

# Lecture #4 (nn)

F5611 Machine Learning for Astronomers  
by Martin Topinka

<https://github.com/toastmaker/f5611-ML4A>

**“Deep learning (neural networks) is like teenage sex: everyone talks about it, nobody really knows how to do it, everyone thinks everyone else is doing it, so everyone claims they are doing it....”**

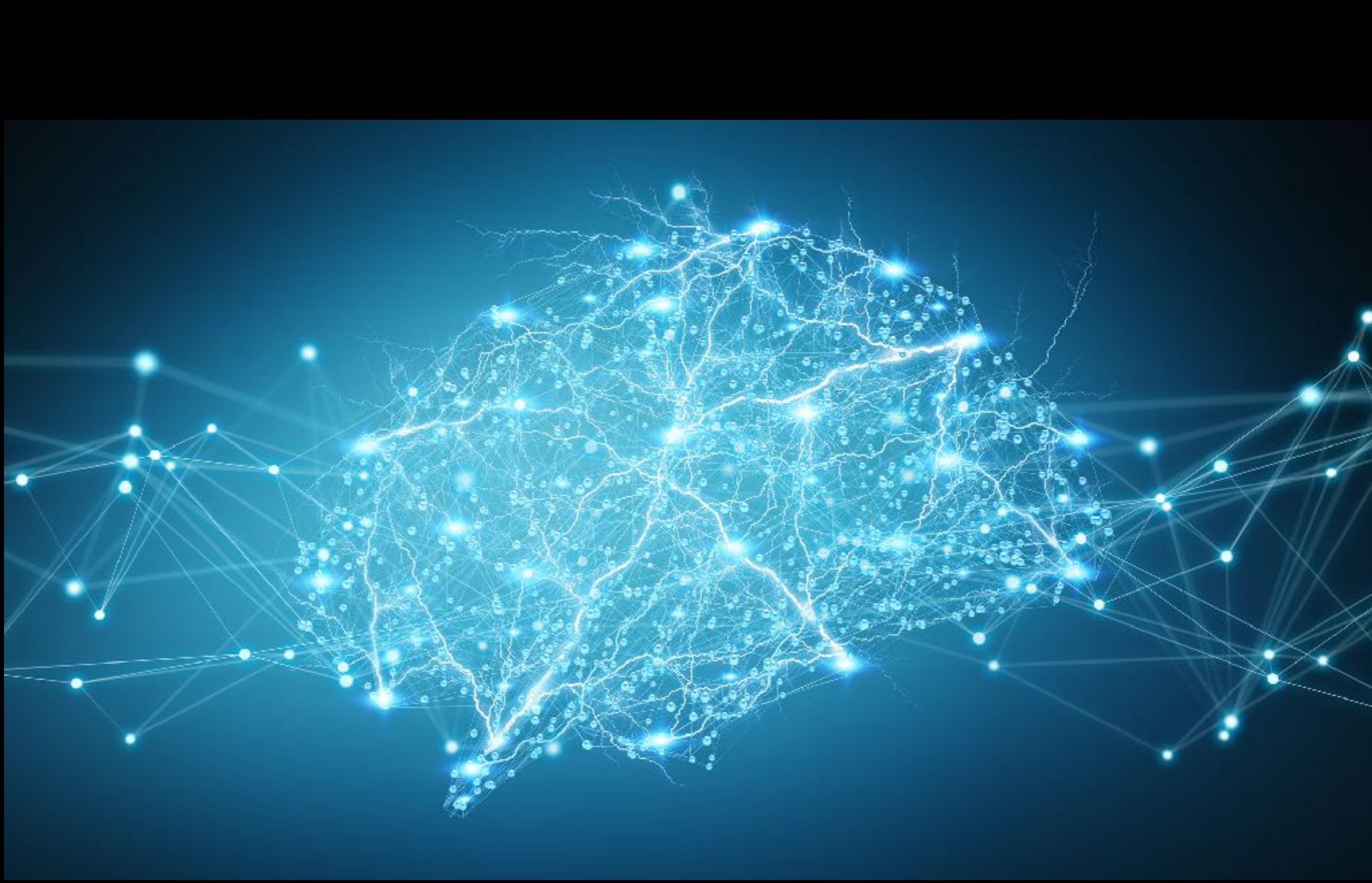
*– a ML expert*

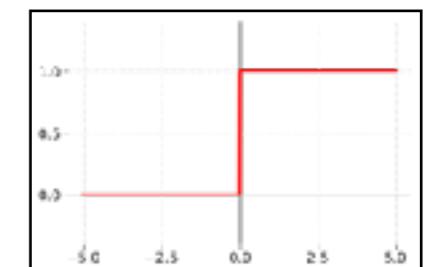
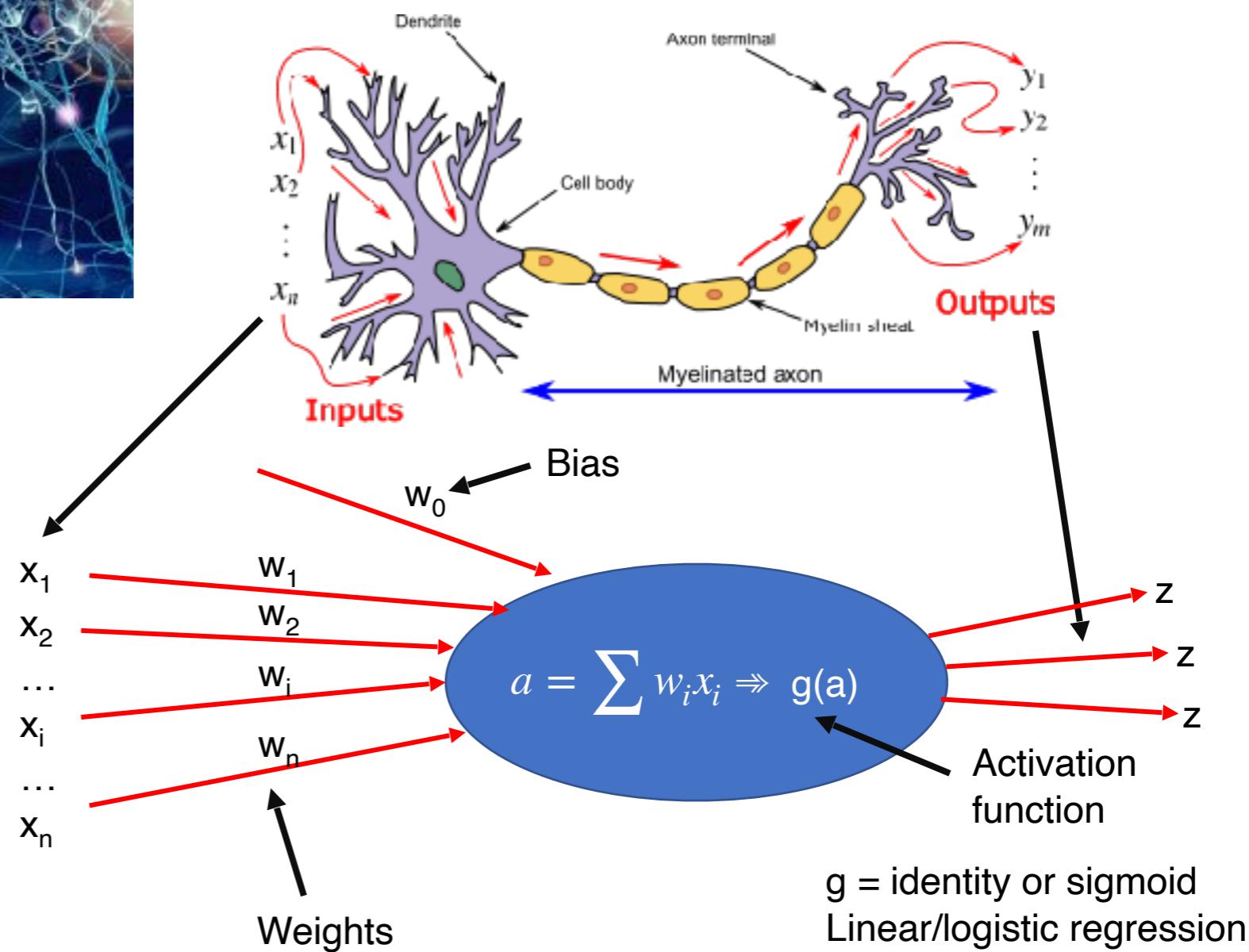
**Buzz words: machine learning, deep learning, AI, neural network, ...**

**Similar to other buzz words: Bayesian, gravitational waves, exoplanets, dark matter...**

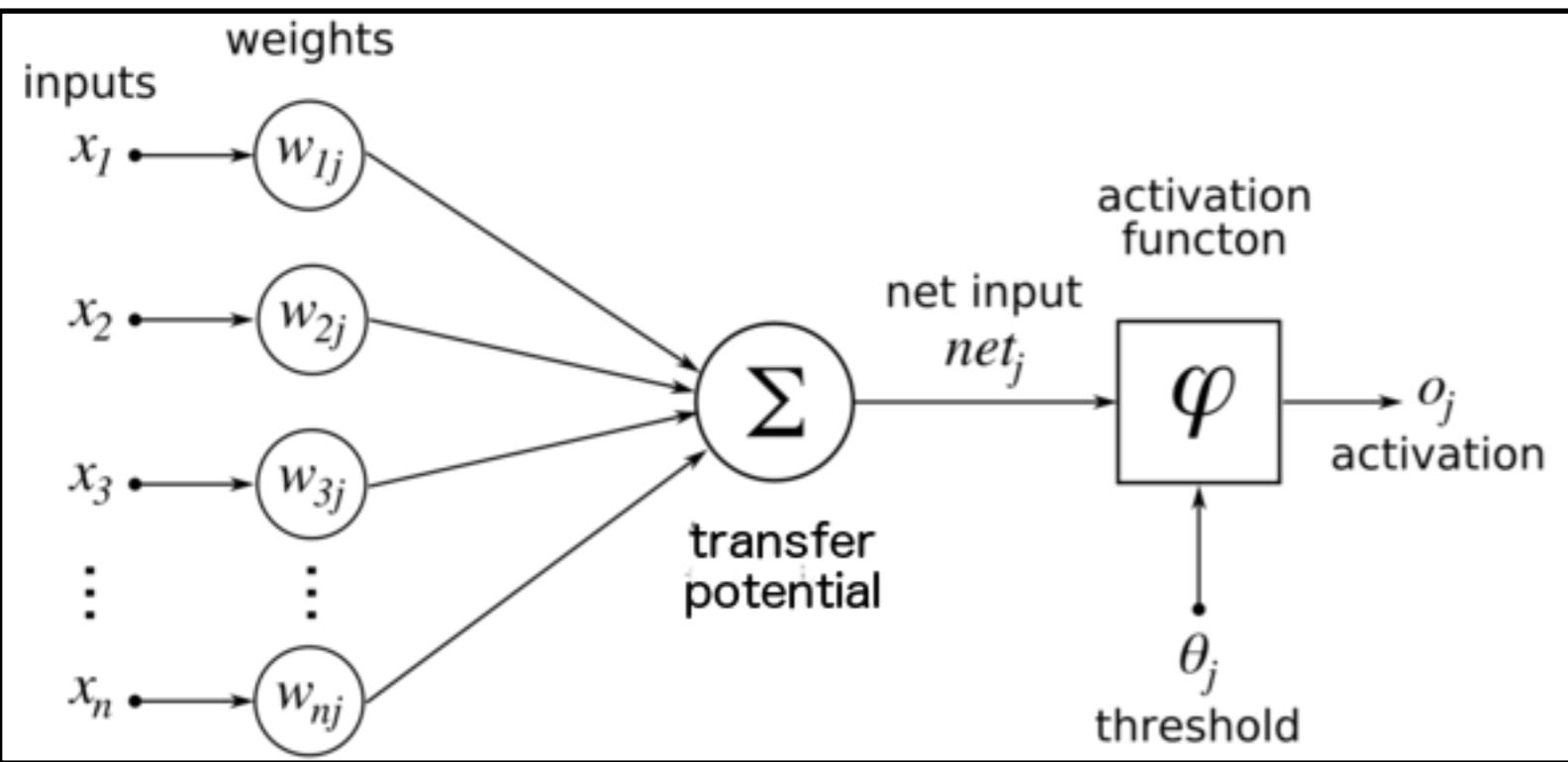
According to a report by MMC Ventures, close to 40% of European companies claiming to use AI don't actually use it





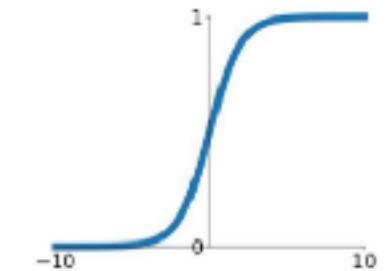


# Activation Functions



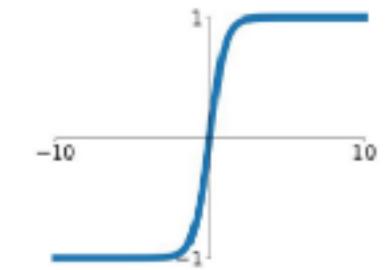
## Sigmoid

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



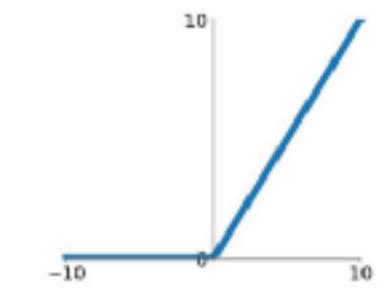
## tanh

$$\tanh(x)$$

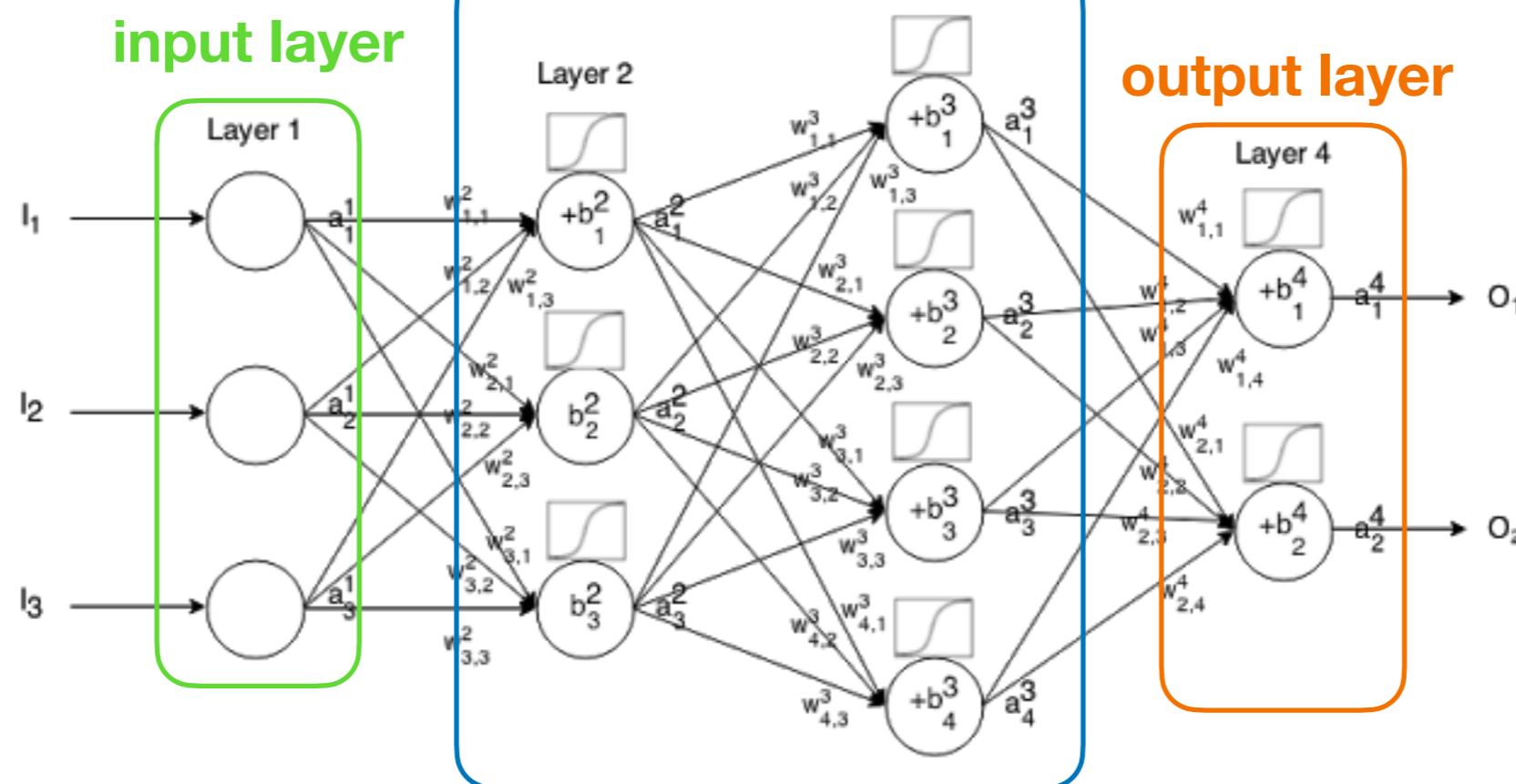


## ReLU

$$\max(0, x)$$



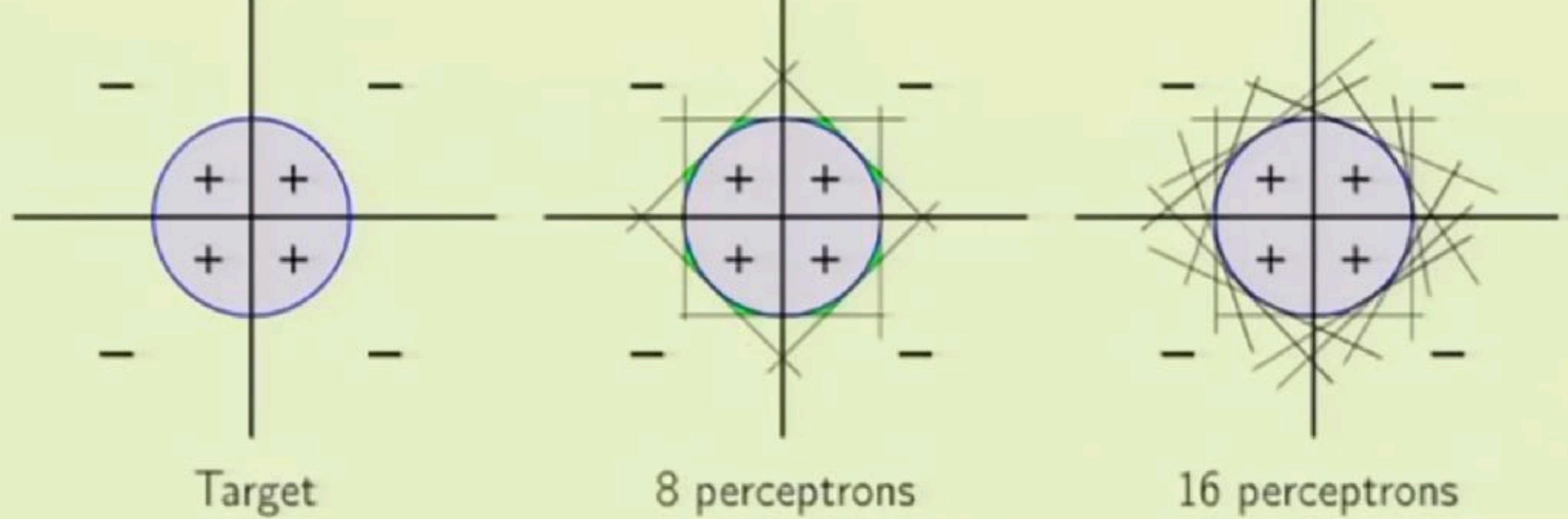
## hidden layers

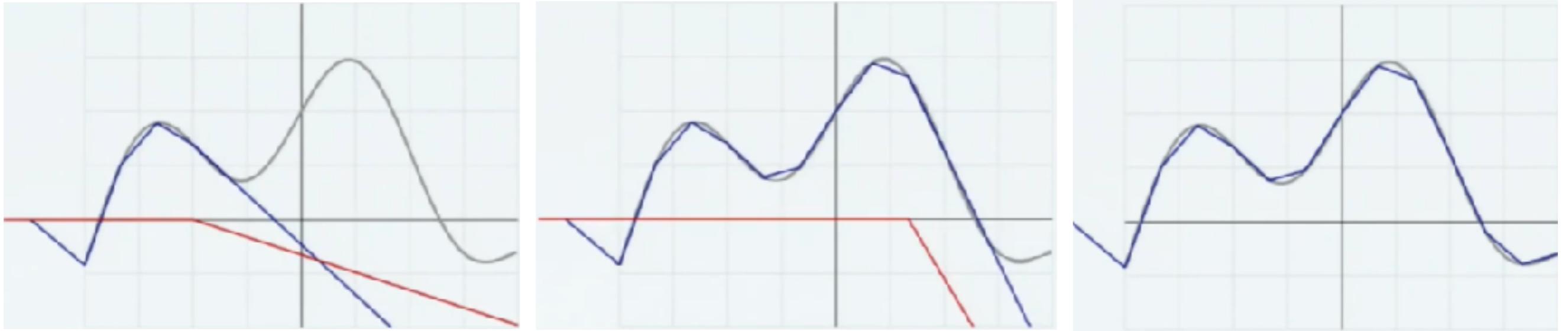


# Neural Network

Linear  
classifiers





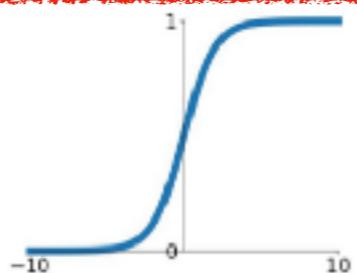


## Activation Functions

[0,1]  
"probability"  
final layer for  
classification

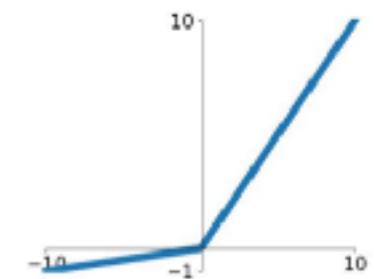
### Sigmoid

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



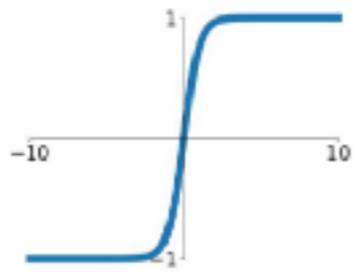
### Leaky ReLU

$$\max(0.1x, x)$$



### tanh

$$\tanh(x)$$

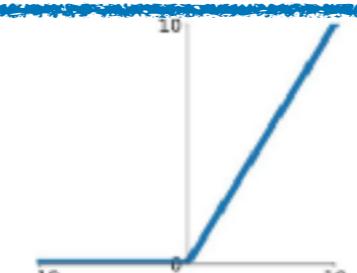


### Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

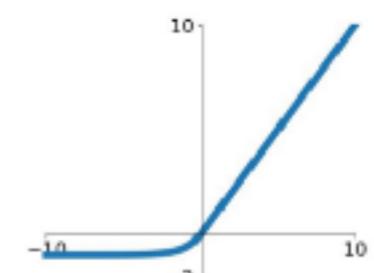
### ReLU

$$\max(0, x)$$



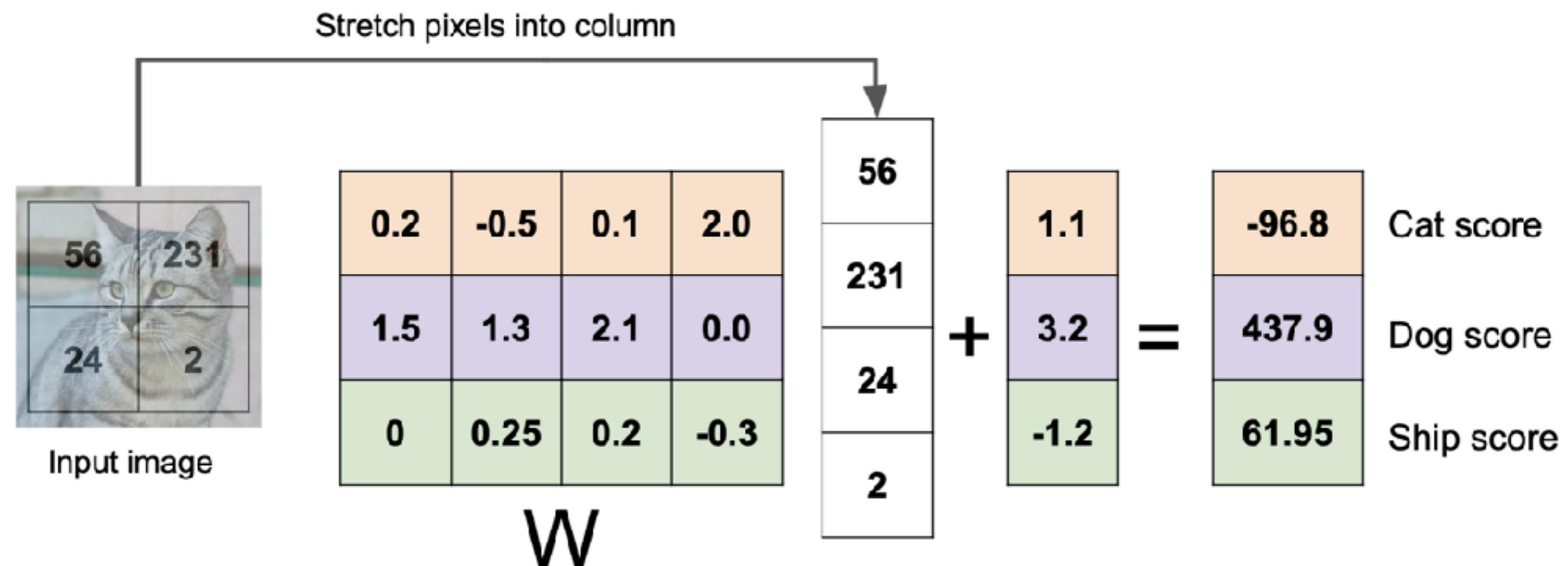
### ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

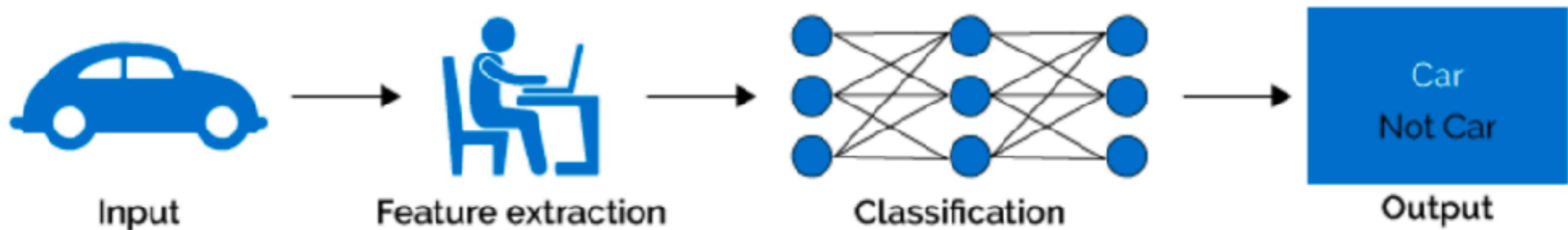


robust  
against  
"vanishing  
gradient"

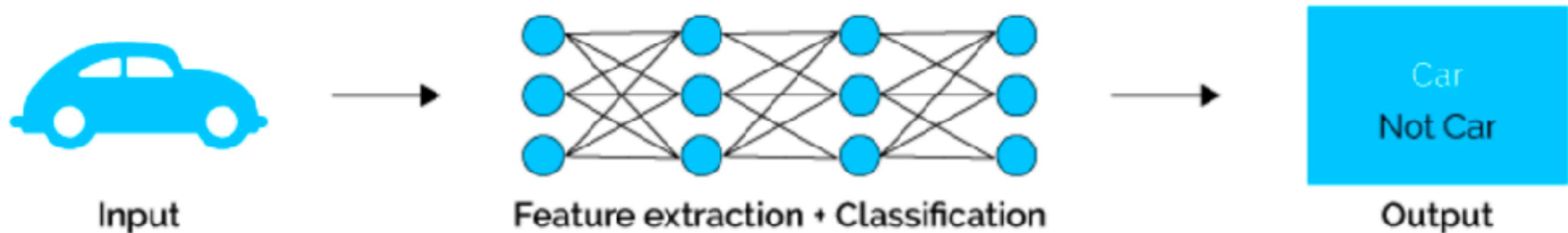
## Example with an image with 4 pixels, and 3 classes (**cat/dog/ship**)



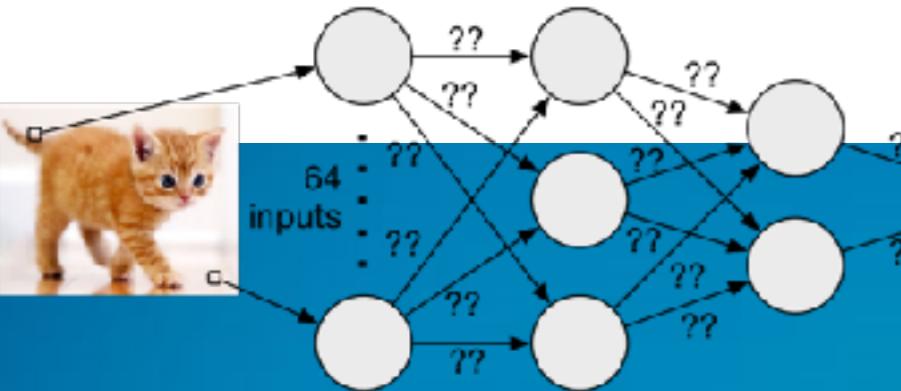
## Machine Learning



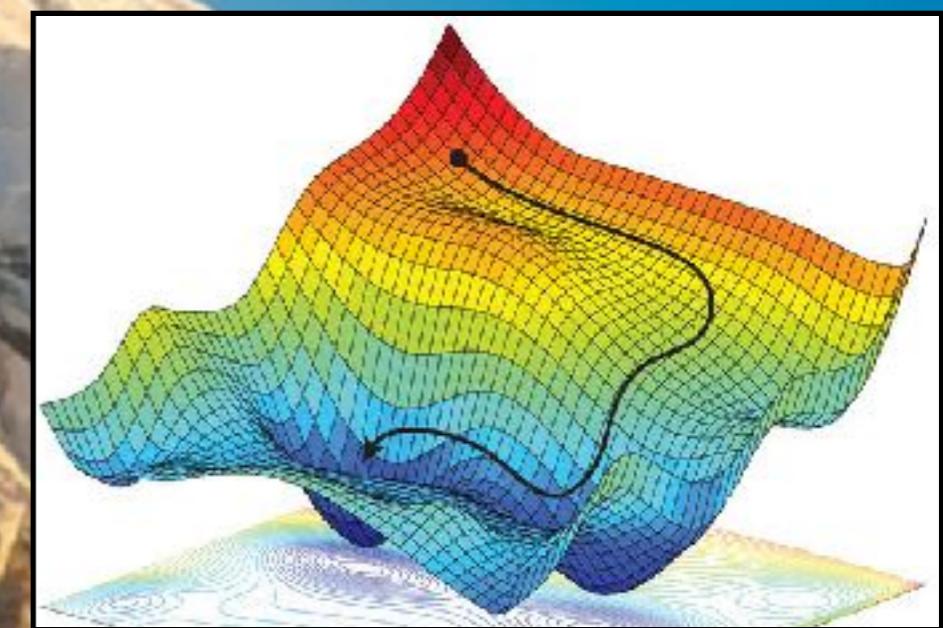
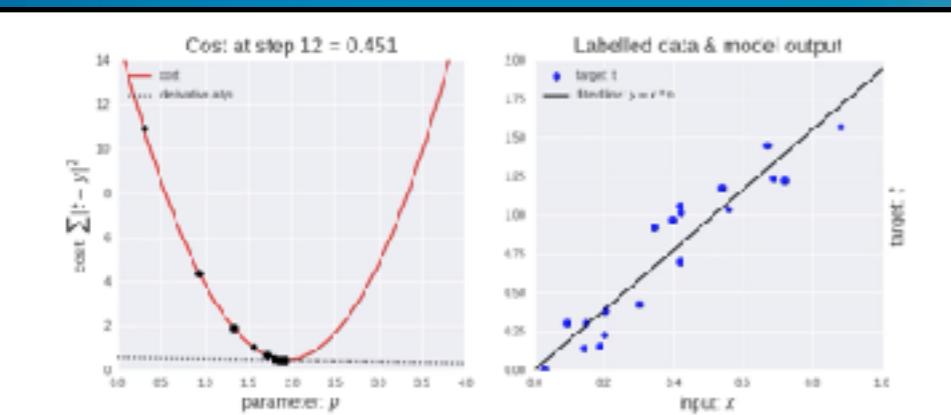
## Deep Learning



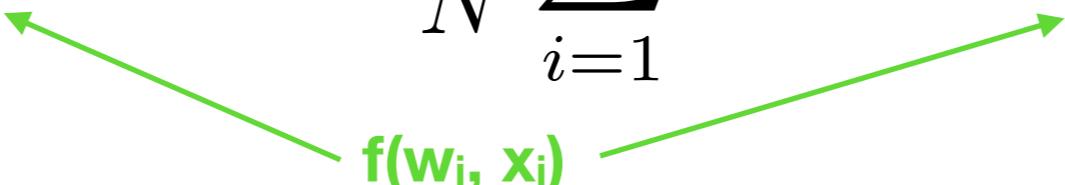
# (Stochastic) Gradient Descent



$$w = w - \alpha \nabla_w J(w)$$



$$\langle -\nabla_w L_i(f(x_i), y_i) \rangle = \frac{1}{N} \sum_{i=1}^N -\nabla_w L(f(x_i), y_i) = -\nabla_w L$$



### Advantages of SGD:

- cheaper CPU, less memory
- randomisation (lesser chance to get stuck in a shallow local minimum thanks to noise)

## Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W) + \lambda R(W)$$

Full sum expensive  
when N is large!

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Approximate sum  
using a **minibatch** of  
examples  
32 / 64 / 128 common

```
# Vanilla Minibatch Gradient Descent
```

```
while True:
```

```
    data_batch = sample_training_data(data, 256) # sample 256 examples
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
    weights += - step_size * weights_grad # perform parameter update
```

**Batch size** = #data points used to  
determine 1 gradient direction  
**Epoch** = # of loops over all data

# SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

# SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

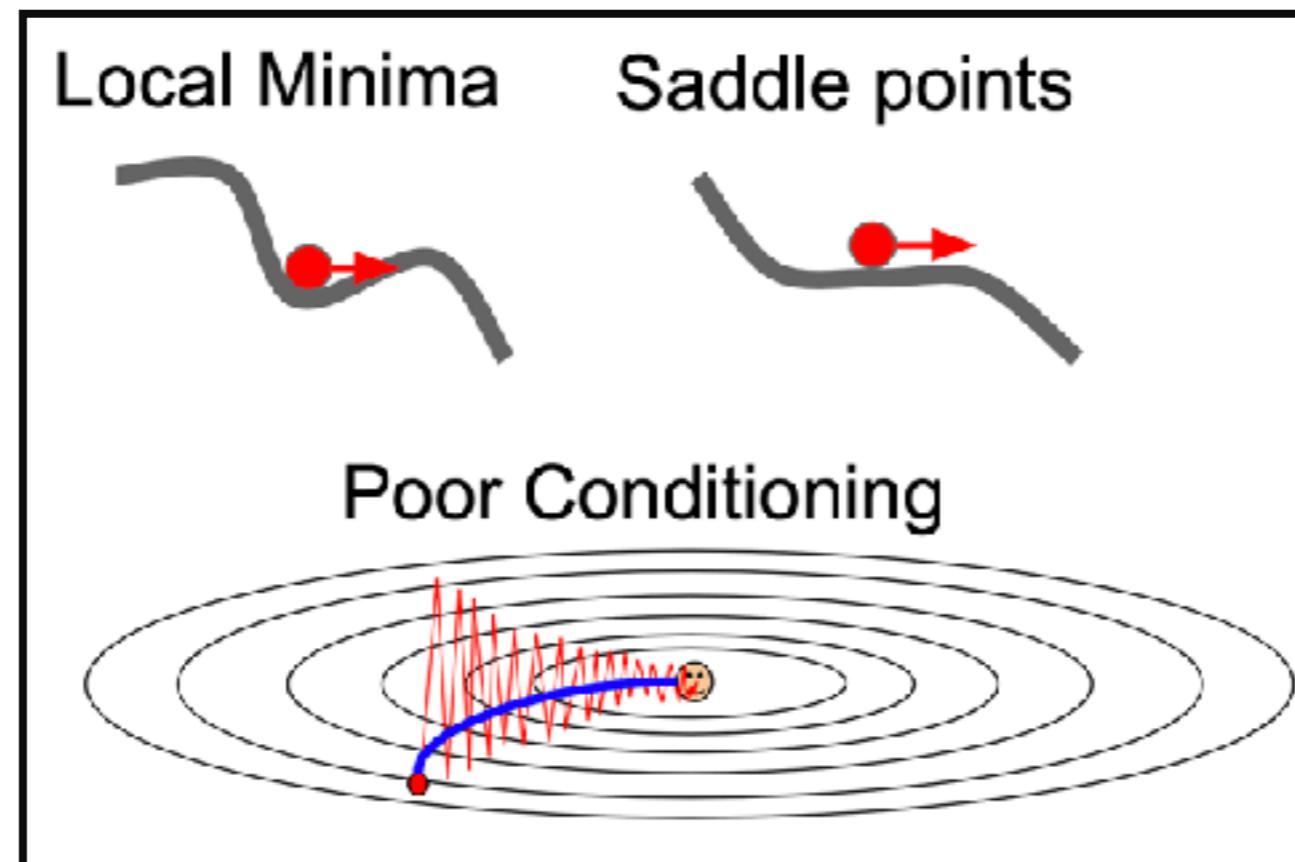
```
while True:
```

```
    dx = compute_gradient(x)  
    x += learning_rate * dx
```

```
vx = 0
```

```
while True:  
    dx = compute_gradient(x)  
    vx = rho * vx + dx  
    x += learning_rate * vx
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99



## AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



## RMSProp

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

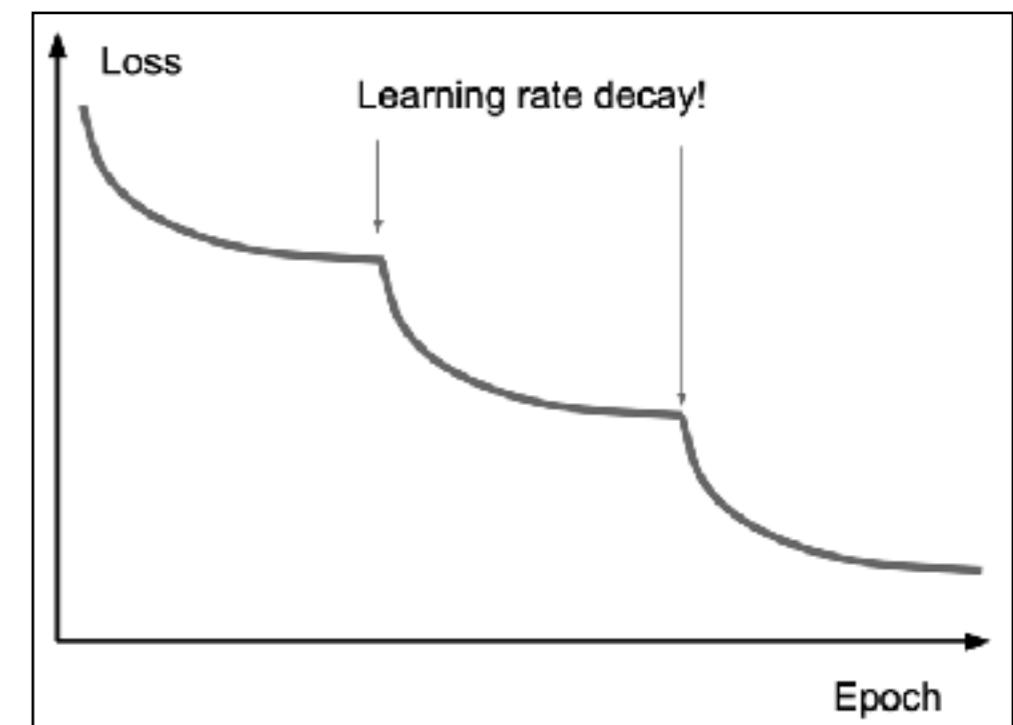
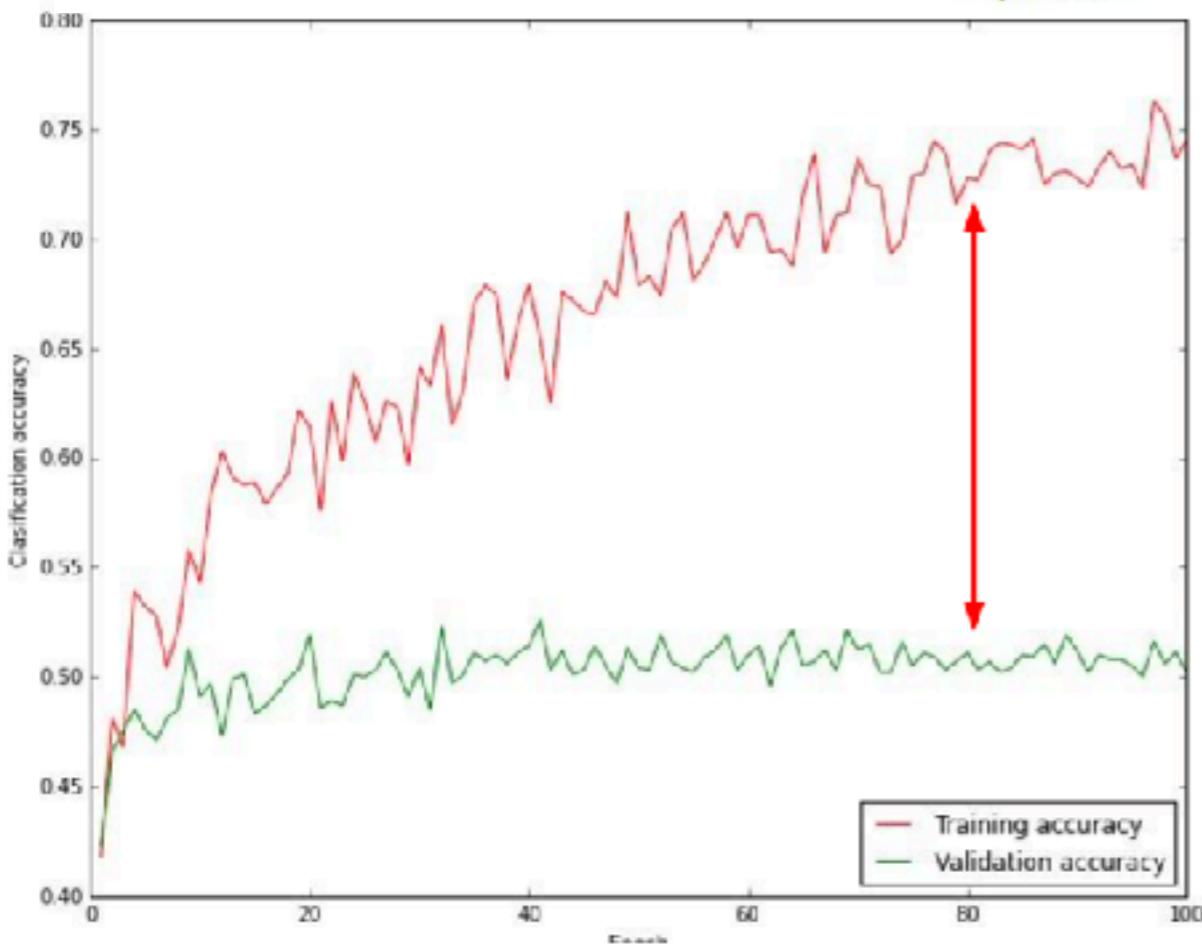
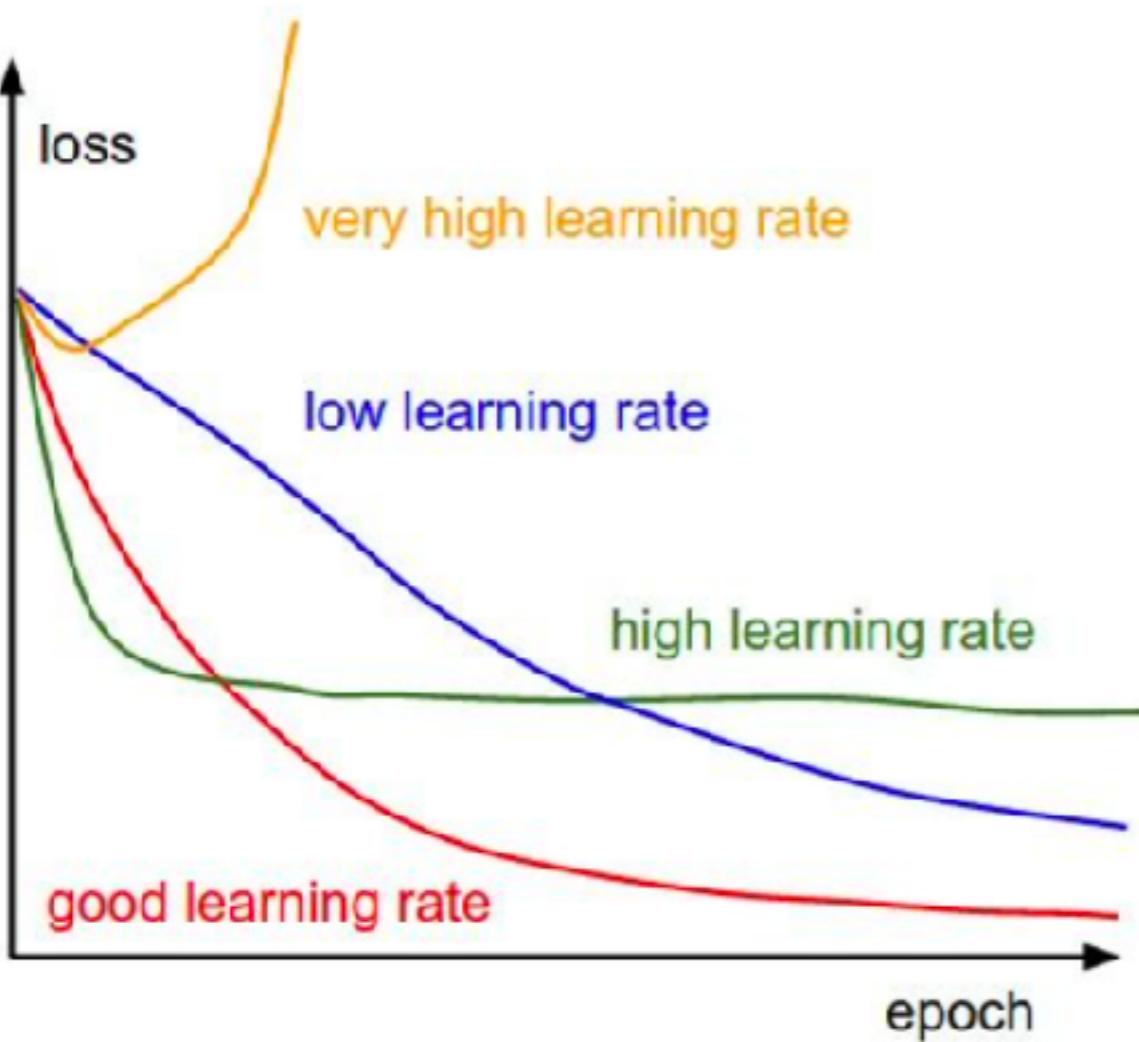
## Adam (best?)

```
first_moment = 0
second_moment = 0
for t in range(num_iterations):
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7)
```

Momentum

Bias correction

AdaGrad / RMSProp



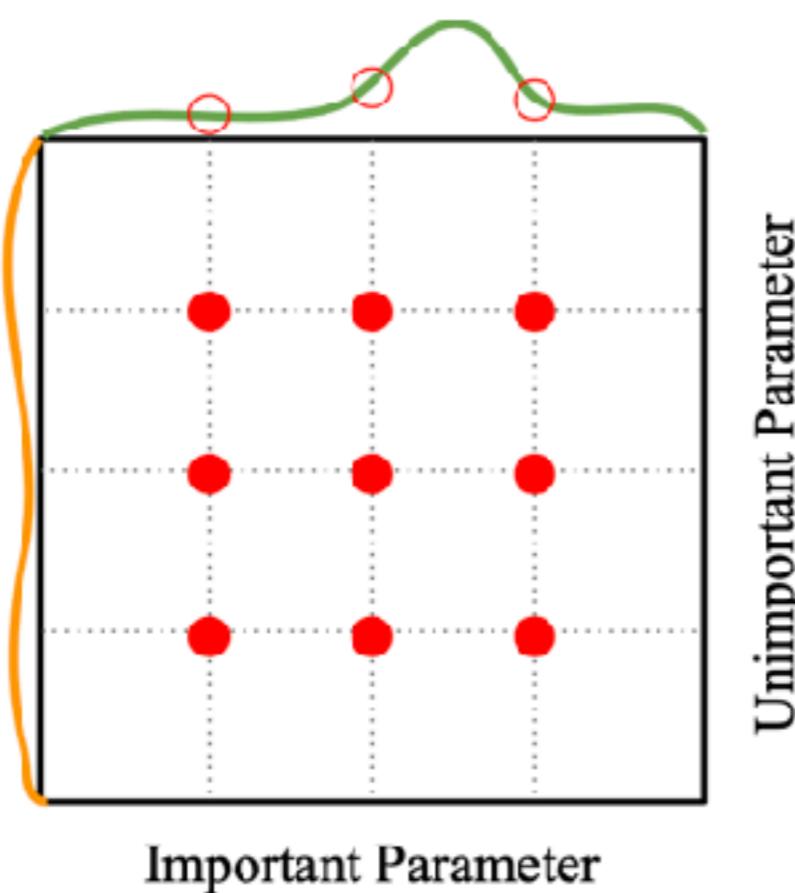
**big gap = overfitting**  
**=> increase regularization strength?**

**no gap**  
**=> increase model capacity?**

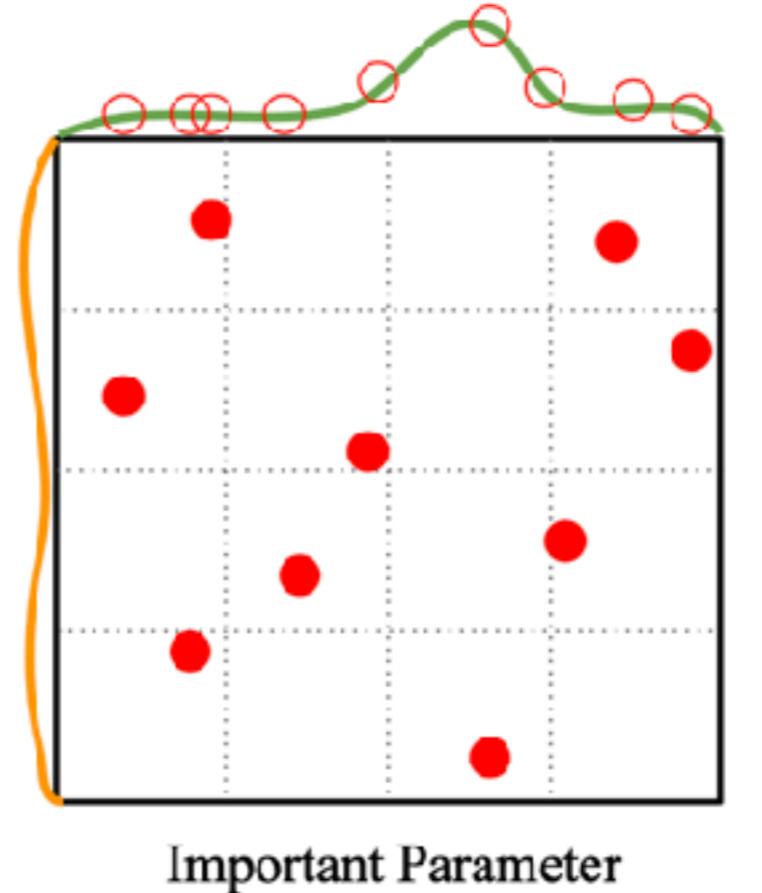
# Random Search vs. Grid Search

*Random Search for  
Hyper-Parameter Optimization*  
Bergstra and Bengio, 2012

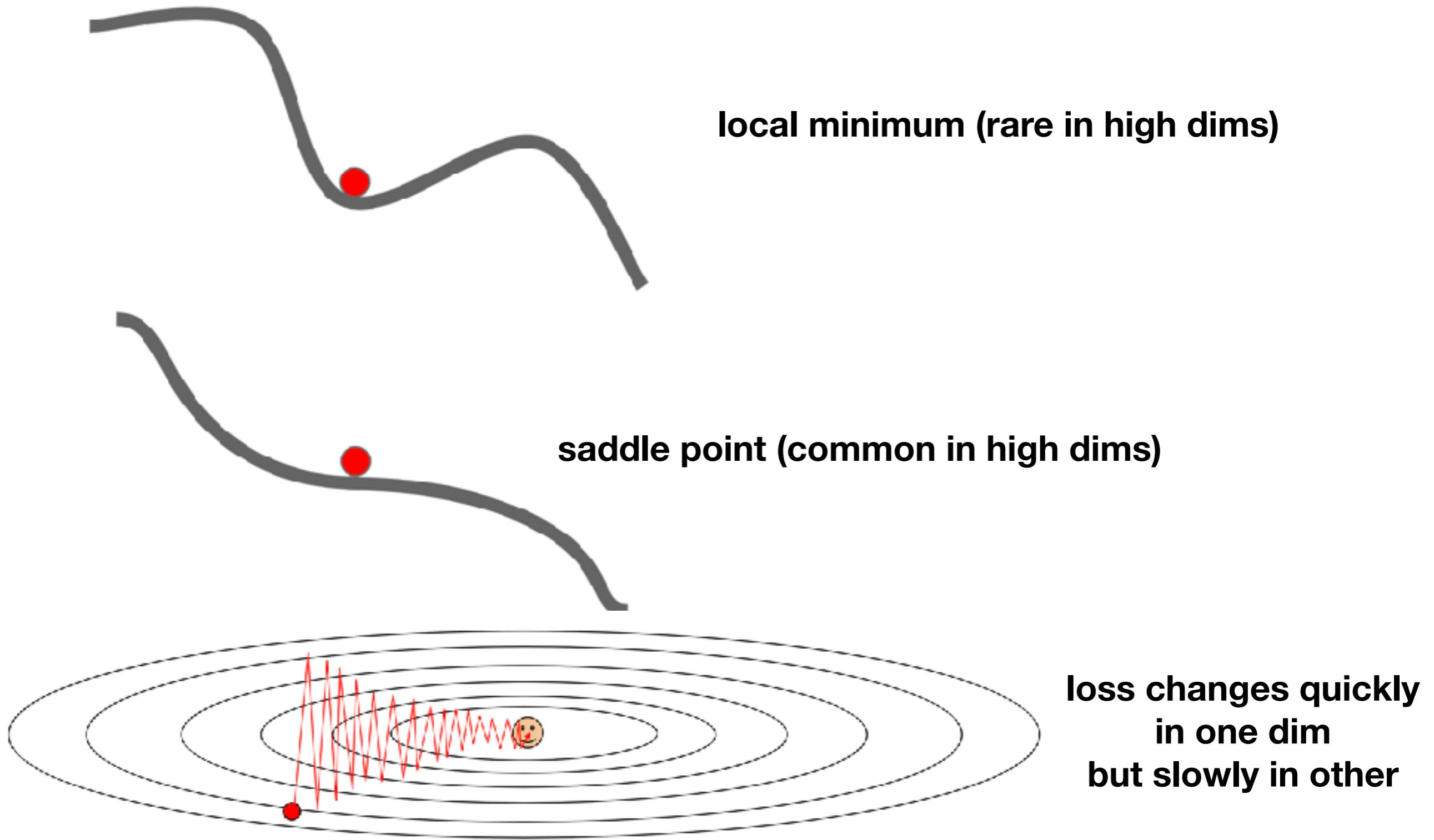
Grid Layout



Random Layout



# Problems with Gradient Descent



# THIS IS YOUR MACHINE LEARNING SYSTEM?

YUP! YOU POUR THE DATA INTO THIS BIG  
PILE OF LINEAR ALGEBRA, THEN COLLECT  
THE ANSWERS ON THE OTHER SIDE.

# WHAT IF THE ANSWERS ARE WRONG?

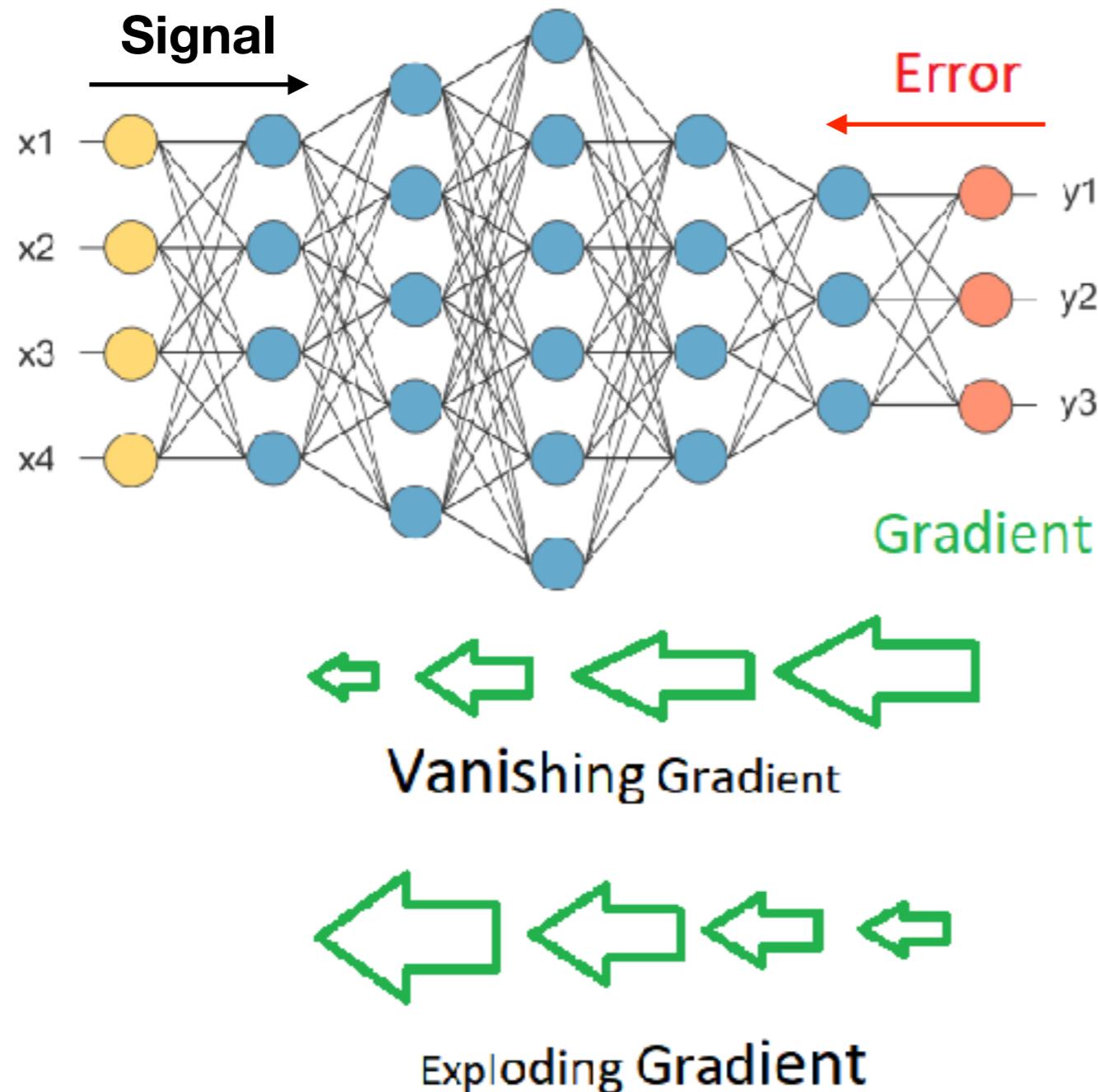
JUST STIR THE PILE UNTIL  
THEY START LOOKING RIGHT.



# Back-propagation

Forward-propagation: get **estimates** during training and **predictions** then

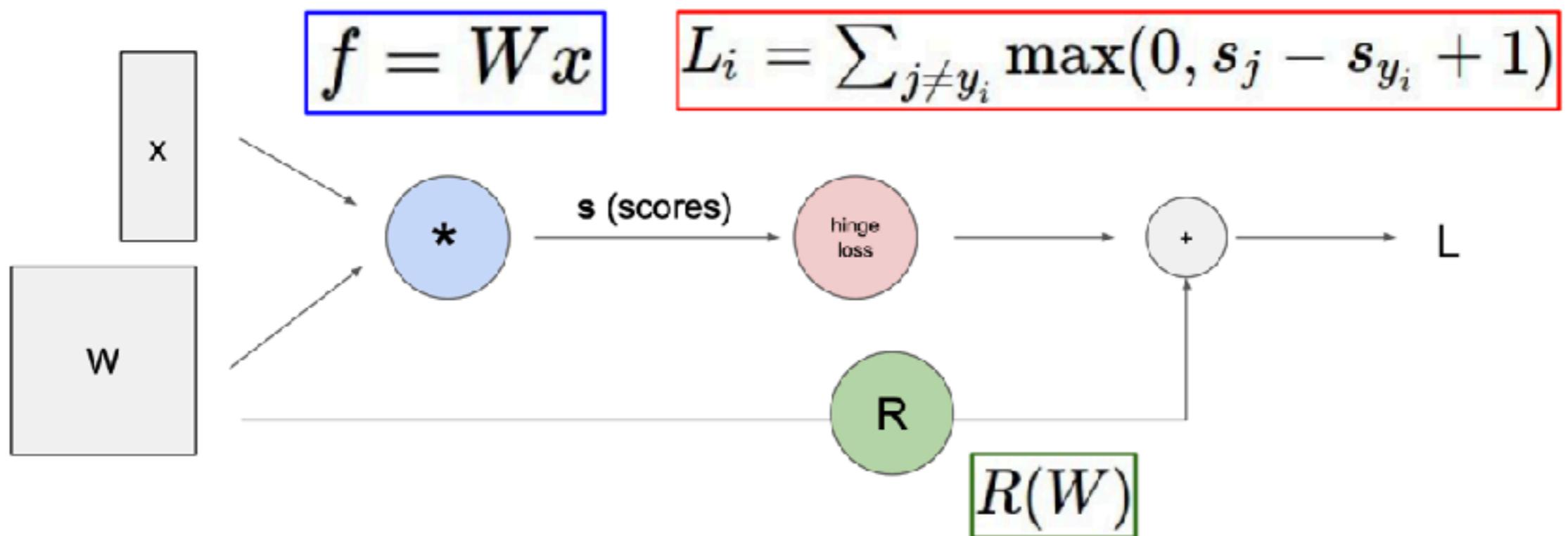
Back-propagation: apply chain rule to gradient of loss function to **adjust weights/biases**



## Remedies against vanishing gradient:

- ReLU
- Re-normalisation

# Computational graphs



## Backpropagation: a simple example

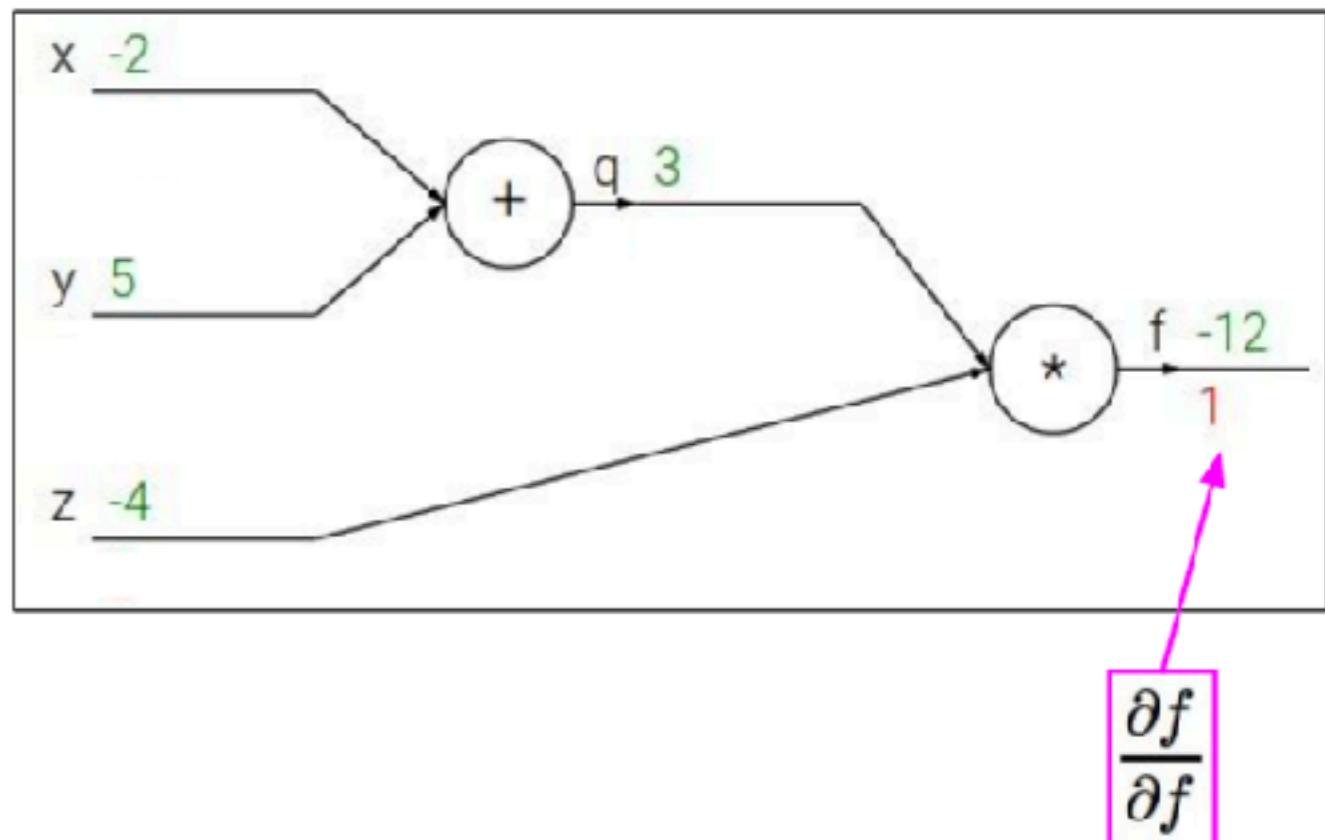
$$f(x, y, z) = (x + y)z$$

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

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

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

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



## Backpropagation: a simple example

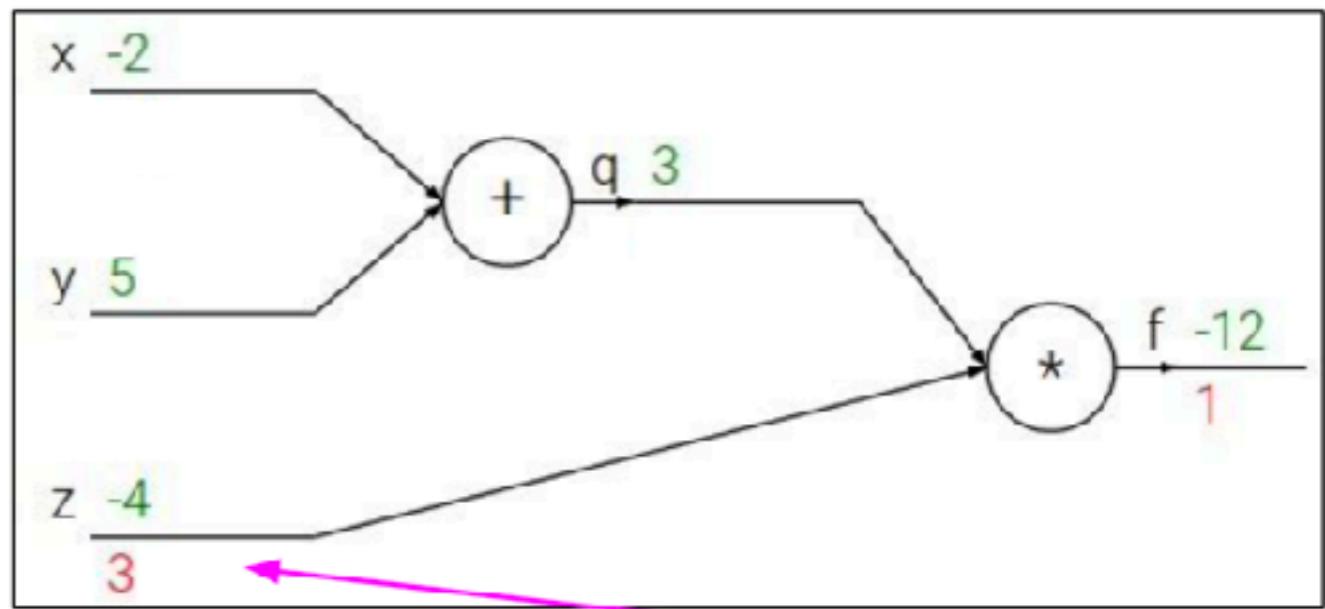
$$f(x, y, z) = (x + y)z$$

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

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

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

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



$$\frac{\partial f}{\partial z}$$

## Backpropagation: a simple example

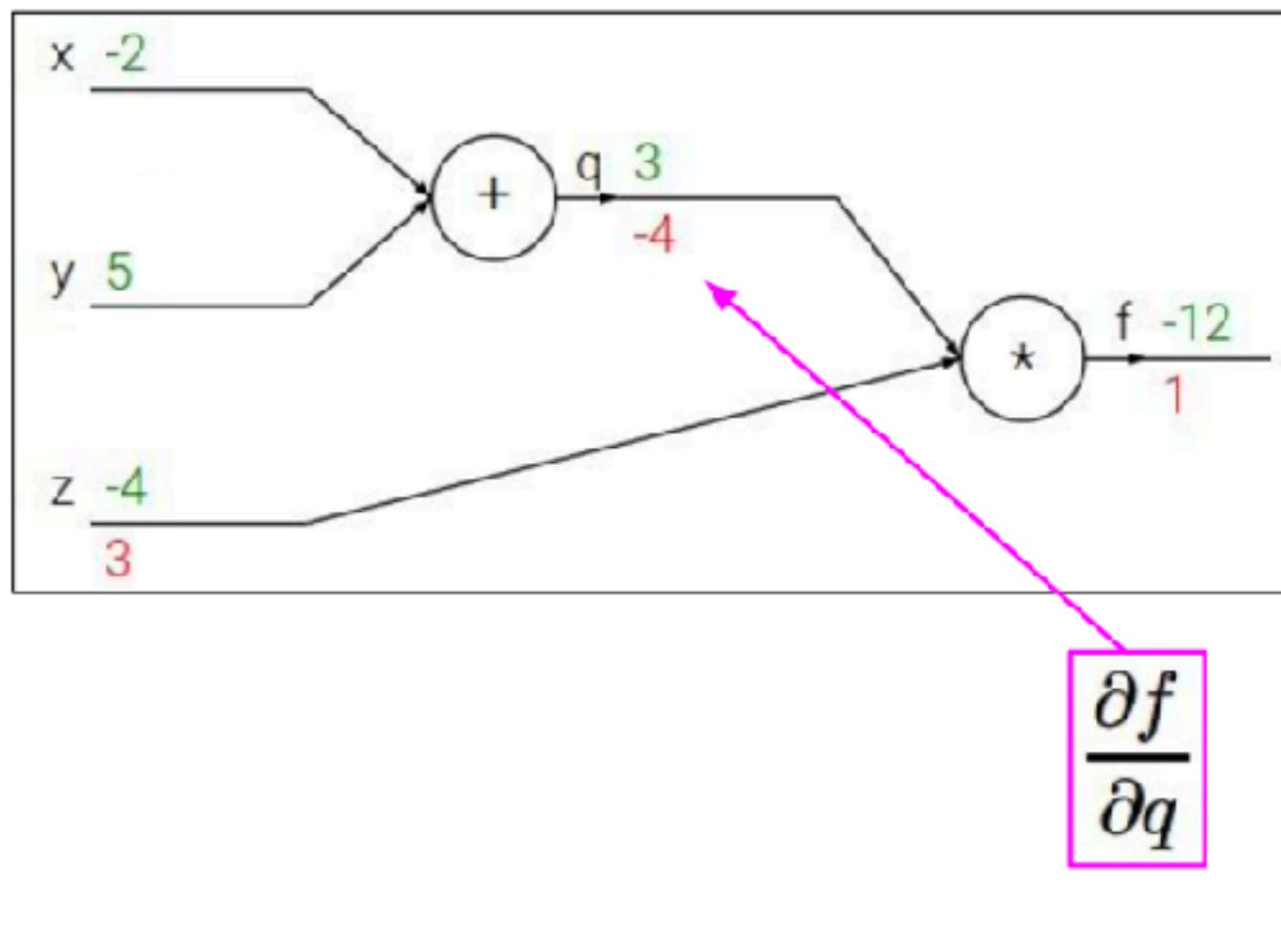
$$f(x, y, z) = (x + y)z$$

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

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

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

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



## Backpropagation: a simple example

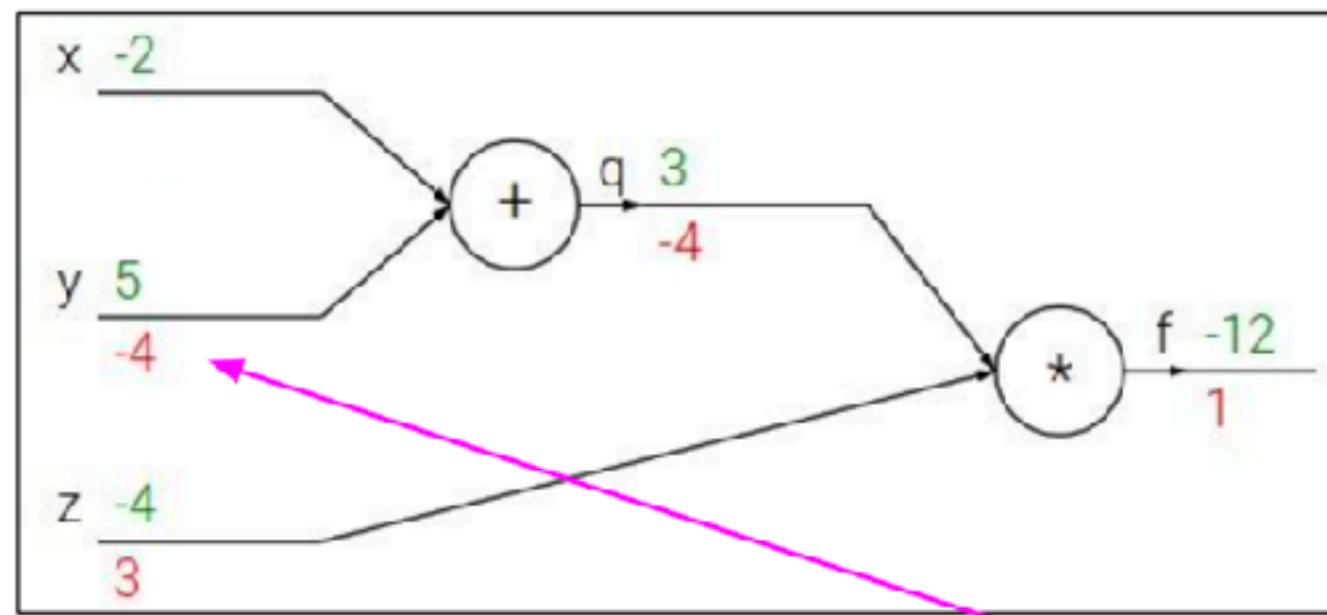
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Chain rule:

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

$$\frac{\partial f}{\partial y}$$

## Backpropagation: a simple example

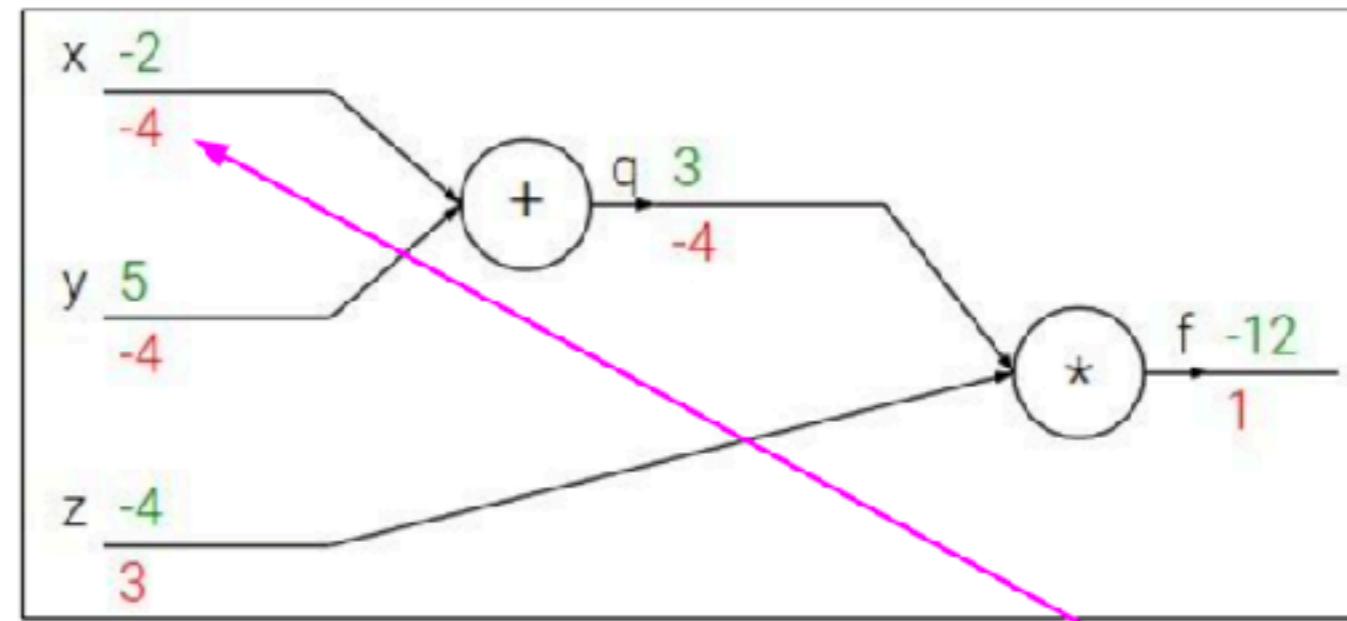
$$f(x, y, z) = (x + y)z$$

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

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

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

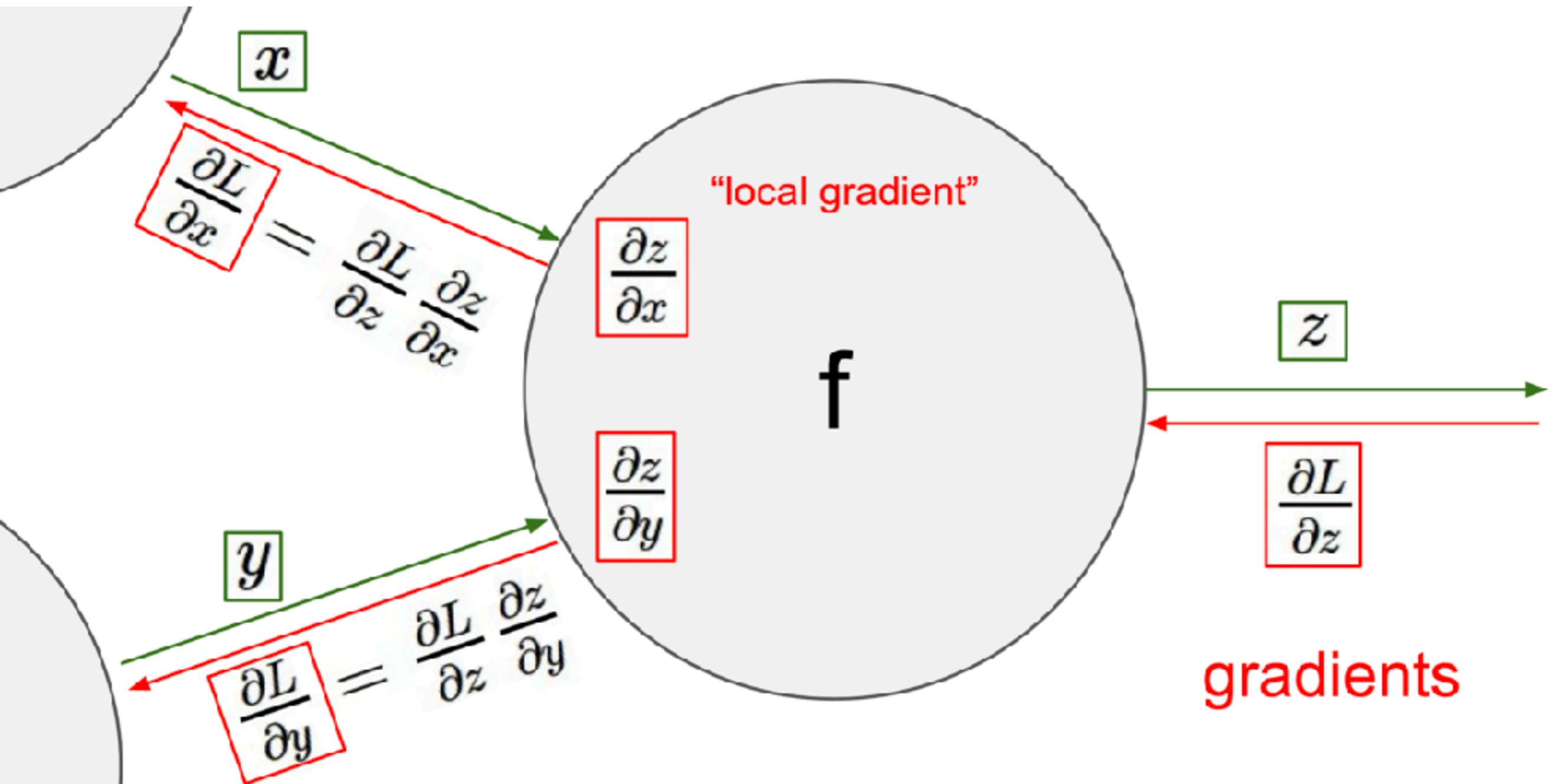
Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

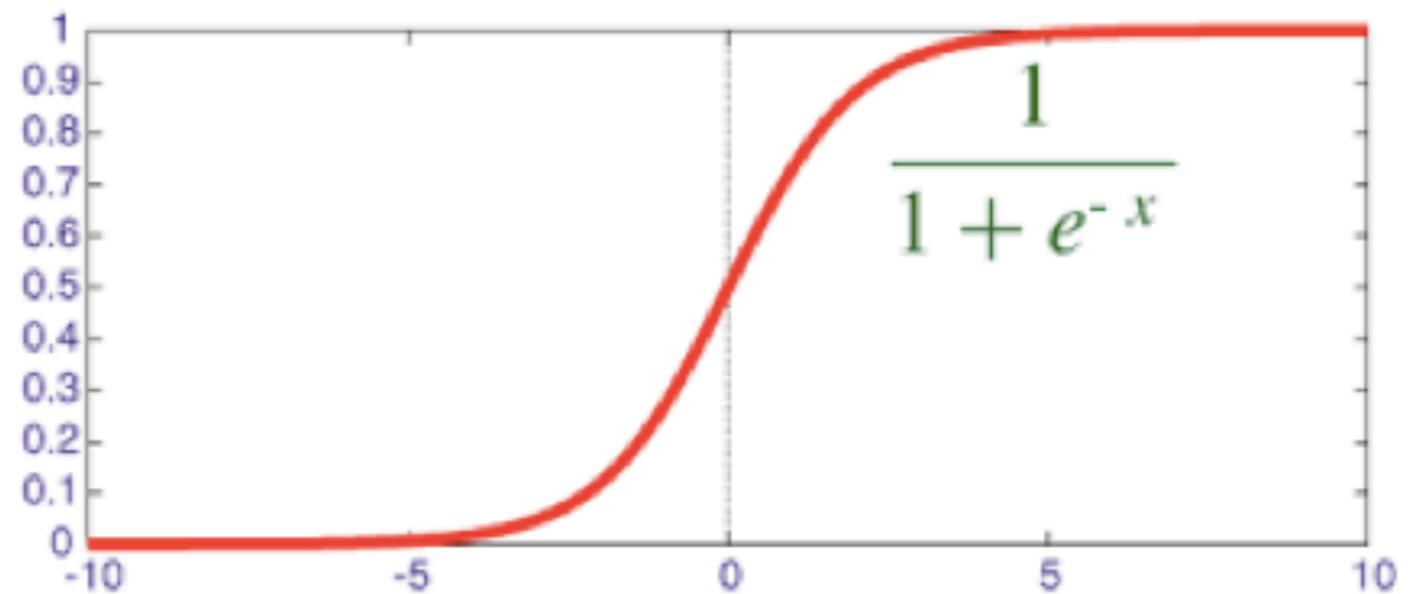
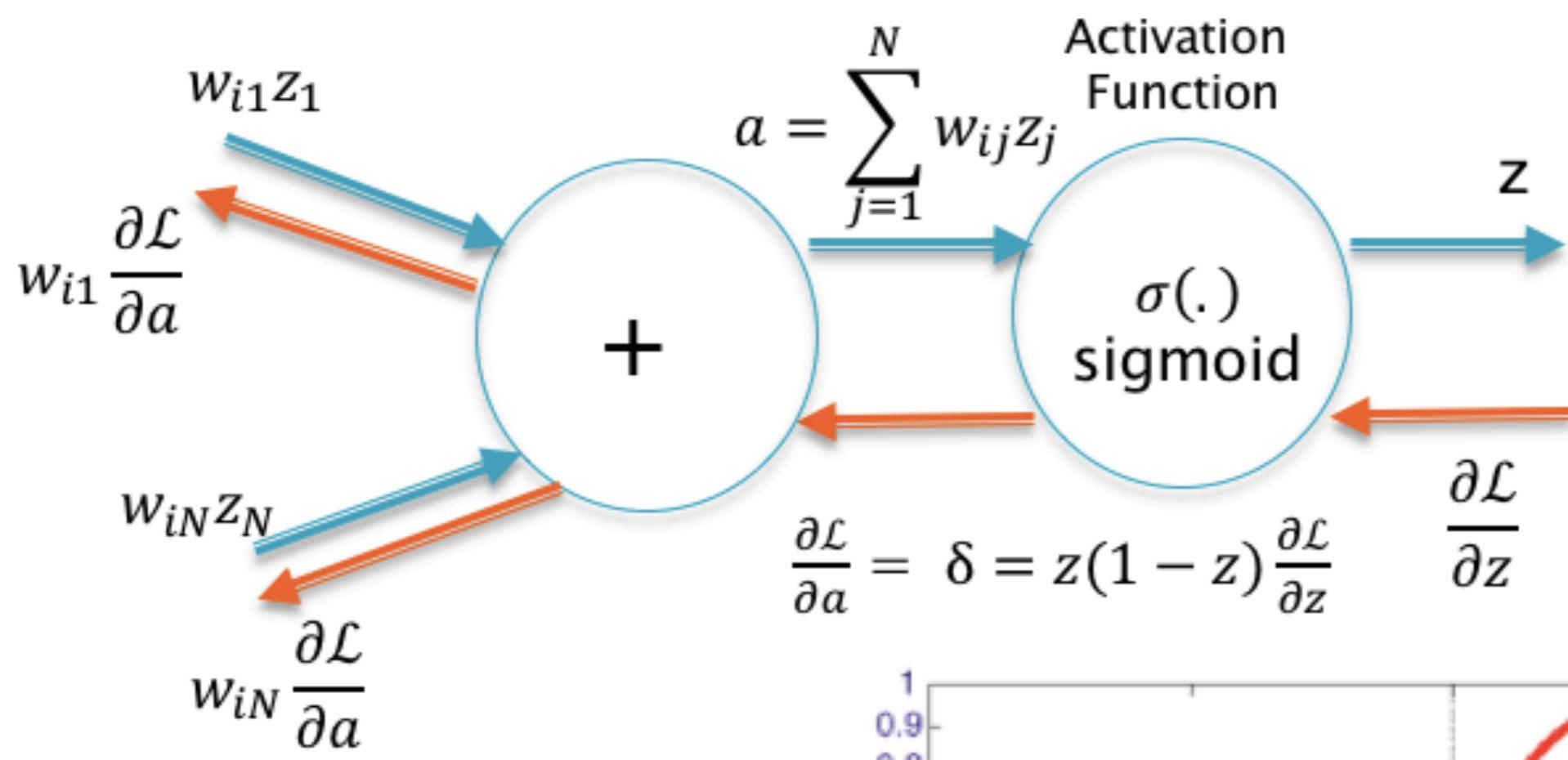


Chain rule:

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

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





$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}$$

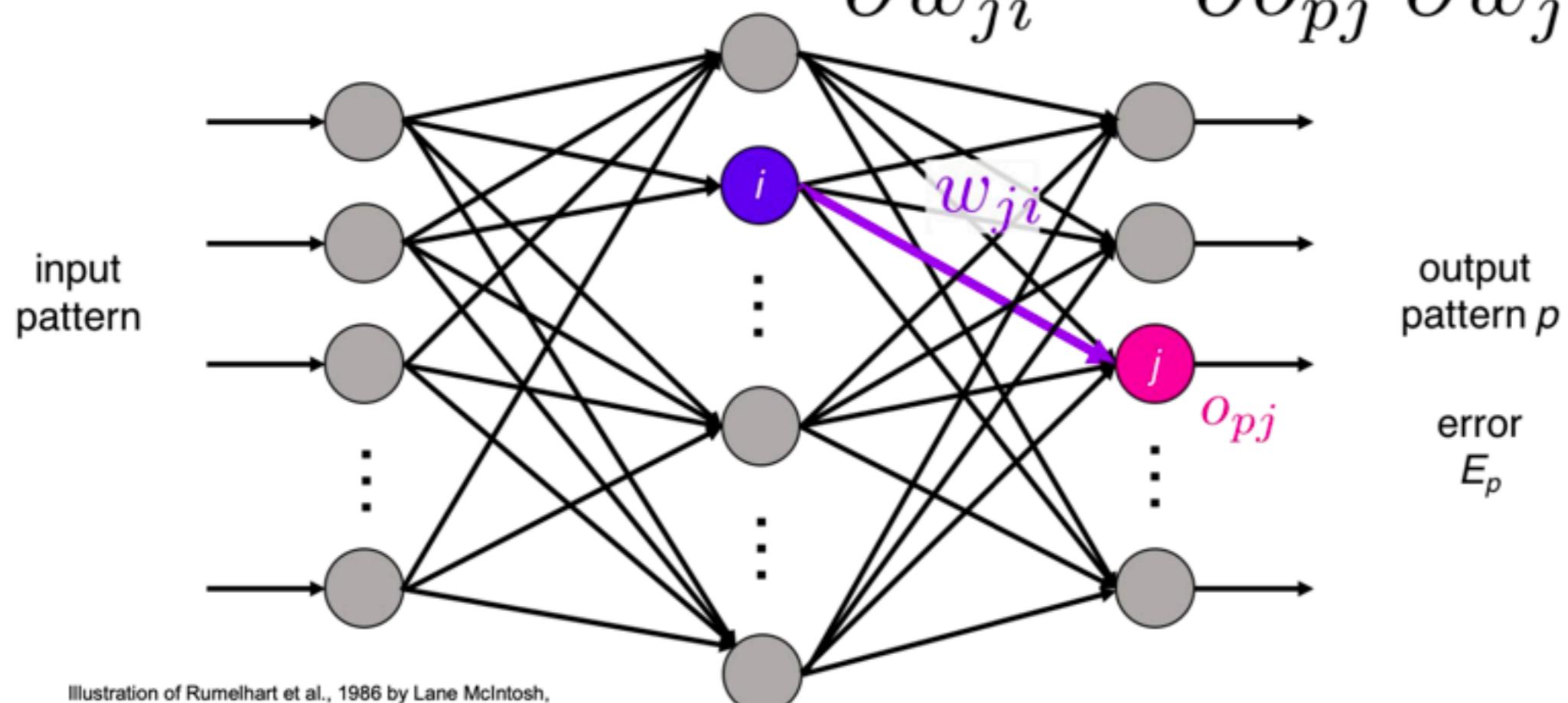
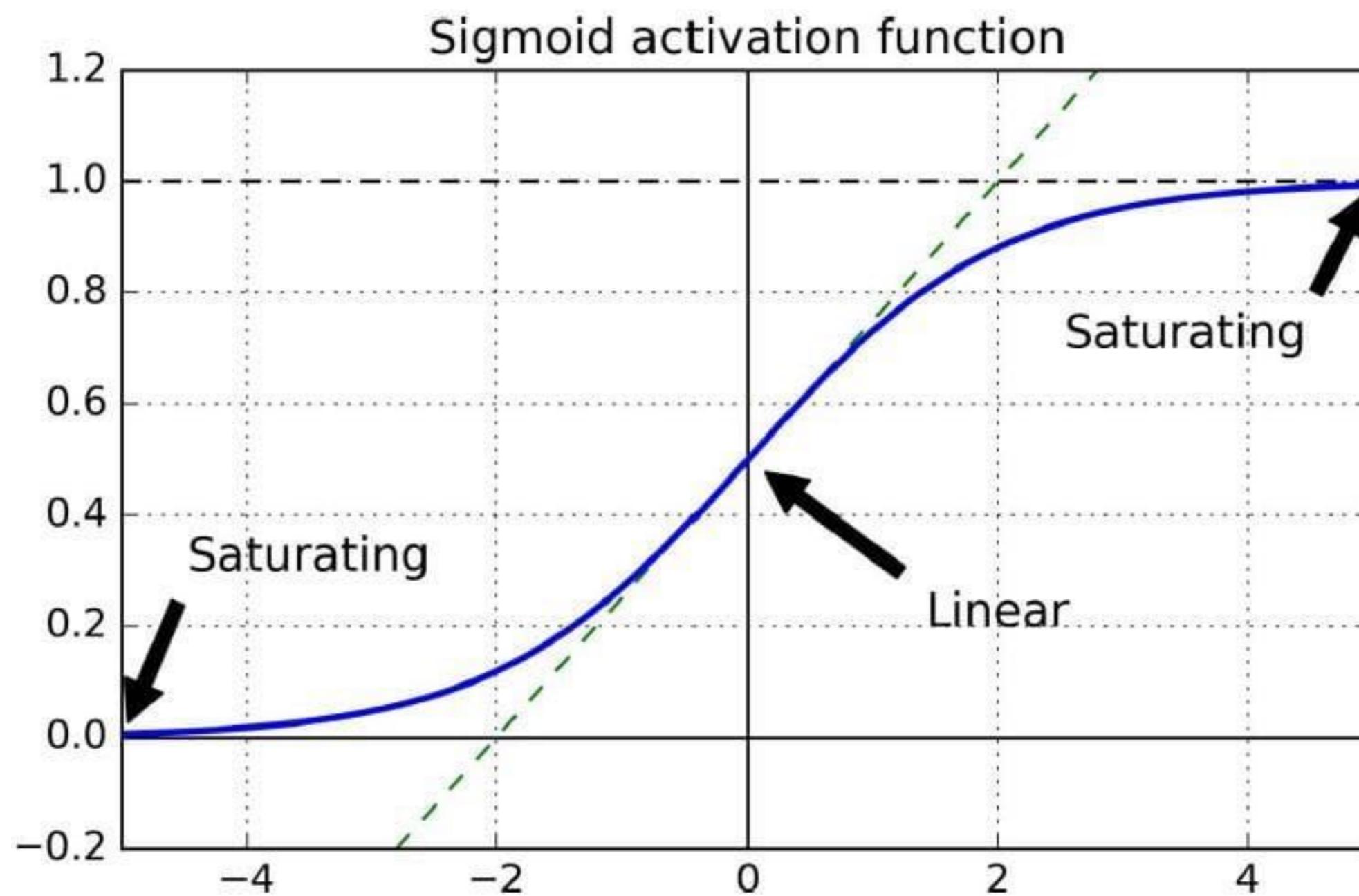


Illustration of Rumelhart et al., 1986 by Lane McIntosh,  
copyright CS231n 2017

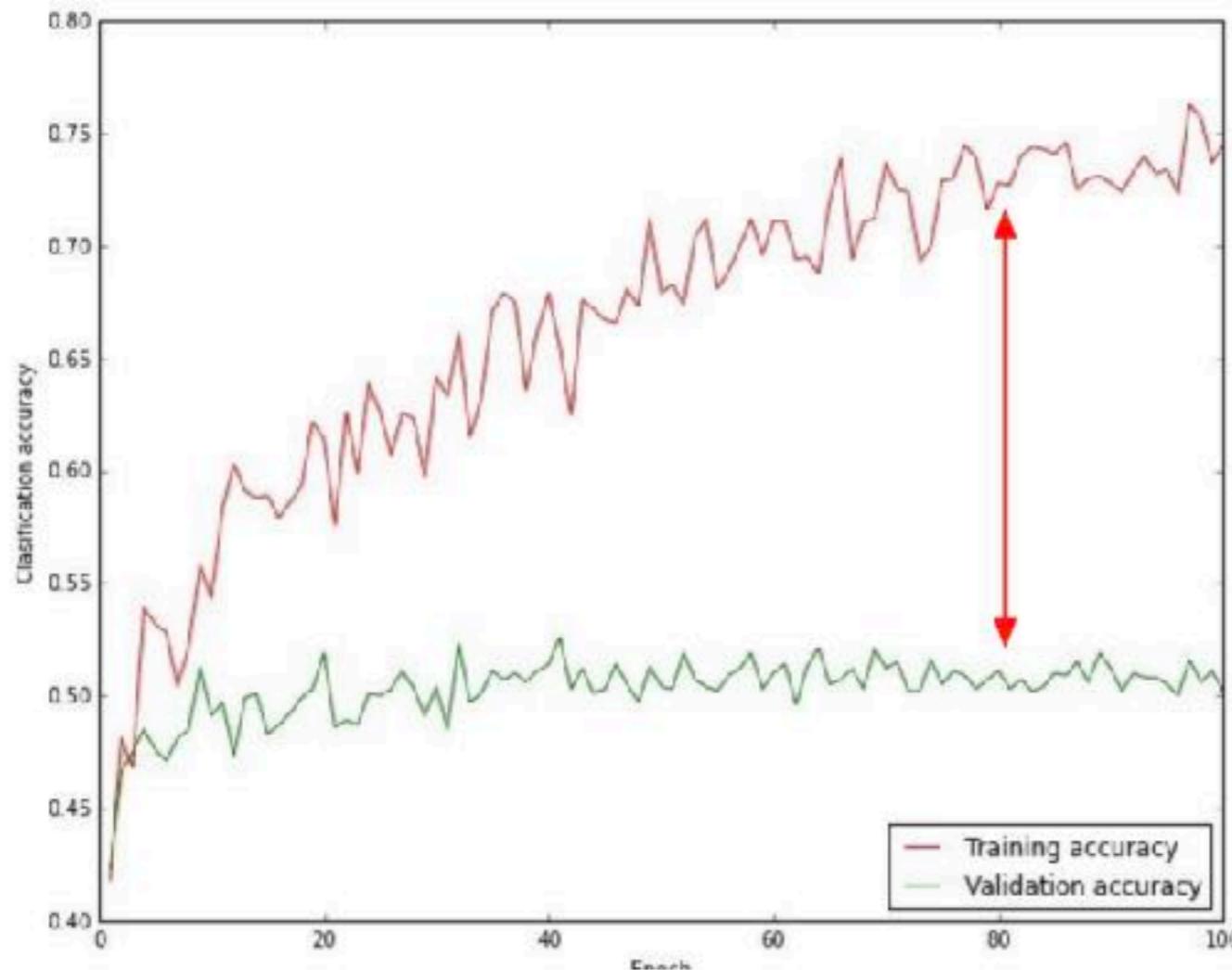
# Batch SGD scheme

Loop:

1. **Sample** a batch of data
2. **Forward** propagation through the neural network graph, calculate the loss
3. **Back-propagate** to calculate the gradients
4. **Update** weight parameters using the gradient



# Fighting over-fitting

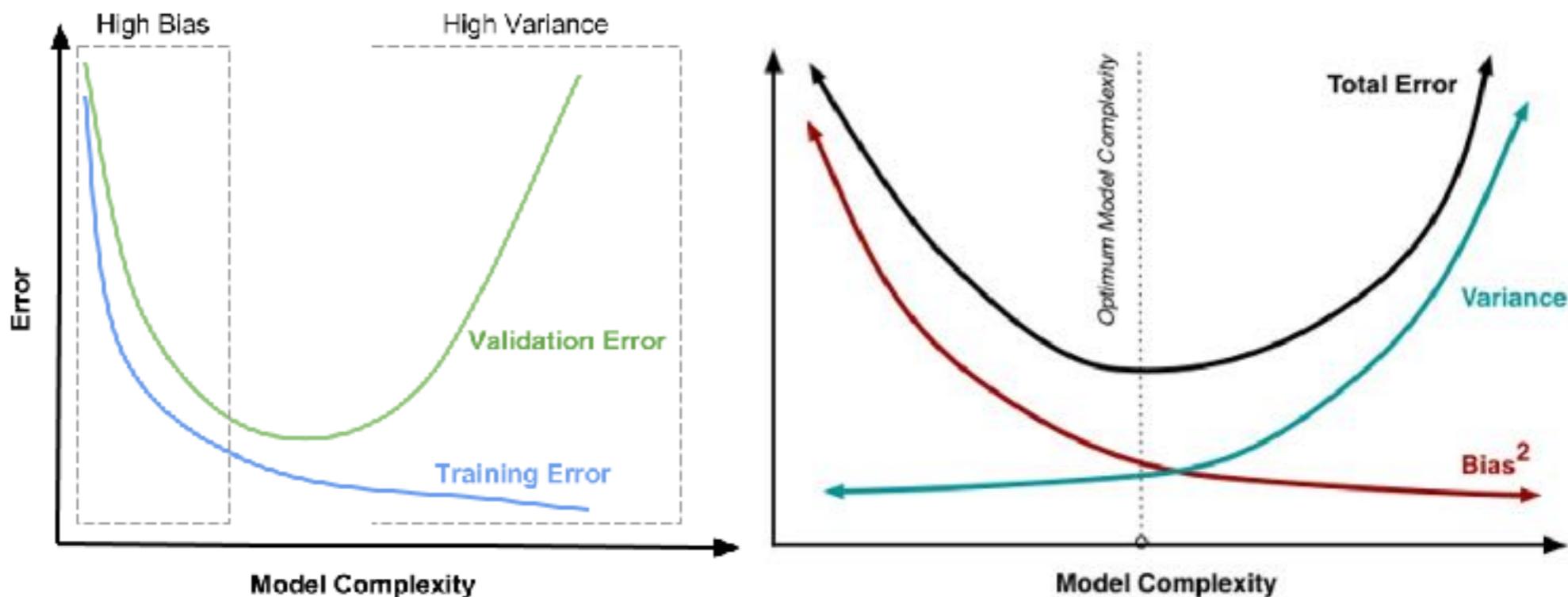


**big gap = overfitting**  
=> increase regularization strength?

**no gap**  
=> increase model capacity?

# Fighting over-fitting

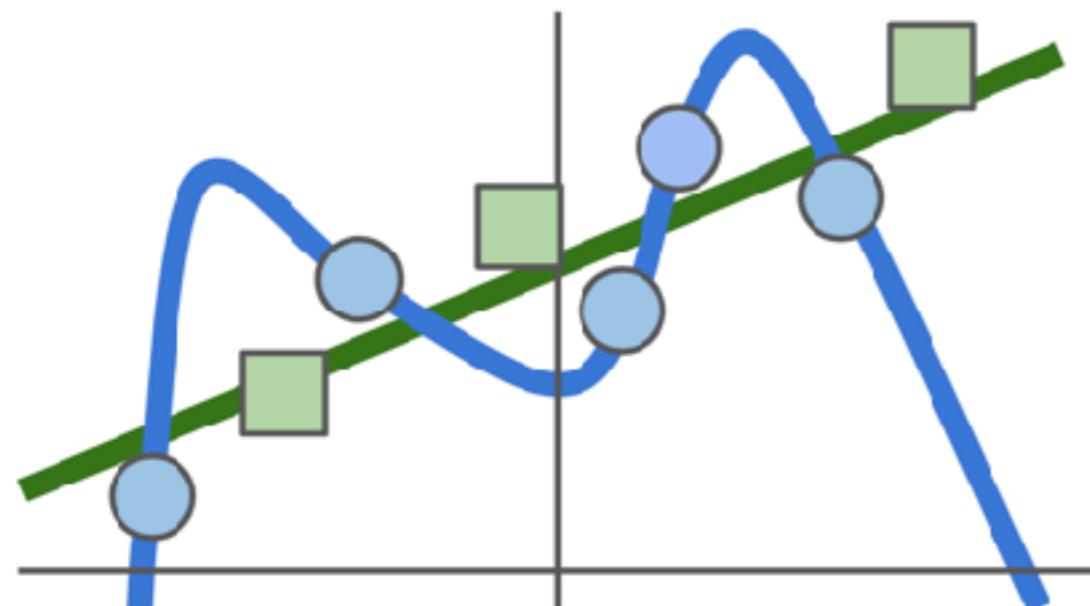
- Regularisation
- Dropout
- Early stopping
- Simplify the architecture
- More data (even if you have to make it up: translate, rotate, flip, crop, lighten/darken, add noise)  
# of weights  $\sim$  VC dimension  $\sim$  # degrees of freedom



$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i) + \lambda R(W)$$

**Data loss:** Model predictions should match training data

**Regularization:** Model should be “simple”, so it works on test data



**Occam's Razor:**  
*“Among competing hypotheses,  
the simplest is the best”*  
William of Ockham, 1285 - 1347

# Regularization

$\lambda$  = regularization strength  
(hyperparameter)

$$L = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

**L2 regularization**

$$R(W) = \sum_k \sum_l W_{k,l}^2 \quad \text{"weight decay"}$$

**L1 regularization**

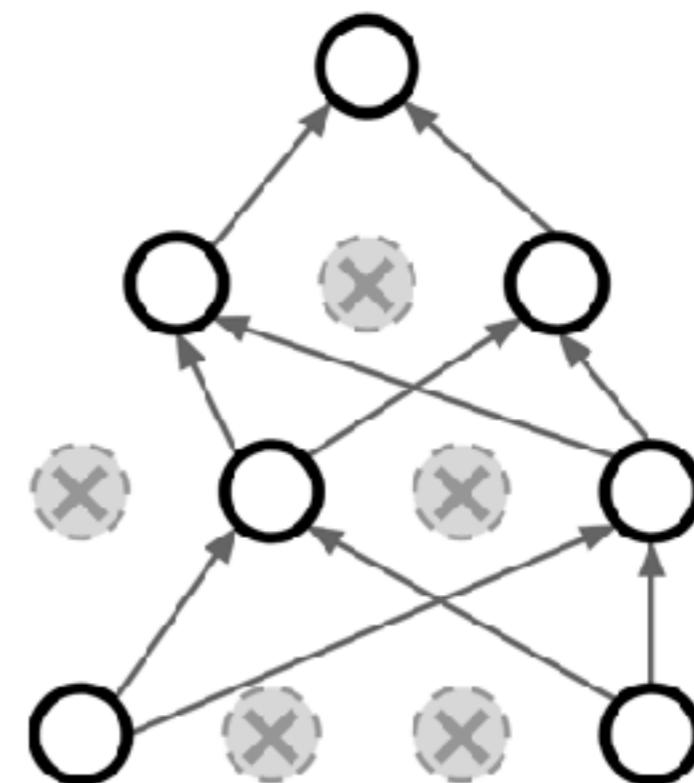
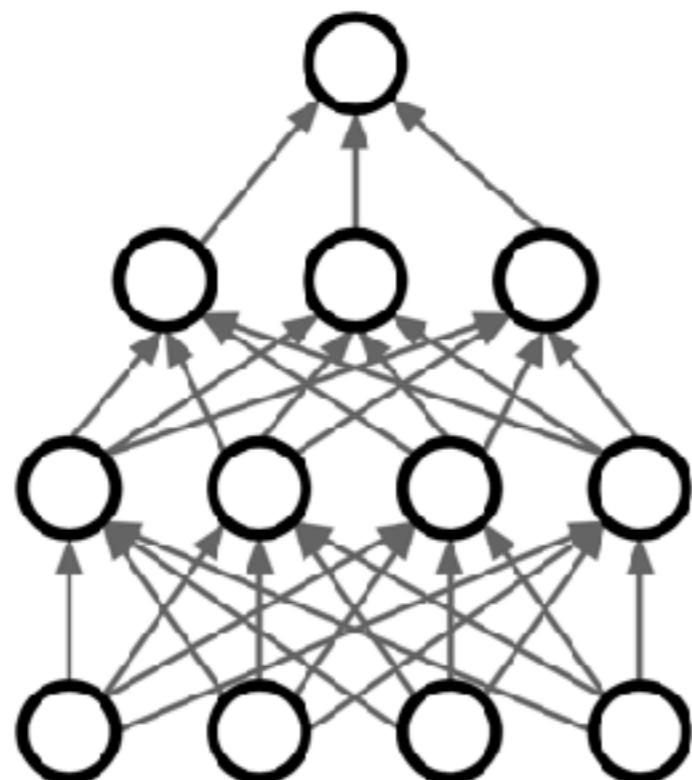
$$R(W) = \sum_k \sum_l |W_{k,l}|$$

**Elastic net (L1 + L2)**

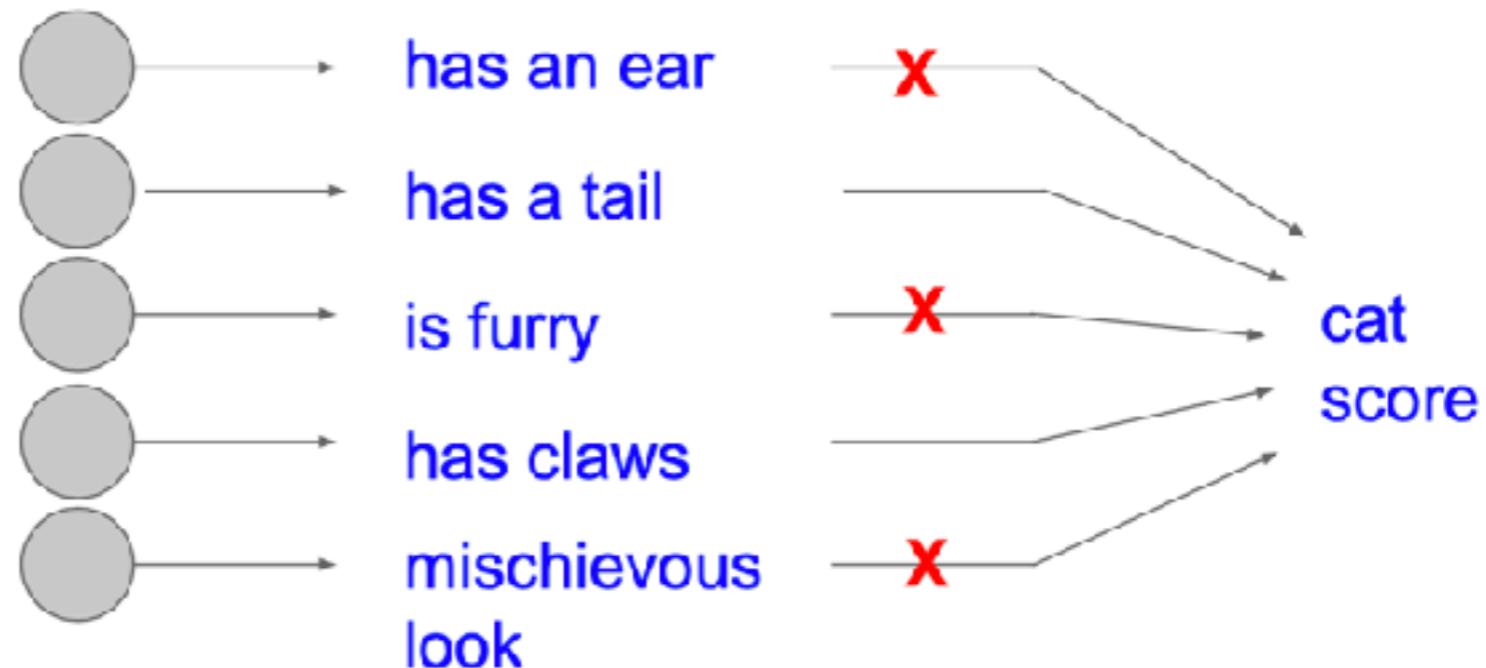
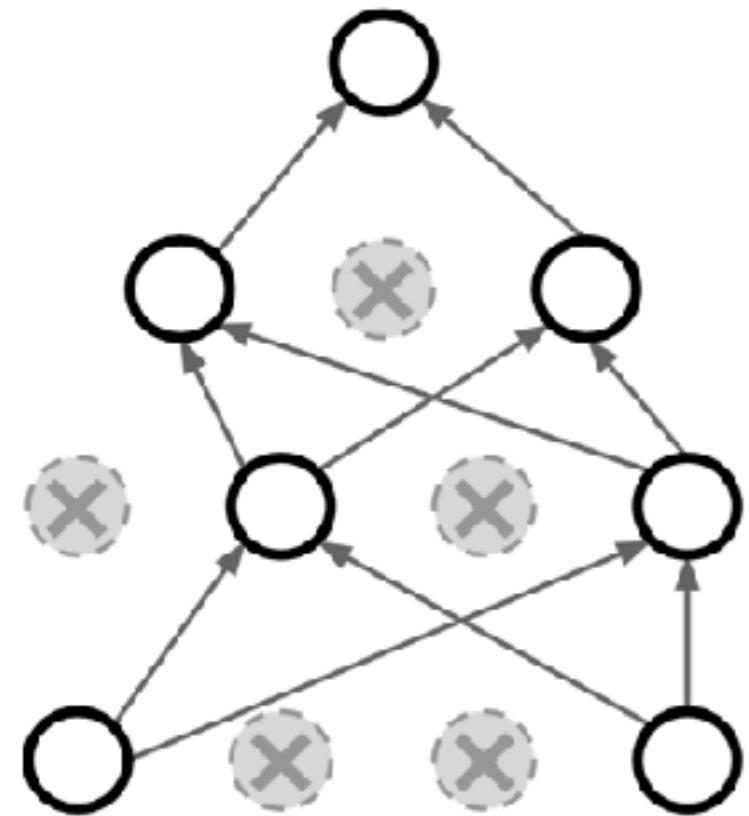
$$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$$

# Regularization: Dropout

In each forward pass, randomly set some neurons to zero  
Probability of dropping is a hyperparameter; 0.5 is common



Forces the network to have a redundant representation;  
Prevents co-adaptation of features

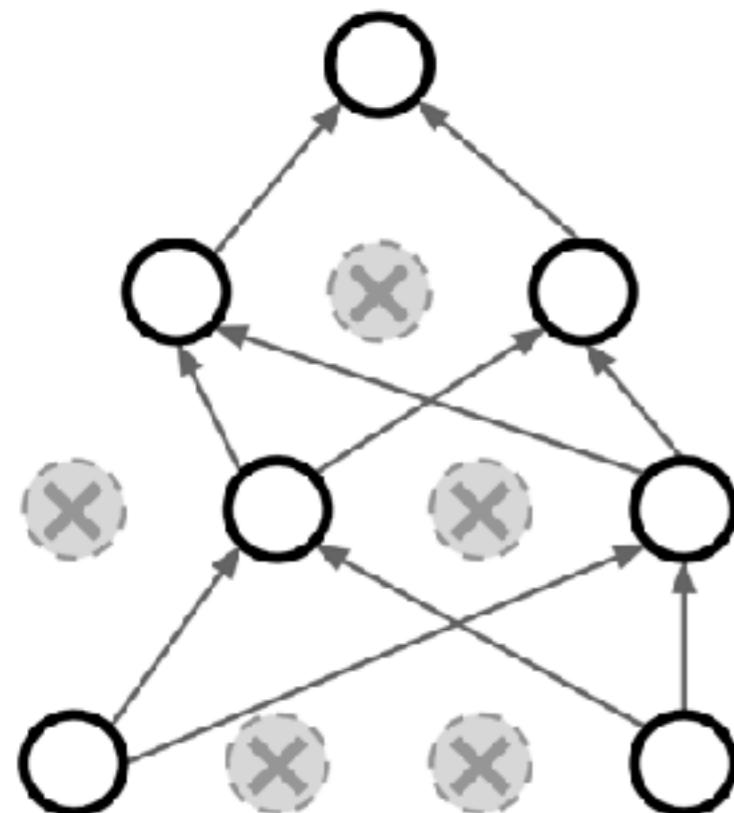


Another interpretation:

Dropout is training a large **ensemble** of models (that share parameters).

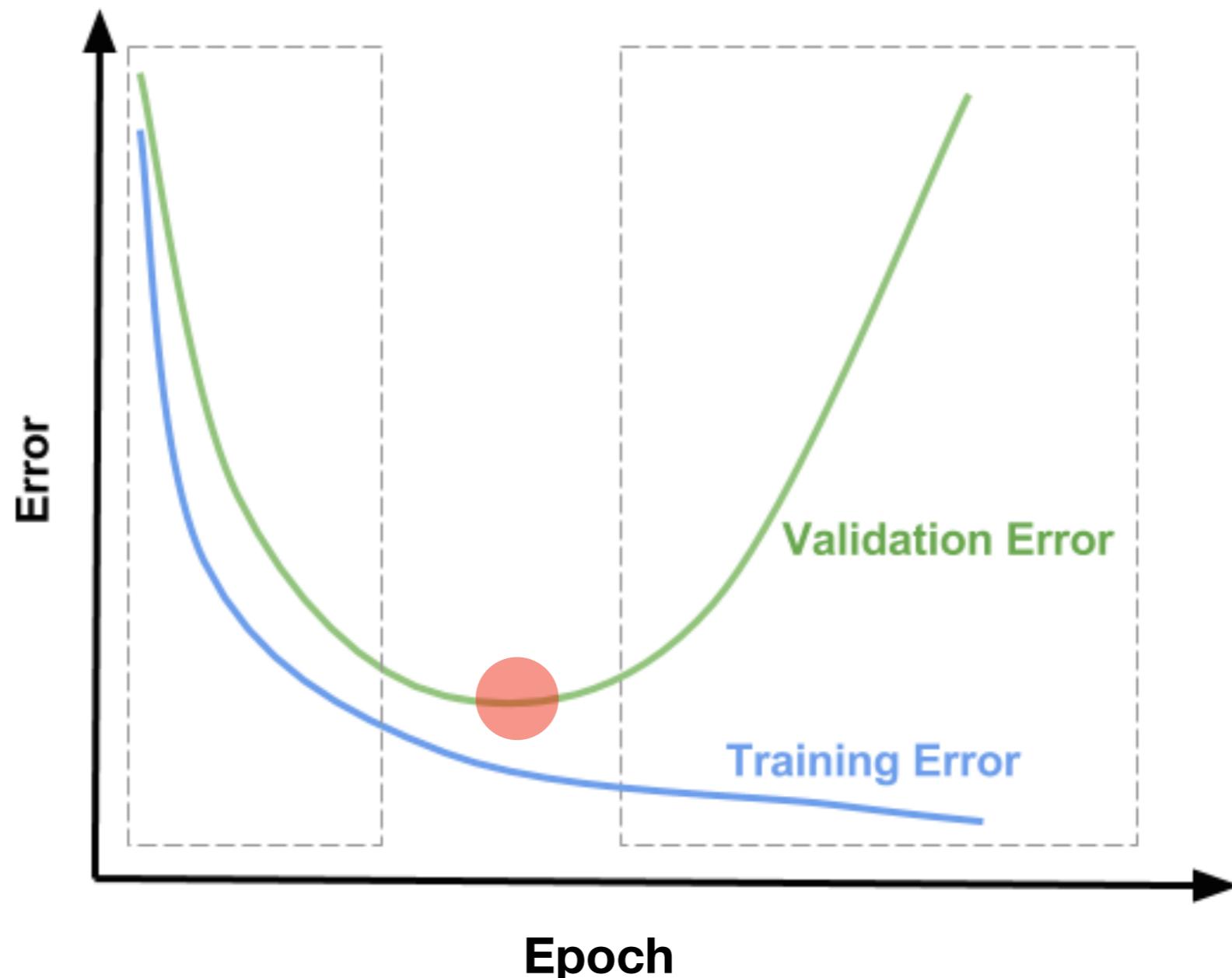
Each binary mask is one model

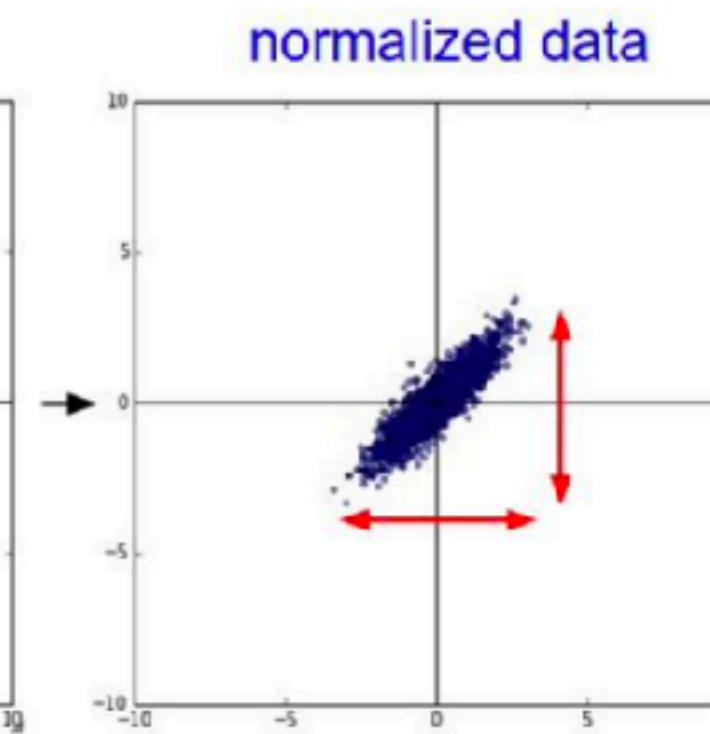
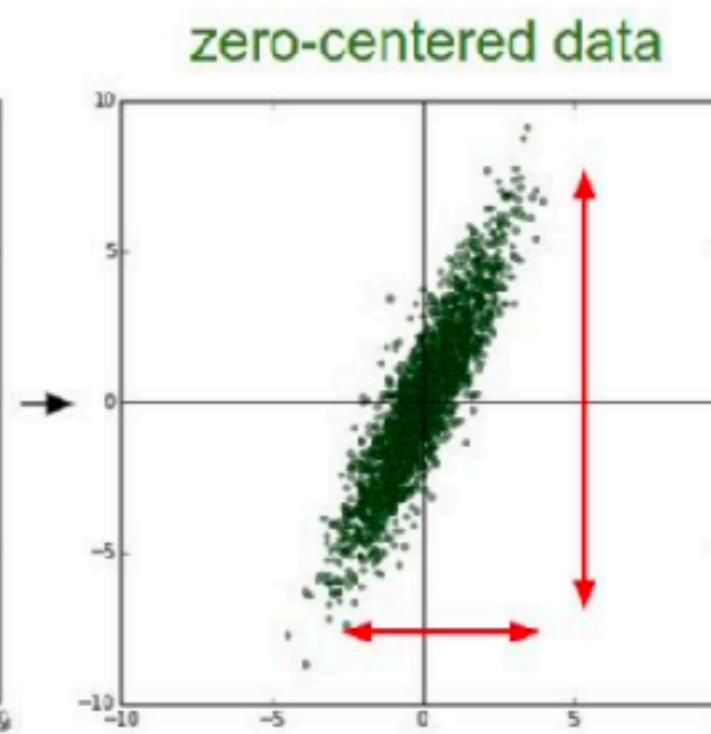
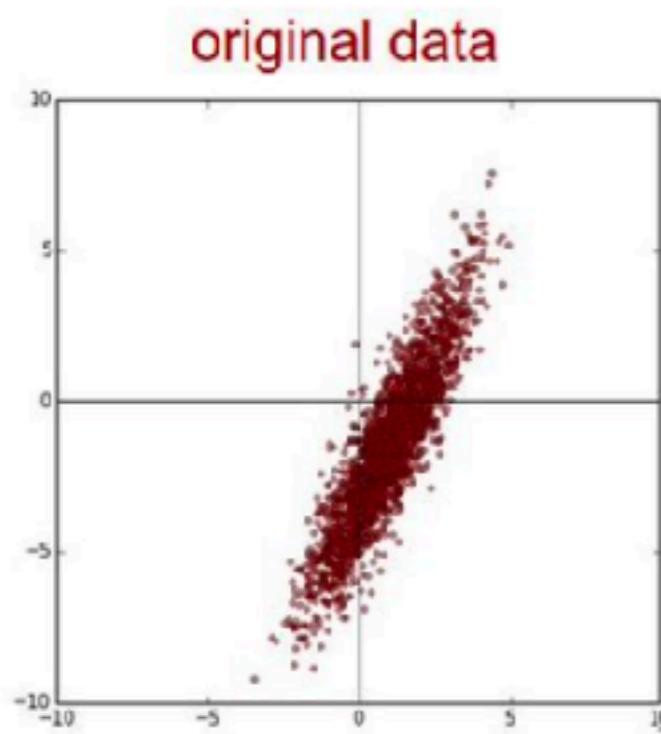
An FC layer with 4096 units has  $2^{4096} \sim 10^{1233}$  possible masks!  
Only  $\sim 10^{82}$  atoms in the universe...



# Early Stopping

@best or @plateau

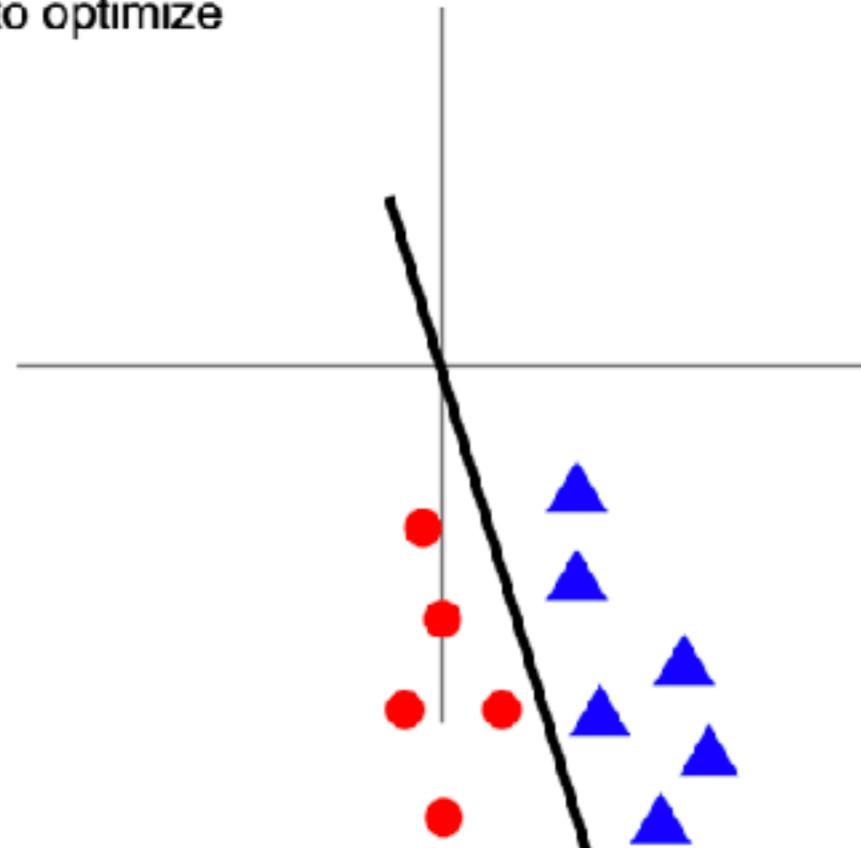




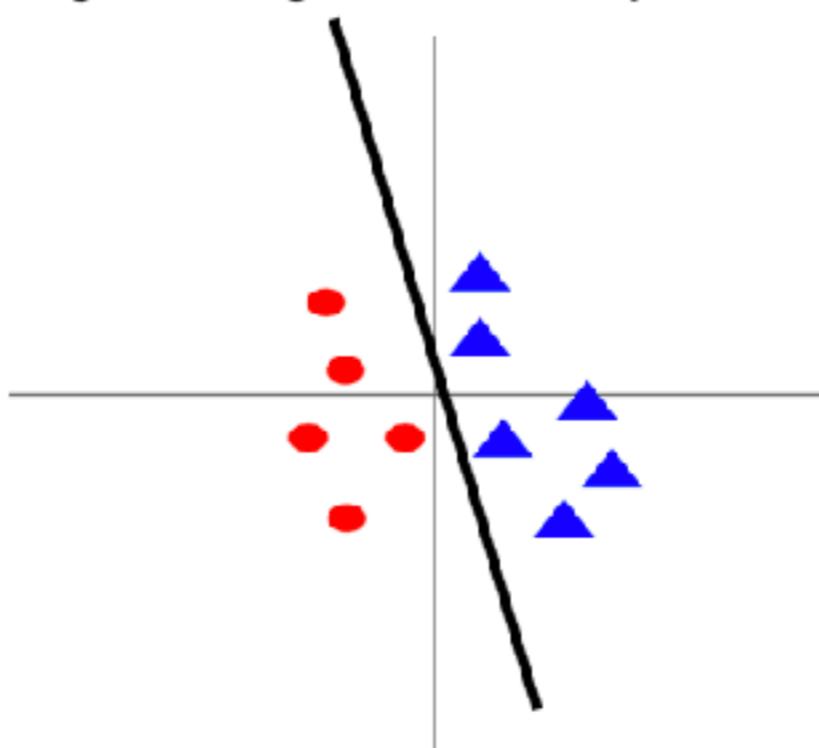
```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

**Before normalization:** classification loss very sensitive to changes in weight matrix; hard to optimize

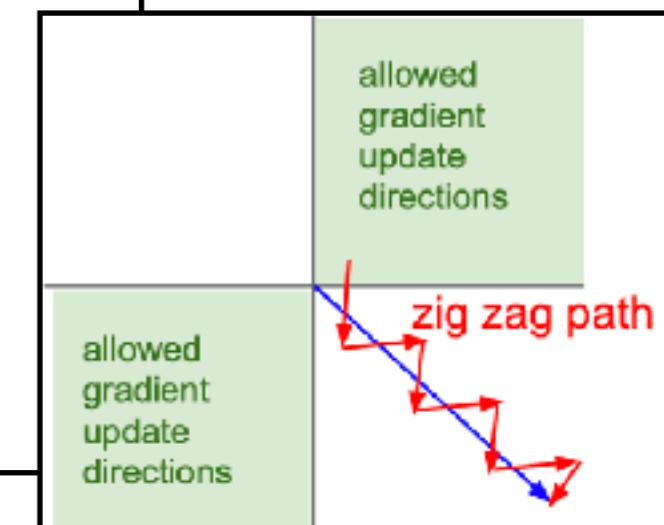


**After normalization:** less sensitive to small changes in weights; easier to optimize

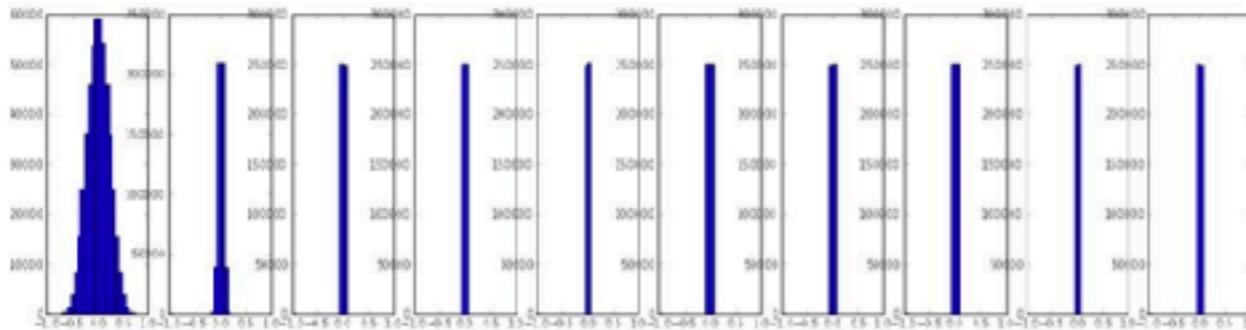


if output is always positive,  
i.e. sigmoid

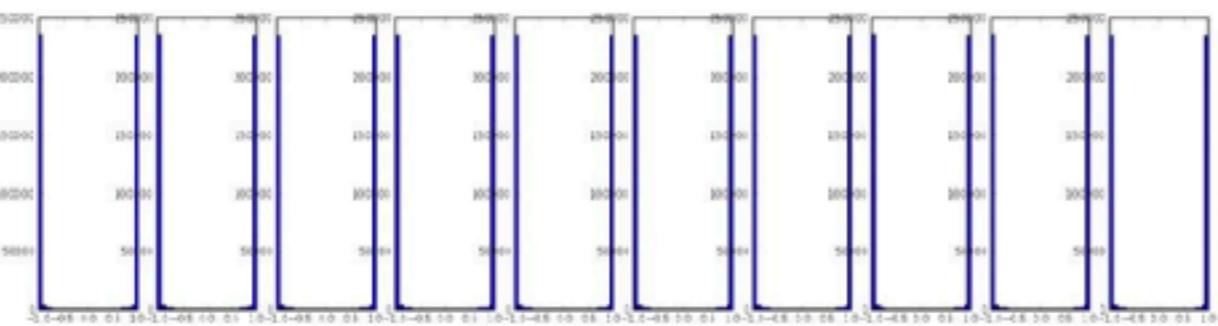
$$f\left(\sum_i w_i x_i + b\right)$$



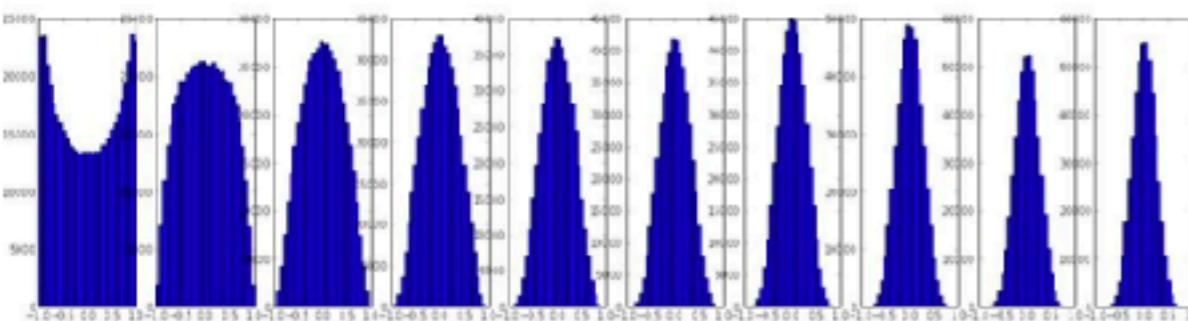
# Weight initialisation



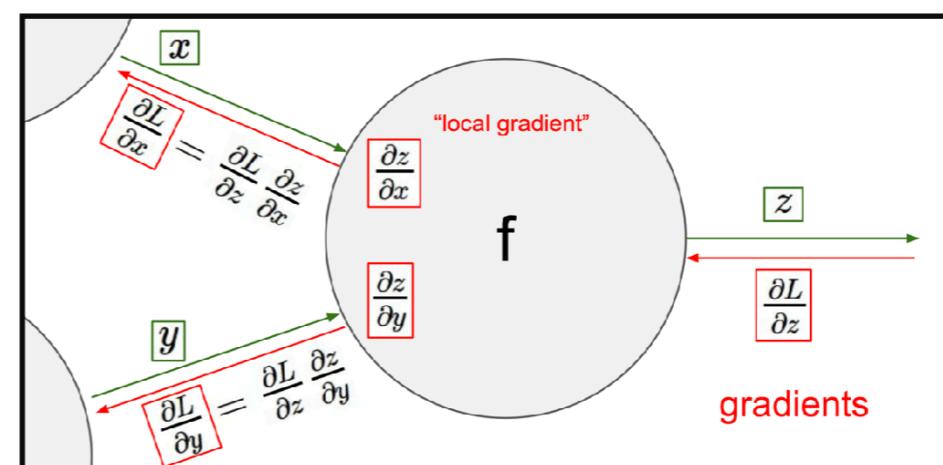
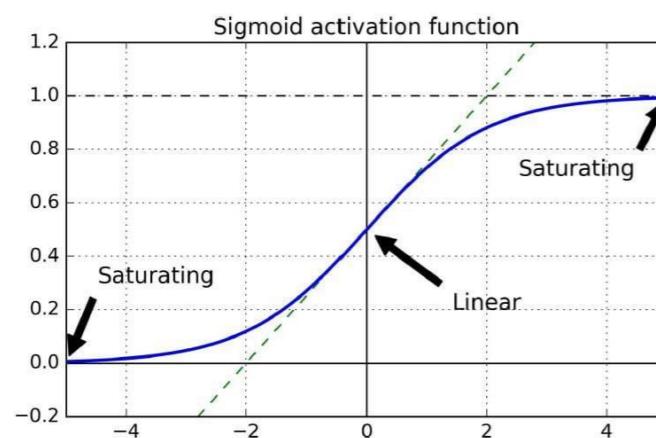
**Initialization too small:**  
Activations go to zero, gradients also zero,  
No learning



**Initialization too big:**  
Activations saturate (for tanh),  
Gradients zero, no learning



**Initialization just right:**  
Nice distribution of activations at all layers,  
Learning proceeds nicely



# Batch Normalization

[Ioffe and Szegedy, 2015]

Normalize:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

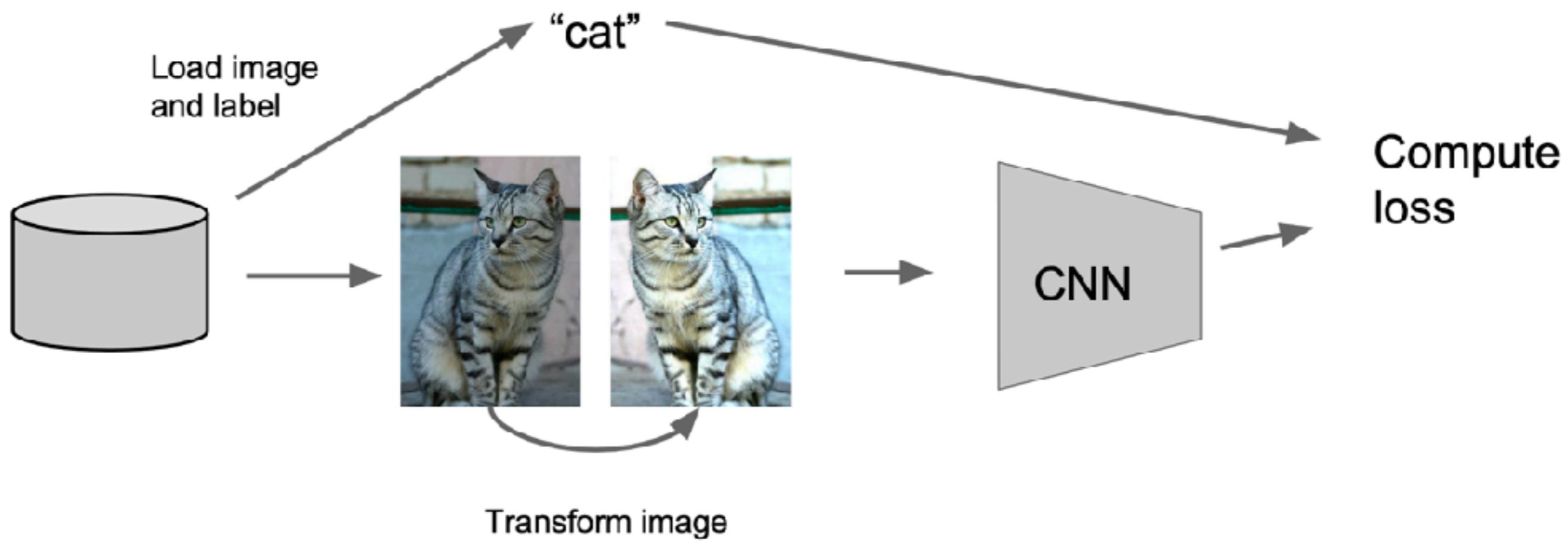
Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\text{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \text{E}[x^{(k)}]$$

to recover the identity mapping.

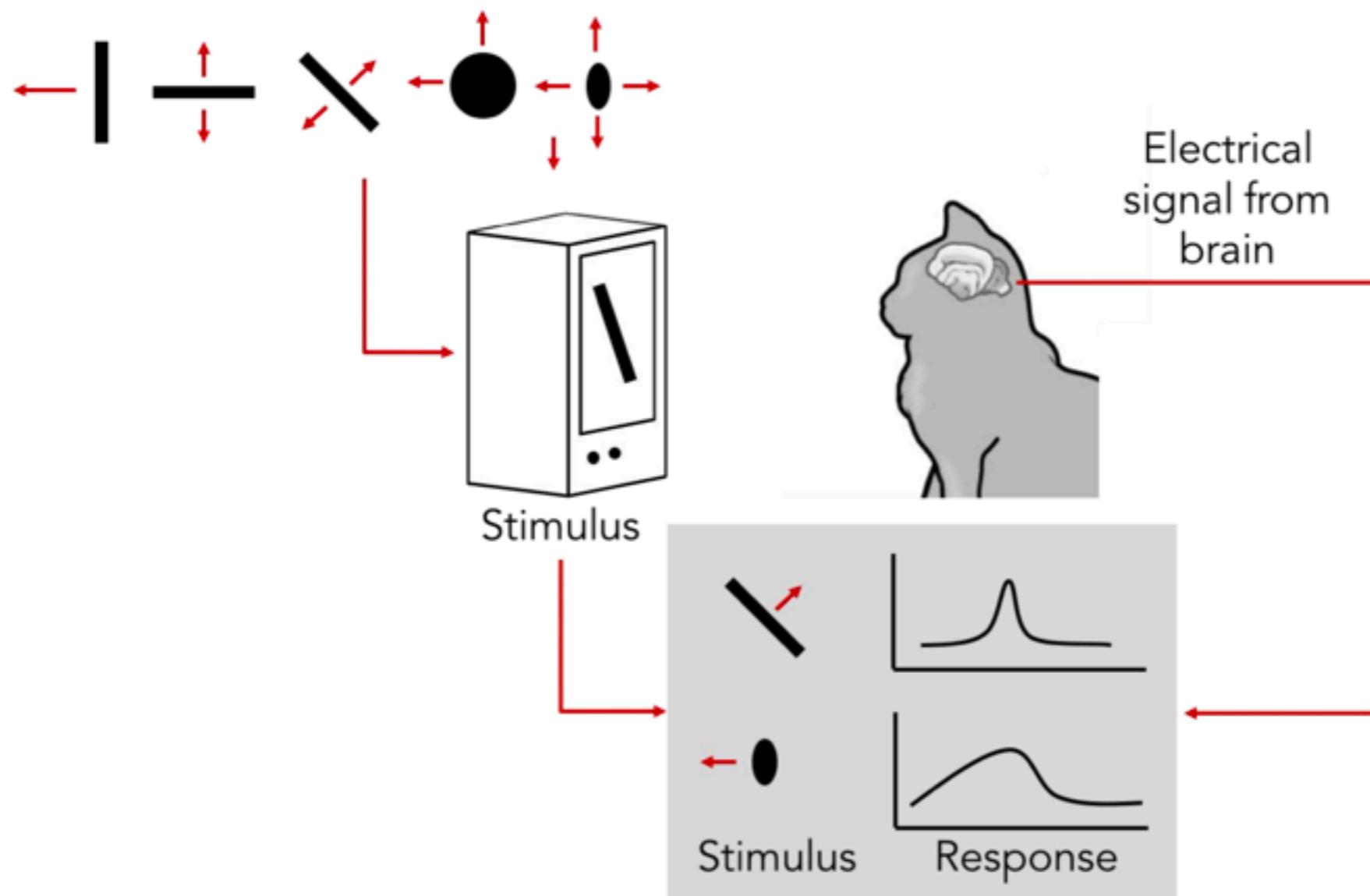
# Regularization: Data Augmentation

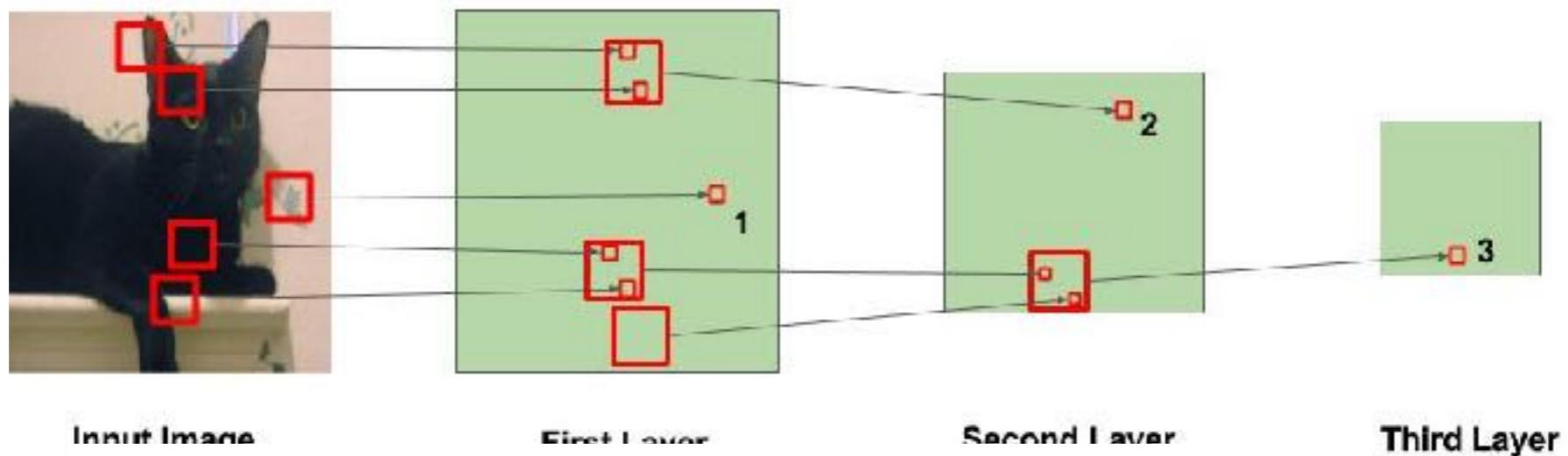


**translate, rotate, flip, crop, lighten/darken, mask, add noise, ...  
use your fantasy...**

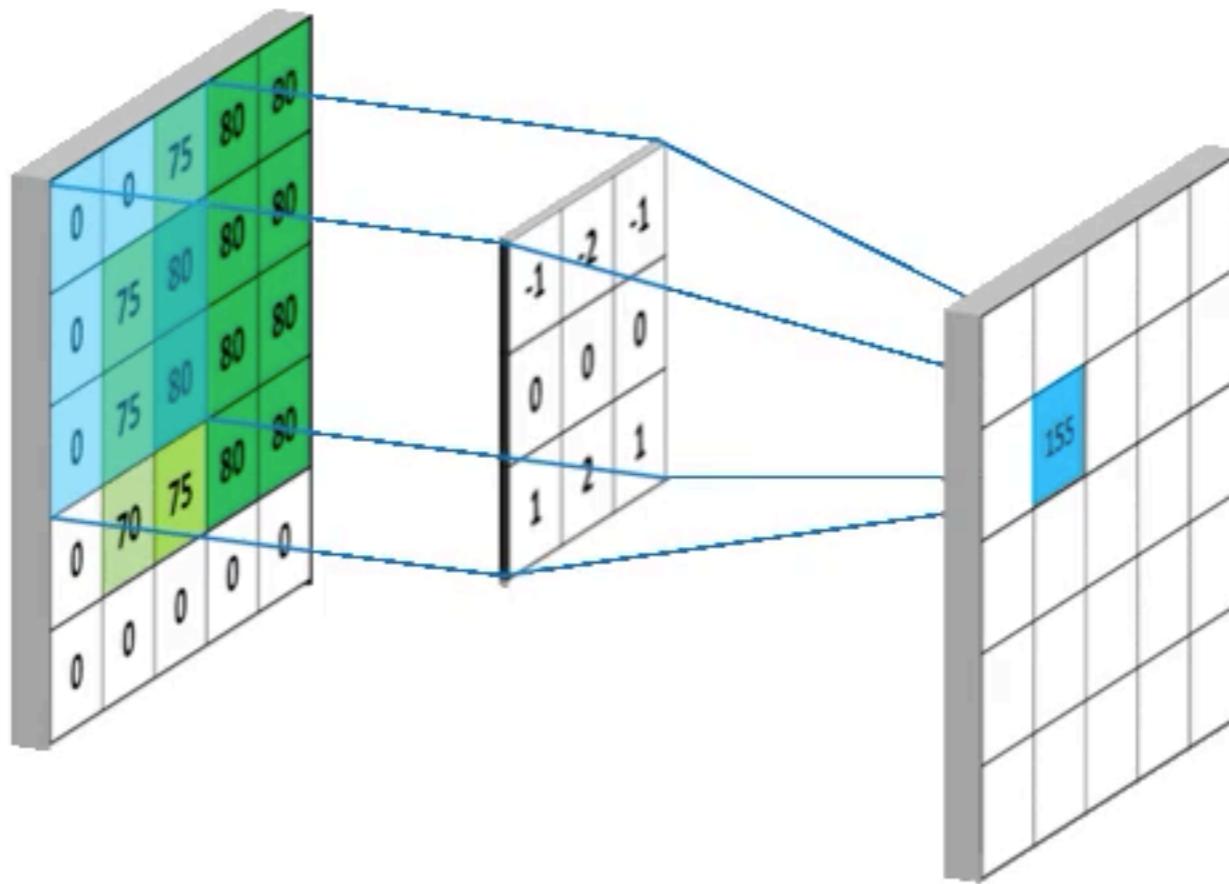


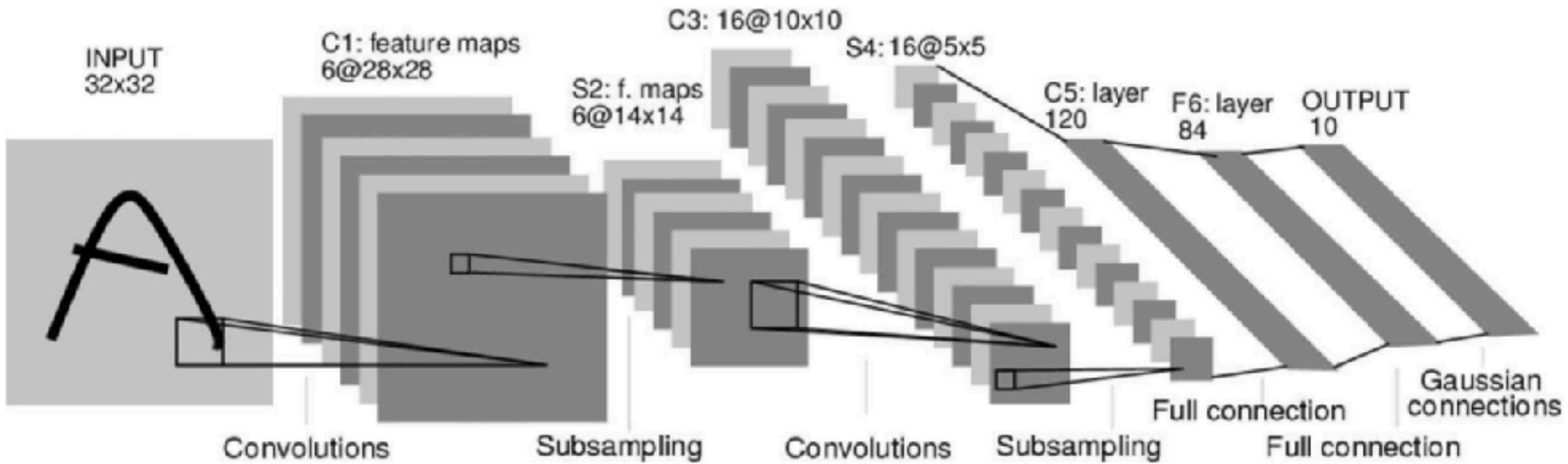
**Boom of species 543 million years ago due to enhanced vision**



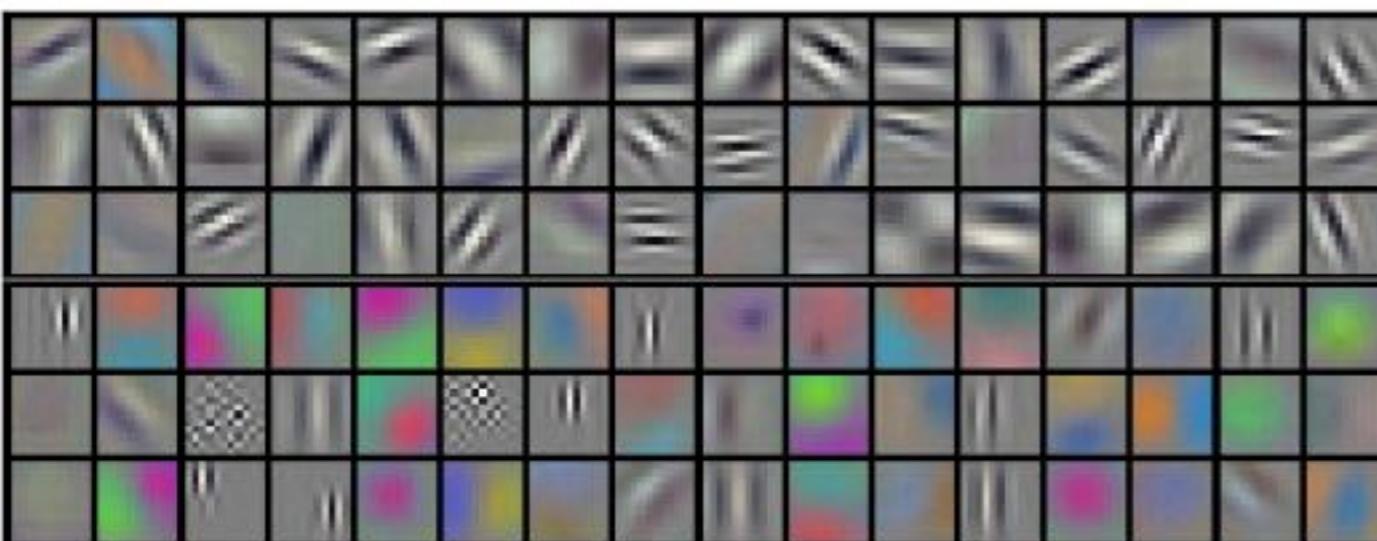


## Convolution (with a kernel values that are parameters to be fit)

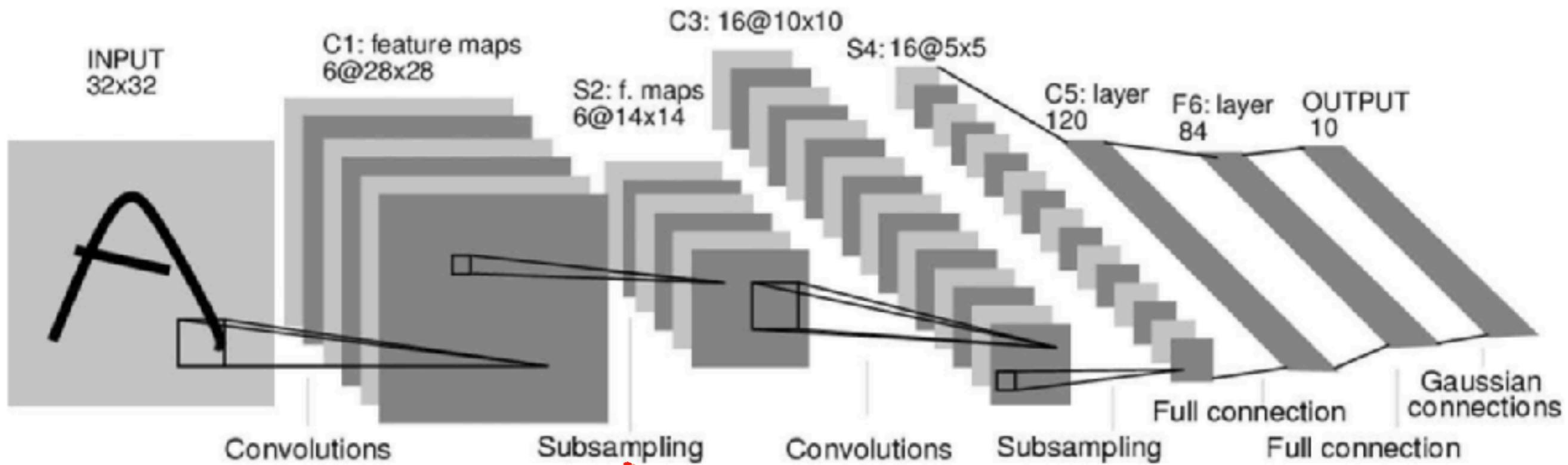




**feature kernels**



**flattening  
and  
fully connected  
neural network**



**pooling (max, min, mean...)**

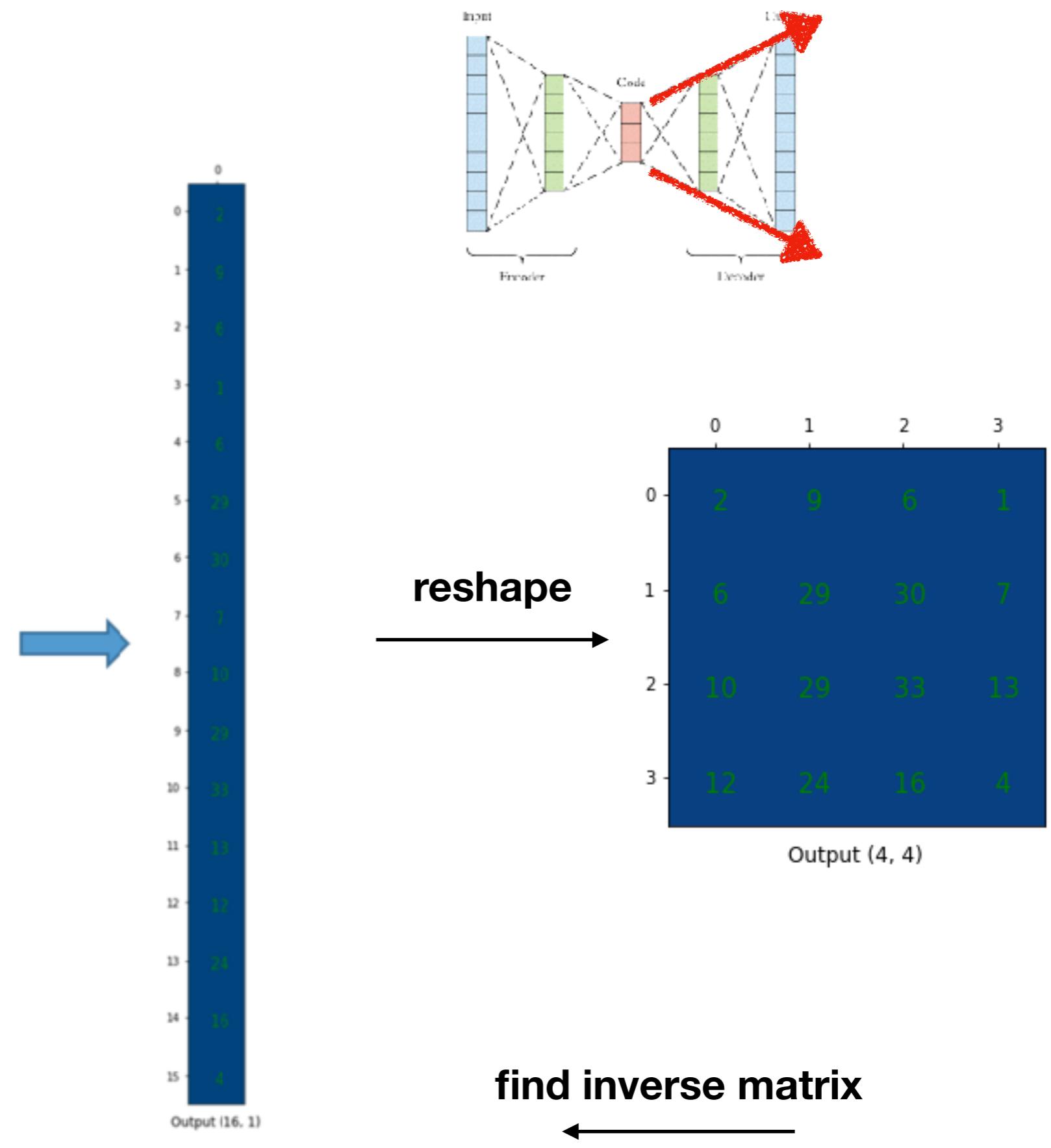
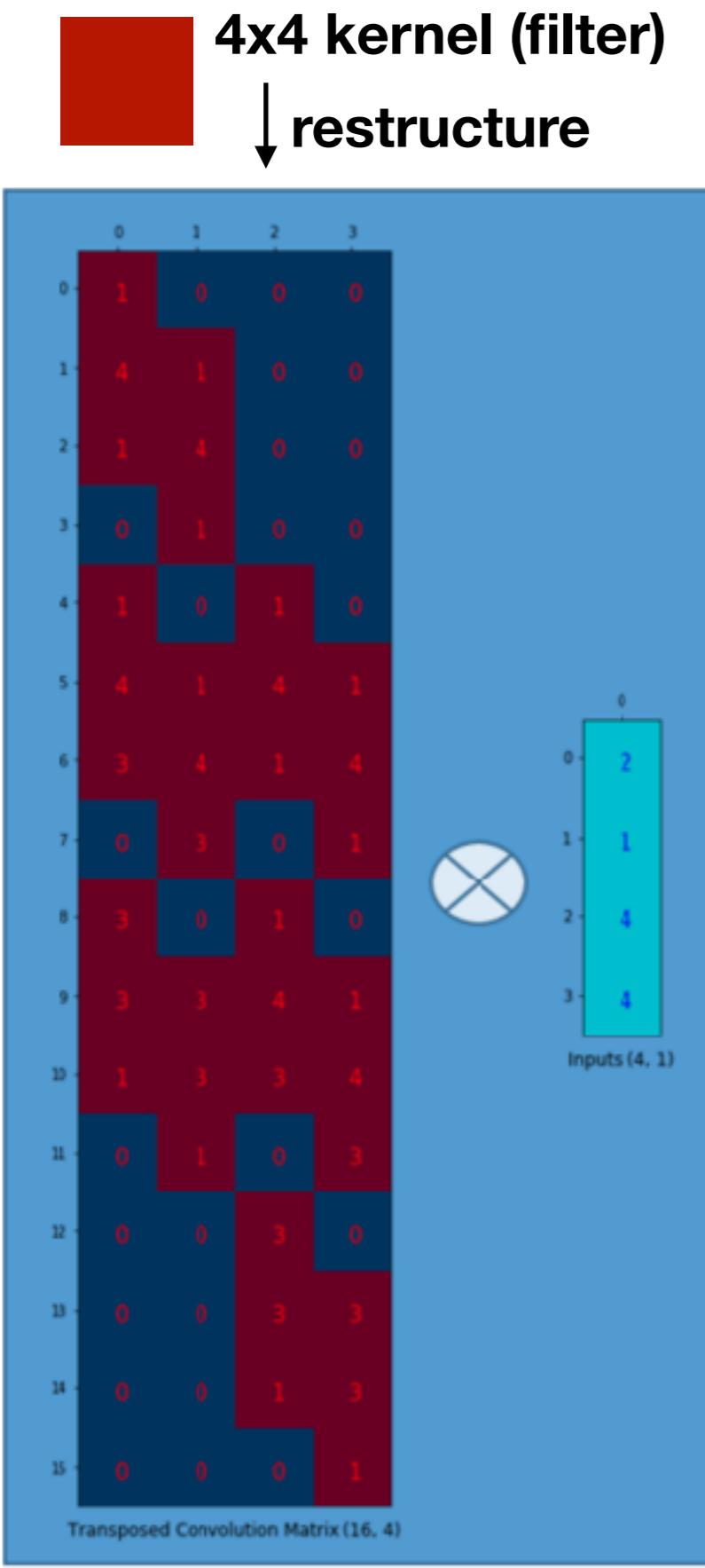
**steps in convolution can be also used**

12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

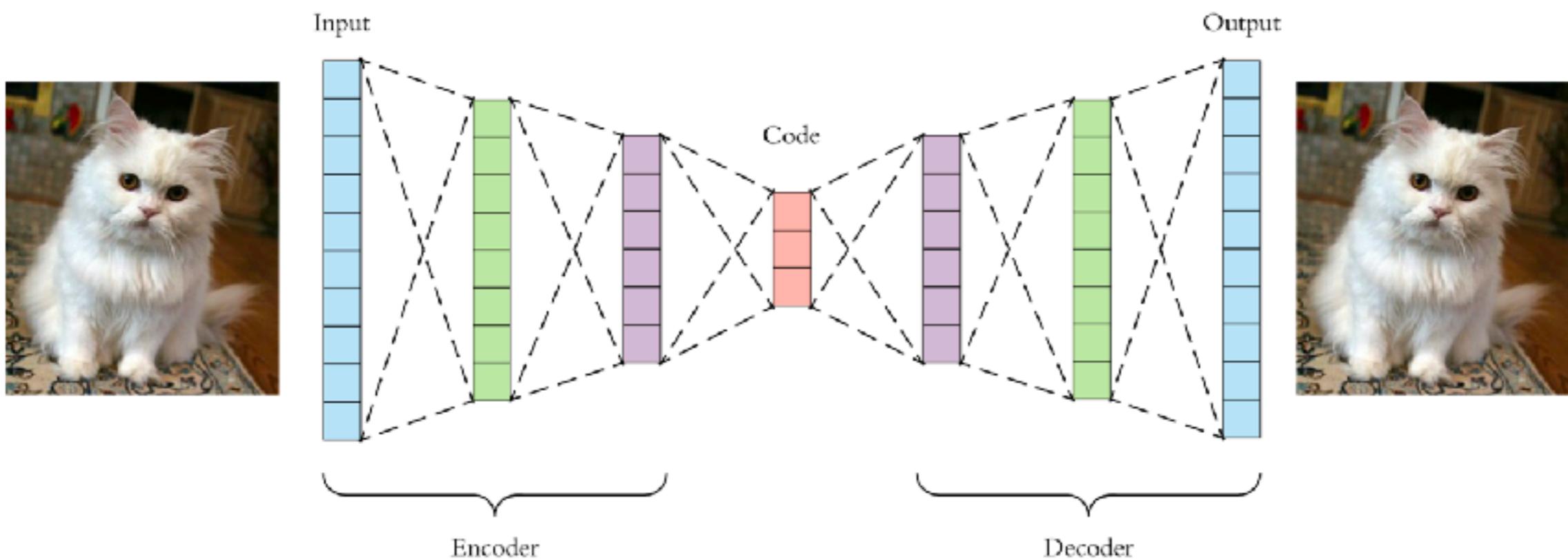
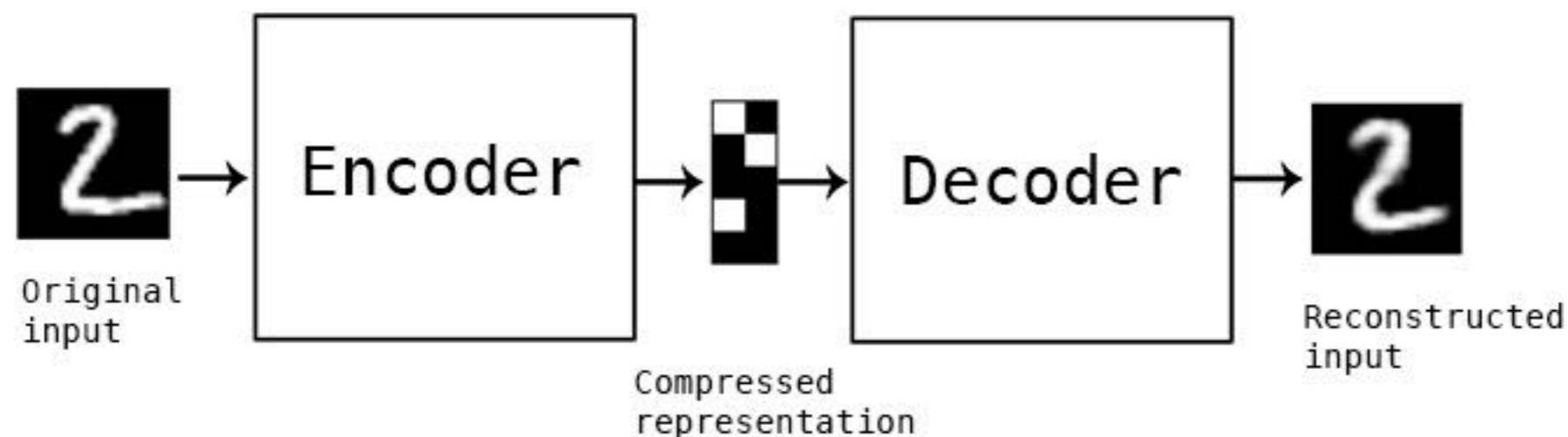
$2 \times 2$  Max-Pool

20	30
112	37

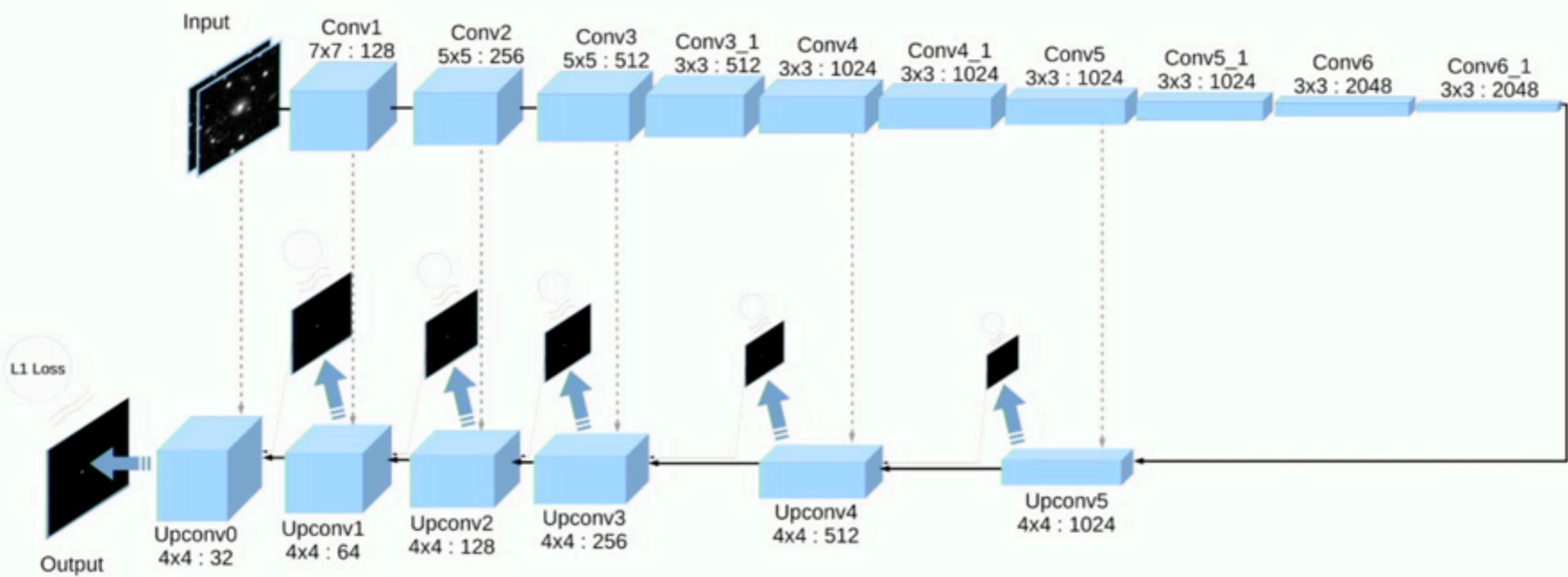
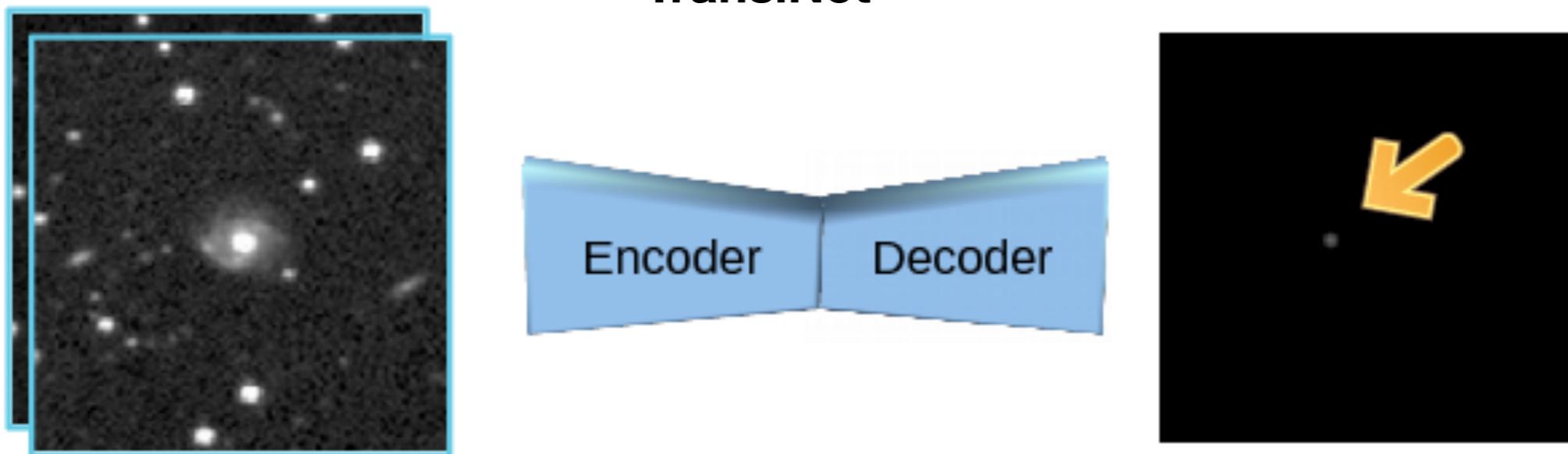
# Transposed convolution (up or inverse convolution)



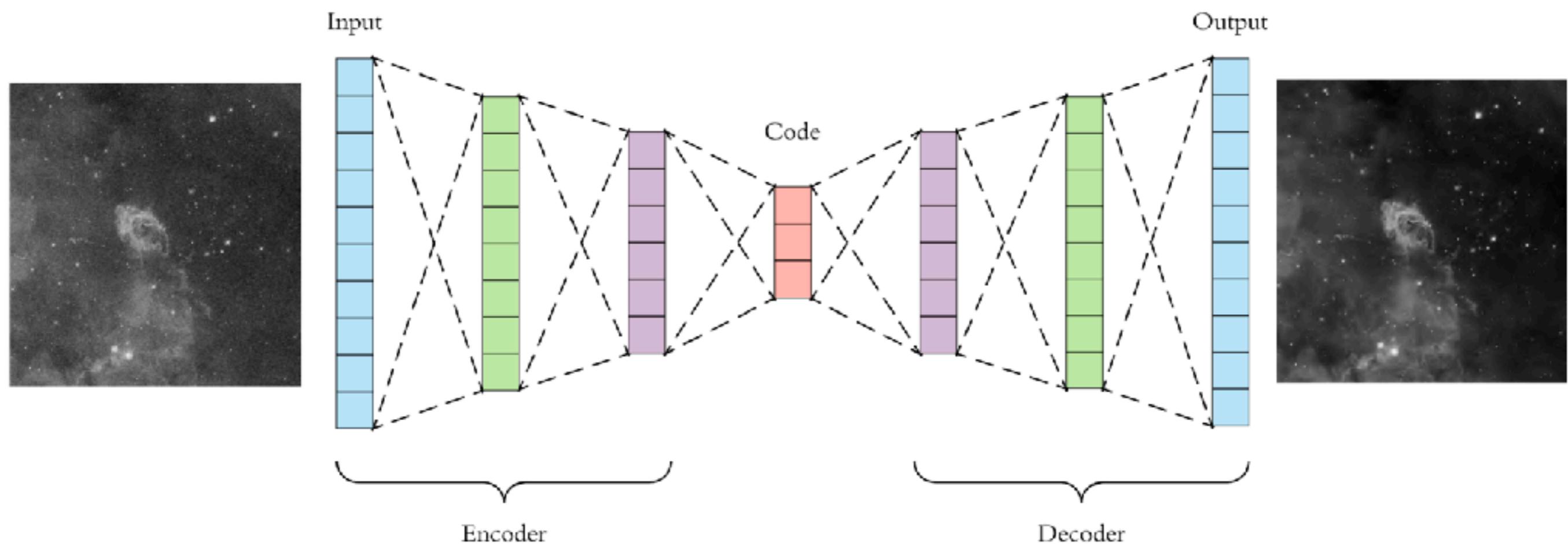
# Encoder-decoders (Autoencoders)



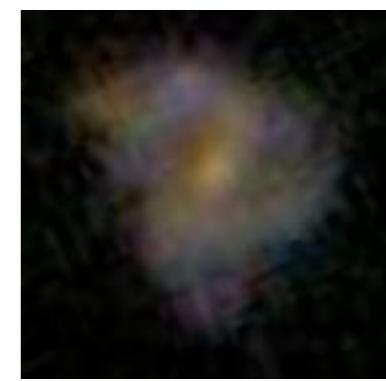
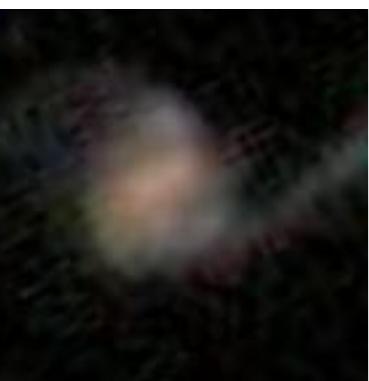
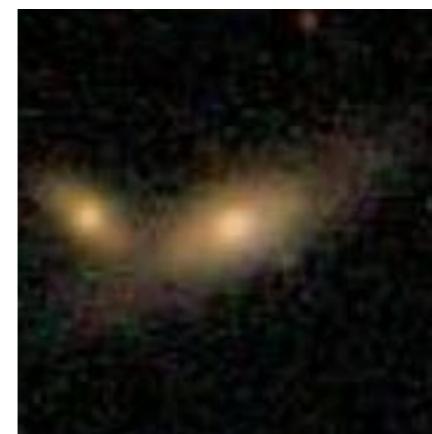
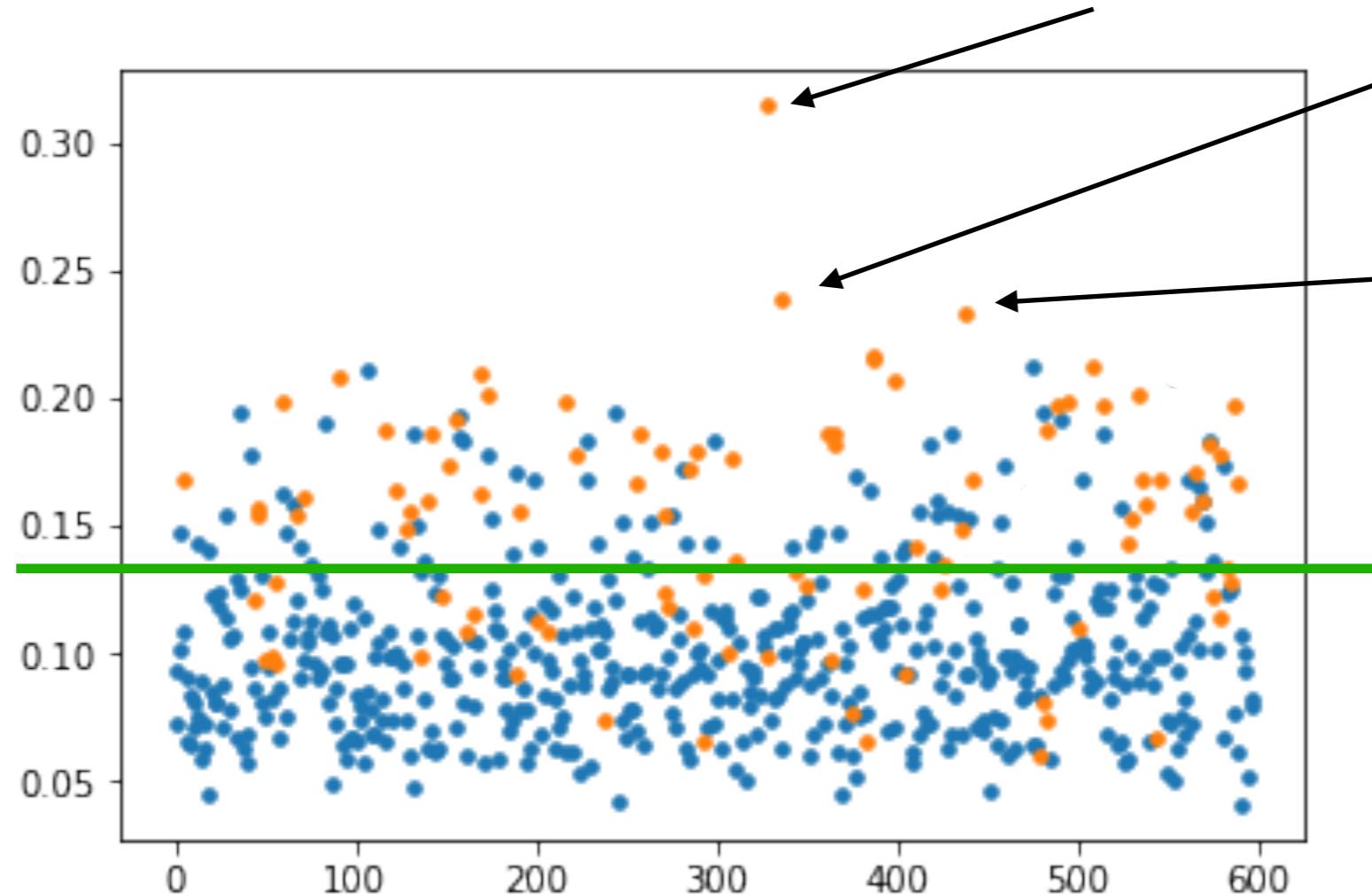
# TransiNet



# E.g. De-noising



# Anomaly detection



# Encoder/decoder usage

- Dimensionality reduction
- Denoising
- Outlier detection (measure of reconstruction error)

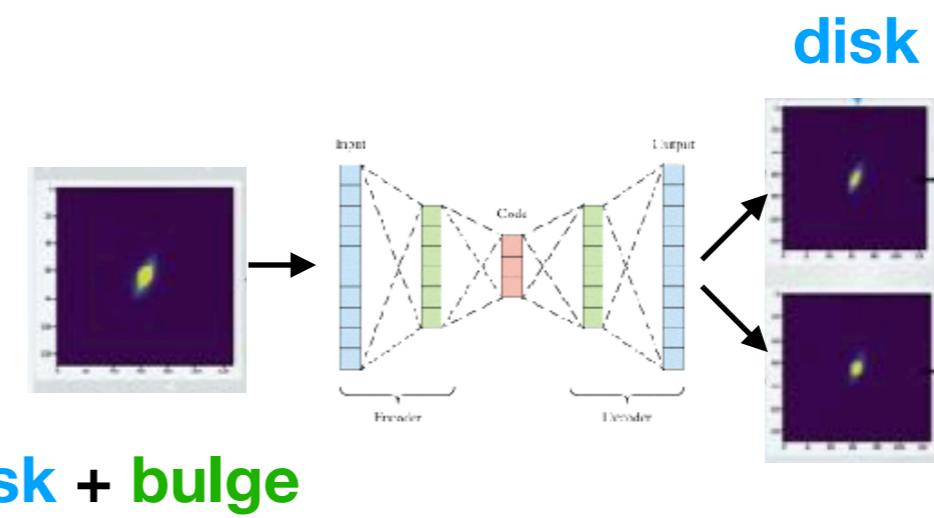
- Image segmentation

- Deblending

- Predicting next move

- Many others...

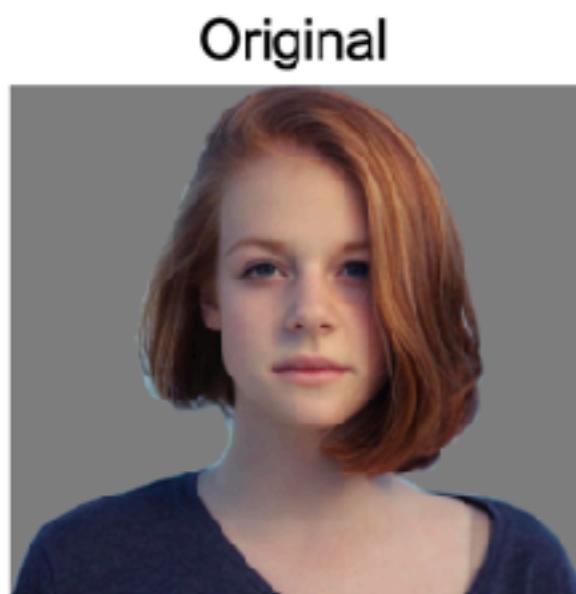
- (Can be considered as an unsupervised learning)



disk + bulge

bulge

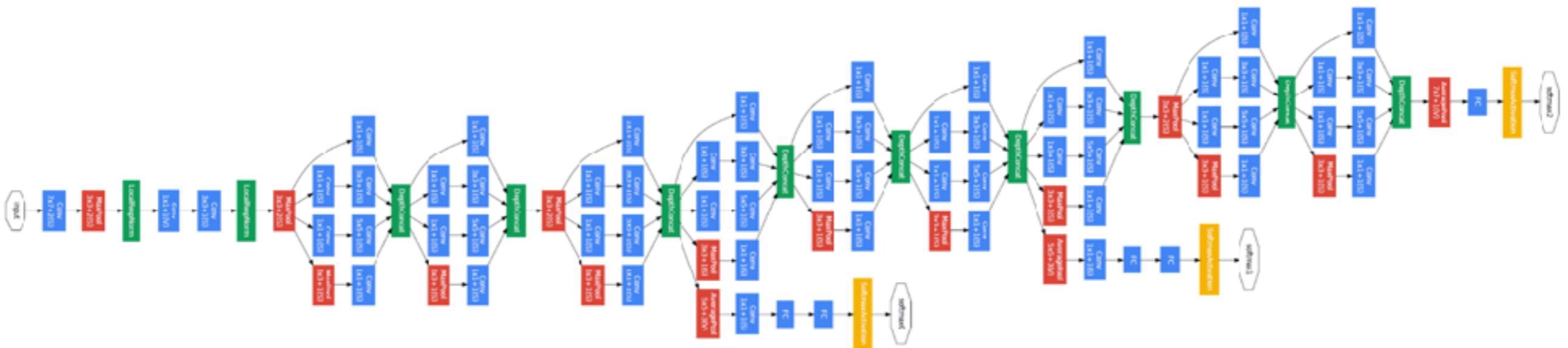
- Distance metrics on pixels are not informative



(all 3 images have same L2 distance to the one on the left)

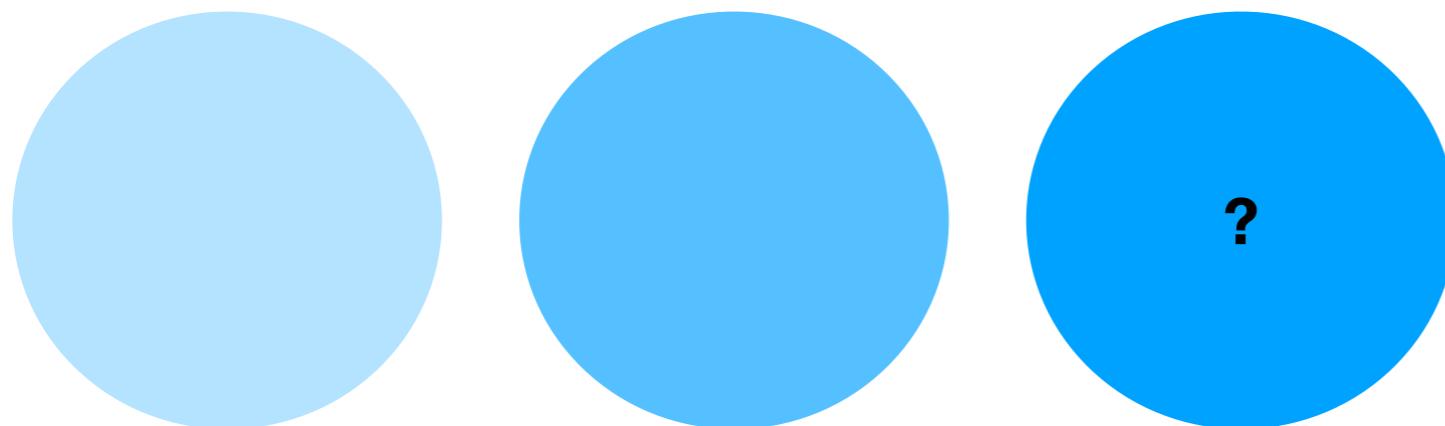
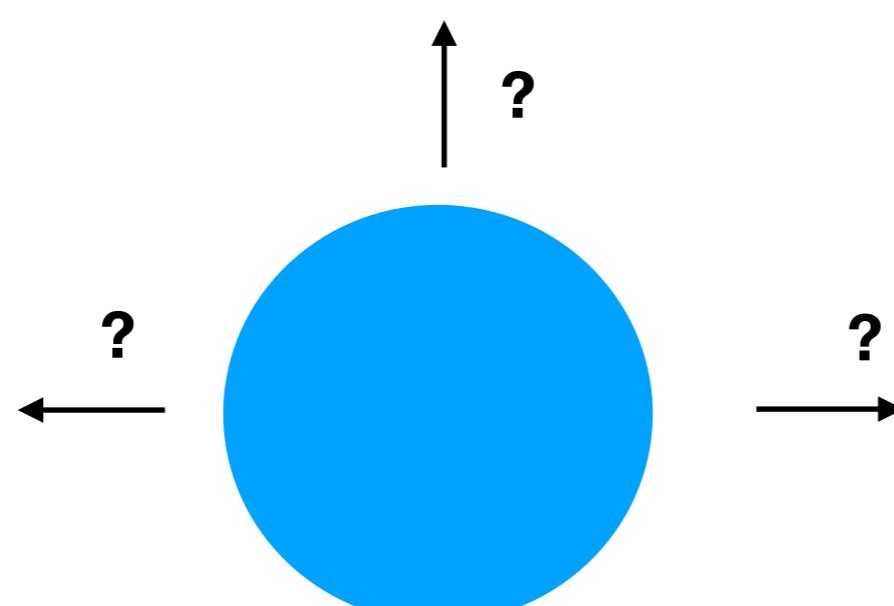
Original image is  
CC0 public domain

# Transfer Learning



<https://keras.io/api/applications/>

# Recurrent Neural Network



# Recurrent Neural Network

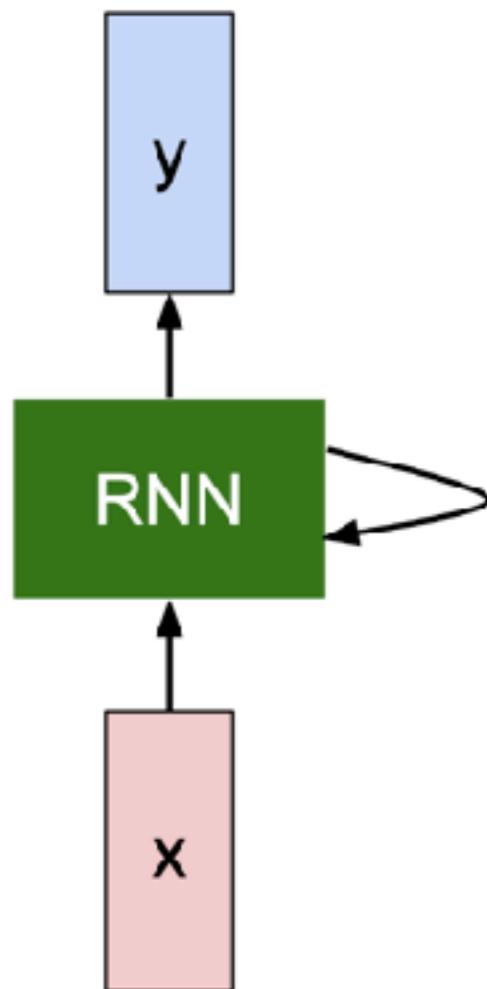
We can process a sequence of vectors  $\mathbf{x}$  by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

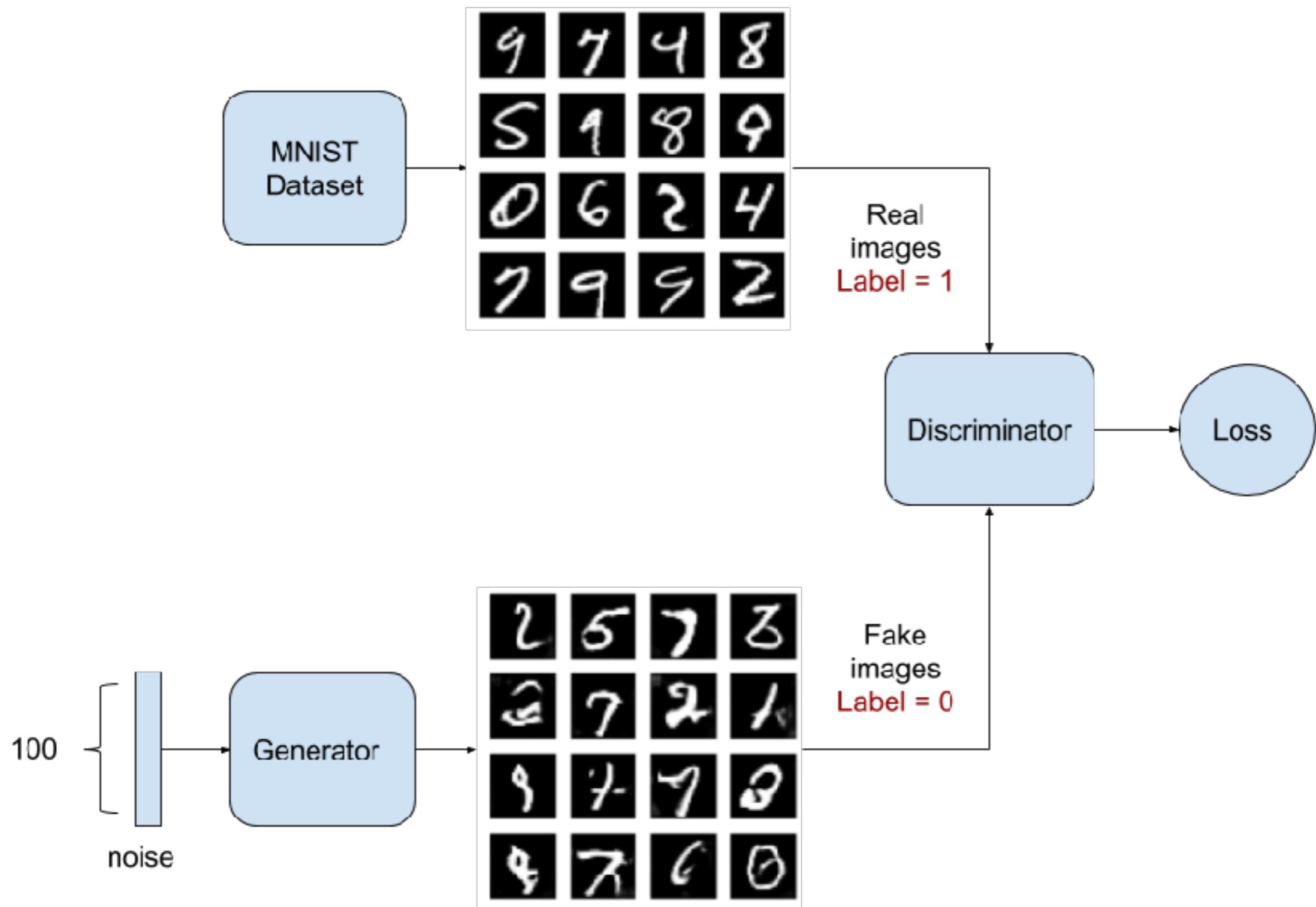
new state      old state      input vector at  
some function      with parameters  $W$       some time step

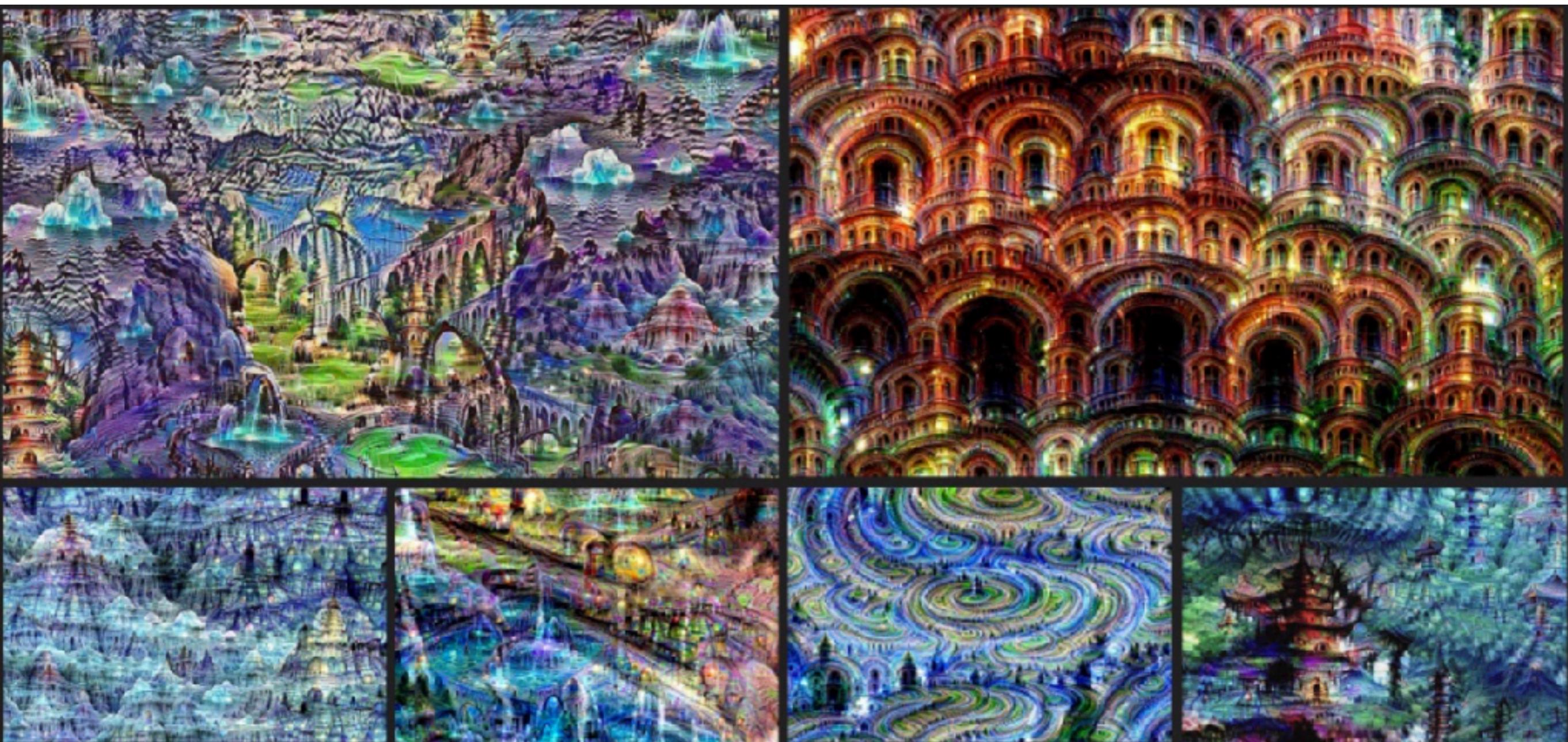
RNN pseudo-code

```
rnn = RNN()  
ff = FeedForwardNN()  
hidden_states = [0,0,0,0]  
  
for word in input:  
    output, hidden_state = rnn(word, hidden_state)  
  
    prediction = ff(output)
```

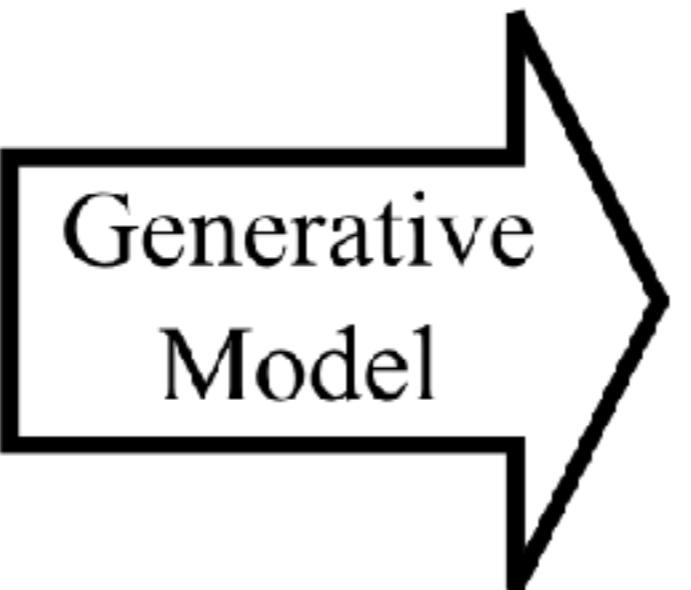
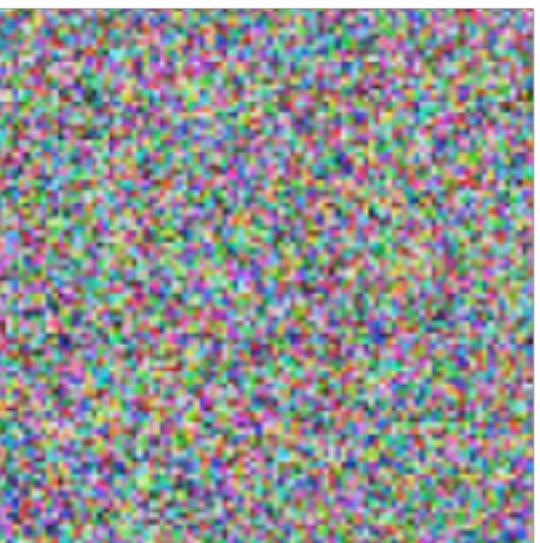


# GAN (Generative Adversarial Network)





Noise  $\sim N(0,1)$





Elliptical



Barred Spiral



Spiral



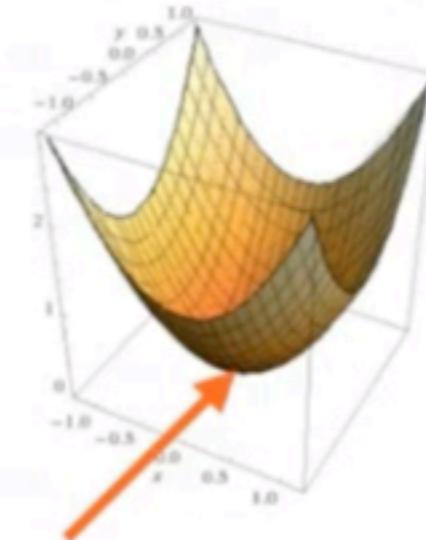
# GAN usage

- Mock catalogs
- Denoising
- Compete gaming

**Warning: Hard to train**

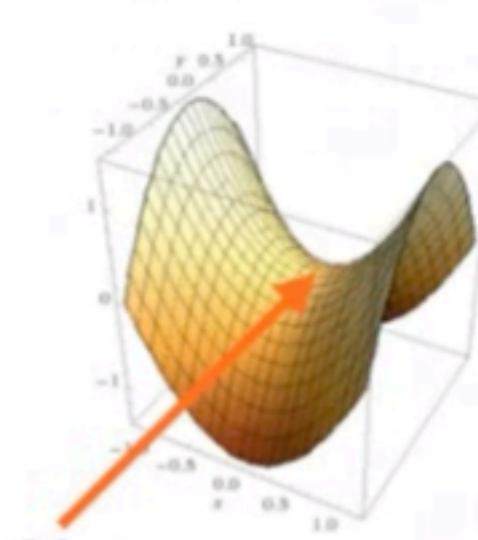
## Adversarial Machine Learning

Traditional ML:  
optimization



Minimum  
One player,  
one cost

Adversarial ML:  
game theory



Equilibrium  
More than one player,  
more than one cost



AI Research

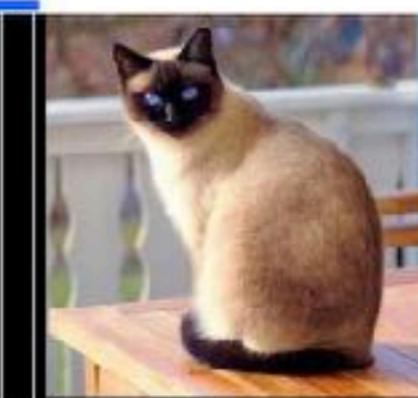
Research areas ▾

Publications

Experiments ▾

Careers

Blog



## 2. Simulate Attack

Adversarial noise type

C&amp;W Attack



Determine strength

None low med high

## 3. Defend attack

Gaussian Noise

None low med high

Spatial Smoothing

None low med high

Feature Squeezing

None low med high

Visual

Code

Original



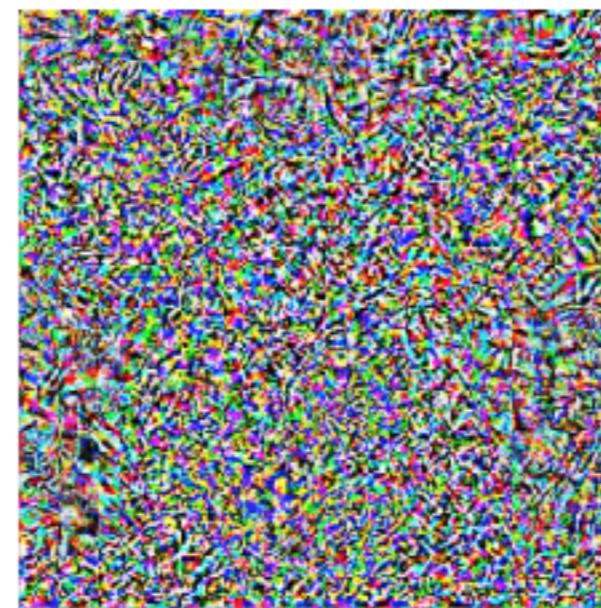
Modified



**“pig” (91%)**



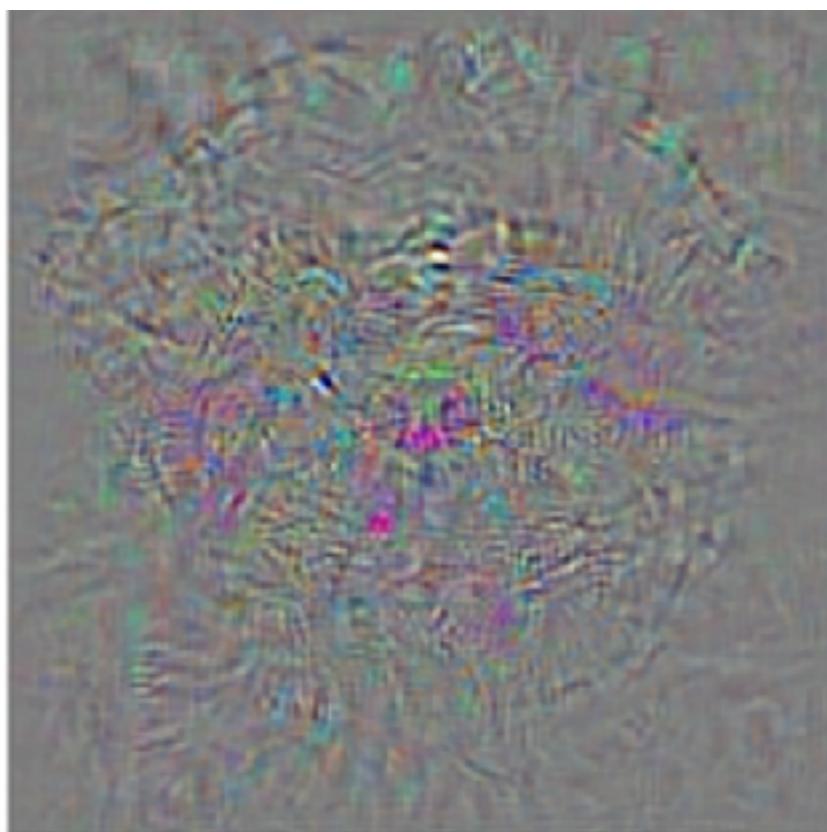
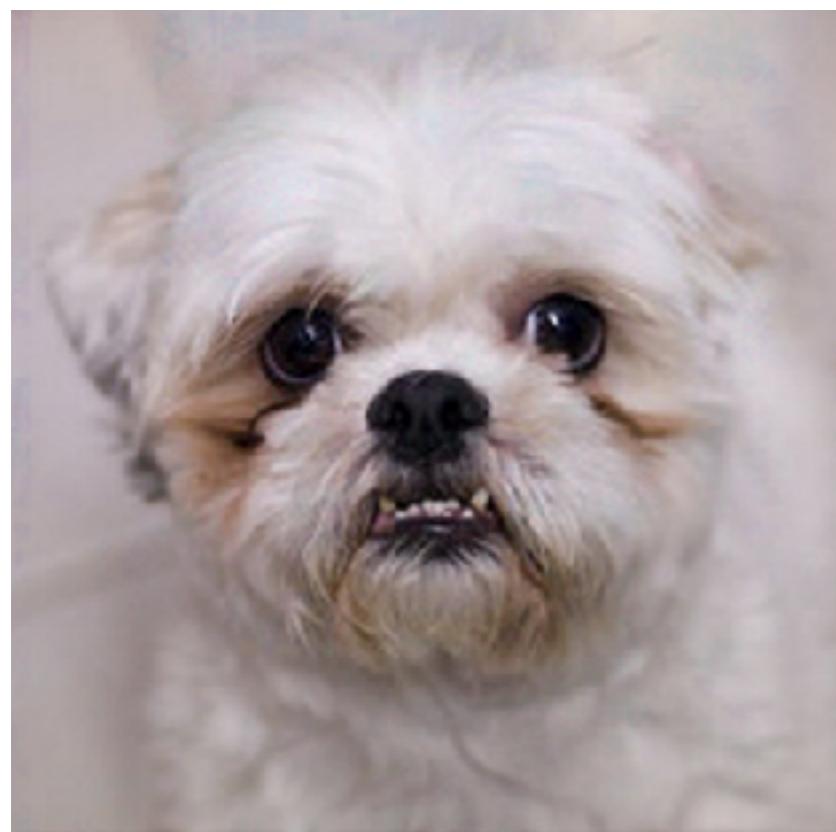
**noise (NOT random)**



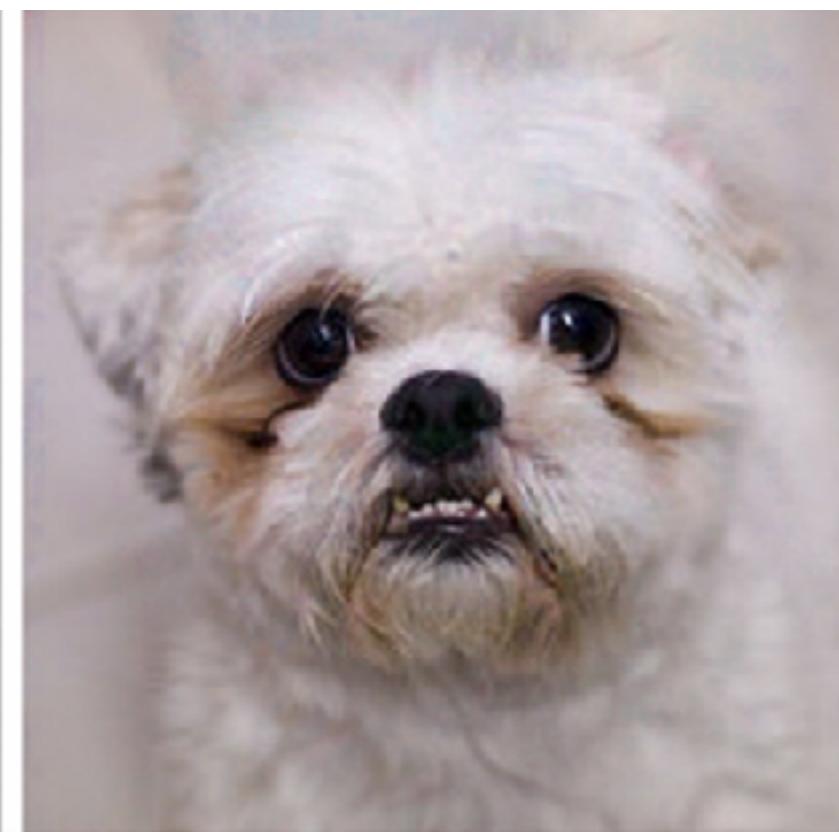
**“airliner” (99%)**



**+ 0.005 x**



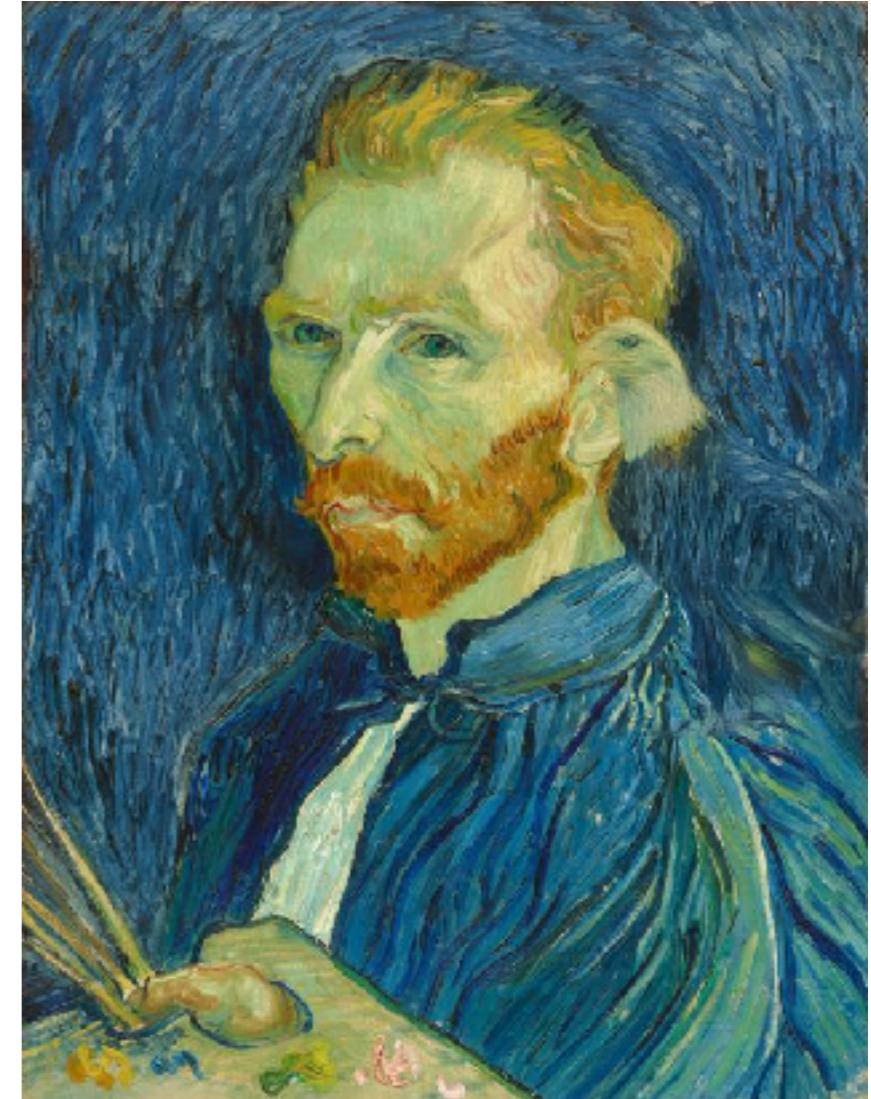
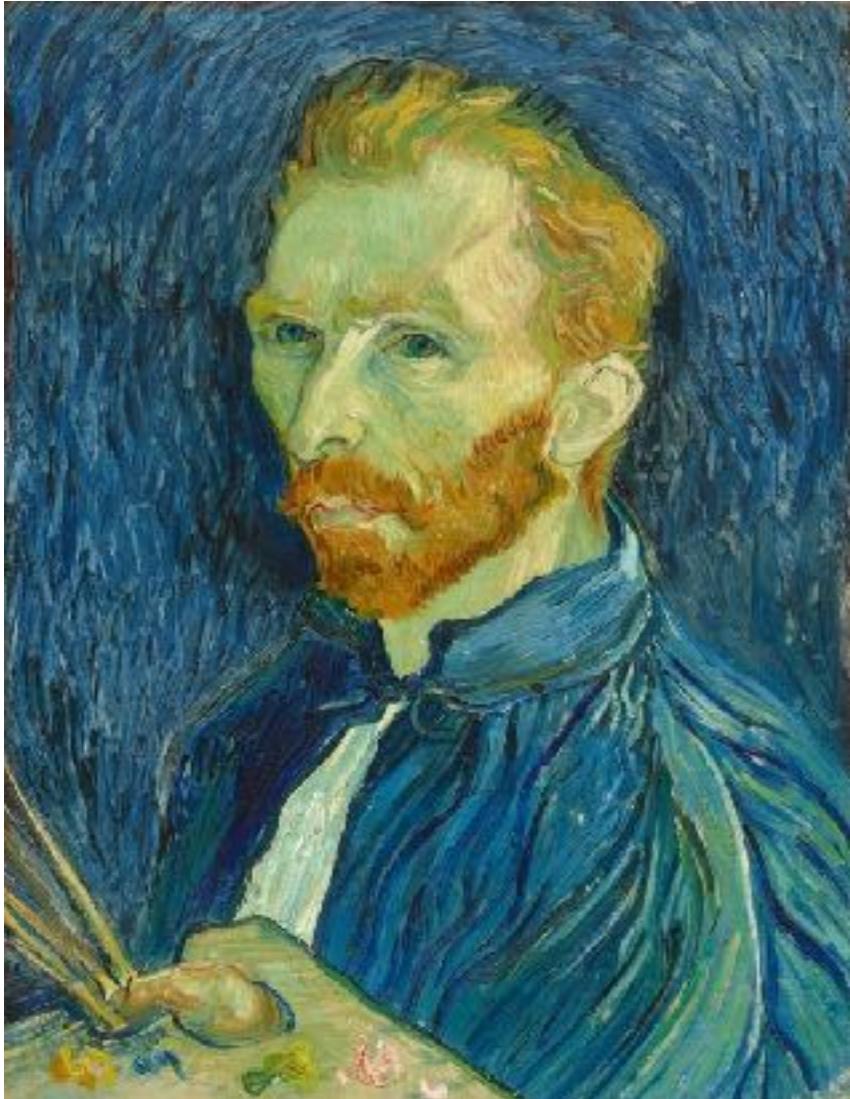
**=**



**dog**

**+noise**

**ostrich**



### Transfer style learning

<https://github.com/luanjunfj/deep-painterly-harmonization>

# Logic behind Keras

1- **Import** model class

```
from keras.models import Sequential
```

2 - **Instantiate** model class

```
model = Sequential()
```

3 - **Add layers** with the add() method specifying input\_dim or input\_shape

```
model.add(Dense(32, input_dim=784))
```

4 - Add activation functions

```
model.add(Activation('relu'))
```

5 - Configure **training** with compile(loss=,optimizer=, metrics[])

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy',
               metrics=['accuracy'])
```

6 - **Train** with the fit() method

```
model.fit(data, labels, epochs=10, batch_size=32)
```

7- **Evaluate** the model performance with the evaluate() method:

```
score = model.evaluate(x_test, y_test, verbose=0)
```

8 – Make **predictions** with predict():

```
predictions = model.predict(x_test)
```

in TensorFlow

in anaconda for python 3.6



cs321n — vim ~/Dropbox/lmaf/ffinder/scripts/simple\_nn.py — 94x28

```
# import keras
from tensorflow import keras

# get data
(train_images, train_labels), (test_images, test_labels) = keras.datasets.mnist.load_data()

# setup model
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer=keras.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# train model
model.fit(train_images, train_labels, batch_size=50, epochs=5)

# evaluate performance
test_loss, test_acc = model.evaluate(test_images, test_labels)
print("Test accuracy:", test_acc)

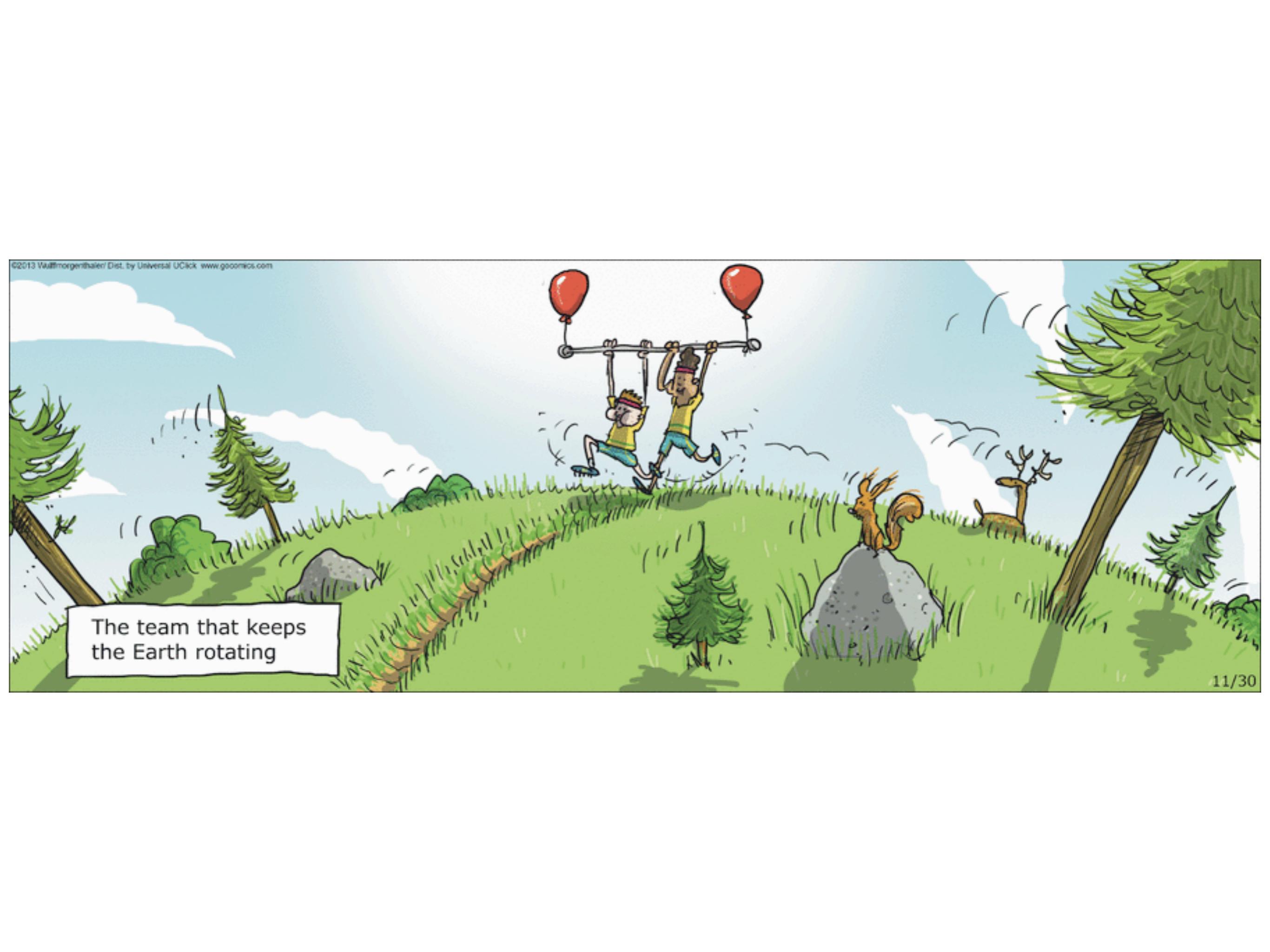
# make predictions
predictions = model.predict(test_images)
```

# Top 5 (supervised)

Algorithm	Comments
Neural Networks	<ul style="list-style-type: none"><li>• Take long to train - lot of CPU</li><li>• Overfits</li><li>• Requires lot of data</li></ul>
Gradient Boosted Trees	<ul style="list-style-type: none"><li>• Fast</li><li>• Overfit danger</li></ul>
Random Forest	<ul style="list-style-type: none"><li>• Robust to overfitting</li></ul>
SVM w/non-linear kernel	<ul style="list-style-type: none"><li>• Pretty good</li></ul>
Gaussian Processes	<ul style="list-style-type: none"><li>• non-parametric fitting</li></ul>



**Deep learning can identify the  
patients gender with 95% accuracy!**



The team that keeps  
the Earth rotating