(Deep) Neural Network Basics

Kyriakos Efthymiadis

3, May 2019

VRIJE
UNIVERSITEIT
BRUSSEL

What are they?

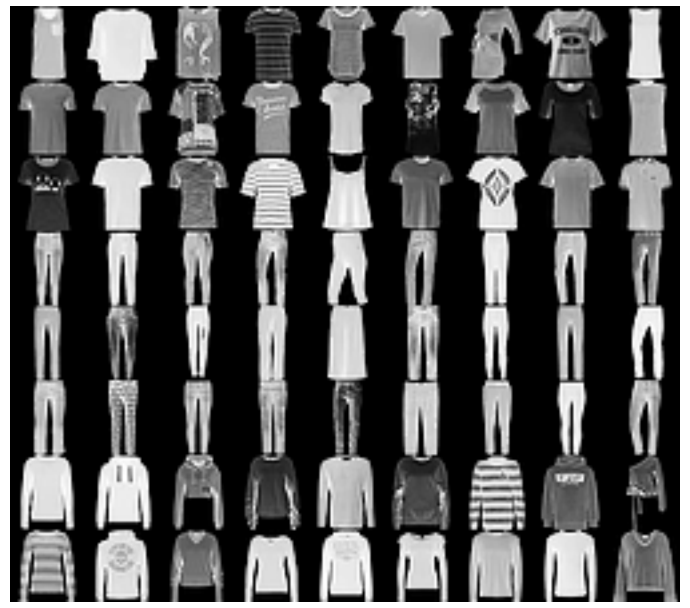
- Linear transformation + non-linear activation functions
- Massive modeling power by composing large structures of these modules
- Inspired by how the brain works, very coarse approximation
  - Humans neuron switching time ~ .001s
  - # neurons ~ 10^{10}
  - Scene recognition ~ .1s
- ANNs
  - Many neurons
  - Many connections
  - Highly parallel distributed process

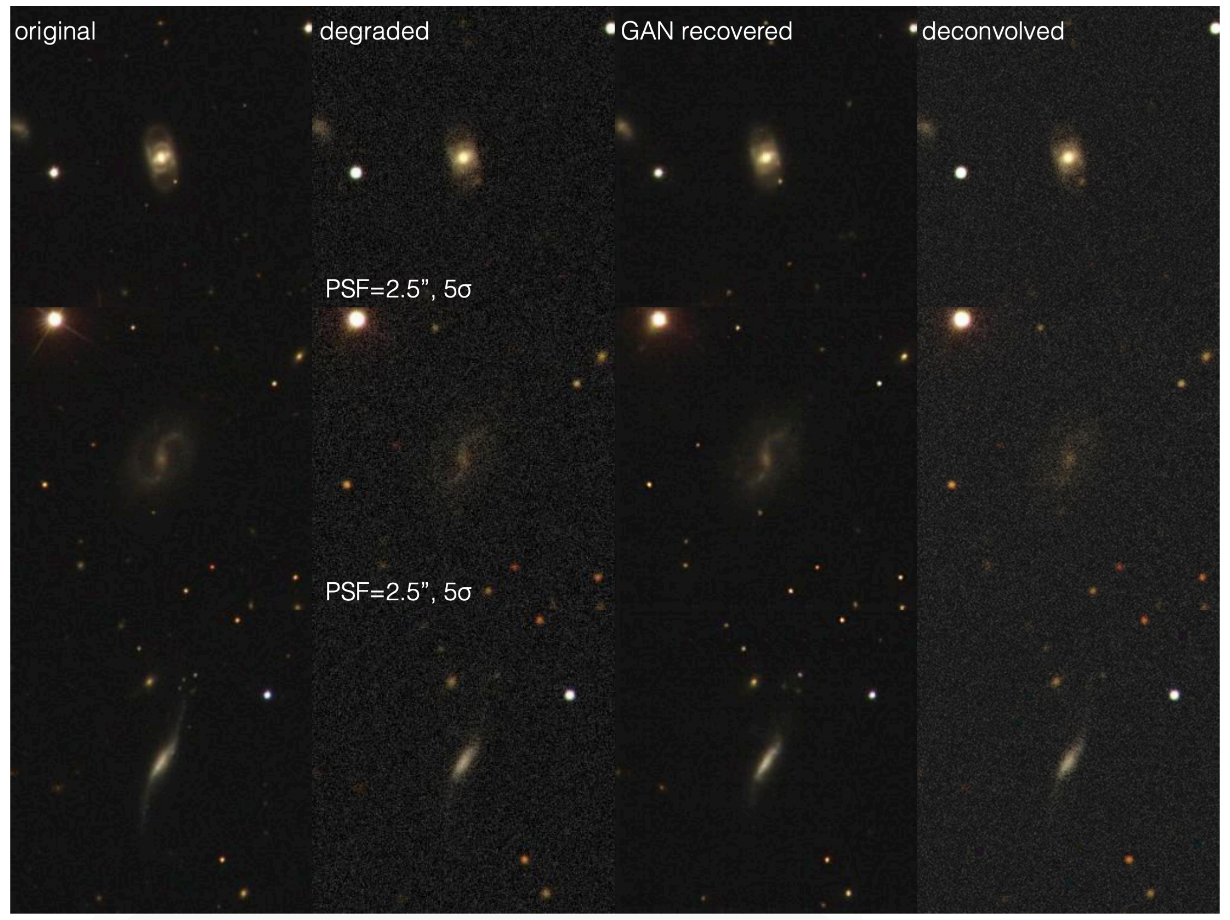
When to consider?

- High dimensional input
- Structure in data
- Explainability is not an issue
- Now pretty much state of the art in most tasks

Good at

- Computer vision
- Machine translation
- Speech recognition
- Self-driving cars
- RL
- Neural art
- …and more

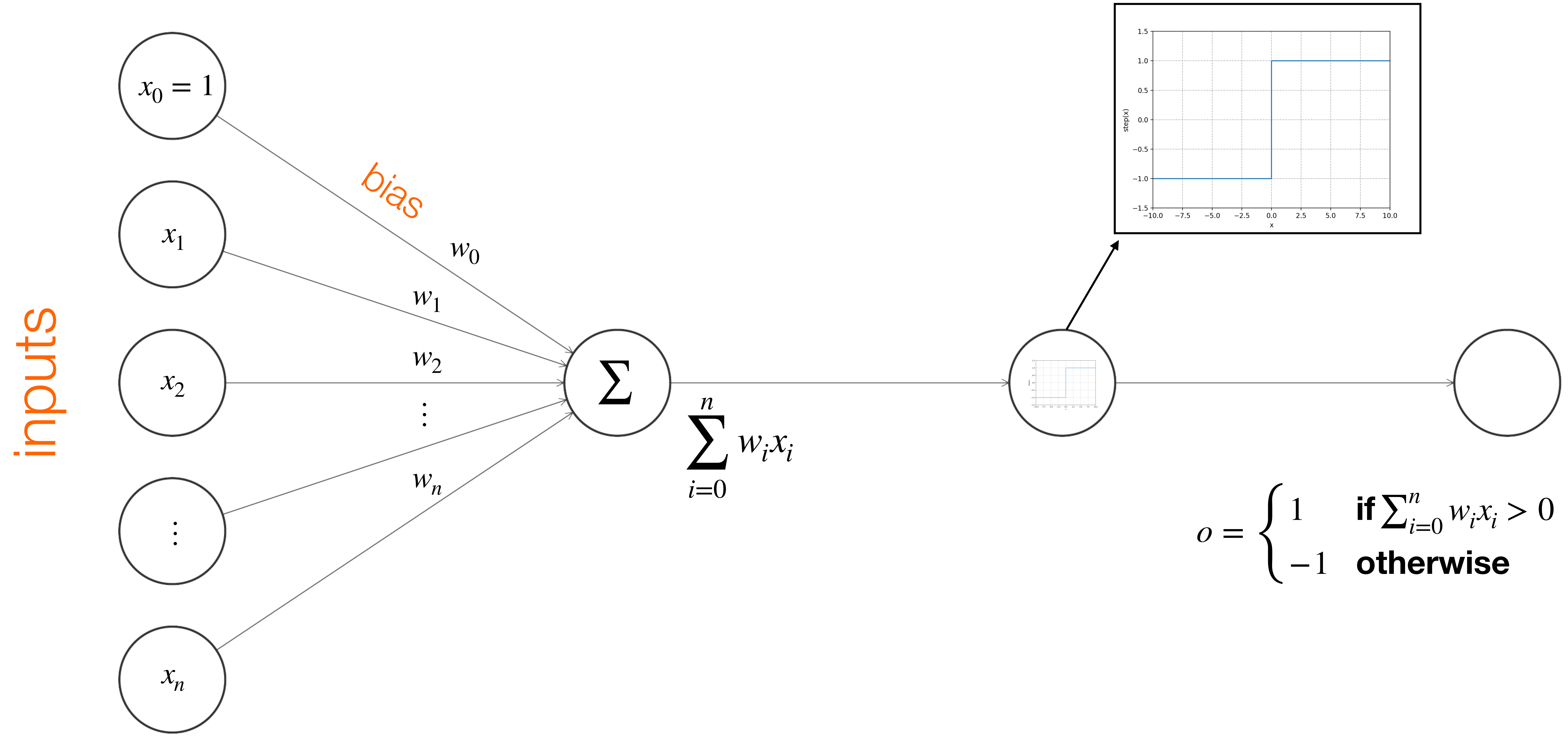




Some terminology

- Unit = Neuron
- Activation Function = Non-linearity
- Linear Transformation + Activation Function = Layer
  - …or not
- Dense Layer = Fully Connected
- Convnet = Convolutional Neural Network
- Filter = Kernel (in convnets)

Perceptron



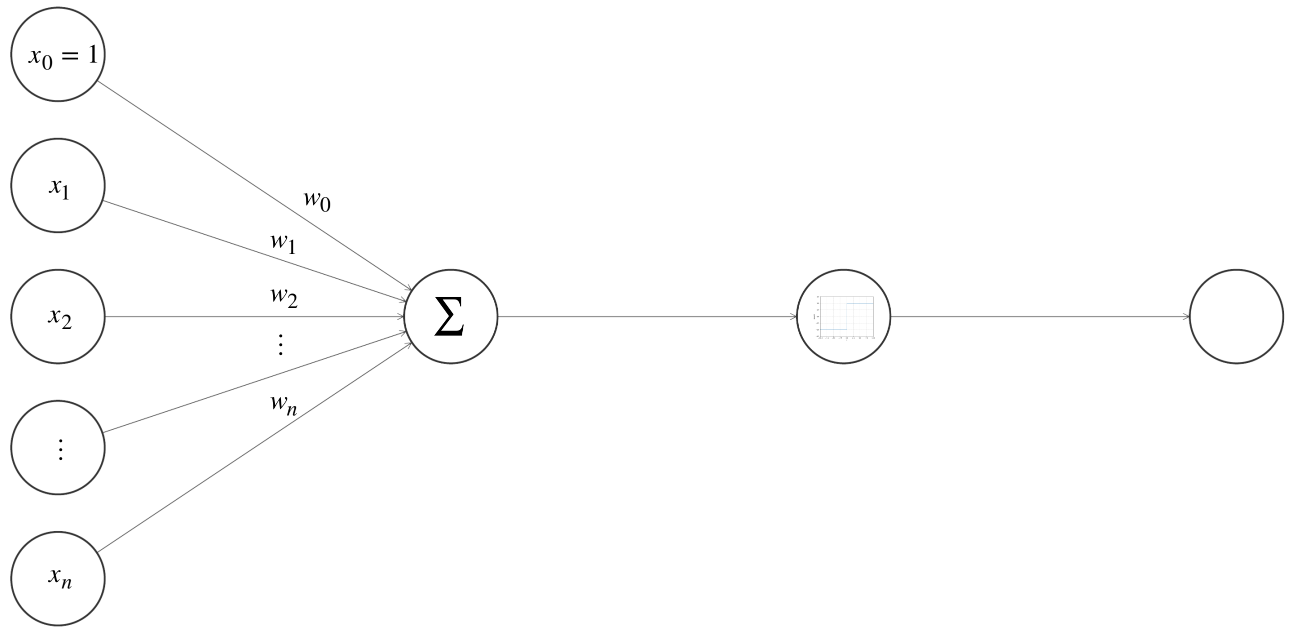
$$o = \begin{cases} 1 & \textbf{if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 & \textbf{otherwise} \end{cases}$$

Rosenblatt 1958

Perceptron

Output

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \ldots + w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$



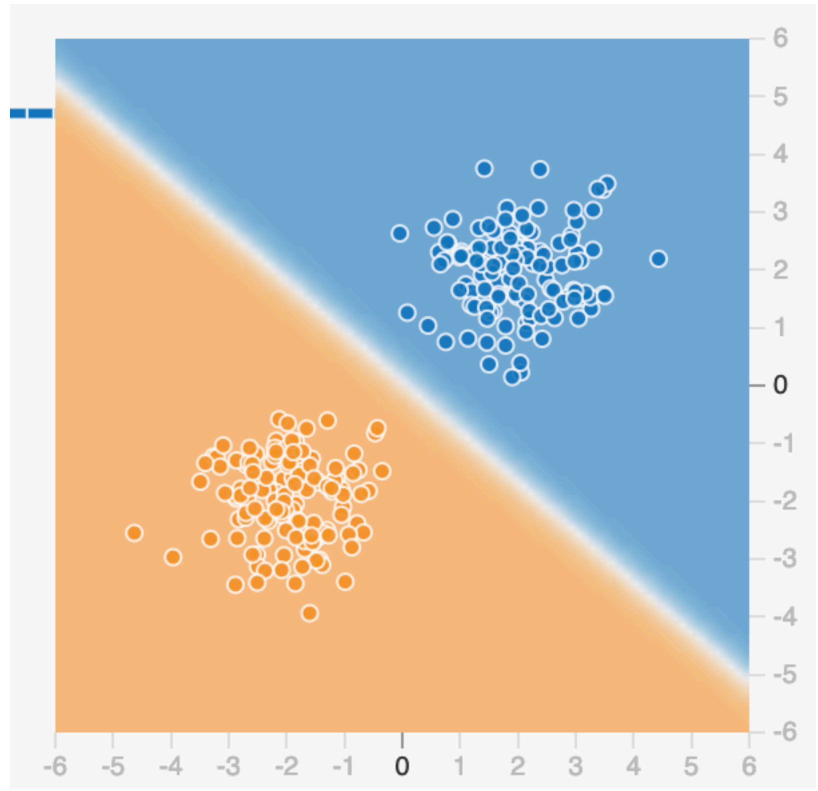
In vector notation

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise} \end{cases}$$

or

$$o(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x}^\top \mathbf{w} > 0 \\ -1 & \text{otherwise} \end{cases}$$

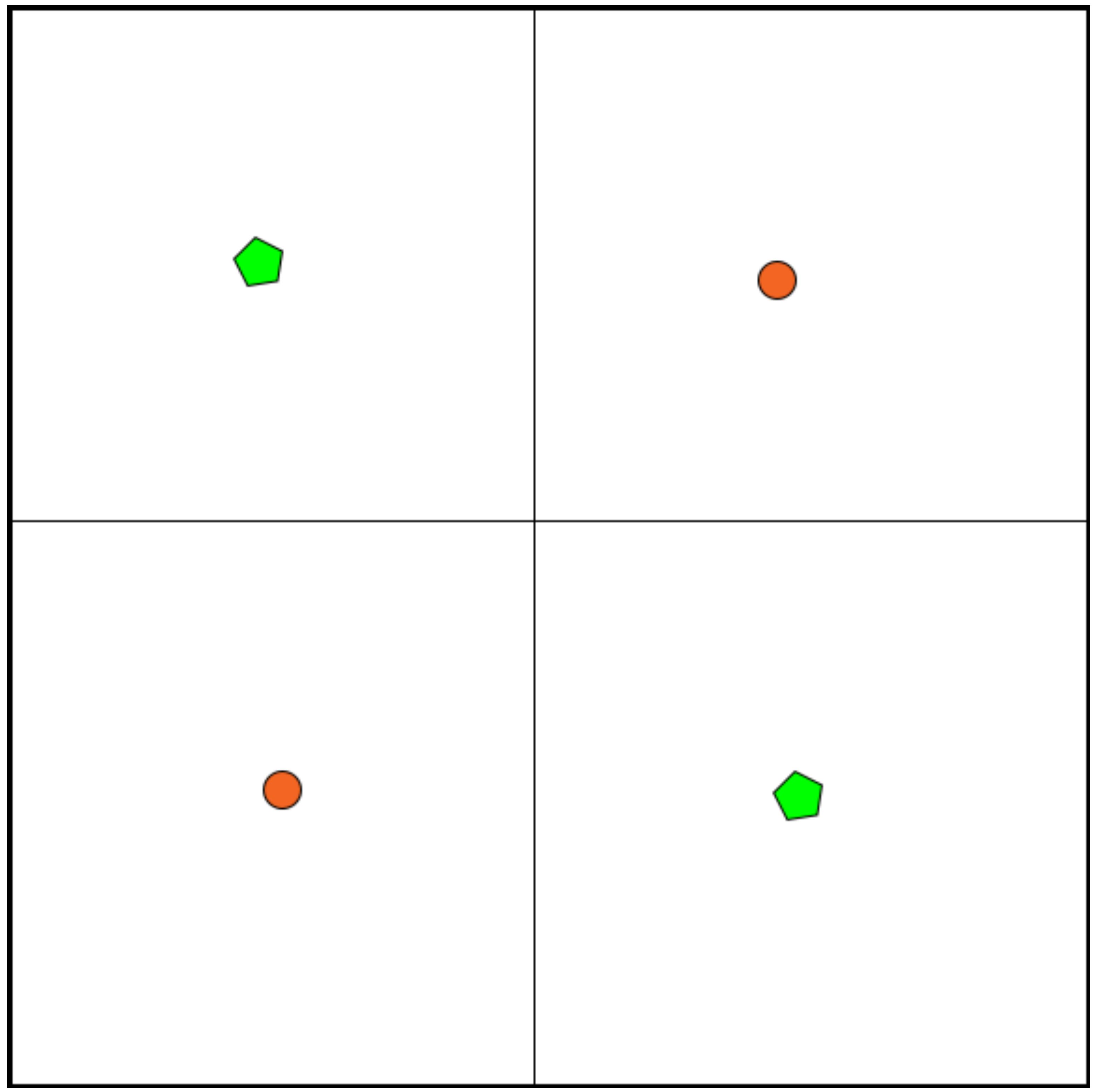
Representation power of Perceptron



- Can represent many boolean functions
- What weights represent AND?
- What weight represent OR?

Representation power of Perceptron

- How about XOR?
- Not linearly separable
- We need something more



Perceptron training rule

Classification output ± 1

1. Random weights
2. Apply perceptron to each example
3. Modify weight on misclassification
4. Repeat

Perceptron training rule

Weight are changed according to the training rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Perceptron training rule

Example

$$\Delta w_i = \eta(t - o)x_i$$

- If $t = o$ then no change
- If $t - o > 0$ then $w_0 + w_1x_1 + \ldots + w_nx_n < 0$
  - but needs to be > 0
  - if $x_i > 0$ then increase weight
  - else decrease
- What happens in the opposite case?
- Proven to converge if data linearly separable and η sufficiently small

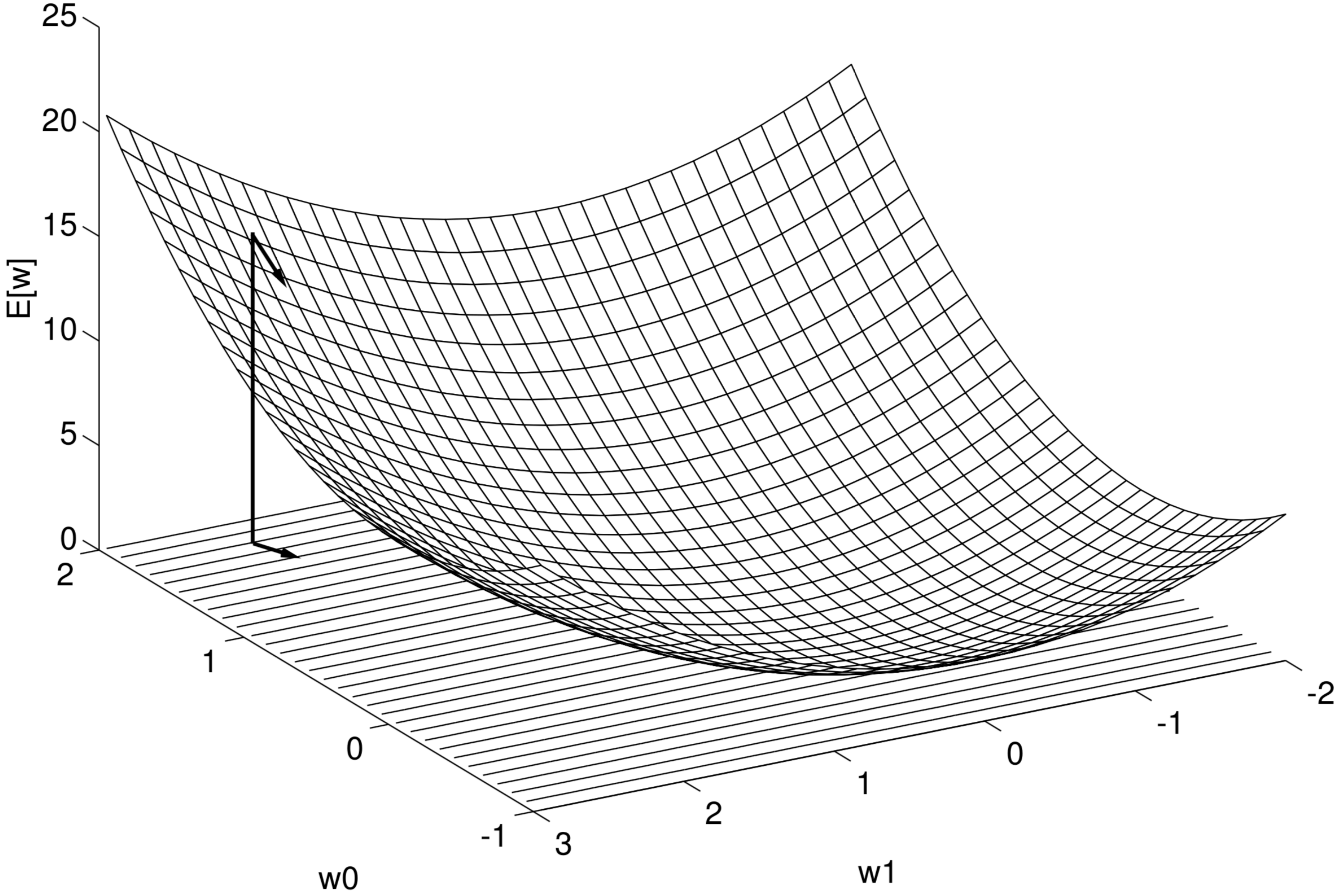
Gradient Descent

- Consider simple linear unit with no threshold
- We want to learn those weight that minimize the training error

$$E[\vec{w}] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

where *D* is the set of training examples

Hypothesis space



Gradient Descent

- Finding the direction of steepest descent
  - compute the derivative of the error with respect to weights

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right]$$

- Training rule

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

$$\Delta\vec{w} = -\eta\nabla E(\vec{w})$$

Gradient Descent

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\
&= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x_d}) \\
&= \sum_{d \in D} (t_d - o_d)(-x_{id})
\end{aligned}
$$

Gradient Descent

- Important general paradigm for learning
- Search a space of continuous parameterized hypotheses when the error is differentiable

- Practical difficulties
  - slow convergence
  - multiple local minima

SGD

- Common variation to alleviate issues
- Compute weight updates after each example

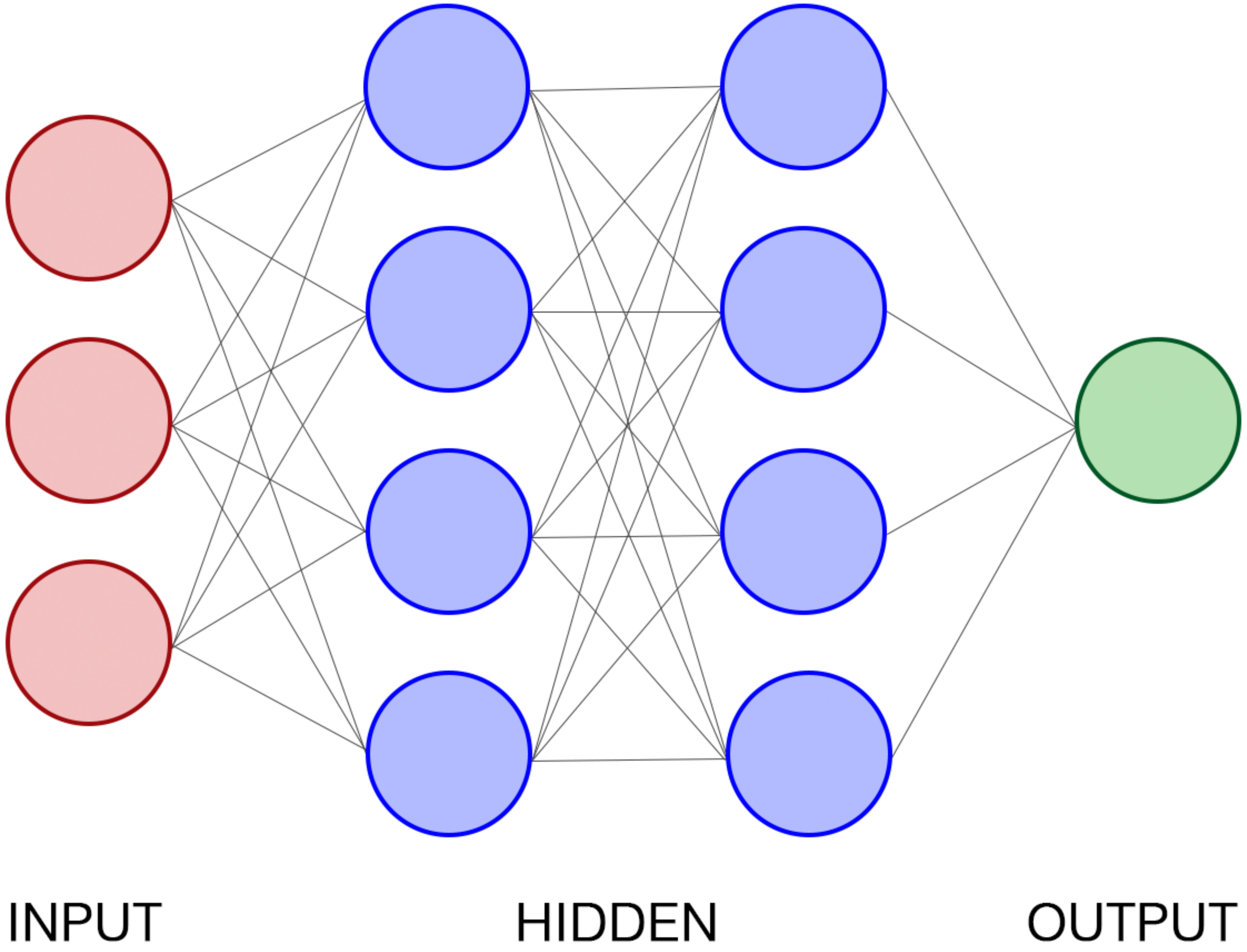
$$\Delta w_i = \eta(t - o)x_i$$

- Minibatch is in between

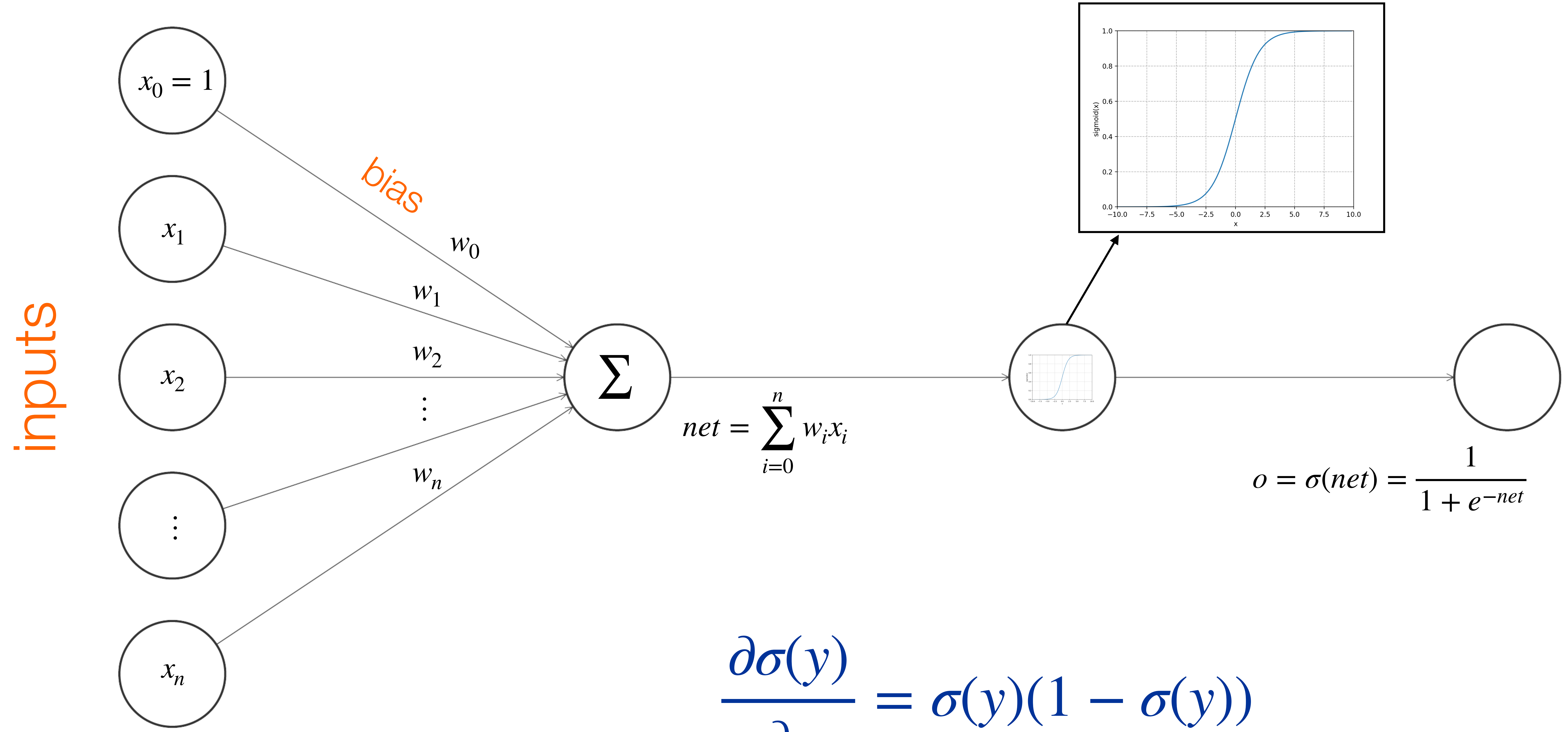
MLP of Sigmoid Units

- Perceptrons only linearly separable
- Add non-linear differentiable activations
- Add multiple layers
- Able to capture highly non-linear decision surfaces
- Trained using backpropagation

MLP



Sigmoid Units



$$\frac{\partial \sigma(y)}{\partial y} = \sigma(y)(1 - \sigma(y))$$

Backpropagation

- Learn the weights of MLP

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

- Learn in a large hypothesis space
  - defined by all possible weight values
- Gradient descent to minimize error

Backpropagation - 2 Layers

Initialize weights

Until satisfied do

1. Input training example through network and compute output
2. For each network output unit *k* calculate error
$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$
3. For each hidden unit *h* calculate error
$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$$
4. Update each network weight
$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji} \qquad \Delta w_{ji} = \eta \delta_j x_{ji}$$

Training in a nutshell

- Initialize weights
- Pass examples to network and compare to target
- Calculate the error
- Calculate the derivative wrt to the weights of the network
- Propagate backwards the gradients and update weights
- Do until satisfactory results
- For more about neural nets read Deep Learning(Goodfellow et. al 2016) https://www.deeplearningbook.org/

Going deeper

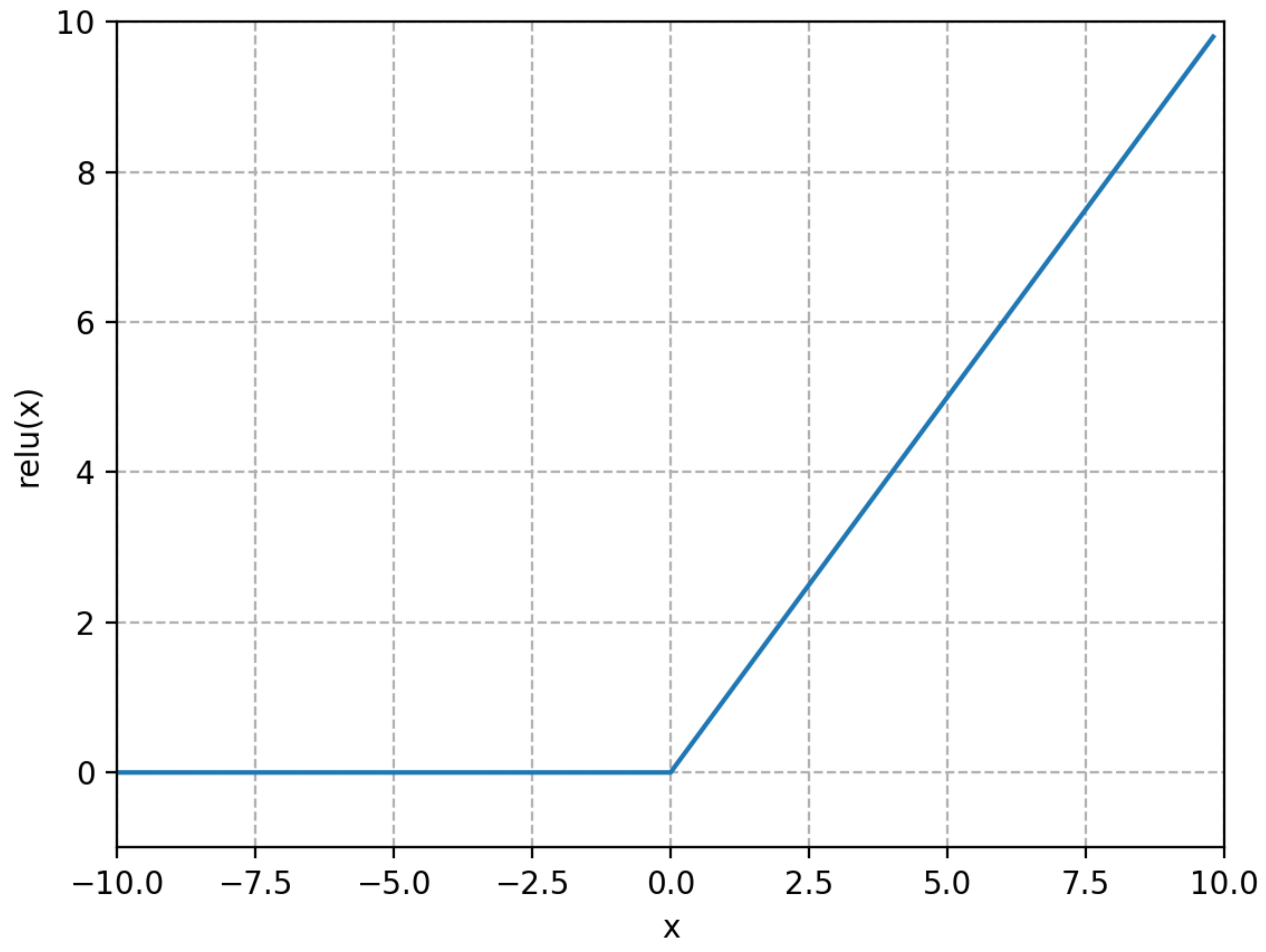
- Provides more benefits
- 1 hidden layer is universal FA, but requires many units
- Deep nets are more powerful
- Break down problem in a hierarchical fashion
  - edges to shapes to objects to scenes
- Multiple points in input, map to same output
- Results in exponentially more linear regions when deep, compared to polynomial when wide [Montufar et. al 2014]

Some practical issues

ReLU

- ReLU instead of sigmoid
- Simpler and cheaper than sigmoid
- Favorable gradient properties

$$y_i = maximum(0, x_i)$$



Some practical issues

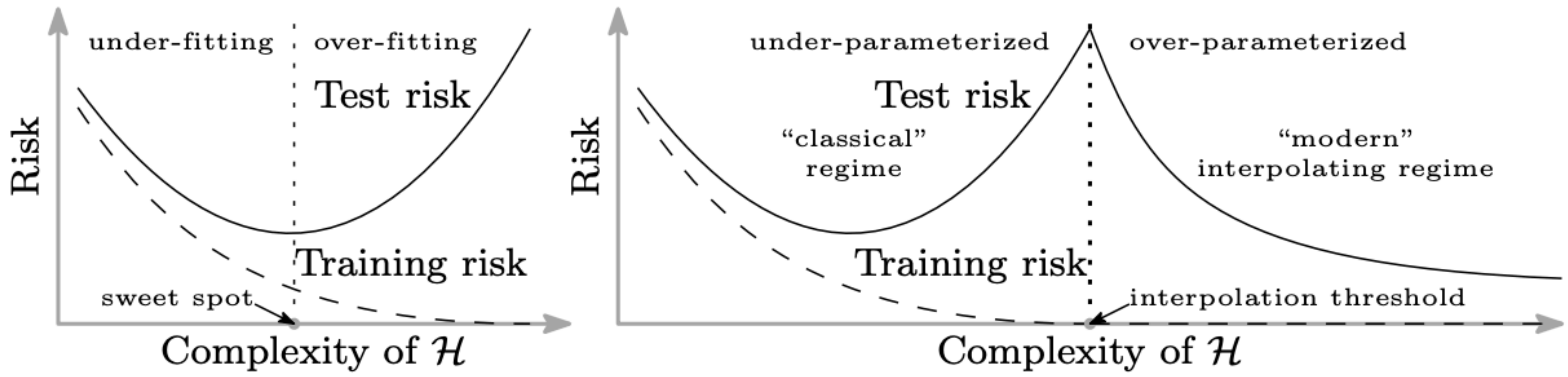
Overfitting and regularization

- Early stopping
- Weight decay
  - keeps weights from growing large
  - not for relu
- Dropout
  - randomly set activations to 0
  - promotes “individuality”

Some practical issues

Overfitting and regularization

- Classical bias-variance doesn't apply in deep learning
- # parameters not a good measure of inductive bias
- Large models might be able to discover better functions



Belkin et. al. 2018(arxiv)

Some practical issues

Weight initialization

- Weight should be small
  - too small results in vanishing gradients
  - too big in exploding
- Research on this
  - Xavier initialization and more
- Batch normalization
  - scale and offset activations
  - good for training too

Some practical issues

Debugging

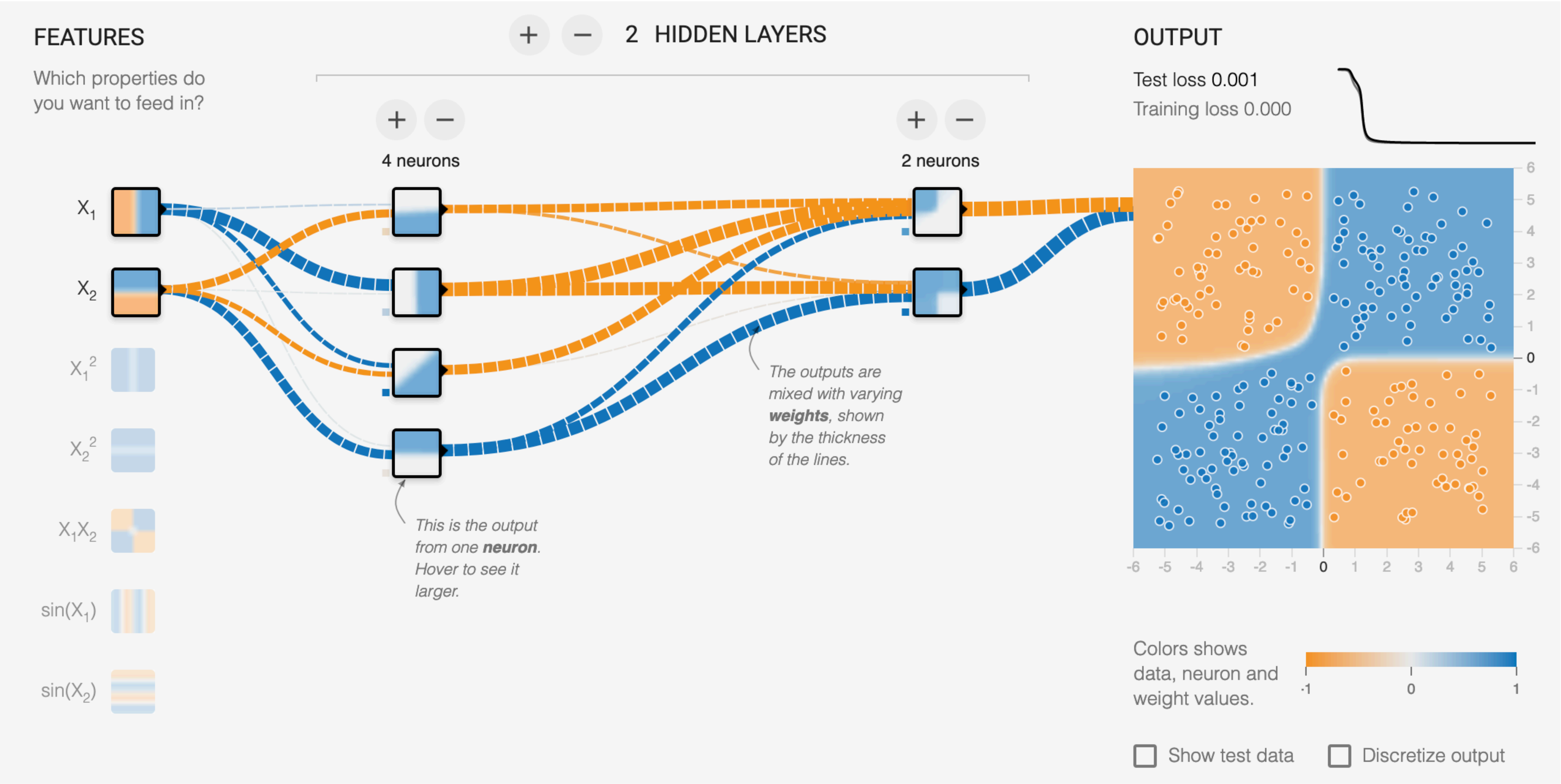
- Check your loses
- Check your gradients
- Check dead units
- Good tip, try to overfit in a small set of your problem
  - if loss != 0 on small datasets then something must be wrong

Some practical issues

Implementation

- In general you will not implement units, layers, activations, losses
- Many great libraries/frameworks
- TensorFlow by Google
- PyTorch from Facebook
- Example later

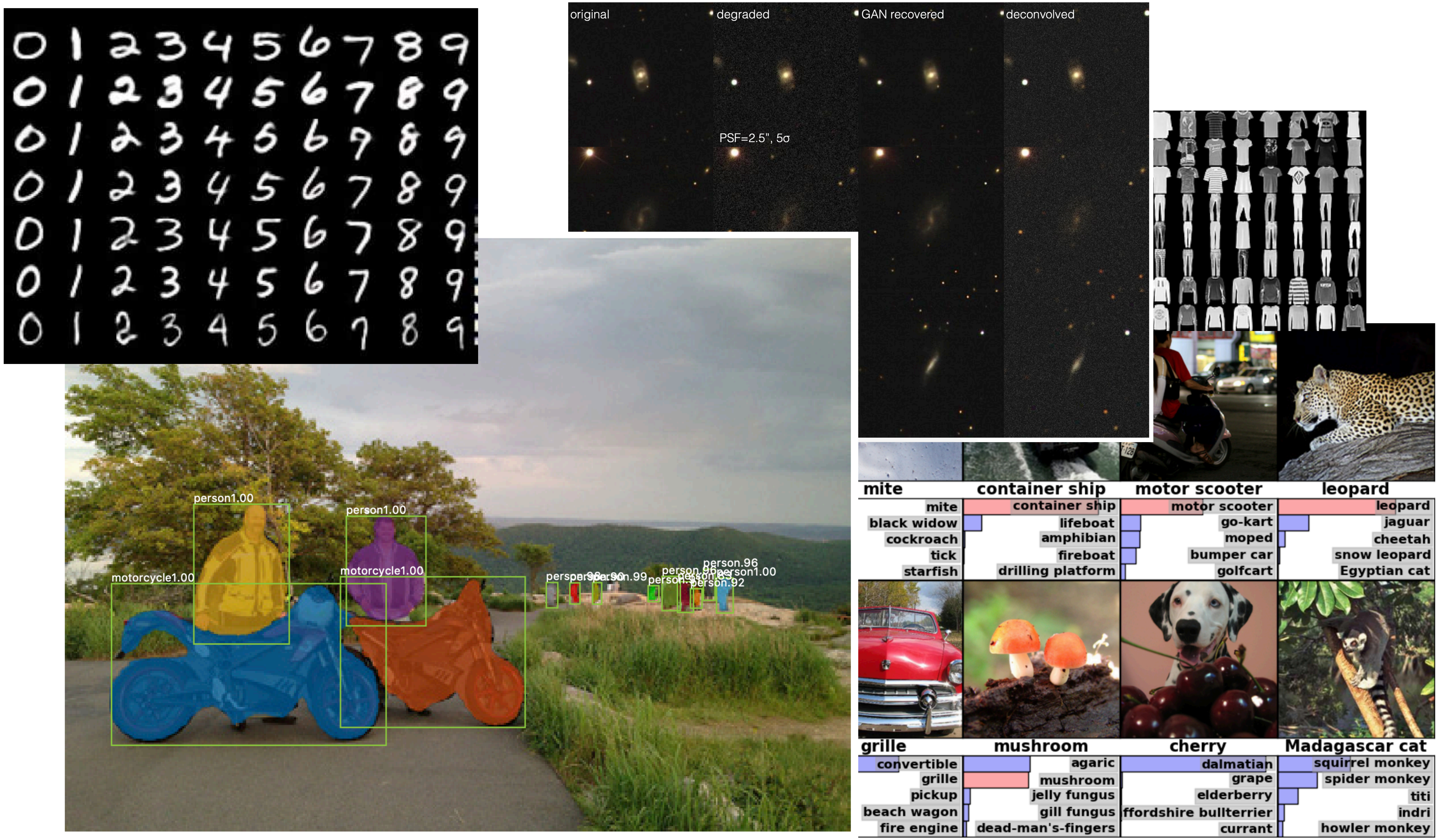
Tensorflow playground



https://playground.tensorflow.org

Convolutional Networks

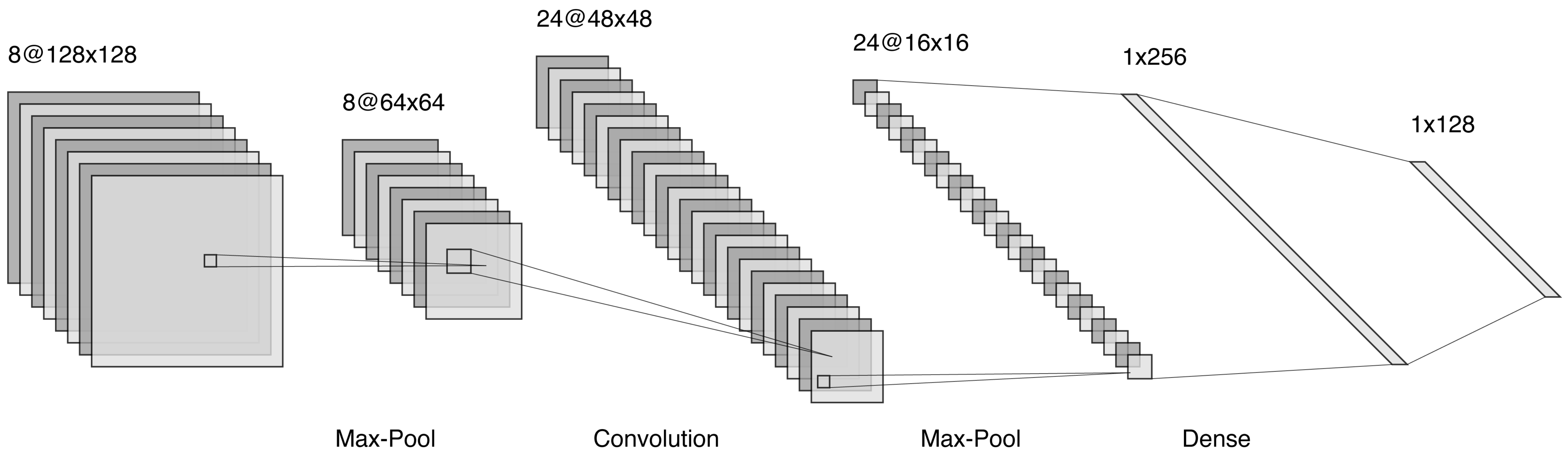
Convnets


original
degraded
GAN recovered
deconvolved
PSF=2.5", 5σ
person1.00
person1.00
motorcycle1.00
motorcycle1.00
person.96
person.92
mite
container ship
motor scooter
leopard
mite
black widow
cockroach
tick
starfish
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
go-kart
moped
bumper car
golfcart
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
mushroom
cherry
Madagascar cat
convertible
grille
pickup
beach wagon
fire engine
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
dalmatian
grape
elderberry
ffordshire bullterrier
currant
squirrel monkey
spider monkey
titi
indri
howler monkey

Convnets

- Can take advantage of spatial information
- Provide translation invariance
- Weight sharing
- Extract features hierarchically in deep networks(edges to scenes)

Convnets

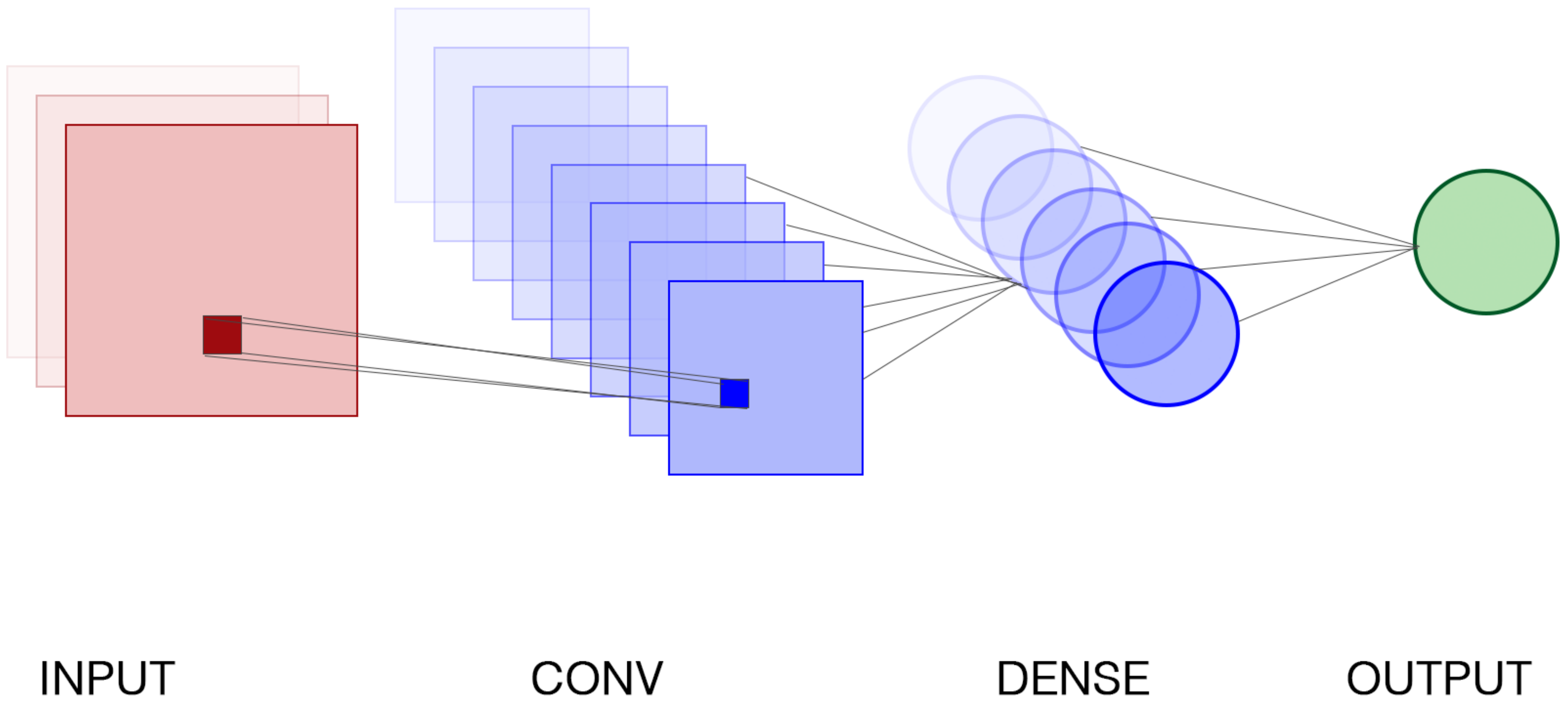


24@48x48
8@128x128
24@16x16
1x256
8@64x64
1x128
Max-Pool
Convolution
Max-Pool
Dense

Convnets

Built using a series of

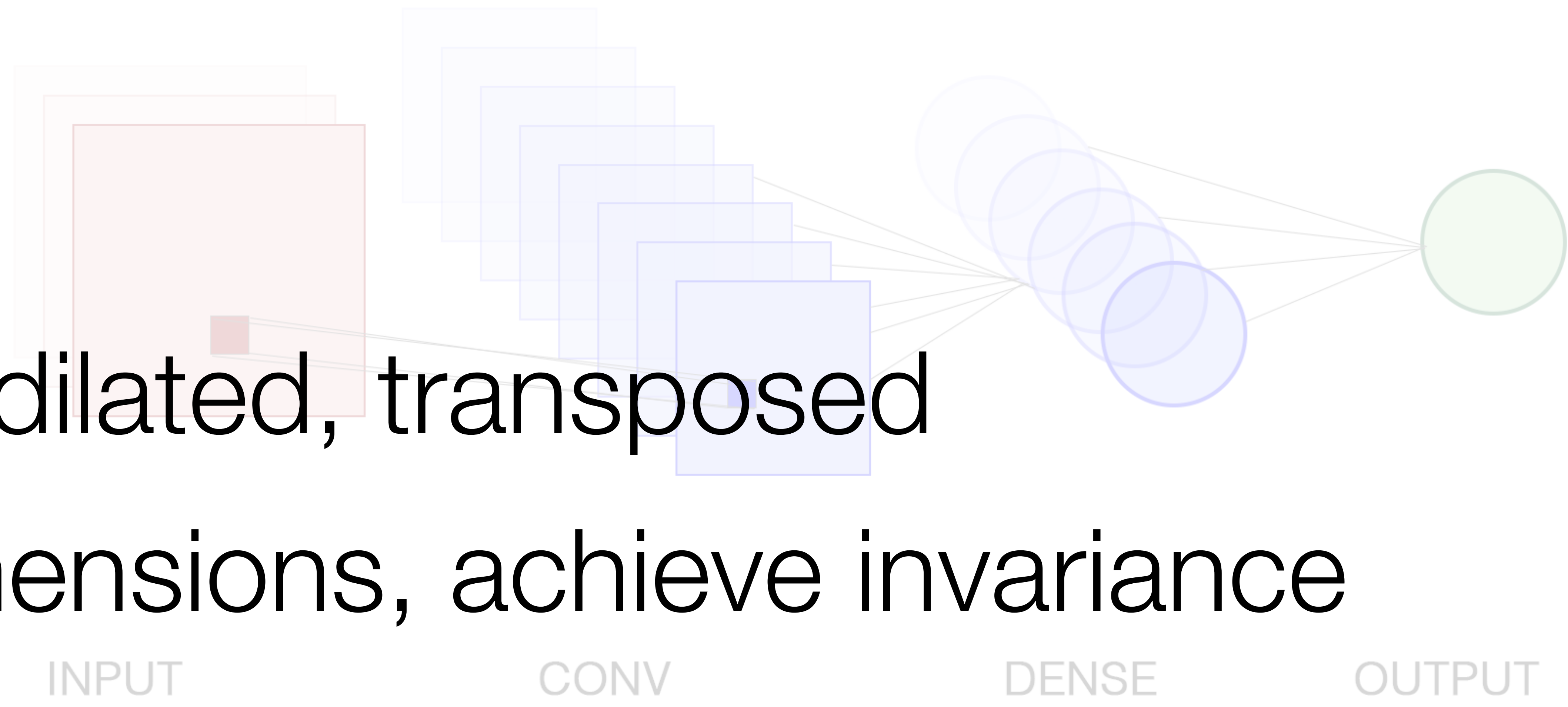
- Convolutional layers
- Non-linearities
- Pooling layers
- Output



Convnets

At each layer

- Convolve input with sets of weights(filters)
- Produce feature maps
- Filters choices 3x3, 5x5, 7x7
- Can be valid, same, strided, dilated, transposed
- Pooling to reduce spatial dimensions, achieve invariance
- Learning filter weights with back-propagation



Convolution - Cross Correlation

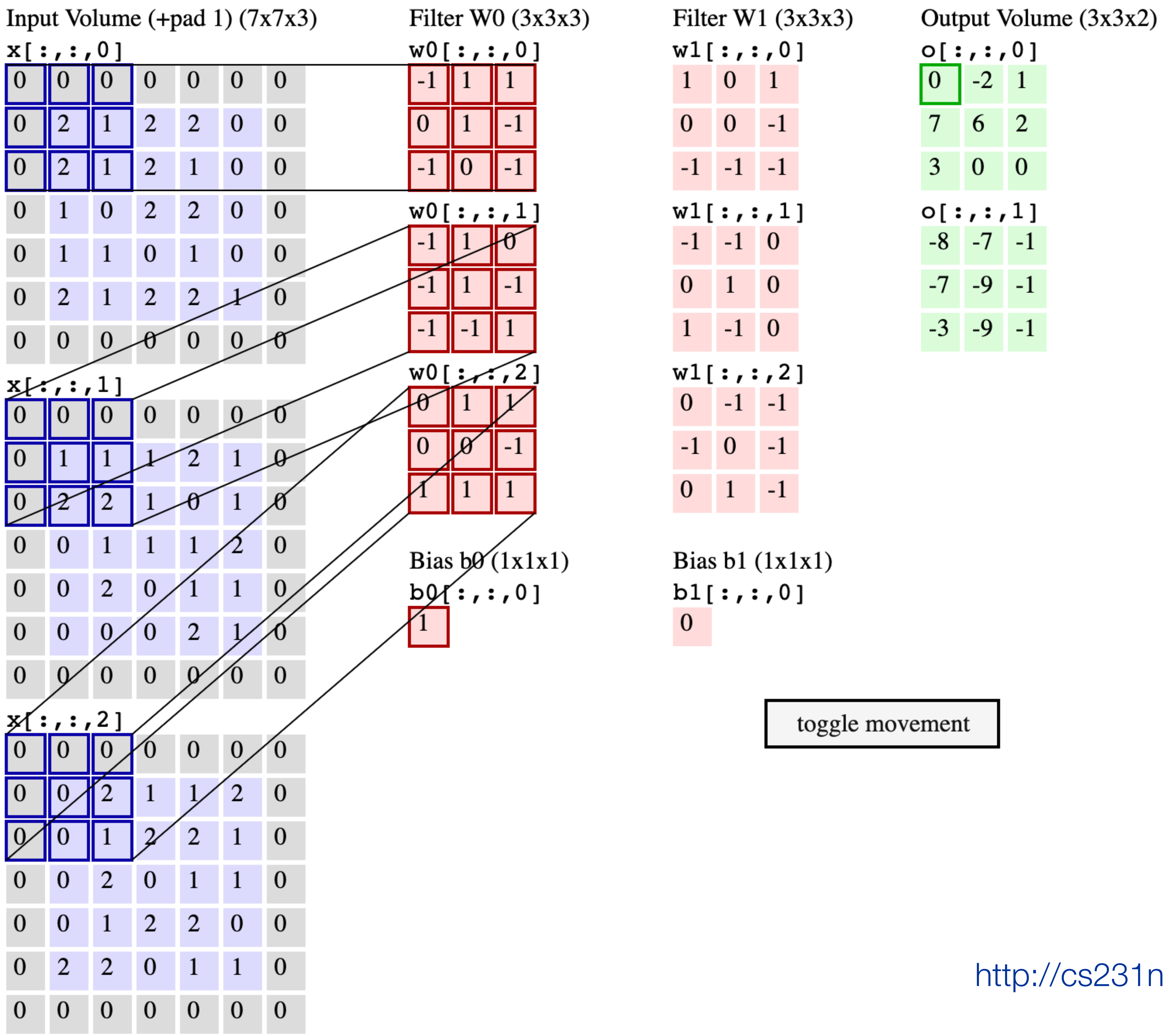
Convolution operation

$$a^l = \sigma\Big(\sum_m w_m^l * x + b^l\Big)$$

actually cross correlation

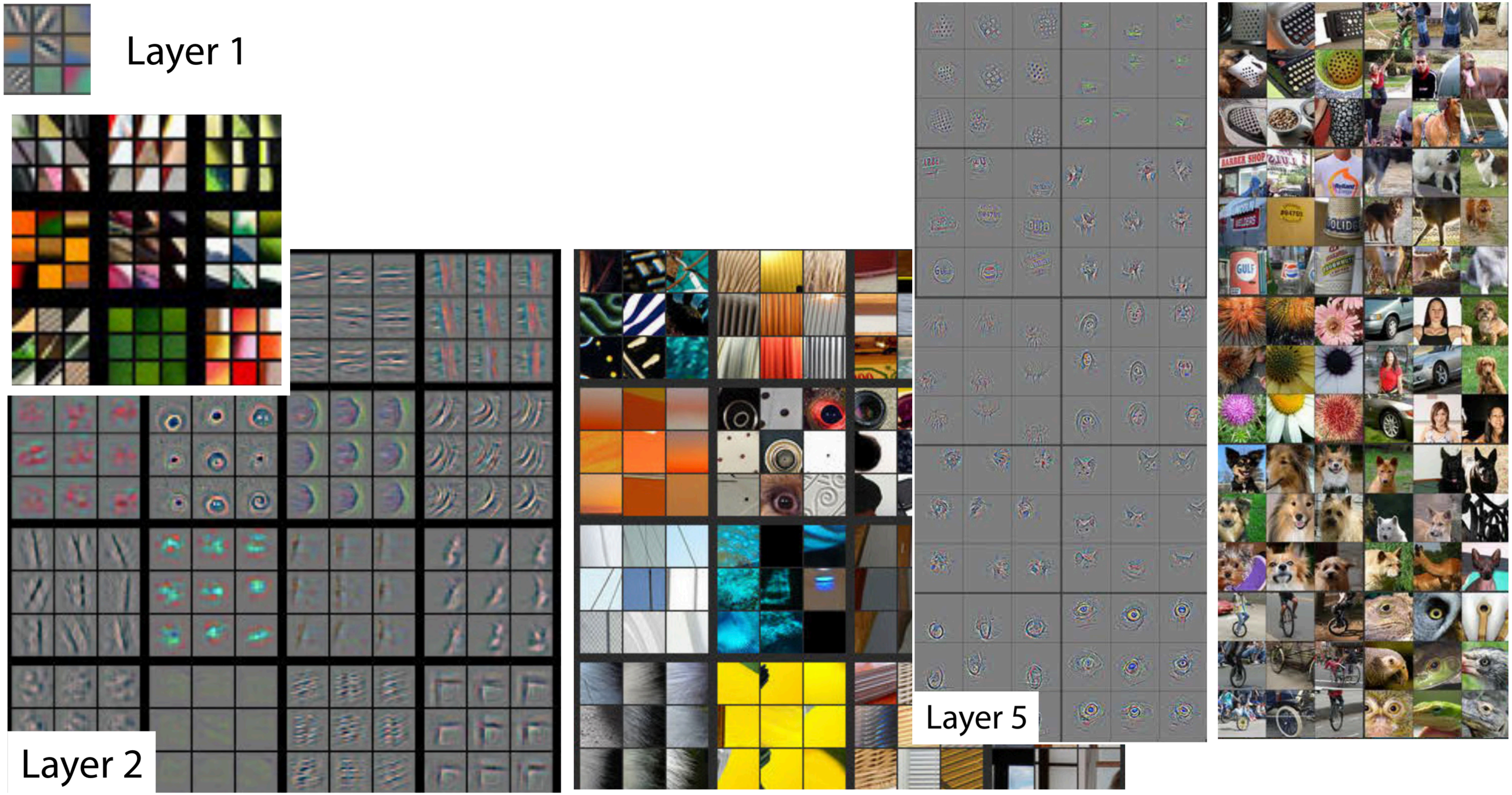
More in Deep Learning [Goodfellow et. al, 2016]

Convolution example



http://cs231n.github.io/convolutional-networks/

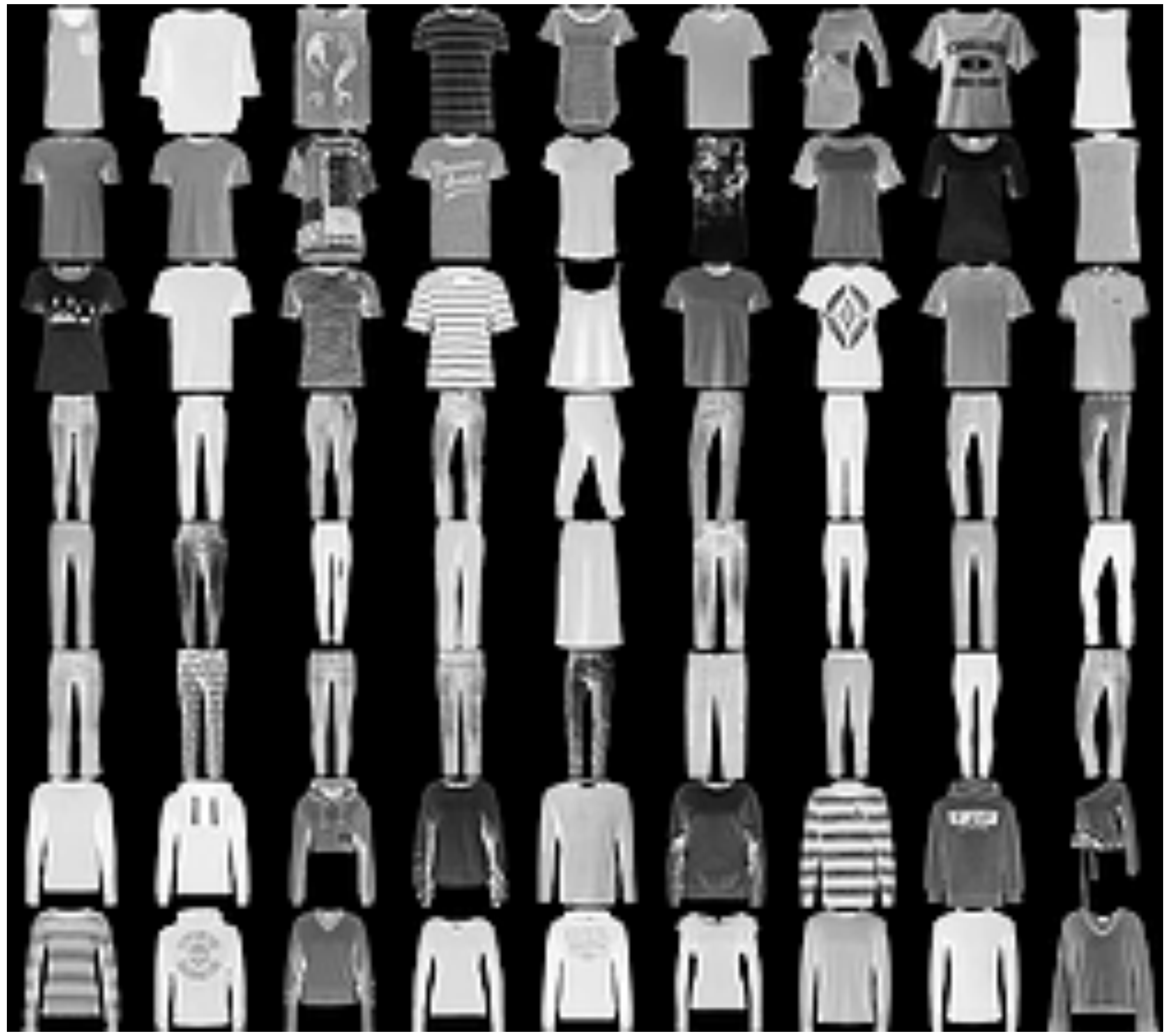
What do filters learn?



An Example in Image Recognition

Fashion MNIST

- Dataset by Zalando to replace MNIST
- 28x28 grayscale images
- 10 classes



Steps

- Load data
- Define network structure
- Define training loop
- Decide on loss(criterion)
- Start learning
- Use early stopping

Model in PyTorch

```
class FashionSimpleNet(nn.Module):

    """ Simple network"""

    def __init__(self):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1,32, kernel_size=3, padding=1), # 28
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2), # 14

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2) # 7
        )
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(64 * 7 * 7, 128),
            nn.ReLU(inplace=True),
            nn.Linear(128, 10)
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), 64 * 7 * 7)
        x = self.classifier(x)
        return x
```

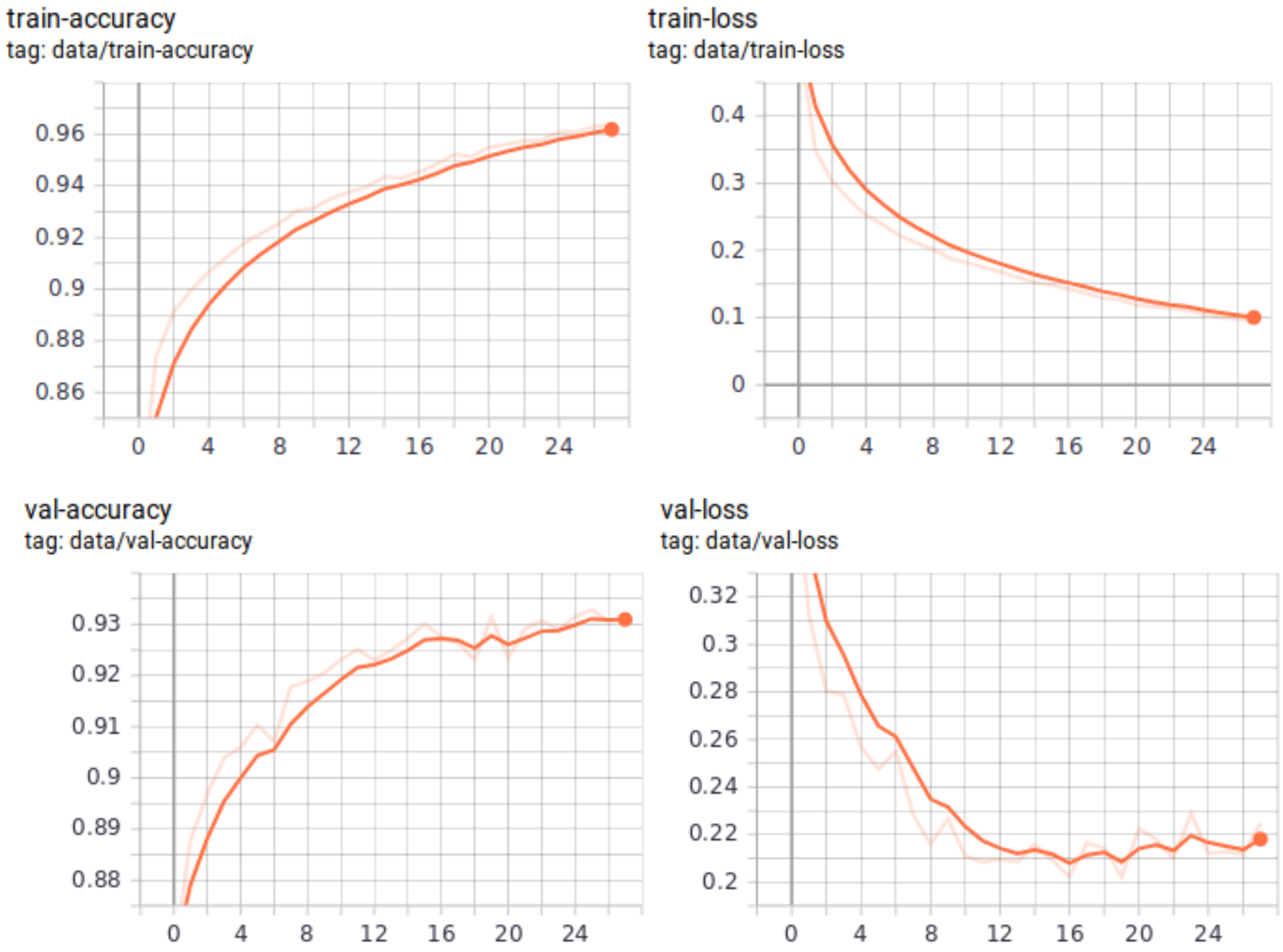
Typical training

```python
def run_model(net, loader, criterion, optimizer, train = True):
    running_loss = 0
    running_accuracy = 0

    # Set mode
    if train:
        net.train()
    else:
        net.eval()


    for i, (X, y) in enumerate(loader):
        # Pass to gpu or cpu
        X, y = X.to(device), y.to(device)

        # Zero the gradient
        optimizer.zero_grad()

        with torch.set_grad_enabled(train):
            output = net(X)
            _, pred = torch.max(output, 1)
            loss = criterion(output, y)

        # If on train backpropagate
        if train:
            loss.backward()
            optimizer.step()

        # Calculate stats
        running_loss += loss.item()
        running_accuracy += torch.sum(pred == y.detach())
    return running_loss / len(loader), running_accuracy.double() / len(loader.dataset)
```

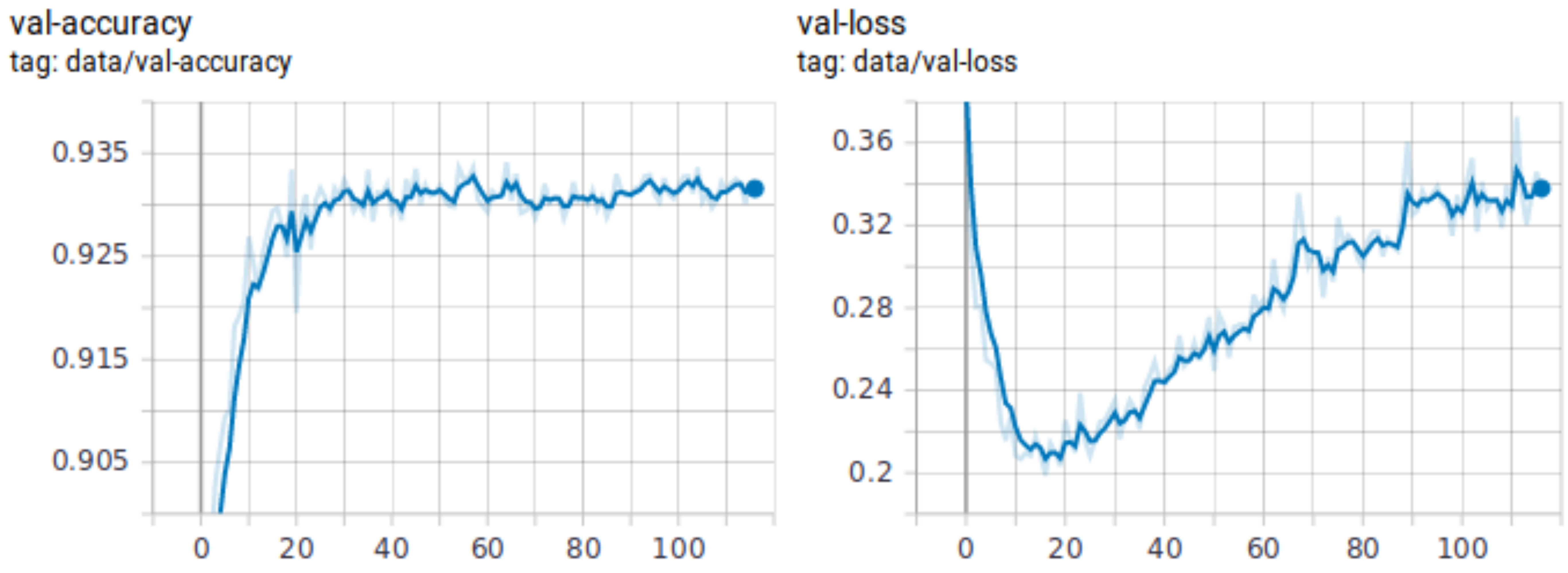
The main loop

```python
# Init network, criterion and early stopping
net = model.__dict__[args.model]().to(device)
criterion = torch.nn.CrossEntropyLoss()

# Define optimizer
optimizer = optim.Adam(net.parameters())

# Train the network
patience = args.patience
best_loss = 1e4
writeFile = open('{}/stats.csv'.format(current_dir), 'a')
writer = csv.writer(writeFile)
writer.writerow(['Epoch', 'Train Loss', 'Train Accuracy', 'Validation Loss', 'Validation Accuracy'])
for e in range(args.nepochs):
    start = time.time()
    train_loss, train_acc = run_model(net, train_loader,
                                      criterion, optimizer)
    val_loss, val_acc = run_model(net, val_loader,
                                      criterion, optimizer, False)
    end = time.time()

    # print stats
    stats = """Epoch: {}\t train loss: {:.3f}, train acc: {:.3f}\t
            val loss: {:.3f}, val acc: {:.3f}\t
            time: {:.1f}s""".format(e+1, train_loss, train_acc, val_loss,
                                        val_acc, end - start)
    print(stats)

    # Write to csv file
    writer.writerow([e+1, train_loss, train_acc.item(), val_loss, val_acc.item()])
    # early stopping and save best model
    if val_loss < best_loss:
        best_loss = val_loss
        patience = args.patience
        utils.save_model({
            'arch': args.model,
            'state_dict': net.state_dict()
        }, 'saved-models/{}-run-{}.pth.tar'.format(args.model, run))
    else:
        patience -= 1
        if patience == 0:
            print('Run out of patience!')
            writeFile.close()
            break
```

Some results



No early stopping



- Rosenblatt, F., 1958. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), p.386.
- Goodfellow, I., Bengio, Y. and Courville, A., 2016. Deep learning. MIT press.
- Montufar, G.F., Pascanu, R., Cho, K. and Bengio, Y., 2014. On the number of linear regions of deep neural networks. In Advances in neural information processing systems (pp. 2924-2932).
- Schawinski, K., Zhang, C., Zhang, H., Fowler, L. and Santhanam, G.K., 2017. Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit. Monthly Notices of the Royal Astronomical Society: Letters, 467(1), pp.L110-L114.
- He, K., Gkioxari, G., Dollár, P. and Girshick, R., 2017. Mask r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 2961-2969).
- Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).
- Zeiler, M.D. and Fergus, R., 2014, September. Visualizing and understanding convolutional networks. In European conference on computer vision (pp. 818-833). Springer, Cham.
- Xiao, H., Rasul, K. and Vollgraf, R., 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747.
- Belkin, Mikhail, et al. "Reconciling modern machine learning and the bias-variance trade-off." arXiv preprint arXiv: 1812.11118 (2018).
- Pascanu, Razvan, et al. "On the saddle point problem for non-convex optimization." arXiv preprint arXiv:1405.4604 (2014).
- Santurkar, Shibani, et al. "How does batch normalization help optimization?." Advances in Neural Information Processing Systems. 2018.