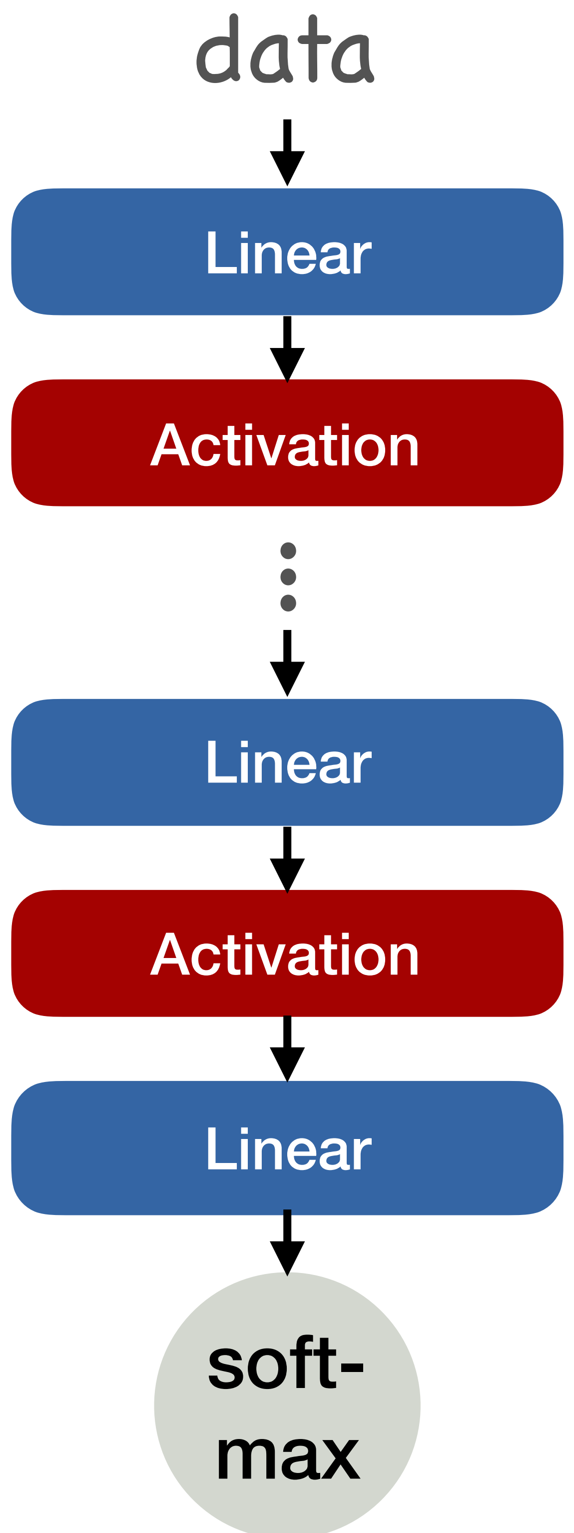
Beyond linear models

Linear models

- Limitations of linear models
- Linear to deep networks

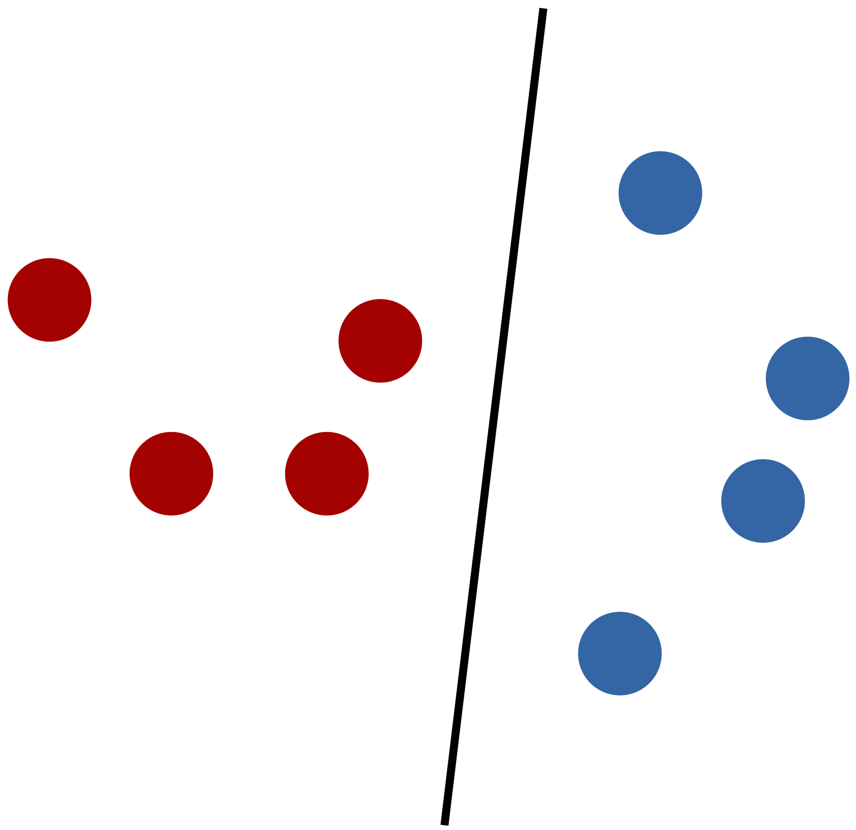
Deep networks

- Modular design
- Non-linearities and activation functions
- Outputs and losses
- Optimization

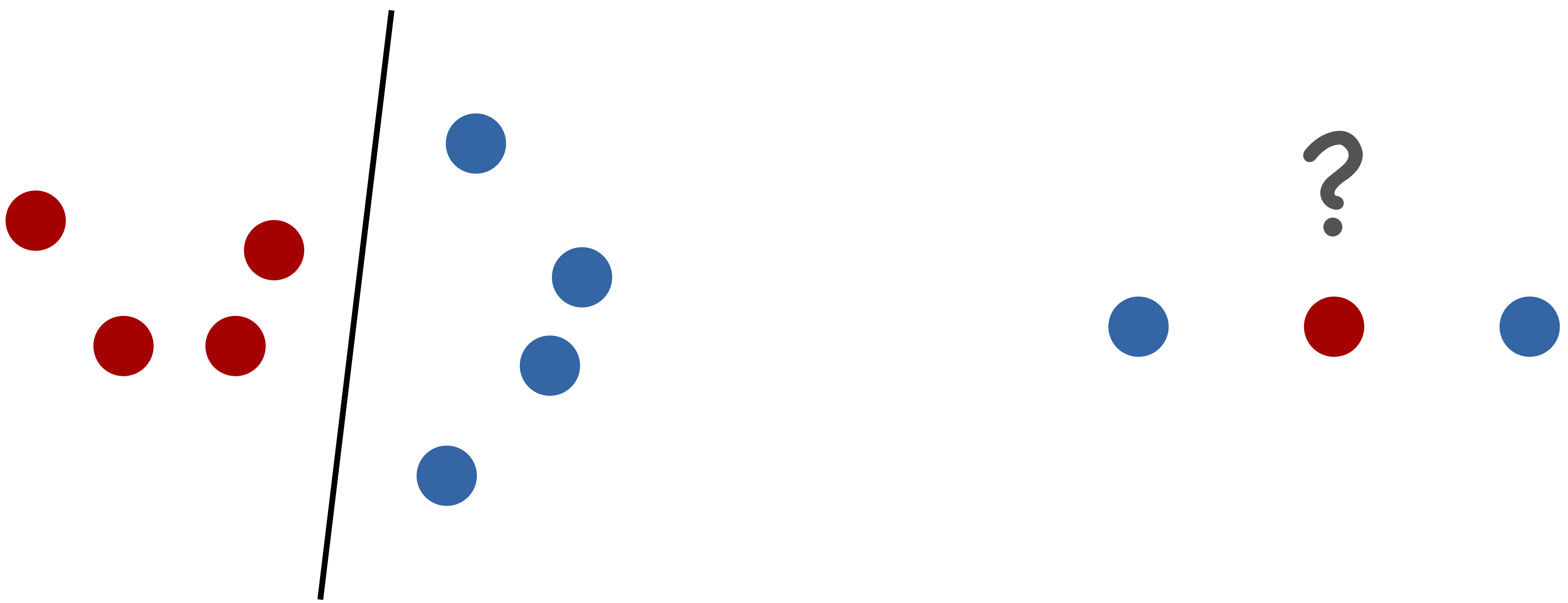


Limitations of linear models

Linear classifier

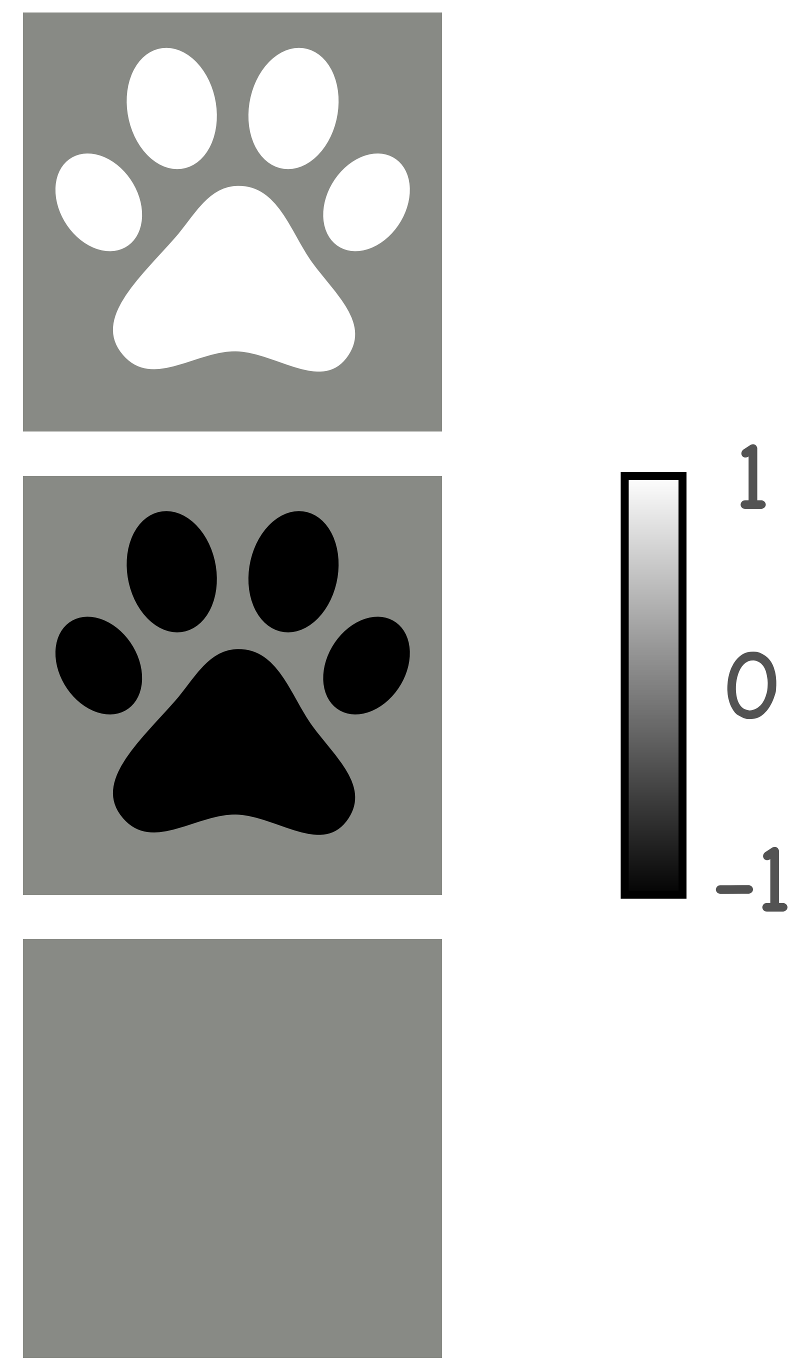


Linear classifier



A simple example

- Binary paw classification
  - Dog paw or not

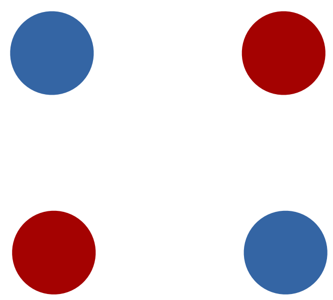


A simple example

Why does the linear model break?

- By linearity:
  - $\mathbf{w}^\top \mathbf{x}_1 + b > 0$
    $\mathbf{w}^\top \mathbf{x}_2 + b > 0$
  - Then $\mathbf{w}^\top \mathbf{x} + b > 0$
    for any $\mathbf{x} = \alpha \mathbf{x}_1 + (1 - \alpha) \mathbf{x}_2$

- Cannot learn xor

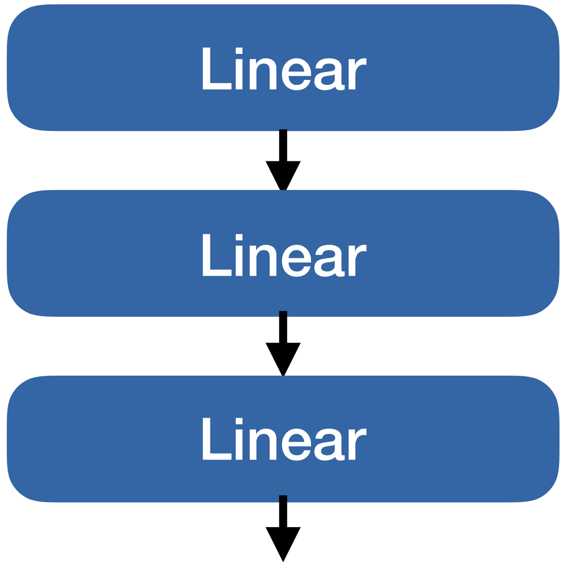


Does adding more linear layers help?

- No

- Combination of linear layers still linear

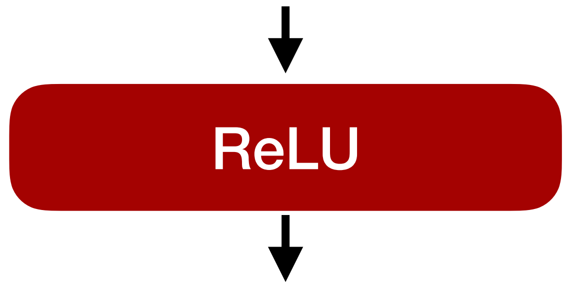
- $\mathbf{W}_2(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$
$= (\mathbf{W}_2\mathbf{W}_1)\mathbf{x} + (\mathbf{W}_2\mathbf{b}_1 + \mathbf{b}_2)$



Non-linearities

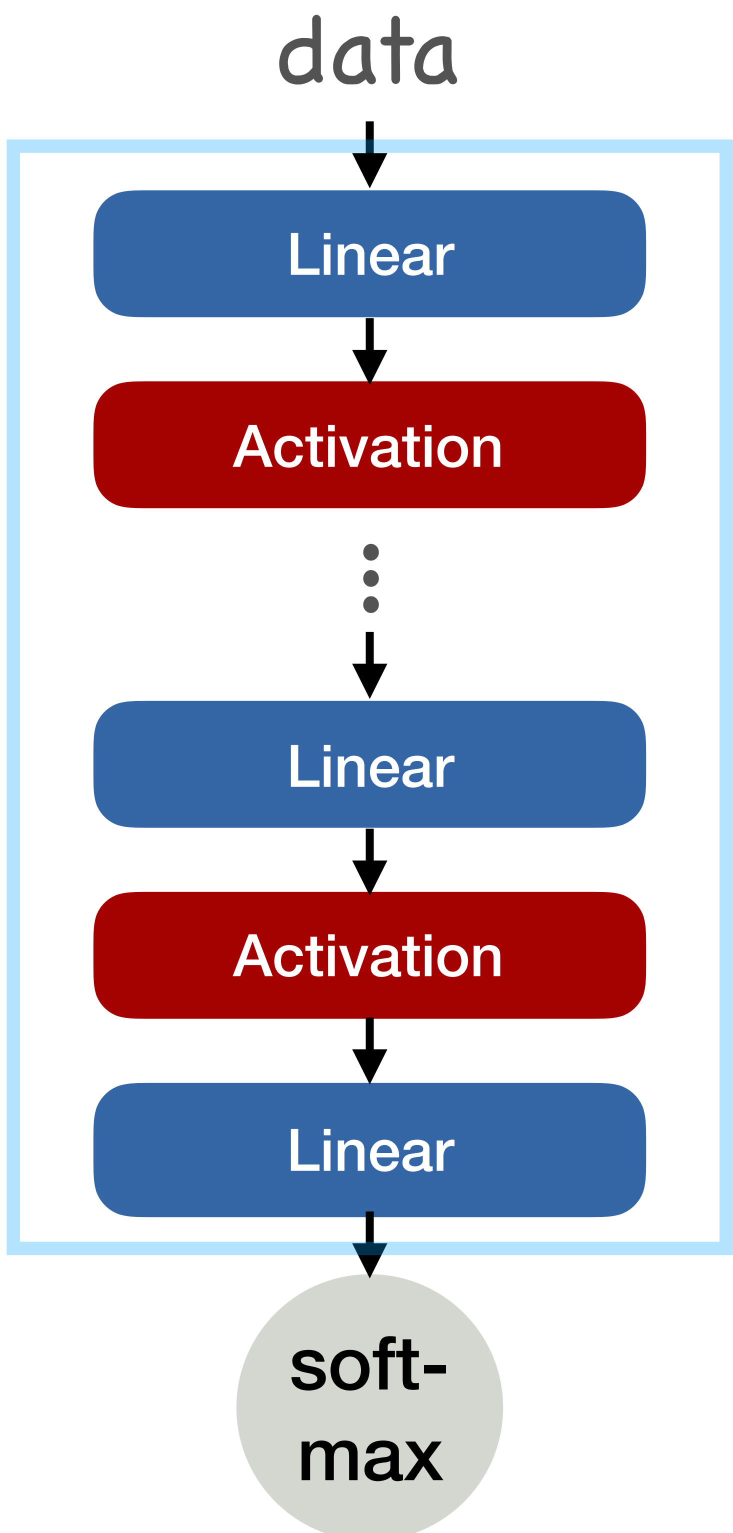
Non-linearities

- Rectified Linear Unit
  - $\mathrm{ReLU}(x) = \max(x, 0)$
- Non-linear and differentiable almost everywhere



Deep networks

- Alternates linear and non-linear layers

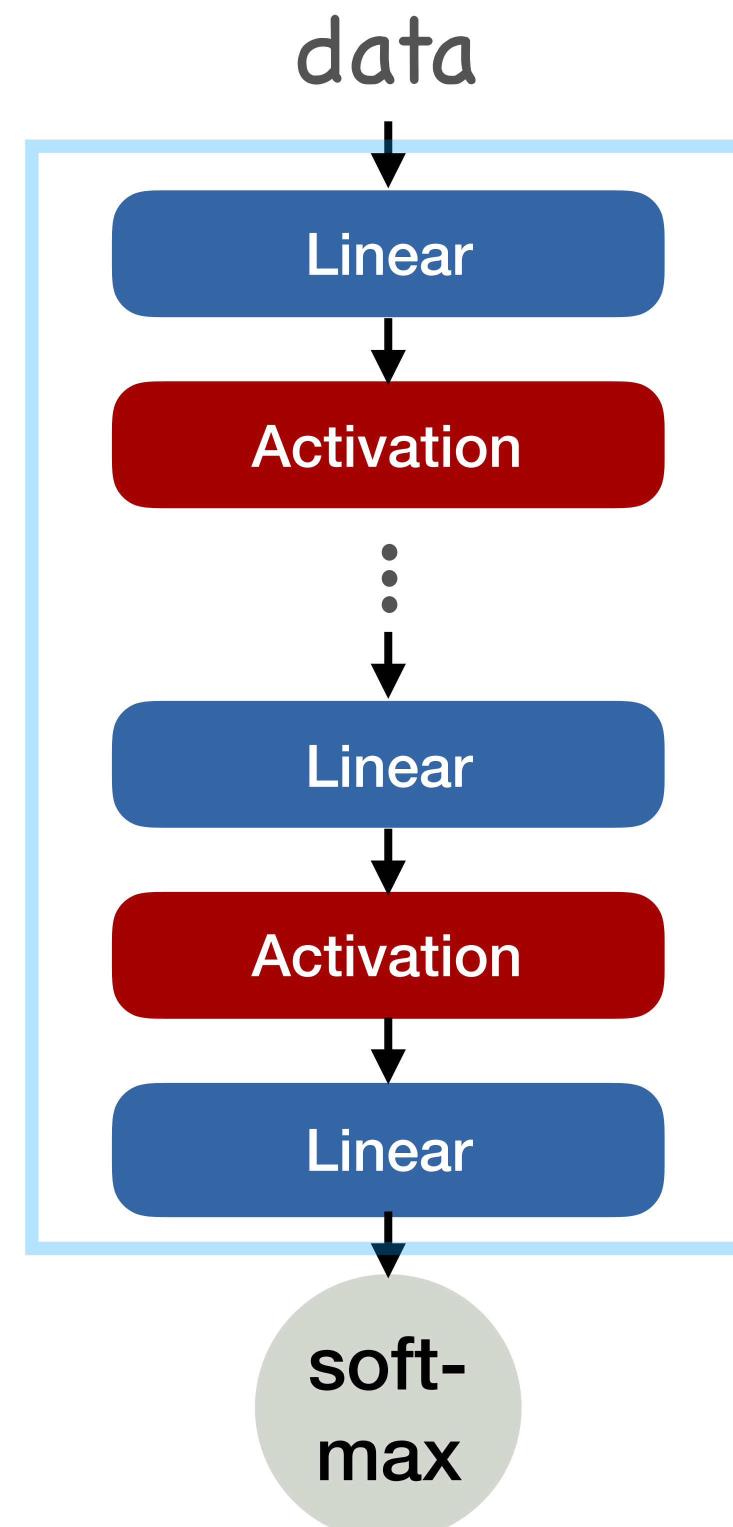


A simple example

- “Shallow” network
- Dog paw or not?

Deep networks

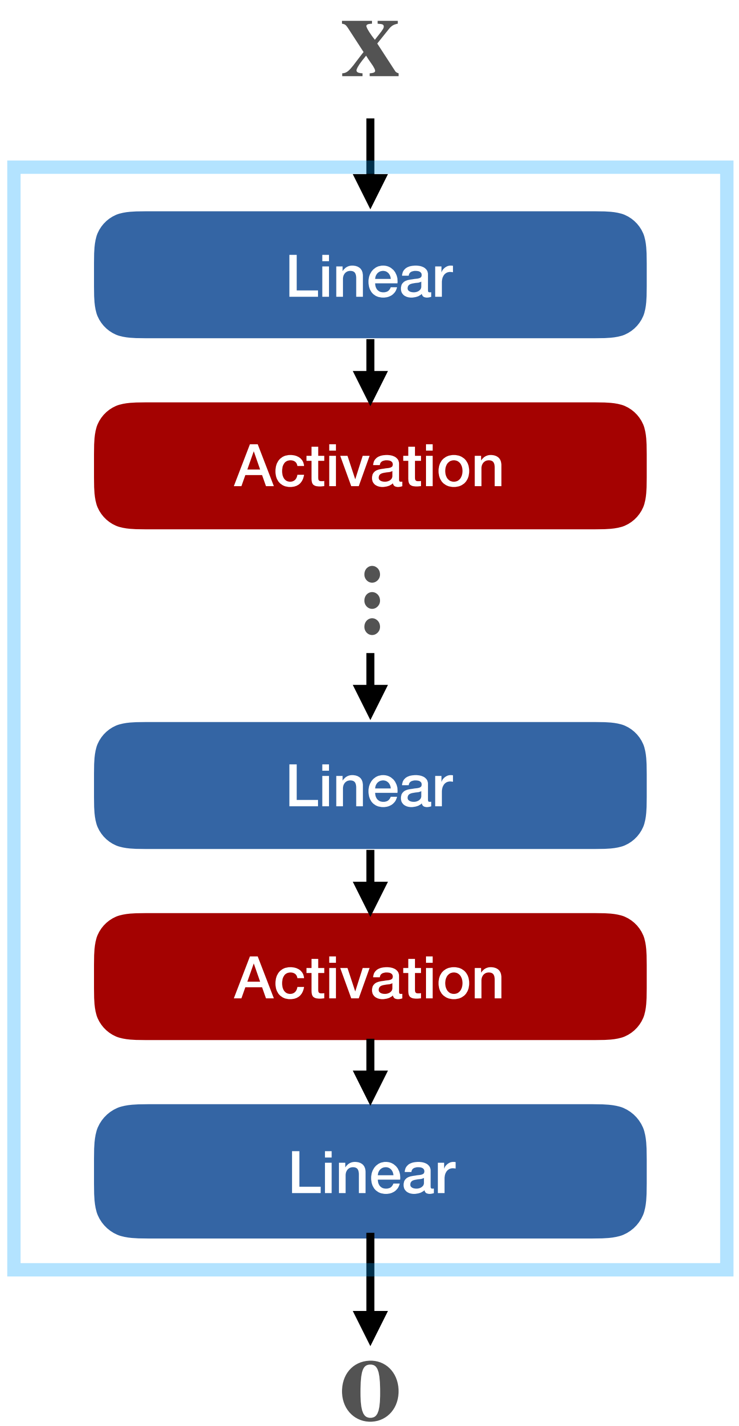
- Class of continuous functions $f_\theta : \mathbf{x} \to o$
  - Parameters θ
- Can approximate any continuous function



Output representations

Inputs and outputs of networks

- Input:
    - Tensor $\mathbf{x}$
- Output:
    - Tensor $\mathbf{o}$



Regression

- vanilla tensor $\hat{\mathbf{y}} = \mathbf{o}$

Positive regression

- Option 1: ReLU
  - $\hat{\mathbf{y}} = \max(\mathbf{o}, 0)$
- Option 2: Soft ReLU
  - $\hat{\mathbf{y}} = \log(1 + e^{\mathbf{o}})$

Binary Classification

- Option 1: Thresholding
  - $\hat{\mathbf{y}} = \mathbf{o} > 0$
- Option 2: Logistic Regression
  - $p(1) = \sigma(\mathbf{o})$

General Classification

- Output more values, one per class
- Option 1: argmax
  - $\hat{y} = \operatorname{argmax}_i \mathbf{o}_i$
- Option 2: softmax
  - $p(y) = \operatorname{softmax}(\mathbf{o})_y$

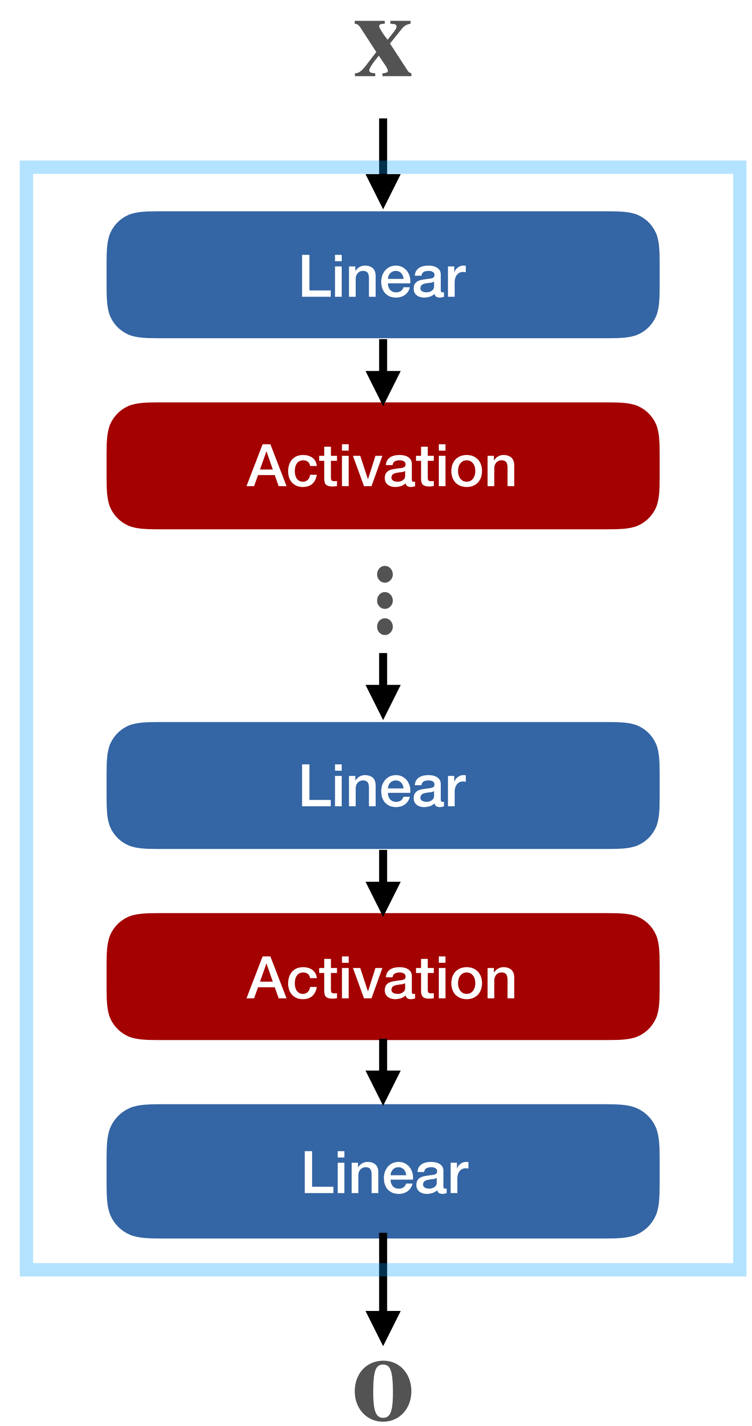
Output representations in practice

- Do not add into model
- Always output raw values

Loss functions

Inputs and outputs of networks

- Input:
  - Tensor $\mathbf{x}$
- Output:
  - Tensor $\mathbf{o}$



Regression

- L1 loss
  - $\ell = |\mathbf{y} - \mathbf{o}|$
- L2 loss
  - $\ell = \|\mathbf{y} - \mathbf{o}\|^2$

Classification

- Compute likelihood
  - $p(1) = \sigma(o)$
  - $\mathbf{p} = \mathrm{softmax}(\mathbf{o})$
- Cross entropy / -Log likelihood
  - $-\log(p(y))$

Classification losses in practice

- $\sigma(o) = 0$ for $o \to -50$
  - $\log(\sigma(o)) = \log(0)$ is undefined
- Combine $\log$ and σ
  - `BCEWithLogitsLoss`
  - `CrossEntropyLoss`

05

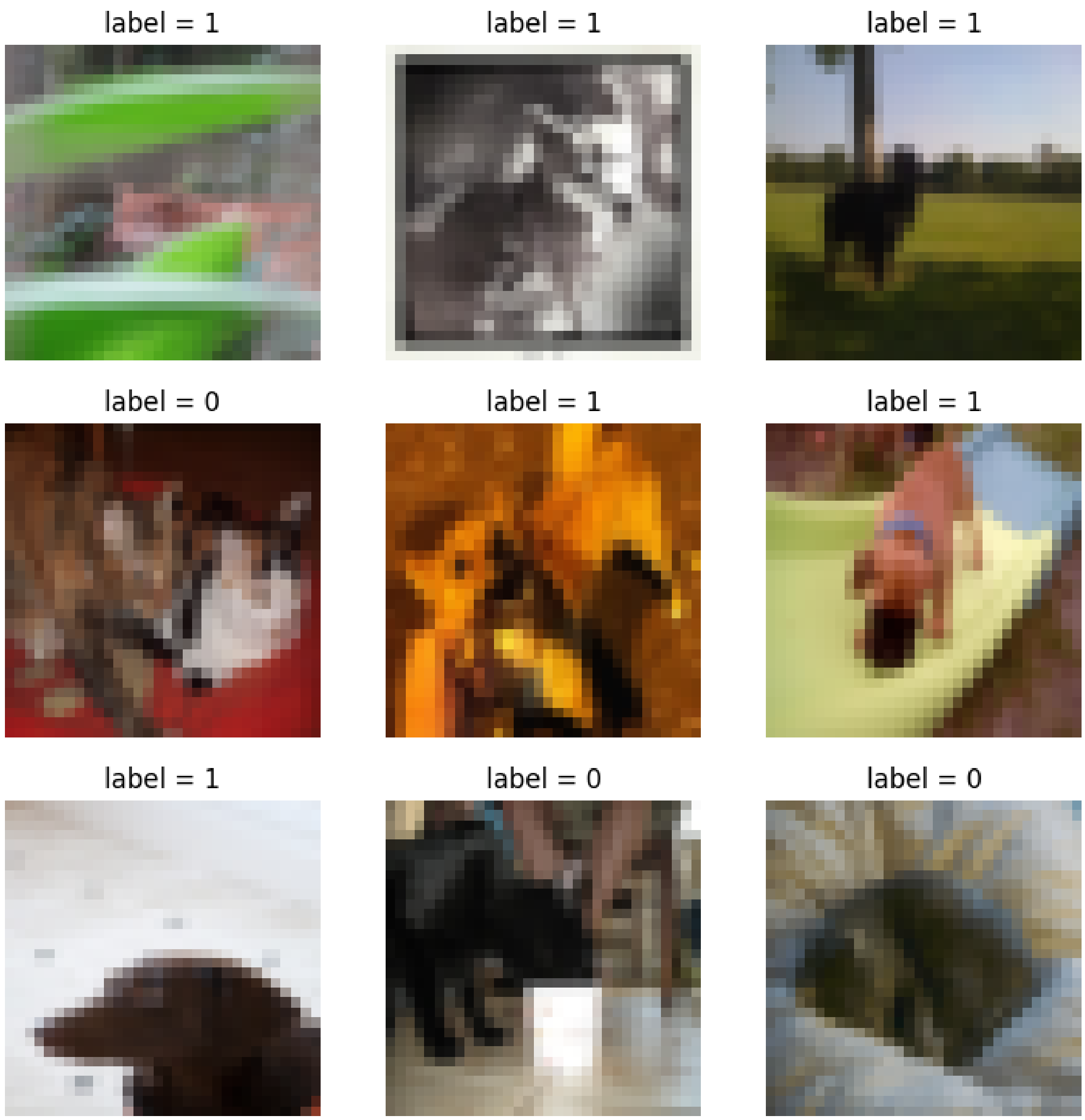
January 23, 2024

[1]:
```
%pylab inline
import torch
# Making sure we can find the data loader
import sys
sys.path.append('..')
sys.path.append('../..')
from data import load
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
# Let's load the dataset
train_data, train_label = load.get_dogs_and_cats_data(resize=(32,32),␣
↪n_images=100)
input_size = 32*32*3
to_image = load.to_image_transform()
```

[3]:
```
figure(figsize=(9,9))
# Plot the first 9 images (all cats)
for i, (data, label) in enumerate(zip(train_data[:9],train_label[:9])):
    subplot(3,3,1+i)
    imshow(to_image(data))
    title('label = %d'%label)
    axis('off')
```



[4]:
```
class Network1(torch.nn.Module):
    def __init__(self, n_hidden=100):
        super().__init__()
        self.linear1 = torch.nn.Linear(input_size, n_hidden)
        self.activation = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(n_hidden, 1)

    def forward(self, x):
        return self.linear2(self.activation(self.linear1(x.view(x.size(0),
 -1))))
```

[5]:
```
# Create the network
net1 = Network1(100)
# Run an image through it
print( net1(train_data).view(-1).detach().numpy() )
```

```
[-0.11666452 -0.01218061 -0.02647986 -0.05582439 -0.06884713 -0.07426661
 -0.11988138 -0.1508754  -0.09473266 -0.10555119 -0.11205442 -0.07451358
 -0.0114538  -0.03645098 -0.06230658 -0.05653633 -0.20482528 -0.10789528
 -0.09619182 -0.07970785 -0.12600176 -0.05568714  0.00681441 -0.14392285
 -0.12056908 -0.12222722 -0.10236578 -0.24464132 -0.04859121 -0.20569515
 -0.09635501 -0.09910933 -0.19064035 -0.03423431 -0.08156552 -0.15408464
 -0.02703031 -0.14437844 -0.12504634 -0.1360234  -0.06810249 -0.0230343
 -0.20682594 -0.05603386 -0.14723194 -0.14903176 -0.04260117 -0.06173945
 -0.0771127  -0.07101672 -0.13693443 -0.16763532 -0.12506317 -0.0840079
 -0.08369774 -0.00672981  0.00092466 -0.07512563  0.00588925 -0.10326148
  0.01207101 -0.14386982 -0.09098267 -0.18191543 -0.12514006  0.01794229
 -0.17029686 -0.19610392 -0.11749545 -0.10202523 -0.11068309 -0.0528039
 -0.06975032 -0.06040952 -0.03295252 -0.06511676 -0.13189809 -0.06583469
 -0.04055259 -0.0561346  -0.02101819 -0.03989616 -0.05980999 -0.17139488
 -0.02495858 -0.12706935 -0.20838268 -0.06545367 -0.03899799 -0.07631201
 -0.11093491 -0.02930016 -0.20559758 -0.05399808 -0.06877933 -0.12470949
 -0.08101448 -0.09563463 -0.08646178 -0.12490054]
```

[6]:
```
class Network2(torch.nn.Module):
    def __init__(self, *hidden_size):
        super().__init__()
        layers = []
        # Add the hidden layers
        n_in = input_size
        for n_out in hidden_size:
            layers.append(torch.nn.Linear(n_in, n_out))
            layers.append(torch.nn.ReLU())
            n_in = n_out

        # Add the classifier
        layers.append(torch.nn.Linear(n_out, 1))
        self.network = torch.nn.Sequential(*layers)

    def forward(self, x):
        return self.network(x.view(x.size(0), -1))
```

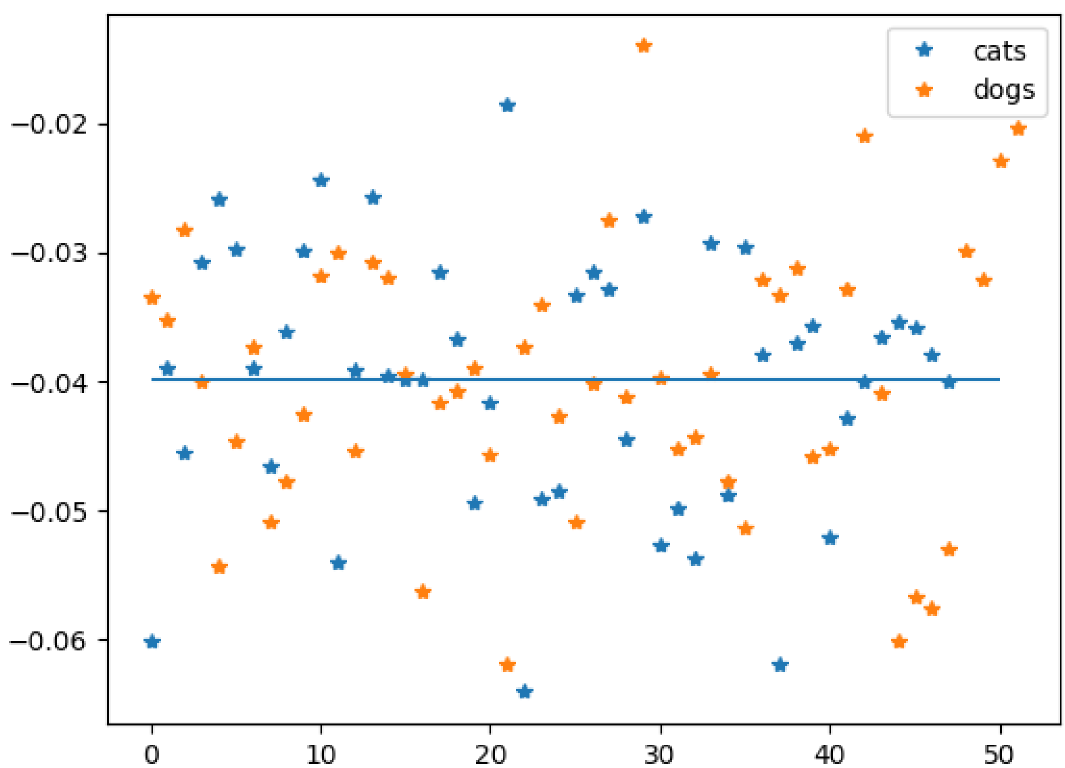
[7]:
```
# Create the network
net2 = Network2(100, 50, 50)
# Run an image through it
print( net2(train_data).view(-1).detach().numpy() )
```

```
[-0.03342889 -0.0351971  -0.02820556 -0.06019168 -0.04007806 -0.05432146
 -0.04470311 -0.03901586 -0.04551048 -0.03731422 -0.05093319 -0.03082472
```

```
-0.0259193  -0.04776499 -0.04250351 -0.029782   -0.03178665 -0.03900372
-0.04654014 -0.03617655 -0.03003443 -0.04539562 -0.02983748 -0.03075305
-0.03199217 -0.0243466  -0.054005   -0.03918456 -0.03946443 -0.05627764
-0.04166208 -0.02576187 -0.03961597 -0.04083673 -0.03986252 -0.03892076
-0.0456367  -0.03987778 -0.06198189 -0.03728481 -0.03155395 -0.036805
-0.03400616 -0.04274697 -0.04948325 -0.05085205 -0.04161504 -0.01854287
-0.04013046 -0.06409714 -0.04915904 -0.02758344 -0.04848149 -0.03329031
-0.04119885 -0.03161262 -0.03294888 -0.04454168 -0.02716818 -0.05270016
-0.01403332 -0.04991054 -0.03975805 -0.05376447 -0.04529271 -0.0293345
-0.04886403 -0.04442658 -0.0296757  -0.03936881 -0.04778569 -0.05143041
-0.03218606 -0.03332945 -0.0312345  -0.03794158 -0.04582901 -0.06189429
-0.04525457 -0.03291155 -0.03706281 -0.02100243 -0.03566442 -0.05207755
-0.04284387 -0.0400481  -0.03666504 -0.03544376 -0.04095592 -0.03590231
-0.06017258 -0.03795452 -0.05677029 -0.05759883 -0.05293075 -0.03998635
-0.02990015 -0.03219899 -0.02292    -0.02041762]
```

[8]:
```
plot( net2(train_data[train_label==0]).view(-1).detach().numpy(), '*',␣
 ↪label='cats')
plot( net2(train_data[train_label==1]).view(-1).detach().numpy(), '*',␣
 ↪label='dogs')
hlines(net2(train_data).detach().numpy().mean(), 0, 50)
legend()
```

[8]: <matplotlib.legend.Legend at 0x7fc99558c8b0>



5

[ ]:

```
```

Optimization of deep networks

Data

- Input: $\{\mathbf{x}_0, \ldots, \mathbf{x}_{N-1}\}$
- Label: $\{\mathbf{y}_0, \ldots, \mathbf{y}_{N-1}\}$
- Dataset: $D = \{(\mathbf{x}_0, \mathbf{y}_0), \ldots, (\mathbf{x}_{N-1}, \mathbf{y}_{N-1})\}$

( , dog)

( , dog)

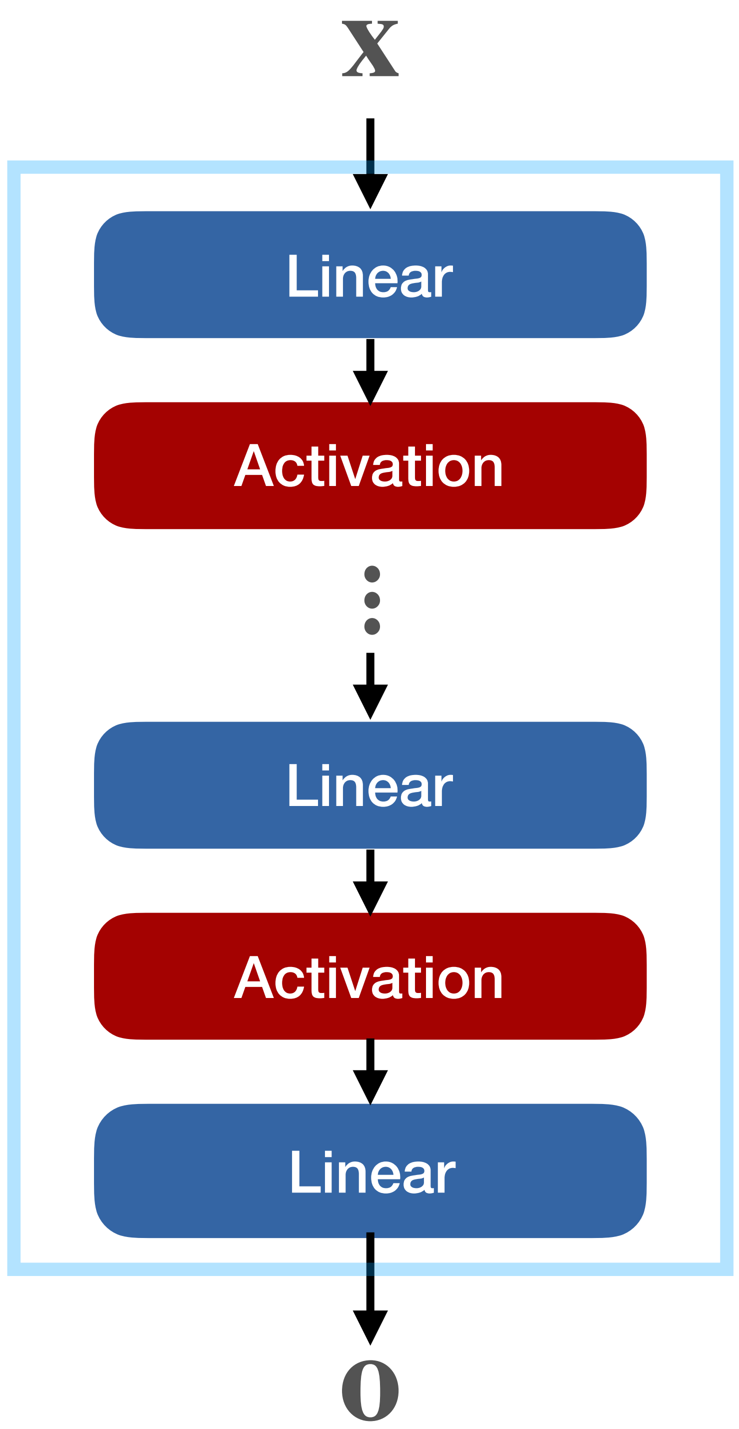
( , cat)

( , cat)

$\vdots$

Model

- Deep network $f : (\mathbf{x}, \theta) \rightarrow \mathbf{o}$
  - Layers of computation
  - Parameters θ
  - Differentiable computation graph

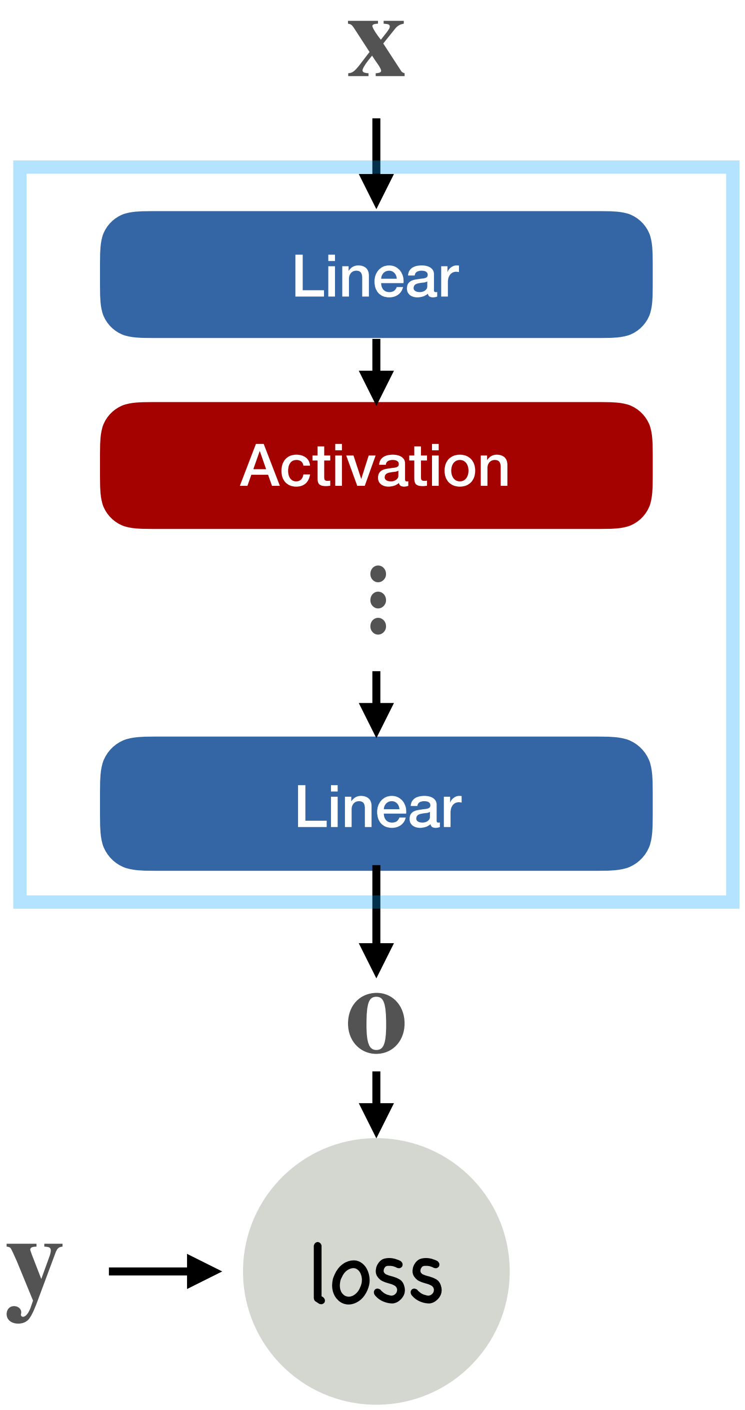


Loss

- Differentiable $\ell(\mathbf{o}, \mathbf{y})$
- Regression
  - Distance norm

    $$\ell(\mathbf{o}, \mathbf{y}) = \|\mathbf{o} - \mathbf{y}\|$$
- Classification
  - Cross Entropy

    $$\ell(\mathbf{o}, y) = -\log p(y)$$
- Over training dataset
  - $L(\theta) = \mathbb{E}_{\mathbf{x},\mathbf{y}\sim D}[\ell(f(\mathbf{x}, \theta), \mathbf{y})]$

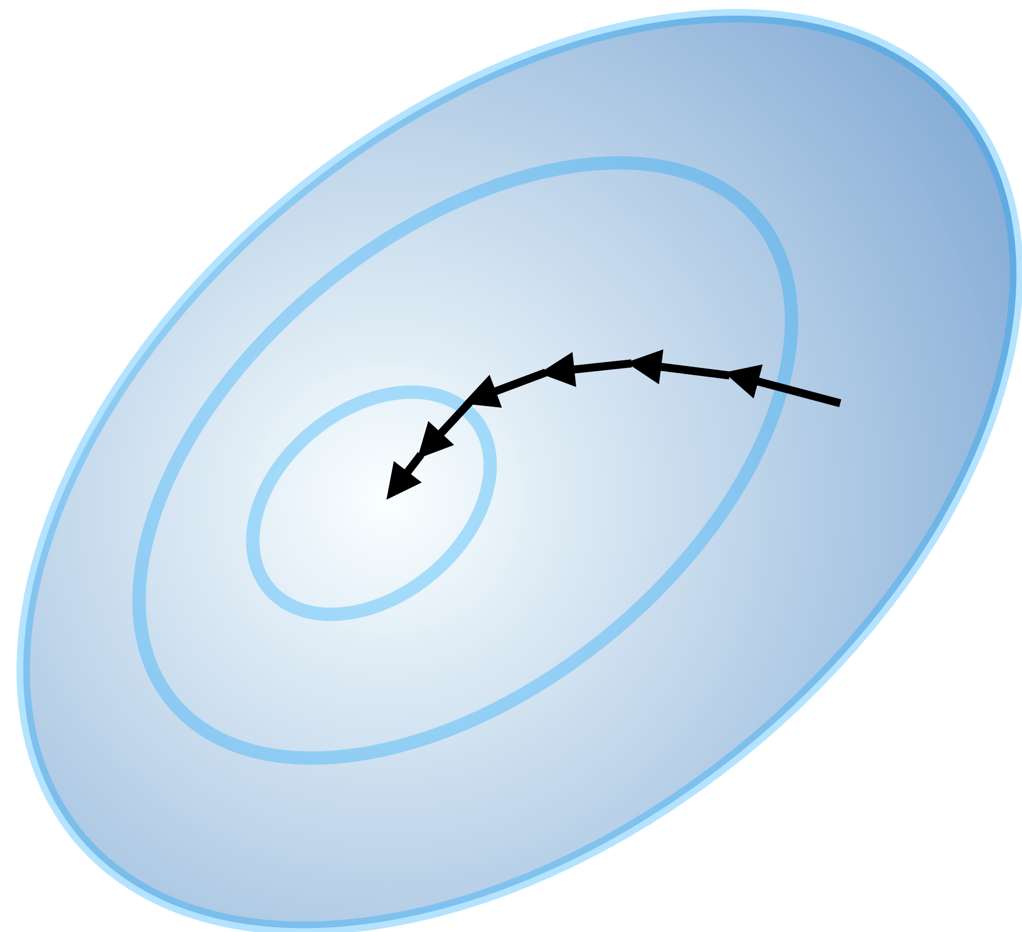


Optimization

- Minimize $L(\theta)$

Gradient Descent

- Repeat until convergence:
  - $\theta := \theta - \epsilon \frac{dL(\theta)}{d\theta}$



Issue with Gradient Descent

- Slow to compute gradient
  - $$\frac{dL(\theta)}{d\theta} = \mathbb{E}_{\mathbf{x},\mathbf{y}\in D}[\frac{d\ell(f(\mathbf{x},\theta),\mathbf{y})}{d\theta}]$$

Stochastic Gradient Descent

Gradient Descent

- Repeat until convergence:
  - $\theta := \theta - \epsilon \frac{dL(\theta)}{d\theta}$

Gradient Descent

- Repeat until convergence:
  - $\theta_0 := \theta$
  - for $\mathbf{x}, \mathbf{y} \sim D$:
    - $\theta := \theta - \epsilon \dfrac{d\ell(f(\mathbf{x}, \theta_0), \mathbf{y})}{d\theta_0}$

Stochastic Gradient Descent

- Repeat until convergence:
  - for $\mathbf{x}, \mathbf{y} \sim D$:
    - $\theta := \theta - \epsilon \dfrac{d\ell(f(\mathbf{x}, \theta), \mathbf{y})}{d\theta}$

Terminology

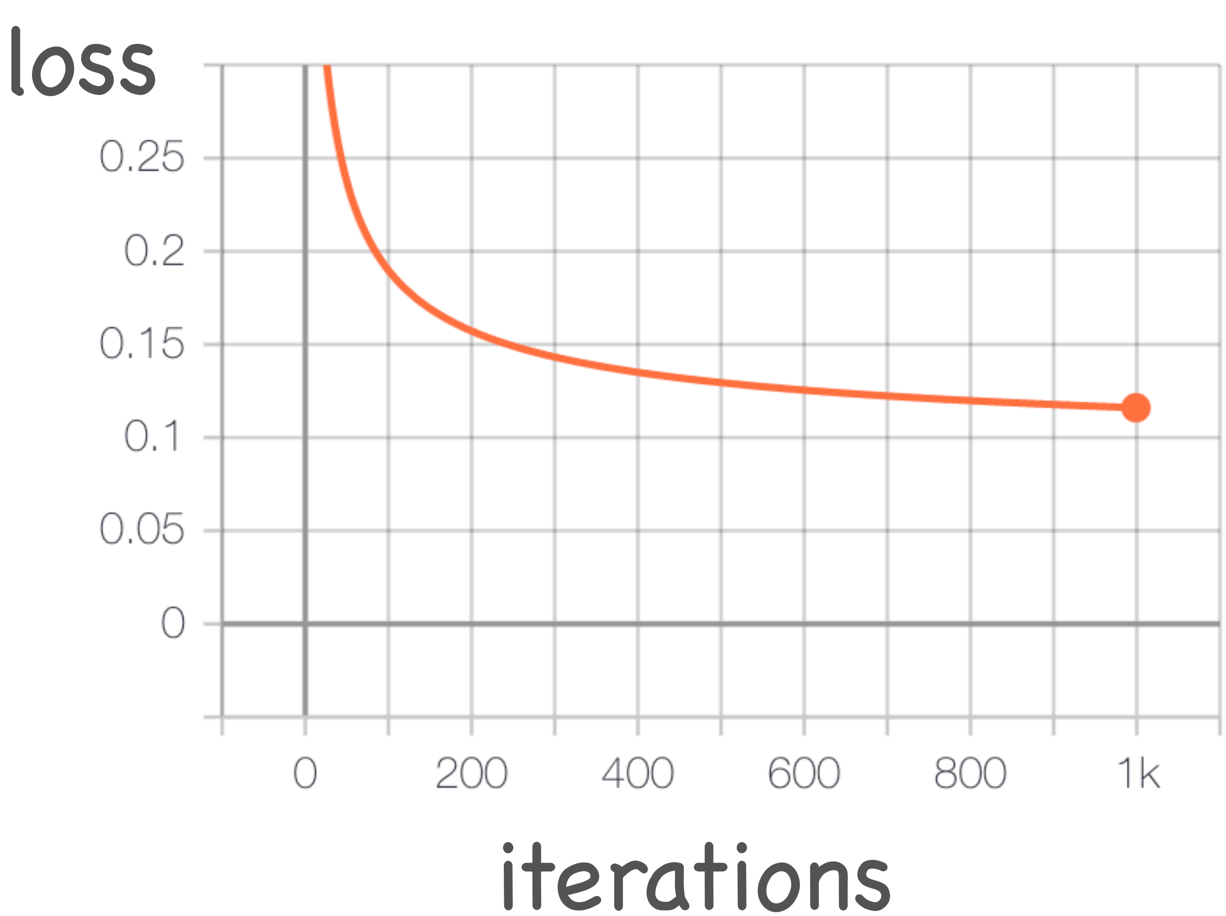
- Repeat until convergence:
    - for $\mathbf{x}, \mathbf{y} \sim D$:
        - $$\theta := \theta - \epsilon \frac{d\ell(f(\mathbf{x}, \theta), \mathbf{y})}{d\theta}$$

Practical SGD

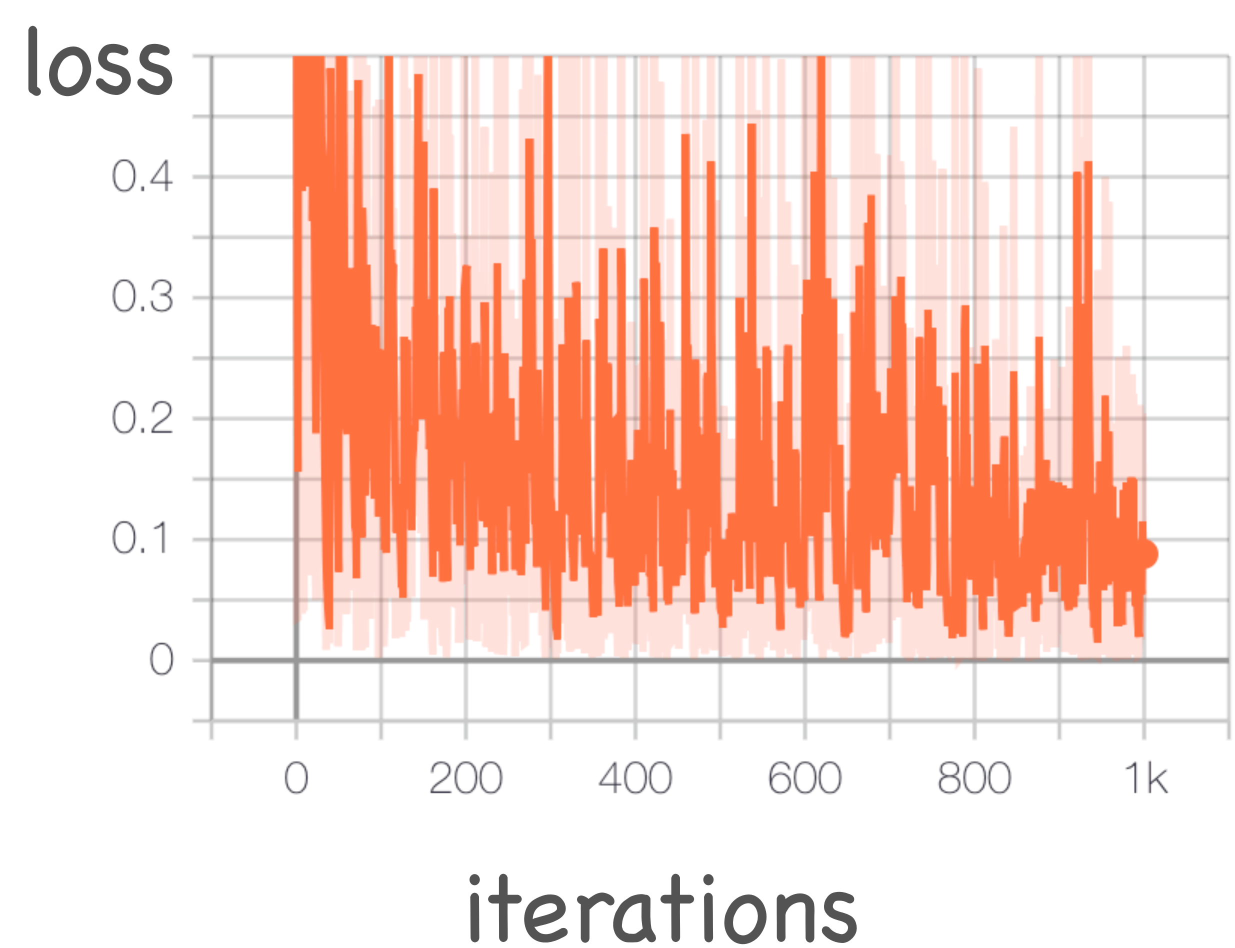
- For n epochs:
  - for $\mathbf{x}, \mathbf{y} \sim D$:
    - $\theta := \theta - \epsilon \frac{d\ell(f(\mathbf{x}, \theta), \mathbf{y})}{d\theta}$

Learning curves

GD



SGD



The Variance of SGD

$$\frac{d\ell(f(\mathbf{x},\theta),\mathbf{y})}{d\theta} \neq \frac{dL(\theta)}{d\theta}$$

- Variance
  - $$\mathbb{E}_{\mathbf{x},\mathbf{y}\sim D}\left[\left(\frac{d\ell(f(\mathbf{x},\theta),\mathbf{y})}{d\theta} - \frac{dL(\theta)}{d\theta}\right)^2\right]$$

$$= \mathbb{E}_{\mathbf{x},\mathbf{y}\sim D}\left[\left(\frac{d\ell(f(\mathbf{x},\theta),\mathbf{y})}{d\theta}\right)^2\right] - \left(\frac{dL(\theta)}{d\theta}\right)^2$$

Mini-batches

Stochastic Gradient Descent

- For n epochs:
  - for $\mathbf{x}, \mathbf{y} \sim D$:
    - $\theta := \theta - \epsilon \frac{d\ell(f(\mathbf{x}, \theta), \mathbf{y})}{d\theta}$

Stochastic Gradient Descent

- For n epochs:
    - for i in $0, \ldots, |D| - 1$
        - $\mathbf{x}, \mathbf{y} := D_i$
        - $\theta := \theta - \epsilon \frac{d\ell(f(\mathbf{x}, \theta), \mathbf{y})}{d\theta}$

Mini-batches

- For n epochs:
  - Split dataset D into m mini-batches $B_0, \ldots B_{m-1}$ of size BS
  - for each batch B_i
    - $\theta := \theta - \epsilon \mathbb{E}_{\mathbf{x},\mathbf{y} \sim B_i} \left[\dfrac{d\ell(f(\mathbf{x}, \theta), \mathbf{y})}{d\theta} \right]$

Variance of mini-batches

- Variance of SGD
  - $$\mathbb{E}_{\mathbf{x},\mathbf{y}\sim D}\left[\left(\frac{d\ell(f(\mathbf{x},\theta),\mathbf{y})}{d\theta}\right)^2\right]-\left(\frac{dL(\theta)}{d\theta}\right)^2$$
- Variance of SGD with mini-batches
  - $$\mathbb{E}_{B_i}\left[\left(\mathbb{E}_{\mathbf{x},\mathbf{y}\sim B_i}\left[\frac{d\ell(f(\mathbf{x},\theta),\mathbf{y})}{d\theta}\right]\right)^2\right]-\left(\frac{dL(\theta)}{d\theta}\right)^2$$

Always use mini-batches

Variance of mini-batches

Jensen's inequality

$$\left(\mathbb{E}_{\mathbf{x},\mathbf{y}\sim B_i}\left[\frac{d\ell\left(f(\mathbf{x},\theta),\mathbf{y}\right)}{d\theta}\right]\right)^2 \leq \mathbb{E}_{\mathbf{x},\mathbf{y}\sim B_i}\left[\left(\frac{d\ell\left(f(\mathbf{x},\theta),\mathbf{y}\right)}{d\theta}\right)^2\right]$$

Momentum

Stochastic Gradient Descent

- For n epochs:
  - for $\mathbf{x}, \mathbf{y} \sim D$:
    - $\theta := \theta - \epsilon \frac{d\ell(f(\mathbf{x}, \theta), \mathbf{y})}{d\theta}$

Momentum

- $\mathbf{v} := 0$
- For n epochs:
  - for $\mathbf{x}, \mathbf{y} \sim D$
    - $\mathbf{v} := \rho\mathbf{v} + \dfrac{d\ell(f(\mathbf{x}, \theta), \mathbf{y})}{d\theta}$
    - $\theta := \theta - \epsilon\mathbf{v}$

Variance reduction of Momentum

- Variance of SGD

$$\mathbb{E}_{\mathbf{x},\mathbf{y}\sim D}\left[\left(\frac{d\ell(f(\mathbf{x},\theta),\mathbf{y})}{d\theta}\right)^2\right] - \left(\frac{dL(\theta)}{d\theta}\right)^2$$

Momentum

10

January 23, 2024

[1]:

```
%pylab inline
import torch
# Making sure we can find the data loader
import sys
sys.path.append('..')
sys.path.append('../..')
from data import load
device = torch.device('cuda') if torch.cuda.is_available() else torch.
 ↪device('cpu')
print('device = ', device)
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
device =  cuda
```

[2]:

```
# Let's load the dataset
train_data, train_label = load.get_dogs_and_cats_data(resize=(32,32))
valid_data, valid_label = load.get_dogs_and_cats_data(split='valid',␣
 ↪resize=(32,32))
input_size = 32*32*3
to_image = load.to_image_transform()

train_data, train_label = train_data.to(device), train_label.to(device)
valid_data, valid_label = valid_data.to(device), valid_label.to(device)
```

[3]:

```
class Network2(torch.nn.Module):
    def __init__(self, *hidden_size):
        super().__init__()
        layers = []
        # Add the hidden layers
        n_in = input_size
        for n_out in hidden_size:
            layers.append(torch.nn.Linear(n_in, n_out))
            layers.append(torch.nn.ReLU())
            n_in = n_out

        # Add the classifier
```

```
        layers.append(torch.nn.Linear(n_out, 1))
        self.network = torch.nn.Sequential(*layers)

    def forward(self, x):
        return self.network(x.view(x.size(0), -1)).view(-1)
```

[4]:

```
%load_ext tensorboard
import tempfile
log_dir = tempfile.mkdtemp()
%tensorboard --logdir {log_dir} --reload_interval 1 --bind_all
```

```
<IPython.core.display.HTML object>
```

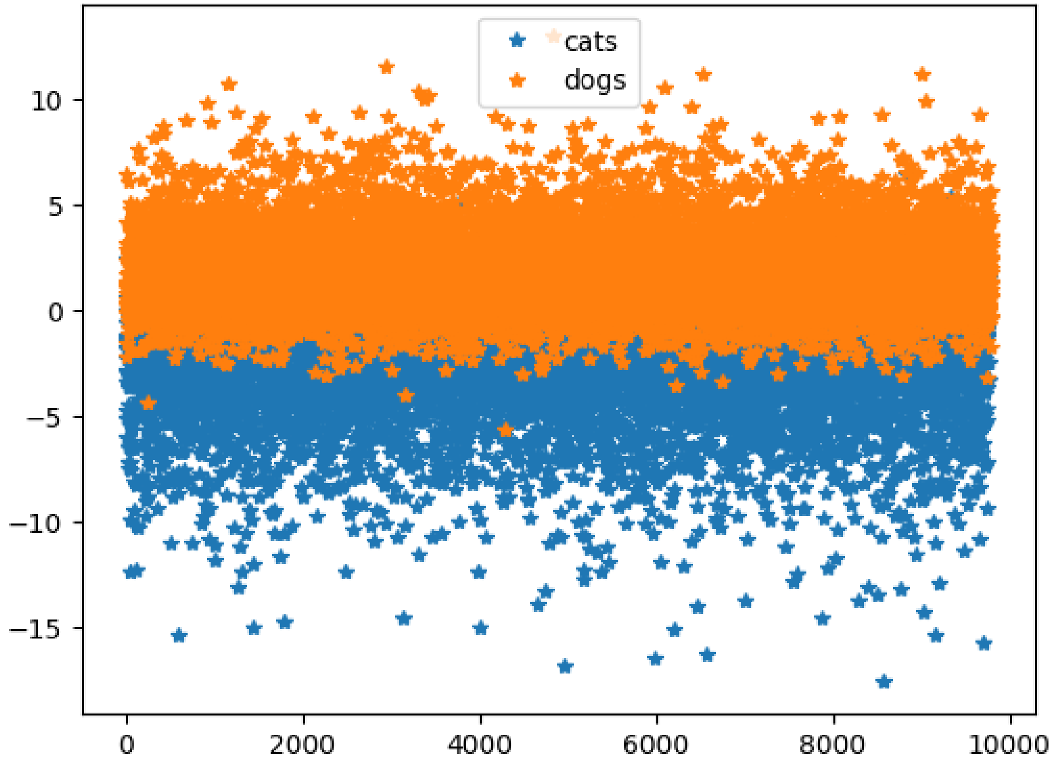
[5]:

```
import torch.utils.tensorboard as tb
n_epochs = 100
batch_size = 128

train_logger = tb.SummaryWriter(log_dir+'/deepnet1/train', flush_secs=1)
valid_logger = tb.SummaryWriter(log_dir+'/deepnet1/valid', flush_secs=1)

# Create the network
net2 = Network2(100,50,50).to(device)

# Create the optimizer
optimizer = torch.optim.SGD(net2.parameters(), lr=0.01, momentum=0.9,␣
 ↪weight_decay=1e-4)

# Create the loss
loss = torch.nn.BCEWithLogitsLoss()

# Start training
global_step = 0
for epoch in range(n_epochs):
    # Shuffle the data
    permutation = torch.randperm(train_data.size(0))

    # Iterate
    train_accuracy = []
    for it in range(0, len(permutation)-batch_size+1, batch_size):
        batch_samples = permutation[it:it+batch_size]
        batch_data, batch_label = train_data[batch_samples],␣
 ↪train_label[batch_samples]

        # Compute the loss
        o = net2(batch_data)
        loss_val = loss(o, batch_label.float())
```

```
        train_logger.add_scalar('train/loss', loss_val, global_step=global_step)
        # Compute the accuracy
        train_accuracy.extend(((o > 0).long() == batch_label).cpu().detach().
↪numpy())

        optimizer.zero_grad()
        loss_val.backward()
        optimizer.step()

        # Increase the global step
        global_step += 1

    # Evaluate the model
    valid_pred = net2(valid_data) > 0
    valid_accuracy = float((valid_pred.long() == valid_label).float().mean())

    train_logger.add_scalar('train/accuracy', np.mean(train_accuracy),␣
↪global_step=global_step)
    valid_logger.add_scalar('valid/accuracy', valid_accuracy,␣
↪global_step=global_step)
```

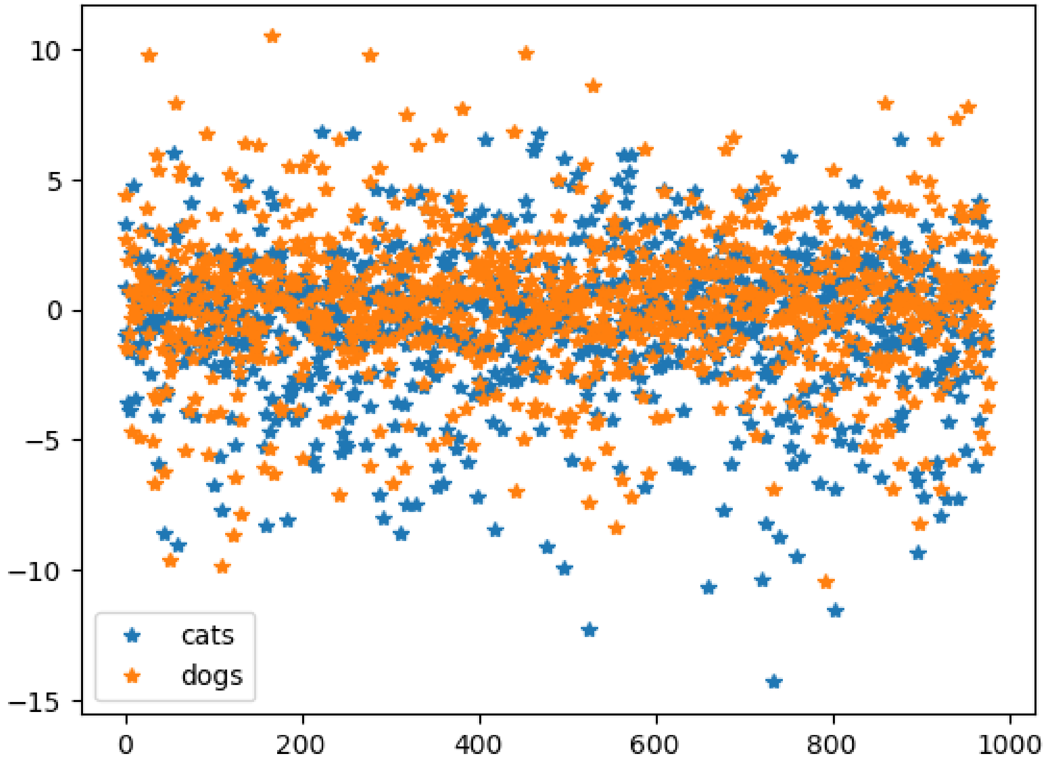
[6]:
```
plot( net2(train_data[train_label==0]).view(-1).cpu().detach().numpy(), '*',␣
↪label='cats')
plot( net2(train_data[train_label==1]).view(-1).cpu().detach().numpy(), '*',␣
↪label='dogs')
legend()
```

[6]:
```
<matplotlib.legend.Legend at 0x7f3ba1400310>
```



[7]:
```
plot( net2(valid_data[valid_label==0]).view(-1).cpu().detach().numpy(), '*',␣
 ↪label='cats')
plot( net2(valid_data[valid_label==1]).view(-1).cpu().detach().numpy(), '*',␣
 ↪label='dogs')
legend()
```

[7]: <matplotlib.legend.Legend at 0x7f3ba147a3e0>



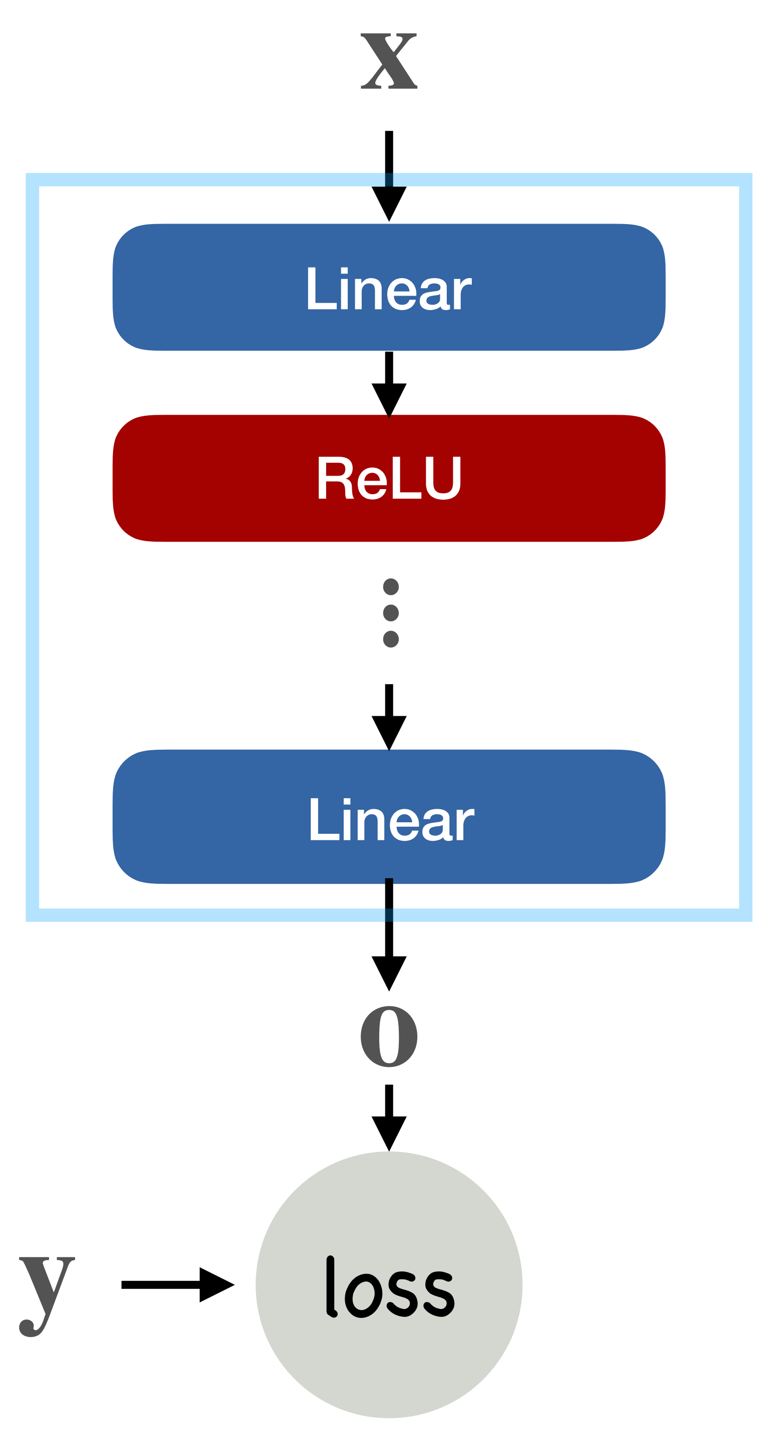
10
5
0
−5
−10
−15
0
200
400
600
800
1000
cats
dogs

[ ]:

What is a layer?

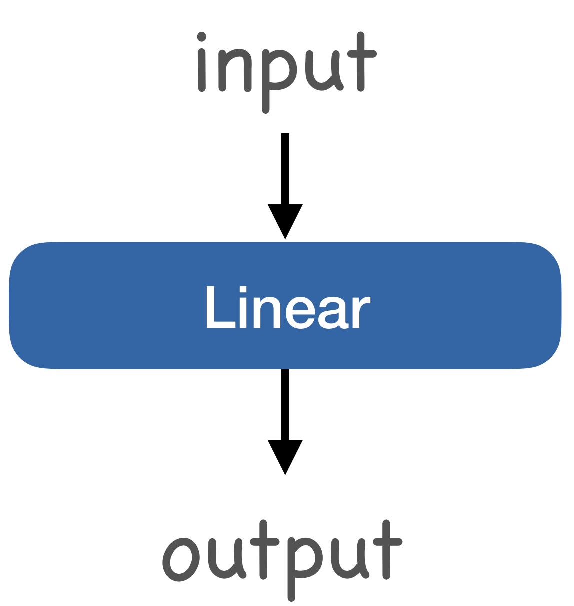
Examples of layers

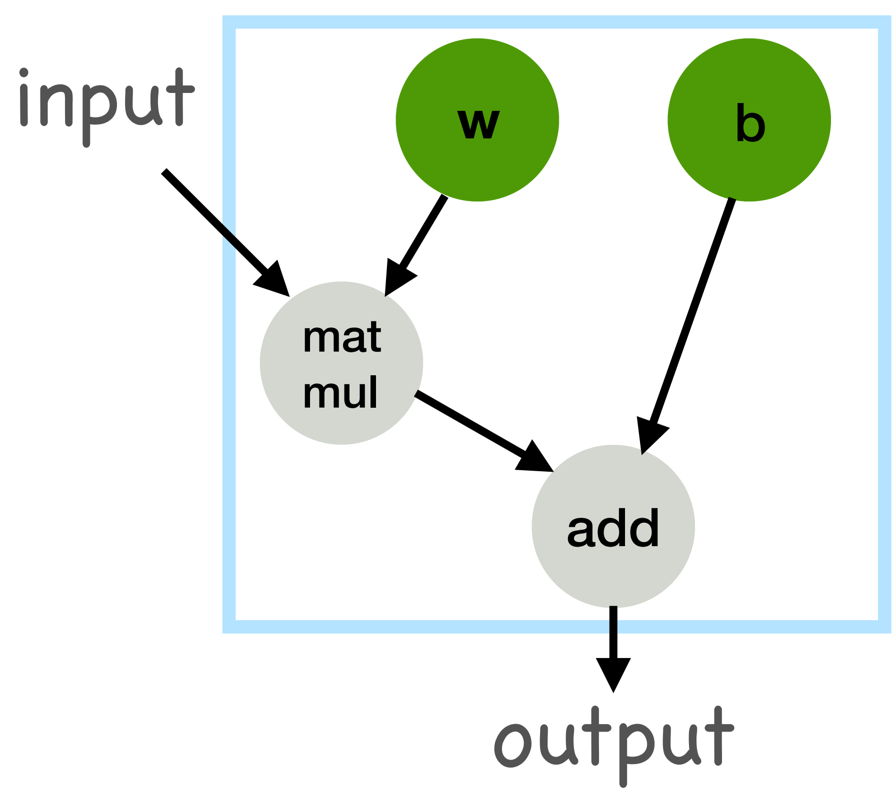
- Linear
- ReLU
- Loss functions



What is a layer?

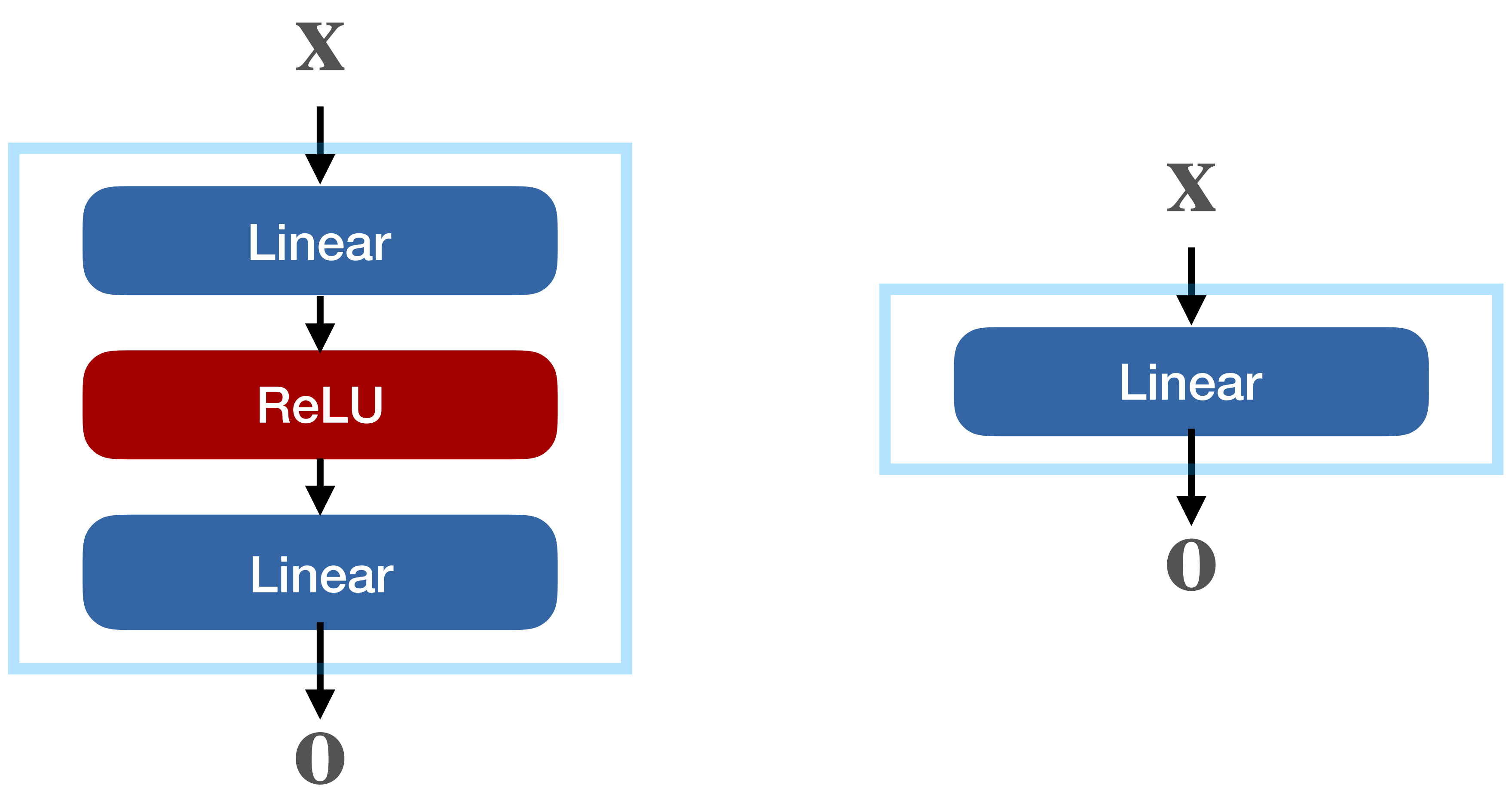
- Largest computational unit that remains unchanged throughout different architectures



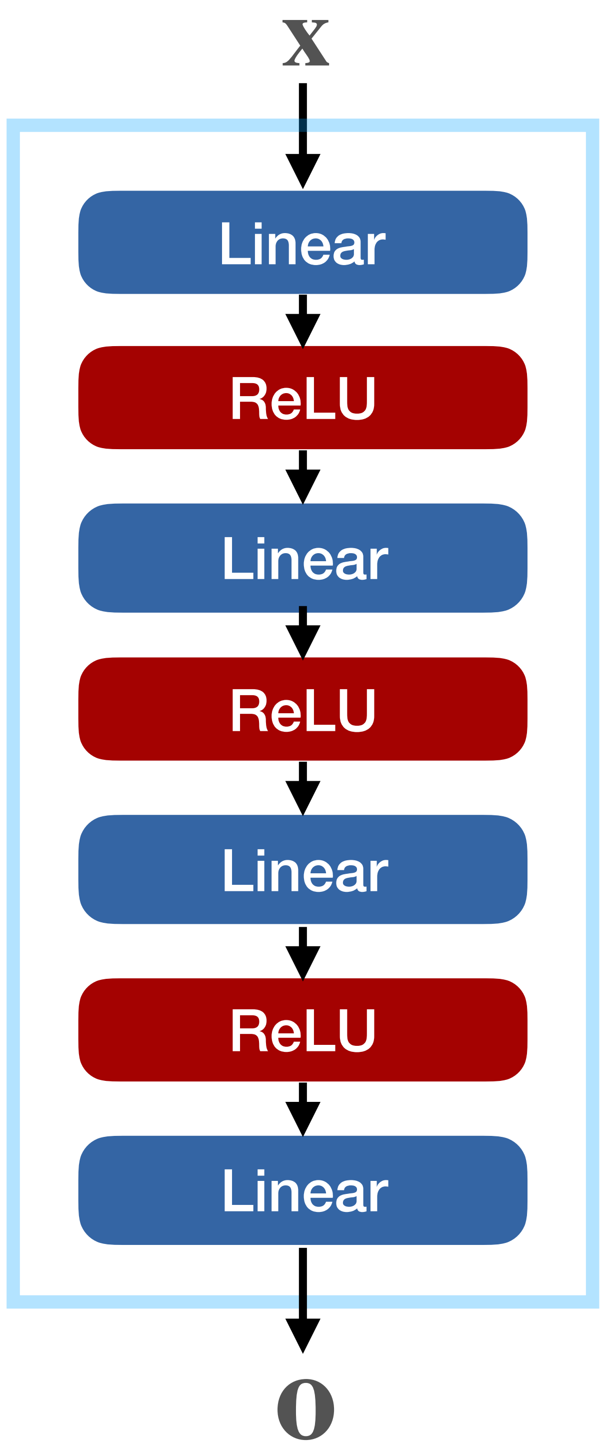


How many layers does a deep network have?

- We only count linear layers



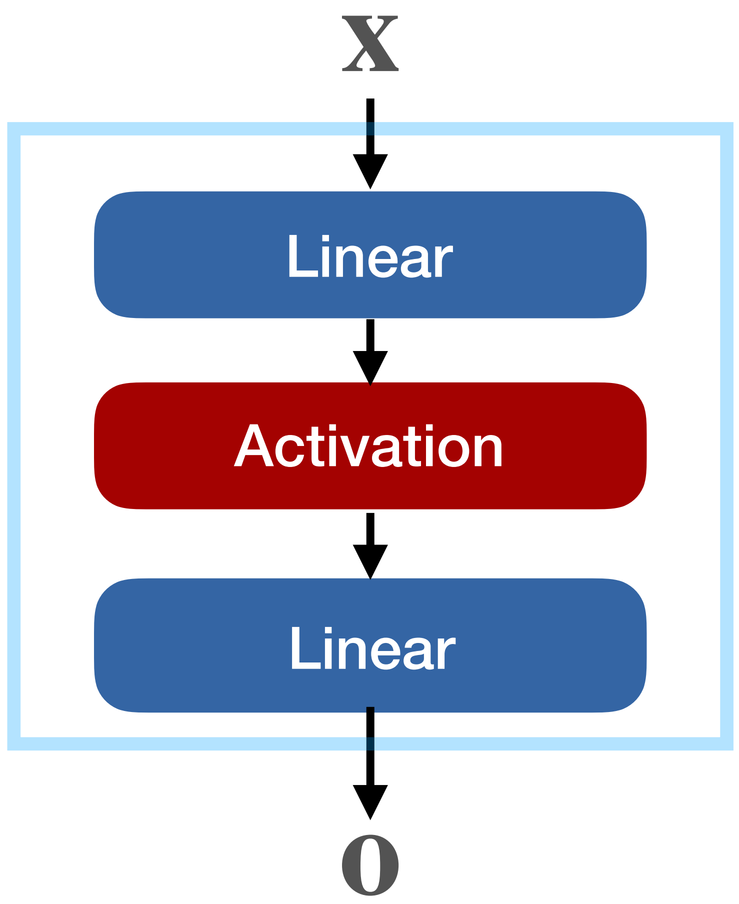
Layer naming



Activation functions

Non-linearities

- Allow a deep network to model arbitrary differentiable functions



Zoo of activation functions

ReLU

Leaky ReLU

PReLU

ELU

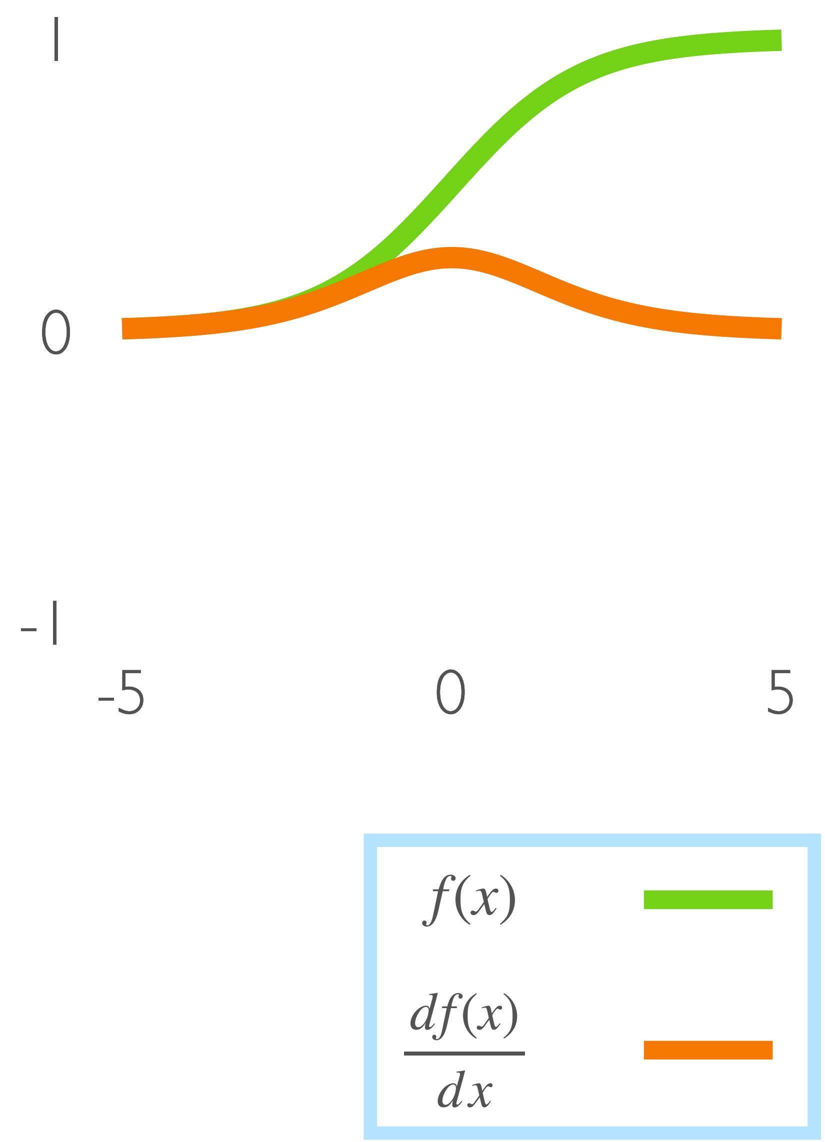
tanh

Sigmoid

Maxout

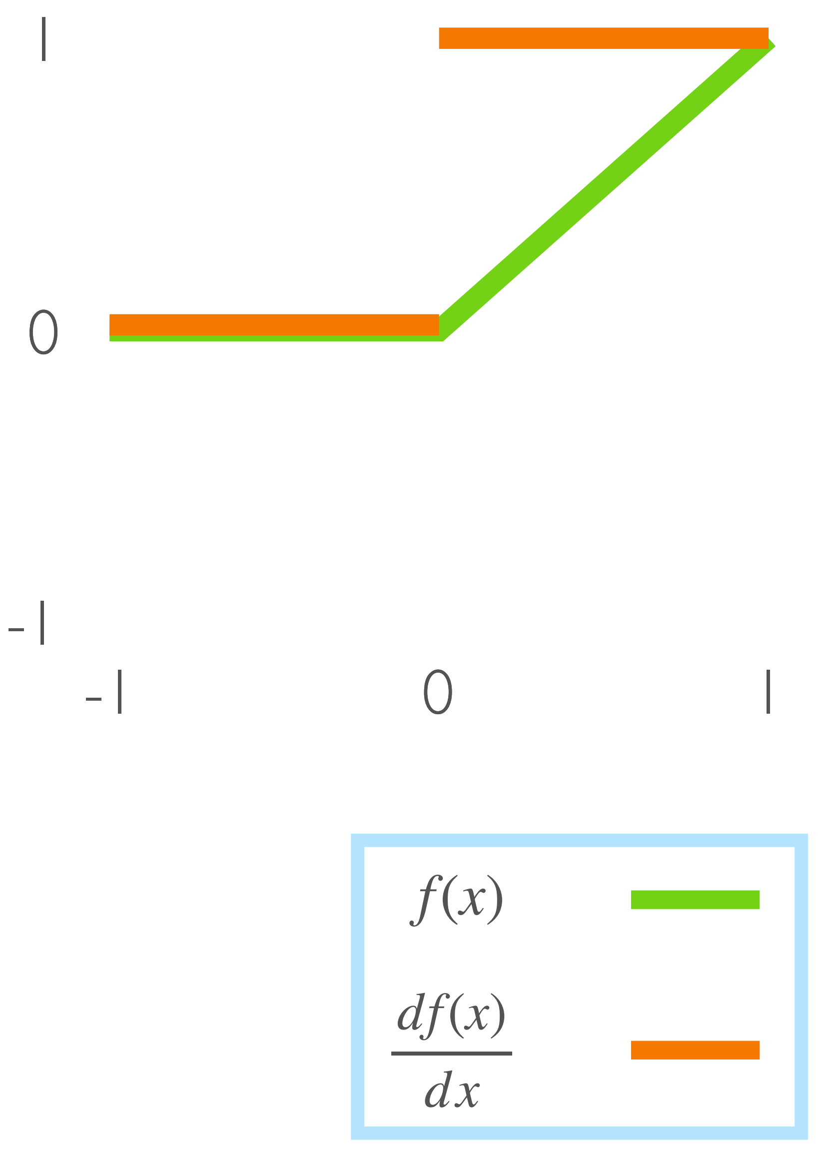
Sigmoid

- $$\frac{1}{1 + e^{-x}}$$
- Same as tanh
- Do not use



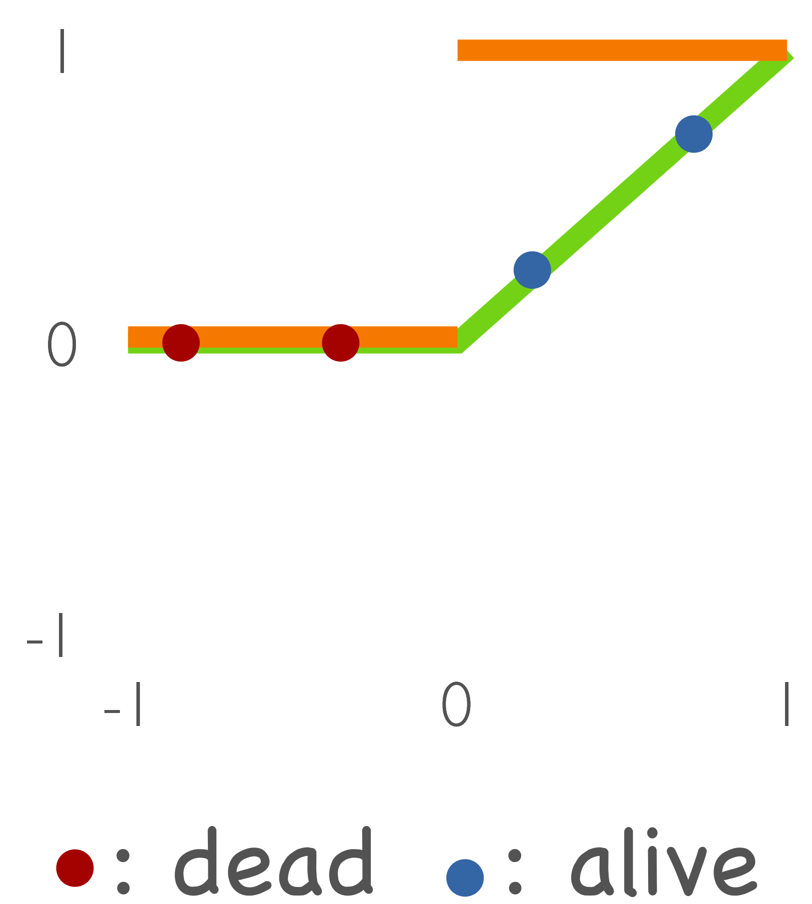
ReLU

- $\max(x, 0)$



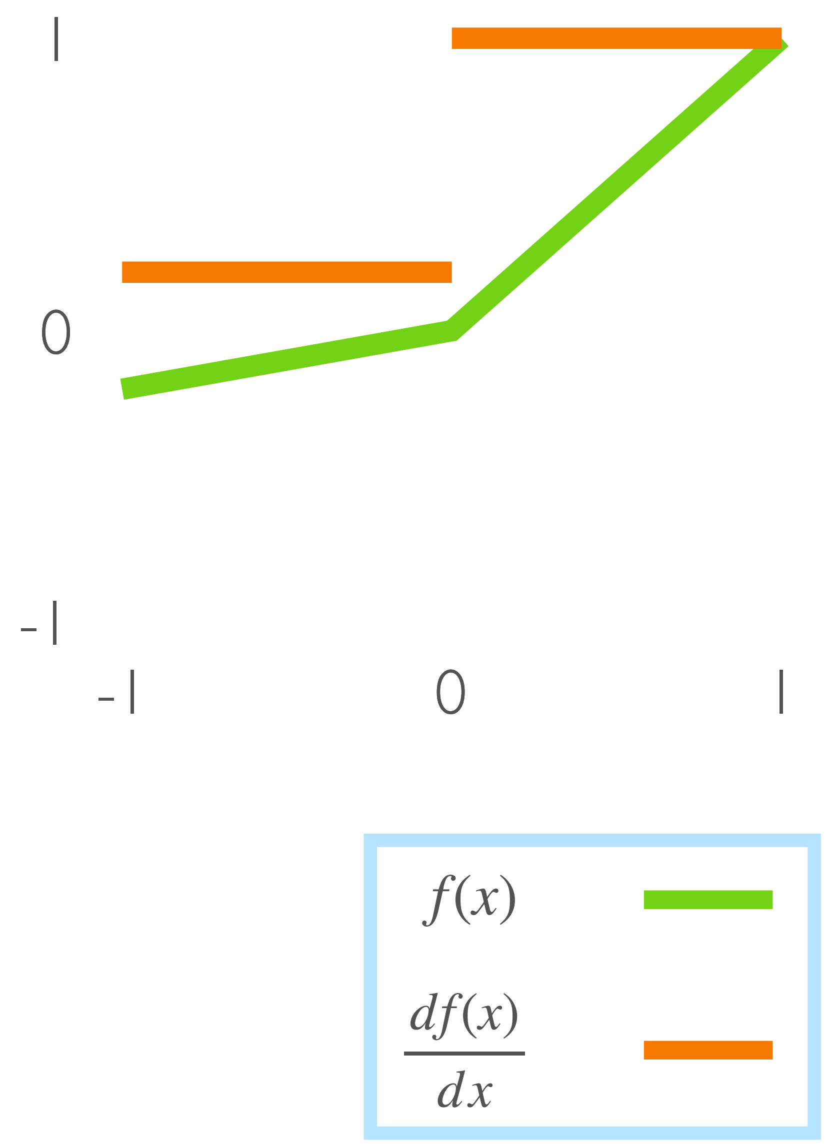
Dead ReLUs

- Prevent dead ReLUs:
  - Initialize Network carefully
  - Decrease the learning rate



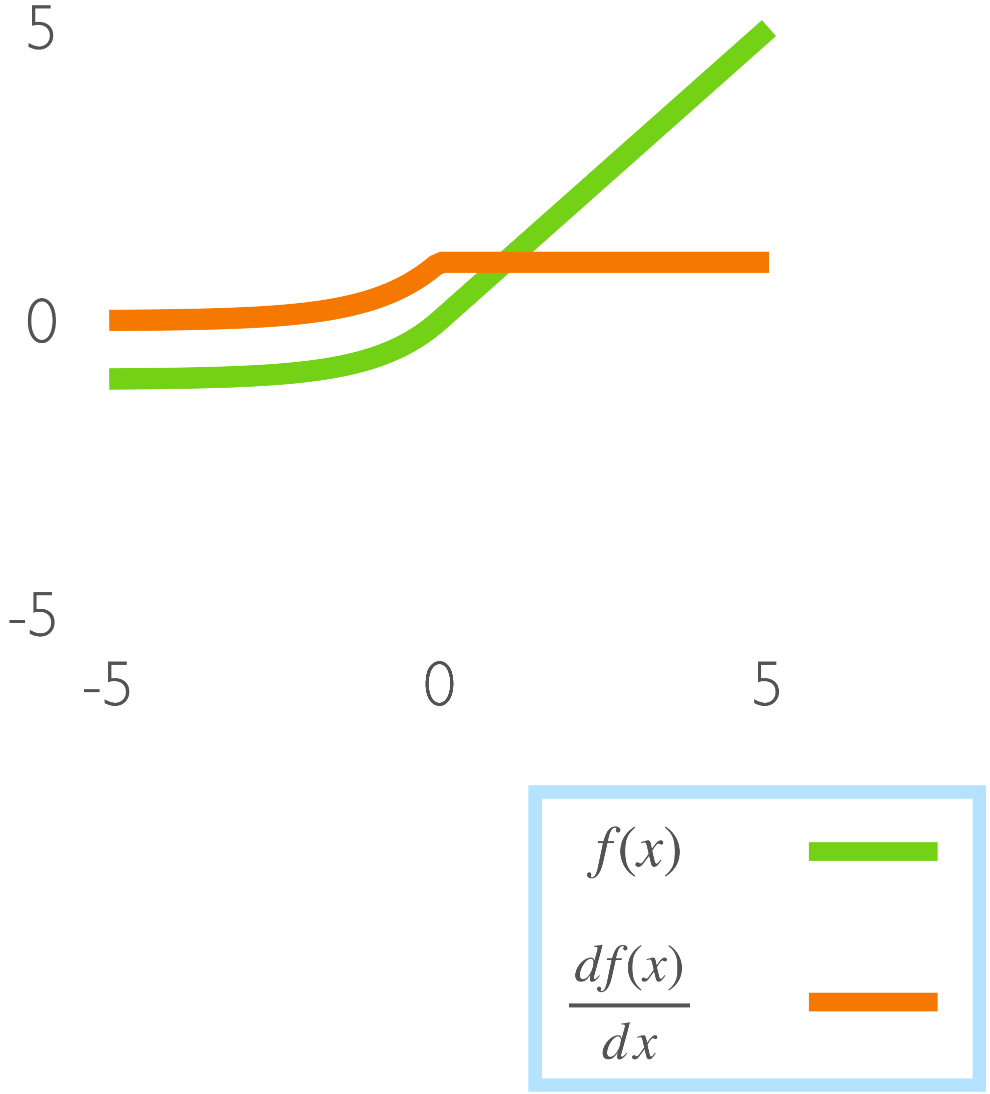
Leaky ReLU

- $\max(x, \alpha x)$
- For $0 < \alpha < 1$
- Called PReLU if α is learned



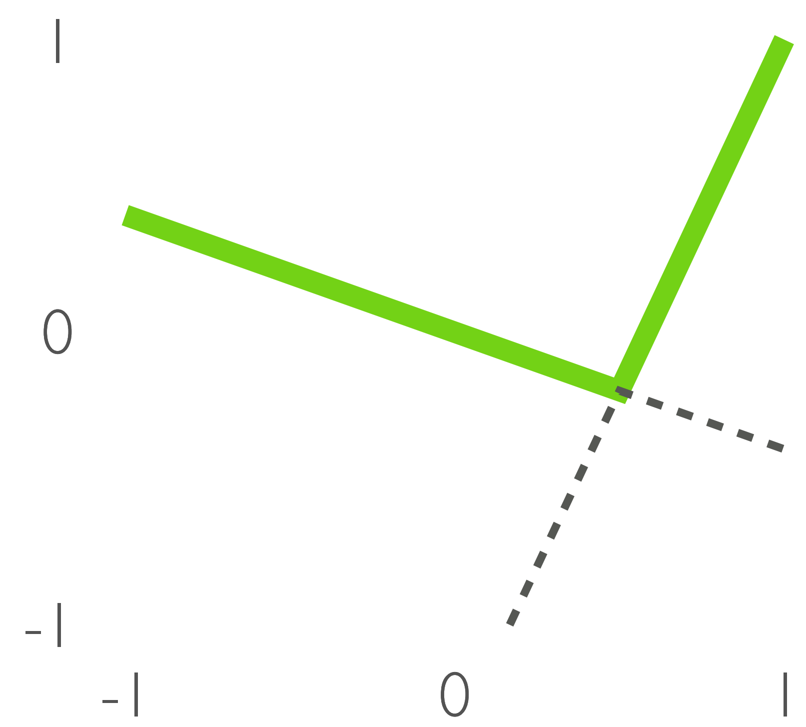
ELU

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



Maxout

- $\max(x_1, x_2)$



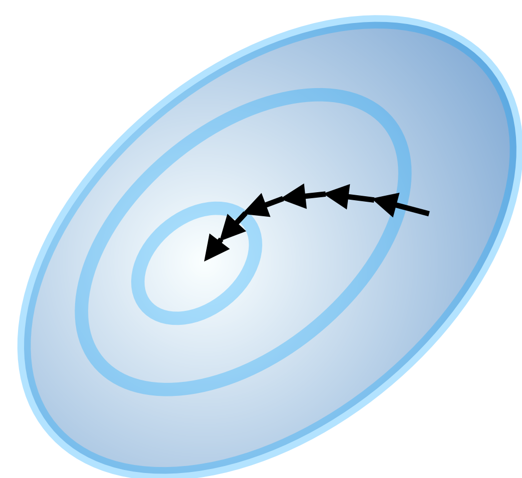
Which activation to choose?

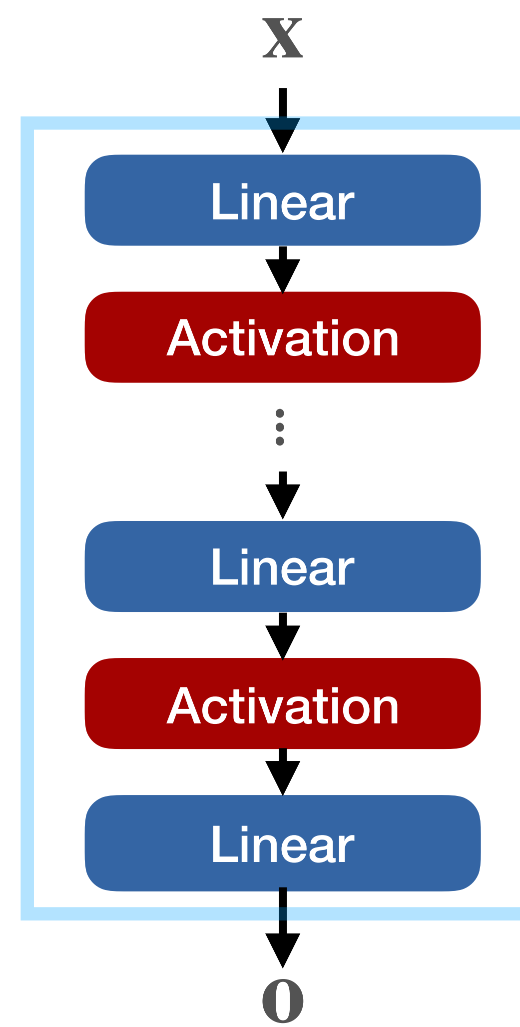
- ReLU
  - Carefully initialize
  - Small enough learning rate
- If ReLU fails, try:
  - Leaky ReLU / PReLU
- Avoid sigmoid and tanh

Hyper-parameters

Hyper-parameters

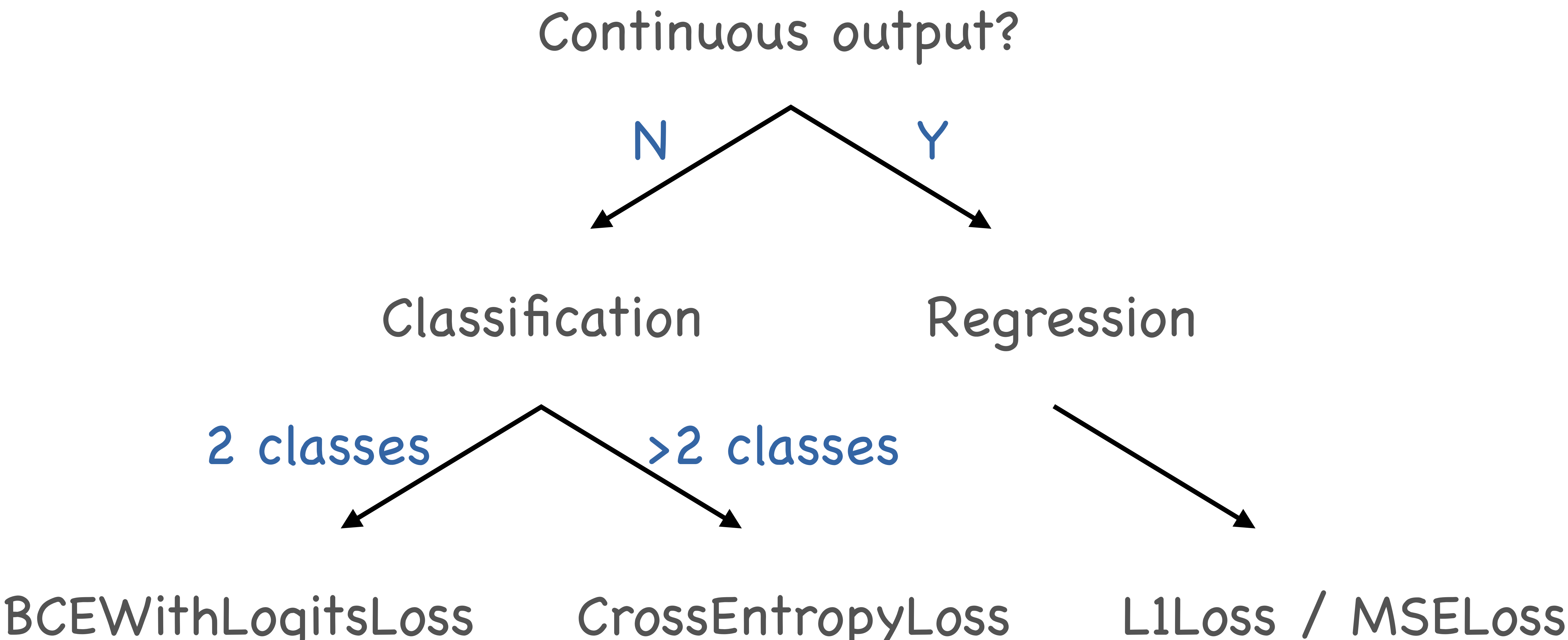
- Any parameters set by hand and not learned by SGD





Summary, a practical guide to deep network design

Loss functions



Activation functions

- Try ReLU
- If ReLU fails, try:
    - Leaky ReLU / PReLU

SGD

- Select batch size
- Tune learning rate
- momentum = 0.9