Introduction to Machine Learning: Discriminative Models

Farzad Kamalabadi

University of Illinois Urbana-Champaign

Heliophysics Summer School, 2026

Outline of this lecture

- Introduction to pattern recognition and classification
- Linear regression
- Logistic regression
- Support Vector Machines
- Multiclass regression and SVM
- Deep Learning

Reading Material

- K. Murphy; Machine Learning: A Probabilistic Perspective

History (1930s to 1950s):

- Neurology research showed brain as electrical network of neurons
- Cybernetics (Norbert Wiener) described control and stability in electrical networks
- Information theory (Claude Shannon) described digital signals
- Theory of computation (Alan Turing) described digital nature of computation
- Electronic brain/neural net (Walter Pitts and Warren McCulloch)

Marvin Minsky (1955):

A Proposal for the

DARTMOUTH SUMMER RESEARCH PROJECT ON ARTIFICIAL INTELLIGENCE

June 17 - Aug. 16

We propose that a 2 month, 10 man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.

Mentioned aspects:

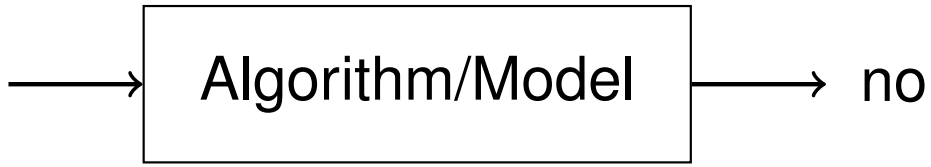
- Automatic computers
- Programming a computer with language
- Neuron nets
- Theory on complexity of calculation
- Self-improvement
- Abstractions
- Randomness and Creativity

Where pattern recognition/AI shows up ‘recently’:

- Chess play (Deep Blue, IBM, 1996, 1997)
- Jeopardy (Watson, IBM, 2011)
- Playing Atari games (DQN, Deepmind, 2015)
- Game of Go (AlphaGo, Deepmind, 2016)
- Autonomous Driving
- Generative Pre-trained Transformer (GPT), OpenAI, 2022; GPT5, 2025)
- DeepMind’s latest weather prediction model, GenCast, outperforms traditional physics-based models and even other AI models, in terms of accuracy and speed.

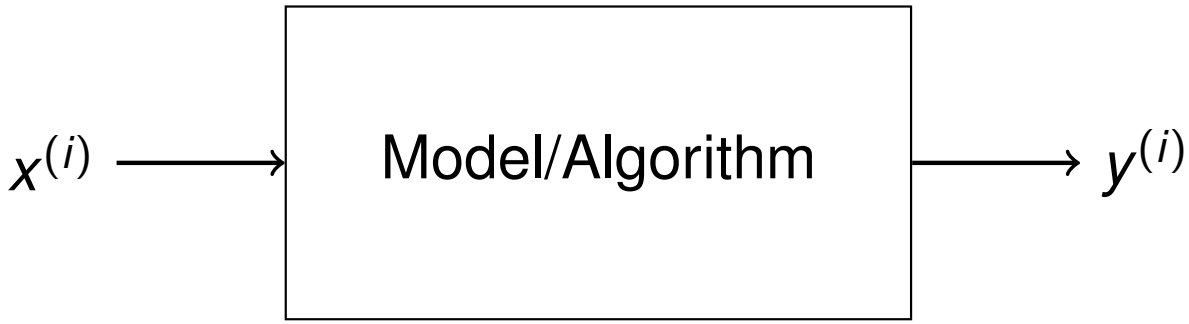
Example:

Let's get a computer to recognize whether there is a cat in the image.



Formulation of pattern recognition for this lecture (discriminative models):

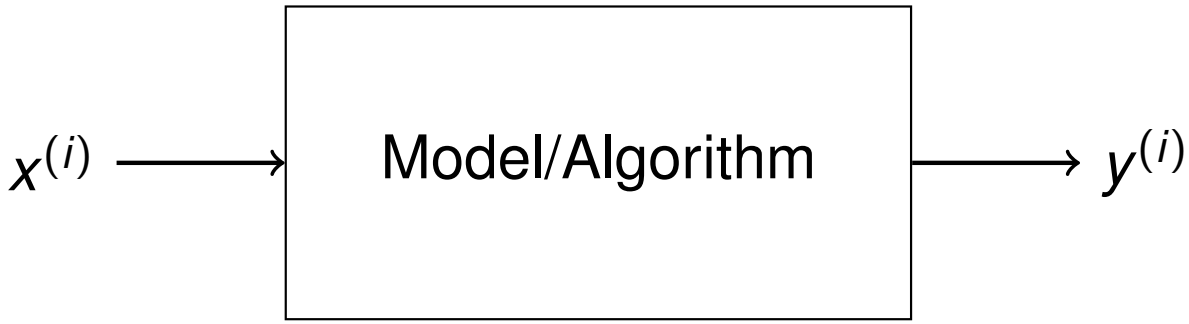
- input data/value/vector: $x^{(i)}$
- label/output: $y^{(i)}$



What do we call this process?

- Inference
- Prediction

What is learning?



- Model/Algorithm depends on parameters $\boldsymbol{w}$
- Learning/Fitting of parameters $\boldsymbol{w}$
- Based on dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$

How could we recognize a cat? What should the algorithm do?

How is an image represented in a computer?

69	69	70	70	71	72	72
79	80	80	81	82	83	83
87	88	88	89	90	91	91
95	96	96	97	98	99	99
108	108	109	110	110	111	112
119	119	120	120	121	122	122
130	131	131	132	133	134	134
144	144	145	145	146	147	148
158	158	159	160	162	163	164
172	172	173	174	175	176	177
187	188	189	190	191	192	193
198	198	199	200	202	203	204
207	207	208	209	210	212	212

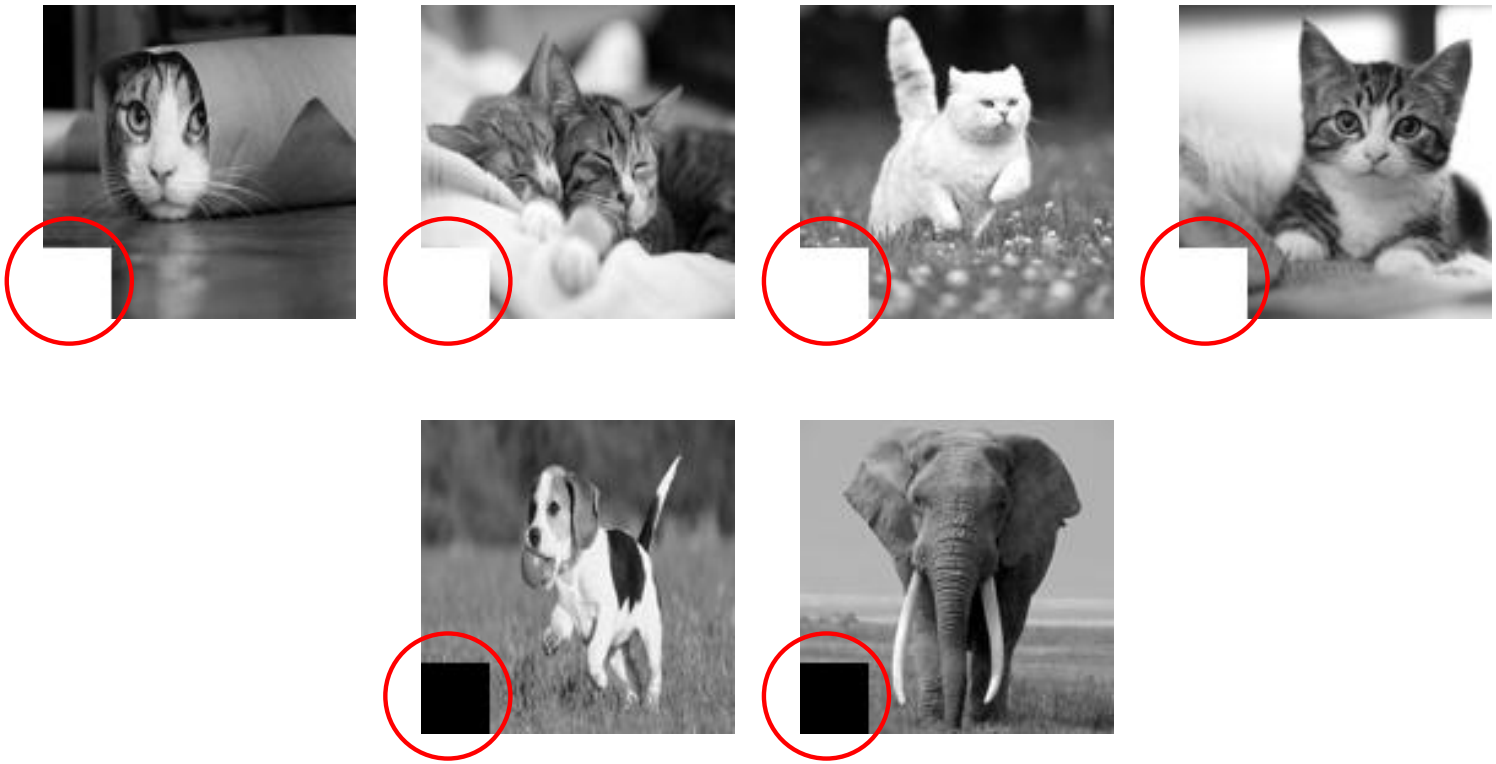
Given that we know how a computer ‘sees’ an image, how can it recognize cats?



Still very hard to describe what a cat looks like.

Let’s look at tons of examples (Dataset).

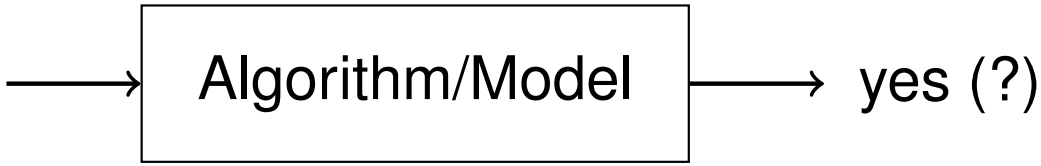
Dataset (Thousands of examples):



Instead of asking to recognize cats in general, let's recognize cats in this dataset

Algorithm: if bottom left corner is black (0) say no, otherwise say yes

Algorithm: if bottom left corner is black (0) say no, otherwise say yes



Works perfect on our dataset. :)
But does not generalize to other data. :(

Conclusion:

We designed a simple “classifier” that works on this dataset but doesn’t work on real data.

To our rescue:

Machine learning has found mechanisms to search for mappings which **generalize**.

Scope of this lecture:

In this lecture we talk about algorithms and models. A detailed treatment of **generalization** is beyond our scope.

Categorization of machine learning/pattern recognition algorithms according to

- Available annotated data (supervised vs. unsupervised)
- Complexity of model (linear vs. non-linear)
- Structure of output (independent vs. structured)
- Modeling of data ($x^{(i)}$) or label ($y^{(i)}$) (generative vs. discriminative)

Classification Framework: Formalism

Main Components

- an *input* (also called *observation* or *data point*), denoted by x
- a **discrete** *output* (also called *label* or *class*), denoted by y
- a *decision function* (also called *classification rule*), $y = f(x)$

The main problem in Pattern Recognition/Machine Learning is to construct the *decision function*.

Linear Regression

Notation:

- Vector: $\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n$
- Matrix: $\mathbf{X} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,m} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,m} \end{bmatrix} \in \mathbb{R}^{n \times m}$
- Norm: $\|x^{(1)} - x^{(2)}\|_2^2 = \sum_{i=1}^{n} (x_i^{(1)} - x_i^{(2)})^2$ distance between two points in n dimensions
- Transpose: $\mathbf{X}^T = \begin{bmatrix} x_{1,1} & \cdots & x_{n,1} \\ \vdots & \ddots & \vdots \\ x_{1,m} & \cdots & x_{m,n} \end{bmatrix} \in \mathbb{R}^{m \times n}$

$$\boldsymbol{x}^T = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix} \in \mathbb{R}^{1 \times n}$$

- Matrix multiplication: $\mathbf{X}^T \boldsymbol{x}$ or $\mathbf{X}\boldsymbol{x}$?

Discrete Probability: $y \in \{1, \ldots, 6\}$

- Discrete probability distribution: $p(Y = y) \in [0, 1]$ with $\sum_{y \in \{1,\ldots,6\}} p(Y = y) = 1$
- Abbreviation: $p(Y = y) = p(y) \in [0, 1]$
- Expectation: $\mathbb{E}_{p(y)}[f(y)] = \sum_{y \in \{1,\ldots,6\}} p(y) f(y)$

Continuous probability: $y \in \mathbb{R}$

- $p(Y = 1) = 0$
- Probability density function: $p(y) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-1}{2\sigma^2}(y-\mu)^2\right)$
- Mean: $\mathbb{E}_{p(y)}[y] = \int_{-\infty}^{\infty} yp(y)dy = \mu$
- Variance: $\mathbb{E}_{p(y)}[(y-\mu)^2] = \sigma^2$

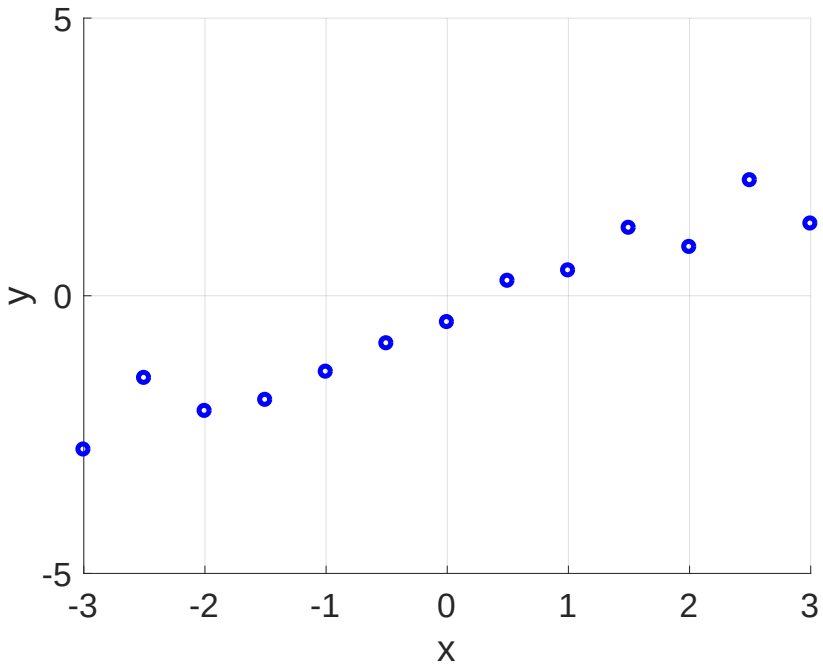
Multivariate continuous probability: $\boldsymbol{y} \in \mathbb{R}^n$ $\mu \in \mathbb{R}^n$

- n-dimensional density:
  $p(\boldsymbol{y}) = p(y_1, \ldots, y_n) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} \exp\left(\frac{-1}{2}(\boldsymbol{y}-\mu)^T\Sigma^{-1}(\boldsymbol{y}-\mu)\right)$
- Covariance matrix: Σ

Multivariate calculus: $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{w} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{n \times n}$

- Multivariate function: $f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$
- Derivative: $\frac{\partial f}{\partial \boldsymbol{x}} = \boldsymbol{w}$ (e.g., Eq. (69))
- Multivariate function: $f(\boldsymbol{x}) = \boldsymbol{x}^T \mathbf{A} \boldsymbol{x}$
- Derivative: $\frac{\partial f}{\partial \boldsymbol{x}} = (\mathbf{A} + \mathbf{A}^T)\boldsymbol{x}$ (e.g., Eq. (78) or Eq. (81))

Linear Regression - The Problem:



Given outcomes $y^{(i)} \in \mathbb{R}$ for covariates $x^{(i)} \in \mathbb{R}$,

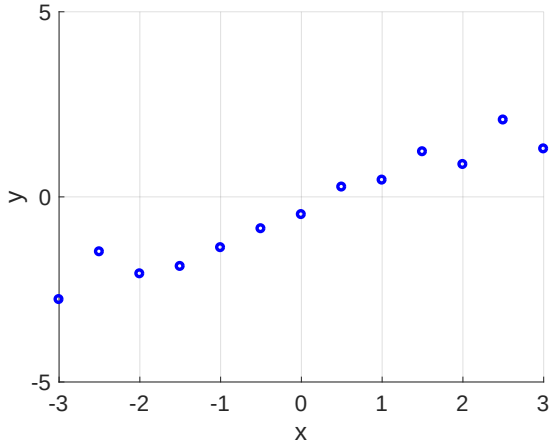
what is/are the underlying system/model/model parameters?

Let's assume a linear model with parameters $w_1 \in \mathbb{R}$ and $w_2 \in \mathbb{R}$

$$y = w_1 \cdot x + w_2$$

Given a dataset of N pairs (x, y):

$$\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$$



How do we find the parameters w_1, w_2?

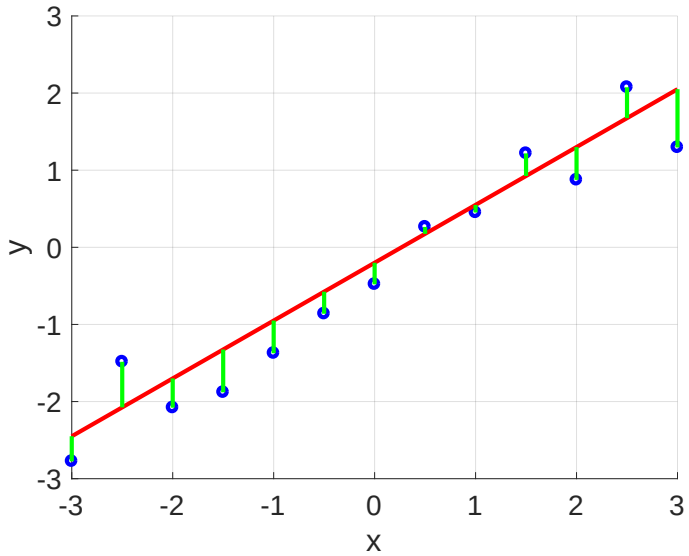
Assuming model

$$y = w_1 \cdot x + w_2$$

Find parameters w_1, w_2 such that the squared error is small

$$\arg\min_{w_1, w_2} \frac{1}{2} \sum_{i=1}^{N} \left(y^{(i)} - w_1 \cdot x^{(i)} - w_2 \right)^2$$

What exactly is the error?



Program:

$$\arg\min_{w_1, w_2} \frac{1}{2} \sum_{i=1}^{N} \left(y^{(i)} - w_1 \cdot x^{(i)} - w_2 \right)^2$$

Vector notation:

$$\arg\min_{w_1, w_2} \frac{1}{2} \left\| \underbrace{\begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{bmatrix}}_{\mathbf{Y} \in \mathbb{R}^N} - \underbrace{\begin{bmatrix} x^{(1)} & 1 \\ \vdots & \vdots \\ x^{(N)} & 1 \end{bmatrix}}_{\mathbf{X}^\top \in \mathbb{R}^{N \times 2}} \cdot \underbrace{\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}}_{\mathbf{w} \in \mathbb{R}^2} \right\|_2^2$$

Program:

$$\arg\min_{\boldsymbol{w}} \underbrace{\frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{X}^\top \boldsymbol{w}\|_2^2}_{\text{cost/loss function}}$$

How to solve the program:

- Take derivative w.r.t. $\boldsymbol{w}$ of cost function
- Set derivative w.r.t. $\boldsymbol{w}$ to zero
- Solve for $\boldsymbol{w}$

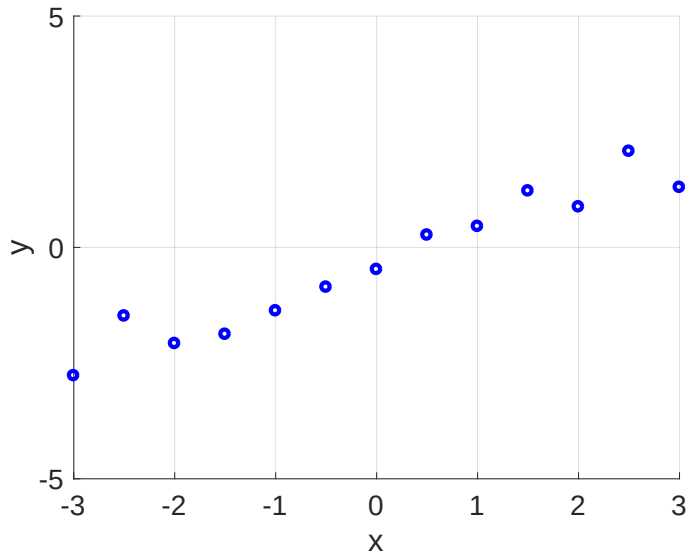
Derivative:

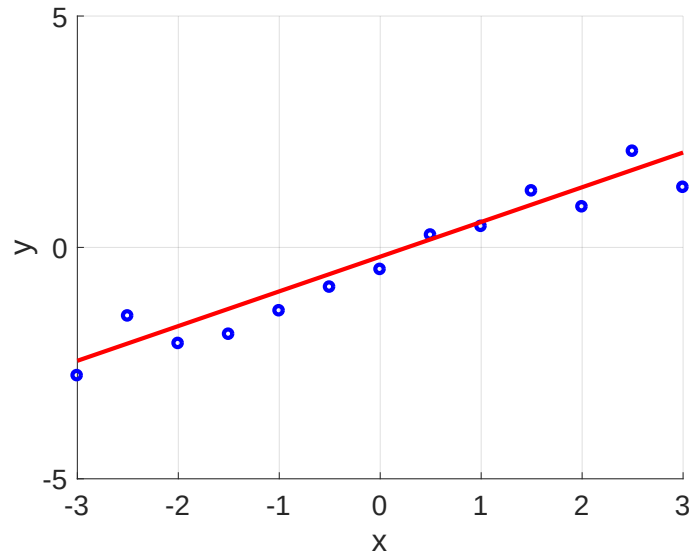
$$\boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{w}^* - \boldsymbol{X}\boldsymbol{Y} = 0$$

Solution:

$$\boldsymbol{w}^* = \left(\boldsymbol{X}\boldsymbol{X}^\top\right)^{-1} \boldsymbol{X}\boldsymbol{Y}$$

Linear regression:





Extensions:

- Higher dimensional problems ($\boldsymbol{x}^{(i)} \in \mathbb{R}^d$)
- Regularization
- Higher order polynomials

Higher dimensional problems ($\boldsymbol{x}^{(i)} \in \mathbb{R}^d$, $y^{(i)} \in \mathbb{R}$)
Model:

$$y^{(i)} = w_0 + \sum_{k=1}^{d} \boldsymbol{x}_k^{(i)} w_k$$

Program:

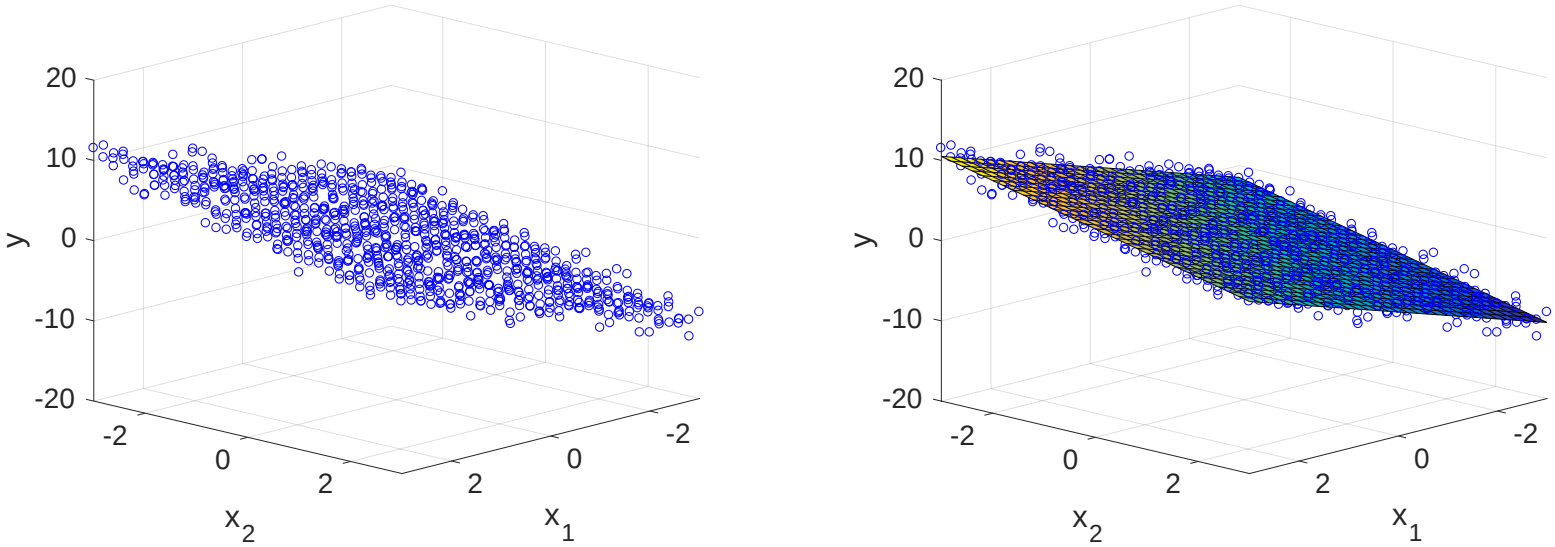
$$\arg\min_{\boldsymbol{w}} \frac{1}{2} \| \underbrace{\boldsymbol{Y}}_{\in \mathbb{R}^N} - \underbrace{\boldsymbol{X}^\top}_{\in \mathbb{R}^{N \times (d+1)}} \underbrace{\boldsymbol{w}}_{\in \mathbb{R}^{d+1}} \|_2^2$$

Solution: (obviously the same as before)

$$\boldsymbol{w}^* = \left(\boldsymbol{X}\boldsymbol{X}^\top\right)^{-1} \boldsymbol{X}\boldsymbol{Y}$$

Example:

$$\boldsymbol{w}^* = \left(\boldsymbol{X}\boldsymbol{X}^\top\right)^{-1}\boldsymbol{X}\boldsymbol{Y}$$



What if $N < d + 1$?

Regularization:

we want to make sure that the parameters are not too large

we want to make sure we can invert the matrix

Program:

$$\arg\min_{\boldsymbol{w}} \underbrace{\frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{X}^\top \boldsymbol{w}\|_2^2 + \frac{C}{2}\|\boldsymbol{w}\|_2^2}_{\text{cost function}}$$

Solution:

$$\boldsymbol{w}^* = \left(\boldsymbol{X}\boldsymbol{X}^\top + C\boldsymbol{I}\right)^{-1} \boldsymbol{X}\boldsymbol{Y}$$

Higher order polynomials ($x^{(i)} \in \mathbb{R}$, $y^{(i)} \in \mathbb{R}$)

Model:

$$y^{(i)} = w_2 \cdot \left(x^{(i)}\right)^2 + w_1 \cdot x^{(i)} + w_0$$

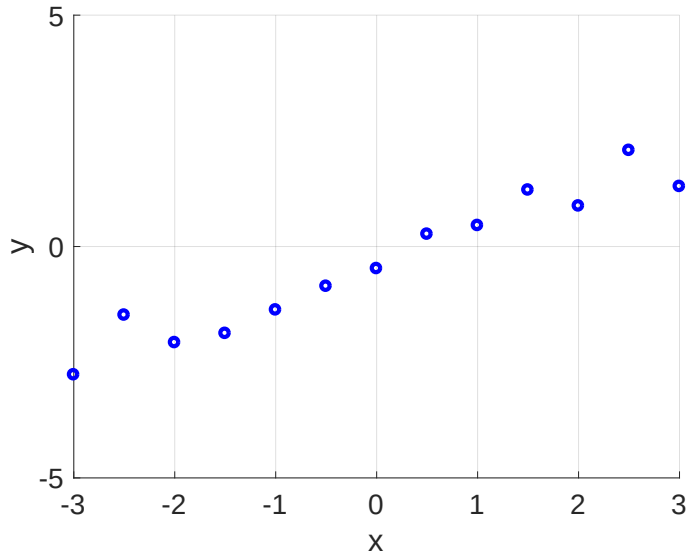
Program:

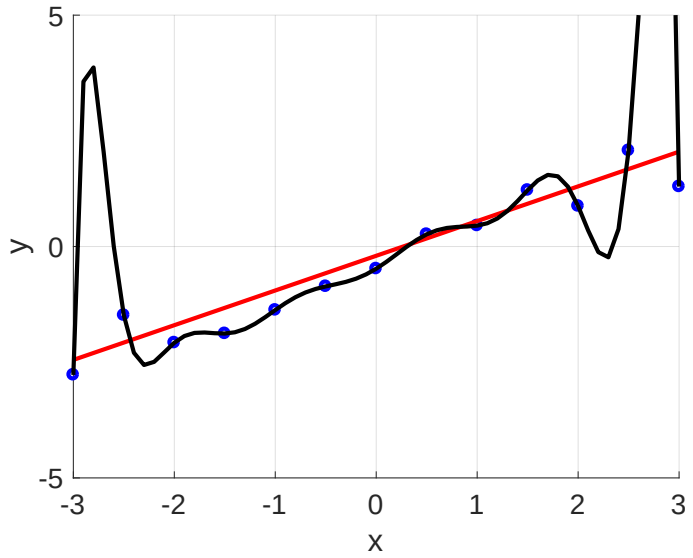
$$\arg \min_{w_0, w_1, w_2} \frac{1}{2} \left\| \underbrace{\begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{bmatrix}}_{\mathbf{Y} \in \mathbb{R}^N} - \underbrace{\begin{bmatrix} \left(x^{(1)}\right)^2 & x^{(1)} & 1 \\ \vdots & \vdots & \vdots \\ \left(x^{(N)}\right)^2 & x^{(N)} & 1 \end{bmatrix}}_{\Phi^\top \in \mathbb{R}^{N \times M}} \cdot \underbrace{\begin{bmatrix} w_2 \\ w_1 \\ w_0 \end{bmatrix}}_{\mathbf{w} \in \mathbb{R}^M} \right\|_2^2$$

Solution:

$$\boldsymbol{w}^* = \left(\Phi \Phi^\top\right)^{-1} \Phi \boldsymbol{Y}$$

Example:





Which model is more reasonable?

Generalizing all aforementioned cases:

- $x^{(i)}$ is some data (e.g., images)
- $\phi(x^{(i)}) \in \mathbb{R}^M$ is a transformation into a feature vector

Model:

$$y^{(i)} = \phi(x^{(i)})^\top \boldsymbol{w}$$

Program:

$$\arg\min_{\boldsymbol{w}} \frac{1}{2} \sum_{i=1}^{N} \left(y^{(i)} - \phi(x^{(i)})^\top \boldsymbol{w} \right)^2$$

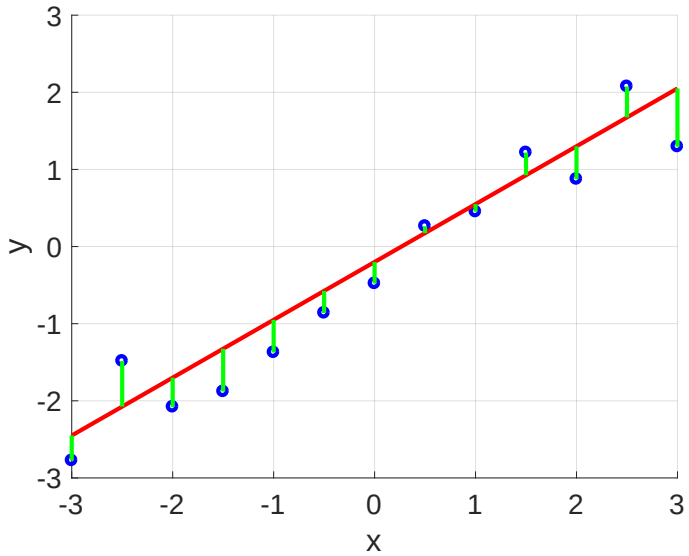
Solution:

$$\boldsymbol{w}^* = \left(\Phi \Phi^\top \right)^{-1} \Phi \boldsymbol{Y} \quad \text{where} \quad \Phi = \left[\phi(x^{(1)}), \cdots, \phi(x^{(N)}) \right] \in \mathbb{R}^{M \times N}$$

Linear regression:

- So far: Error view

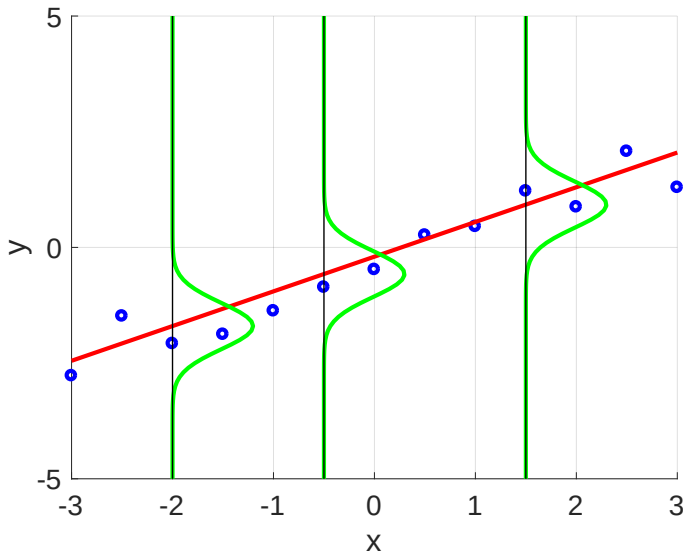
$$\left(y^{(i)} - \phi(x^{(i)})^\top \boldsymbol{w}\right)^2$$



- Alternatively: Probabilistic view

A probabilistic interpretation of linear regression:
Model: Gaussian distribution

$$p(y^{(i)}|x^{(i)}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y^{(i)} - \boldsymbol{w}^\top \phi(x^{(i)}))^2\right)$$



How to find $\boldsymbol{w}$?

Model:

$$p(y^{(i)}|x^{(i)}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y^{(i)} - \boldsymbol{w}^\top \phi(x^{(i)}))^2\right)$$

Maximize likelihood of given dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}$ assuming samples to be drawn independently from an identical distribution (i.i.d.).

$$p(\mathcal{D}) = \prod_{(x^{(i)},y^{(i)})\in\mathcal{D}} p(y^{(i)}|x^{(i)}) = \frac{1}{(2\pi\sigma^2)^{\frac{N}{2}}} \exp\left(-\frac{1}{2\sigma^2}\|\boldsymbol{Y} - \Phi^\top \boldsymbol{w}\|_2^2\right)$$

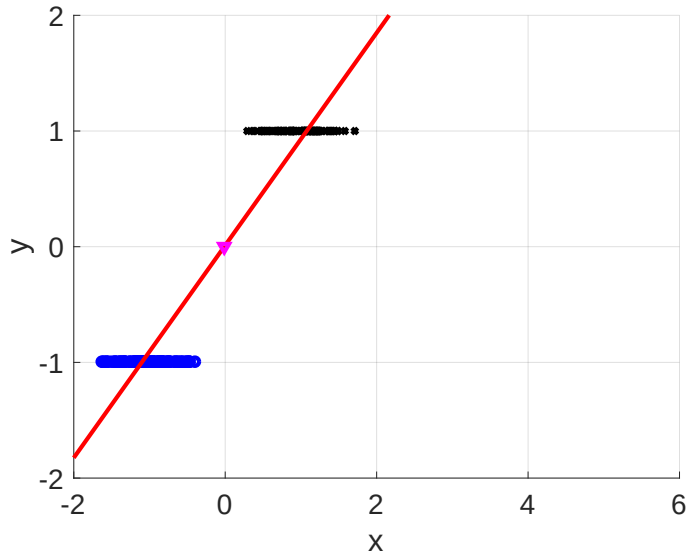
$$\arg\max_{\boldsymbol{w}} p(\mathcal{D}) = \arg\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{Y} - \Phi^\top \boldsymbol{w}\|_2^2$$

Linear regression for classification?

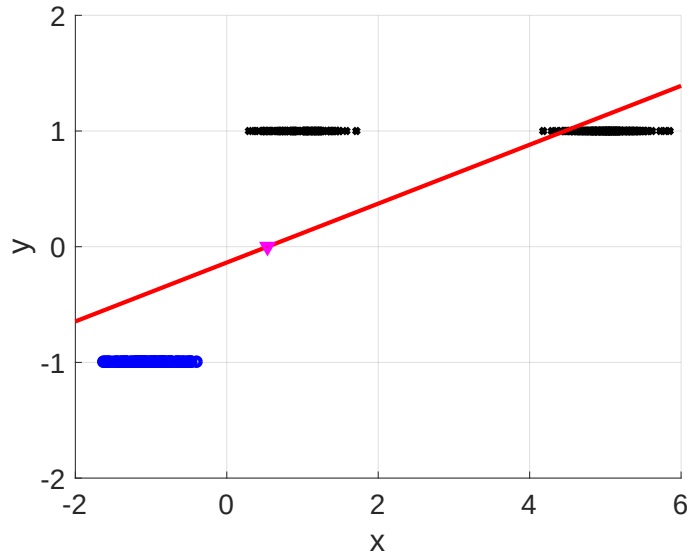
$$y^{(i)} \in \{-1, 1\}$$

Model:

$y = w_1 x + w_0$ threshold at $y = 0$



perfect classification



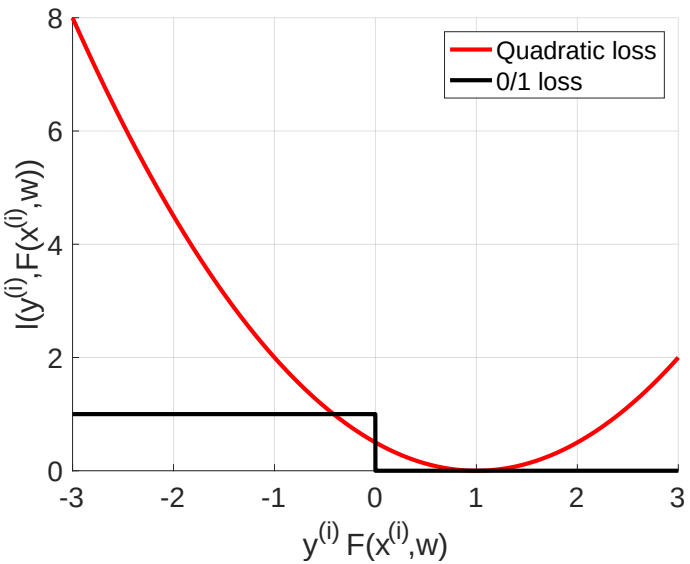
decision boundary shifted

Why is this?

Linear regression: Quadratic loss (recall $y^{(i)} \in \{-1, 1\}$)

$$\ell(y_i, \phi(x^{(i)})^\top \boldsymbol{w}) = \frac{1}{2}(y^{(i)} - \phi(x^{(i)})^\top \boldsymbol{w})^2$$

$$\overset{(y^{(i)})^2=1}{=} \frac{1}{2}(1 - \underbrace{y^{(i)} \underbrace{\phi(x^{(i)})^\top \boldsymbol{w}}_{F(x^{(i)}, \boldsymbol{w})}}_{F(x^{(i)}, \boldsymbol{w}, y^{(i)})})^2$$



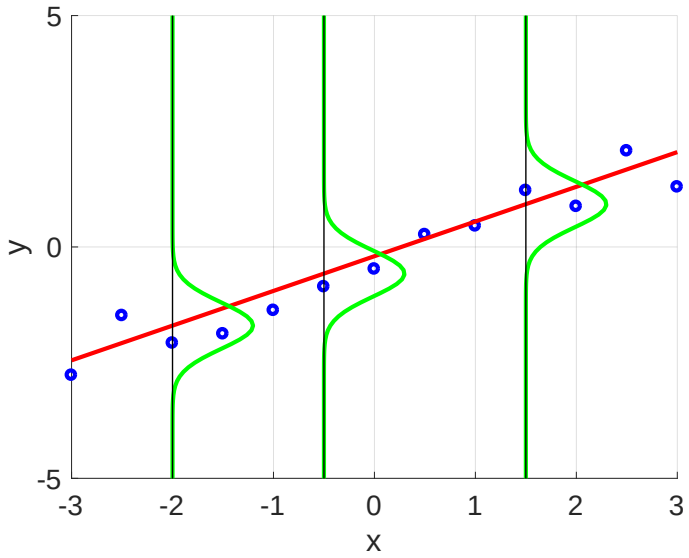
We penalize samples that are 'very easy to classify.'

How to fix this?
Next topic...

Logistic Regression

A probabilistic interpretation of linear regression ($y^{(i)} \in \mathbb{R}$):
Model: Gaussian distribution

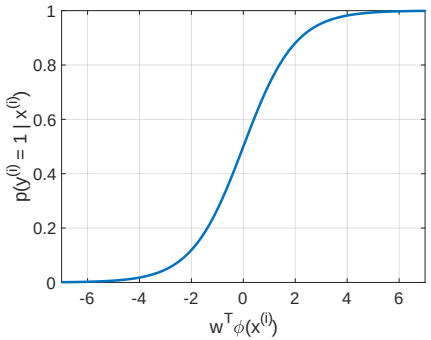
$$p(y^{(i)}|x^{(i)}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y^{(i)} - \boldsymbol{w}^\top \phi(x^{(i)}))^2\right)$$



Logistic Regression:

Another probabilistic formulation for classification ($y^{(i)} \in \{-1, 1\}$):

Model:



$$p(y^{(i)} = 1|x^{(i)}) = \frac{1}{1 + \exp(-\boldsymbol{w}^T \phi(x^{(i)}))}$$

$$p(y^{(i)} = -1|x^{(i)}) = 1 - p(y^{(i)} = 1|x^{(i)}) = \frac{1}{1 + \exp(\boldsymbol{w}^T \phi(x^{(i)}))}$$

Taken together:

$$p(y^{(i)}|x^{(i)}) = \frac{1}{1 + \exp(-y^{(i)} \boldsymbol{w}^T \phi(x^{(i)}))}$$

What to do with this model?

$$p(y^{(i)}|x^{(i)}) = \frac{1}{1 + \exp(-y^{(i)} \boldsymbol{w}^T \phi(x^{(i)}))}$$

Recall that we are given a dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}$. How about we choose $\boldsymbol{w}$ which maximizes the likelihood/probability of this dataset?

Assumption:

Samples/Data points are i.i.d.

$$p(\mathcal{D}) = \prod_{(x^{(i)},y^{(i)})\in\mathcal{D}} p(y^{(i)}|x^{(i)})$$

Choose $\boldsymbol{w}$ to maximize probability:

$$\max_{\boldsymbol{w}} \prod_{(x^{(i)},y^{(i)})\in\mathcal{D}} p(y^{(i)}|x^{(i)})$$

Model:

$$p(y^{(i)}|x^{(i)}) = \frac{1}{1+\exp(-y^{(i)}\boldsymbol{w}^T\phi(x^{(i)}))}$$

Task:

$$\arg\max_{\boldsymbol{w}} \prod_{(x^{(i)},y^{(i)})\in\mathcal{D}} p(y^{(i)}|x^{(i)}) = \arg\min_{\boldsymbol{w}} \sum_{(x^{(i)},y^{(i)})\in\mathcal{D}} -\log p(y^{(i)}|x^{(i)})$$

Combined:

$$\min_{\boldsymbol{w}} \sum_{(x^{(i)},y^{(i)})\in\mathcal{D}} \log\left(1+\exp(-y^{(i)}\boldsymbol{w}^T\phi(x^{(i)}))\right)$$

Comparison

Linear regression

Program:

$$\min_{\boldsymbol{w}} \sum_{(x^{(i)},y^{(i)})\in\mathcal{D}} \underbrace{\frac{1}{2}(1-y^{(i)}\underbrace{\boldsymbol{w}^T\phi(x^{(i)})}_{F(x^{(i)},w)})^2}_{F(x^{(i)},w,y^{(i)})}$$

Logistic regression

Program:

$$\min_{\boldsymbol{w}} \sum_{(x^{(i)},y^{(i)})\in\mathcal{D}} \log\Big(1+\exp(\underbrace{-y^{(i)}\underbrace{\boldsymbol{w}^T\phi(x^{(i)})}_{F(x^{(i)},w)}}_{F(x^{(i)},w,y^{(i)})})\Big)$$

Empirical risk minimization:

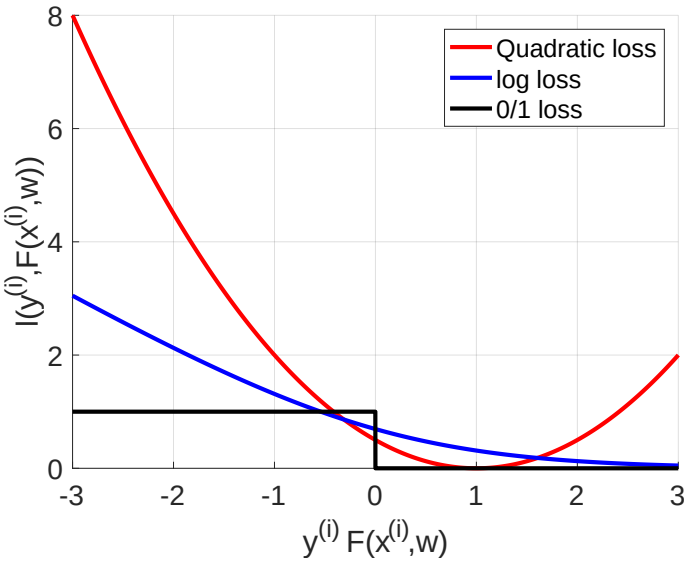
$$\min_{\boldsymbol{w}} \sum_{(x^{(i)},y^{(i)})\in\mathcal{D}} \ell(y^{(i)}, F(x^{(i)}, w))$$

Linear regression:

$$\min_{\boldsymbol{w}} \sum_{(x^{(i)},y^{(i)})\in\mathcal{D}} \frac{1}{2}(1 - y^{(i)} \underbrace{\boldsymbol{w}^T \phi(x^{(i)})}_{F(x^{(i)},w)})^2$$

Logistic regression:

$$\min_{\boldsymbol{w}} \sum_{(x^{(i)},y^{(i)})\in\mathcal{D}} \log\left(1 + \exp(-y^{(i)} \underbrace{\boldsymbol{w}^T \phi(x^{(i)})}_{F(x^{(i)},w)})\right)$$



How to optimize

$$\min_{\boldsymbol{w}} \sum_{(x^{(i)},y^{(i)})\in\mathcal{D}} \log\left(1+\exp(-y^{(i)}\underbrace{\boldsymbol{w}^T\phi(x^{(i)})}_{F(x^{(i)},w)})\right)$$

Can we set the gradient to zero and solve for $\boldsymbol{w}$?

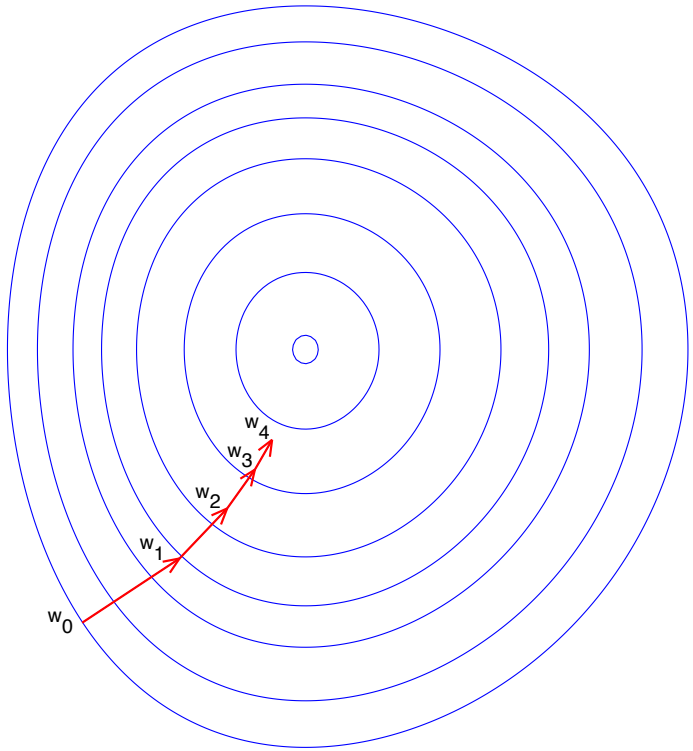
$$\sum_{(x^{(i)},y^{(i)})\in\mathcal{D}} \frac{-y^{(i)}\exp(-y^{(i)}\boldsymbol{w}^T\phi(x^{(i)}))}{1+\exp(-y^{(i)}\boldsymbol{w}^T\phi(x^{(i)}))}\phi(x^{(i)}) = 0$$

No analytic solution for $\boldsymbol{w}$ in general

How to optimize

$$\min_{\boldsymbol{w}} \sum_{(x^{(i)},y^{(i)})\in\mathcal{D}} \log\left(1+\exp(-y^{(i)}\underbrace{\boldsymbol{w}^T\phi(x^{(i)})}_{F(x^{(i)},w)})\right)$$

Gradient descent: (walking down a mountain)



To solve

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}) := \sum_{(x^{(i)},y^{(i)})\in\mathcal{D}} \log\left(1+\exp(-y^{(i)}\underbrace{\boldsymbol{w}^T\phi(x^{(i)})}_{F(x^{(i)},w)})\right)$$

we can use its gradient:

$$\nabla_{\boldsymbol{w}} f(\boldsymbol{w}) = \sum_{(x^{(i)},y^{(i)})\in\mathcal{D}} \frac{-y^{(i)}\exp(-y^{(i)}\boldsymbol{w}^T\phi(x^{(i)}))}{1+\exp(-y^{(i)}\boldsymbol{w}^T\phi(x^{(i)}))}\phi(x^{(i)})$$

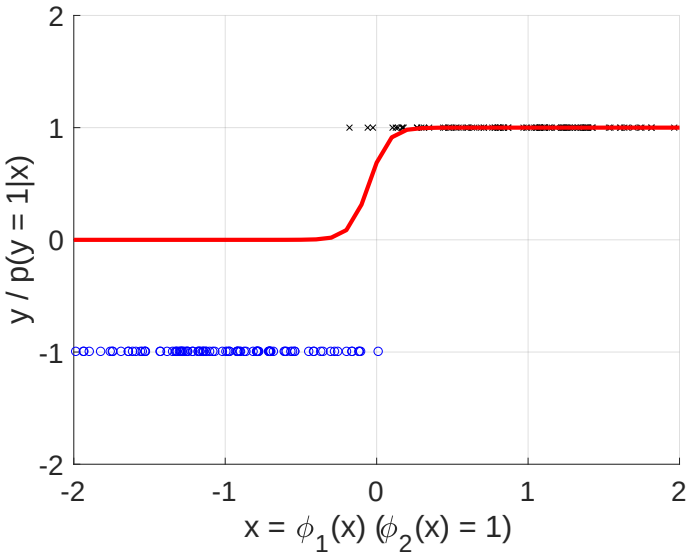
Simple algorithm: Initialize $t=0$, $\boldsymbol{w}_t$, and stepsize α

- Compute gradient $\mathbf{g}_t = \nabla_{\boldsymbol{w}} f(\boldsymbol{w}_t)$
- Update parameters $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \alpha\mathbf{g}_t$
- Update $t \leftarrow t+1$

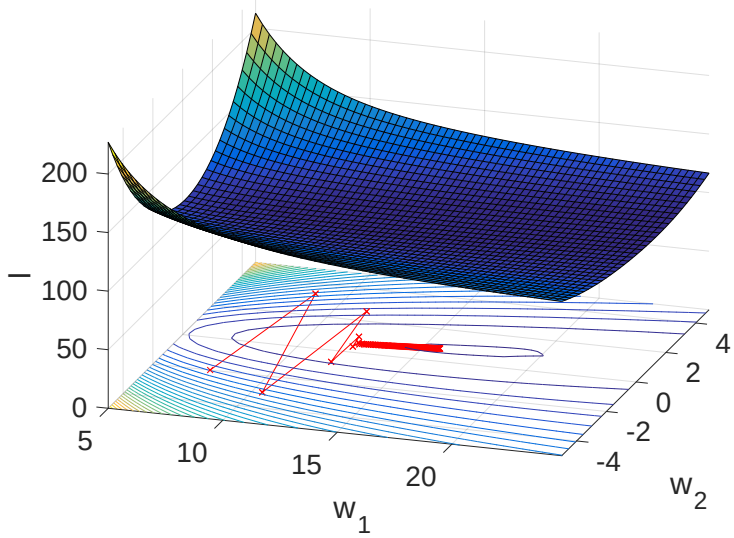
More complex algorithms may be 'better.'

Example:

Data

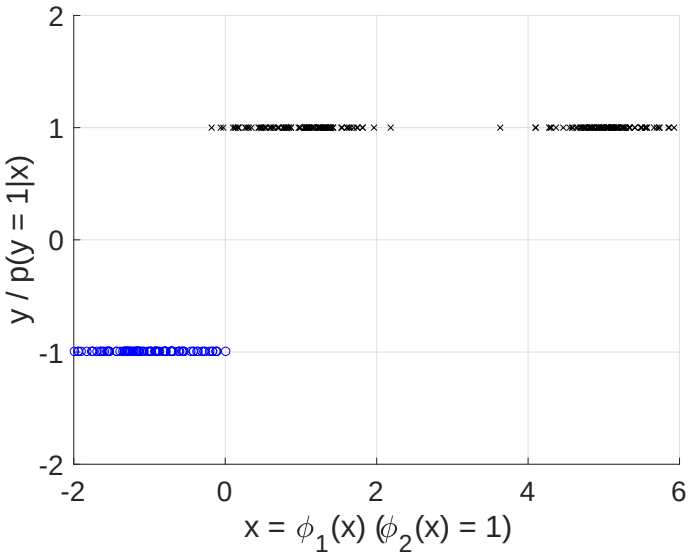


Loss

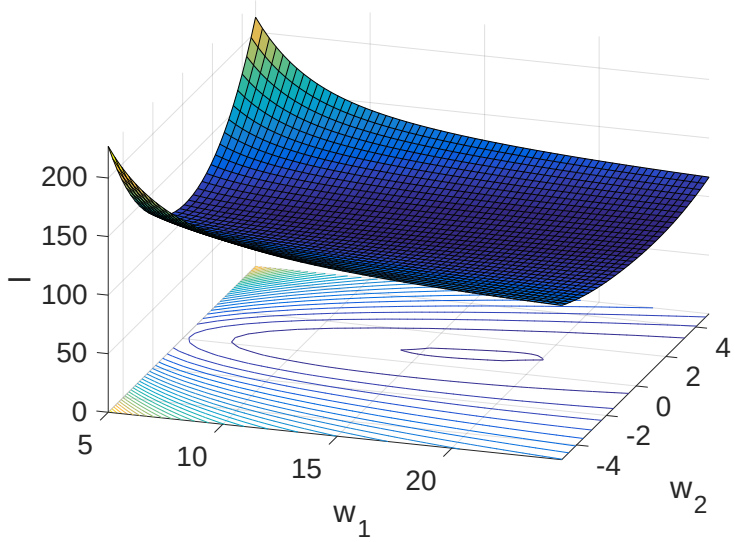


Example:

Data

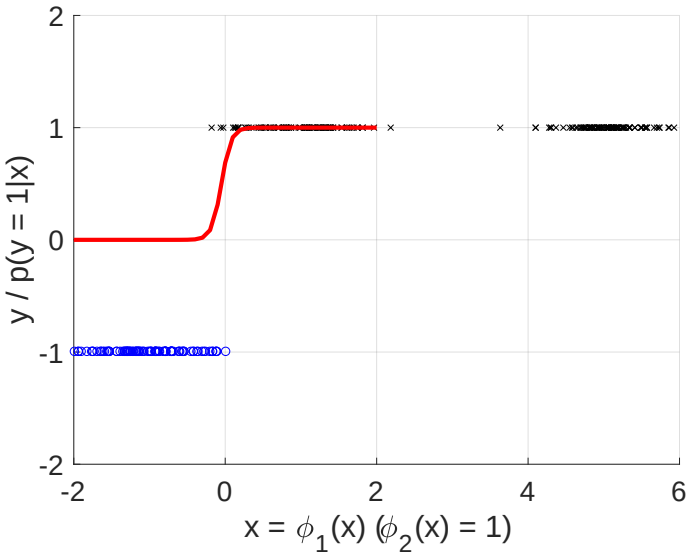


Loss

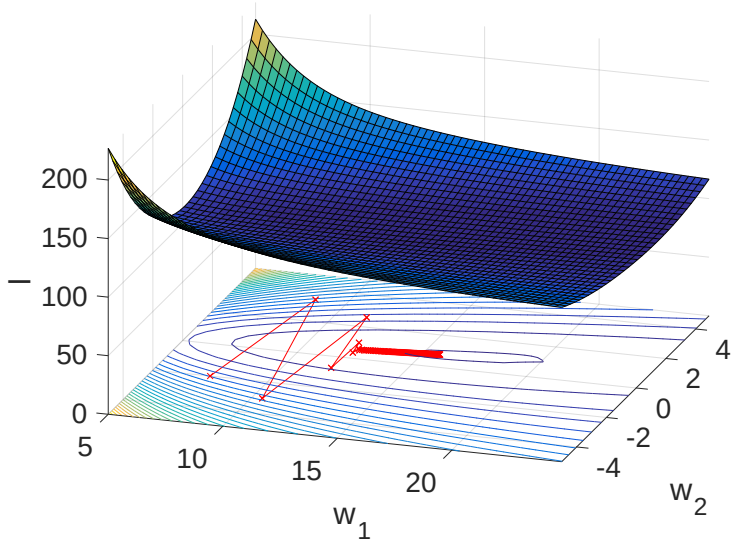


Example:

Data



Loss



Comparison:

Linear regression:

- Closed form solution
- Gaussian probability model
- Not too well suited for classification

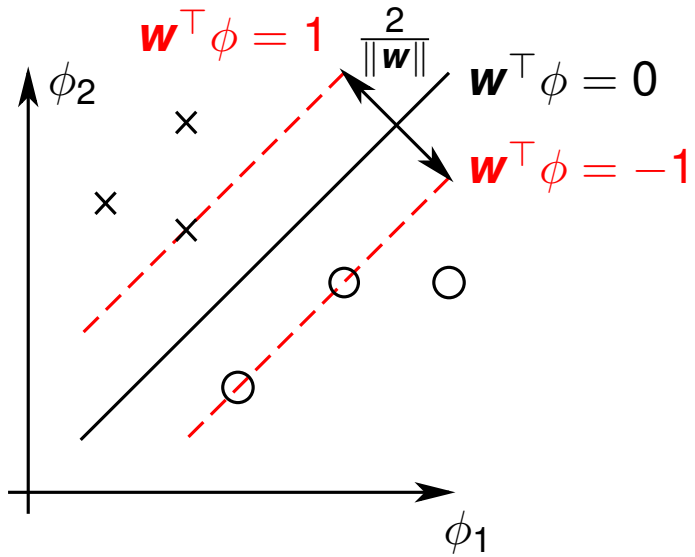
Logistic regression:

- Well suited for binary classification
- Logistic probability model
- No closed form solution

Support Vector Machines

Binary SVM

Intuitively:



Maximize margin $\frac{2}{\|w\|}$:

$$\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|_2^2 \qquad \text{s.t.} \qquad y^{(i)}\boldsymbol{w}^\top\phi(x^{(i)}) \geq \underbrace{1}_{\text{Taskloss: } L} \qquad \forall(\phi(x^{(i)}), y^{(i)}) \in \mathcal{D}$$

Note: any value $L \geq 0$ is okay.

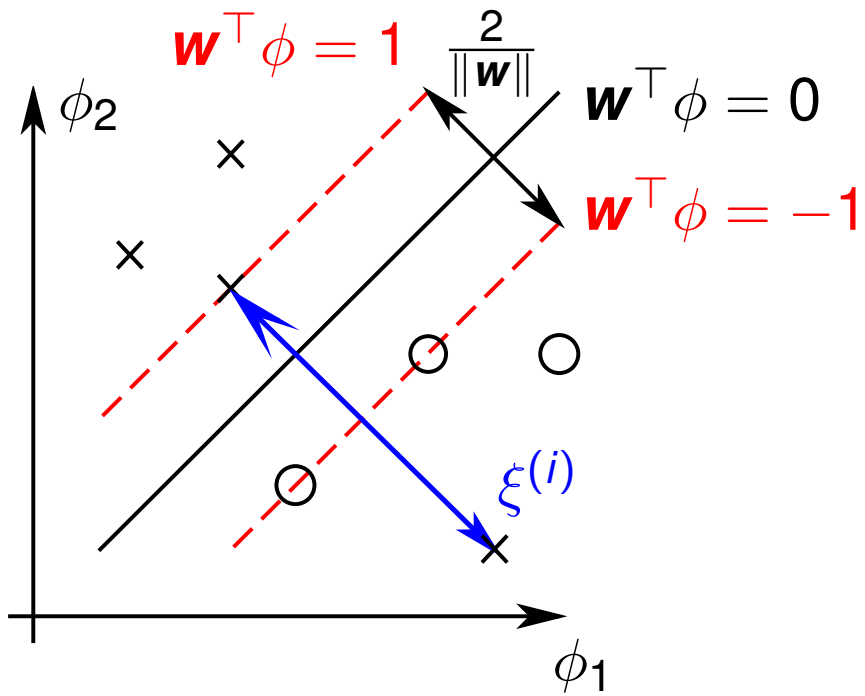
Issue: what if data not linearly separable?

Binary SVM

Introduce slack variables ξ:

$$\min_{\boldsymbol{w},\xi^{(i)}\geq 0} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \xi^{(i)} \qquad \text{s.t.} \qquad y^{(i)}\boldsymbol{w}^\top \phi(x^{(i)}) \geq 1 - \xi^{(i)} \quad \forall i \in \mathcal{D}$$

Intuitively:



Binary SVM:

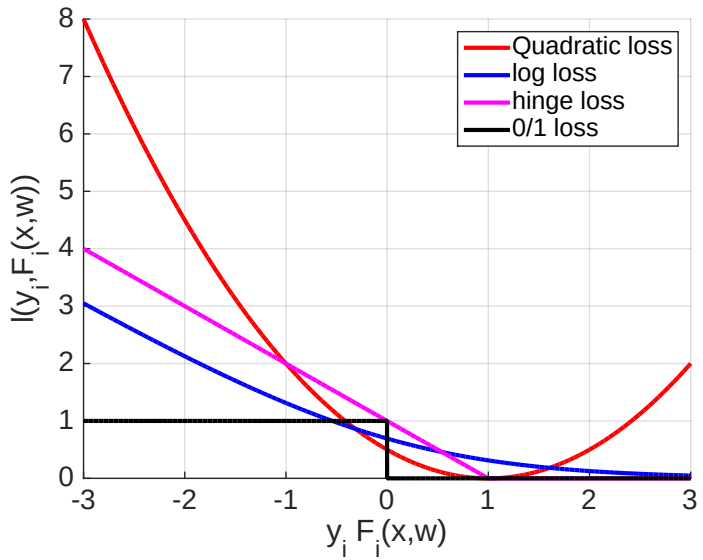
$$\min_{\boldsymbol{w},\xi^{(i)}\geq 0} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \xi^{(i)} \qquad \text{s.t.} \qquad y^{(i)}\boldsymbol{w}^\top \phi(x^{(i)}) \geq 1 - \xi^{(i)} \quad \forall i \in \mathcal{D}$$

Equivalent:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \max\{0, 1 - y^{(i)}\boldsymbol{w}^\top \phi(x^{(i)})\}$$

Empirical risk minimization:

$$\min_{\boldsymbol{w}} R(\boldsymbol{w}) + \sum_{i\in\mathcal{D}} \ell(y^{(i)}, F(x^{(i)}, \boldsymbol{w}))$$



How to optimize the binary SVM objective (primal problem):

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \max\{0, 1 - y^{(i)}\boldsymbol{w}^\top \phi(x^{(i)})\}$$

Approaches:

- Optimize the primal via gradient descent
- Optimize the corresponding dual problem

Recap:

- Linear regression:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \frac{1}{2}(1 - y^{(i)} \underbrace{\boldsymbol{w}^T\phi(x^{(i)})}_{F(x^{(i)},\boldsymbol{w})})^2$$

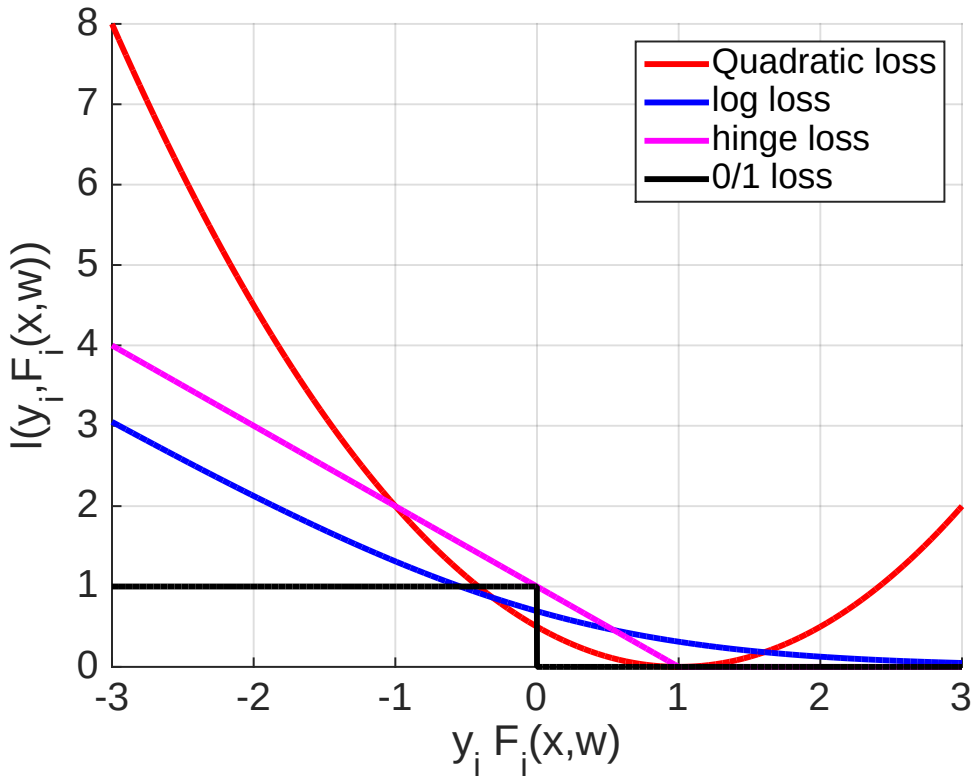
- Logistic regression:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \log\left(1 + \exp(-y^{(i)} \underbrace{\boldsymbol{w}^T\phi(x^{(i)})}_{F(x^{(i)},\boldsymbol{w})})\right)$$

- Binary SVM:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \max\{0, \underbrace{1}_{\text{taskloss}} - y^{(i)} \underbrace{\boldsymbol{w}^\top\phi(x^{(i)})}_{F(x^{(i)},\boldsymbol{w})}\}$$

Recap:


Quadratic loss
log loss
hinge loss
0/1 loss
8
7
6
5
4
3
2
1
0
-3
-2
-1
0
1
2
3
l(y_i,F_i(x,w))
y_i F_i(x,w)

Combining log- and hinge-loss:

$$\lim_{\epsilon \to 0} \epsilon \log\left(1 + \exp \frac{-F}{\epsilon}\right) =$$

$$\overset{F \geq 0}{=} 0$$

$$\overset{F \leq 0}{=} \lim_{\epsilon \to 0} \frac{\log\left(1 + \exp \frac{-F}{\epsilon}\right)}{1/\epsilon}$$

$$= \lim_{\epsilon \to 0} \frac{\frac{\exp \frac{-F}{\epsilon}}{1+\exp \frac{-F}{\epsilon}} \cdot (F/\epsilon^2)}{-1/\epsilon^2}$$

$$= \lim_{\epsilon \to 0} \frac{1}{1 + \exp \frac{F}{\epsilon}} \cdot (-F)$$

$$= -F$$

In summary:

$$\lim_{\epsilon \to 0} \epsilon \log\left(1 + \exp \frac{-F}{\epsilon}\right) = \max\{0, -F\}$$

SVM as 0-temperature limit of logistic regression

Recap:

- Linear regression:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \frac{1}{2}(1 - y^{(i)} \underbrace{\boldsymbol{w}^T\phi(x^{(i)})}_{F(x^{(i)},\boldsymbol{w})})^2$$

- Logistic regression:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \log\left(1 + \exp(-y^{(i)} \underbrace{\boldsymbol{w}^T\phi(x^{(i)})}_{F(x^{(i)},\boldsymbol{w})})\right)$$

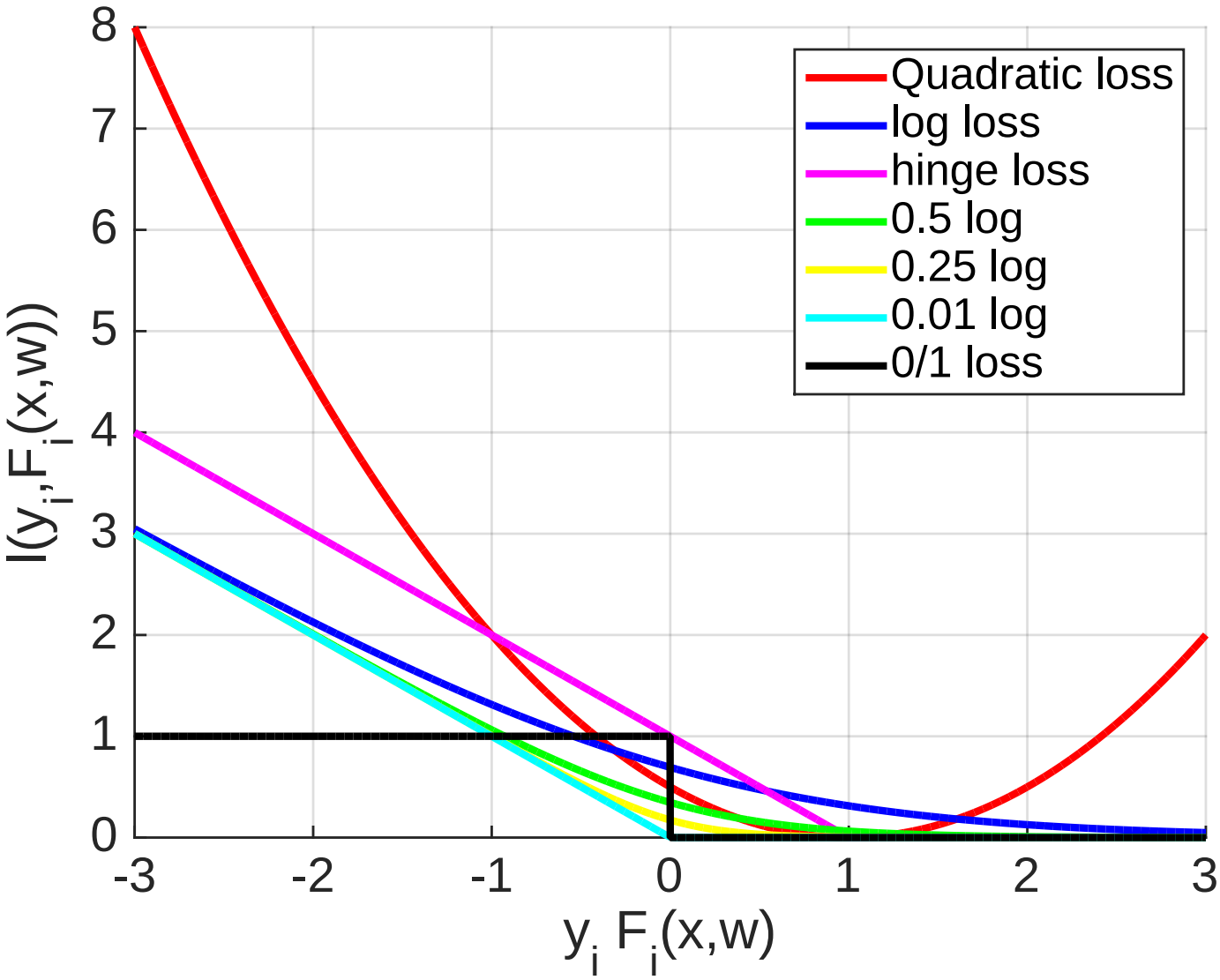
- Binary SVM:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \max\{0, \underbrace{1}_{\text{taskloss}} - y^{(i)} \underbrace{\boldsymbol{w}^\top\phi(x^{(i)})}_{F(x^{(i)},\boldsymbol{w})}\}$$

- General binary classification:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \epsilon \log\left(1 + \exp\left(\frac{L - y^{(i)}\boldsymbol{w}^T\phi(x^{(i)})}{\epsilon}\right)\right)$$

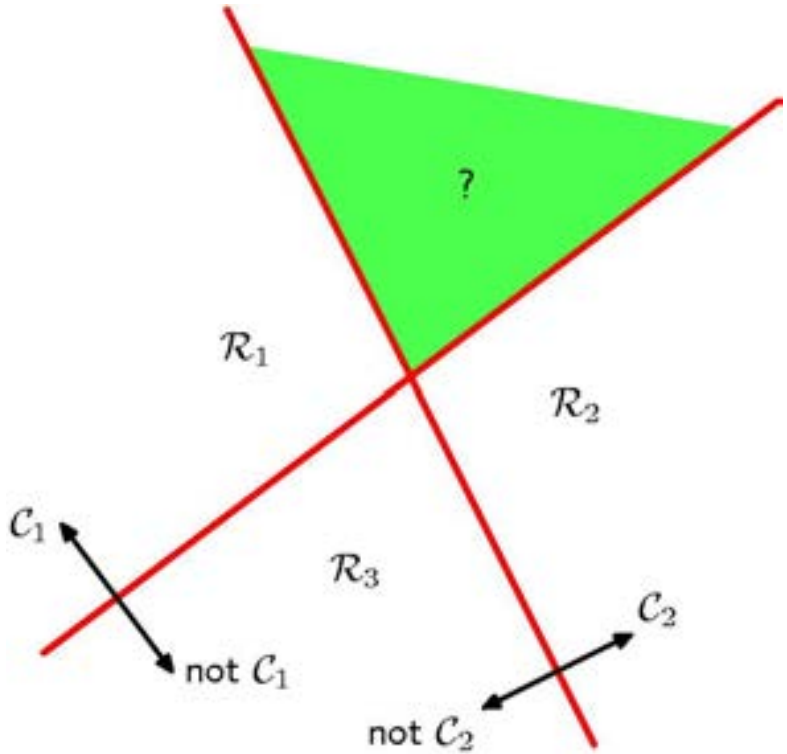
Different loss functions



Multiclass Classification and Kernel Methods

Multiclass classification: How to classify between K classes?

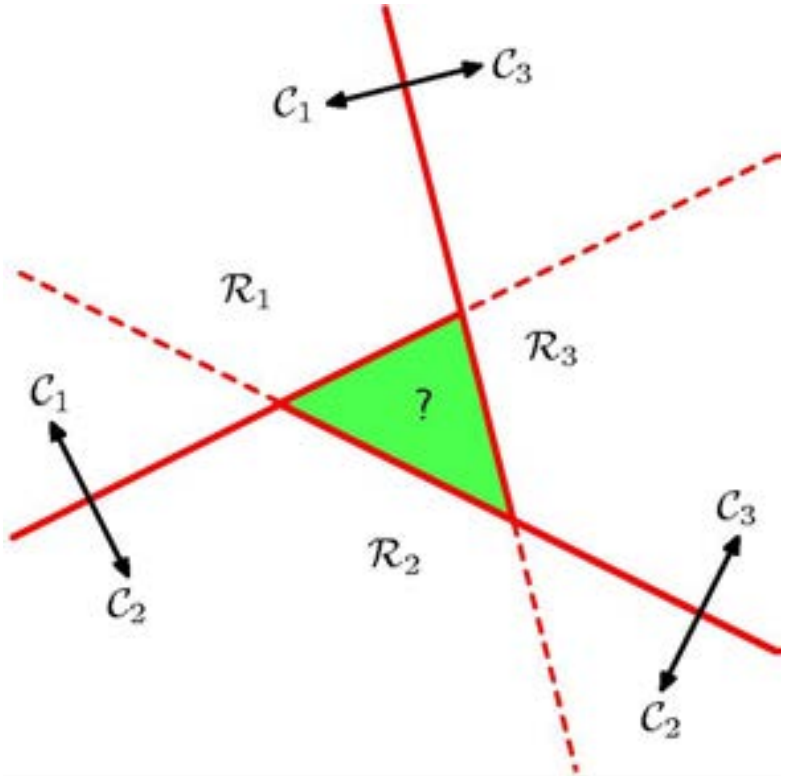
- **1 vs all** or **1 vs rest** classifier
- Use $K - 1$ classifiers, each solving a two class problem which separates a point in class k from point not in this class



- Issue: more than one good answer or no good answer

Multiclass classification: How to classify between K classes?

- **1 vs 1** classifier
- Use $K(K-1)/2$ two-way classifiers, one for each possible pair of classes
- Classify sample according to majority vote



- Issue: two-way preferences need not be transitive

Multiclass classification: How to classify between K classes?

- Use a multinomial distribution over $y \in \{0, 1, \ldots, K-1\}$
- Use K weight vectors $\boldsymbol{w}_{(y)}$
- How to parameterize the multinomial distribution?

$$p(y = k|x^{(i)}) = \frac{\exp \boldsymbol{w}_{(k)}^{\top} \phi(x^{(i)})}{\sum_{j \in \{0,1,\ldots,K-1\}} \exp \boldsymbol{w}_{(j)}^{\top} \phi(x^{(i)})}$$

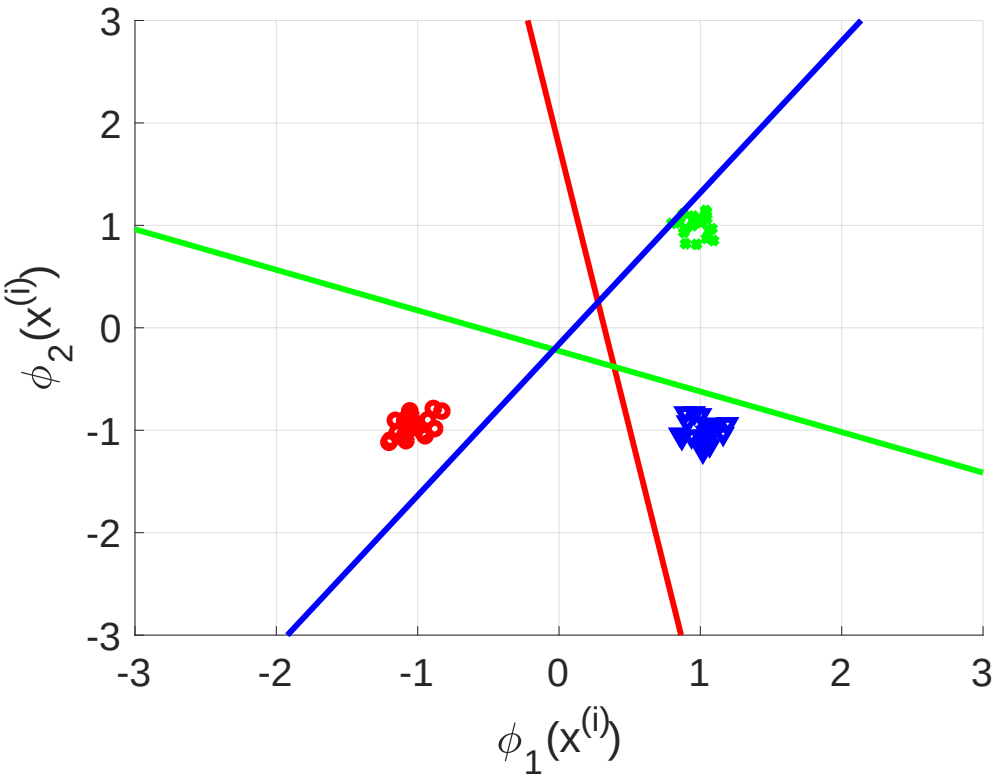
- We are using one parameter vector $\boldsymbol{w}_{(y)}$ per class
- Maximizing the likelihood as before

$$\arg\max_{\boldsymbol{w}} \prod_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} p(y = y^{(i)}|x^{(i)}) = \arg\min_{\boldsymbol{w}} \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} -\log p(y = y^{(i)}|x^{(i)})$$

Questions:

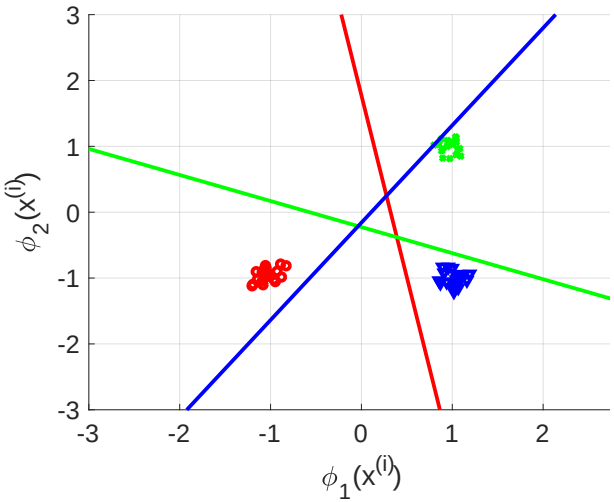
- Example to build intuition?
- How does this relate to our earlier binary setting?

Example: $(\phi_3(x^{(i)}) = 1)$

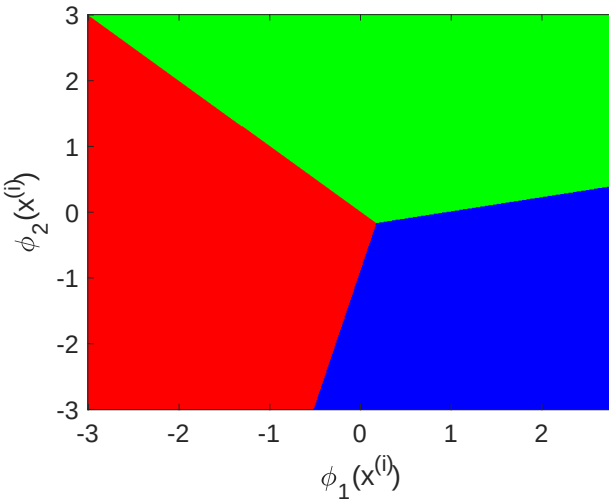


Example: $(\phi_3(x^{(i)}) = 1)$

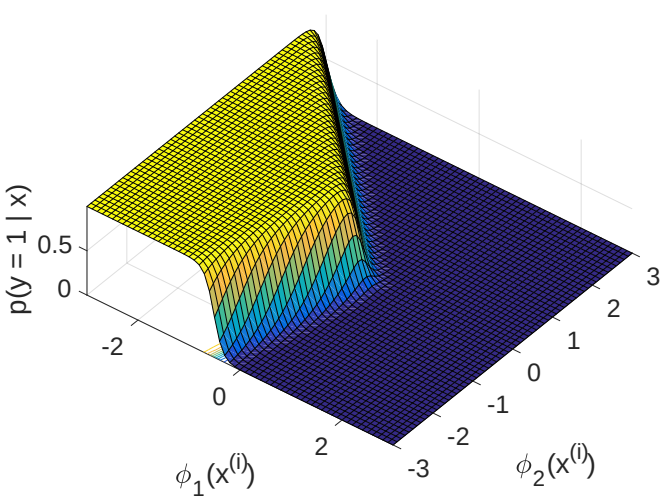
Setup



Decision Boundary



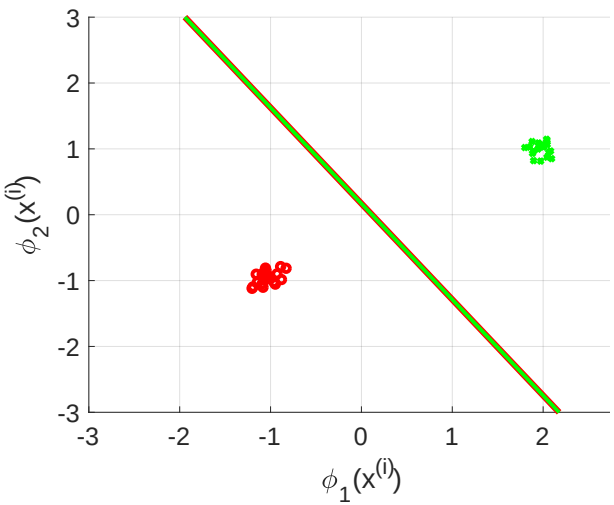
Probability for $k = 1$



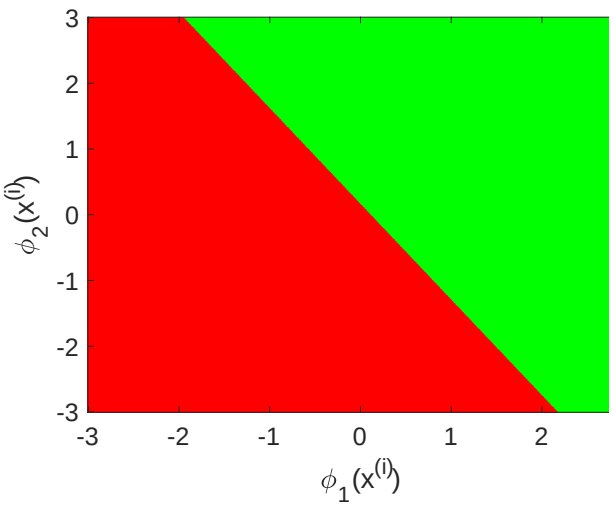
Example: What happens when we have two classes?

- We get two hyperplanes (lines in 2D). How do they related to binary logistic regression?

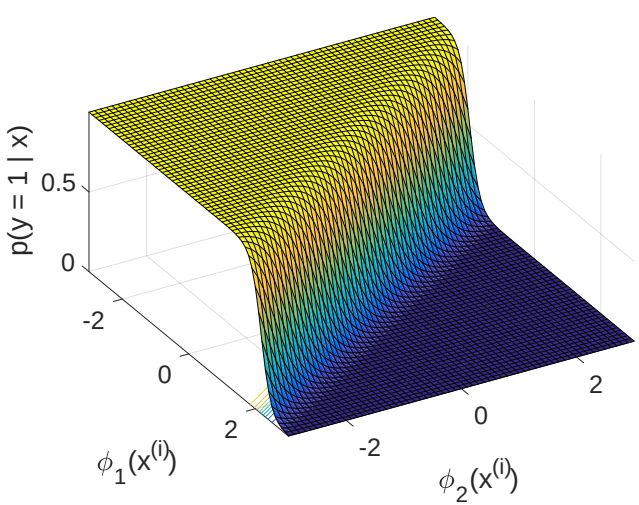
Setup



Decision Boundary



Probability for $k = 1$

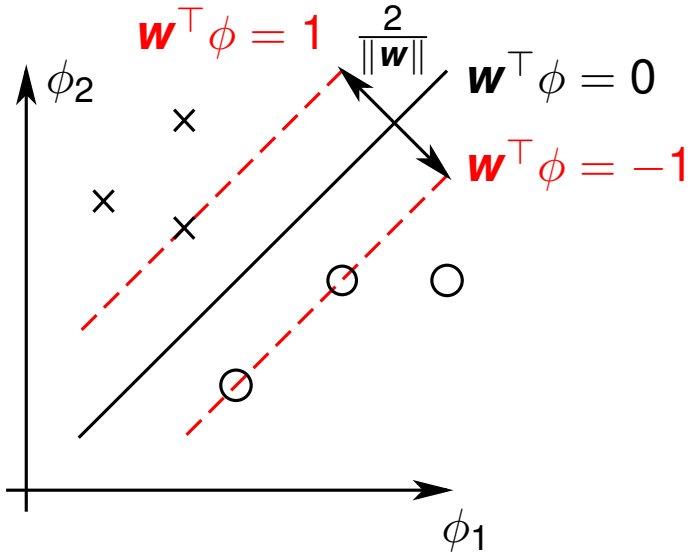


Lines coincide ($\boldsymbol{w}_{(1)} = -\boldsymbol{w}_{(2)}$)

How about multi-class SVM?

Recap: Binary SVM

Intuitively:



$$u_1 + c\boldsymbol{w} = u_2 \qquad \boldsymbol{w}^\top u_2 = -1$$

$$\boldsymbol{w}^\top(u_1 + c\boldsymbol{w}) = -1$$

$$c = \frac{-2}{\|\boldsymbol{w}\|_2^2}$$

$$\|u_1 - u_2\|_2 = \| - c\boldsymbol{w}\|_2 = \frac{2}{\|\boldsymbol{w}\|_2^2}\|\boldsymbol{w}\| = \frac{2}{\|\boldsymbol{w}\|}$$

Maximize margin $\frac{2}{\|w\|}$:

$$\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|_2^2 \qquad \text{s.t.} \qquad y^{(i)}\boldsymbol{w}^\top\phi(x^{(i)}) \geq \underbrace{1}_{\text{Taskloss: } L} \qquad \forall (x^{(i)}, y^{(i)}) \in \mathcal{D}$$

Note: any value $L \geq 0$ is okay.

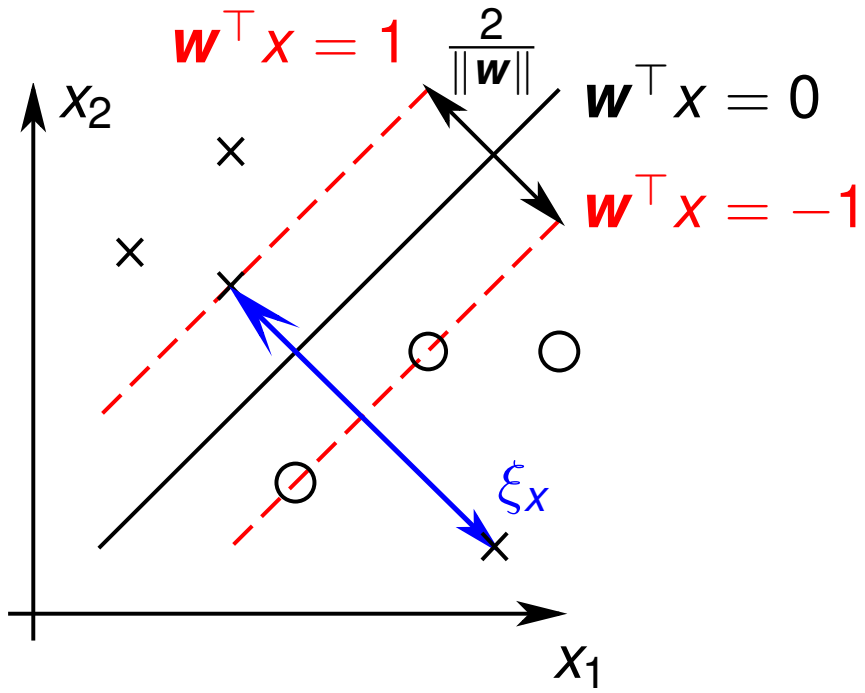
Issue: what if data not linearly separable?

Recap: Binary SVM

Introduce slack variables $\xi^{(i)}$:

$$\min_{\boldsymbol{w},\xi^{(i)}\geq 0} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \xi^{(i)} \qquad \text{s.t.} \qquad y^{(i)}\boldsymbol{w}^\top \phi(x^{(i)}) \geq 1 - \xi^{(i)} \quad \forall i \in \mathcal{D}$$

Intuitively:



How about multi-class SVM? Recall $y \in \{0, 1, \ldots, K-1\}$ and class vectors $\boldsymbol{w}_{(y)}$
What do we want?

$$\boldsymbol{w}_{y^{(i)}}^T \phi(x^{(i)}) \geq \boldsymbol{w}_{\hat{y}}^T \phi(x^{(i)}) \qquad \forall i \in \mathcal{D}, \hat{y} \in \{0, 1, \ldots, K-1\}$$

$$\min_{\boldsymbol{w}, \xi^{(i)} \geq 0} \frac{C}{2} \|\boldsymbol{w}\|_2^2 + \sum_{i \in \mathcal{D}} \xi^{(i)} \quad \text{s.t.} \quad \boldsymbol{w}_{y^{(i)}}^T \phi(x^{(i)}) - \boldsymbol{w}_{\hat{y}}^T \phi(x^{(i)}) \geq 1 - \xi^{(i)} \quad \forall i, \hat{y}$$

Multiclass SVM objective:

$$\min_{\boldsymbol{w},\xi^{(i)}\geq 0} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \xi^{(i)} \quad \text{s.t.} \quad \boldsymbol{w}_{y^{(i)}}^T \phi(x^{(i)}) - \boldsymbol{w}_{\hat{y}}^T \phi(x^{(i)}) \geq 1 - \xi^{(i)} \quad \forall i, \hat{y}$$

Define mapping:

$$\boldsymbol{w} = \begin{bmatrix} \boldsymbol{w}_0 \\ \vdots \\ \boldsymbol{w}_{K-1} \end{bmatrix} \qquad \psi(x^{(i)}, y^{(i)}) = \begin{bmatrix} \phi(x^{(i)})\delta(y^{(i)} = 0) \\ \vdots \\ \phi(x^{(i)})\delta(y^{(i)} = K-1) \end{bmatrix}$$

Equivalent formulation:

$$\min_{\boldsymbol{w},\xi^{(i)}\geq 0} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \xi^{(i)} \quad \text{s.t.} \quad \boldsymbol{w}^T \left(\psi(x^{(i)}, y^{(i)}) - \psi(x^{(i)}, \hat{y})\right) \geq 1 - \xi^{(i)} \quad \forall i, \hat{y}$$

Multiclass SVM objective:

$$\min_{\boldsymbol{w},\xi^{(i)}\geq 0} \frac{C}{2}\|\boldsymbol{w}\|_2^2+\sum_{i\in\mathcal{D}}\xi^{(i)} \quad \text{s.t.} \quad \boldsymbol{w}^T\left(\psi(x^{(i)},y^{(i)})-\psi(x^{(i)},\hat{y})\right)\geq 1-\xi^{(i)} \quad \forall i,\hat{y}$$

Let's get rid of the slack variable:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2+\sum_{i\in\mathcal{D}}\max\{0,\underbrace{\max_{\hat{y}}(1-\boldsymbol{w}^T(\psi(x^{(i)},y^{(i)})-\psi(x^{(i)},\hat{y})))}_{\geq 0}\}$$

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2+\sum_{i\in\mathcal{D}}\max_{\hat{y}}(1-\boldsymbol{w}^T(\psi(x^{(i)},y^{(i)})-\psi(x^{(i)},\hat{y})))$$

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2+\sum_{i\in\mathcal{D}}\underbrace{\max_{\hat{y}}(1+\boldsymbol{w}^T\psi(x^{(i)},\hat{y}))}_{\text{Loss-augmented inference}}-\boldsymbol{w}^T\psi(x^{(i)},y^{(i)})$$

How does multiclass SVM relate to multiclass logistic regression?

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \underbrace{\max_{\hat{y}}(1 + \boldsymbol{w}^T\psi(x^{(i)},\hat{y}))}_{\text{Loss-augmented inference}} - \boldsymbol{w}^T\psi(x^{(i)},y^{(i)})$$

Same temperature argument as in the last lecture.

Solution:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)},\hat{y}) + \boldsymbol{w}^T\psi(x^{(i)},\hat{y}))}{\epsilon} - \boldsymbol{w}^T\psi(x^{(i)},y^{(i)})$$

Our current framework:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)},\hat{y}) + \boldsymbol{w}^T\psi(x^{(i)},\hat{y}))}{\epsilon} - \boldsymbol{w}^T\psi(x^{(i)},y^{(i)})$$

What is a possible issue/limitation?

Linearity in the feature space $\psi(x^{(i)}, y^{(i)})$.

How to fix this?

Use Kernels

Dual of Logistic Regression:

$$\max_{0 \le \lambda^{(i)} \le 1} g(\lambda) := \frac{-1}{2C} \| \sum_i \lambda^{(i)} y^{(i)} \phi(x^{(i)}) \|_2^2 + \sum_i H(\lambda^{(i)})$$

Prediction with dual variables:

$$\boldsymbol{w}^\top \phi(x) = \frac{1}{C} \sum_i \lambda^{(i)} y^{(i)} \phi(x^{(i)})^\top \phi(x)$$

Dual of SVM:

$$\max_{0 \le \alpha \le 1} g(\alpha) := \frac{-1}{2C} \| \sum_i \alpha^{(i)} y^{(i)} \phi(x^{(i)}) \|_2^2 + \sum_i \alpha^{(i)}$$

Prediction with dual variables:

$$\boldsymbol{w}^\top \phi(x) = \frac{1}{C} \sum_i \alpha^{(i)} y^{(i)} \phi(x^{(i)})^\top \phi(x)$$

Observation:

Many inner products $\phi(x^{(i)})^\top \phi(x^{(j)})$ between different samples

Kernel trick:

Replace inner products with call to kernel $\kappa(x^{(i)}, x^{(j)}) \in \mathbb{R}$

Advantage:

No need to explicitly construct feature vector $\phi(x)$

But:

We need to ensure that κ is a valid kernel, i.e., it corresponds to an inner product in some feature space

Example:

$$\begin{aligned}
\kappa(x, z) &= (x^\top z)^2 \\
&= x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\
&= [\ x_1^2 \quad \sqrt{2} x_1 x_2 \quad x_2^2\ ][\ z_1^2 \quad \sqrt{2} z_1 z_2 \quad z_2^2\ ]^\top \\
&= \phi(x)^\top \phi(z)
\end{aligned}$$

How to test validity without explicitly constructing feature vectors:

- Gram matrix $(\boldsymbol{K})_{i,j} = \kappa(x^{(i)}, x^{(j)})$ is positive definite $\forall x^{(i)}, x^{(j)}$
- Decompose/Compose kernel into/from known kernels

Example kernels:

- Linear kernel:

$$\kappa(x^{(i)}, x^{(j)}) = {x^{(i)}}^\top x^{(j)}$$

- Squared exponential (Gaussian) kernel:

$$\kappa(x^{(i)}, x^{(j)}) = \exp\left(-\frac{1}{2}(x^{(i)} - x^{(j)})^\top \Sigma^{-1}(x^{(i)} - x^{(j)})\right)$$

- Sigmoid kernel:

$$\kappa(x^{(i)}, x^{(j)}) = \tanh(a \cdot {x^{(i)}}^\top x^{(j)} + b)$$

Techniques for constructing new kernels ($c > 0$):

- Positive scaling:

$$\kappa(x^{(i)}, x^{(j)}) = c\kappa_1(x^{(i)}, x^{(j)})$$

- Exponentiation:

$$\kappa(x^{(i)}, x^{(j)}) = \exp(\kappa_1(x^{(i)}, x^{(j)}))$$

- Addition:

$$\kappa(x^{(i)}, x^{(j)}) = \kappa_1(x^{(i)}, x^{(j)}) + \kappa_2(x^{(i)}, x^{(j)})$$

- Multiplication with function:

$$\kappa(x^{(i)}, x^{(j)}) = f(x^{(i)})\kappa_1(x^{(i)}, x^{(j)})f(x^{(j)})$$

- Multiplication:

$$\kappa(x^{(i)}, x^{(j)}) = \kappa_1(x^{(i)}, x^{(j)})\kappa_2(x^{(i)}, x^{(j)})$$

Deep Neural Networks

Goals of this lecture

- Getting to know deep nets
- Understanding forward and backward pass
- Learning about deep net components

Reading material

- I. Goodfellow et al.; Deep Learning; Chapters 6-9

Our current framework:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)},\hat{y}) + \boldsymbol{w}^T\psi(x^{(i)},\hat{y}))}{\epsilon} - \boldsymbol{w}^T\psi(x^{(i)},y^{(i)})$$

What is a possible issue/limitation?

Linearity in the feature space $\psi(x,y)$. Fix: use kernels. But still learning a model **linear** in the parameters $\boldsymbol{w}$

How to fix this?

Replace $\boldsymbol{w}^T\psi(x,y)$ with a general function $F(\boldsymbol{w},x,y)$

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)},\hat{y}) + F(\boldsymbol{w},x^{(i)},\hat{y})}{\epsilon} - F(\boldsymbol{w},x^{(i)},y^{(i)})$$

General framework:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)},\hat{y}) + F(\boldsymbol{w}, x^{(i)}, \hat{y})}{\epsilon} - F(\boldsymbol{w}, x^{(i)}, y^{(i)})$$

How to get to

- **Logistic regression**
- **Binary SVM**
- **Multiclass regression**
- **Multiclass SVM**
- **Deep Learning**

Deep Learning:

What function $F(\boldsymbol{w}, x, y) \in \mathbb{R}$ to choose?

- Choose any differentiable composite function

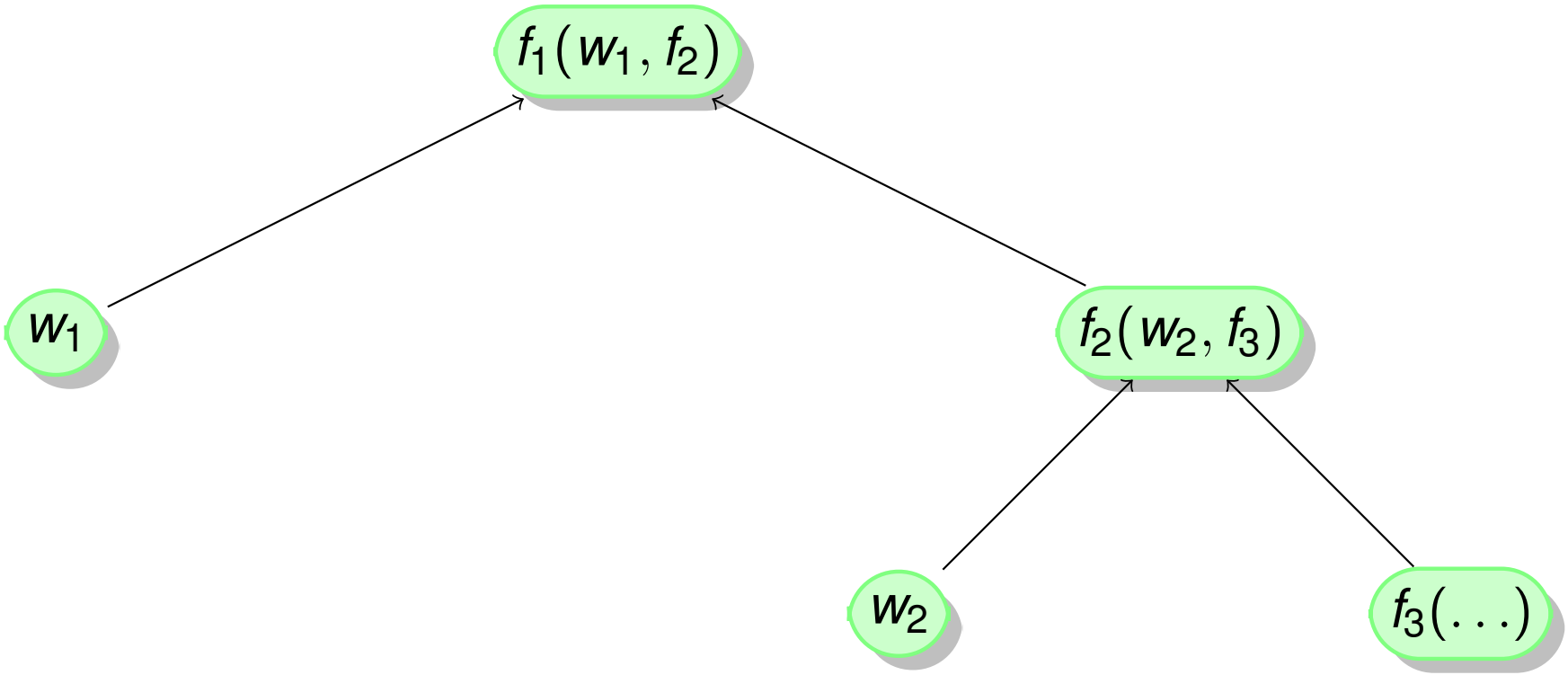
$$F(\boldsymbol{w}, x, y) = f_1(\boldsymbol{w}_1, y, f_2(\boldsymbol{w}_2, f_3(\ldots f_n(\boldsymbol{w}_n, x)\ldots)))$$

- More generally: functions can be represented by an acyclic graph (computation graph)

Example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, f_2(w_2, f_3(\ldots)))$$

Nodes are weights, data, and functions:



Internal representation used by deep net packages.

Non-linear Functions

A neuron's activation is a non-linear function which takes $\xi \in \Re$ as input and produces $\sigma(\xi) \in \Re$ as output.

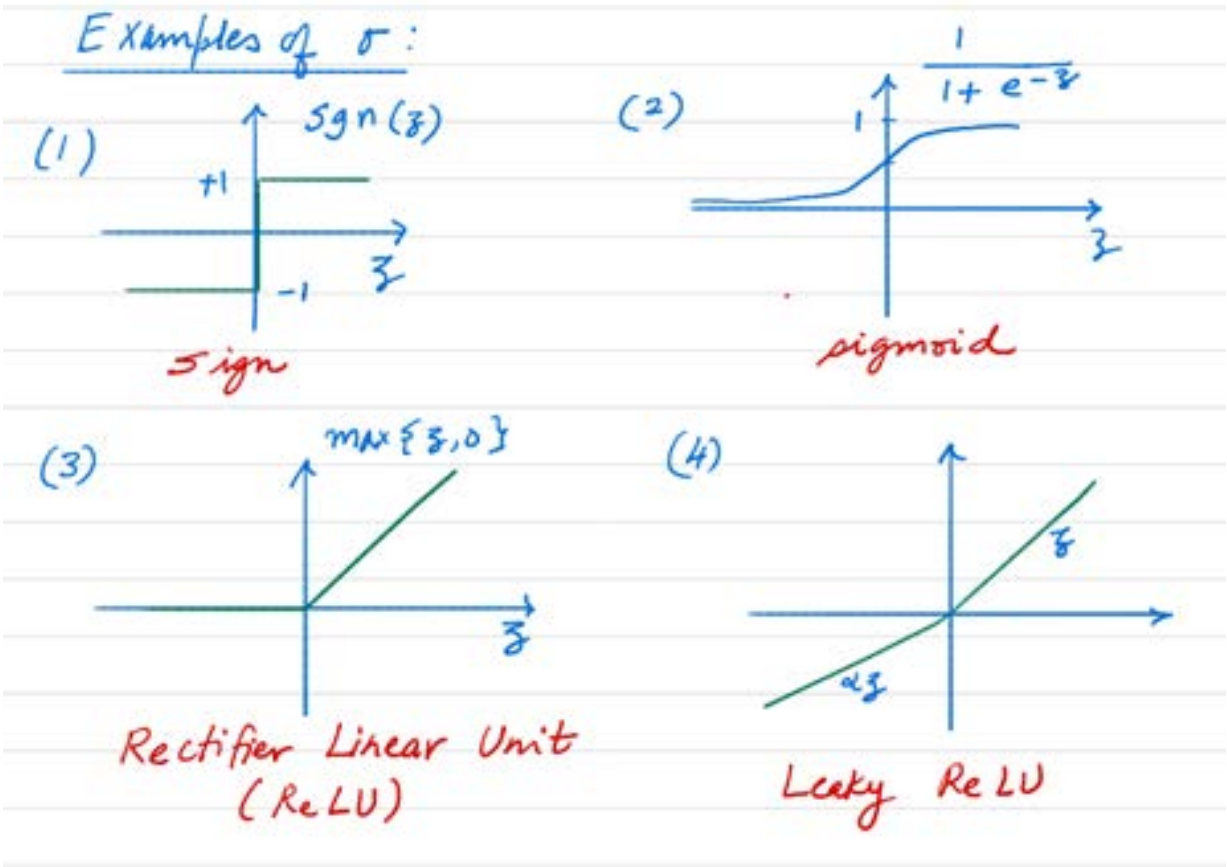


Figure: Examples of σ

Simplified Representation

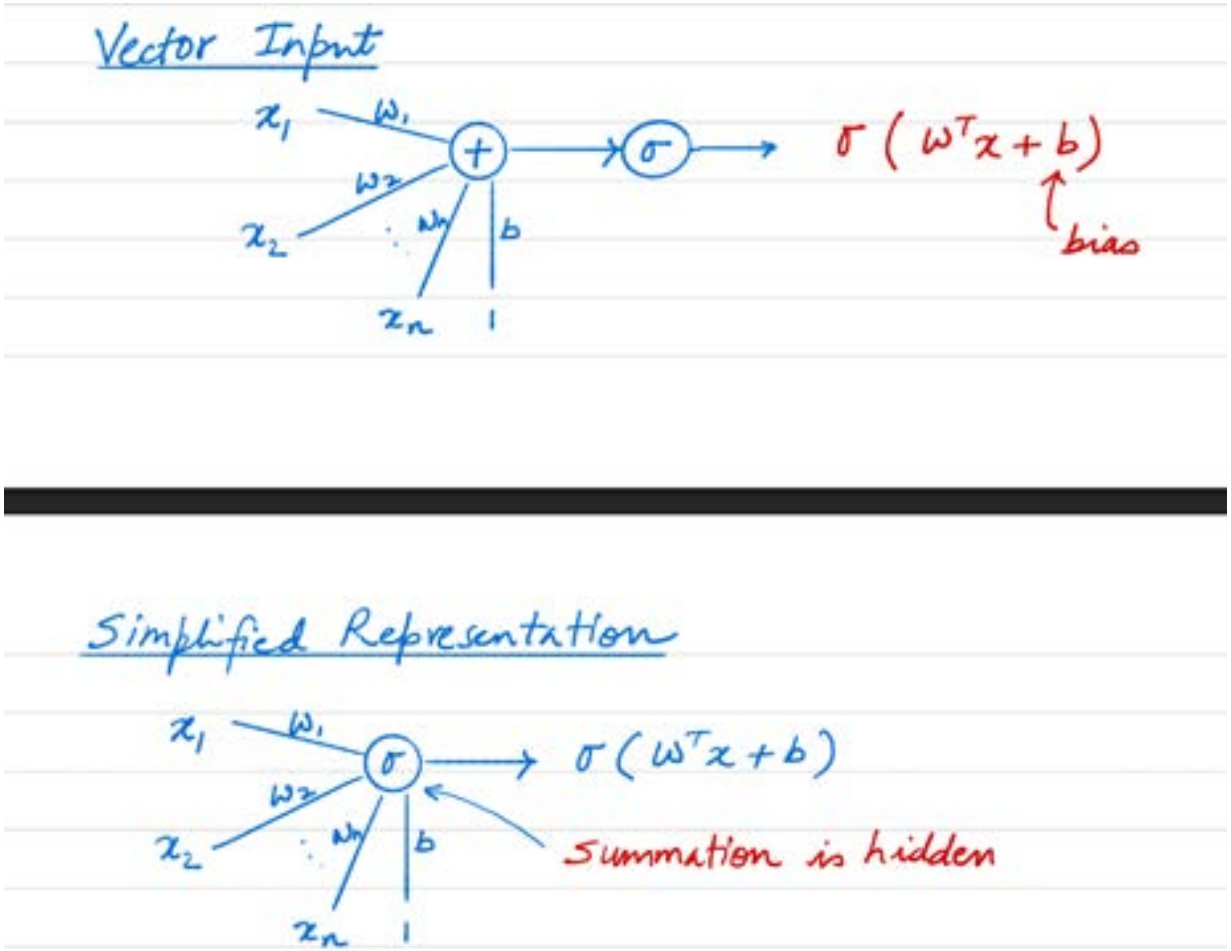
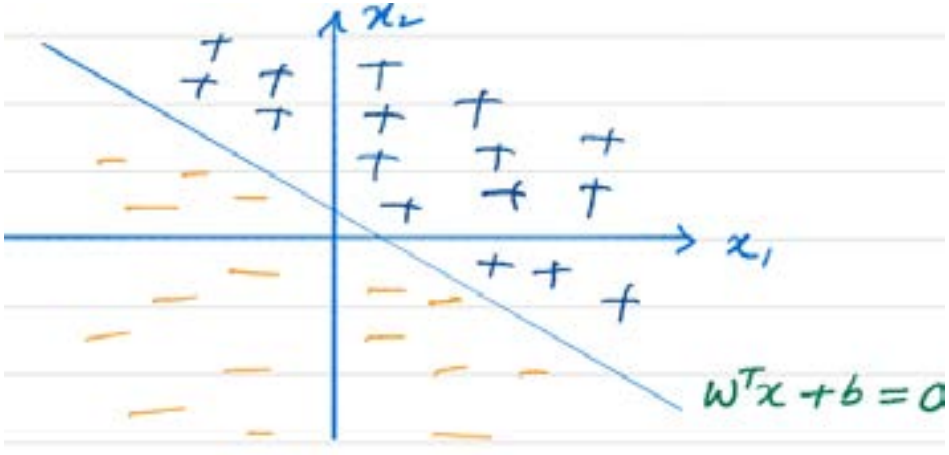


Figure: Enter Caption

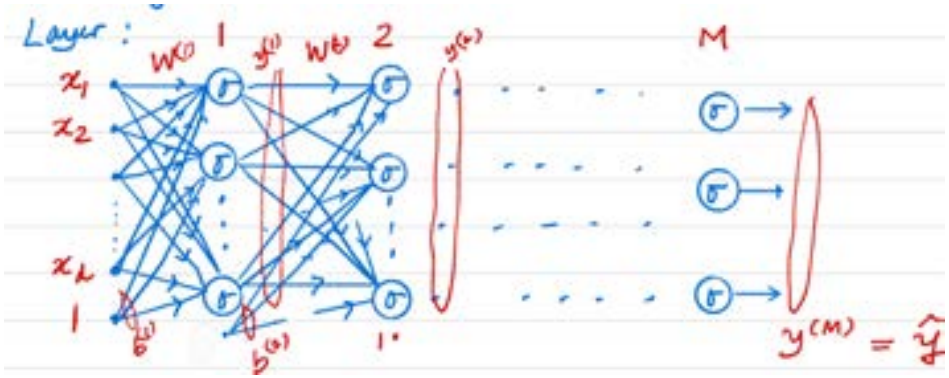
Bias and Boundary

Bias b is important, e.g., if σ is the sign function, we can output +1 if $\omega^T x \geq b$, and -1 if $\omega^T x < b$
$\omega^T x = 0$ is in $\Re^n$, and a neuron simply divides the input space into two parts corresponding to +1 and -1



Multilayer Neural Networks allow us to divide high dimensional space in a less complicated way and approximate functions by mapping inputs and ouputs of NN.

Multilayer Neural Network (Feed Forward)



- Number of neurons in each layer can be different.
- All weights on edge connecting layers m-1 and m are in matrix $W^{(m)}$, with $W_{ij}^{(m)}$ being the weight connecting output j of layer m-1 with neuron i of layer m
- Input to network is vector x; output of layer m is vector $y^{(m)}$
  $y_i^{(1)} = \sigma(x_i^{(1)})$, with $x_i^{(1)} = \sum_j W_{ij}^{(1)} x_j + b_j^{(1)}$
- This can be written compactly as: $y^{(1)} = \sigma(x^{(1)})$, with $x^{(1)} = w^{(1)}x + b^{(1)}$, for σ is applied to each element of $x^{(1)}$

Multilayer Neural Network (Feed Forward)

- $y^{(1)} = \sigma(x^{(1)})$, with $x^{(1)} = w^{(1)}x + b^{(1)}$
- similarly, $y^{(2)} = \sigma(x^{(2)})$, with $x^{(2)} = w^{(2)}y^{(1)} + b^{(2)}$
- .....
- $y^{(M)} = \sigma(x^{(M)})$, with $x^{(M)} = w^{(M)}y^{(M-1)} + b^{(M)}$
- We want the output of NN approximates some desired functions well: $\hat{y} = y^{(M)} \approx f(x) = y$
- How is the function approximated? Use labeled training data, i.e. inputs for which we know the correct outputs, even though f is unknown.
- Minimize empirical loss on training data $J = \sum_{n=1}^{N} L(y[n], \hat{y}[n])$, $\hat{y} = y^{(M)}$

Multilayer Neural Network (Feed Forward)

- In case the desired output y is scalar, then L is often taken to be the square loss: $L(y, \hat{y}) =\mid y - \hat{y} \mid^2$means each layer has only one neuron
- J is a function of W and b
- We wish to minimize J using a gradient descent procedure (assume the loss function is continuous and differentiable)
- Need to compute gradient of J with respect to all weights and biases
- We can compute the gradient for each term separately (partial derivative)
- we need: $\frac{\partial L}{\partial w_{ij}^l}$ and $\frac{\partial L}{\partial b_i^l}$

Back Propagation

- Exploit network structure and Chain Rule to compute gradient terms efficiently
- Layer M Recall $y^{(M)} = \sigma(x^{(M)}), x^{(M)} = w^{(M)} y^{(M-1)} + b^{(M)}$
- $\frac{\partial L}{\partial w_{ij}^{(M)}} = \frac{\partial L}{\partial y_i^{(M)}} * \frac{\partial y_i^{(M)}}{\partial w_{ij}^{(M)}} = \frac{\partial L}{\partial y_i^{(M)}} * \frac{\partial y_i^{(M)}}{\partial x_i^{(M)}} * \frac{\partial x_i^{(M)}}{\partial w_{ij}^{(M)}}$
- $\frac{\partial L}{\partial y_i^{(M)}}$ can easily be computed in practice for each data point (x[n], y[n])
- For example, if $y^{(M)}$ and y are scalar, and we are using square loss, then $\frac{\partial L}{\partial y_i^{(M)}}(y[n], y^{(M)}[n]) = 2(y^{(M)}[n] - y[n])$
- $\frac{\partial y_i^{(M)}}{\partial x_i^{(M)}} = \frac{d\sigma(x_i^{(M)})}{dx_i^{(M)}} = \sigma'(x_i^{(M)})$, assume σ is differentiable
- $x_i^{(M)} = \sum_k w_{(ik)}^{(M)} y_{(k)}^{(M-1)} + b_i^{(M)}$, $\frac{\partial x_i^{(M)}}{\partial w_{ij}^{(M)}} = y_j^{(M-1)}$
- Thus, $\frac{\partial L}{\partial w_{ij}^{(M)}} = \frac{\partial L}{\partial y_i^{(M)}} * \sigma'(x_i^{(M)}) * y_j^{(M-1)}$

Back Propagation

- Similarly, $\frac{\partial L}{\partial b_i^{(M)}} = \frac{\partial L}{\partial y_i^{(M)}} * \frac{\partial y_i^{(M)}}{\partial x_i^{(M)}} * \frac{\partial x_i^{(M)}}{\partial b_i^{(M)}} = \frac{\partial L}{\partial y_i^{(M)}} * \sigma'(x_i^{(M)})$
- Lay m, $1 \leq m \leq M$ Recall
  $y^{(m)} = \sigma(x^{(m)}), x^{(m)} = w^{(m)} y^{(m-1)} + b^{(m)}$
- $\frac{\partial L}{\partial w_{ij}^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} * \frac{\partial y_i^{(m)}}{\partial x_i^{(m)}} * \frac{\partial x_i^{(m)}}{\partial w_{ij}^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} * \sigma'(x_i^{(m)}) * y_j^{(m-1)}$
- $y^{(m+1)} = \sigma(x^{(m+1)}), x^{(m+1)} = w^{(m+1)} y^{(m)} + b^{(m+1)}$
- $\frac{\partial L}{\partial y_i^{(m)}} = \sum_k \frac{\partial L}{\partial y_k^{(m+1)}} * \frac{\partial y_k^{(m+1)}}{\partial x_k^{(m+1)}} * \frac{\partial x_k^{(m+1)}}{\partial y_i^{(m)}} = \sum_k \frac{\partial L}{\partial y_k^{(m+1)}} * \sigma'(x_k^{(m+1)}) * w_{(ki)}^{(m+1)}$
  because m+1 is already computed in the previous step(layer)
- Similarly, $\frac{\partial L}{\partial b_i^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} * \frac{\partial y_i^{(m)}}{\partial x_i^{(m)}} * \frac{\partial x_i^{(m)}}{\partial b_i^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} * \sigma'(x_i^{(m)})$

Back Propagation Summary

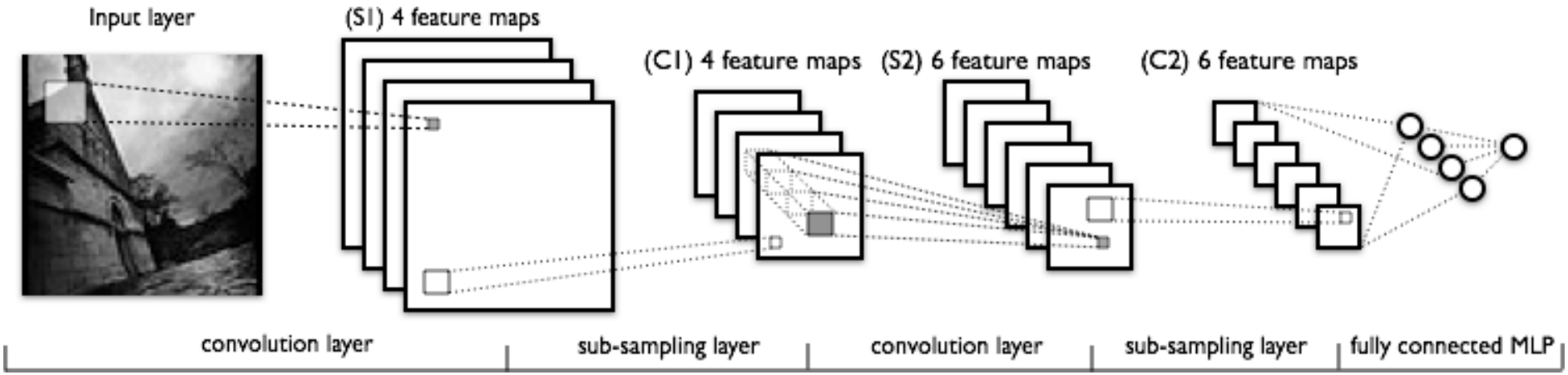
- Compute $\frac{\partial L}{\partial y_i^{(M)}}$ fpr all i
- For m=M down to 1 $\frac{\partial L}{\partial w_{ij}^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} * \sigma'(x_i^{(m)}) * y_j^{(m-1)}$
- $\frac{\partial L}{\partial b_i^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} * \sigma'(x_i^{(m)})$
- where $\frac{\partial L}{\partial y_i^{(m)}} = \sum_k \frac{\partial L}{\partial y_k^{(m+1)}} * \sigma'(x_k^{(m+1)}) * w_{(ki)}^{(m+1)}$

What are the individual functions/layers f_1, f_2 etc.?

- Fully connected layers
- Convolutions
- Rectified linear units (ReLU): $\max\{0, x\}$
- Maximum-/Average pooling
- Soft-max layer
- Dropout

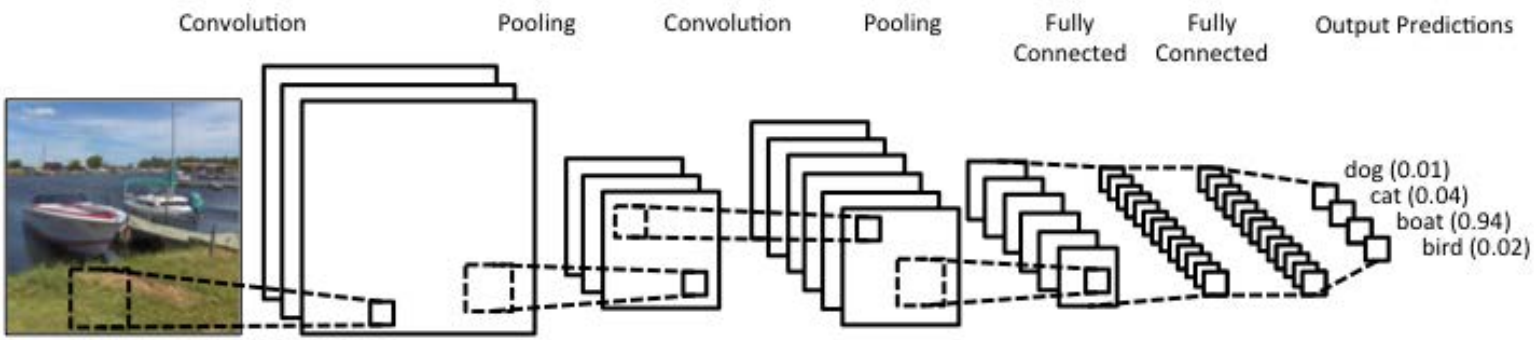
Y. LeCun, 1998

Example function architecture: LeNet



Decreasing spatial resolution and the increasing number of channels

Another deep net:



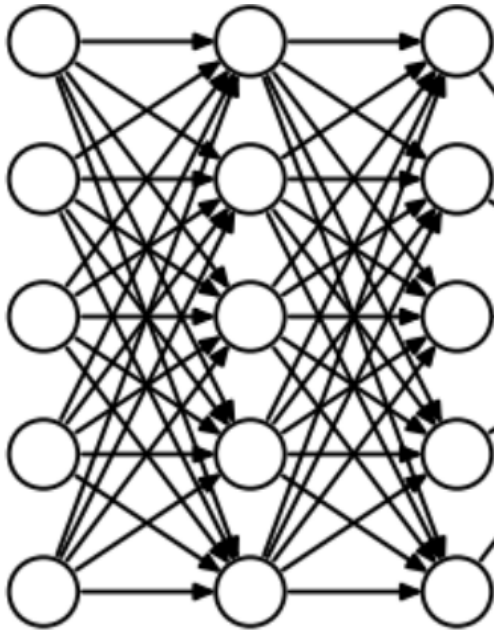
Those nets are structurally simple in that a layer's output is used as input for the next layer. This is not required.

Fully connected layer:

$$\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

Trainable parameters $\boldsymbol{w}$:

- Matrix
- Bias



What's an issue with fully connected layers?

Issue with fully connected layers:

- Suppose the input is an image of size 256×256
- Let the output of this layer have identical size
- How many weights are necessary?

$$2^{32} = 4,294,967,296$$

- How to share weights?

Convolutions:

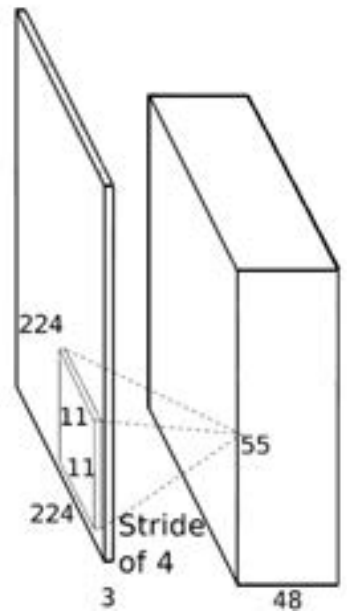
120	190	140	150	200
17	21	30	8	27
89	123	150	73	56
10	178	140	150	18
190	14	76	69	87

x

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

=

	98	98	93	
	84	97	72	
	108	108	91	



Trainable parameters $\boldsymbol{w}$:

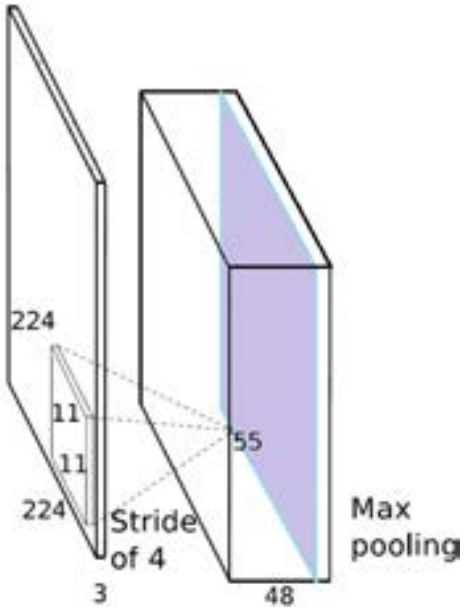
- Filters (width, height, depth, number)
- Bias

Maximum-/Average pooling

Maximum or average over a spatial region

Trainable parameters $\boldsymbol{w}$:

- None



Soft-max layer:

$$x \quad \longrightarrow \quad \frac{\exp x_i}{\sum_j \exp x_j}$$

Trainable parameters $\boldsymbol{w}$:

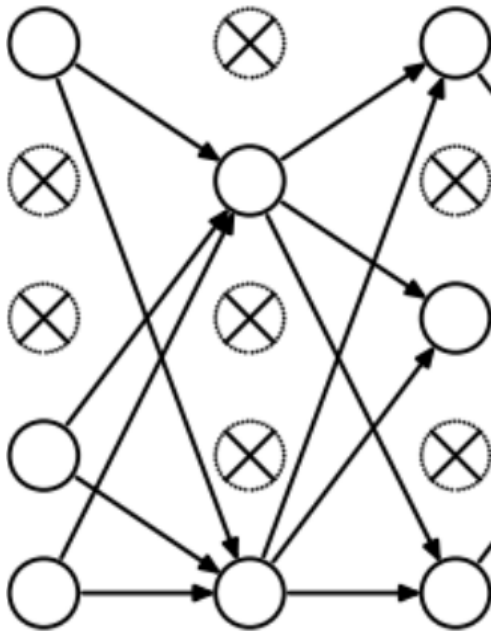
- None

Dropout layer:

Randomly set activations to zero

Trainable parameters $\boldsymbol{w}$:

- None



Recap: Our earlier framework:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)},\hat{y}) + \boldsymbol{w}^T\psi(x^{(i)},\hat{y}))}{\epsilon} - \boldsymbol{w}^T\psi(x^{(i)},y^{(i)}) \right)$$

What is a possible issue/limitation?

Linearity in the feature space $\psi(x,y)$. Fix: use kernels. But still learning a model **linear** in the parameters $\boldsymbol{w}$

How to fix this?

Replace $\boldsymbol{w}^T\psi(x,y)$ with a general function $F(\boldsymbol{w},x,y) \in \mathbb{R}$

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)},\hat{y}) + F(\boldsymbol{w},x^{(i)},\hat{y})}{\epsilon} - F(\boldsymbol{w},x^{(i)},y^{(i)}) \right)$$

Deep net training:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

Often also referred to as minimizing the regularized cross entropy:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 - \sum_{i\in\mathcal{D}} \sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x^{(i)}) \quad \text{with} \quad \begin{cases} p_{\text{GT}}^{(i)}(\hat{y}) & = & \delta(\hat{y} = y^{(i)}) \\ p(\hat{y}|x) & \propto & \exp F(\boldsymbol{w}, x, \hat{y}) \end{cases}$$

What is C? Weight decay (aka regularization constant)

$$\min_{\boldsymbol{w}} \underbrace{\frac{C}{2}\|\boldsymbol{w}\|_2^2}_{\text{weight decay}} \underbrace{- \sum_{i\in\mathcal{D}} \sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x^{(i)})}_{\ell(\text{gt}, F)}$$

Program:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

How to optimize this?

Stochastic gradient descent with momentum: What was this again?

Gradient of

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

is?

$$C\boldsymbol{w} + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} \left(p(\hat{y}|x^{(i)}) - \delta(\hat{y} = y^{(i)}) \right) \frac{\partial F(\boldsymbol{w}, x^{(i)}, \hat{y})}{\partial \boldsymbol{w}}$$

How to compute this numerically:

- $p(\hat{y}|x) = \frac{\exp F(\boldsymbol{w},x,\hat{y})}{\sum_{\tilde{y}} \exp F(\boldsymbol{w},x,\tilde{y})}$ via soft-max which takes logits F as input
- $\frac{\partial F(\boldsymbol{w},x,\hat{y})}{\partial \boldsymbol{w}}$ via backpropagation

Backpropagation example:

$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, x)))$ with activations $\begin{cases} x_2 &= f_3(w_3, x) \\ x_1 &= f_2(w_2, x_2) \end{cases}$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial x_2} \cdot \frac{\partial x_2}{\partial w_3} = \underbrace{\frac{\partial f_1}{\partial f_2}} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial w_3}$$

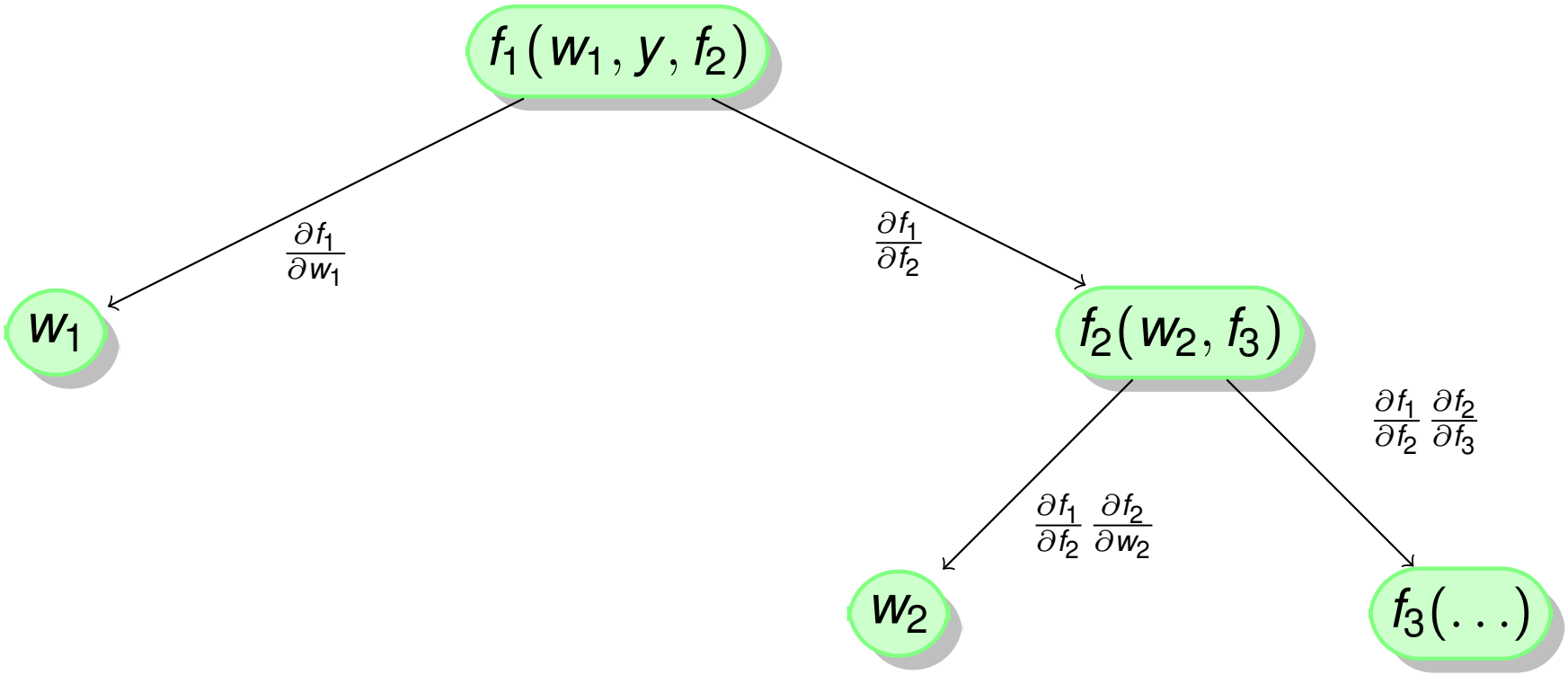
What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_2}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial w_2} = \underbrace{\frac{\partial f_1}{\partial f_2}} \cdot \frac{\partial f_2}{\partial w_2}$$

Generally: To avoid repeated computation, backpropagation on an acyclic graph. Nodes in this graph are weights, data, and functions.

Composite function represented as acyclic graph

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$



Repeated use of chain rule for efficient computation of all gradients

Remark:

Since $F(\boldsymbol{w}, x, y)$ is no longer constrained in any form, the loss function is generally no longer convex.

Implications:

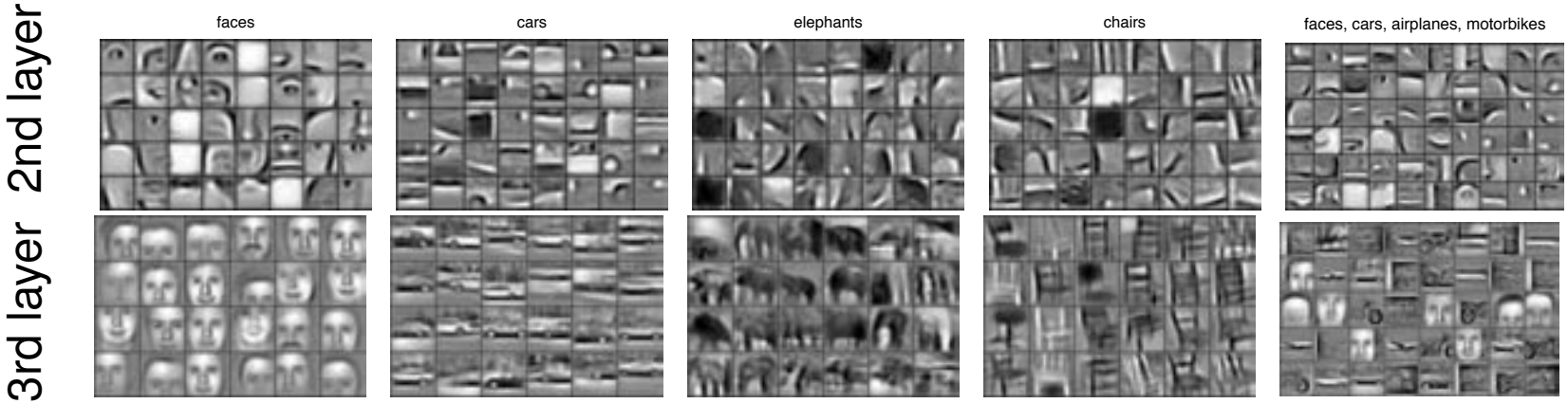
- We are no longer guaranteed to find the global optimum
- Initialization of $\boldsymbol{w}$ matters

Remark:

A deep net with a single fully connected layer is equivalent to logistic regression

Advantages of deep nets compared to usage of hand-crafted features:

Deep nets automatically learn feature space transformations (hierarchical abstractions of data) such that data is easily separable at the output



Disadvantage of deep nets compared to usage of features:

Deep nets are computationally demanding (GPUs) and require significant amounts of training data

Why this recent popularity:

- Sufficient computational resources
- Sufficient data
- Sufficient algorithmic advances
- Sufficient evidence that it works

This combination lead to significant performance improvements on many datasets

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout
  - Decorrelates different units, i.e., they learn different features
- Good initialization heuristics
  - Less prone to getting stuck in bad local optima
- Batch-Normalization during training
  - Normalizes data when training really deep nets
  - Normalize by subtracting mean and dividing by standard deviation