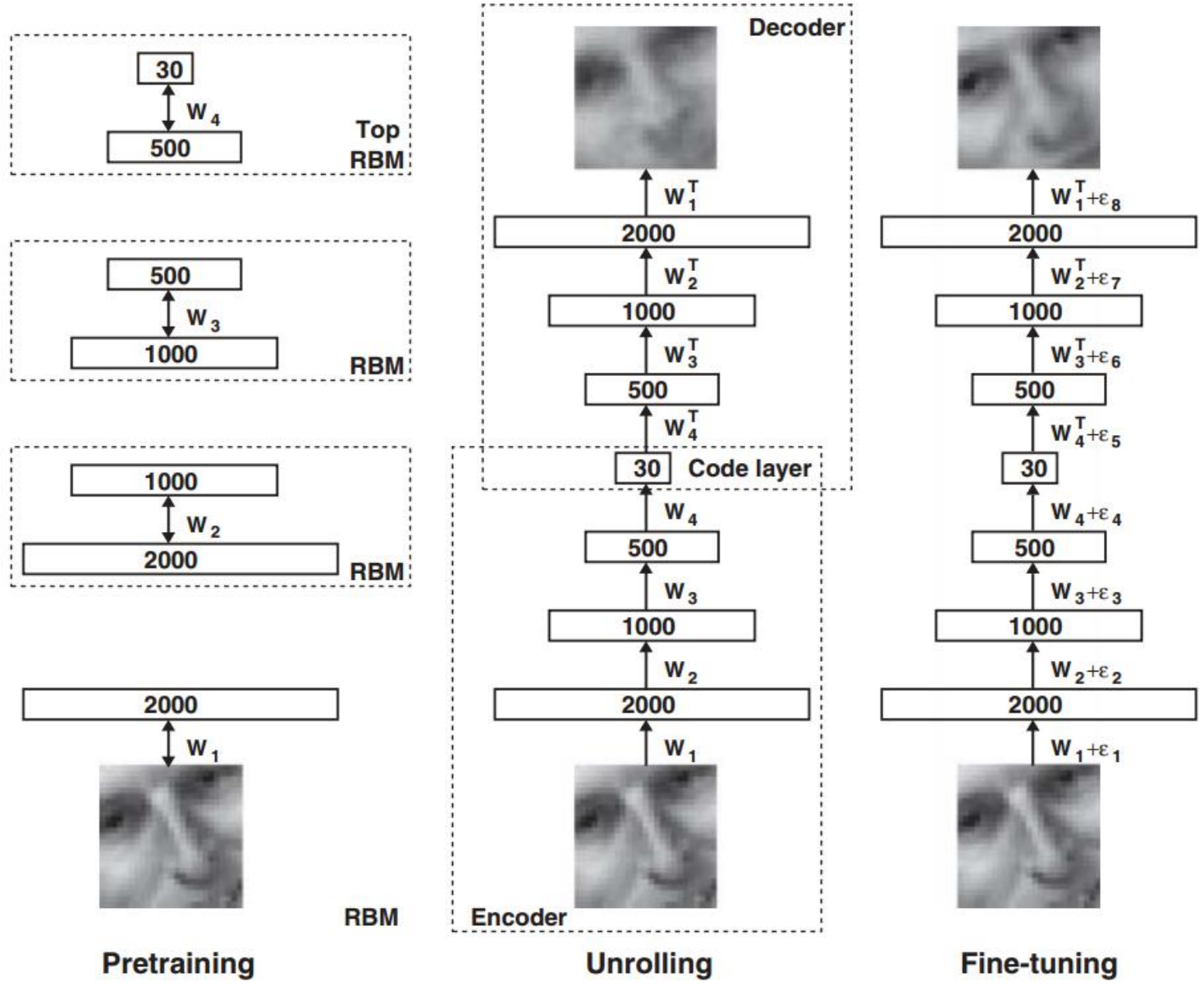
After training throw away Decoder part.

It can work better than PCA.



Training ends often in local optimum...

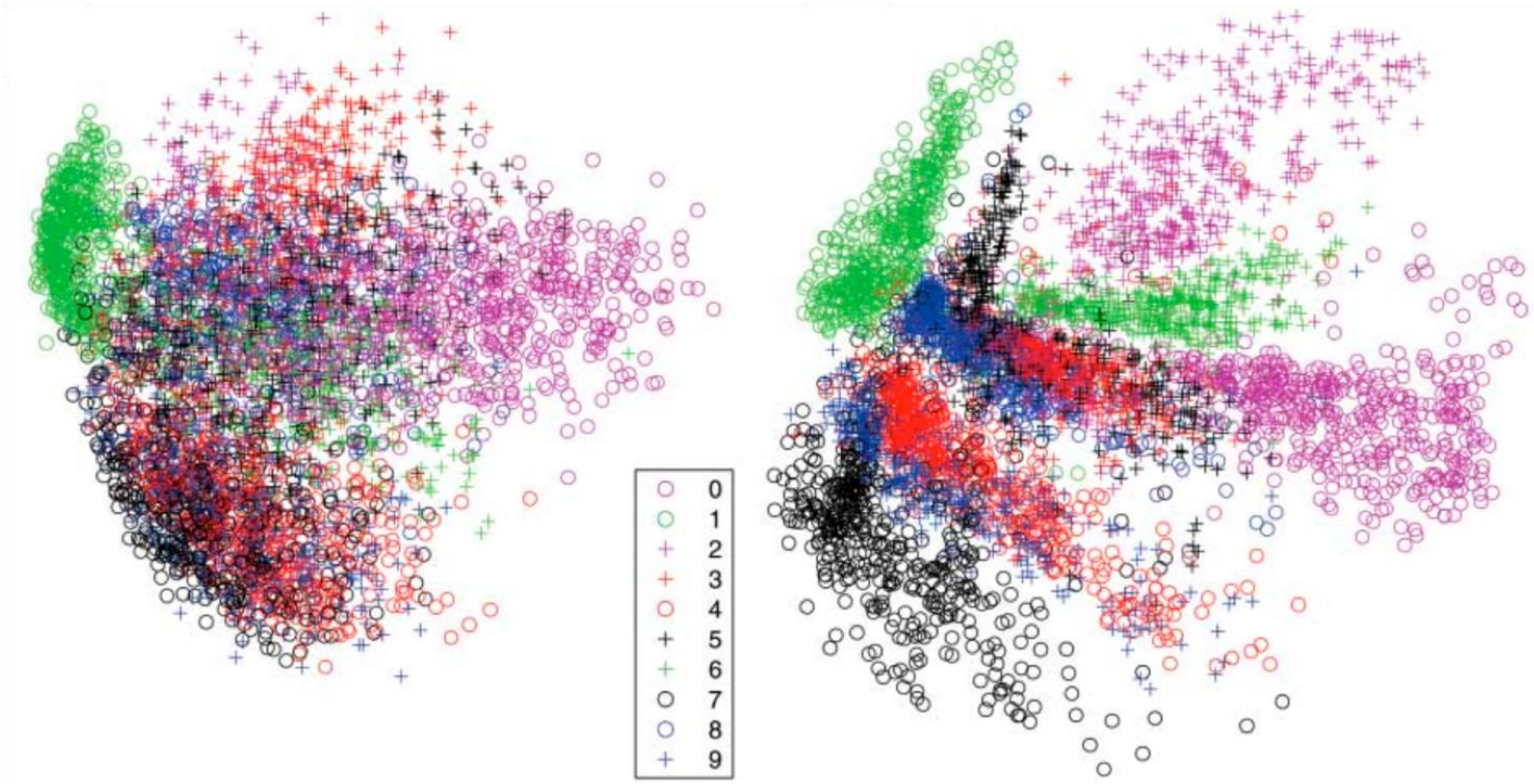
AUTOENCODER PRETRAINED BY RBM



https://github.com/Cospel/rbm-ae-tf

Reducing the Dimensionality of Data with Neural Networks[Science, 2006, Hinton]

AUTOENCODER VS PCA

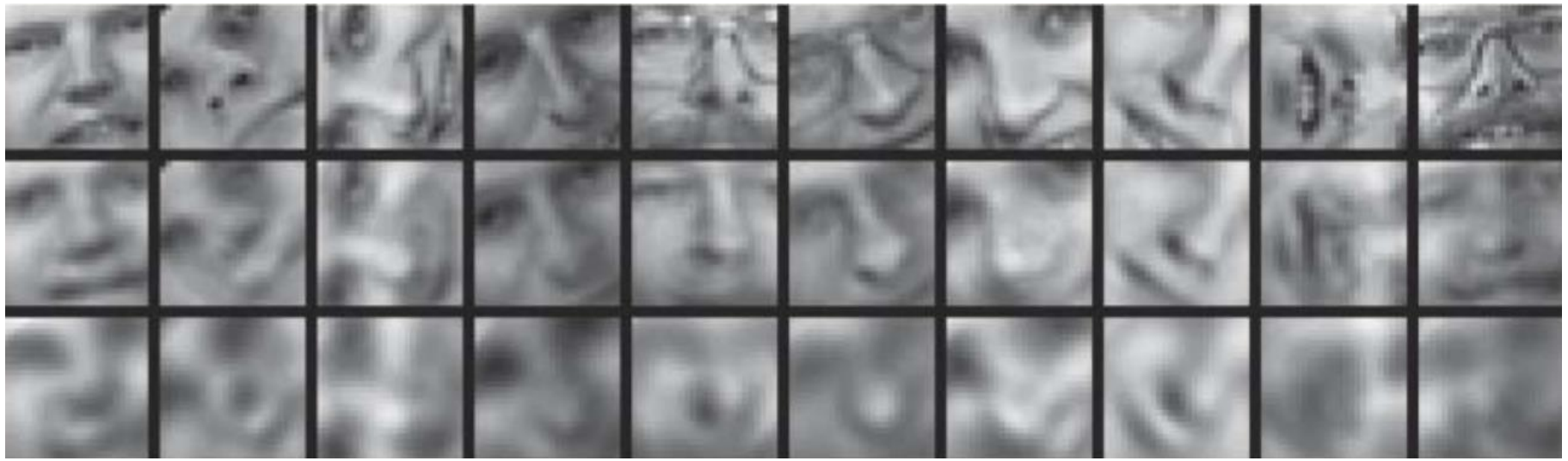


AUTOENCODER VS PCA

Original Input

Autoencoder

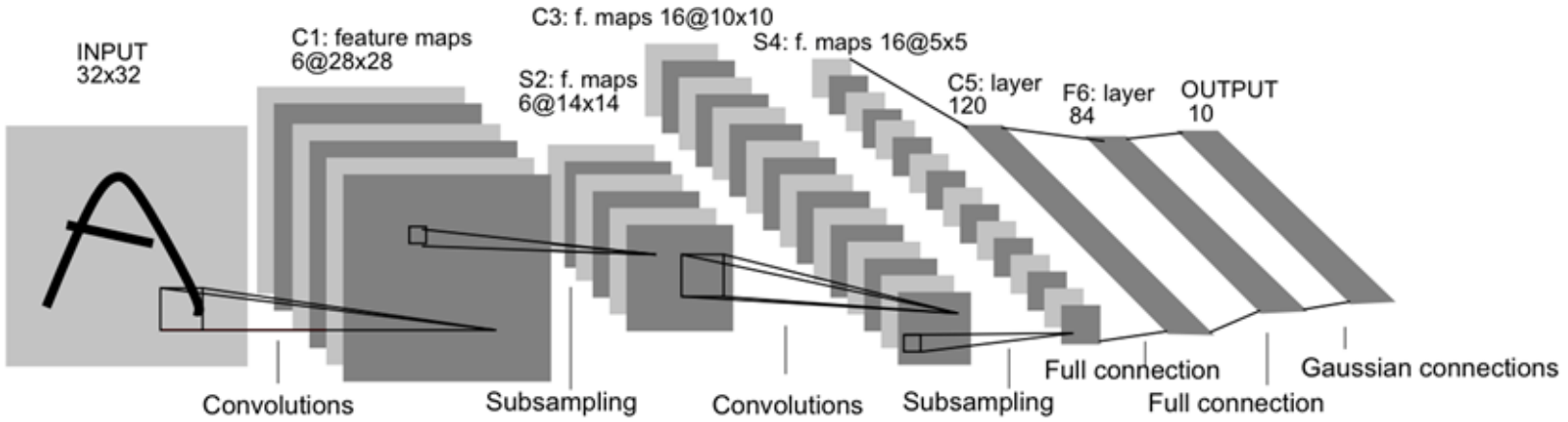
PCA



CONVOLUTIONAL NN

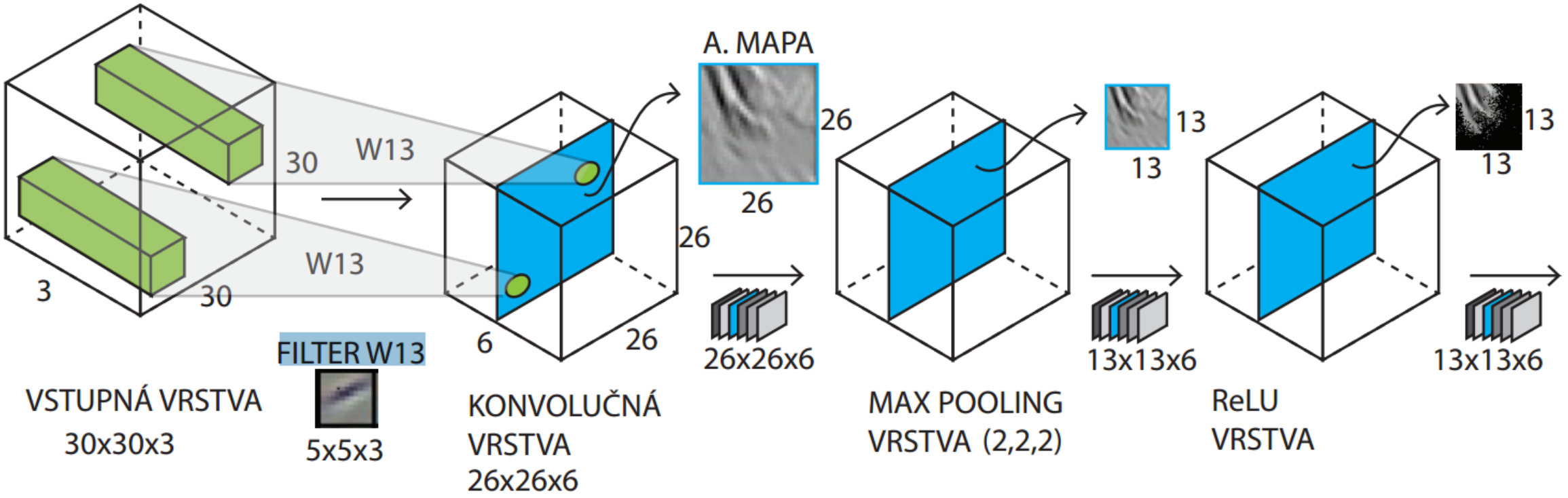
Stack of Convolution, Pooling, ReLU, Fully Connected Layers.
State of the art in computer vision.

Convolutional Layer: Weights Sharing, Local Connectivity
It is impractical to connect neurons to all neurons in the previous volume.

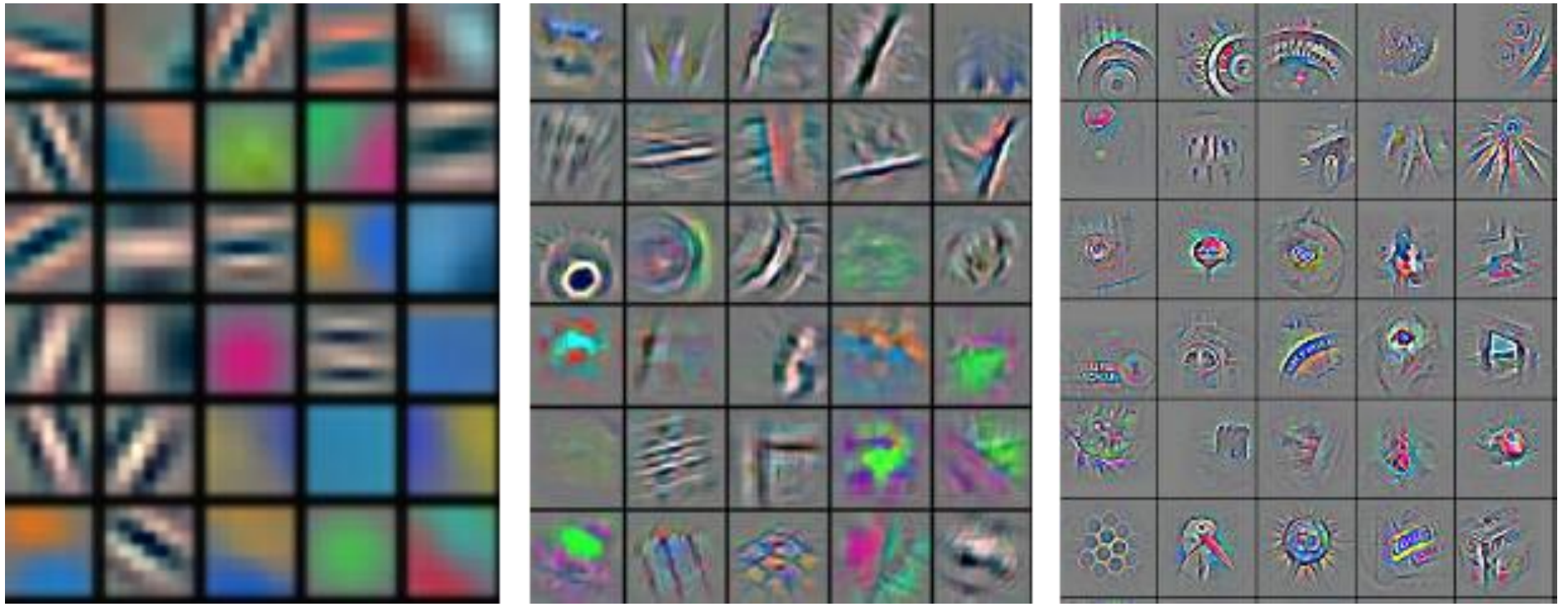


An early (Le-Net5) Convolutional Neural Network design, LeNet-5, used for recognition of digits

INPUT, CONV, POOLING, RELU LAYERS



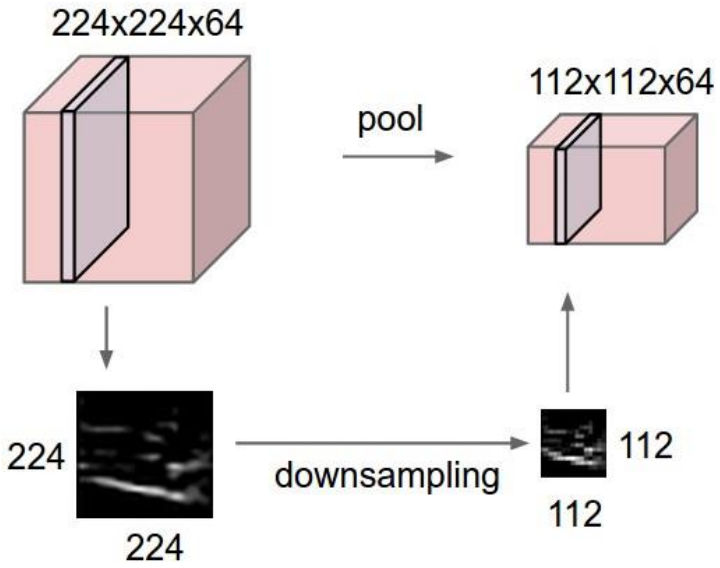
LEARNED CNN FEATURES

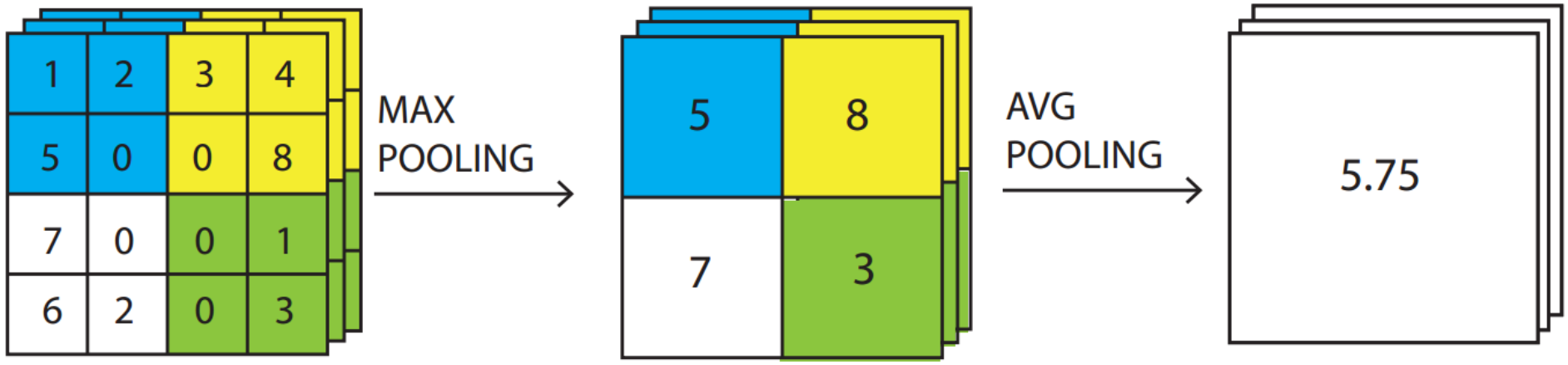


POOLING LAYER

Subsampling the image.

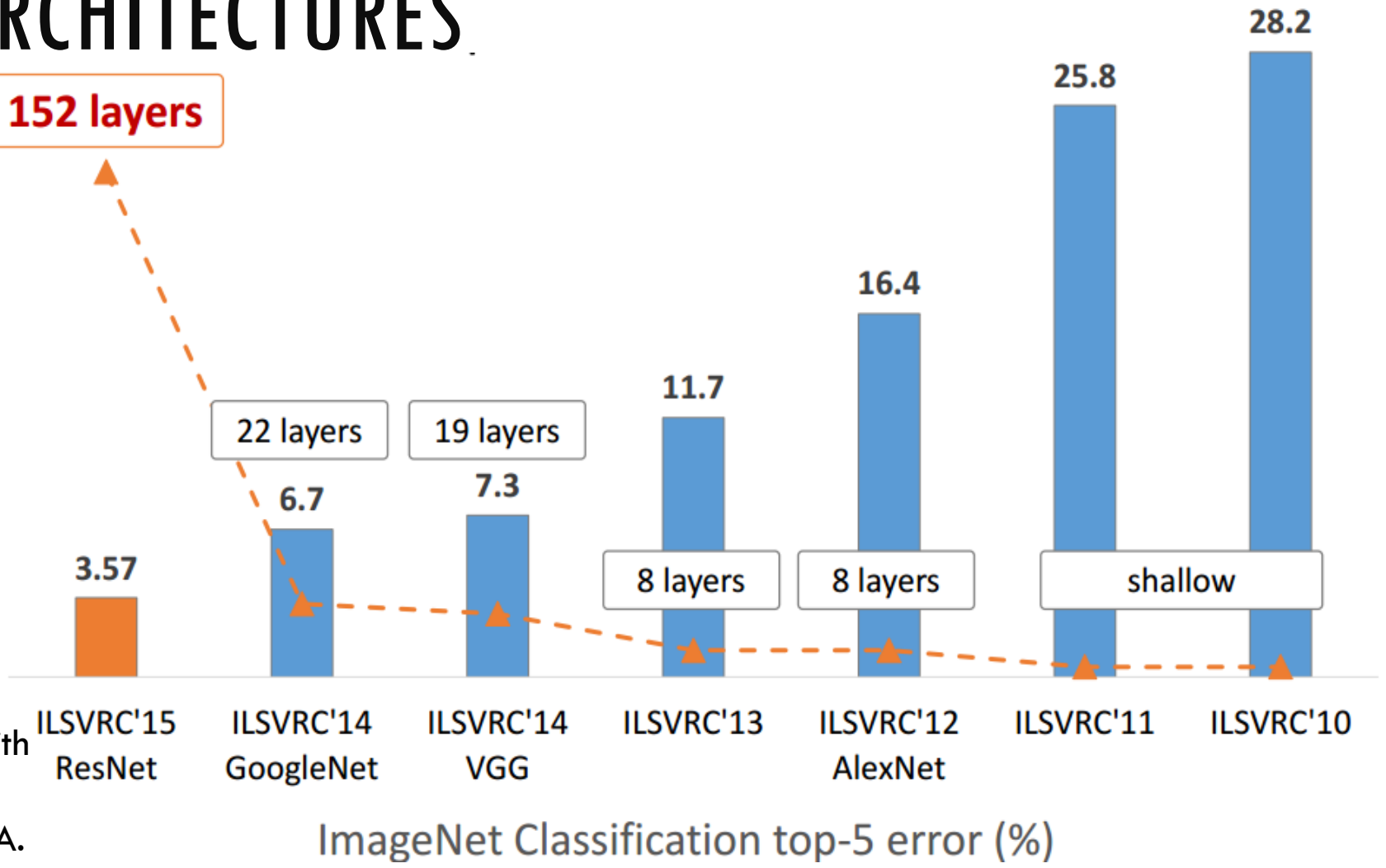
Smaller outputs = faster learning





MANY CNN ARCHITECTURES

1. AlexNet
2. VGG
3. ResNet
4. SqueezeNet
5. GoogleNet
6. ...



[ImageNet Classification with Deep Convolutional Neural Networks, 2012 G.Hinton, A. Krizhevsky]

TRANSFER LEARNING

1. Use existing Weights or entire NN to finetune on new(similar domain) data
2. Use CNN descriptors for algorithms as KNN, SVM, …

Many pretrained models(weights) are available to download on github:

VGG with face descriptors

Models for places

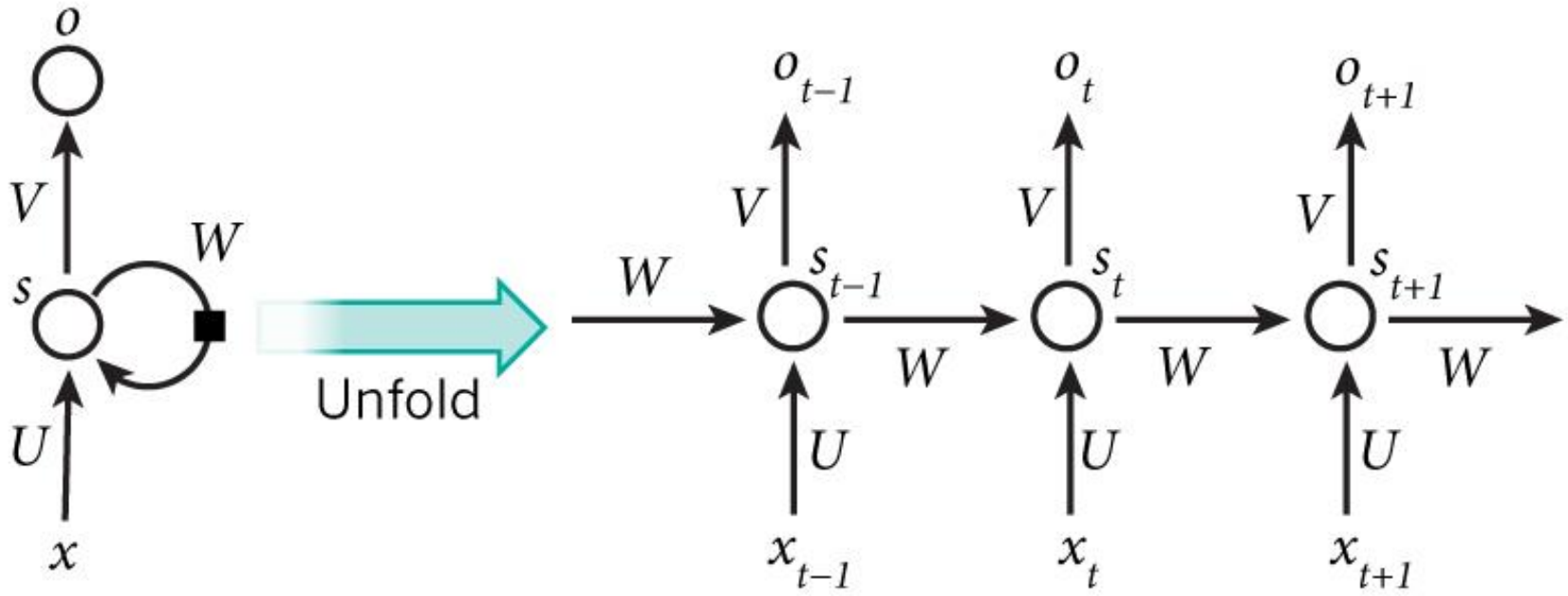
….

RECURRENT NN

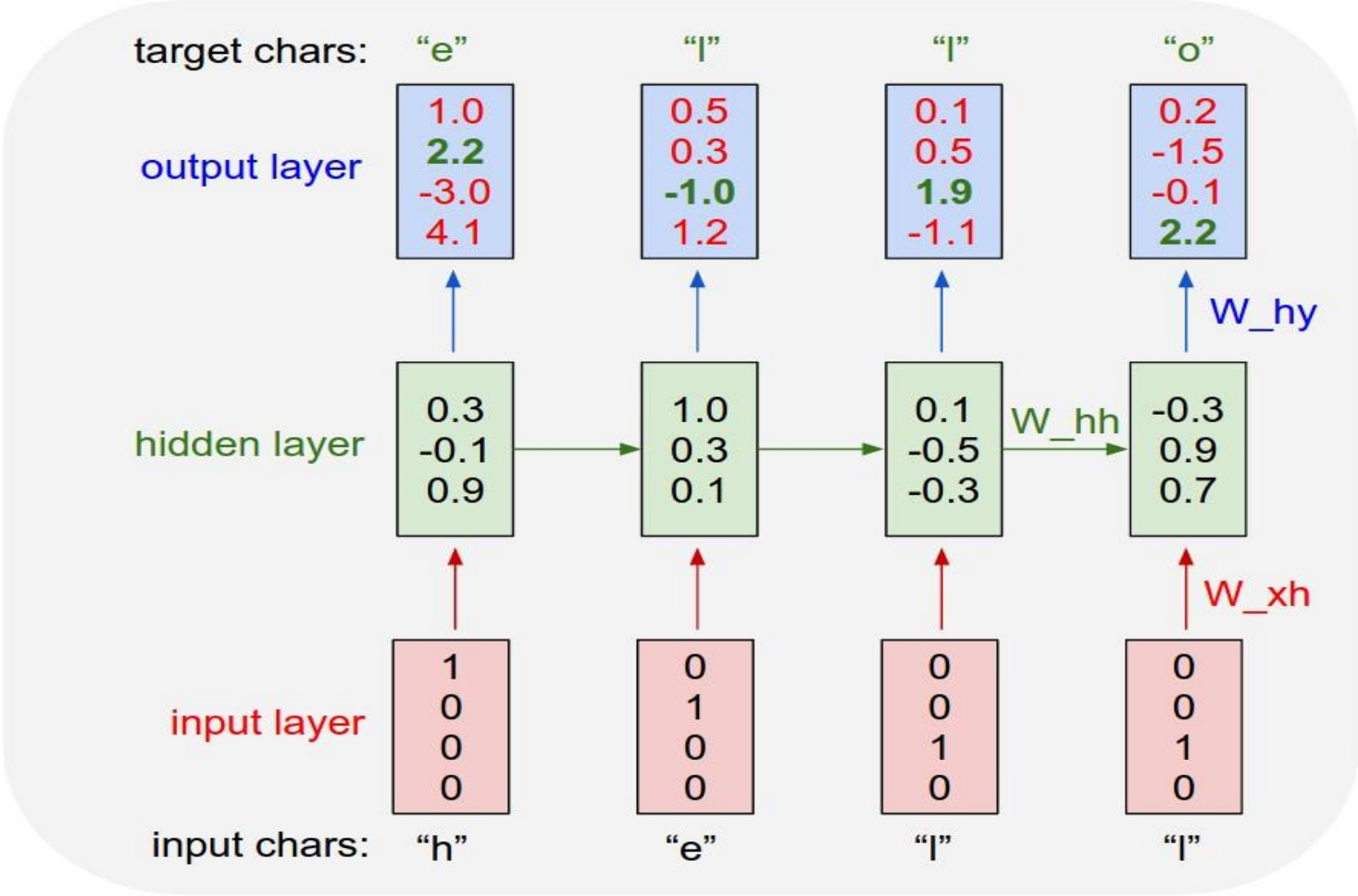
Good for timeseries data, nlp, video sequences, ...

Backpropagation through time …

It has internal hidden state.(memory for sequence)



RECURRENT NN



SIMPLE IMPLEMENTATION

```
class RNN:
  def step(self, x):
    # update the hidden state
    self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))
    # compute the output vector
    y = np.dot(self.W_hy, self.h)
    return y

rnn1 = RNN()
y = rnn1.step(x1_1)
y = rnn1.step(x1_2)
```

FRAMEWORKS

KERAS FEEDFORWARD NET

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD

model = Sequential()
model.add(Dense(64, input_dim=20, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
model.fit(X_train, Y_train, batch_size=16, nb_epoch=10)
result = model.predict(X_test)
```

KERAS RECURRENT NET

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.layers import LSTM

model = Sequential()
model.add(LSTM(32, return_sequences=True, input_shape=(timesteps, data_dim)))
model.add(LSTM(32, return_sequences=True))
model.add(LSTM(32))  # return a single vector of dimension 32
model.add(Dense(10, activation='softmax'))
```

KERAS CONVOLUTIONAL NETWORK

```
image_model = Sequential()
image_model.add(Convolution2D(32, 3, 3, border_mode='valid', input_shape=(3, 100, 100)))
image_model.add(Activation('relu'))
image_model.add(Convolution2D(32, 3, 3))
image_model.add(Activation('relu'))
image_model.add(MaxPooling2D(pool_size=(2, 2)))

image_model.add(Convolution2D(64, 3, 3, border_mode='valid'))
image_model.add(Activation('relu'))
image_model.add(Convolution2D(64, 3, 3))
image_model.add(Activation('relu'))
image_model.add(MaxPooling2D(pool_size=(2, 2)))

image_model.add(Flatten())
image_model.add(Dense(128))

# let's load the weights from a save file.
image_model.load_weights('weight_file.h5')
```

THANK YOU…