Linear classification and regression

© 2019 Philipp Krähenbühl and Chao-Yuan Wu

A simple example

- Breed
- Height
- Does bite?
- Age
- Number of teeth

Regression vs classification

Regression

- Predicts real value
- Order/distance matters
- Objective:
  - getting close to real value

Classification

- Predicts integer / class
- Objective:
  - Guess the right class

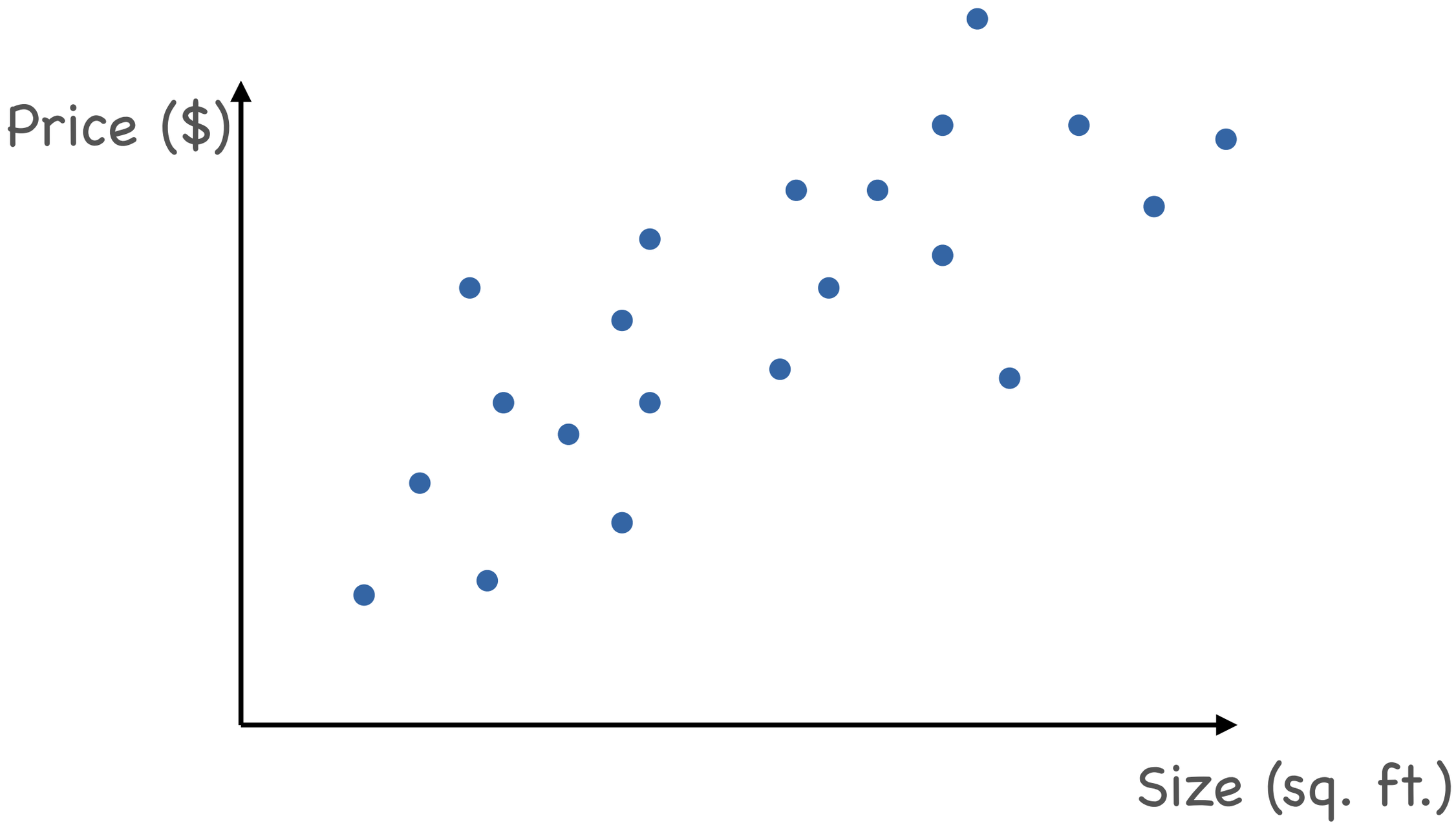
Linear regression

© 2019 Philipp Krähenbühl and Chao-Yuan Wu

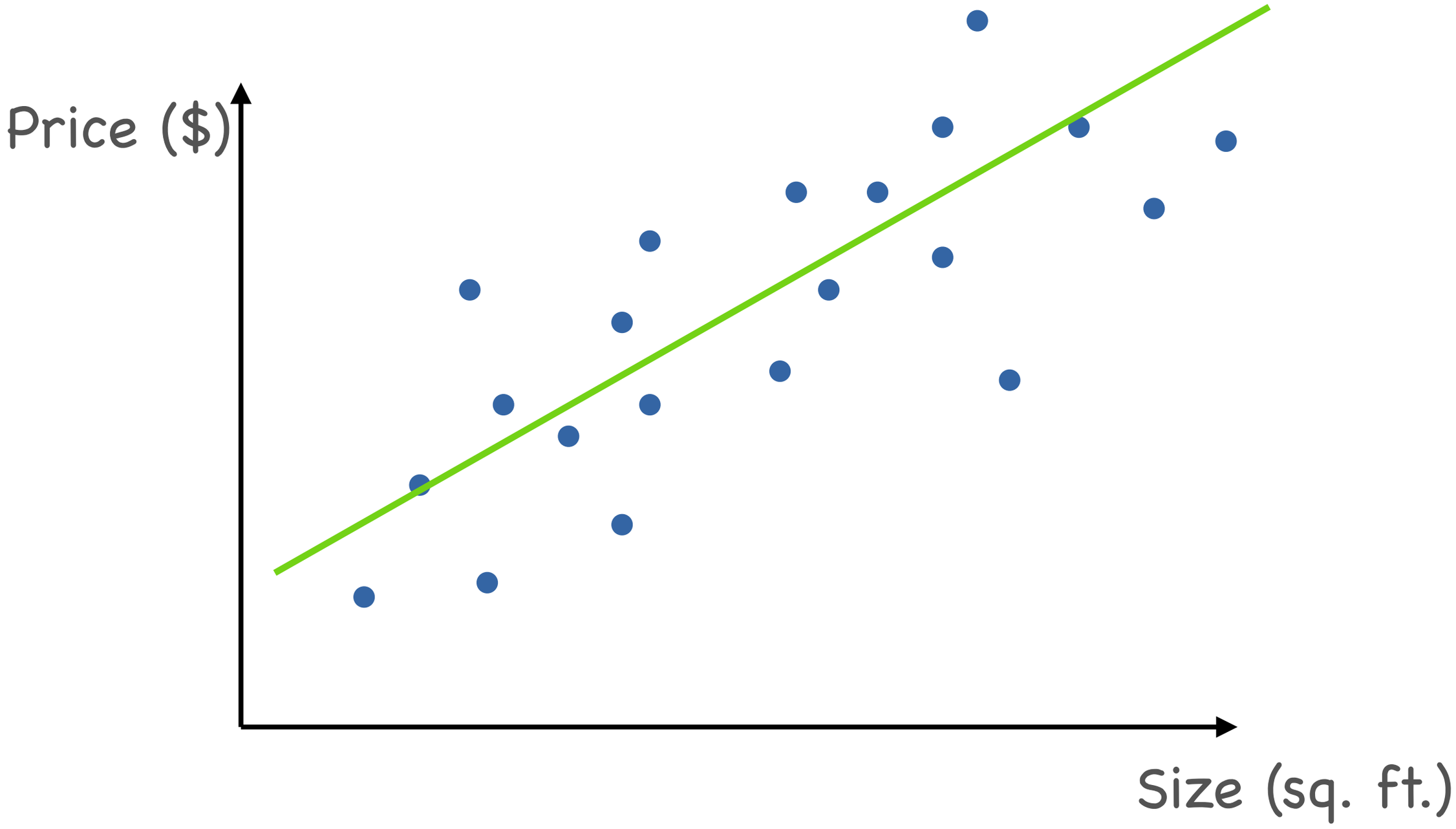
A simple example

- $1,200,000
- 3200 sq. ft.
- 5 beds
- 3 baths

A simple example



A simple example



Linear regression

- Input: $\mathbf{x}$ (tensor)
- Output: $y \in \mathbb{R}$
- Parameters: $\mathbf{w}, b$
- Regression: $\hat{y} = \mathbf{w}^\top \mathbf{x} + b$
- Loss: $\ell(\hat{y}, y) = |\hat{y} - y|$
  or $\ell(\hat{y}, y) = (\hat{y} - y)^2$

Optimization

- Loss: $\ell(\hat{y}, y) = (\hat{y} - y)^2$

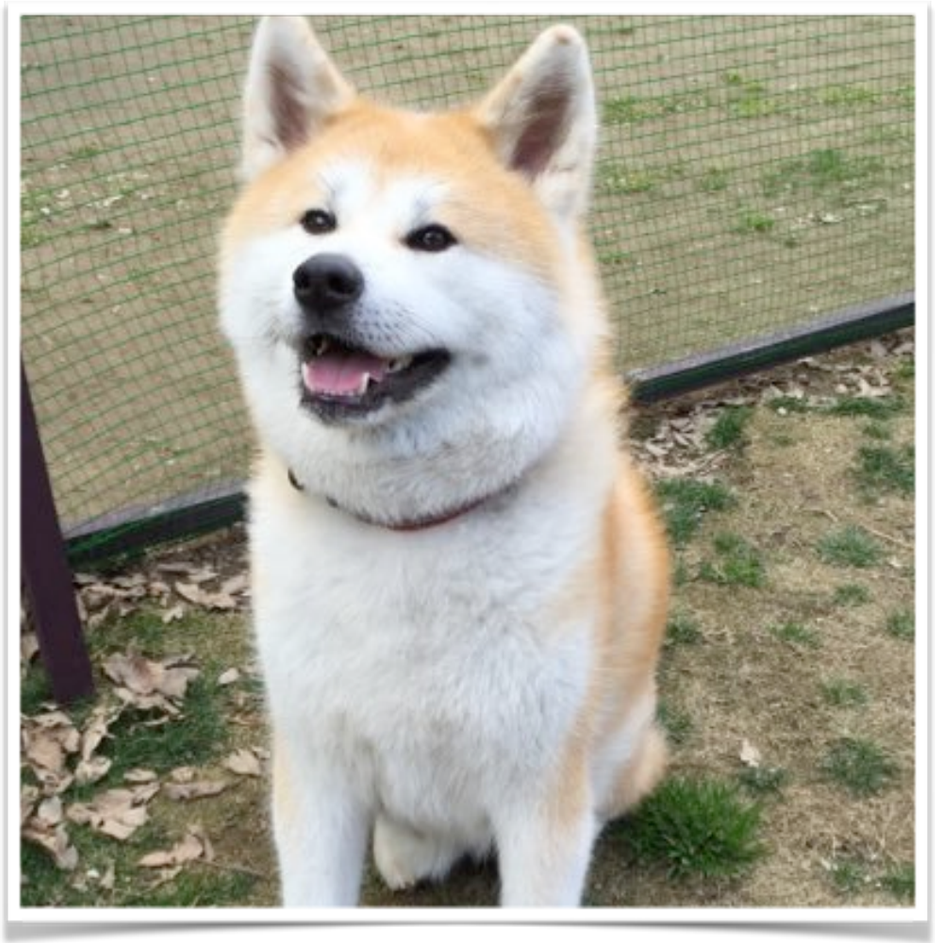
$$= (\mathbf{w}^\top \mathbf{x} + b - y)^2$$

Linear Classification

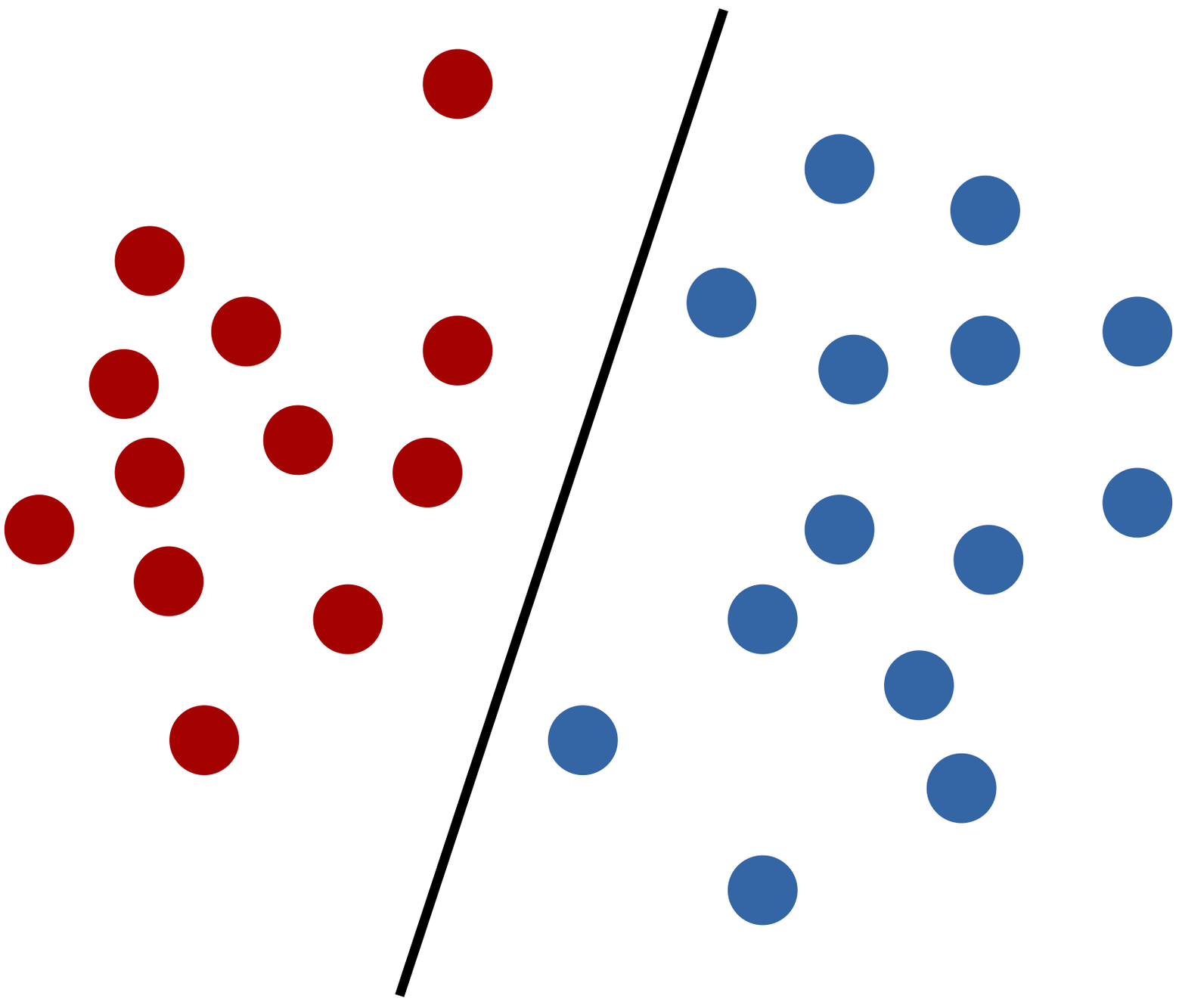
© 2019 Philipp Krähenbühl and Chao-Yuan Wu

A simple example

VS



Linear binary classifier



Linear binary classifier

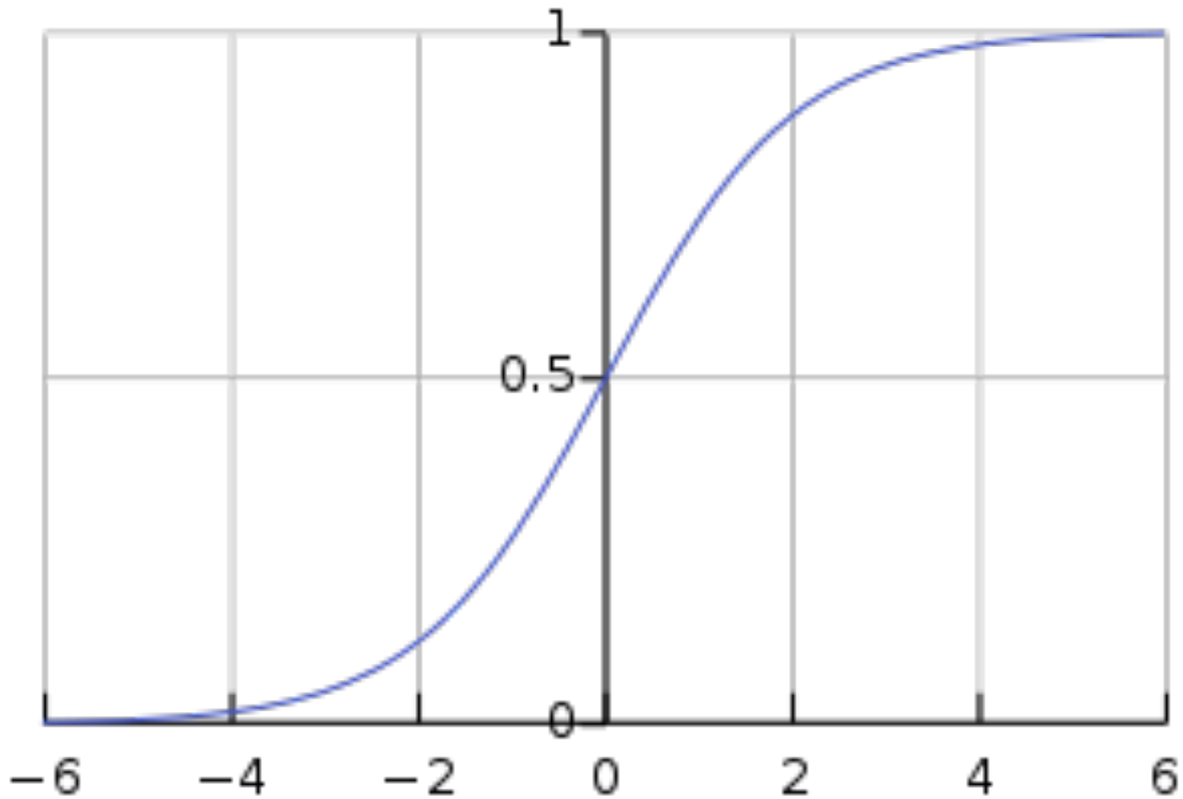
- Input: $\mathbf{x}$ (tensor)
- Label: $y \in \{0,1\}$
- Parameters: $\mathbf{w}, b$
- Prediction $\hat{y} = \begin{cases} 1 & \text{if } \mathbf{w}^\top \mathbf{x} + b > 0 \\ 0 & \text{otherwise} \end{cases}$

Sigmoid function

- Maps $\mathbb{R}$ to $[0,1]$

- $o = \mathbf{w}^\top \mathbf{x} + b$

$$p(y = 1) = \sigma(o) = \frac{1}{1 + e^{-o}}$$



Logistic regression

- Input: $\mathbf{x}$ (tensor)
- Label: $y \in \{0,1\}$
- Parameters: $\mathbf{w}, b$
- $o = \mathbf{w}^\top \mathbf{x} + b$
  - $$\begin{aligned} p(y=1) &= \sigma(o) \\ p(y=0) &= 1 - \sigma(o) \end{aligned} \quad \leftrightarrow \quad p(y) = \sigma(o)^y (1-\sigma(o))^{1-y}$$
- Loss (negative log likelihood):
  - $$-\log p(y) = -y \log(\sigma(o)) - (1-y)\log(1-\sigma(o))$$

03

January 23, 2024

[ ]:
```
%pylab inline
import torch
```

[ ]:
```
x = torch.rand([1000,2])
scatter(*x.numpy().T)
axis('equal')
```

[ ]:
```
x_in_circle = (x**2).sum(1) < 1

scatter(*x.numpy().T, c=x_in_circle.numpy())
axis('equal')
```

[ ]:
```
weights = torch.as_tensor([1,1], dtype=torch.float)
bias = torch.as_tensor(-1, dtype=torch.float)

def classify(x, weights, bias):
    return (x * weights[None,:]).sum(dim=1) + bias > 0

def accuracy(pred_label):
    return (pred_label==x_in_circle).float().mean()

def show(y):
    scatter(*x.numpy().T, c=y.detach().numpy())
    axis('equal')

pred_y = classify(x, weights, bias)
show(pred_y)
print('accuracy', accuracy(pred_y))
```

[ ]:
```
def predict(x, weights, bias):
    logit = (x * weights[None,:]).sum(dim=1) + bias
    return 1/(1+(-logit).exp())

def loss(prediction):
    return -(x_in_circle.float()     * (prediction+1e-10).log() +
             (1-x_in_circle.float()) * (1-prediction+1e-10).log() ).mean()
```

```
p_y = predict(x, weights, bias)
print( 'loss =', loss(p_y), 'accuracy =', accuracy(pred_y) )
```

[ ]:
```
weights = torch.as_tensor([-1,-1], dtype=torch.float)
bias = torch.as_tensor(1.0, dtype=torch.float)

pred_y = classify(x, weights, bias)
p_y = predict(x, weights, bias)

show(pred_y)
print( 'loss =', loss(p_y), 'accuracy =', accuracy(pred_y) )
```

[ ]:
```
weights = torch.as_tensor([-1,-1], dtype=torch.float)
bias = torch.as_tensor(1.2, dtype=torch.float)

pred_y = classify(x, weights, bias)
p_y = predict(x, weights, bias)

show(pred_y)
print( 'loss =', loss(p_y), 'accuracy =', accuracy(pred_y) )
```

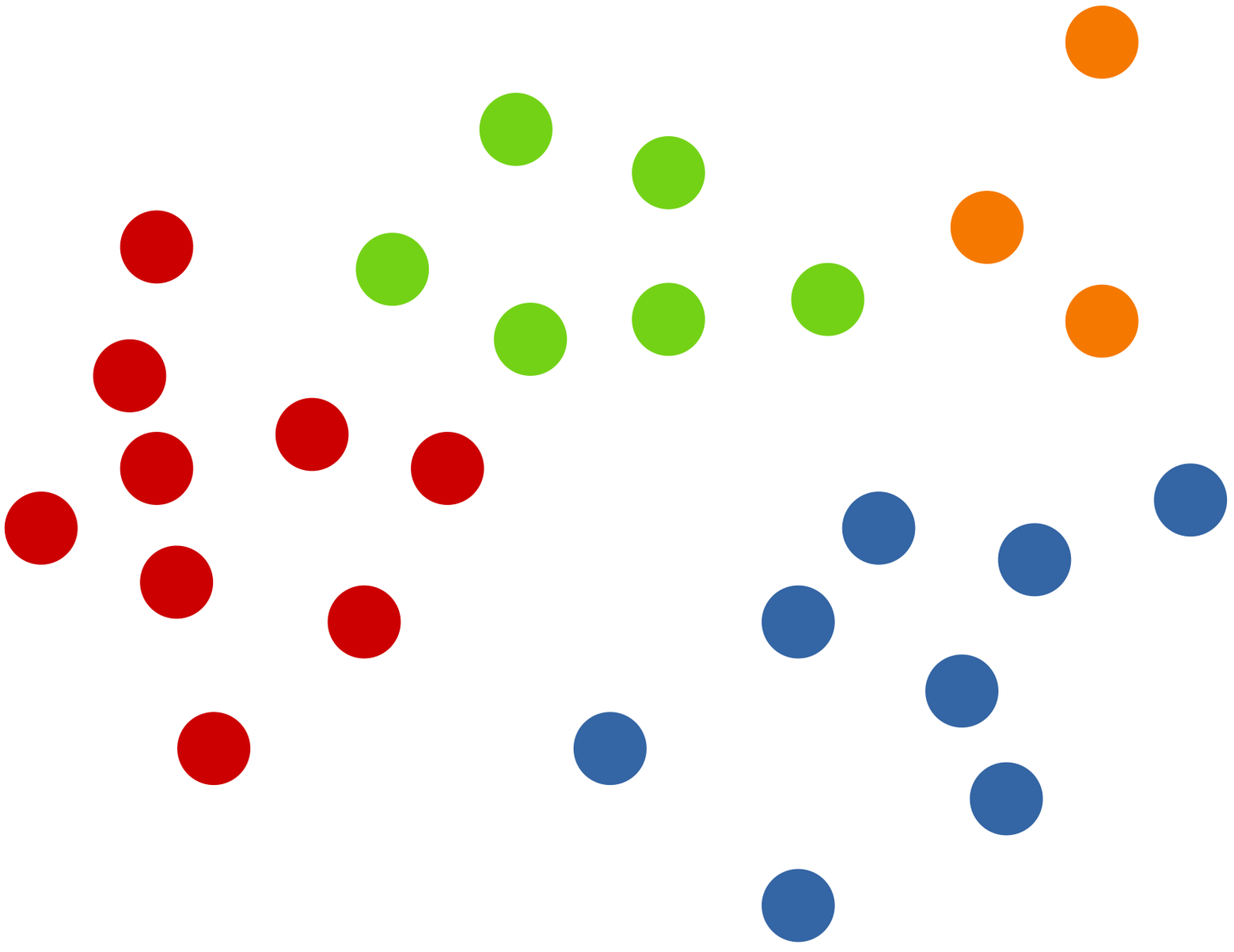
[ ]:
```

```

Linear multi-class classification

© 2019 Philipp Krähenbühl and Chao-Yuan Wu

A simple example

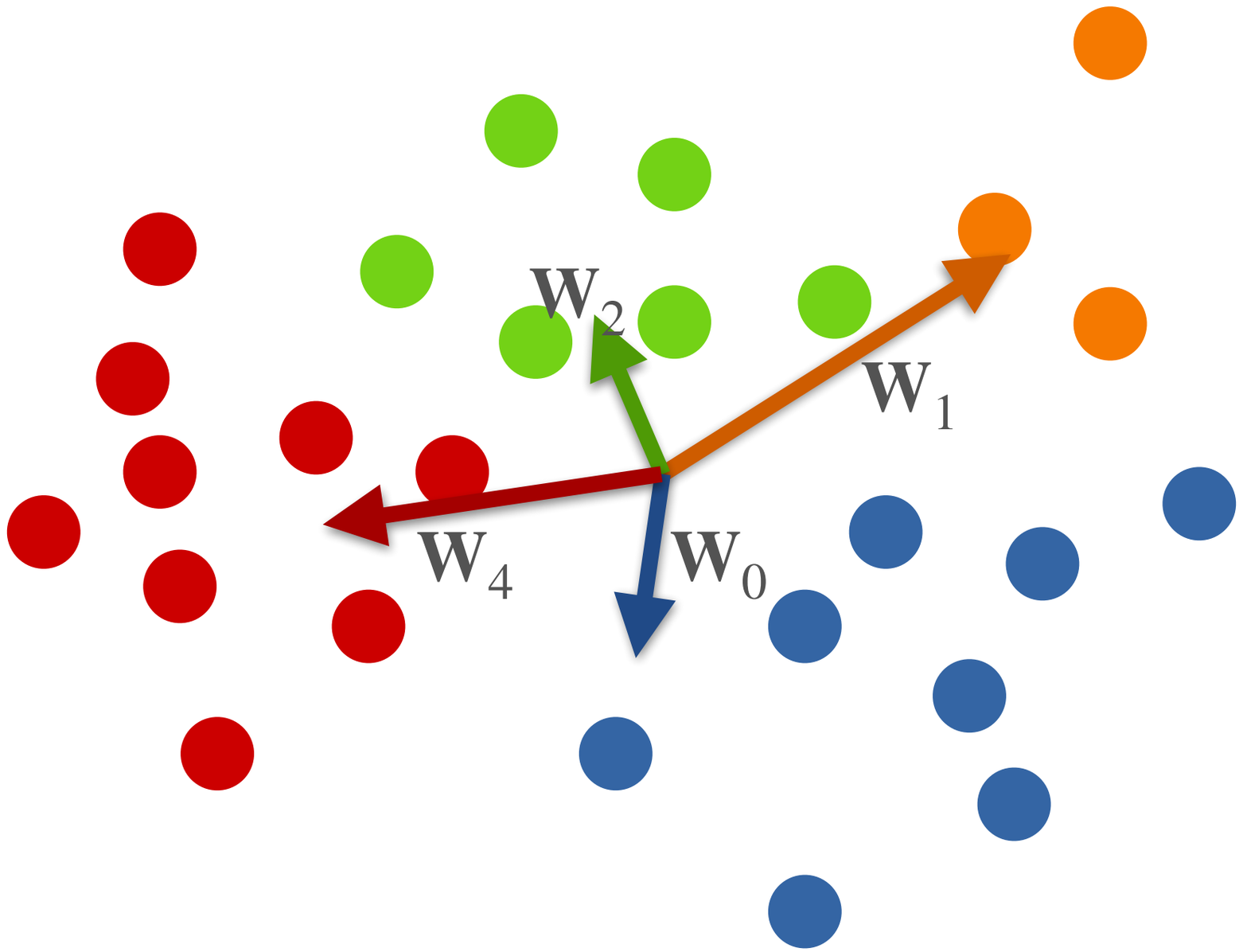
Multinomial linear classifier



Multinomial linear classifier

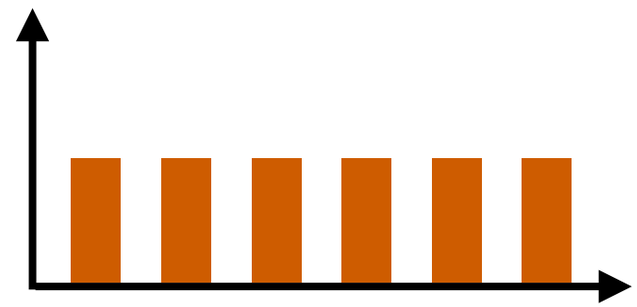
- Input: $\mathbf{x} \in \mathbb{R}^d$ (tensor)
- Label: $y \in \{0,1,\ldots,k\}$
- Parameters: $\mathbf{W} \in \mathbb{R}^{d \times k}, \mathbf{b} \in \mathbb{R}^k$
- $P(y) = \text{softmax}(\mathbf{W}^\top \mathbf{x} + \mathbf{b})_y$

Multinomial linear classifier



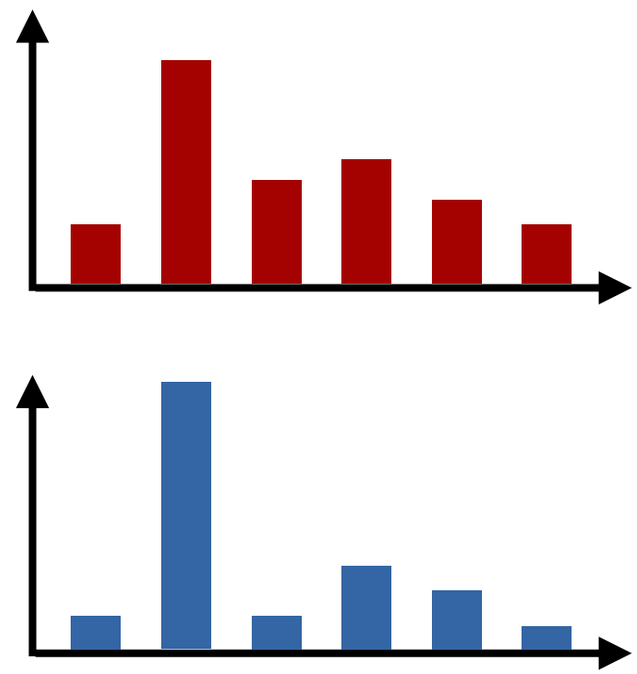
Softmax function

- Maps $\mathbb{R}^k$ to probability p



- $\mathbf{o} = \mathbf{W}^\top \mathbf{x} + \mathbf{b}$

$$\text{softmax}(\mathbf{o})_i = \frac{e^{o_i}}{\sum_{i'} e^{o_{i'}}}$$



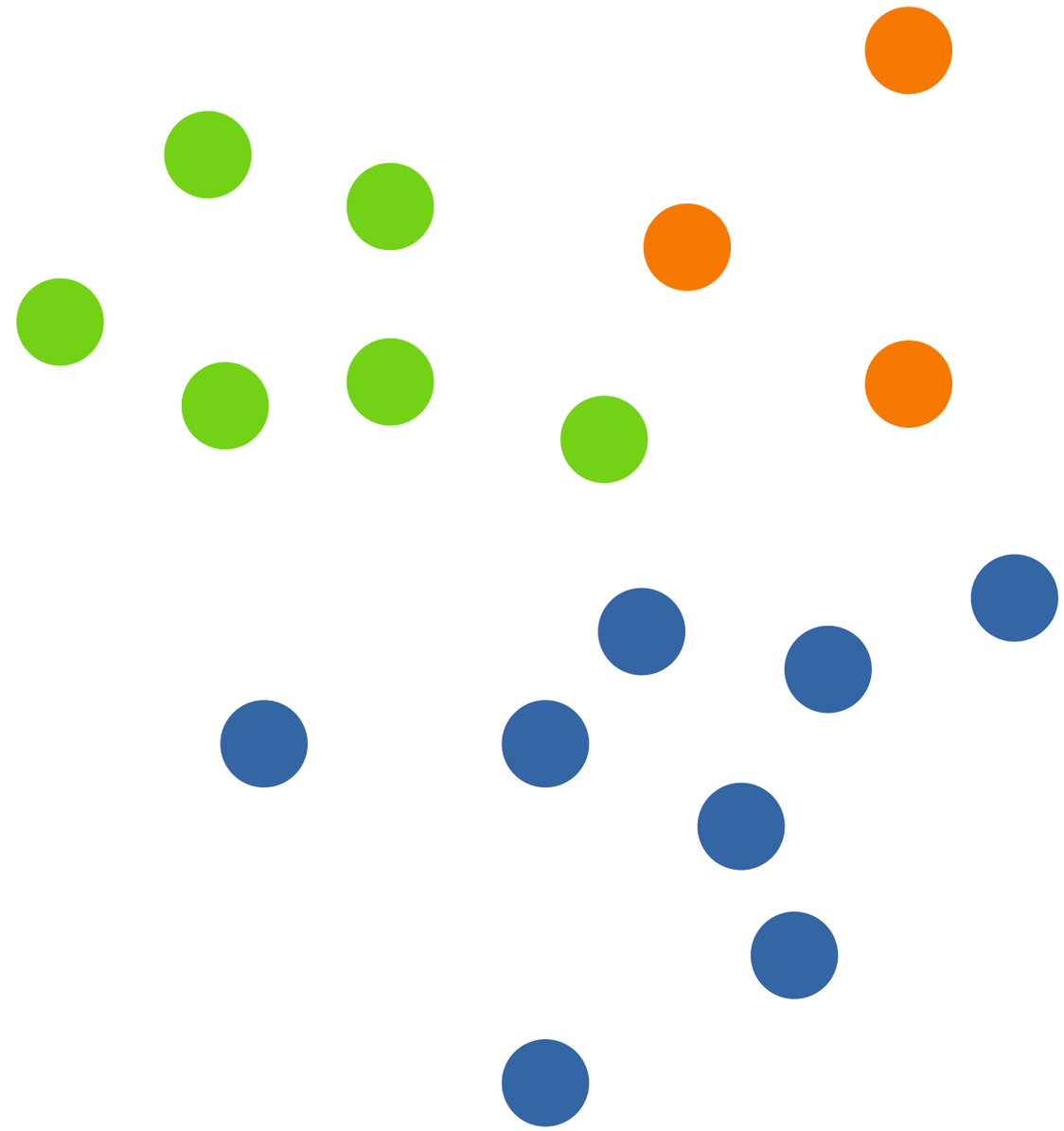
always sum to one

Multinomial logistic regression

- Input: $\mathbf{x} \in \mathbb{R}^d$ (tensor)
- Label: $y \in \{0,1,\ldots,k\}$
- Parameters: $\mathbf{W} \in \mathbb{R}^{d \times k}, \mathbf{b} \in \mathbb{R}^k$
- $P(y) = \text{softmax}(\mathbf{W}^\top \mathbf{x} + \mathbf{b})_y$
- Loss: $-\log p(y)$

One vs all classification

- Train n binary classifier
  - Not calibrated
- Do not use this!



Summary

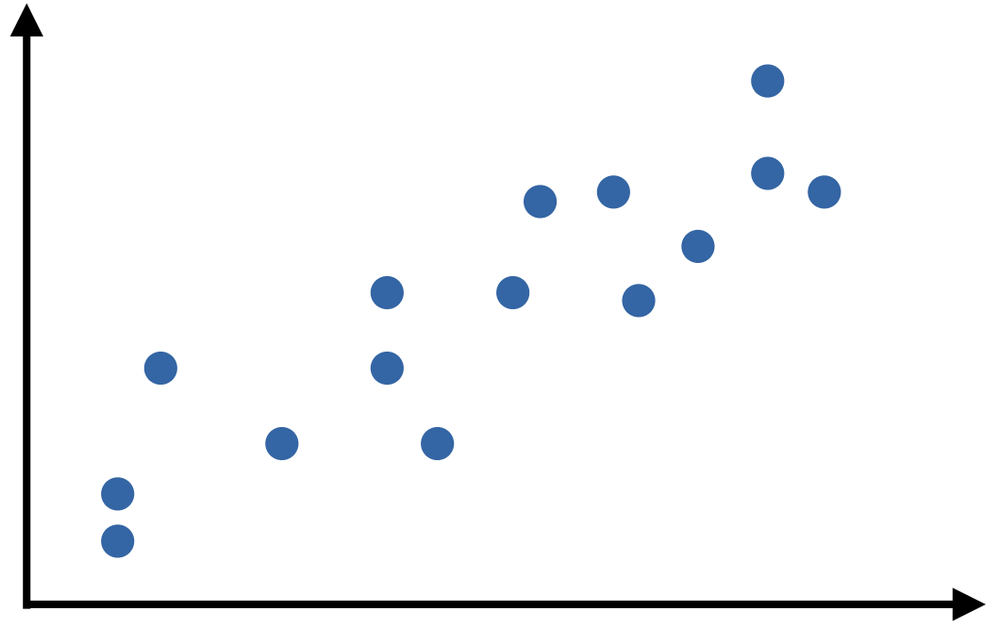
- Multinomial logistic regression
  - Multiple outputs
  - Softmax instead of sigmoid

Optimization & Gradient Descent

© 2019 Philipp Krähenbühl and Chao-Yuan Wu

Data

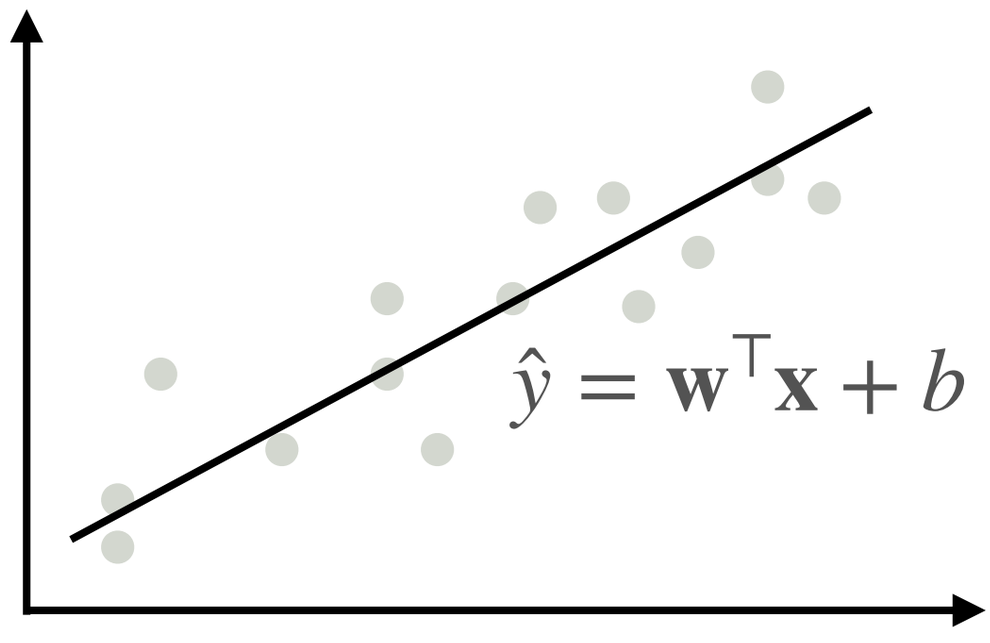
- Input: $\mathbf{x}$ (tensor)
- Output: y



Model

- Structure: e.g. Linear, Logistic or multinomial logistic regression

- Parameters: θ (e.g. $\mathbf{w}, b$)



Loss function

- Loss: $L(\theta) = \sum_i \ell(\theta \mid \mathbf{x}_i, y_i)$

Gradients

- L2 $\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|^2$ where $\hat{y} = \mathbf{w}^\top \mathbf{x} + b$
  - $\nabla_{\mathbf{w}} \ell(\hat{y}, y) = (\hat{y} - y)\mathbf{x}$
  - $\nabla_b \ell(\hat{y}, y) = \hat{y} - y$
- Sigmoid $\ell(o, y) = -\log p(y|o)$ where $o = \mathbf{w}^\top \mathbf{x} + b$
  - $\nabla_{\mathbf{w}} \ell(o, y) = (\sigma(o) - y)\mathbf{x}$
  - $\nabla_b \ell(o, y) = \sigma(o) - y$
- Softmax $\ell(\mathbf{o}, y) = -\log p(y|\mathbf{o})$ where $\mathbf{o} = \mathbf{W}^\top \mathbf{x} + \mathbf{b}$
  - $\nabla_{\mathbf{W}_i} \ell(\mathbf{o}, y) = (\text{softmax}(\mathbf{o})_i - [y = i])\mathbf{x}$
  - $\nabla_{\mathbf{b}_i} \ell(\mathbf{o}, y) = \text{softmax}(\mathbf{o})_i - [y = i]$

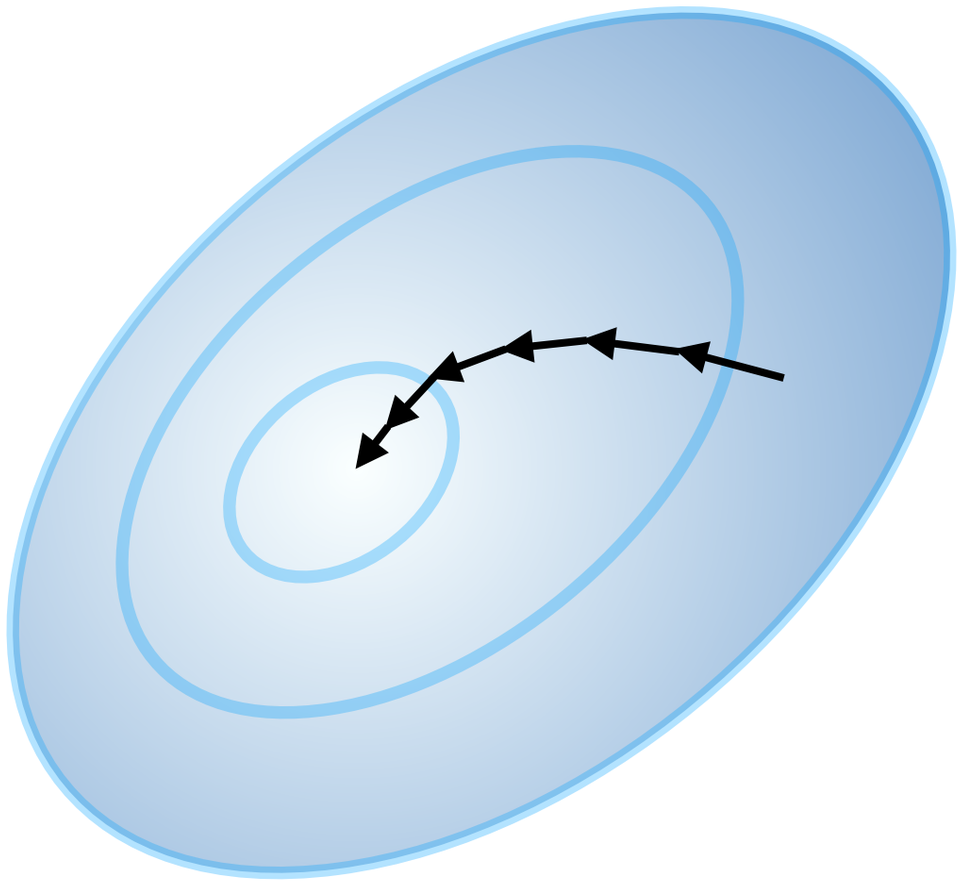
Optimization

- Find parameters θ
- With lowest loss $L(\theta)$

Gradient descent

- Start at random θ
- Update: $\theta' = \theta - \epsilon \frac{\partial L(\theta)}{\partial \theta}$
- $L(\theta') < L(\theta)$ if $\frac{\partial L(\theta)}{\partial \theta} ! = 0$

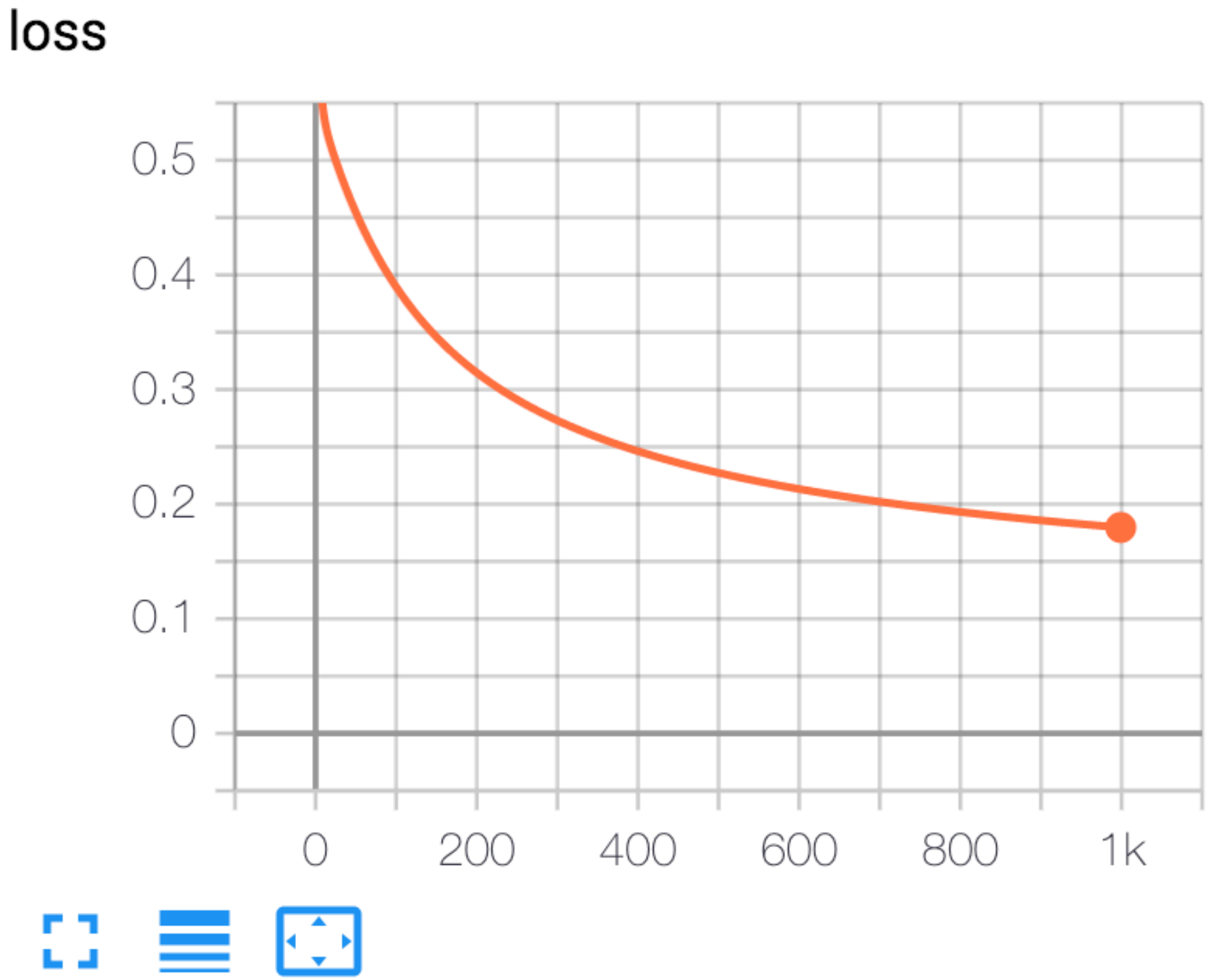
  and ϵ small enough



Gradient descent algorithm

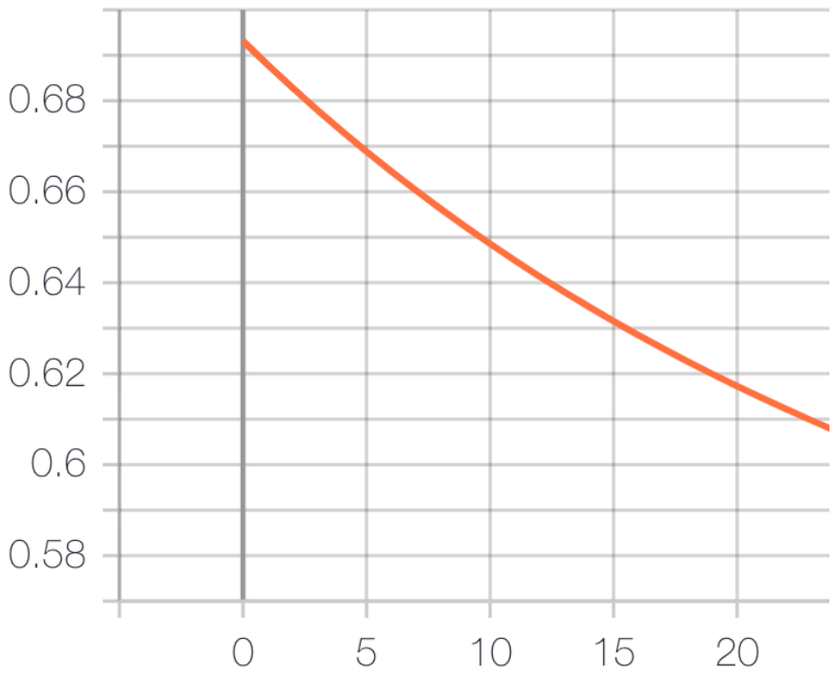
- Randomly initialize θ
- for n iterations
- compute loss $L(\theta)$ and gradient $g = \frac{\partial L(\theta)}{\partial \theta}$
- Update: $\theta -= \epsilon g$

Gradient descent in action

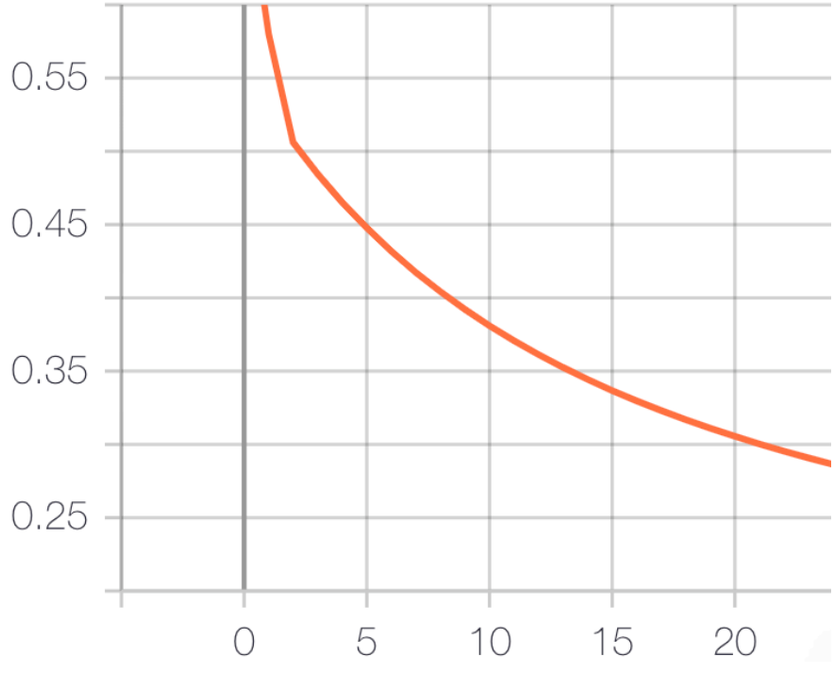


Learning rate matters

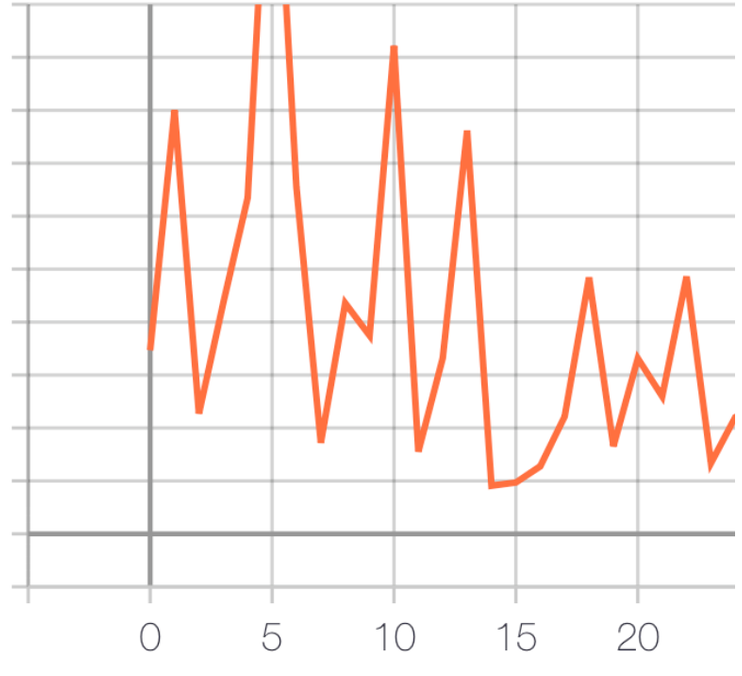
Too small



Just right



Too big



06

January 23, 2024

[ ]:
```
%pylab inline
import torch
```

[ ]:
```
x = torch.rand([1000,2])
x_in_circle = (x**2).sum(1) < 1

def classify(x, weights, bias):
    return (x * weights[None,:]).sum(dim=1) + bias > 0

def accuracy(pred_label):
    return (pred_label==x_in_circle).float().mean()

def show(y):
    scatter(*x.numpy().T, c=y.detach().numpy())
    axis('equal')

def predict(x, weights, bias):
    logit = (x * weights[None,:]).sum(dim=1) + bias
    return 1/(1+(-logit).exp())

def loss(prediction):
    return ( -x_in_circle.float()     * prediction.log() +
             -(1-x_in_circle.float()) * (1-prediction).log() ).mean()

show(x_in_circle)
```

[ ]:
```
weights = torch.as_tensor([-1,-1], dtype=torch.float)
bias = torch.as_tensor(1.0, dtype=torch.float)

pred_y = classify(x, weights, bias)
p_y = predict(x, weights, bias)

show(pred_y)
```

[ ]:
```
import torch.utils.tensorboard as tb
%load_ext tensorboard
import tempfile
```

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

[ ]:
```
logger = tb.SummaryWriter(log_dir+'/linear1')

weights = torch.as_tensor([-1,-1], dtype=torch.float)
bias = torch.as_tensor(1.0, dtype=torch.float)
label = x_in_circle.float()
for iteration in range(5000):
    p_y = predict(x, weights, bias)
    pred_y = classify(x, weights, bias)

    l = loss(p_y)

    logger.add_scalar("loss", l, global_step=iteration)
    logger.add_scalar("accuracy", accuracy(pred_y), global_step=iteration)
    if iteration % 10 == 0:
        fig = figure()
        show(pred_y)
        logger.add_figure('pred_y', fig, global_step=iteration)
        del fig

    # Gradient computation
    gradient_l_f = p_y - label.float()
    gradient_w = (gradient_l_f[:,None]*x).mean(0)
    gradient_b = (gradient_l_f).mean(0)

    # Gradient update
    weights -= 0.5*gradient_w
    bias    -= 0.5*gradient_b

show(pred_y)
```

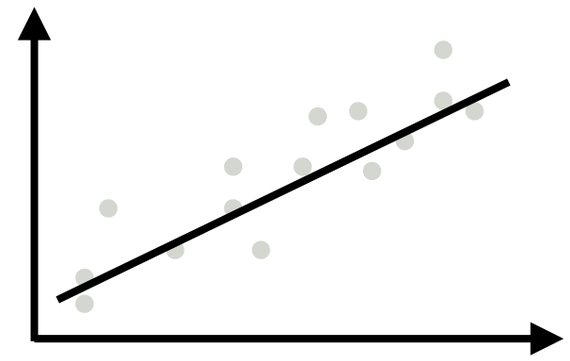
[ ]:
```

```

Computation Graphs

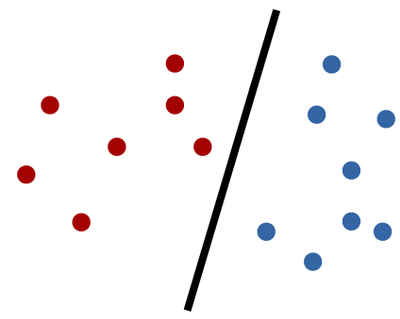
© 2019 Philipp Krähenbühl and Chao-Yuan Wu

Regression and classification

- Models

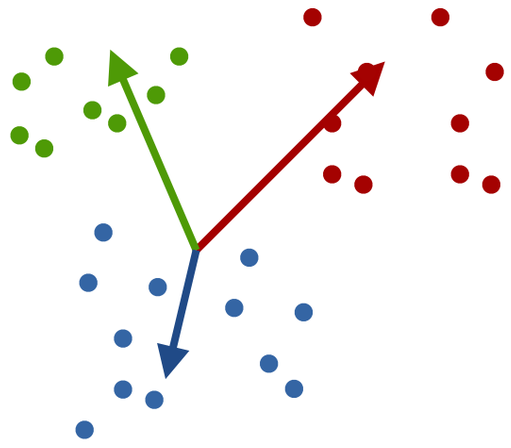


- $\hat{y} = \mathbf{w}^\top \mathbf{x} + b$



- $\hat{p}(1) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$

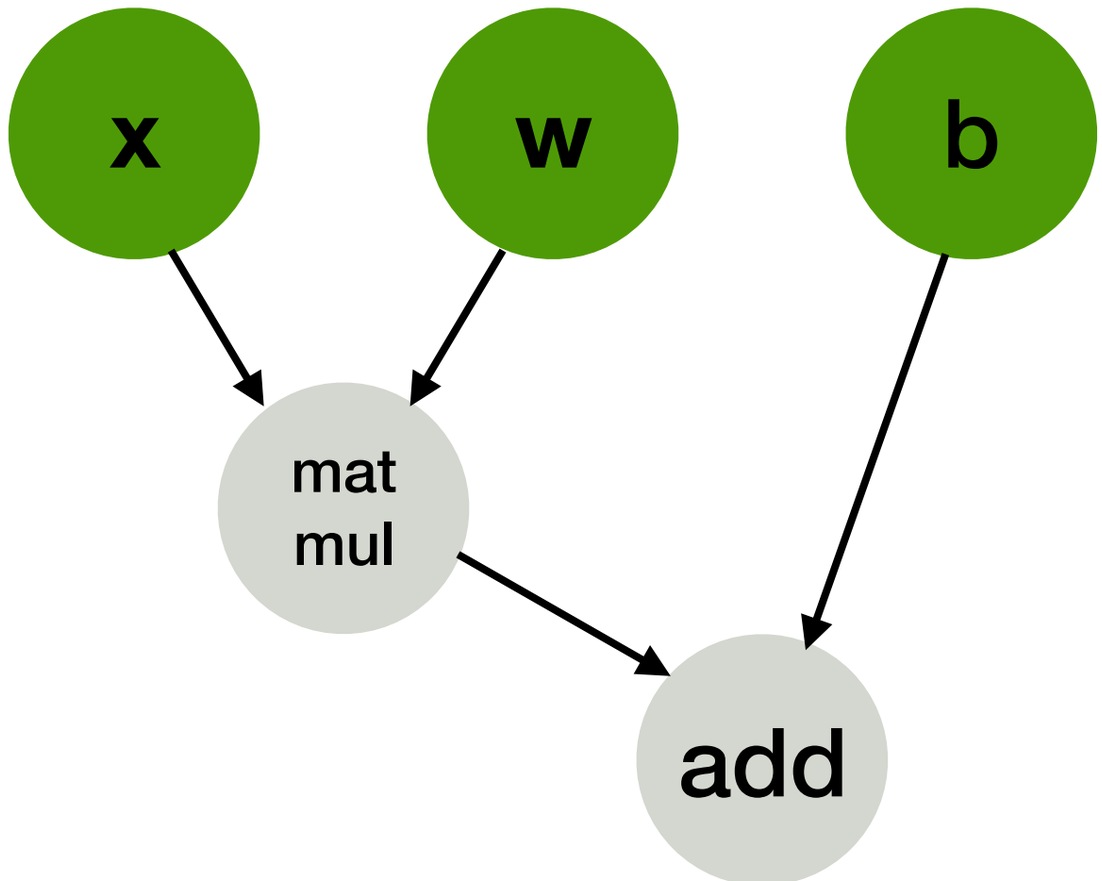
- $\hat{\mathbf{p}} = \text{softmax}(\mathbf{W}^\top \mathbf{x} + \mathbf{b})$



Computation Graph

- Linear regression:

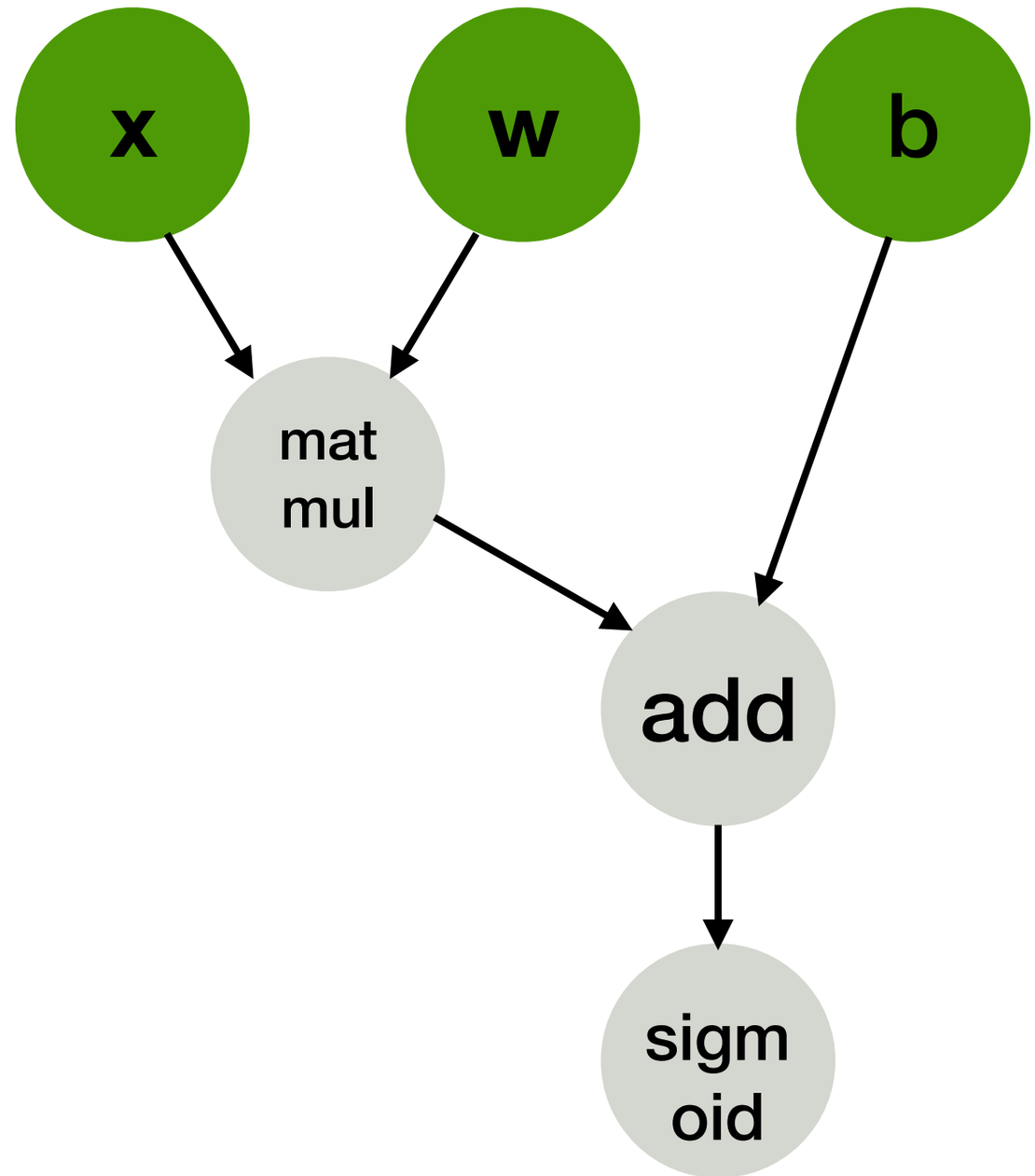
- $\mathbf{w}^\top \mathbf{x} + b$



Computation Graph

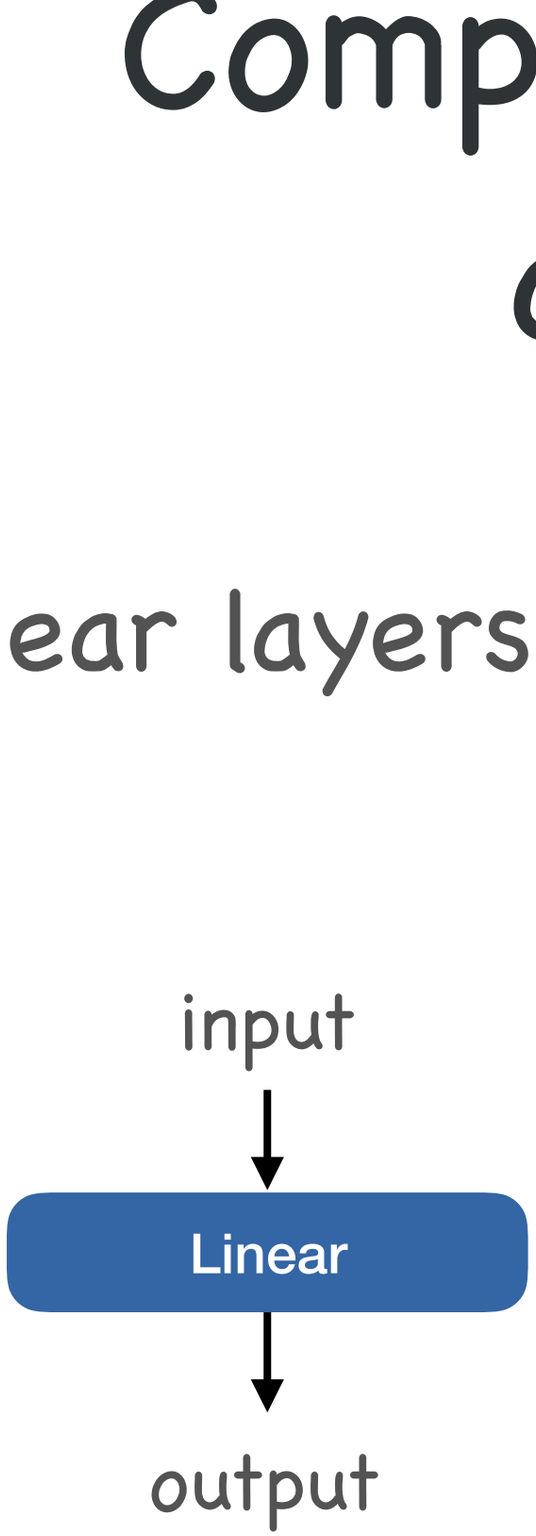
- Logistic regression:

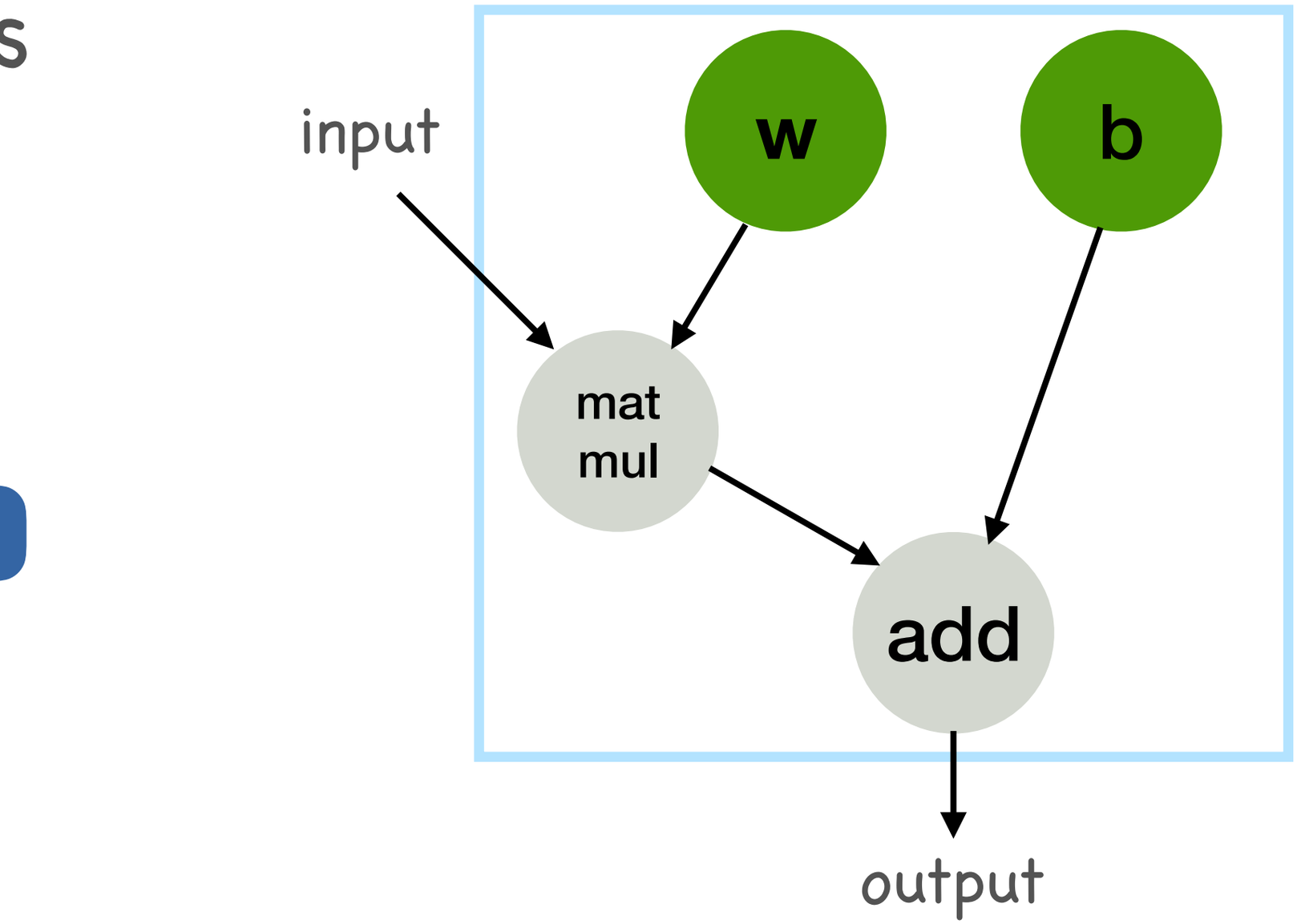
- $\sigma(\mathbf{w}^\top \mathbf{x} + b)$



Computation Graph - abstraction

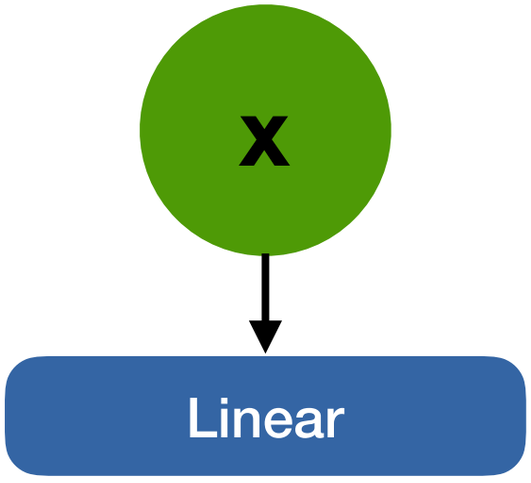
- Linear layers



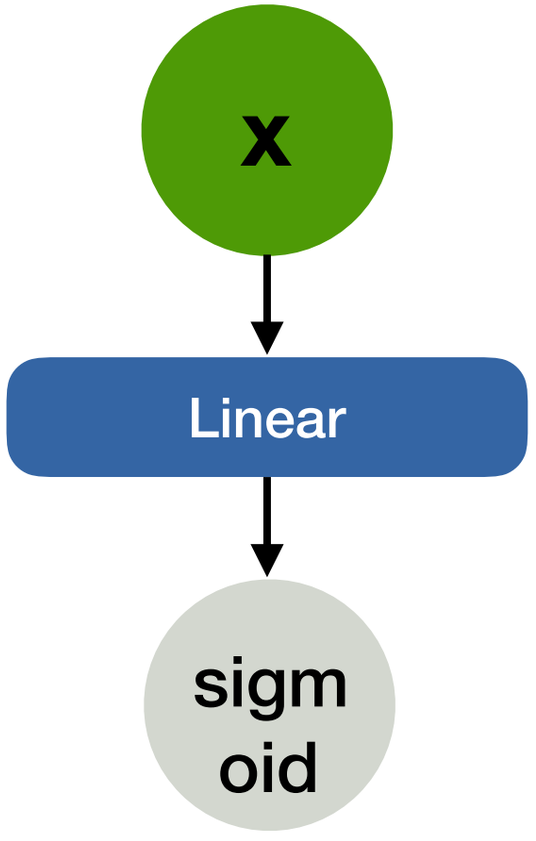


Computation Graph - abstraction

Linear regression:



Logistic regression:



08

January 23, 2024

[ ]:
```
%pylab inline
import torch
from torch.nn.parameter import Parameter
```

[ ]:
```
x = torch.rand([1000,2])
x_in_circle = ((x**2).sum(1) < 1)

def accuracy(pred_label):
    return (pred_label==x_in_circle).float().mean()

def show(pred_label):
    scatter(*x.numpy().T, c=pred_label.numpy())
    axis('equal')

def loss(prediction):
    return -(x_in_circle.float()     * prediction.log() +
             (1-x_in_circle.float()) * (1-prediction).log() ).mean()

show(x_in_circle)
```

[ ]:
```
class Linear(torch.nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.w = Parameter(torch.ones(input_dim))
        self.b = Parameter(-torch.ones(1))

    def forward(self, x):
        return (x * self.w[None,:]).sum(dim=1) + self.b

class LinearClassifier(torch.nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.linear = Linear(input_dim)

    def forward(self, x):
        logit = self.linear(x)
        return 1/(1+(-logit).exp())
```

```
classifier = LinearClassifier(2)

show(classifier(x).detach() > 0.5)
```

[ ]:
```
list(classifier.parameters())
```

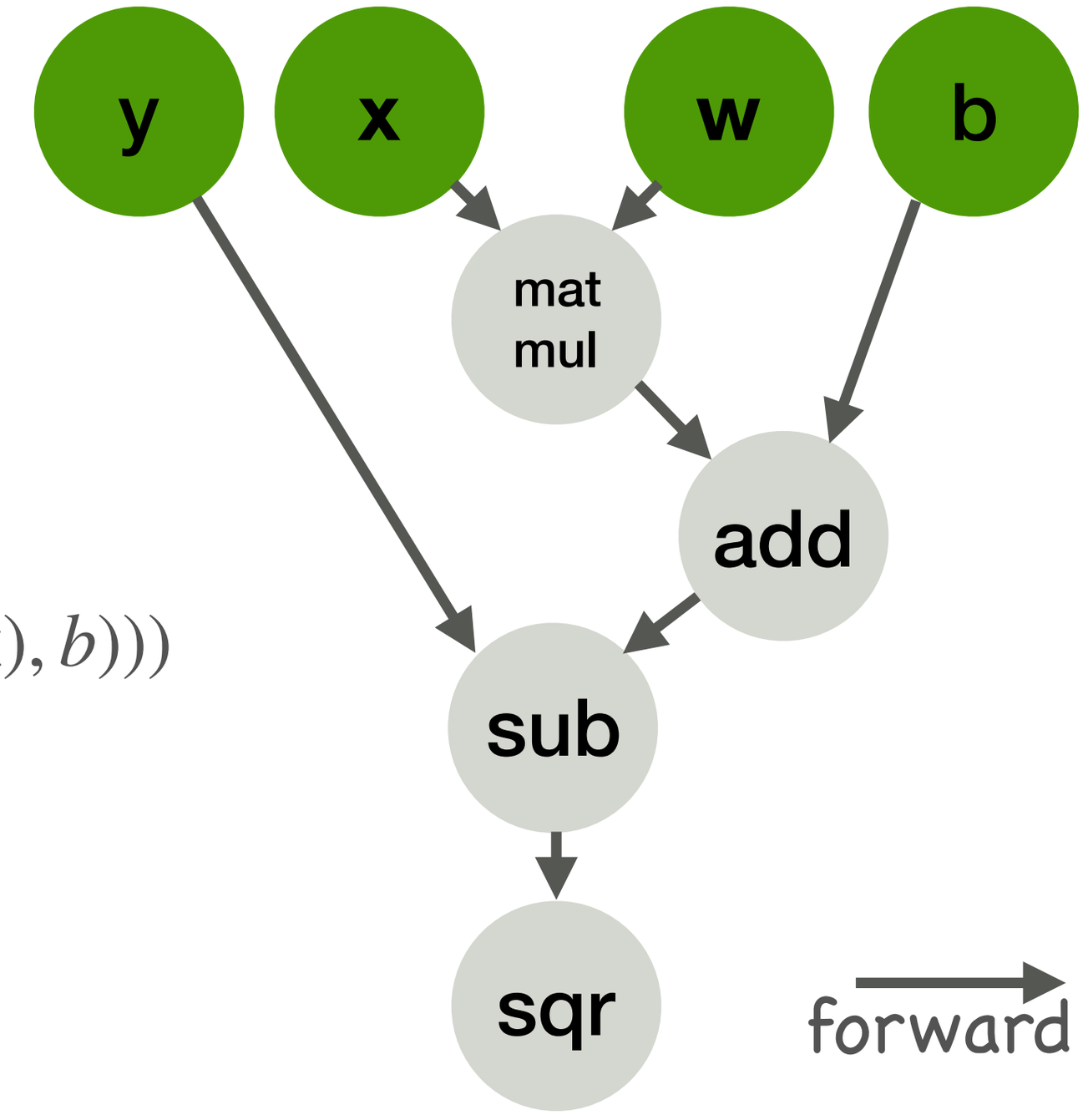
[ ]:
```

```

Gradients on computation graphs

© 2019 Philipp Krähenbühl and Chao-Yuan Wu

Linear regression

- $\ell(\mathbf{w}, b) = (y - \mathbf{w}^\top \mathbf{x} + b)^2$
- sqr(sub(y, add(matmul($\mathbf{w}$, $\mathbf{x}$), b)))



Gradient of linear regression

- $$\frac{d\ell(\mathbf{w}, b)}{d\mathbf{w}} = \frac{d(y - \mathbf{w}^\top \mathbf{x} + b)^2}{d\mathbf{w}}$$

- $$\frac{d}{d\mathbf{w}}\text{sqr}(\text{sub}(y, \text{add}(\text{matmul}(\mathbf{w}, \mathbf{x}), b)))$$

$$= \text{sqr}'(\text{sub}(y, \text{add}(\text{matmul}(\mathbf{w}, \mathbf{x}), b)))$$

$$\frac{d}{d\mathbf{w}}\text{sub}(y, \text{add}(\text{matmul}(\mathbf{w}, \mathbf{x}), b))$$

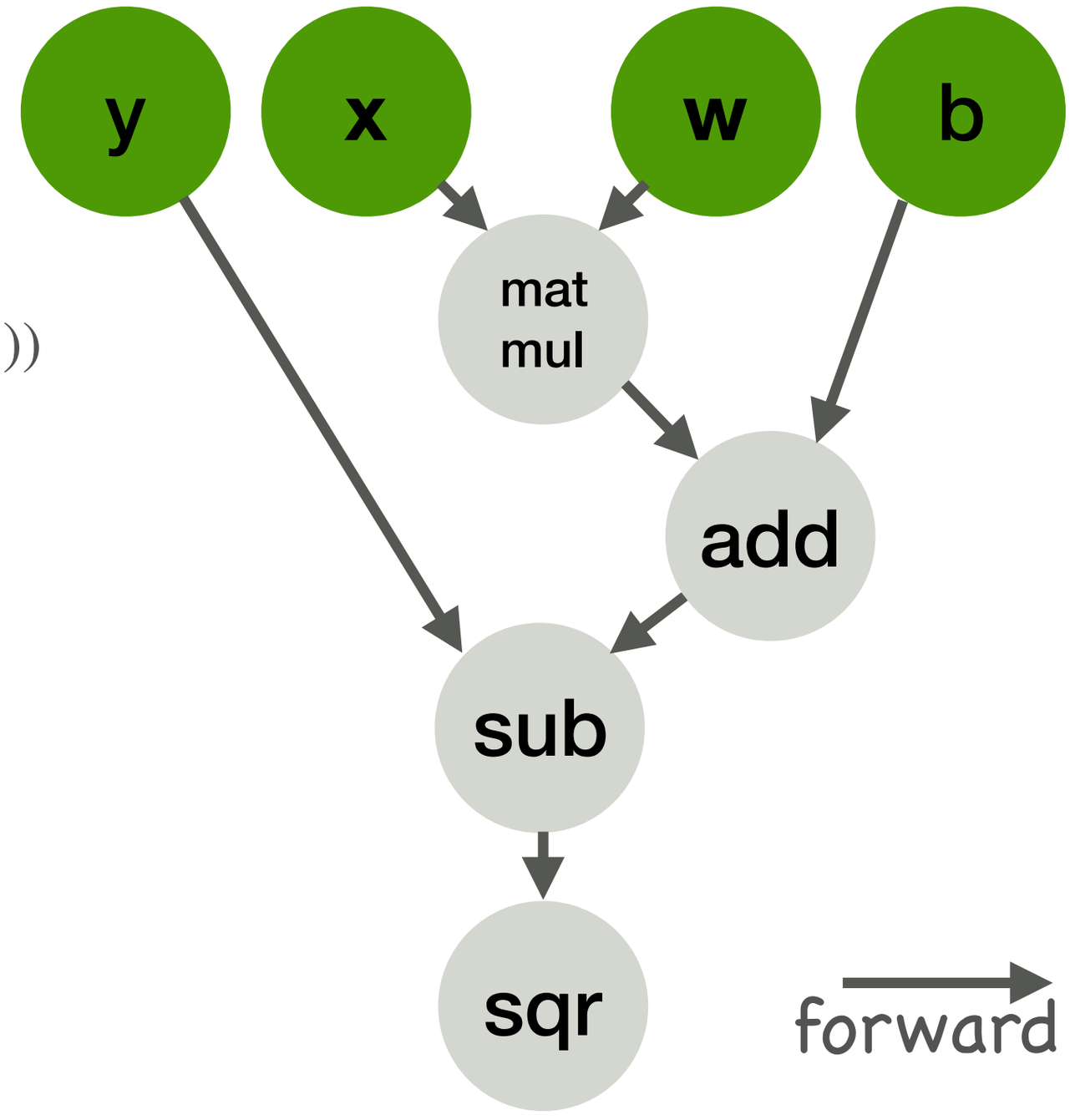
$$= \ldots$$

$$= \text{sqr}'(\text{sub}(y, \text{add}(\text{matmul}(\mathbf{w}, \mathbf{x}), b)))$$

$$\text{sub}'(y, \text{add}(\text{matmul}(\mathbf{w}, \mathbf{x}), b))$$

$$\text{add}'(\text{matmul}(\mathbf{w}, \mathbf{x}), b)$$

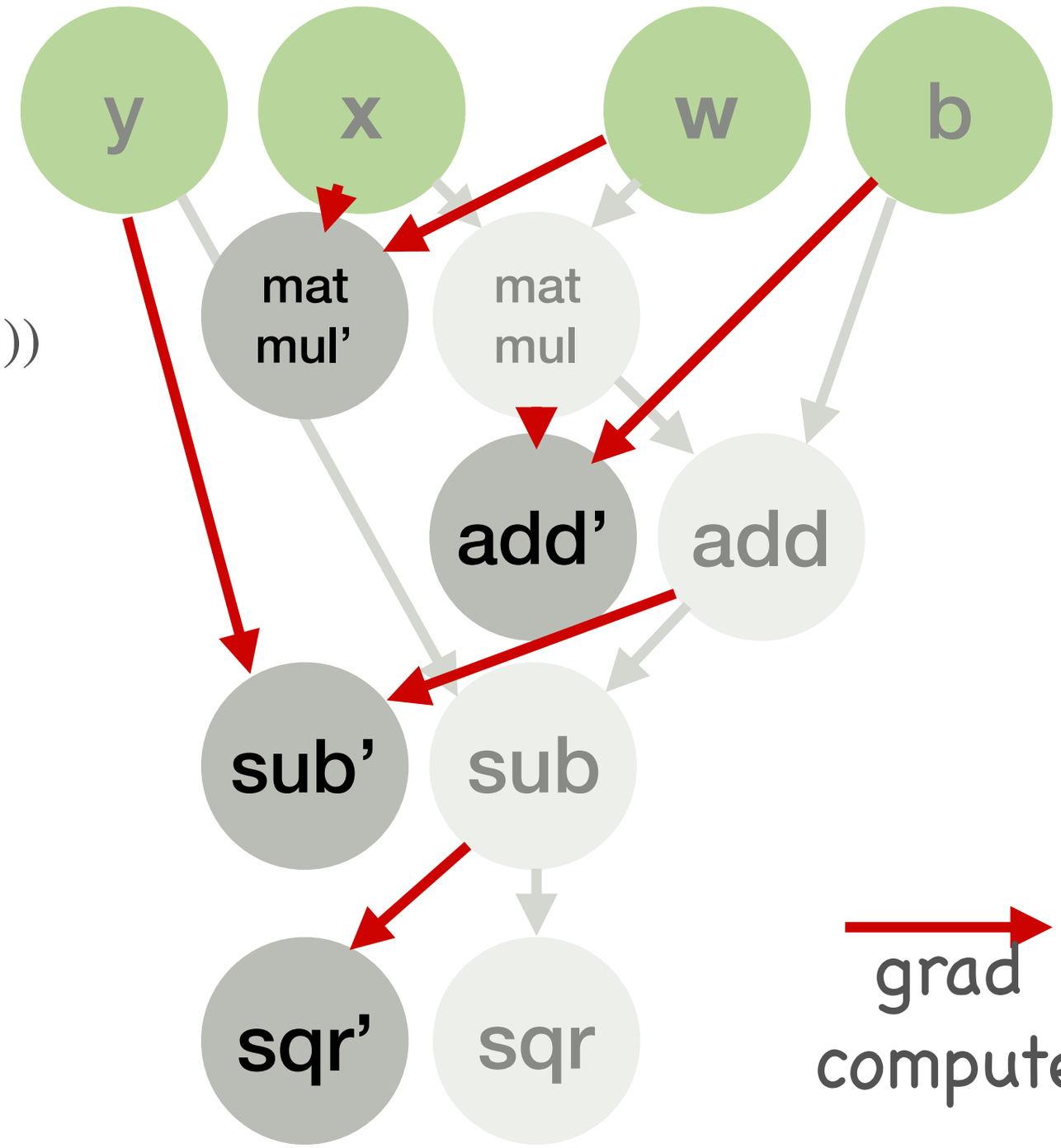
$$\frac{d}{d\mathbf{w}}\text{matmul}(\mathbf{w}, \mathbf{x})$$



Gradient of linear regression

- $$\frac{d\ell(\mathbf{w}, b)}{d\mathbf{w}} = \frac{d(y - \mathbf{w}^\top \mathbf{x} + b)^2}{d\mathbf{w}}$$

- $$\frac{d}{d\mathbf{w}}\text{sqr}(\text{sub}(y, \text{add}(\text{matmul}(\mathbf{w}, \mathbf{x}), b)))$$
$$= \text{sqr}'(\text{sub}(y, \text{add}(\text{matmul}(\mathbf{w}, \mathbf{x}), b)))$$
$$\frac{d}{d\mathbf{w}}\text{sub}(y, \text{add}(\text{matmul}(\mathbf{w}, \mathbf{x}), b))$$
$$= \ldots$$
$$= \text{sqr}'(\text{sub}(y, \text{add}(\text{matmul}(\mathbf{w}, \mathbf{x}), b)))$$
$$\text{sub}'(y, \text{add}(\text{matmul}(\mathbf{w}, \mathbf{x}), b))$$
$$\text{add}'(\text{matmul}(\mathbf{w}, \mathbf{x}), b)$$
$$\frac{d}{d\mathbf{w}}\text{matmul}(\mathbf{w}, \mathbf{x})$$



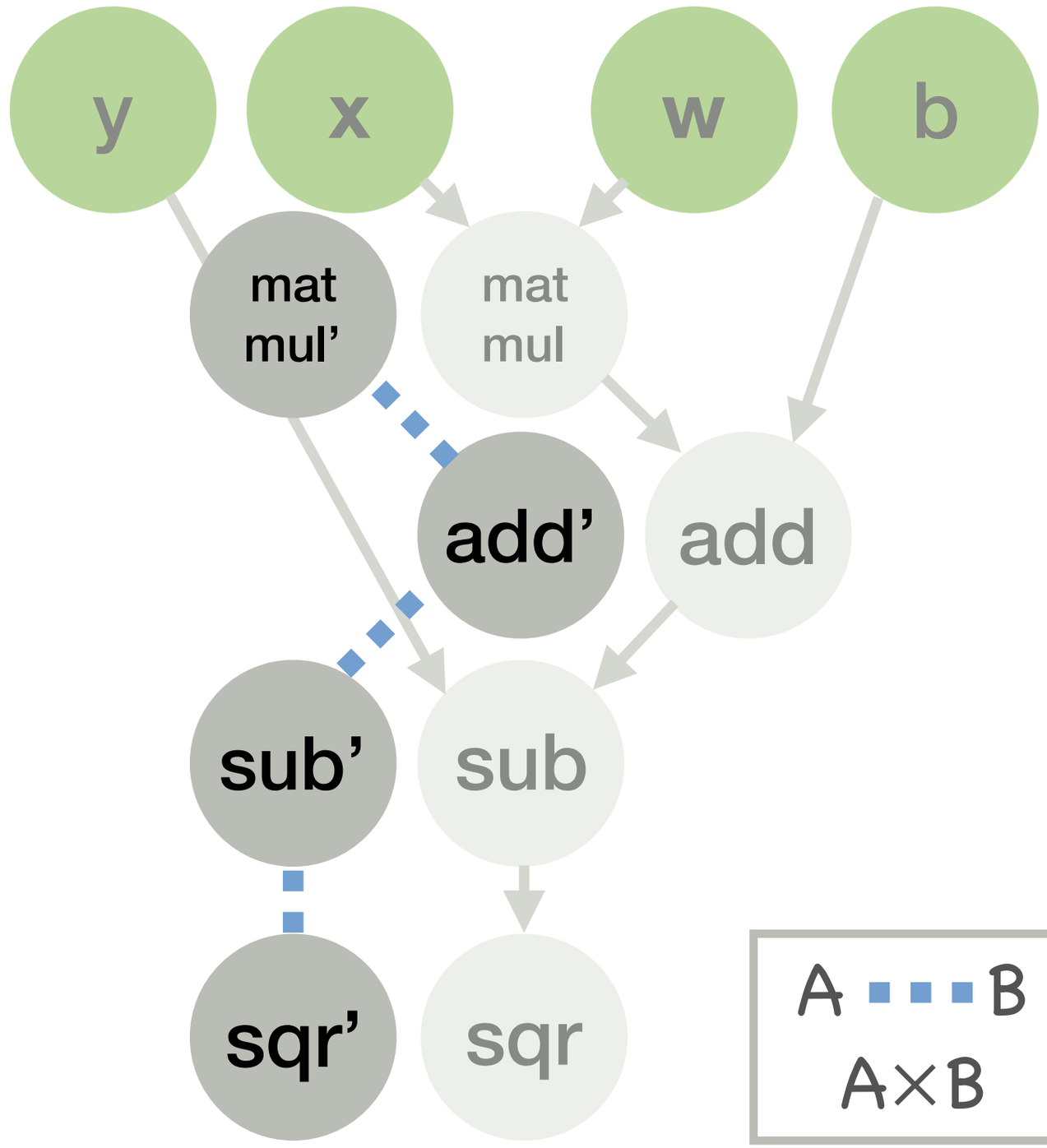
Gradient of linear regression

$$\mathrm{sqr}'(\mathrm{sub}(y, \mathrm{add}(\mathrm{matmul}(\mathbf{w}, \mathbf{x}), b)))$$

$$\mathrm{sub}'(y, \mathrm{add}(\mathrm{matmul}(\mathbf{w}, \mathbf{x}), b))$$

$$\mathrm{add}'(\mathrm{matmul}(\mathbf{w}, \mathbf{x}), b)$$

$$\frac{d}{d\mathbf{w}}\mathrm{matmul}(\mathbf{w}, \mathbf{x})$$



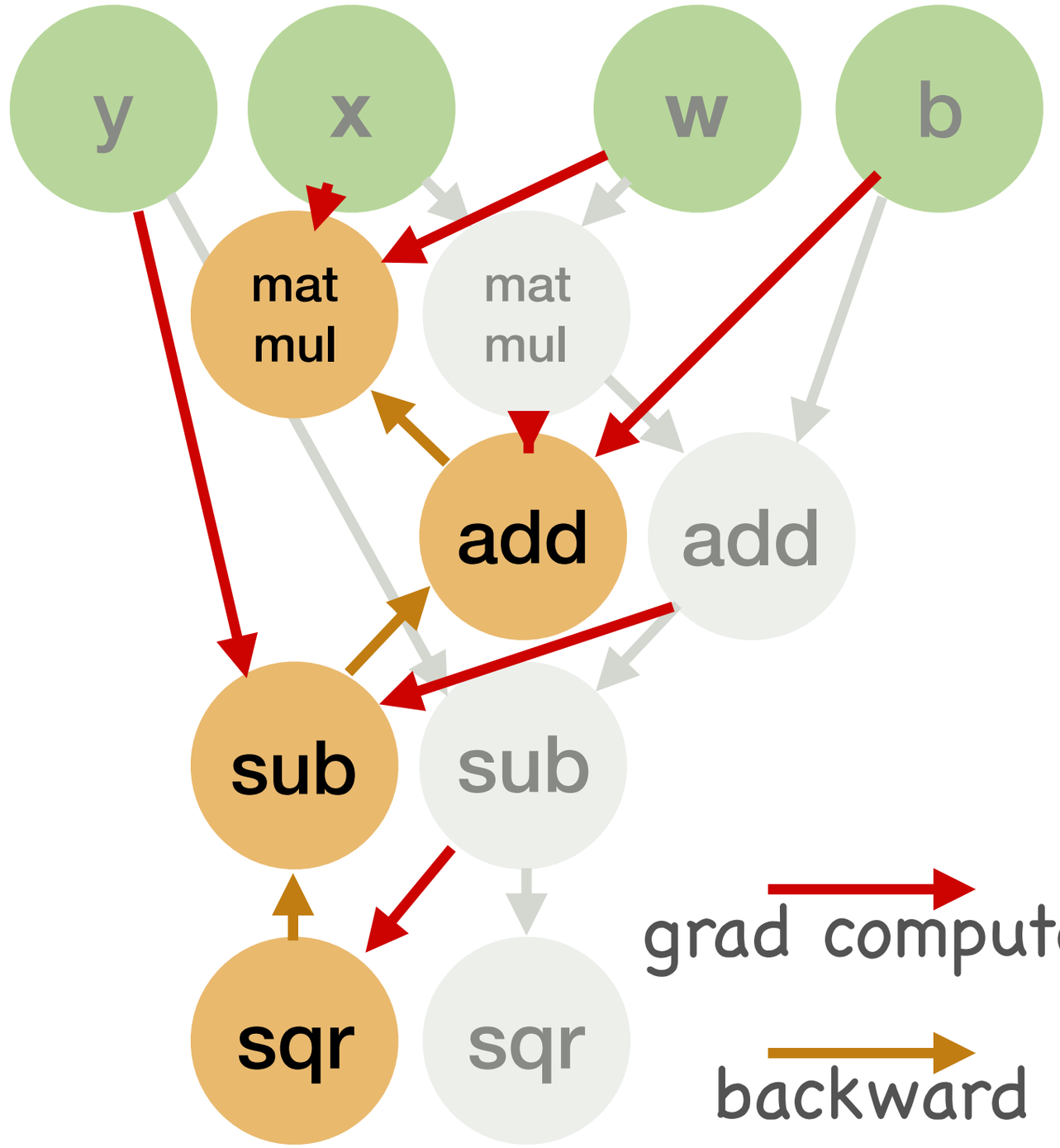
Gradient of linear regression

$$\mathrm{sqr}'(\mathrm{sub}(y, \mathrm{add}(\mathrm{matmul}(\mathbf{w}, \mathbf{x}), b)))$$

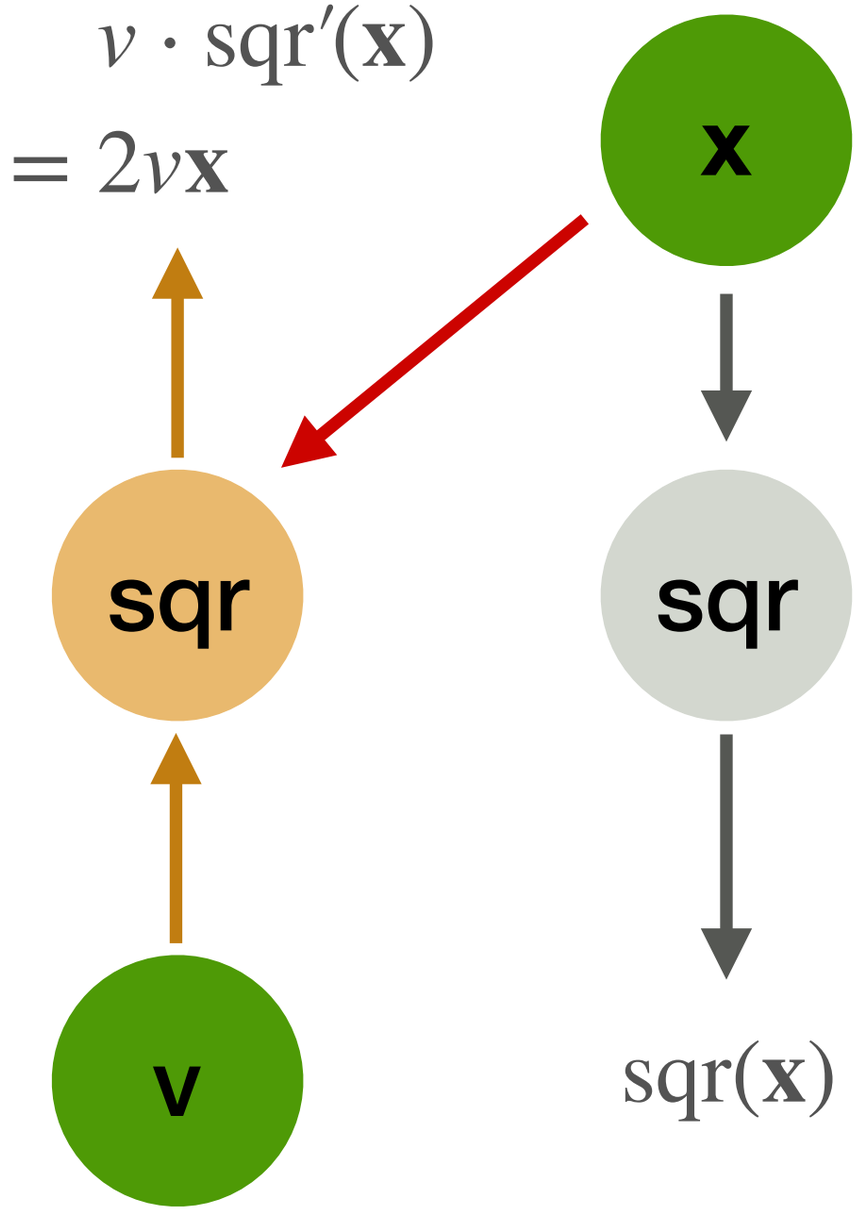
$$\mathrm{sub}'(y, \mathrm{add}(\mathrm{matmul}(\mathbf{w}, \mathbf{x}), b))$$

$$\mathrm{add}'(\mathrm{matmul}(\mathbf{w}, \mathbf{x}), b)$$

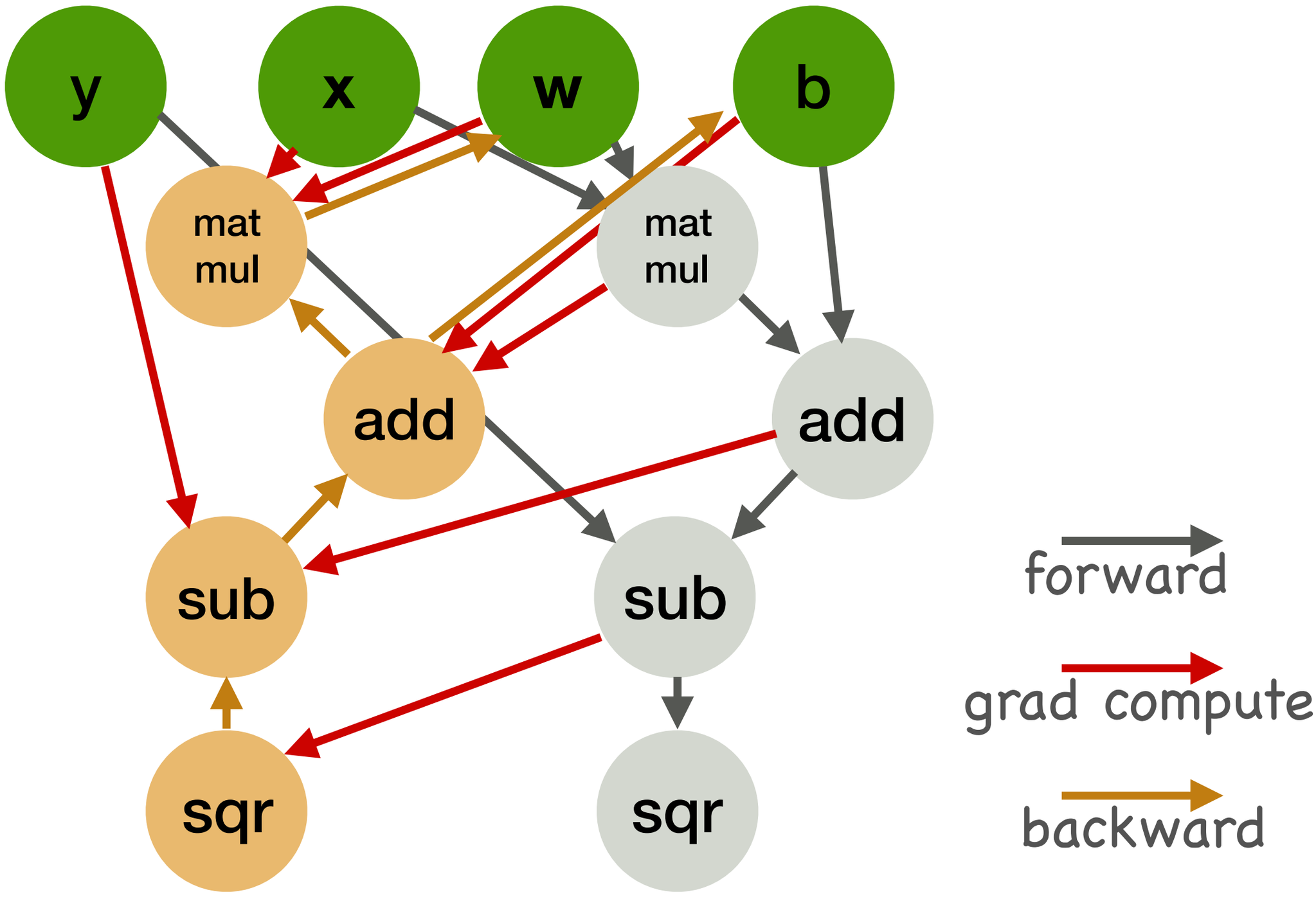
$$\frac{d}{d\mathbf{w}}\mathrm{matmul}(\mathbf{w}, \mathbf{x})$$



Gradient of linear regression



Gradient of linear regression



10

January 23, 2024

[ ]:
```
%pylab inline
import torch
from torch.nn.parameter import Parameter
```

[ ]:
```
x = torch.rand([1000,2])
x_in_circle = ((x**2).sum(1) < 1)

def accuracy(pred_label):
    return (pred_label==x_in_circle).float().mean()

def show(pred_label):
    scatter(*x.numpy().T, c=pred_label.numpy())
    axis('equal')

def loss(prediction):
    return -(x_in_circle.float()     * (prediction+1e-10).log() +
             (1-x_in_circle.float()) * (1-prediction+1e-10).log() ).mean()

class Linear(torch.nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.w = Parameter(torch.zeros(input_dim))
        self.b = Parameter(-torch.zeros(1))

    def forward(self, x):
        return (x * self.w[None,:]).sum(dim=1) + self.b

class LinearClassifier(torch.nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.linear = Linear(input_dim)

    def forward(self, x):
        logit = self.linear(x)
        return 1/(1+(-logit).exp())

show(x_in_circle)
```

[ ]:
```
import torch.utils.tensorboard as tb
%load_ext tensorboard
import tempfile
log_dir = tempfile.mkdtemp()
%tensorboard --logdir {log_dir} --reload_interval 1
```

[ ]:
```
logger = tb.SummaryWriter(log_dir+'/linear1')

classifier = LinearClassifier(2)

for iteration in range(10000):
    p_y = classifier(x)
    pred_y = p_y > 0.5
    l = loss(p_y)

    logger.add_scalar("loss", l, global_step=iteration)
    logger.add_scalar("accuracy", accuracy(pred_y), global_step=iteration)

    if iteration % 100 == 0:
        fig = figure()
        show(pred_y)
        logger.add_figure('pred_y', fig, global_step=iteration)
        del fig

    l.backward()
    for p in classifier.parameters():
        p.data[:] -= 0.5 * p.grad
        p.grad.zero_()

show(pred_y)
```

[ ]:
```
class NonLinearClassifier(torch.nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.linear1 = torch.nn.Linear(input_dim, 100)
        torch.nn.init.normal_(self.linear1.weight, std=0.01)
        torch.nn.init.normal_(self.linear1.bias, std=0.01)
        self.linear2 = Linear(100)

    def forward(self, x):
        logit = self.linear2( torch.relu(self.linear1(x)) )
        return 1/(1+(-logit).exp())


classifier = NonLinearClassifier(2)
show(classifier(x).detach() > 0.5)
```

[ ]:
```
logger = tb.SummaryWriter(log_dir+'/nonlinear1')

classifier = NonLinearClassifier(2)

for iteration in range(10000):
    p_y = classifier(x)
    pred_y = p_y > 0.5
    l = loss(p_y)

    logger.add_scalar("loss", l, global_step=iteration)
    logger.add_scalar("accuracy", accuracy(pred_y), global_step=iteration)

    if iteration % 100 == 0:
        fig = figure()
        show(pred_y)
        logger.add_figure('pred_y', fig, global_step=iteration)
        del fig

    l.backward()
    for p in classifier.parameters():
        p.data[:] -= 0.5 * p.grad
        p.grad.zero_()

show(pred_y)
```

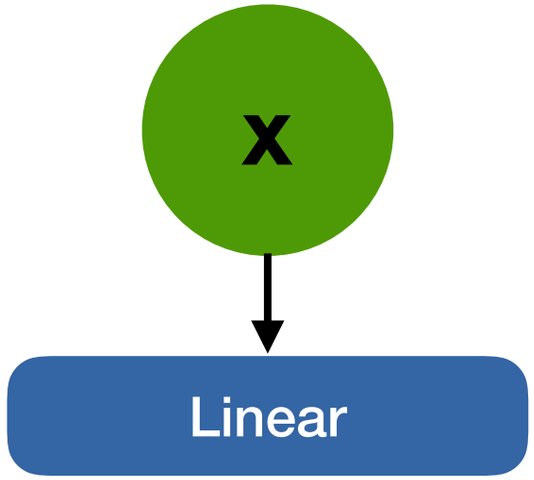
[ ]:
```

```

Summary

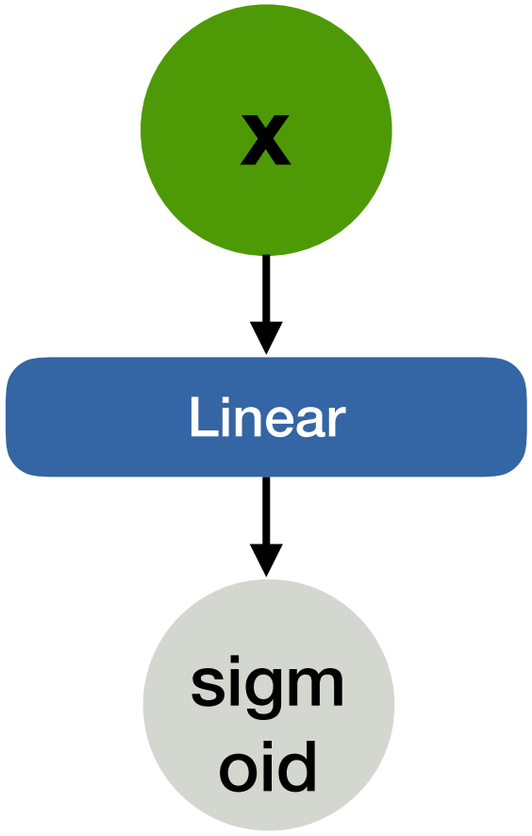
© 2019 Philipp Krähenbühl and Chao-Yuan Wu

Linear regression/classifiers

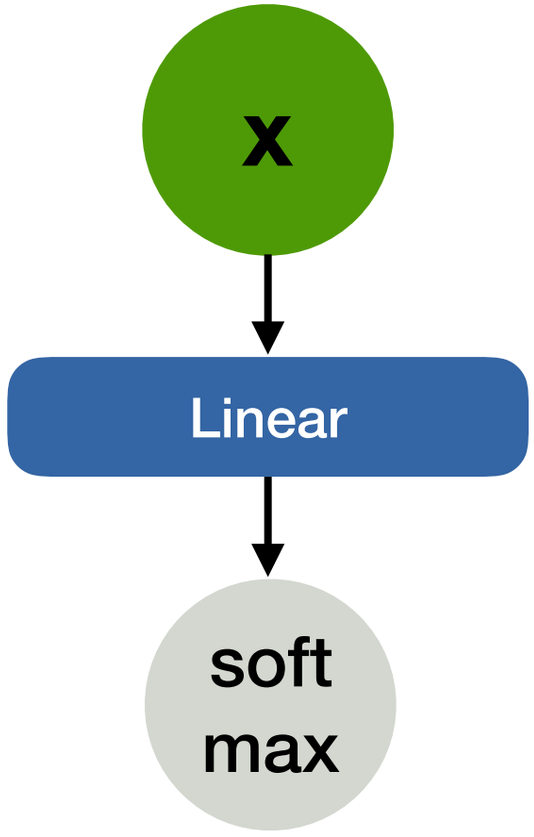
Linear regression



Logistic regression



Multinomial logistic regression



Compute graphs and back-propagation