

Lecture #5

F5611 Machine Learning for Astronomers
by Martin Topinka

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

Lecture by *Ashish Mahabal* (Caltech)

15 Dec 2020

ZTF: Classification of transients at Zwicky Transient Factory + Hands-On-Session

Have ready (colab)

Python modules python 3

- tensorflow
- numpy
- matplotlib
- scikit-learn

Additional Resources

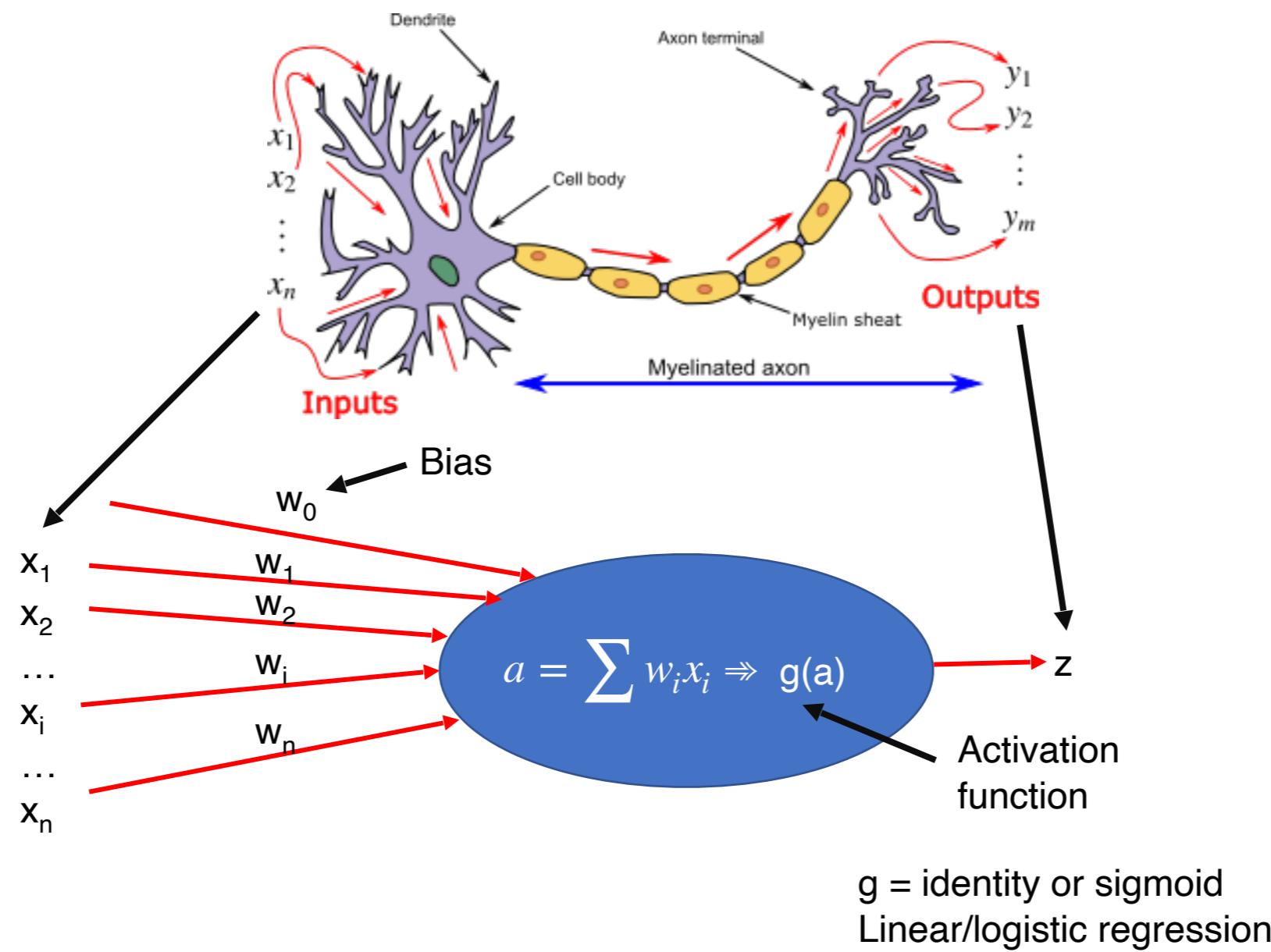
<http://cs231n.stanford.edu>

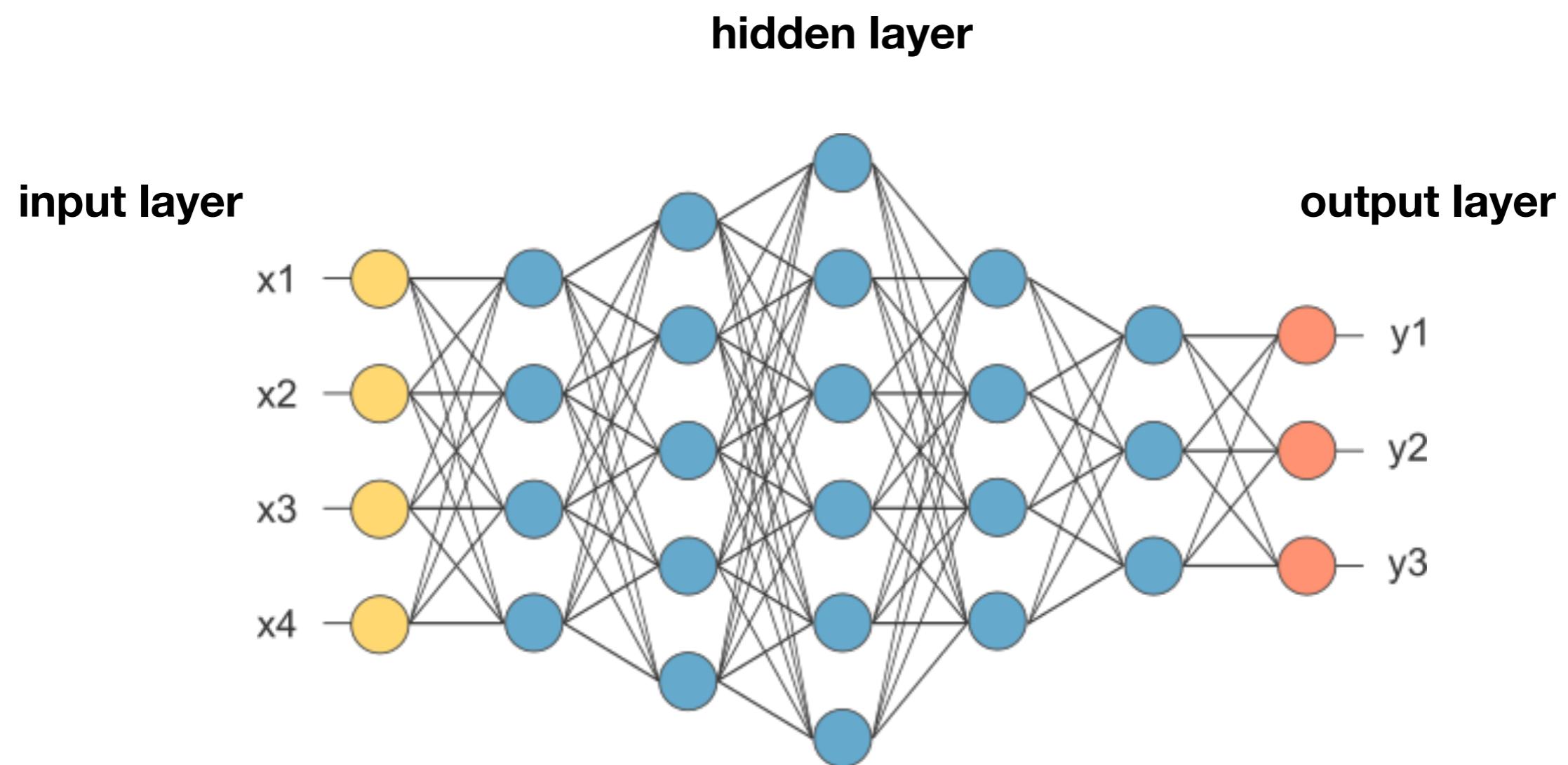
Outline

- Introduction to **Neural Network**: the idea behind, SGD, back-propagation, vanishing gradient, fighting over-fitting)
- **Convolutional Neural Network** – next year by Matej!
- Encoder-decoders, **Auto-Encoders**, **Variational Auto-Encoders**
- **Recurrent Neural Networks**, **Generative Adversarial Networks**
- TensorFlow/Keras
- Hands-on session: NN star/quasar classification, NN Fourier Transform, ConvNN source finding/denoising



Boom of species 543 million years ago due to enhanced vision

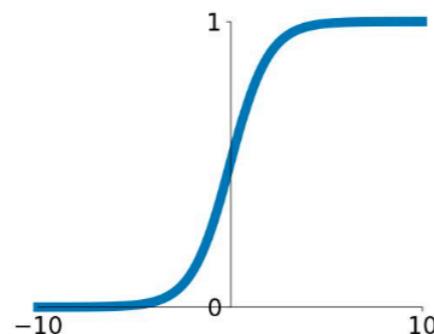




Activation functions

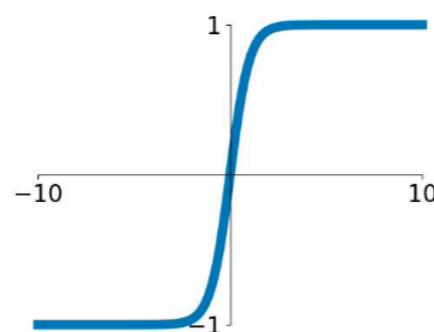
Sigmoid

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



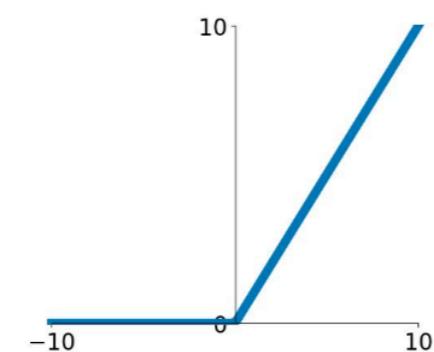
tanh

$$\tanh(x)$$



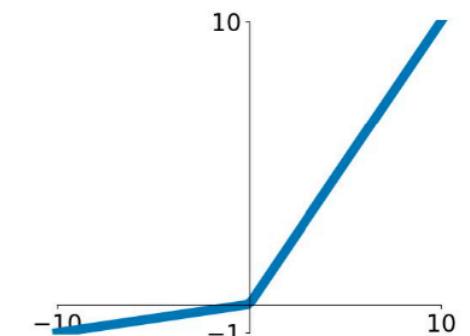
ReLU

$$\max(0, x)$$



Leaky ReLU

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

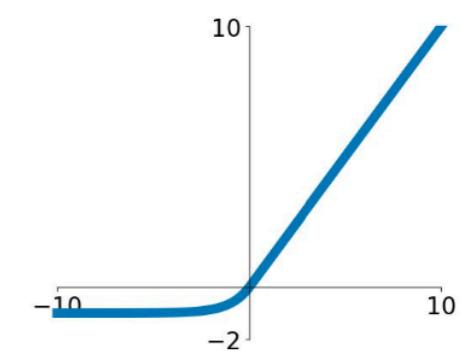


Maxout

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

ELU

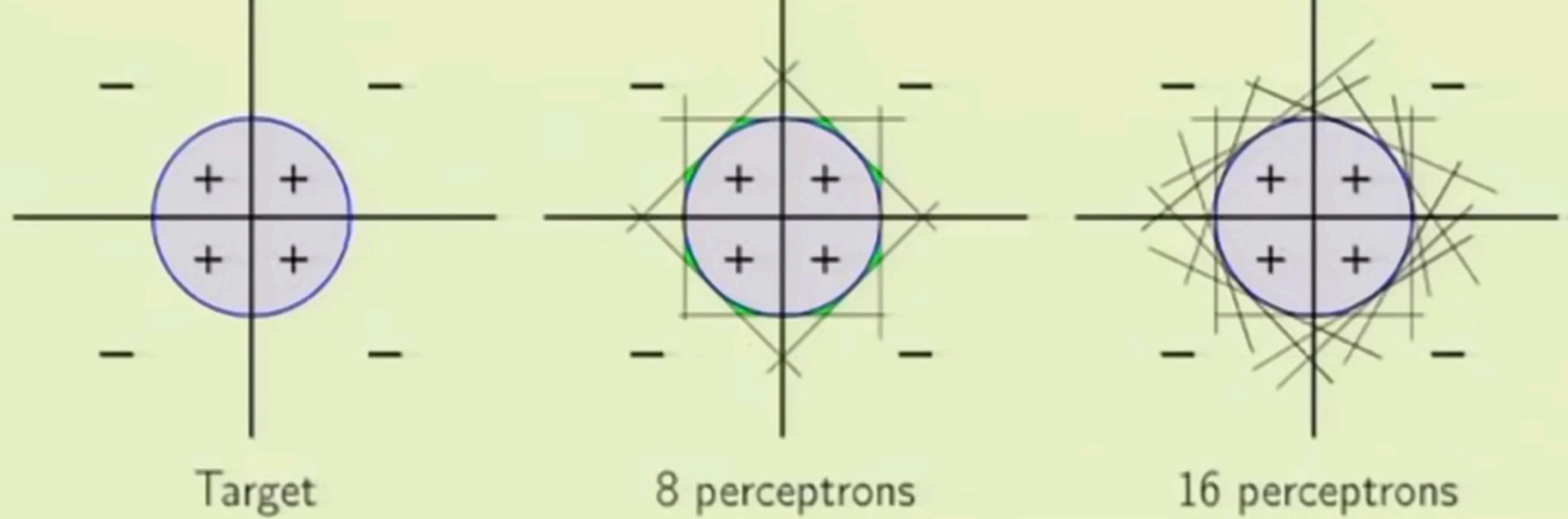
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



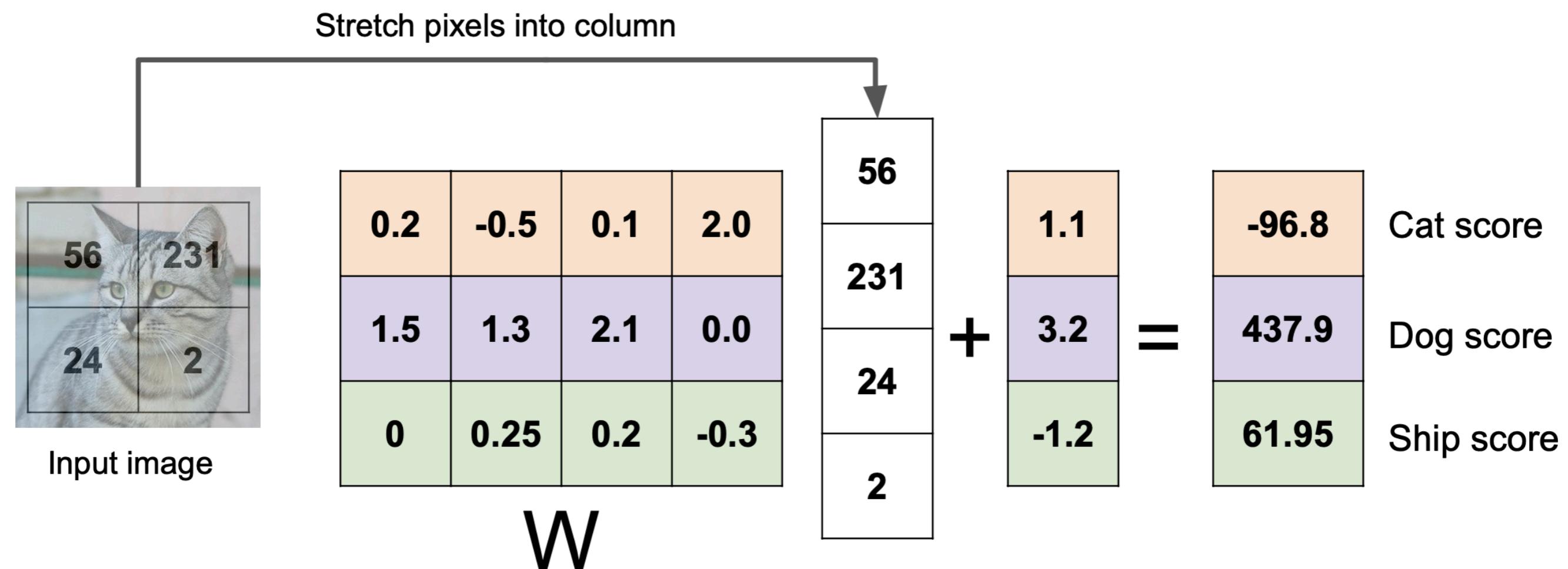
Neural Network

Linear
classifiers

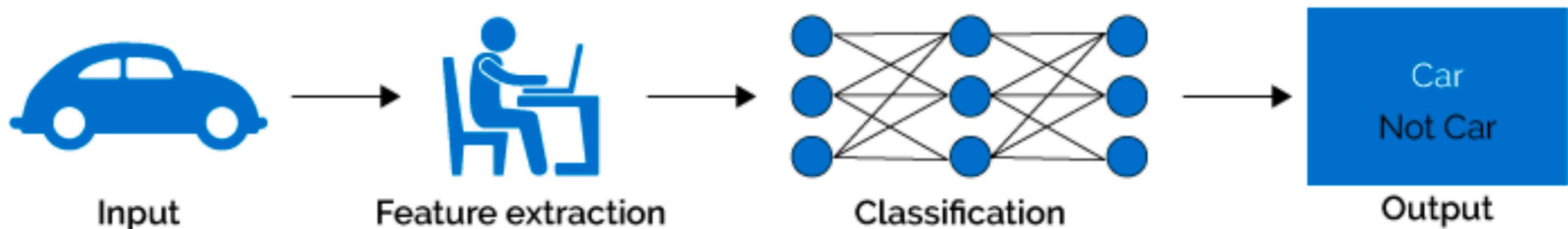




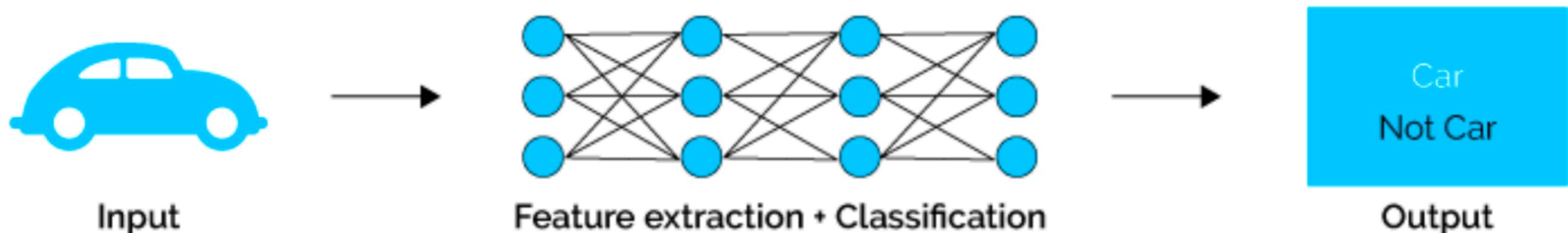
Example with an image with 4 pixels, and 3 classes (**cat/dog/ship**)



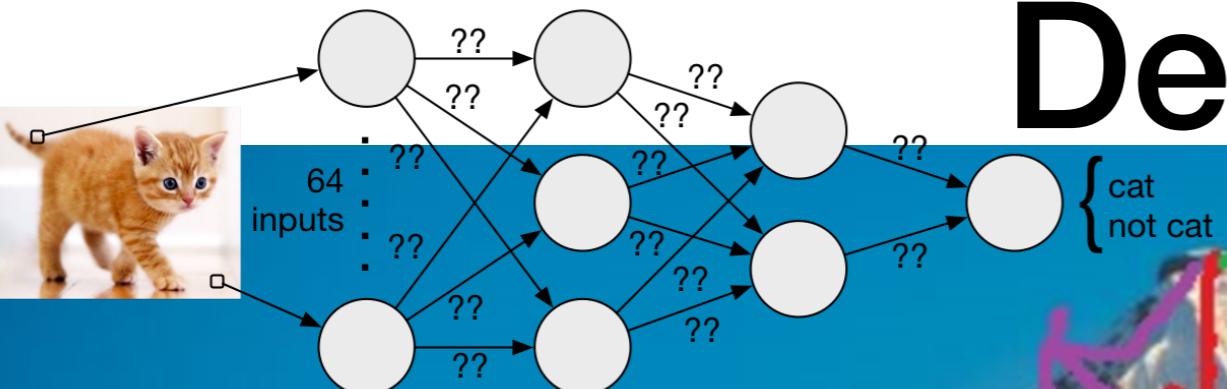
Machine Learning



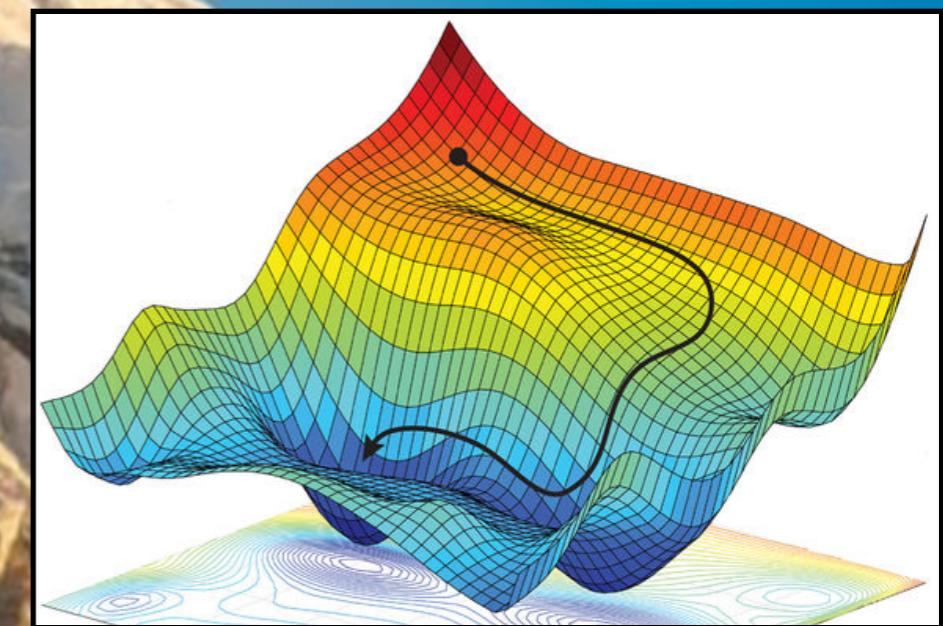
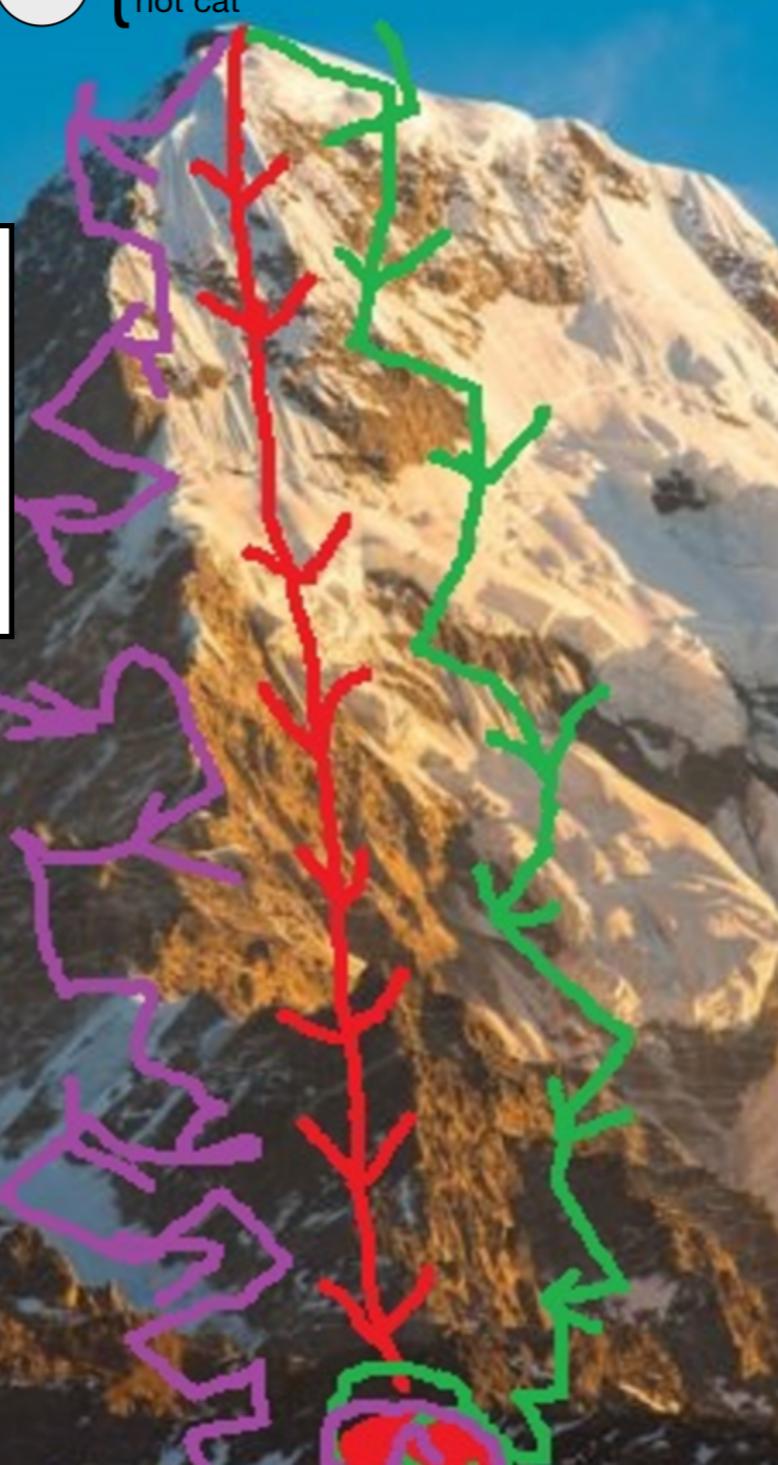
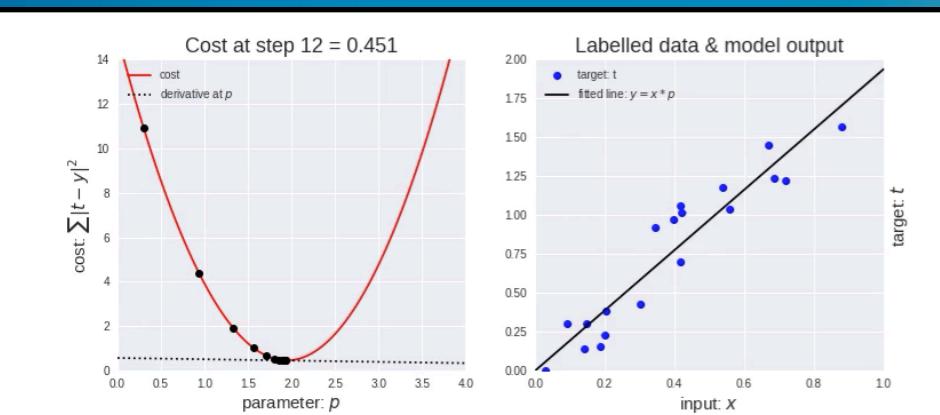
Deep Learning



(Stochastic) Gradient Descent



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



Stochastic Gradient Descent (SGD)

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

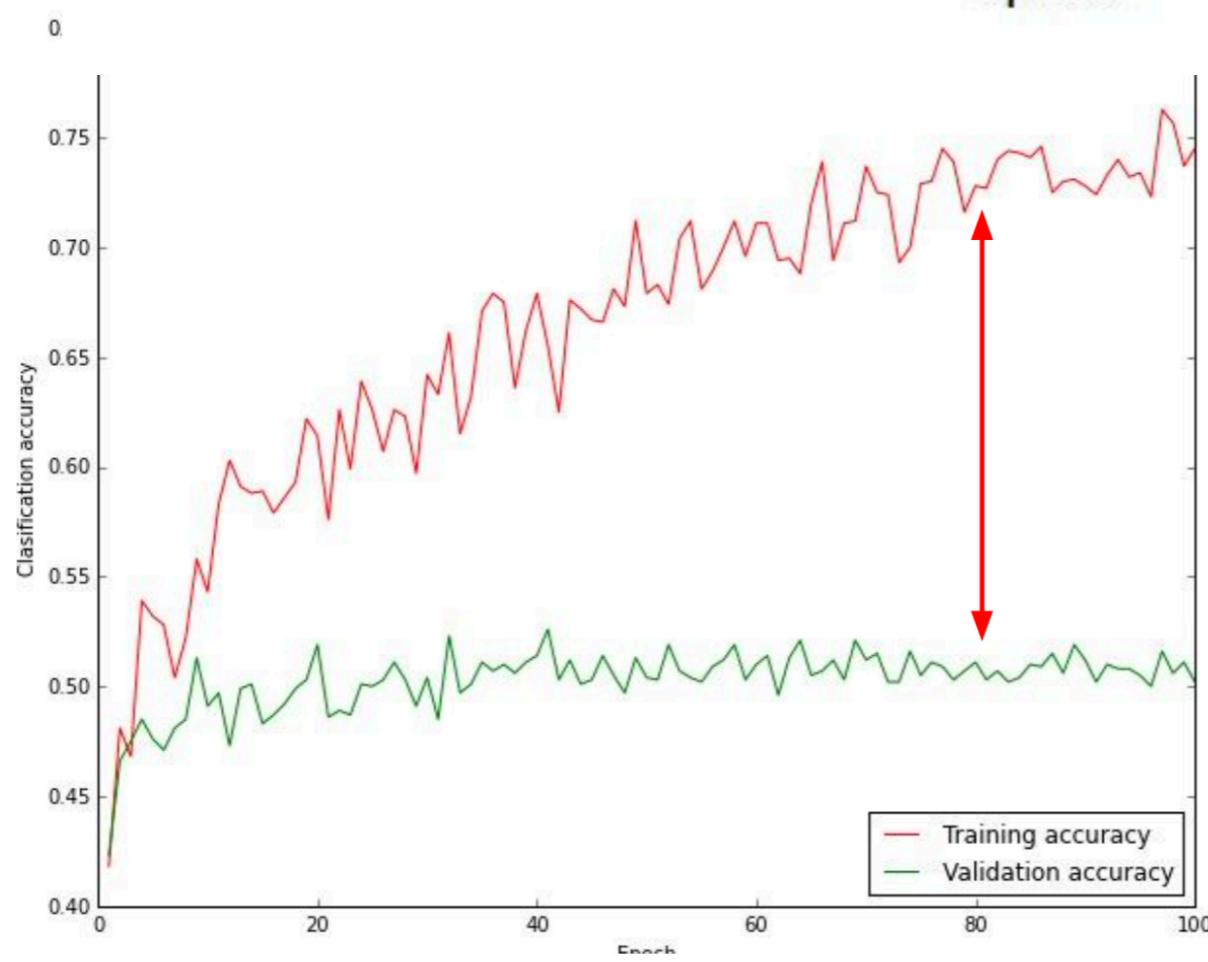
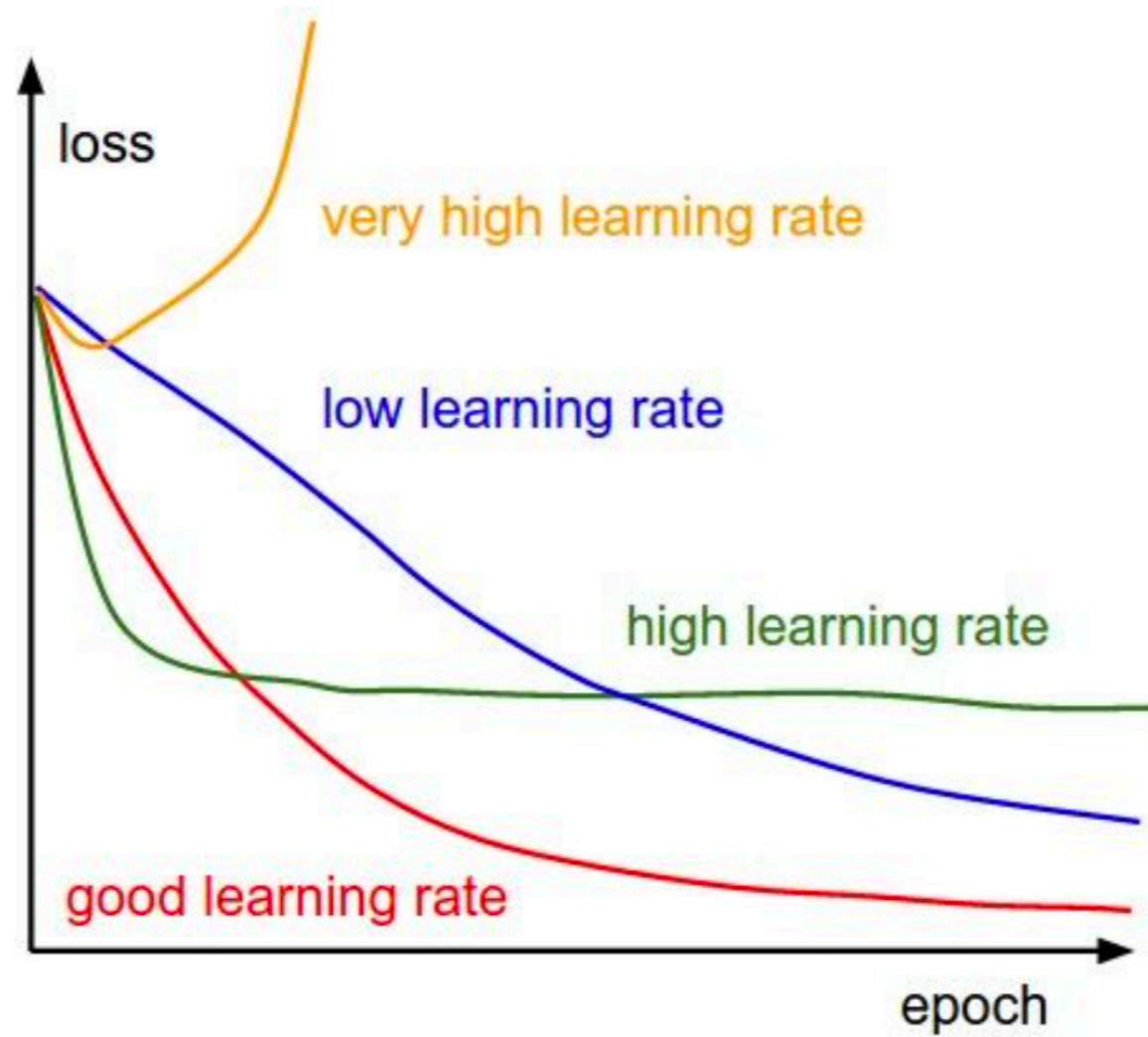
$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Full sum expensive
when N is large!

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
```



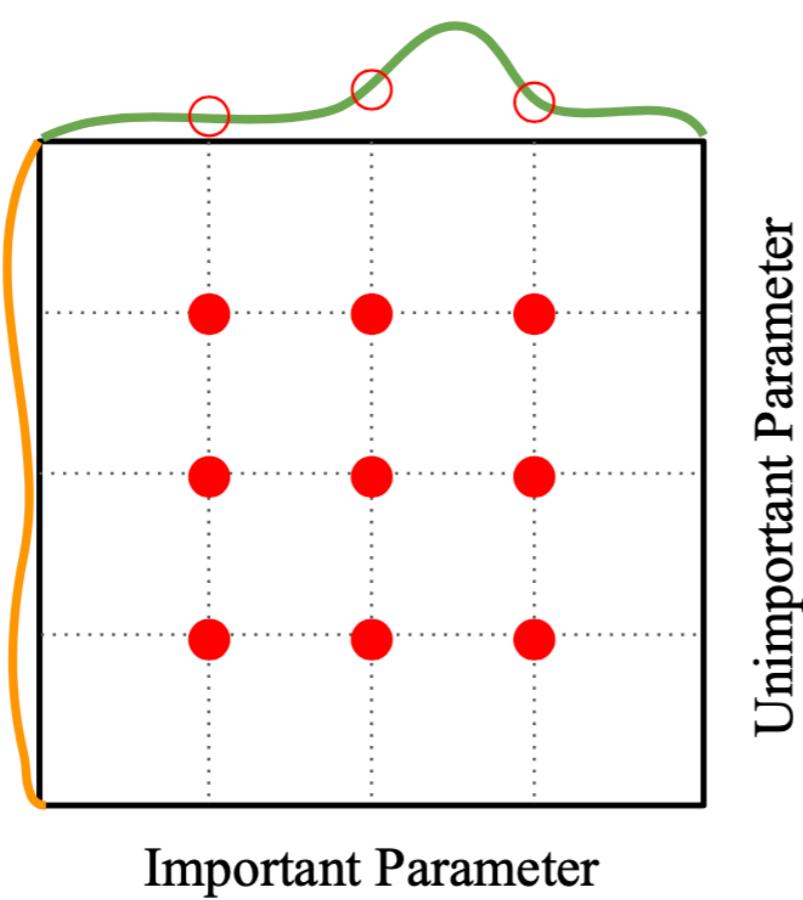
big gap = overfitting
=> increase regularization strength?

no gap
=> increase model capacity?

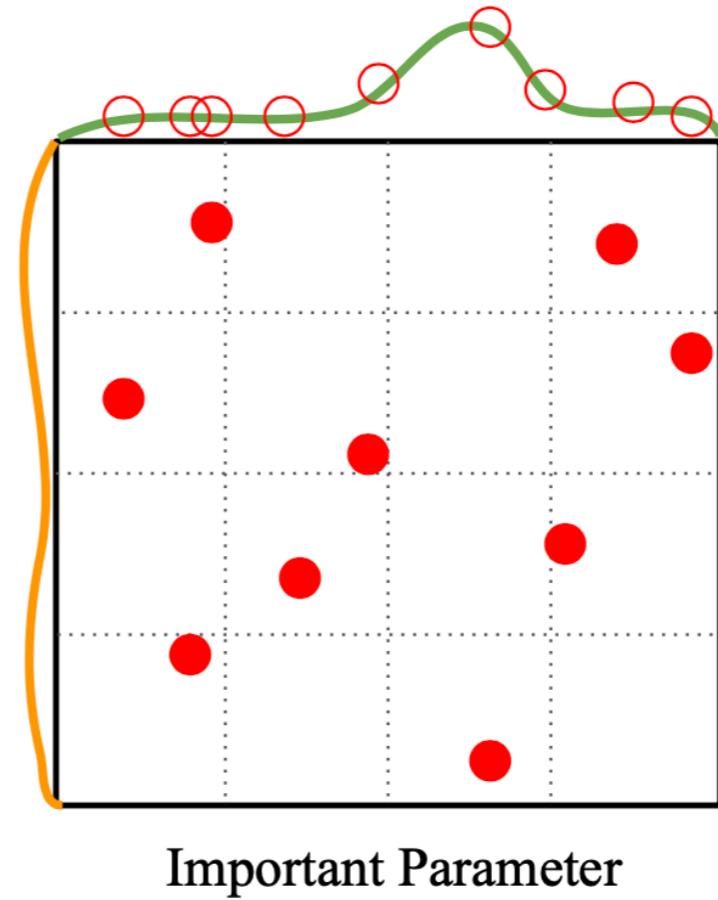
Random Search vs. Grid Search

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

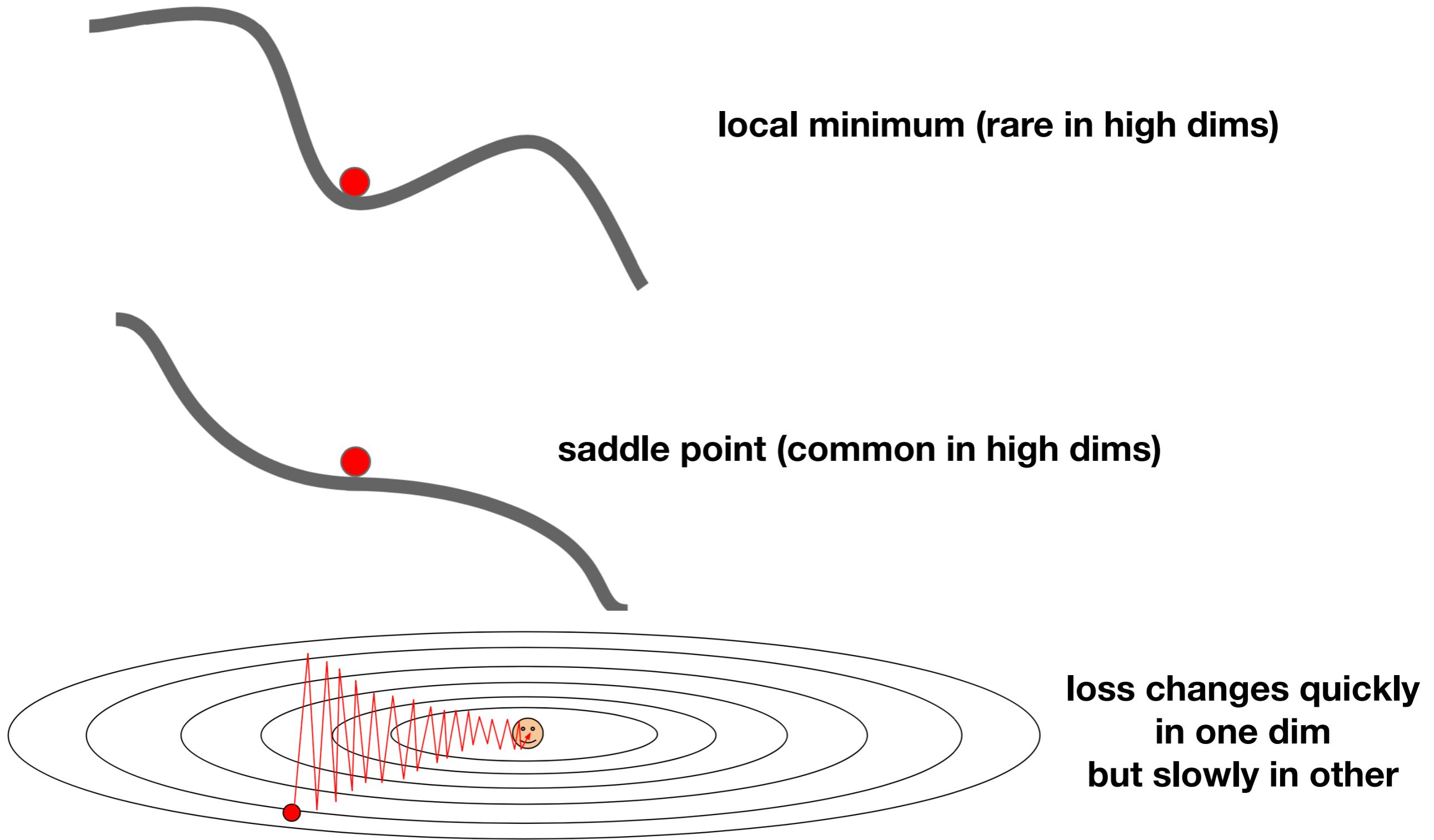
Grid Layout



Random Layout



Problems with Gradient Descent



SGD

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

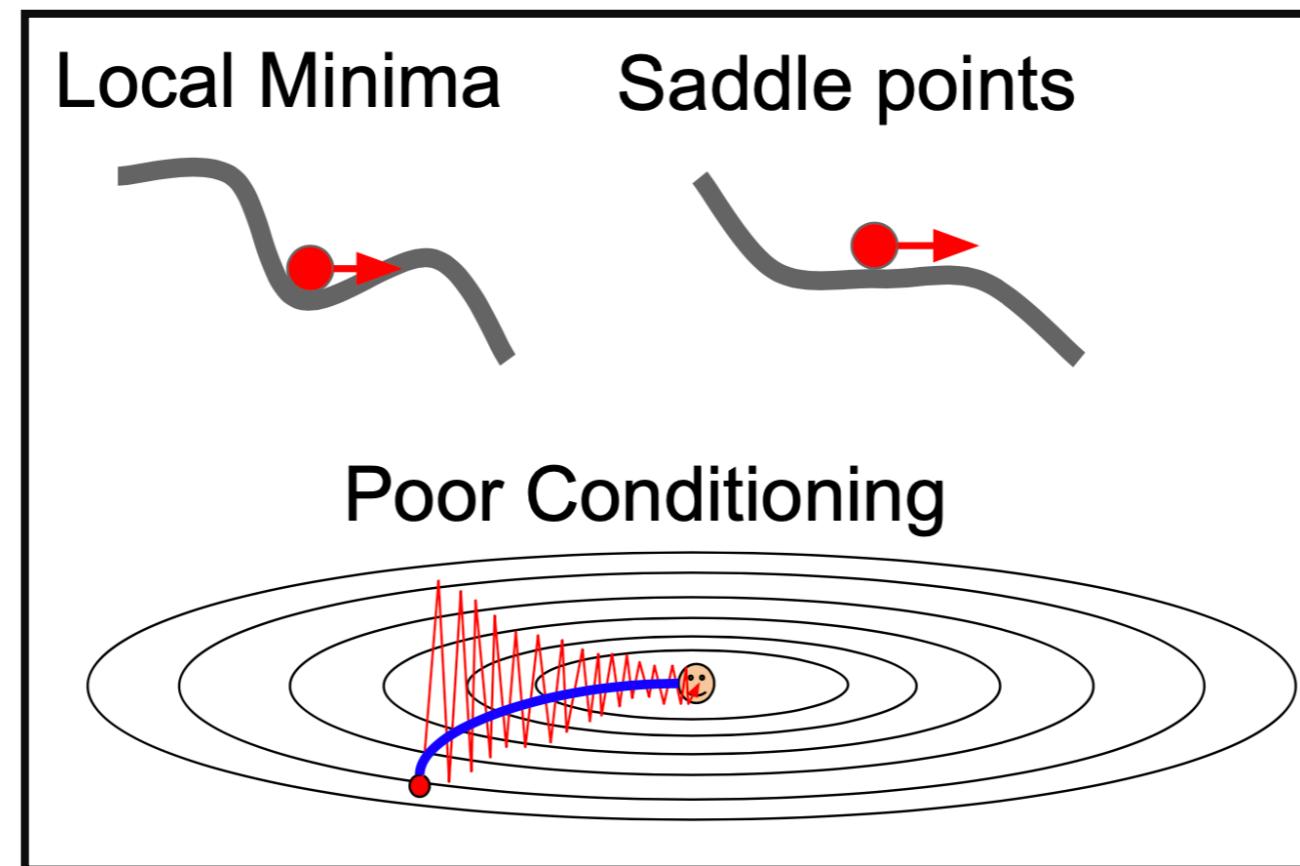
```
while True:  
    dx = compute_gradient(x)  
    x += learning_rate * dx
```

SGD+Momentum

$$\begin{aligned}v_{t+1} &= \rho v_t + \nabla f(x_t) \\x_{t+1} &= x_t - \alpha v_{t+1}\end{aligned}$$

```
vx = 0  
while True:  
    dx = compute_gradient(x)  
    vx = rho * vx + dx  
    x += learning_rate * vx
```

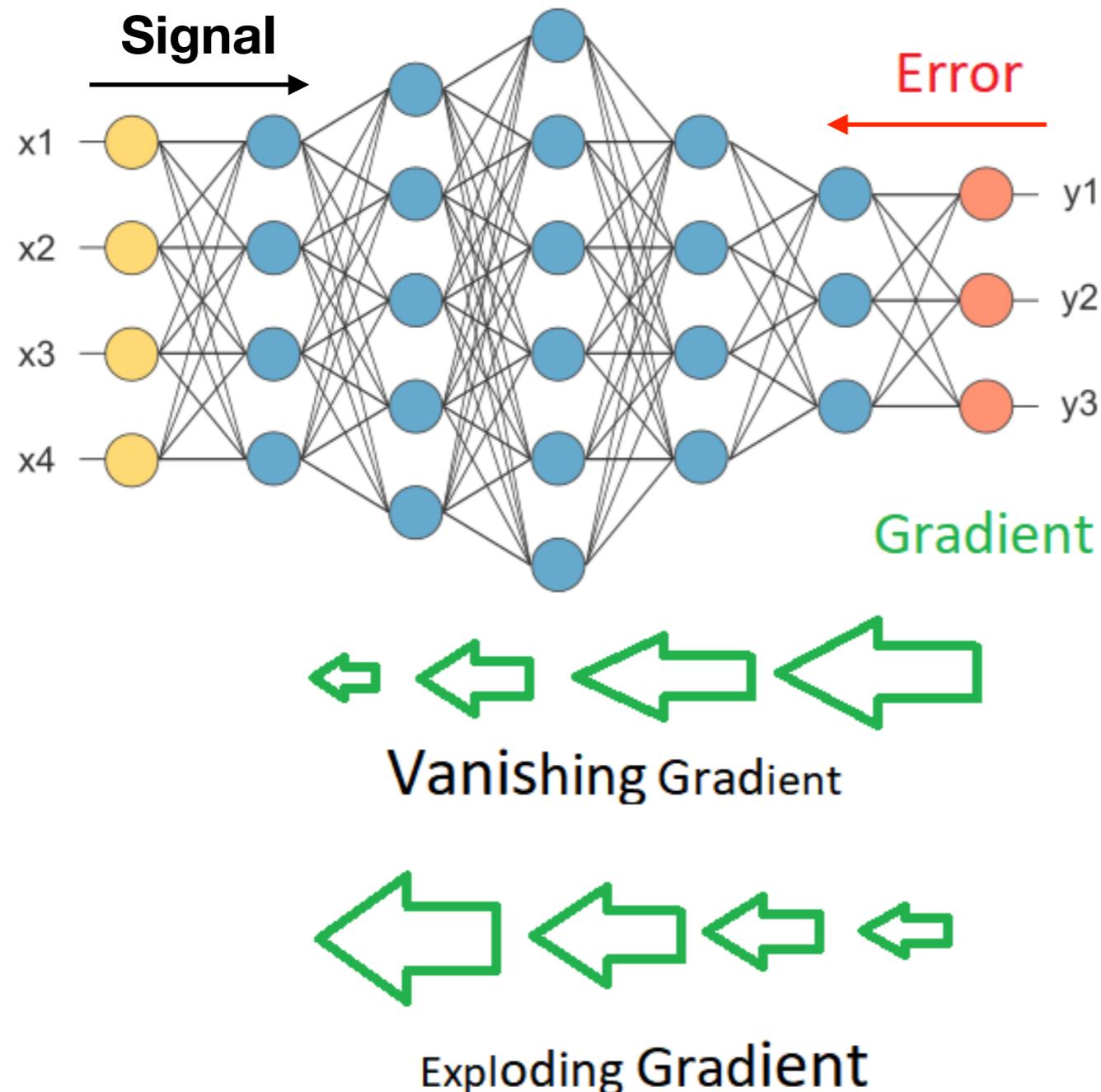
- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99



Back-propagation

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

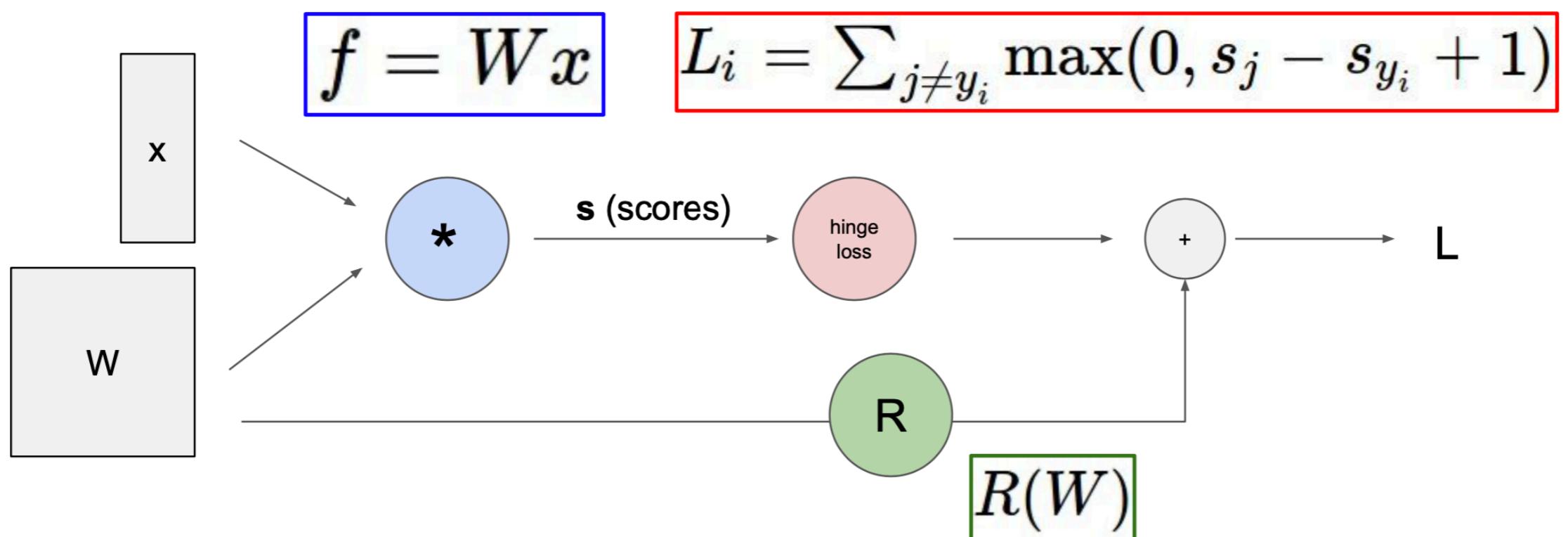
Back-propagation: apply chain rule to gradient of loss function to **adjust weights/biases**



Remedies against vanishing gradient:

- ReLU
- Re-normalisation

Computational graphs



Backpropagation: a simple example

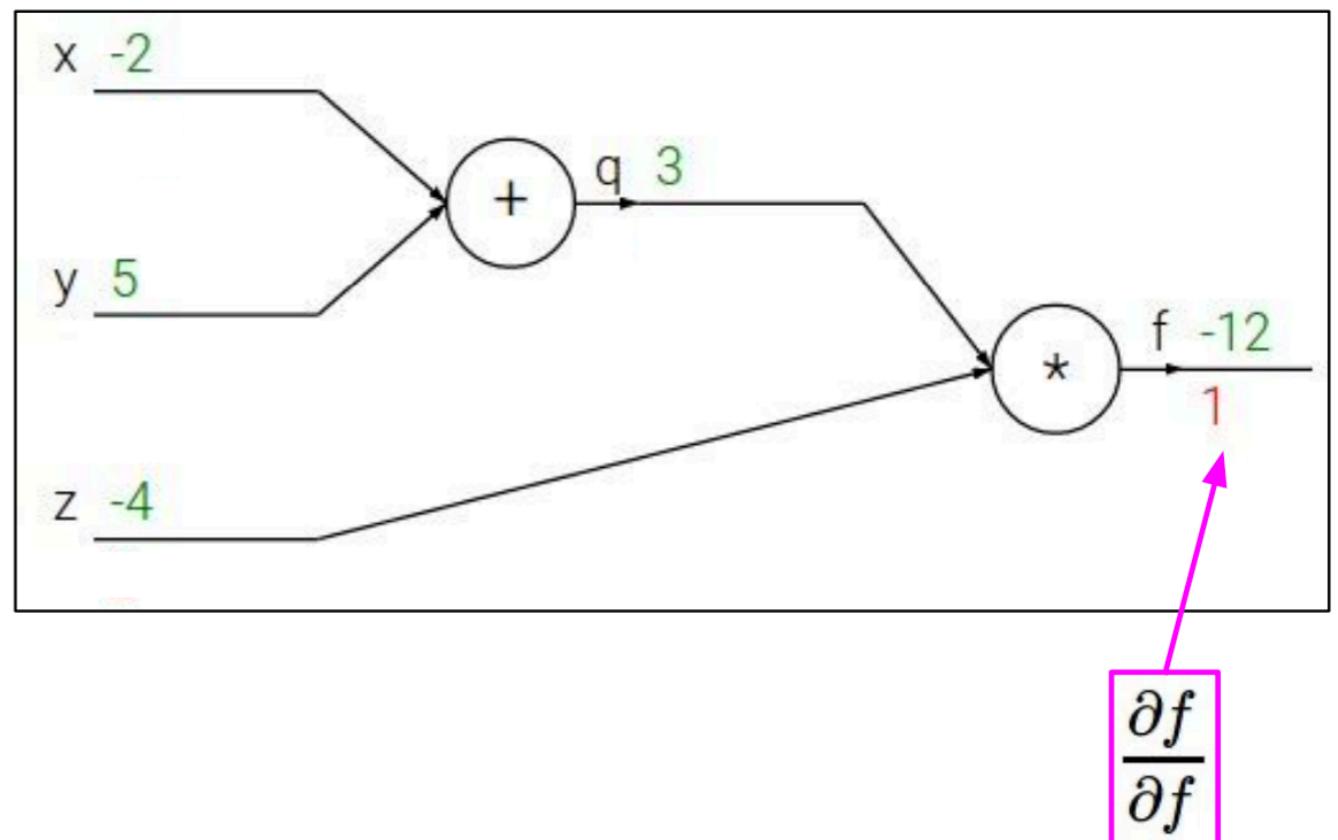
$$f(x, y, z) = (x + y)z$$

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

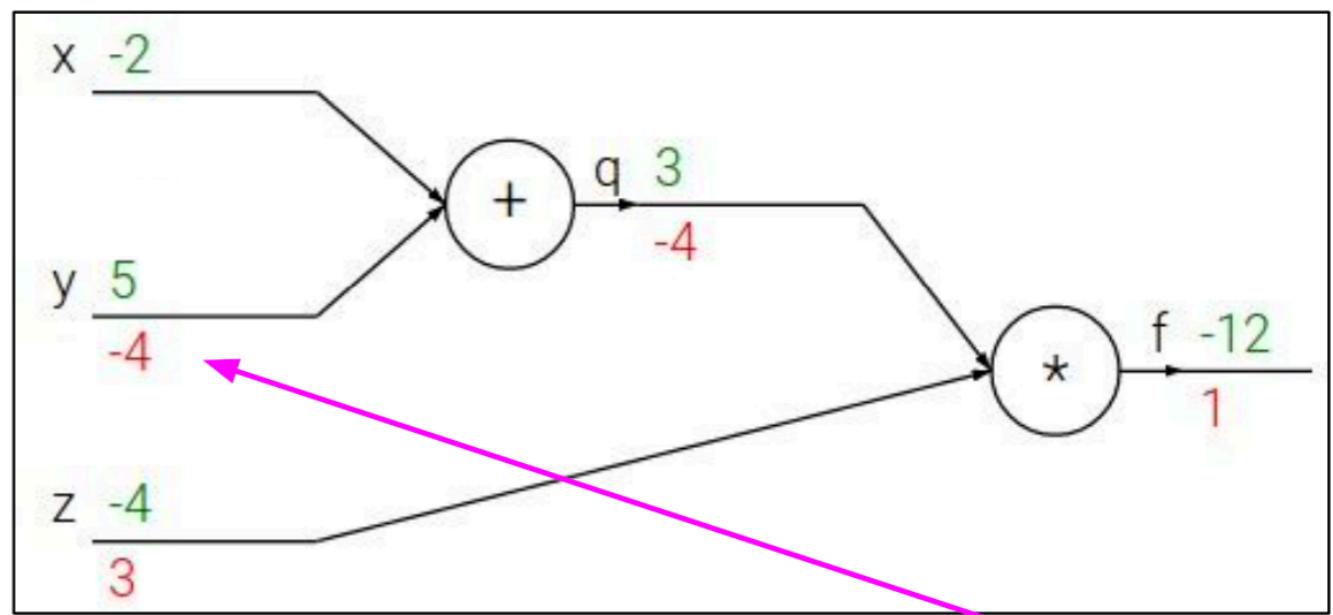
$$f(x, y, z) = (x + y)z$$

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

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

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

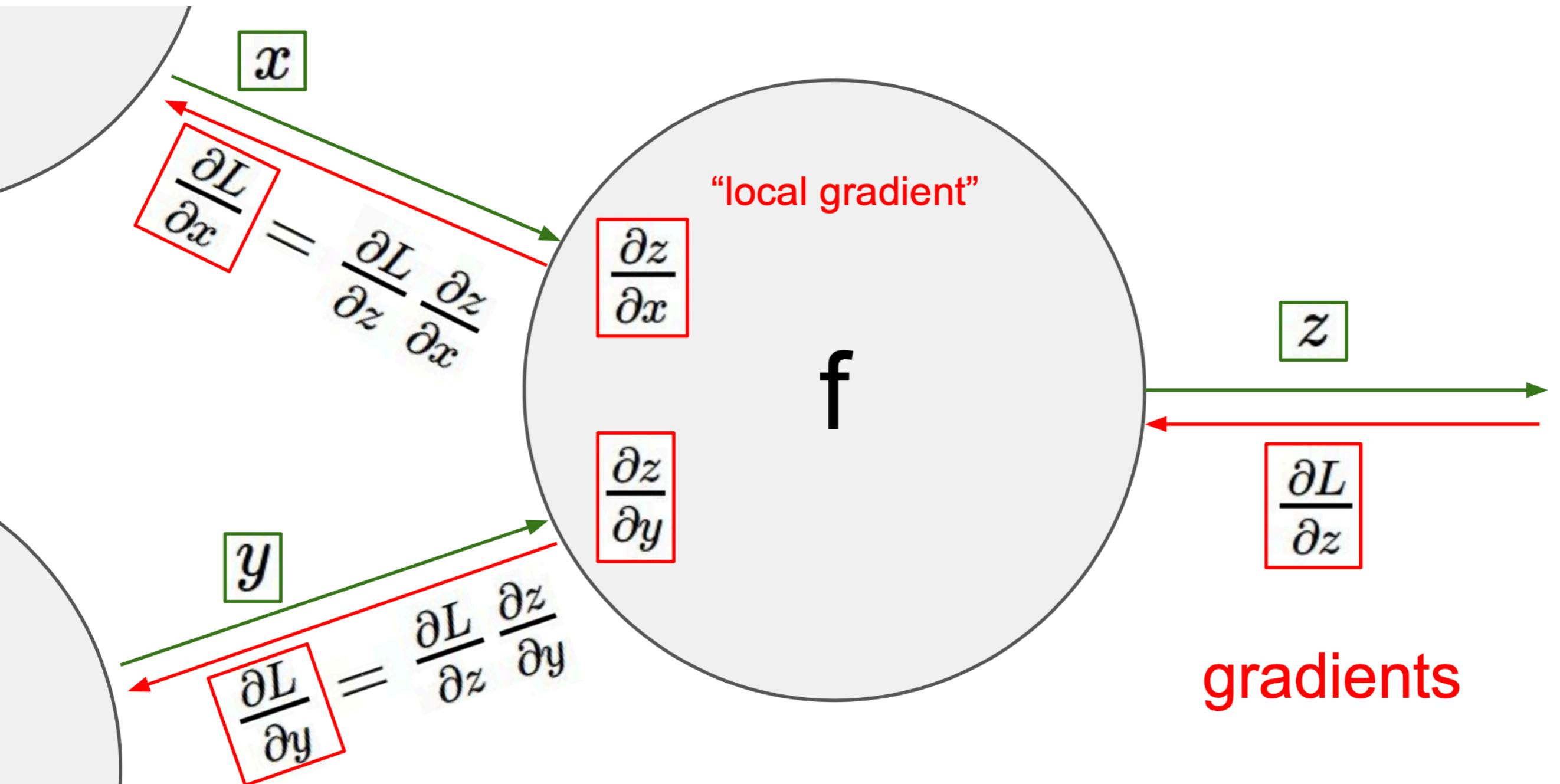
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

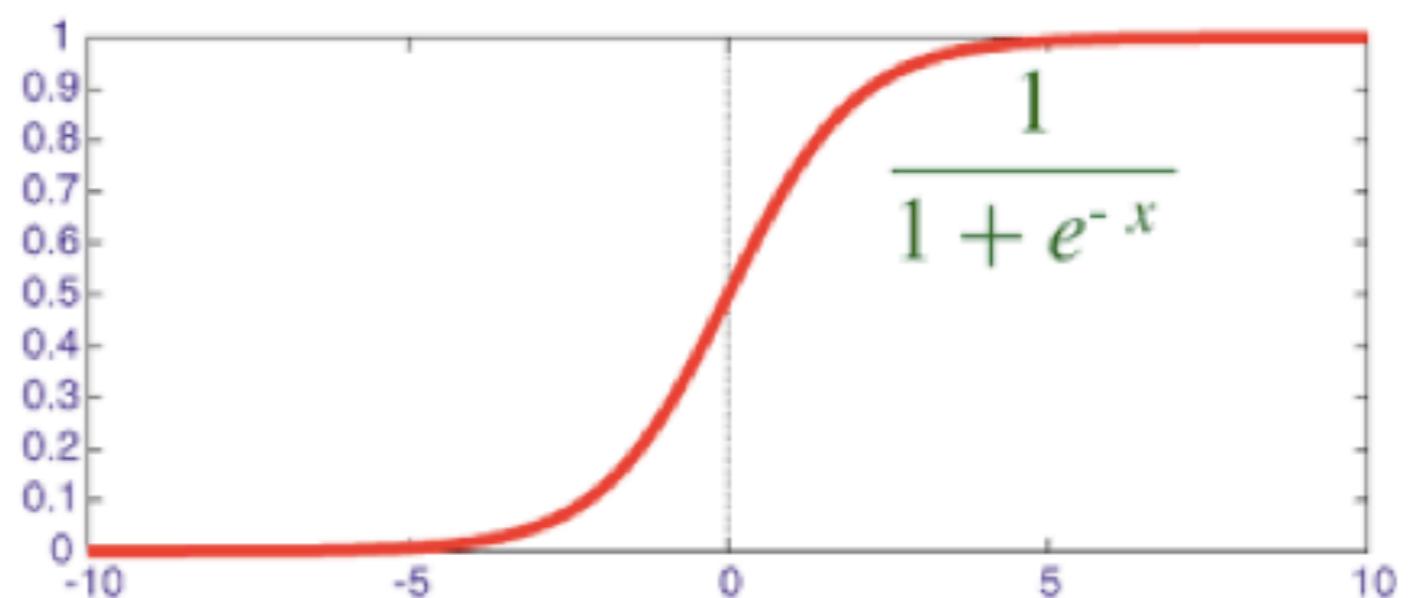
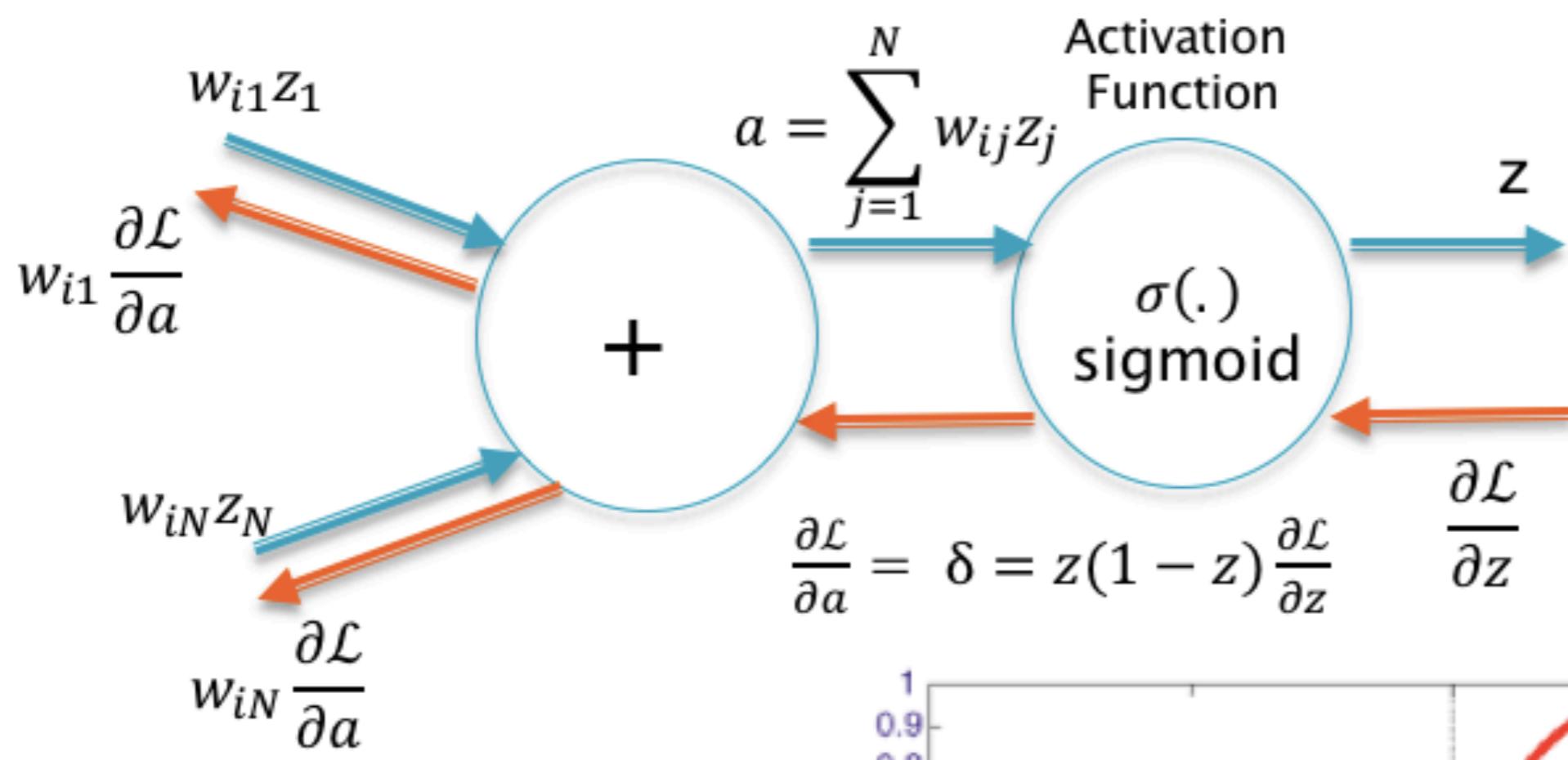


Chain rule:

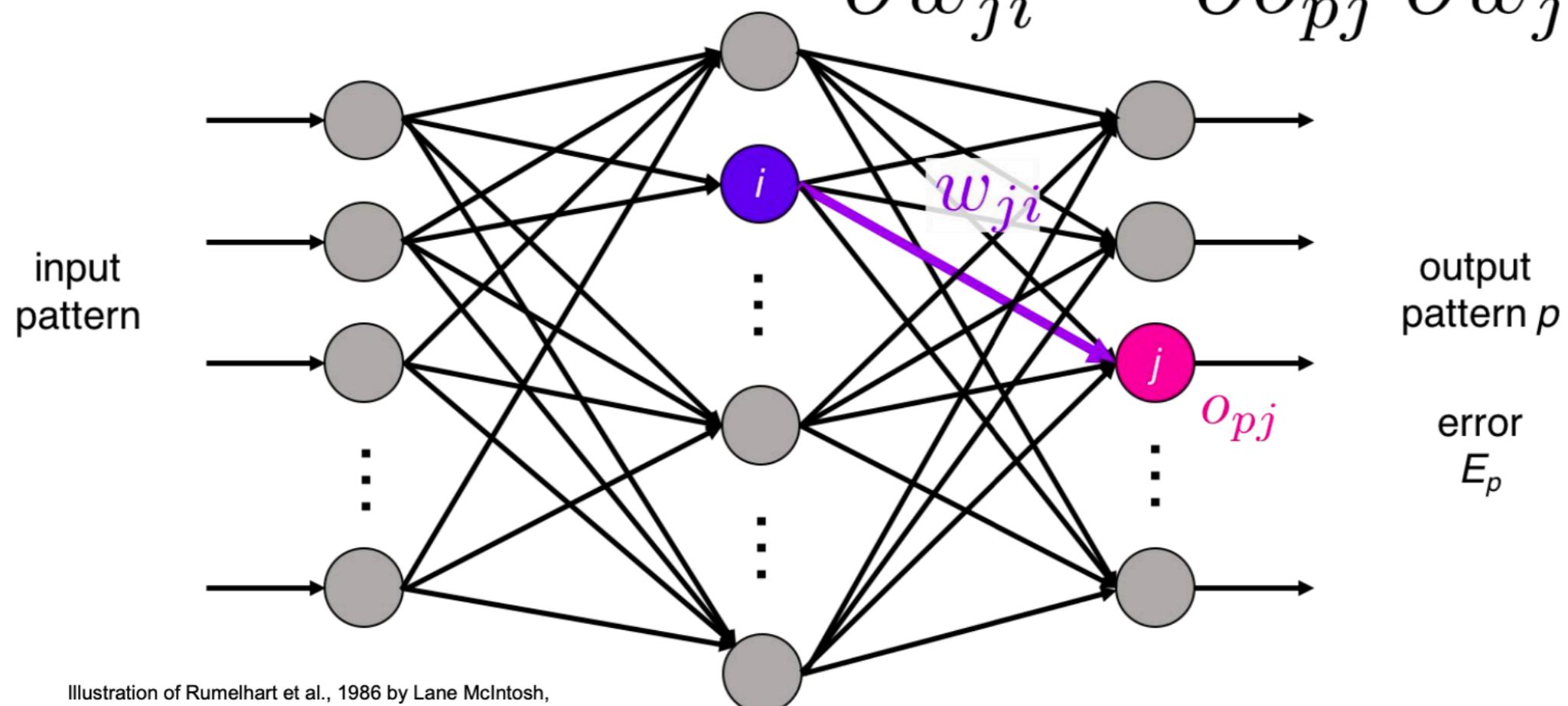
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

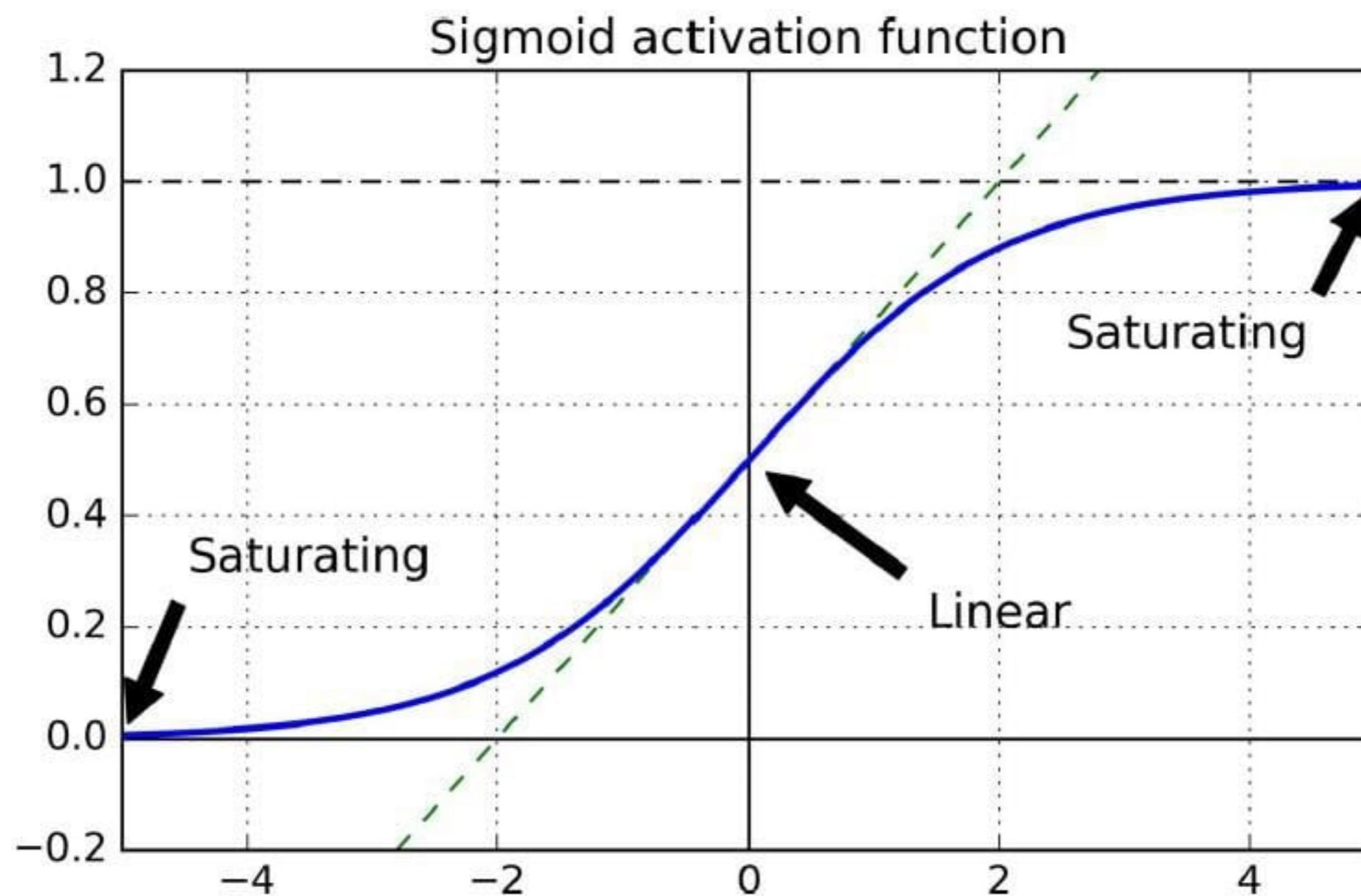
$$\frac{\partial f}{\partial y}$$





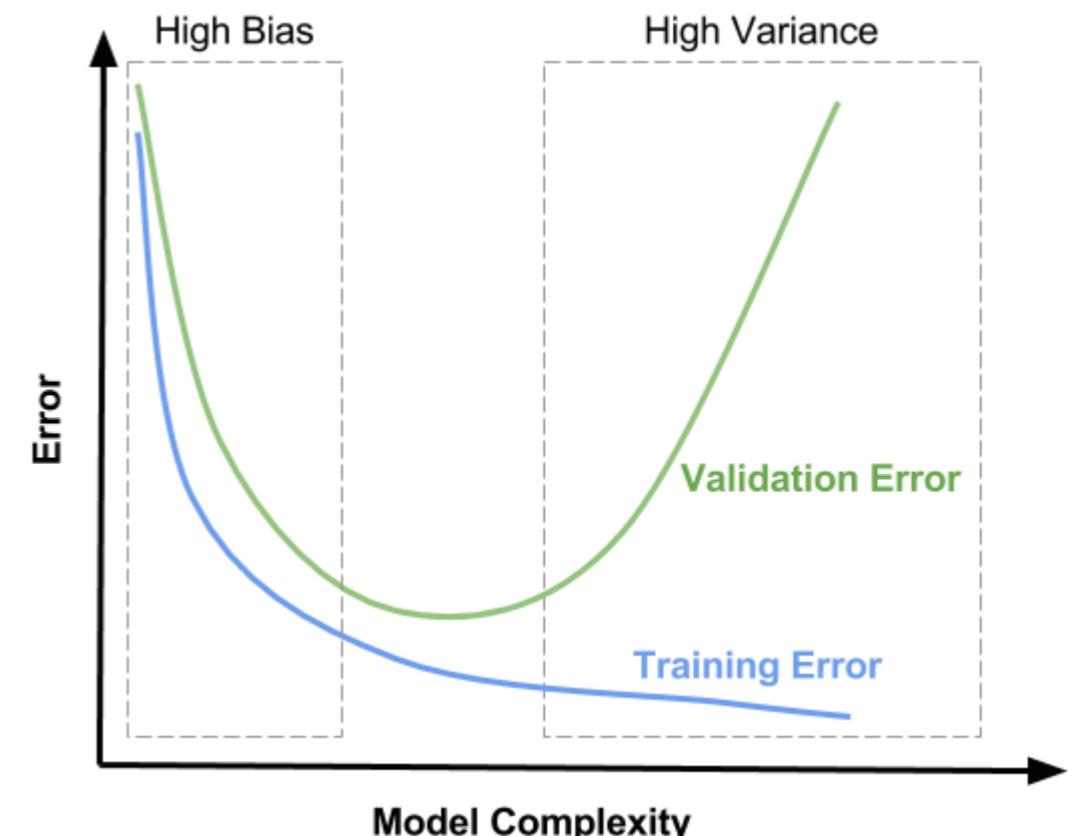
$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}$$





Fighting over-fitting

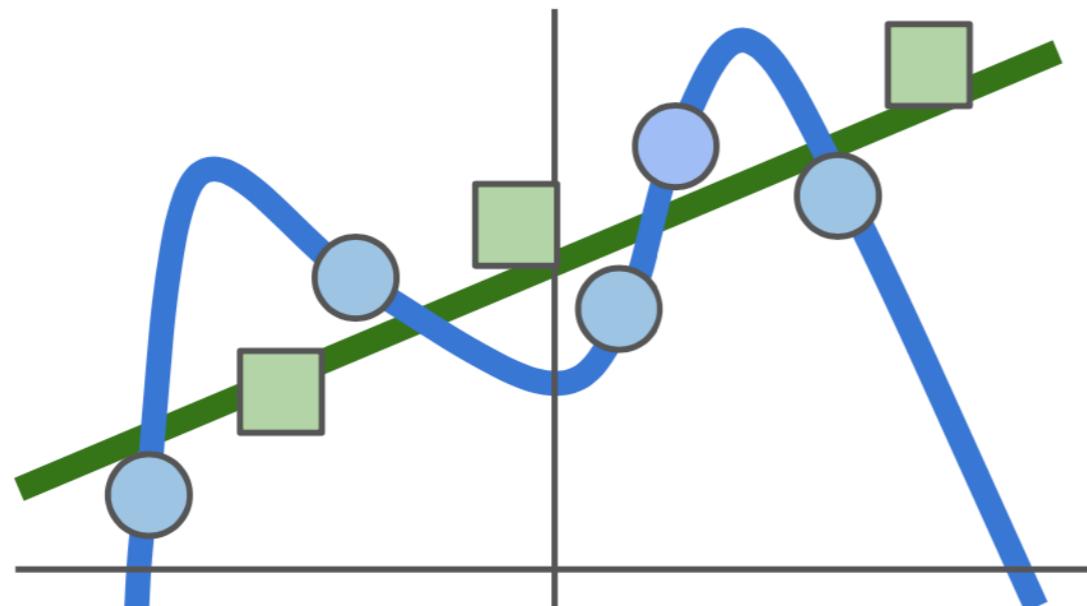
- Regularisation
- Dropout
- Early stopping
- Simplify the architecture
- More data (even if you have to make it up: translate, rotate, flip, crop, lighten/darken, add noise)
of weights ~ VC dimension ~ # degrees of freedom



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

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

λ = 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

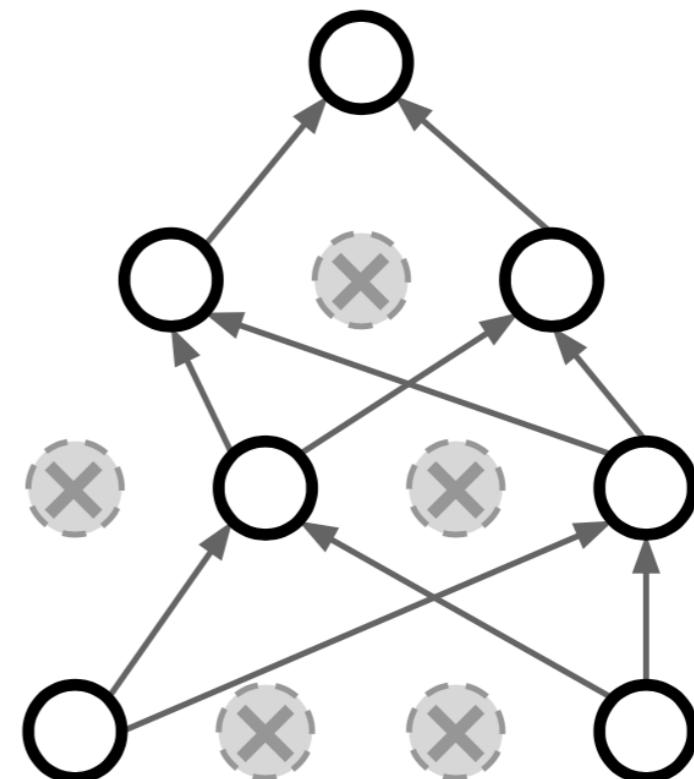
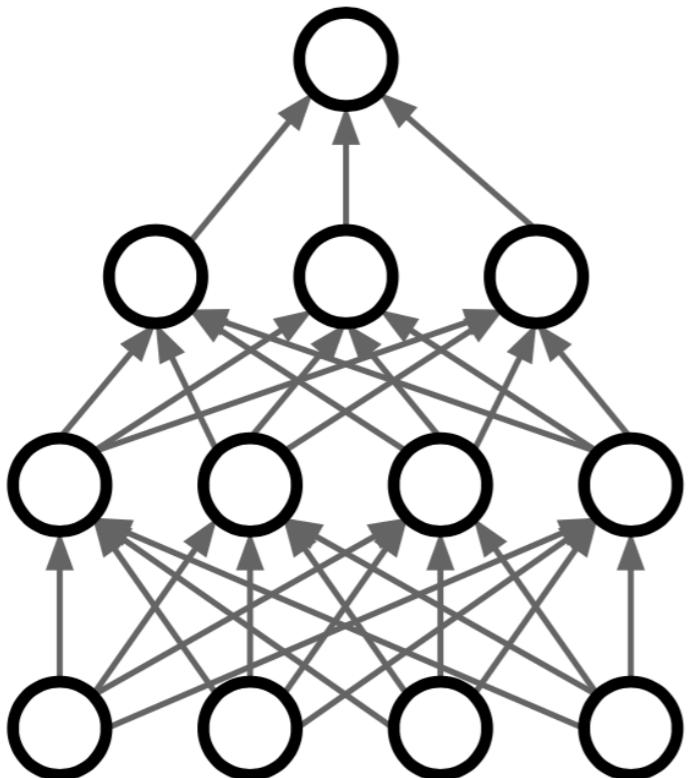
$$R(W) = \sum_k \sum_l |W_{k,l}|$$

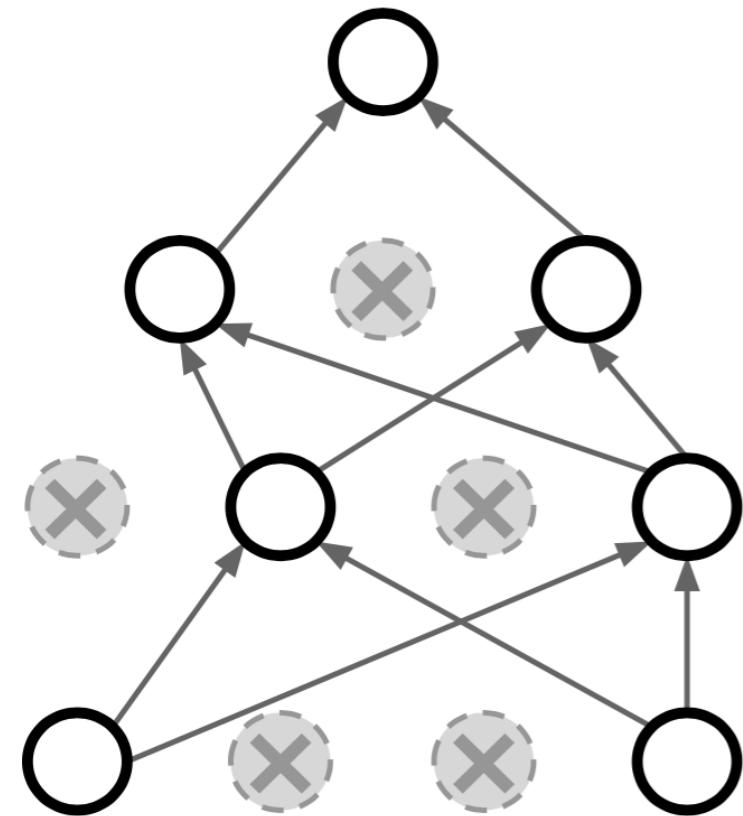
Elastic net (L1 + L2)

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

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

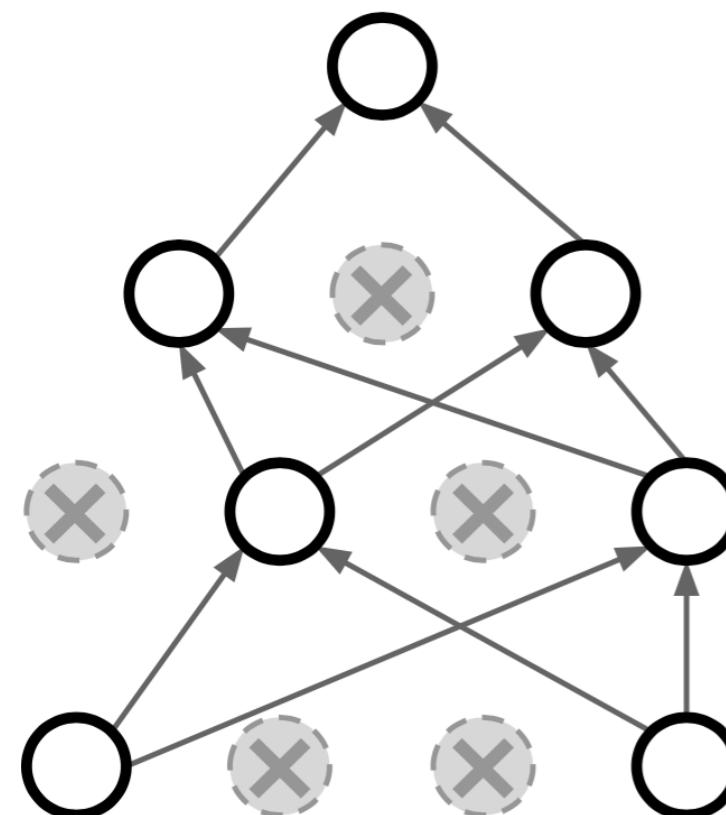


Another interpretation:

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

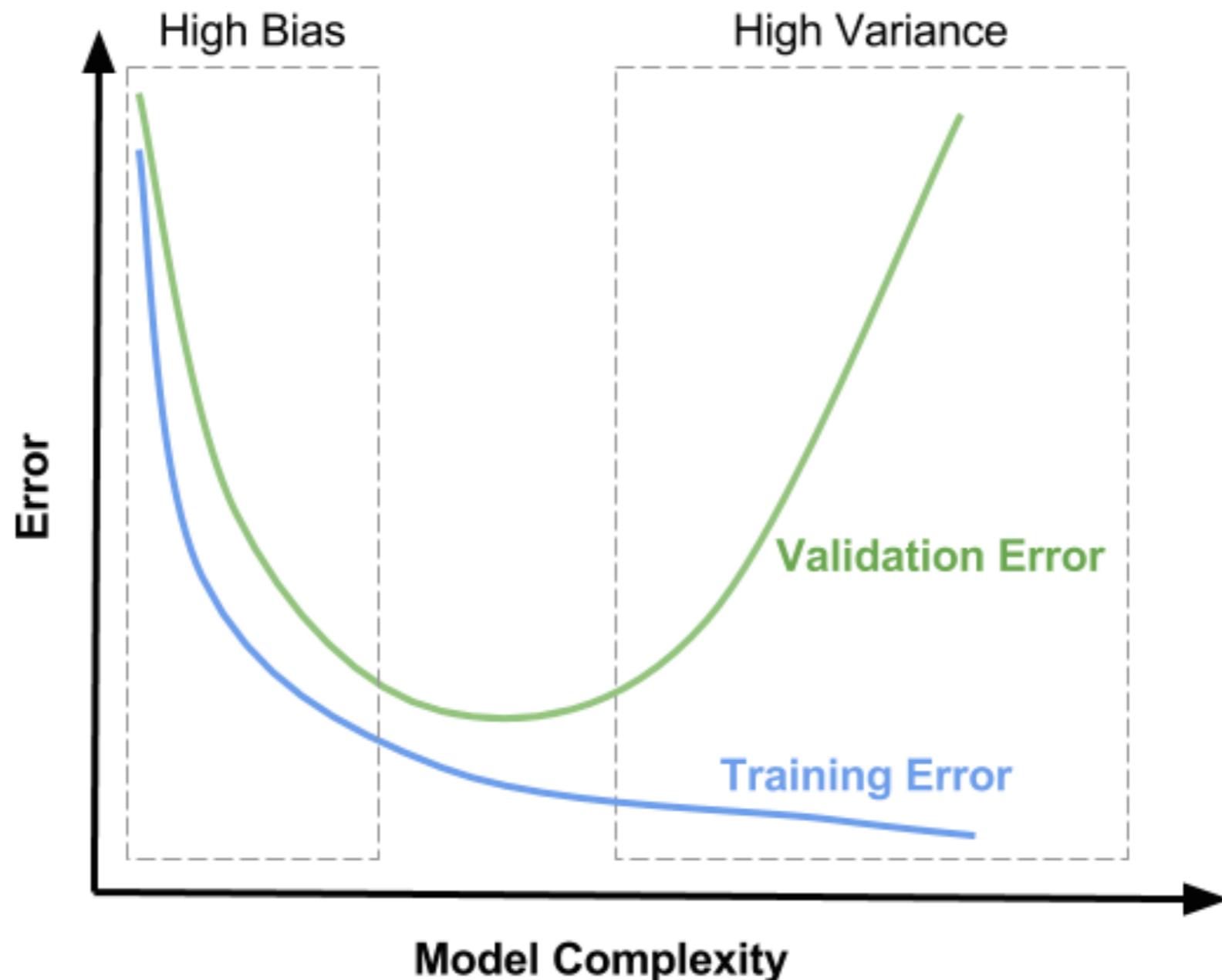
Each binary mask is one model

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

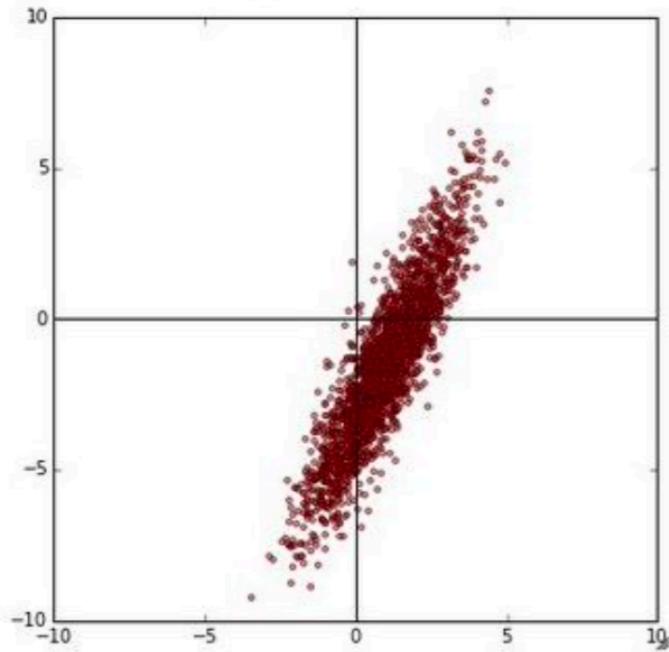


Early Stopping

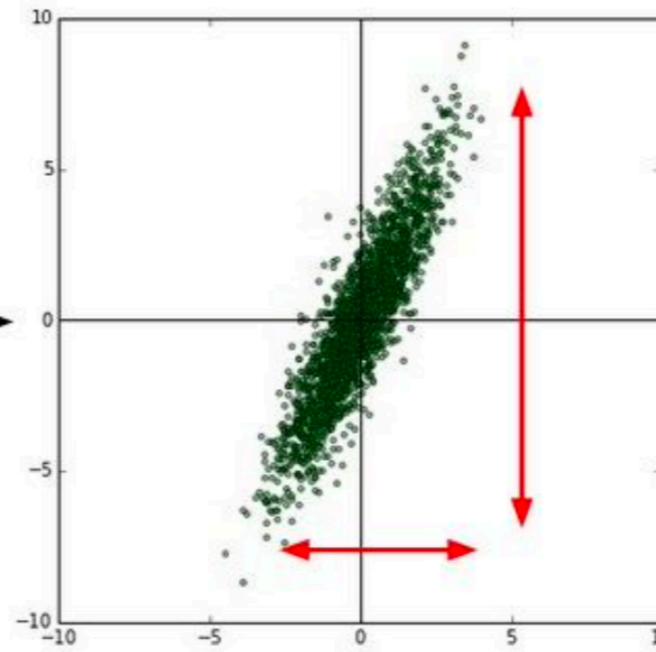
@best or @plateau



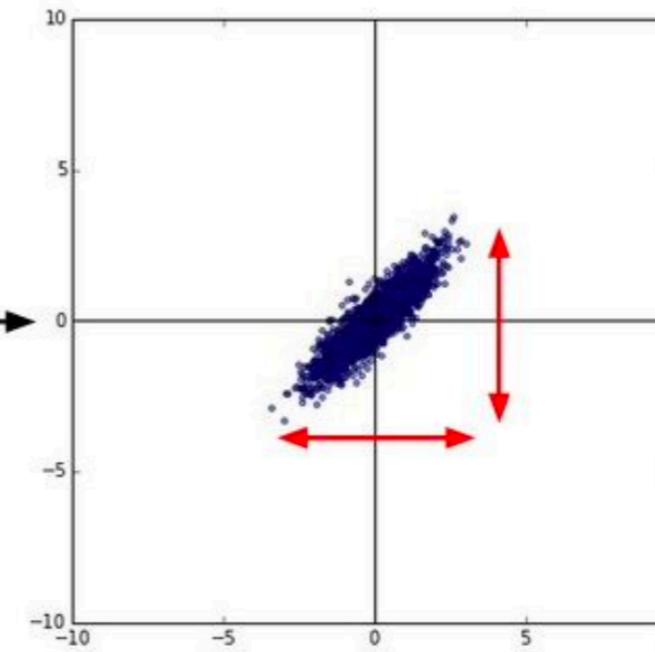
original data



zero-centered data



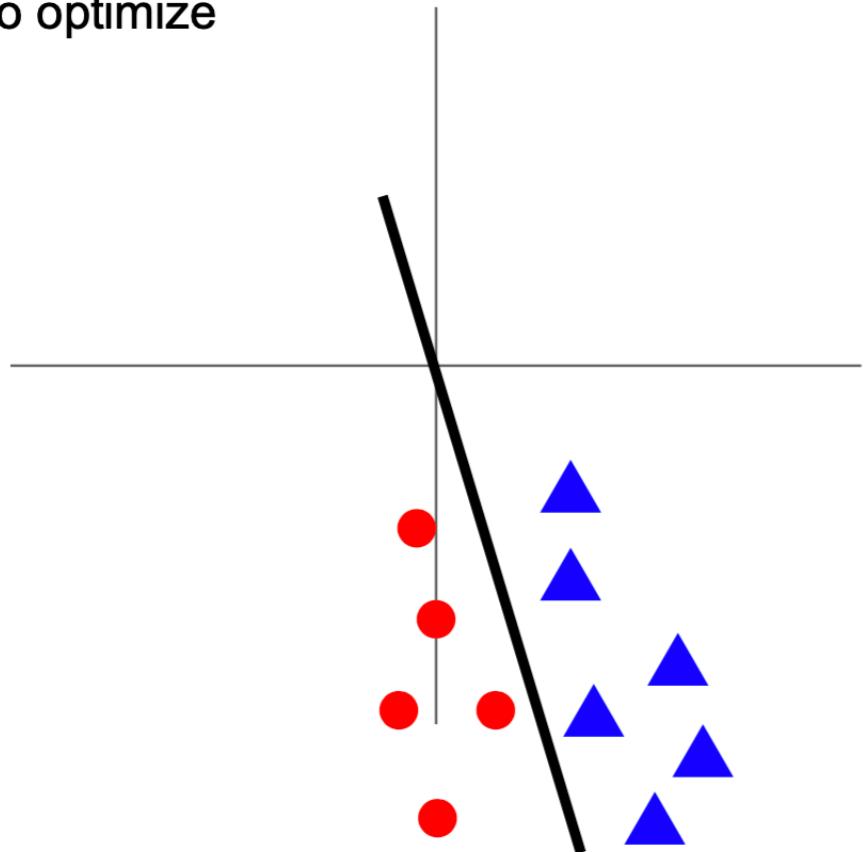
normalized data



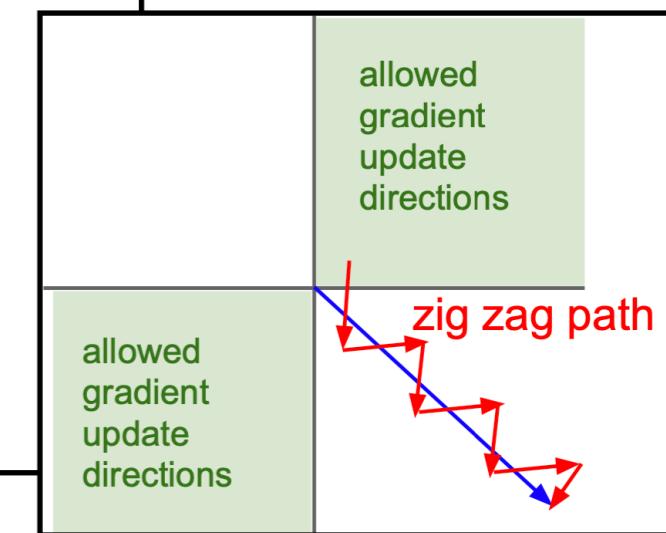
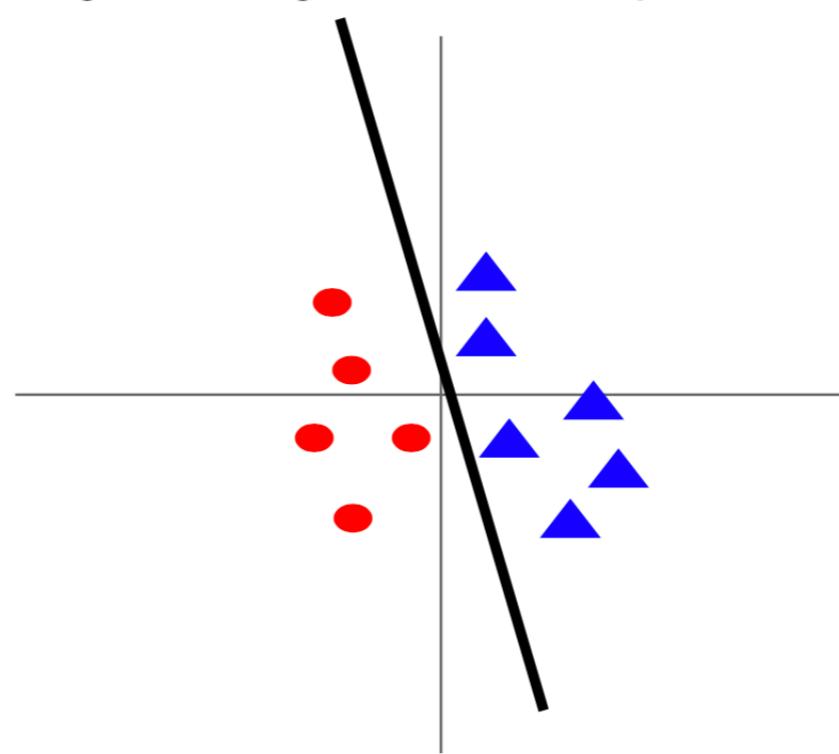
```
X -= np.mean(X, axis = 0)
```

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

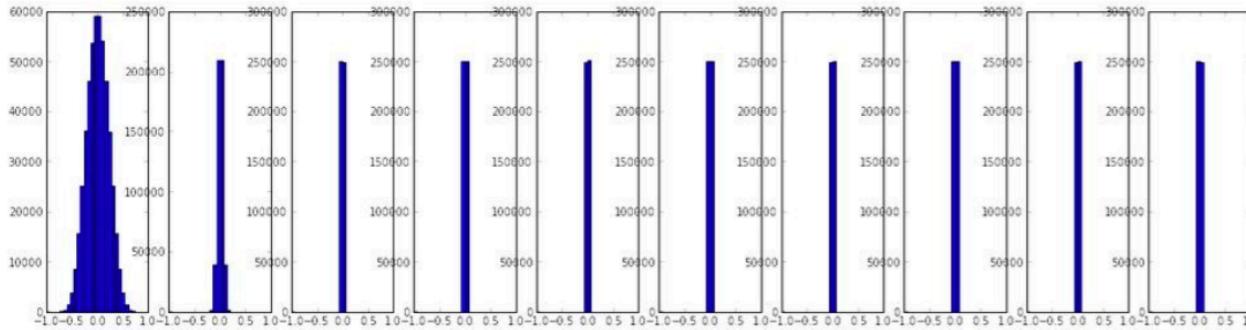
Before normalization: classification loss very sensitive to changes in weight matrix; hard to optimize



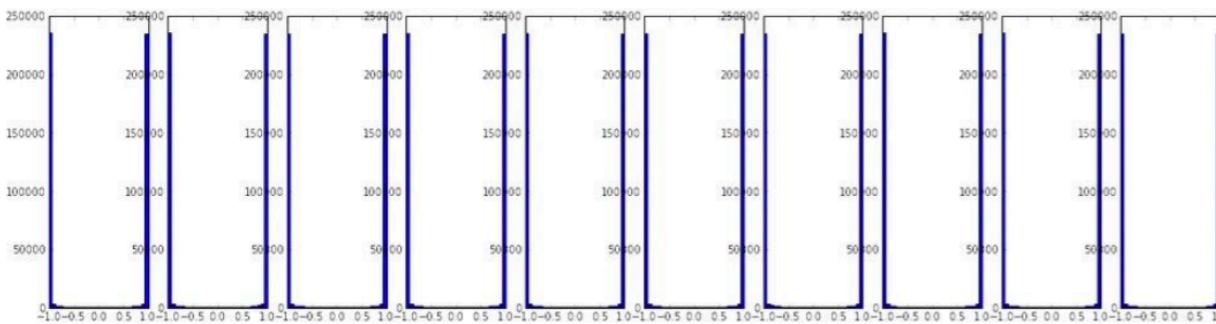
After normalization: less sensitive to small changes in weights; easier to optimize



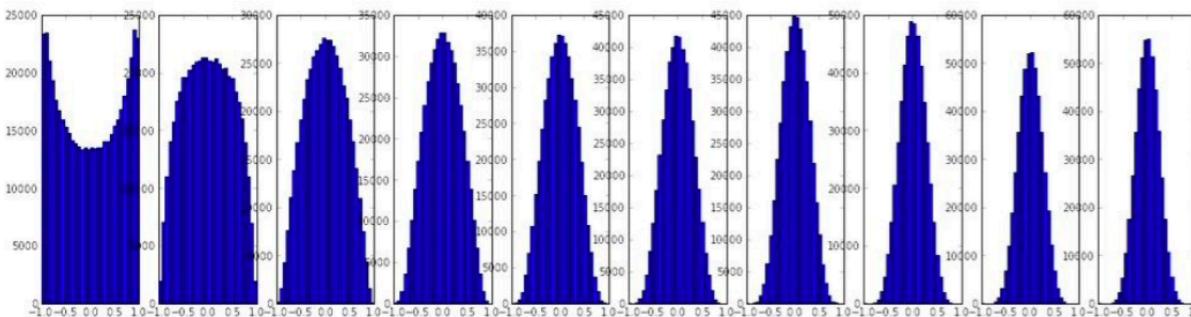
Weight initialisation



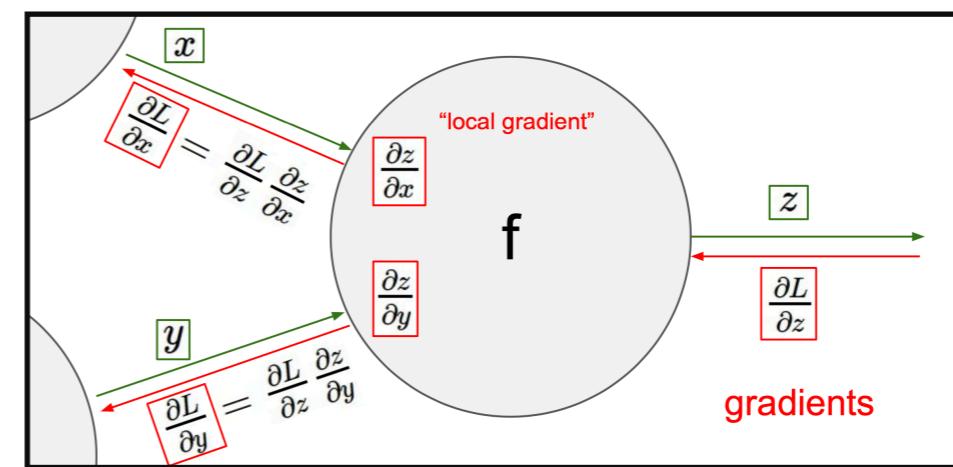
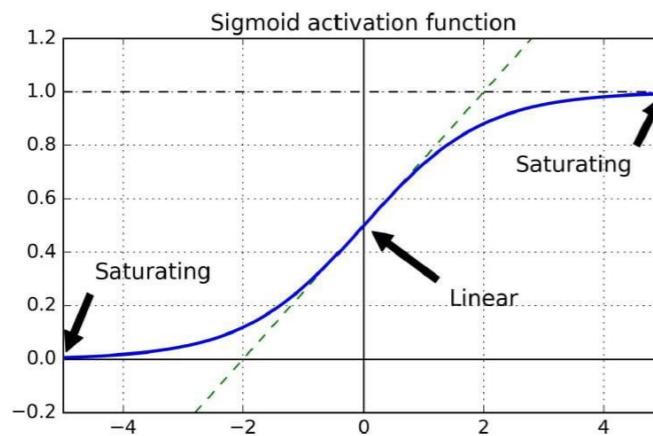
Initialization too small:
Activations go to zero, gradients also zero,
No learning



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:

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

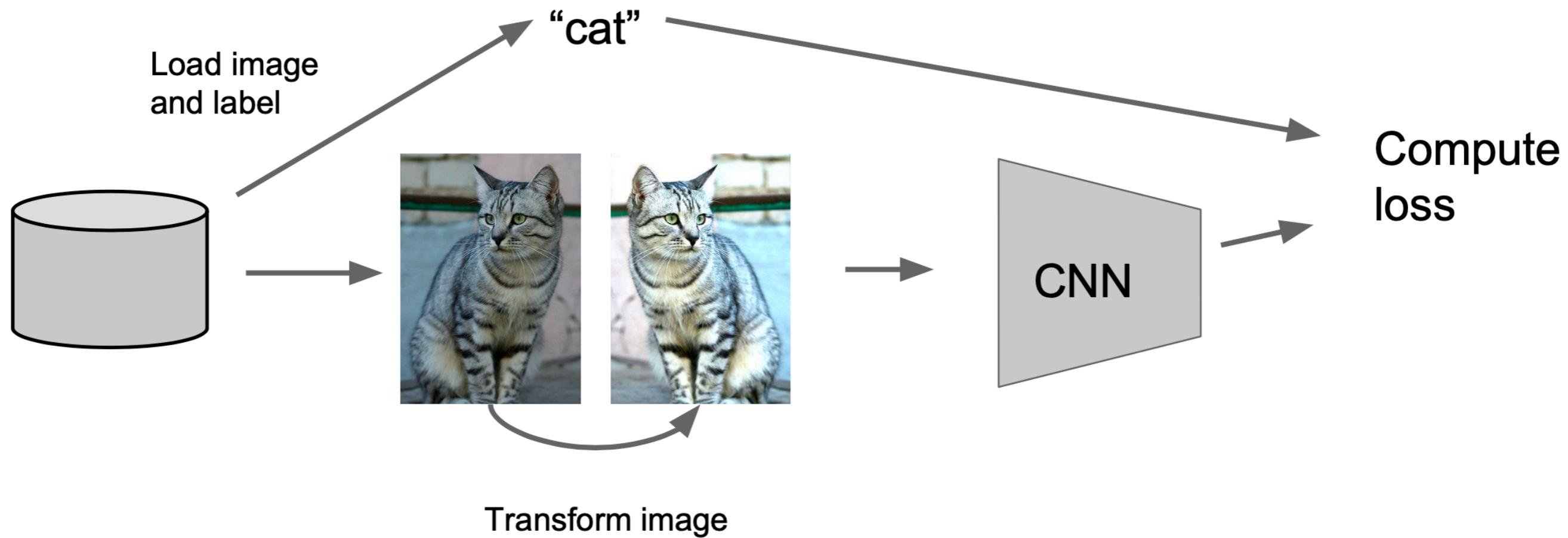
Note, the network can learn:

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

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

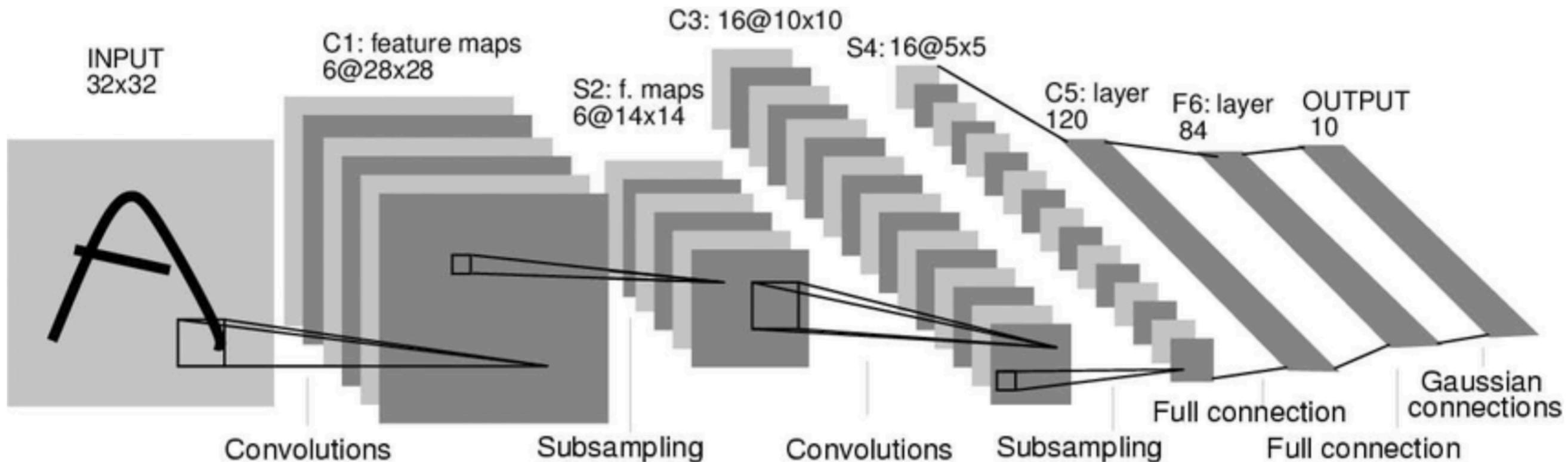
to recover the identity mapping.

Regularization: Data Augmentation

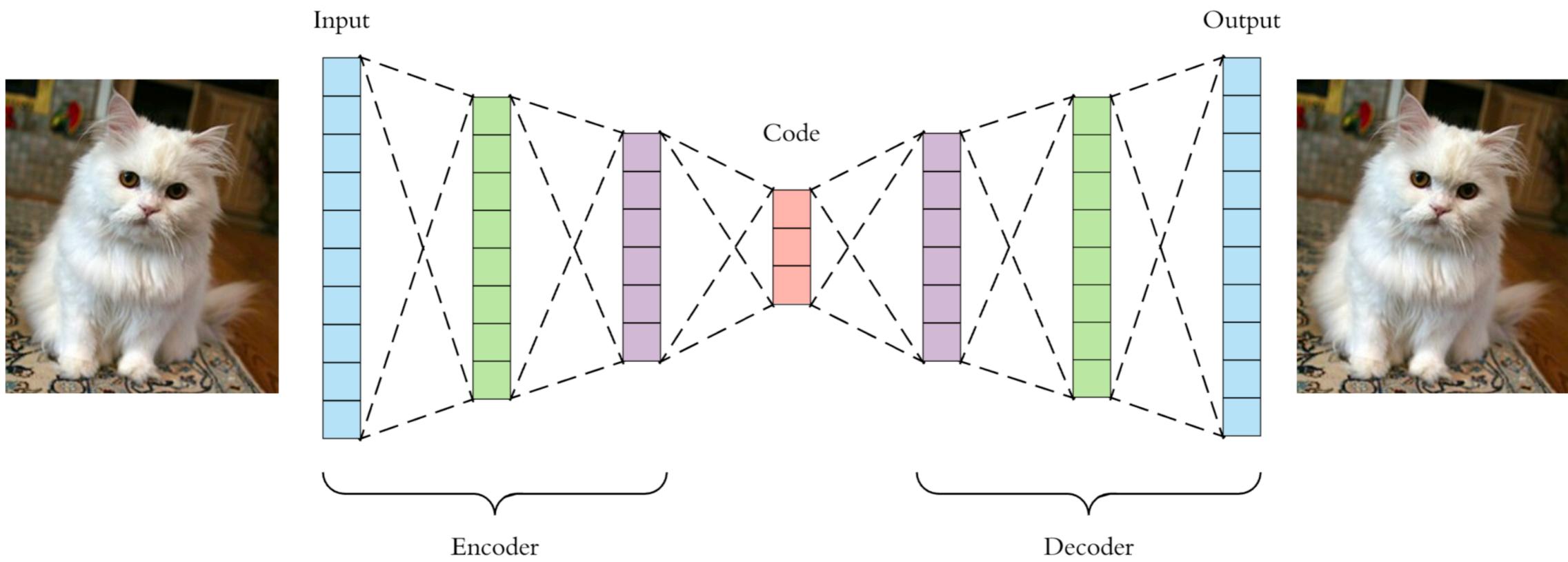
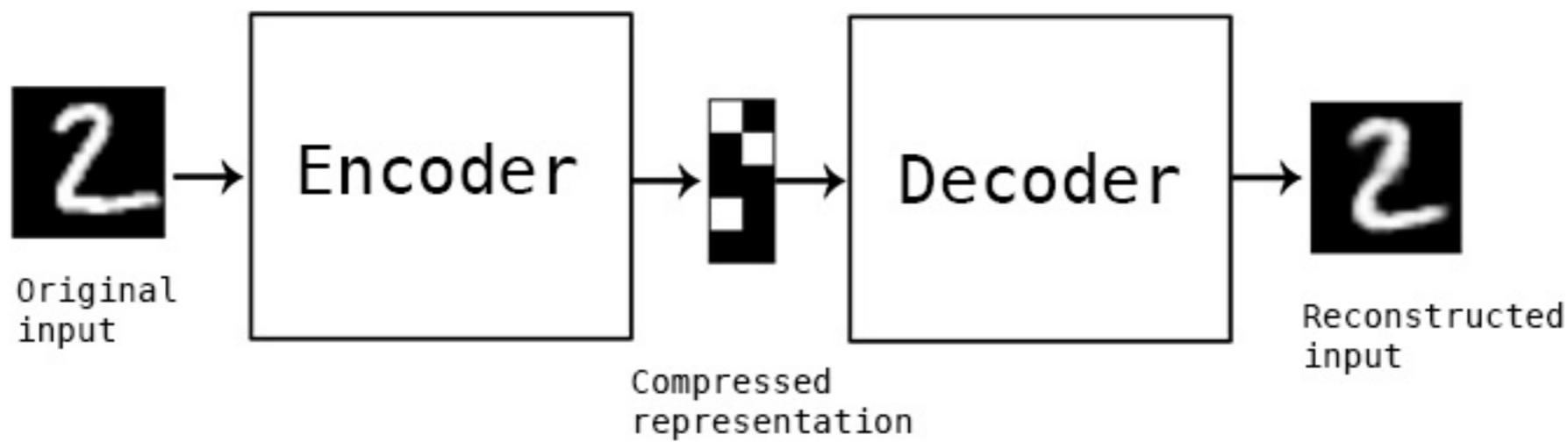


translate, rotate, flip, crop, lighten/darken, mask, add noise, ... use your fantasy...

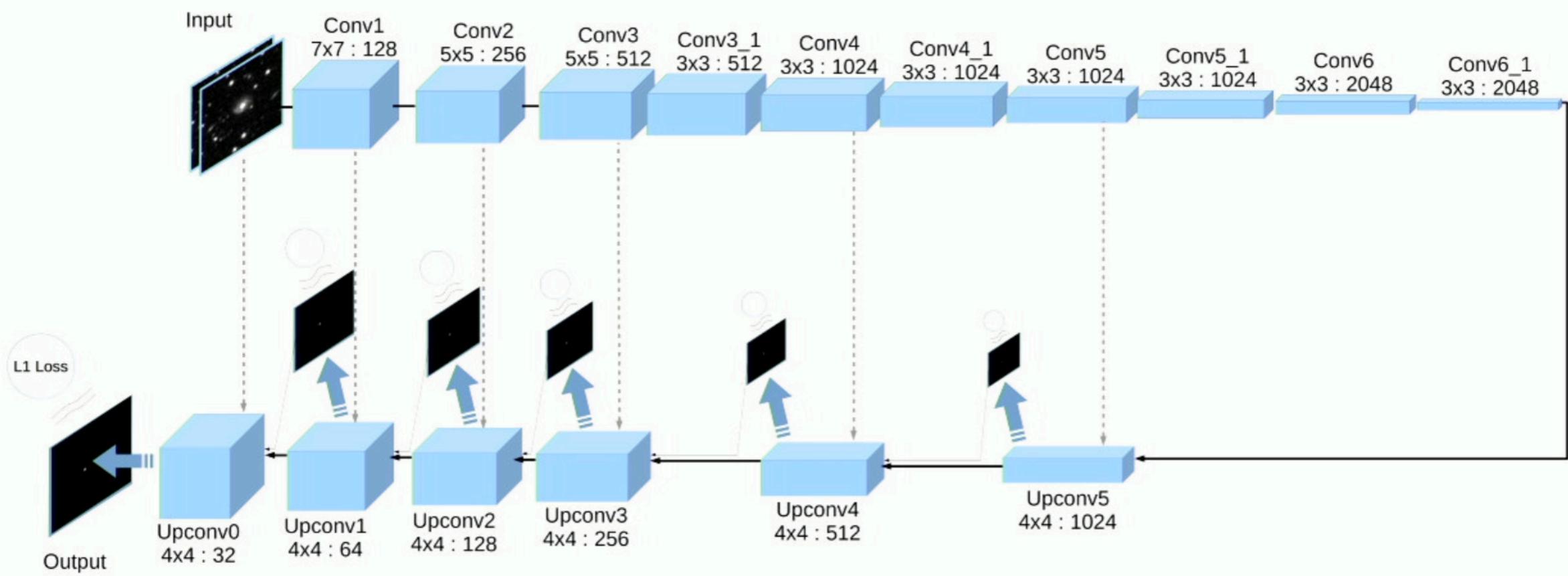
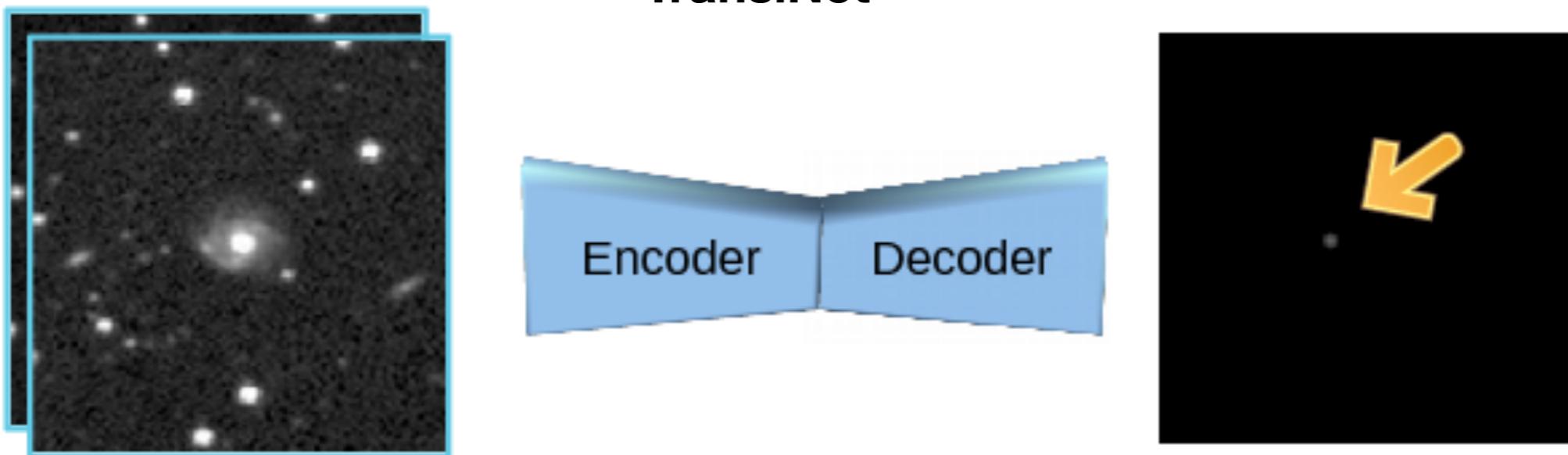
Convolutional Neural Networks



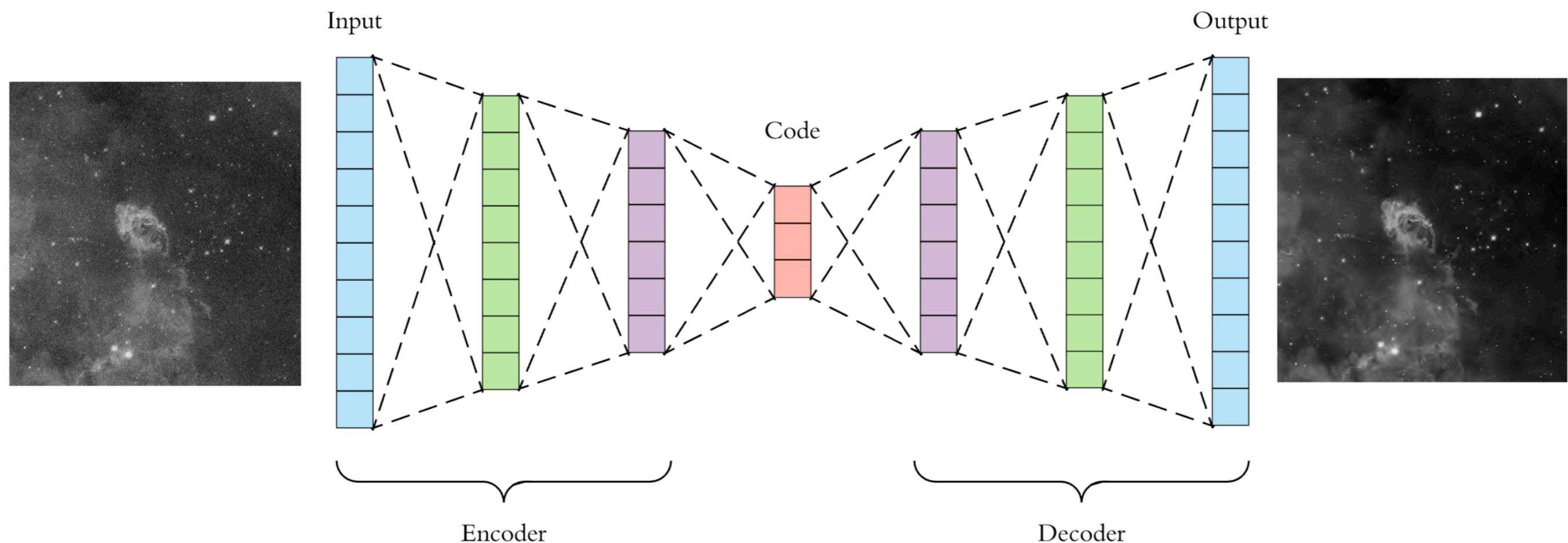
Encoder-decoders (Autoencoders)



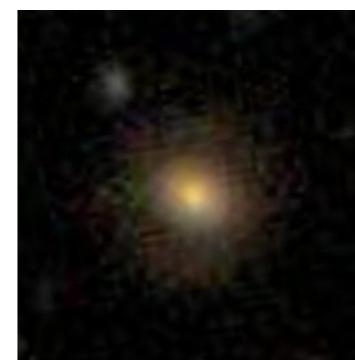
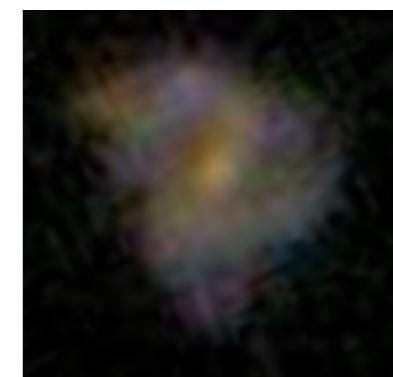
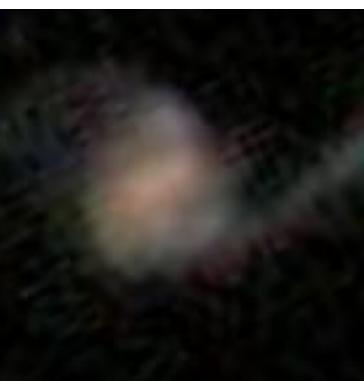
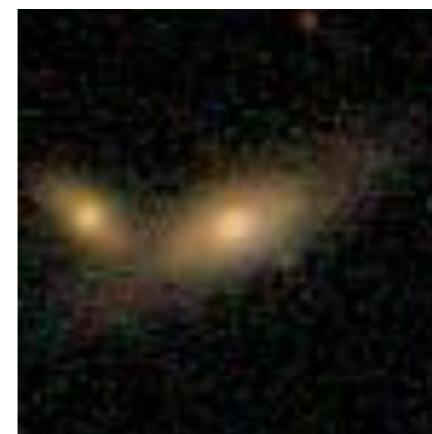
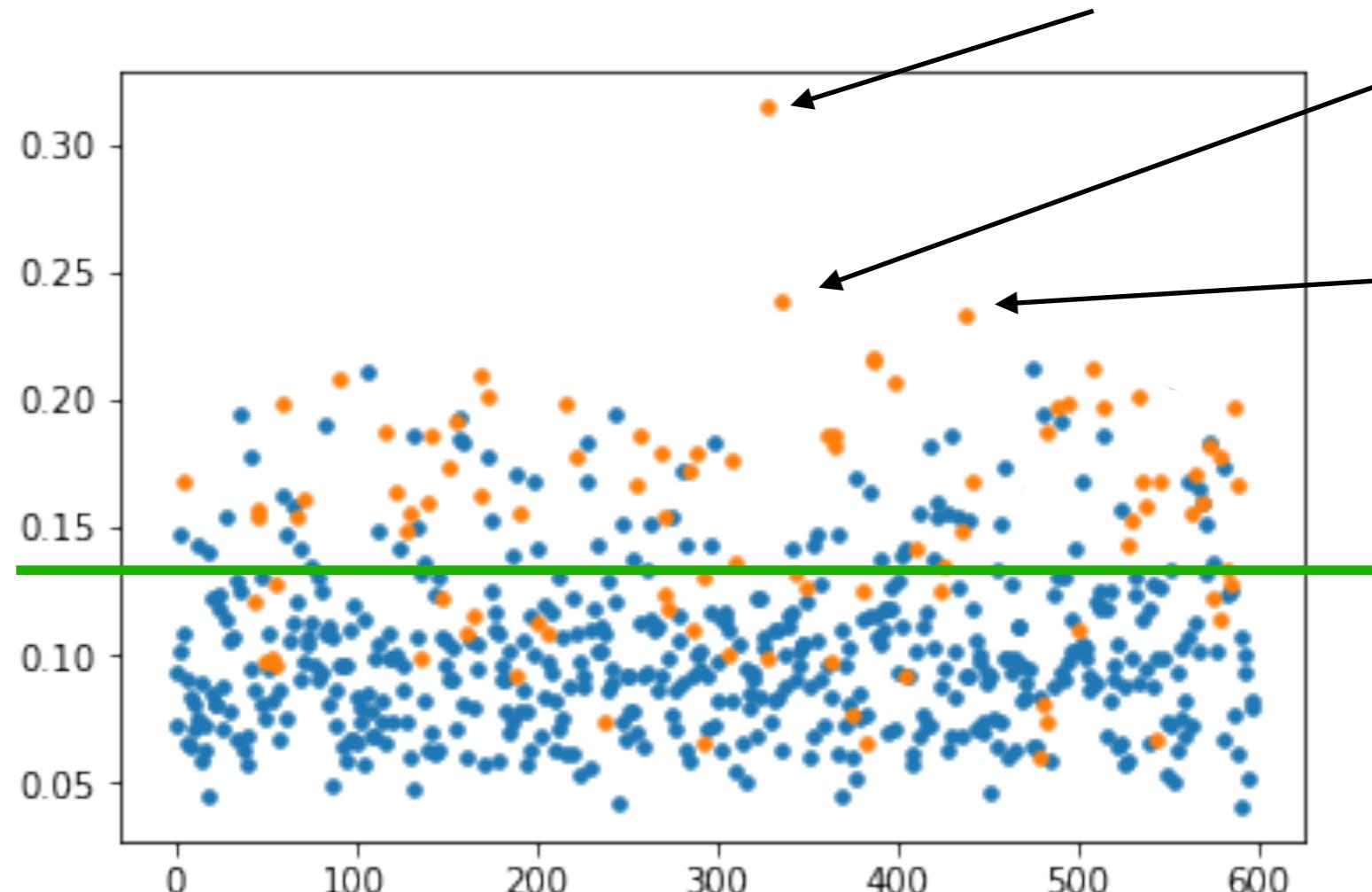
TransiNet



E.g. De-noising



Anomaly detection



Encoder/decoder usage

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

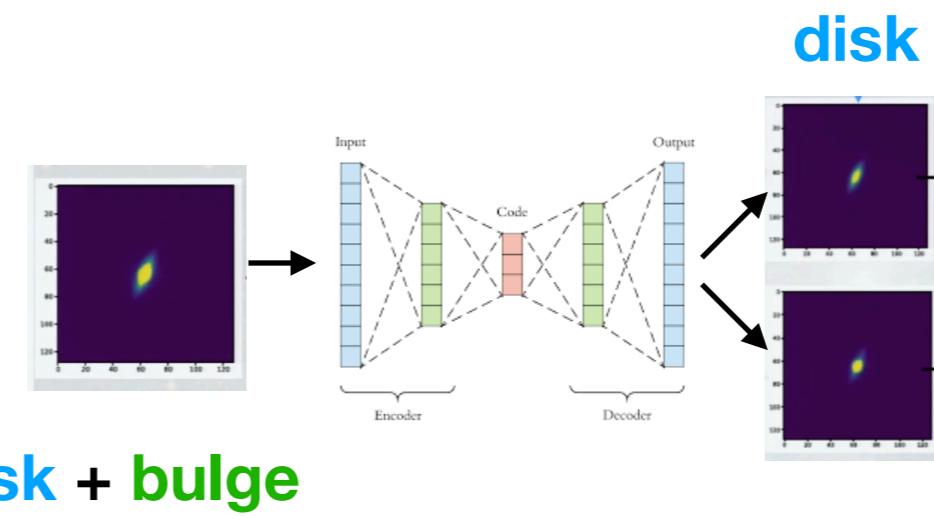
- Image segmentation

- Deblending

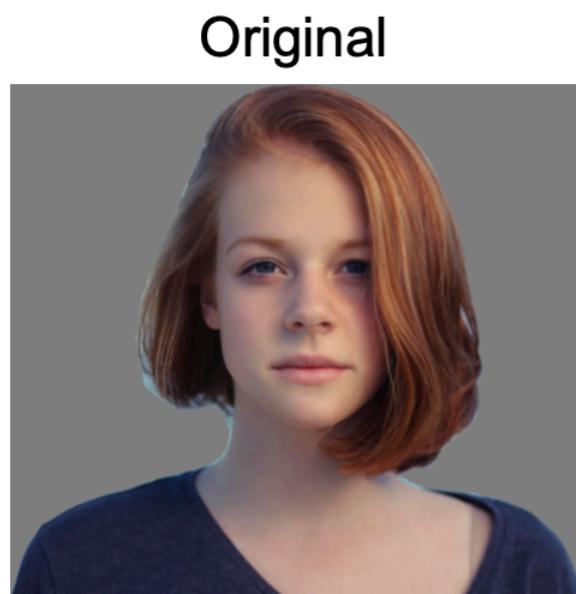
- Predicting next move

- Many others...

- (Can be considered as an unsupervised learning)



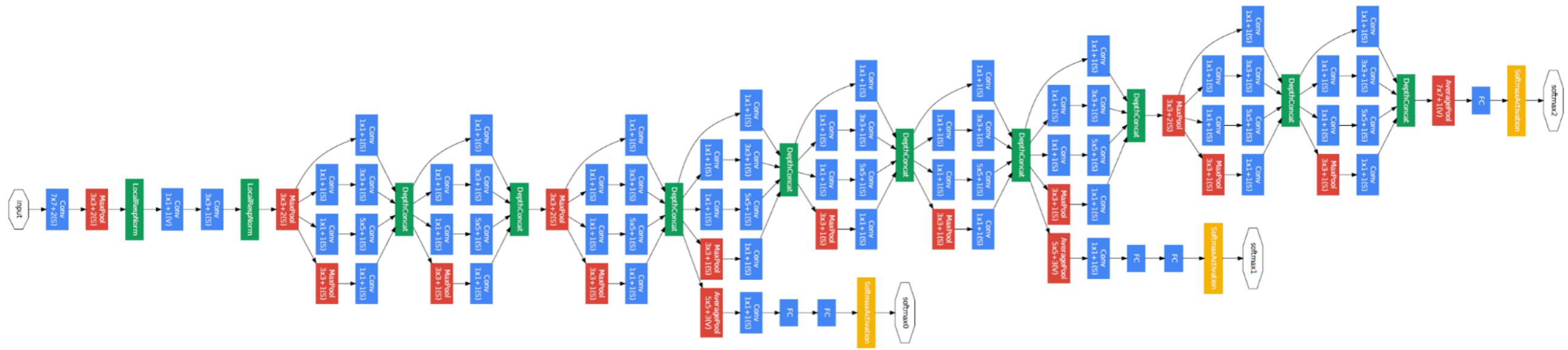
- Distance metrics on pixels are not informative



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

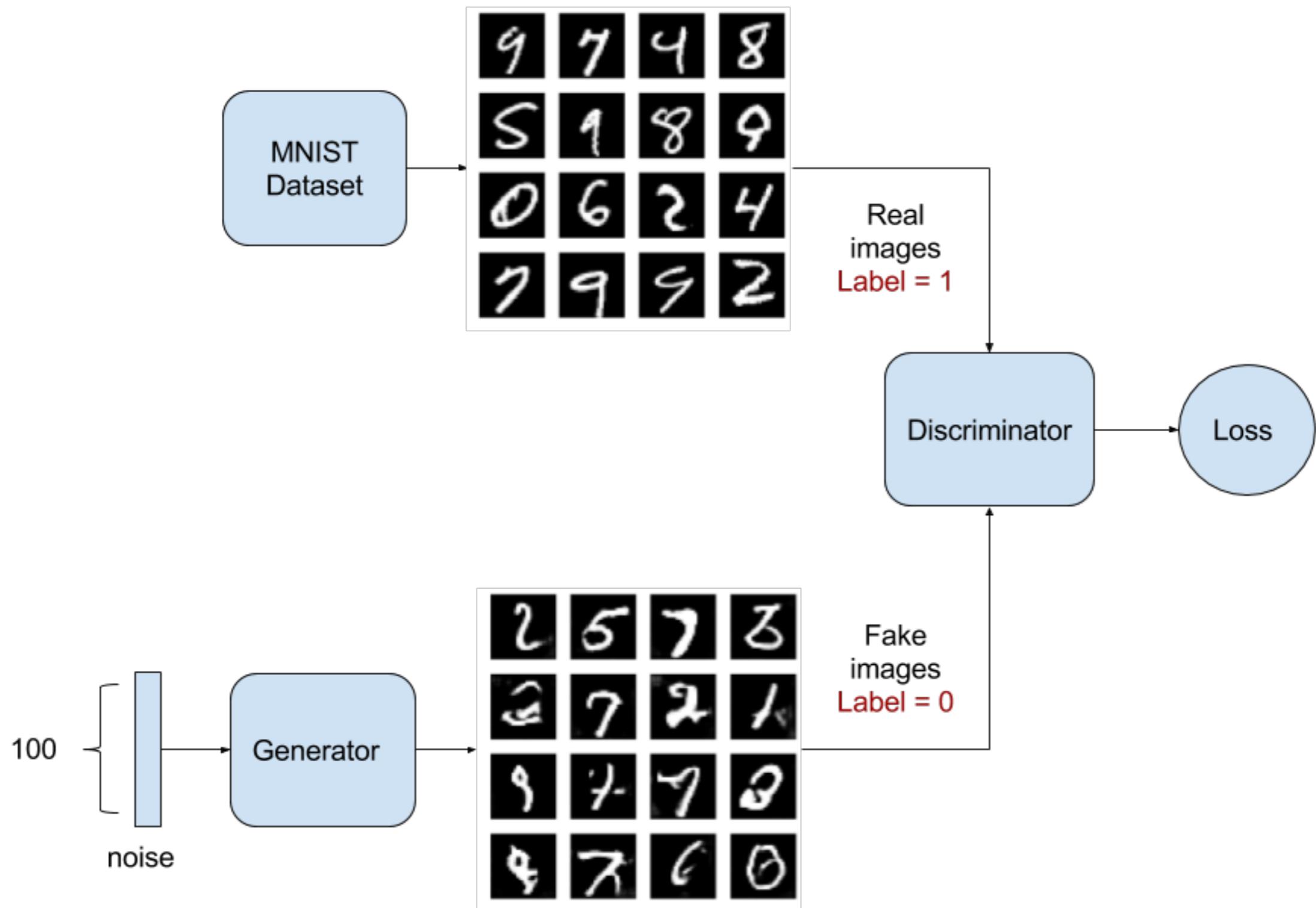
Original image is
CC0 public domain

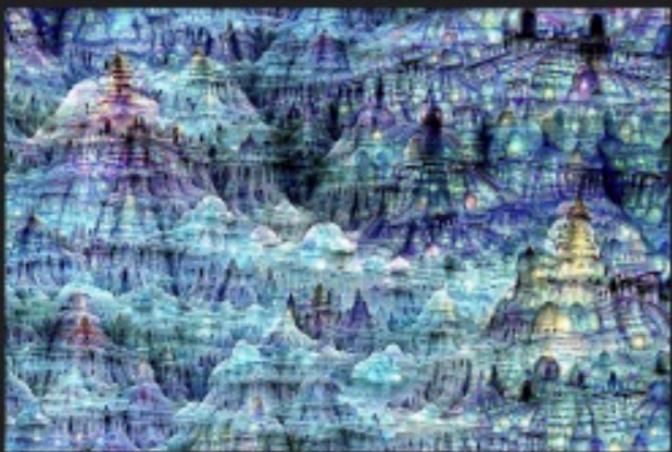
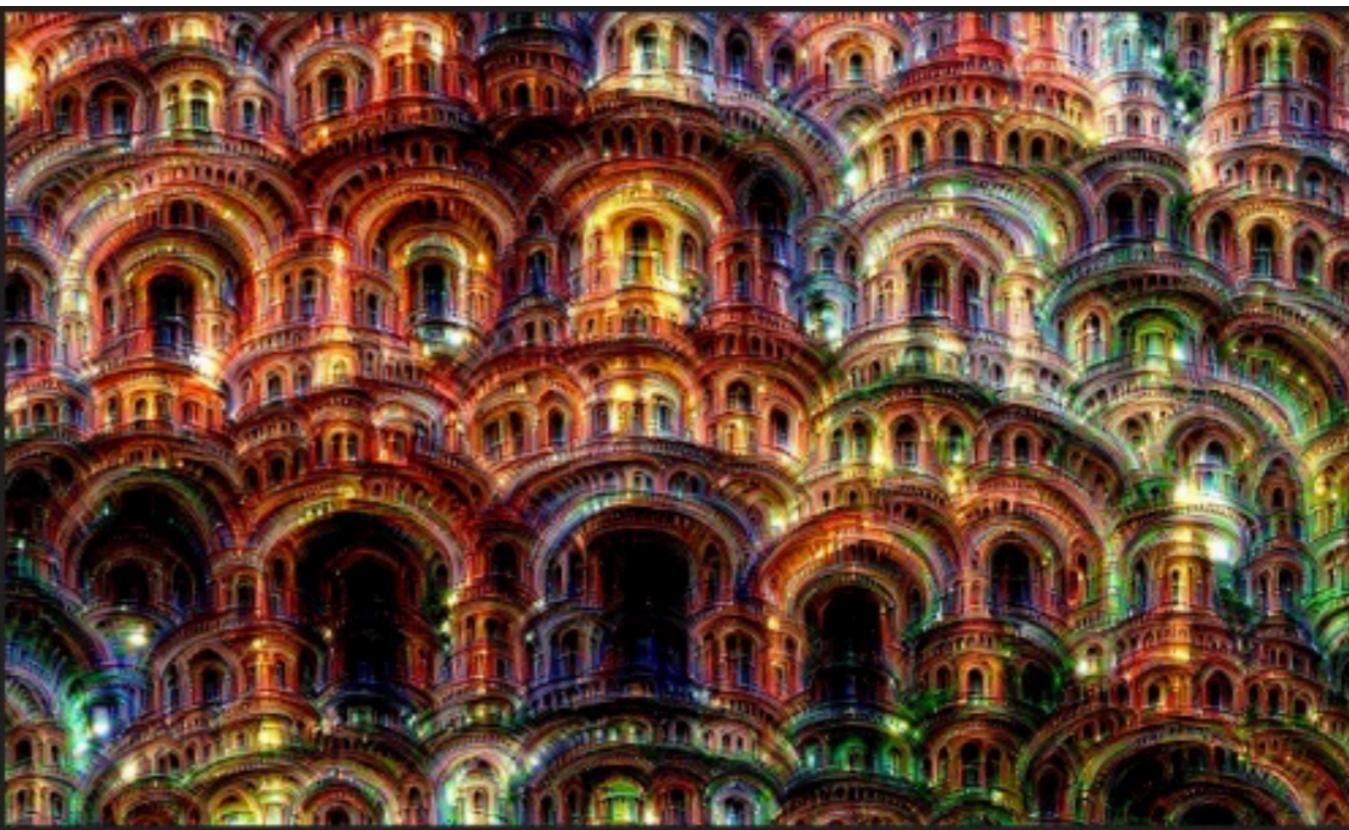
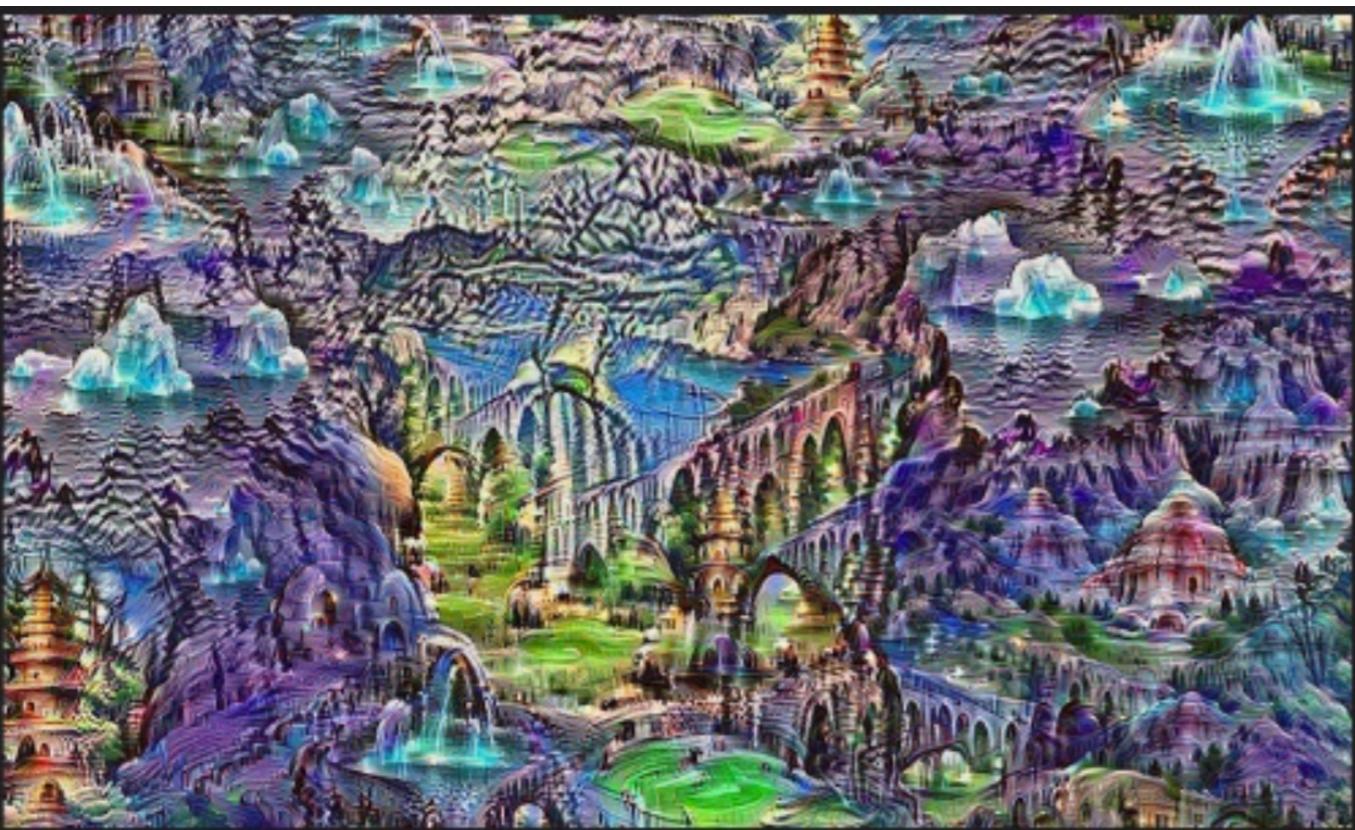
Transfer Learning



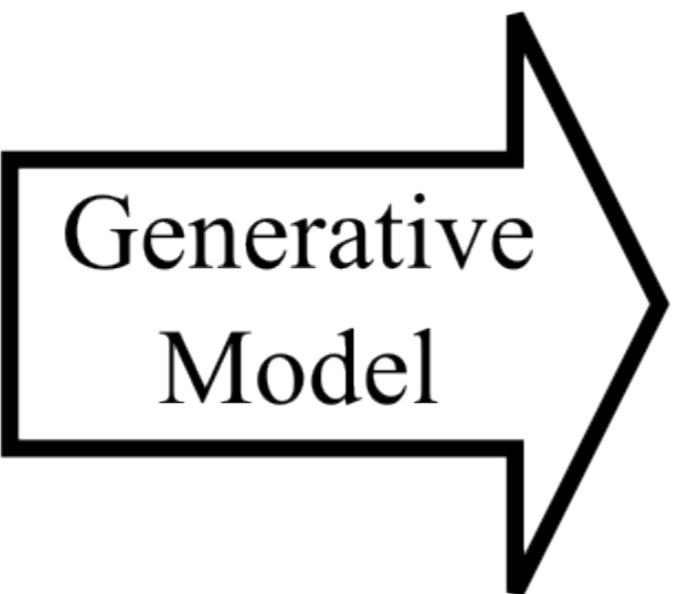
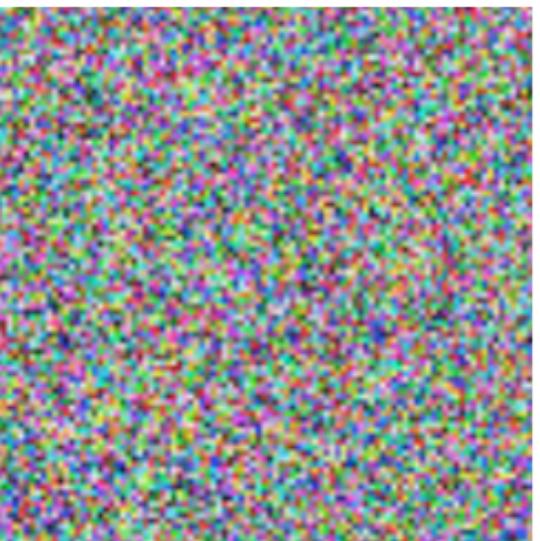
<https://keras.io/api/applications/>

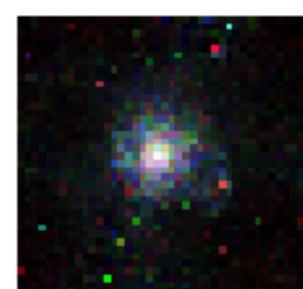
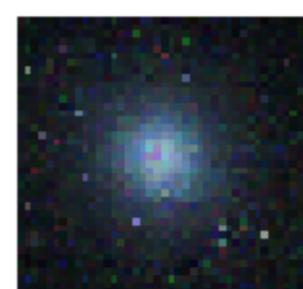
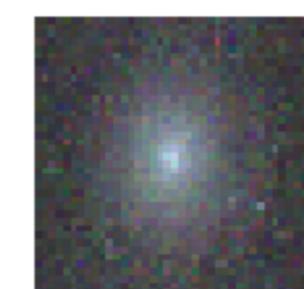
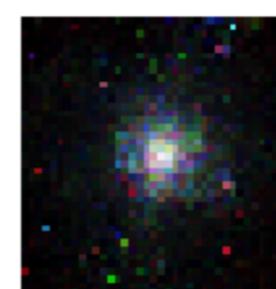
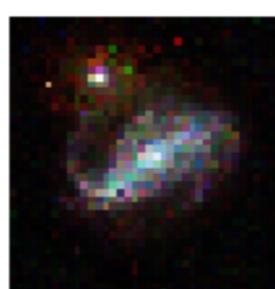
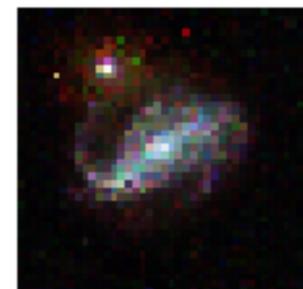
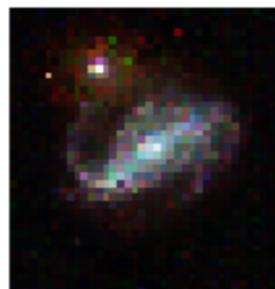
GAN (Generative Adversarial Network)





Noise $\sim N(0,1)$

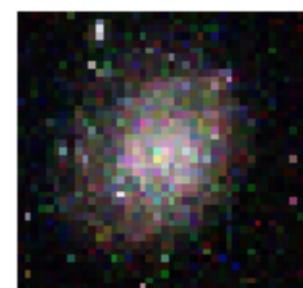
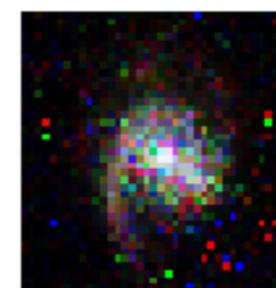
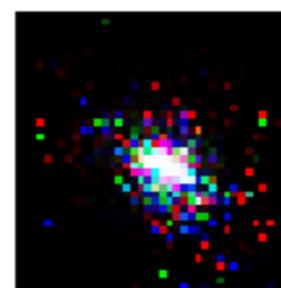




Elliptical

Barred Spiral

Spiral



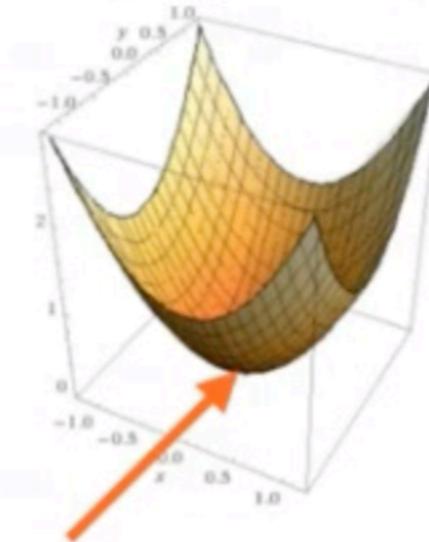
GAN usage

- Mock catalogs
- Denoising
- Compete gaming

Warning: Hard to train

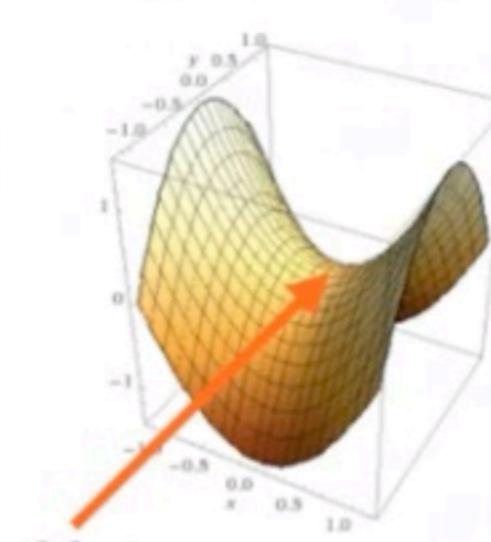
Adversarial Machine Learning

Traditional ML:
optimization



Minimum
One player,
one cost

Adversarial ML:
game theory



Equilibrium
More than one player,
more than one cost



2. Simulate Attack

Visual

Code

Adversarial noise type

C&W Attack



Determine strength

None low med high

3. Defend attack

Gaussian Noise

None low med high

Spatial Smoothing

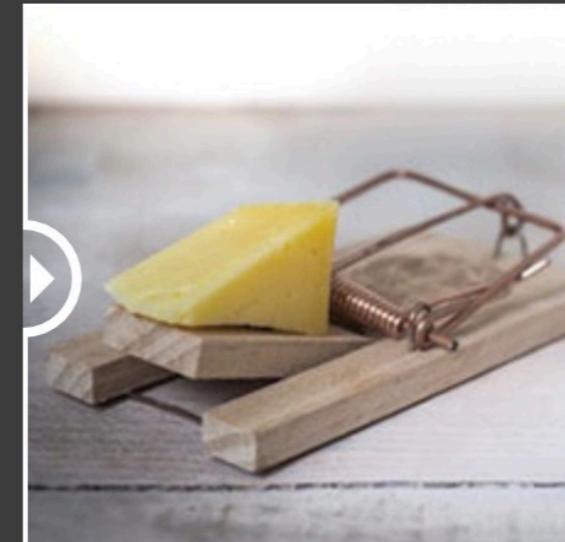
None low med high

Feature Squeezing

None low med high

Original

Modified

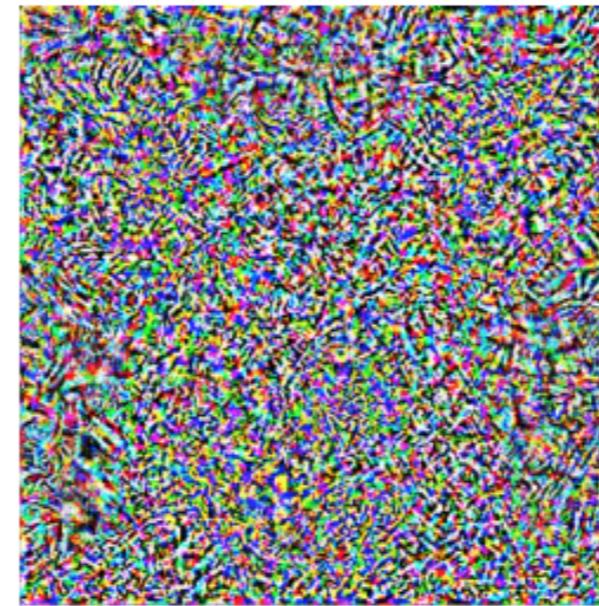


“pig” (91%)



+ 0.005 x

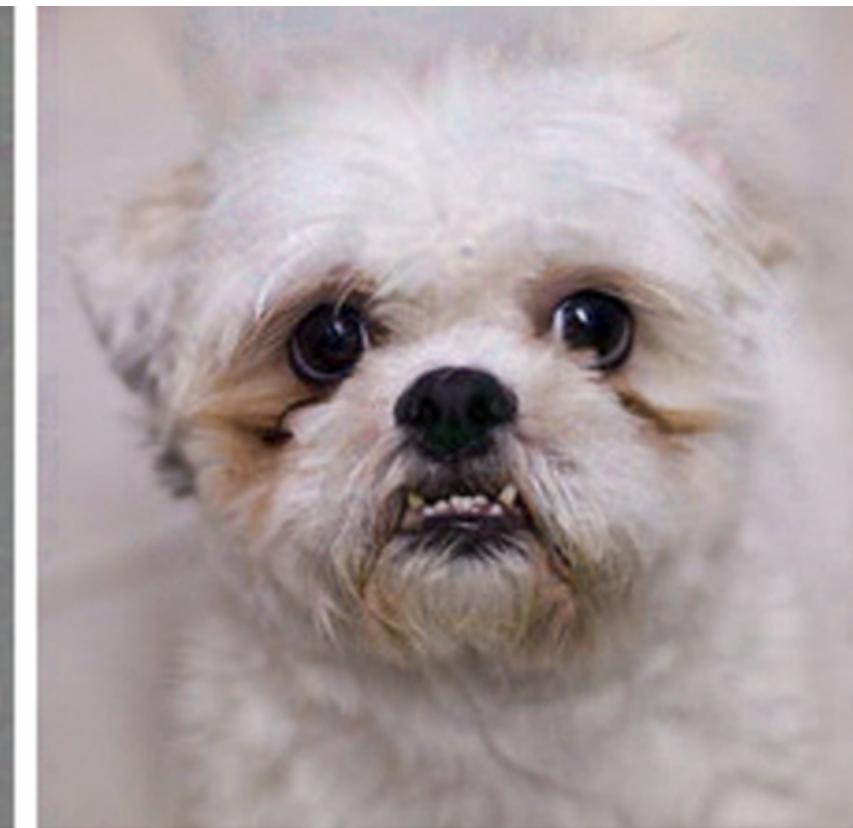
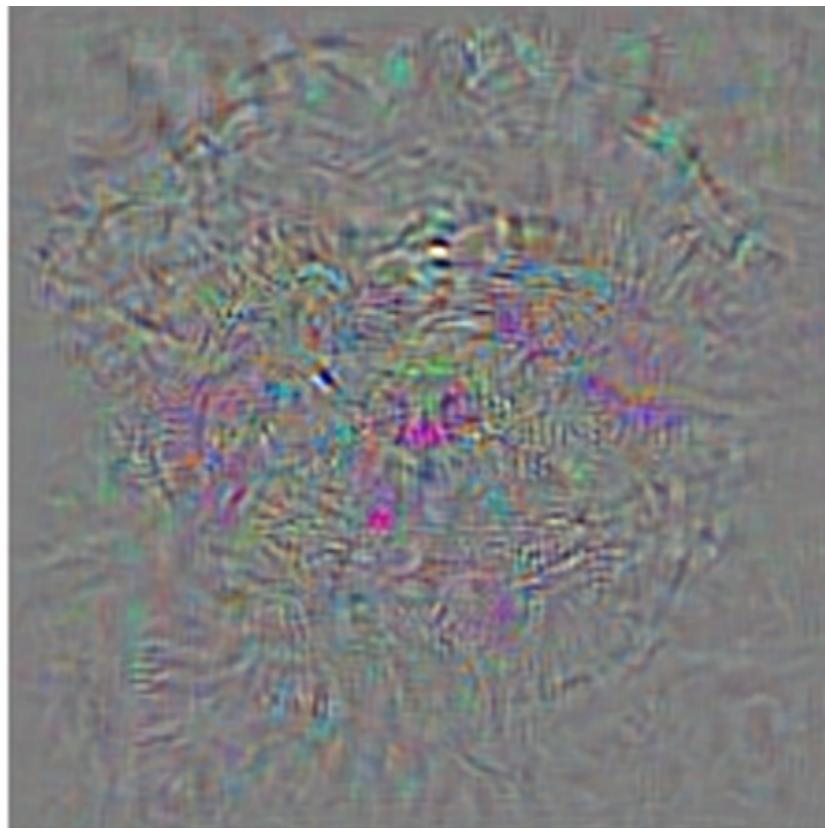
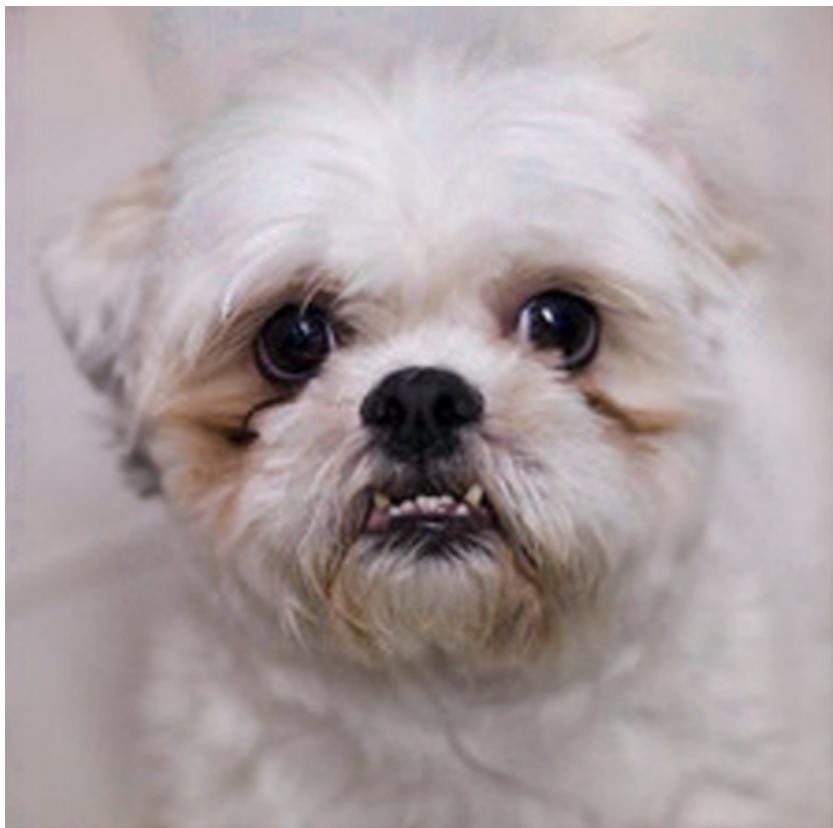
noise (NOT random)



“airliner” (99%)



=



dog

+noise

ostrich

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:

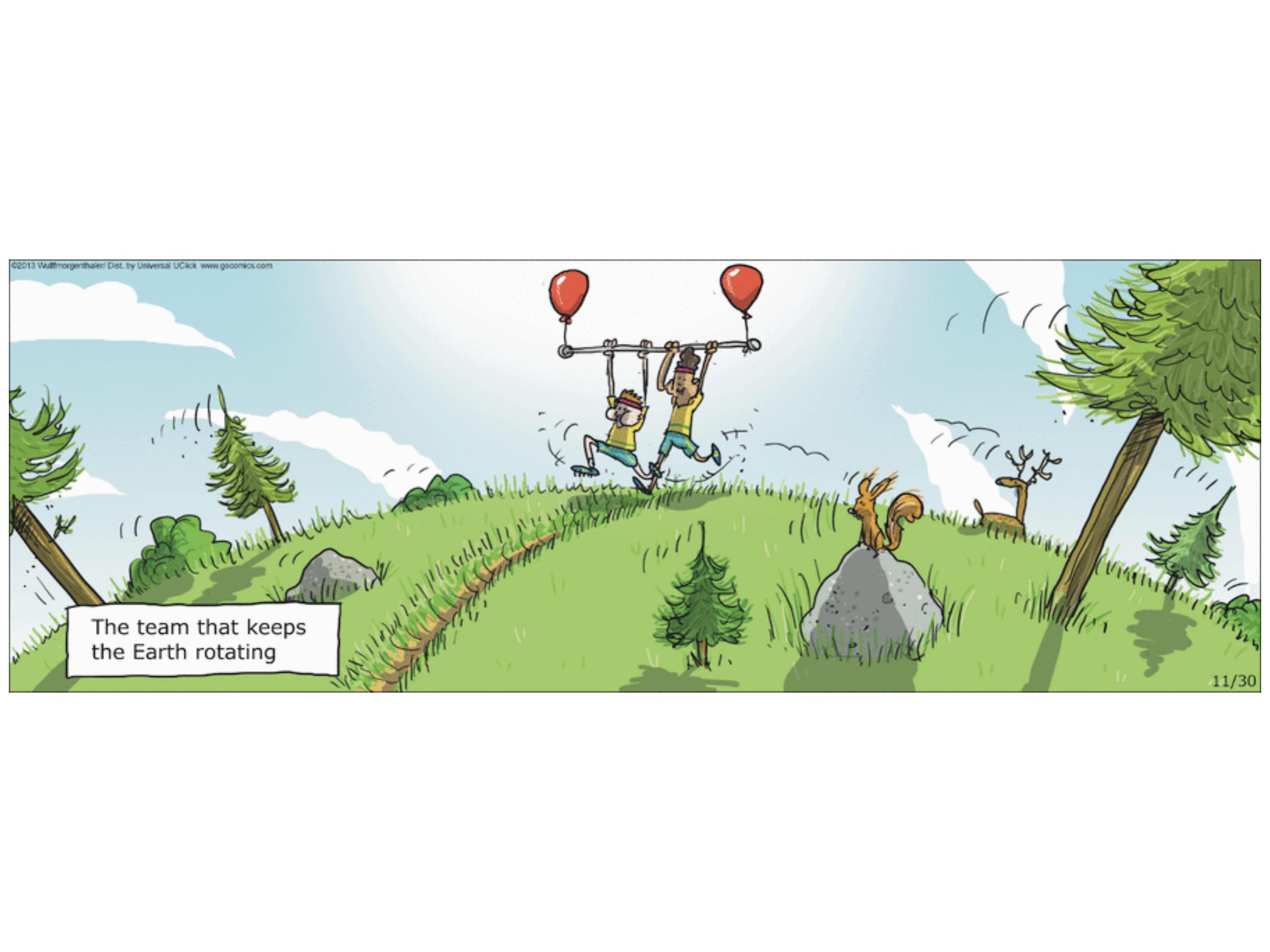
```
score = model.evaluate(x_test, y_test, verbose=0)
```

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

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

in TensorFlow

in anaconda for python 3.6



The team that keeps
the Earth rotating