Math Primer & Neural Network Basics

Florian Buettner

buettner.florian@siemens.com

Today‘s lecture: Math Primer & Neural Network Basics

- Intro to supervised learning
- Math primer: probability theory, linear algebra
- Building blocks of basic neural networks
- Multilayer architectures and Non-linearities

Supervised machine learning

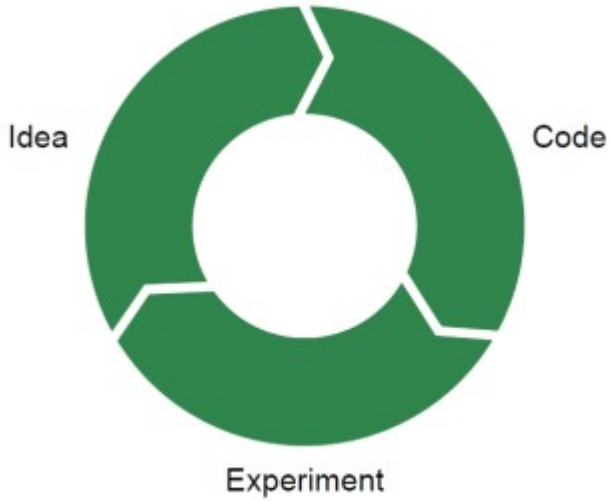
- Learn mapping from input **x** to output *y*, given a labeled set of input-output pairs

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

- When y is categorical
  - Classification
- When y is continuous
  - Regresssion

Four steps of supervised ML

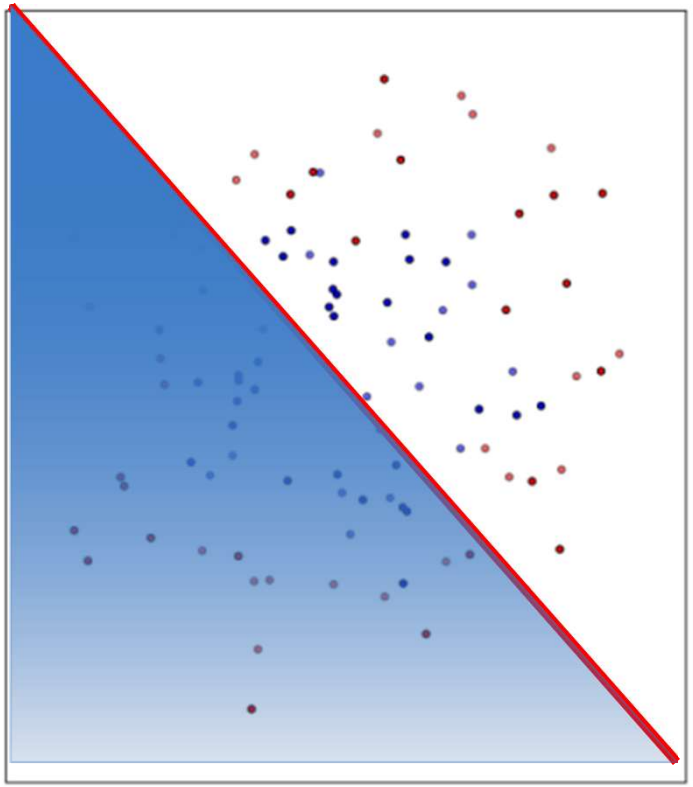
1. Collect data and extract features
2. Build model: choose model class M and loss function l
3. Optimization: minimize the empirical loss
4. Evaluate model performance on independent test data



Overview

- Supervised machine learning techniques
  - Classification
    - Random forest
    - SVM
    - Neural networks
- Evaluation

Linear and non-linear classifiers

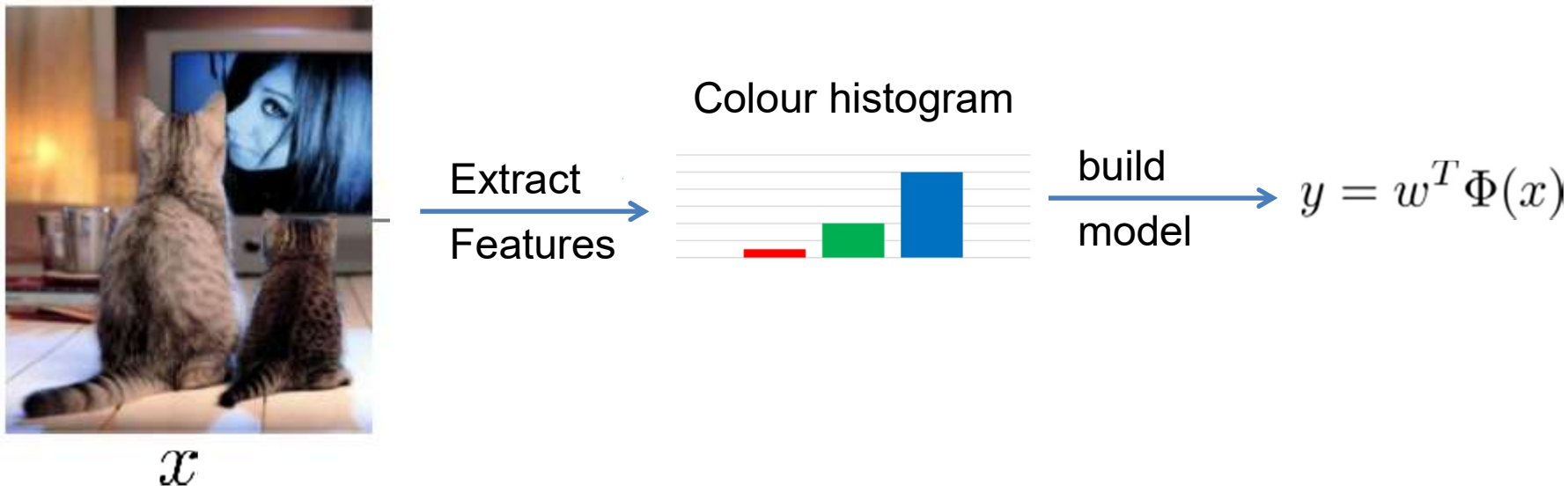


Linear and non-linear classifiers

- Opportunities
  - Can resolve complex interactions between inputs
  - Potentially higher predictive power than linear classifier
- Challenges
  - Hard to fit, easy to overfit
  - Hard to interpret (“black box classifier”)

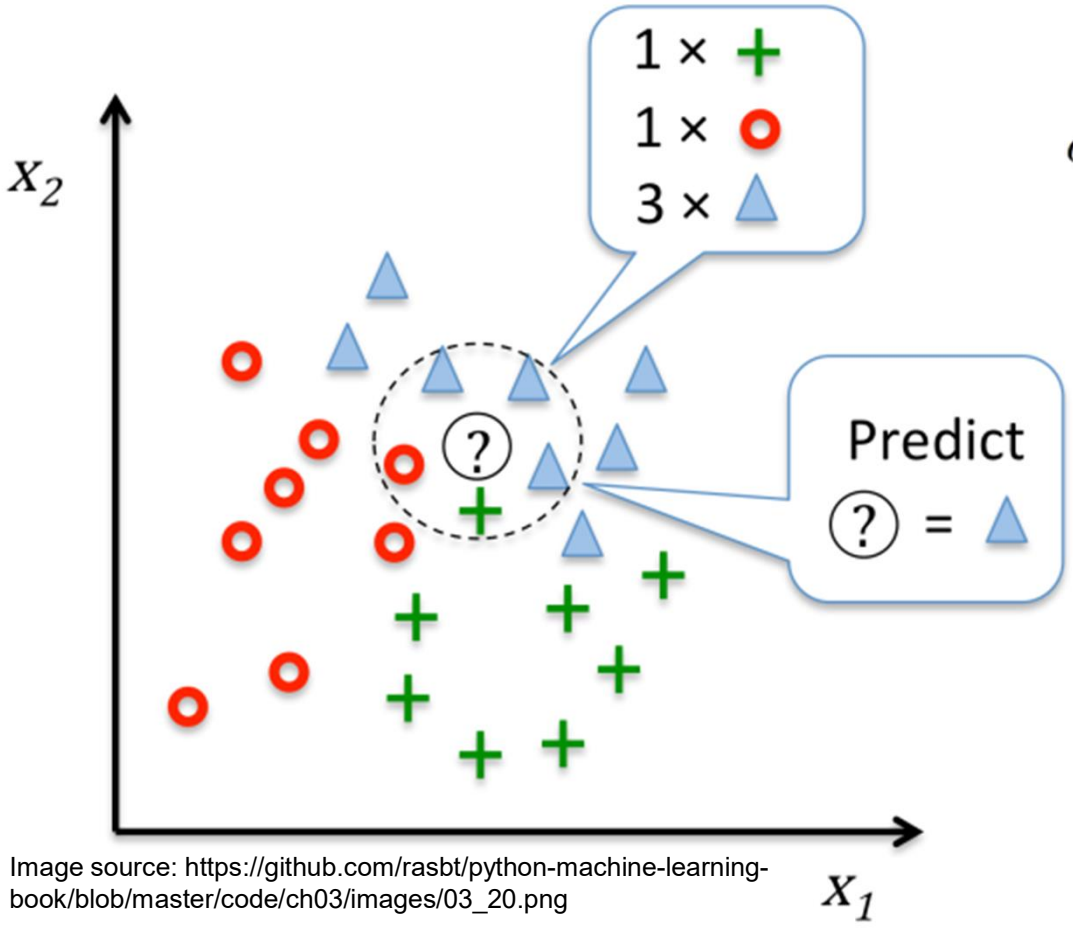
Features

- Statistical features (histograms, moments, …)
- Domain-specifc features (SiFT features, Fourier coefficients, …)



K-Nearest Neighbors

k=5

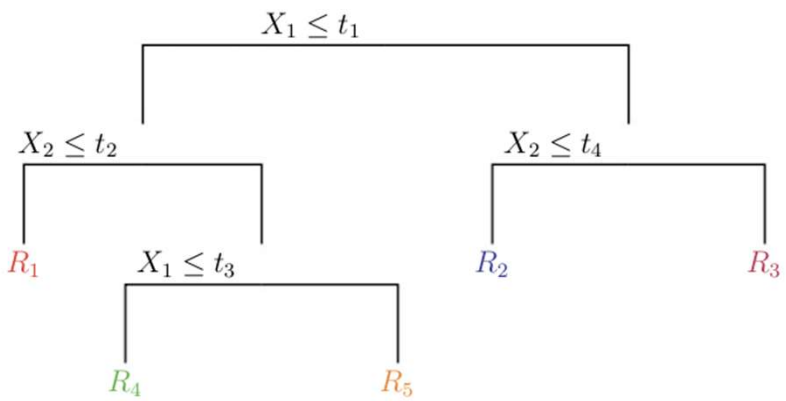


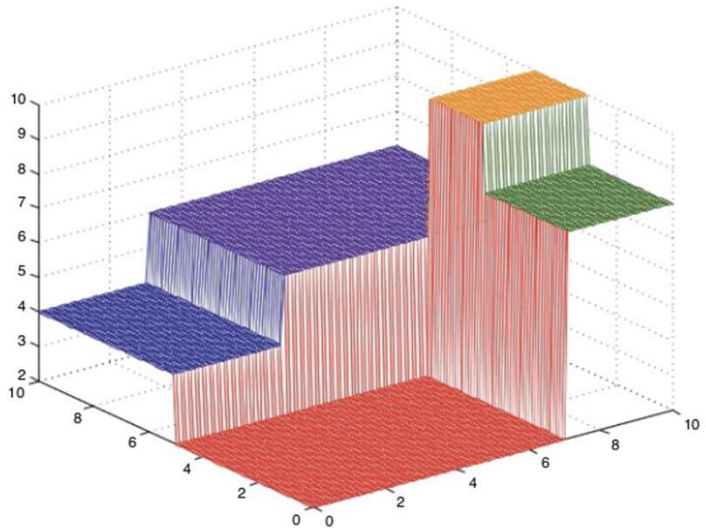
$$d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sqrt[p]{\sum_k |x_k^{(i)} - x_k^{(j)}|^p}$$

Image source: https://github.com/rasbt/python-machine-learning-book/blob/master/code/ch03/images/03_20.png

Decision trees

- Decision trees
  - Recursively partition input
  - Use greedy approaches to find locally optimal MLE
  - Prune back to avoid overfitting



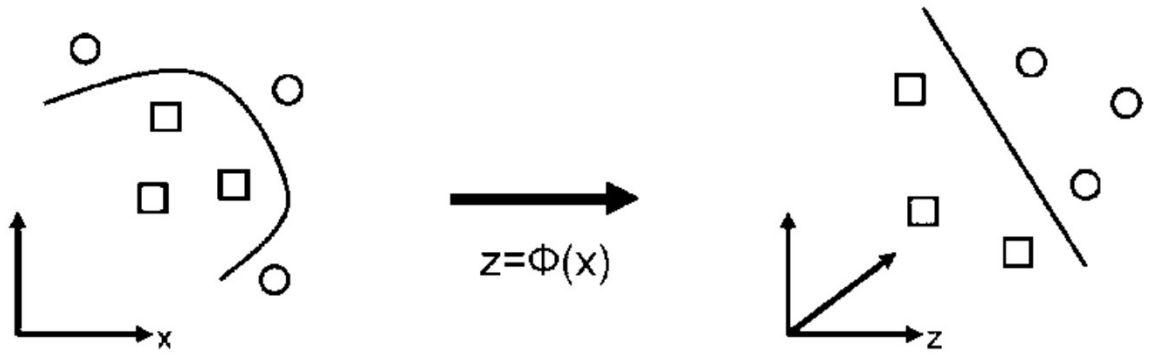


Random forest

- Cons
  - Poor classification power
  - Descision trees instable
    - Small errors on top can have big effect
      - High variance estimators
- Random forests
  - Reduce variance by averaging many estimates (“bagging”)
  - Decorrelate base learners by subsampling samples and input variables
  - Fast, interpretable, high predictive power

Support vector machines

- Use a kernel to map the data in a high-dimensional transformed feature space such that the classes can be separated linearly



Feature extraction and feature selection

- Image features
- Clinical factors
- Environmental factors
- Genetic factors
- But: can yield very large number of predictors

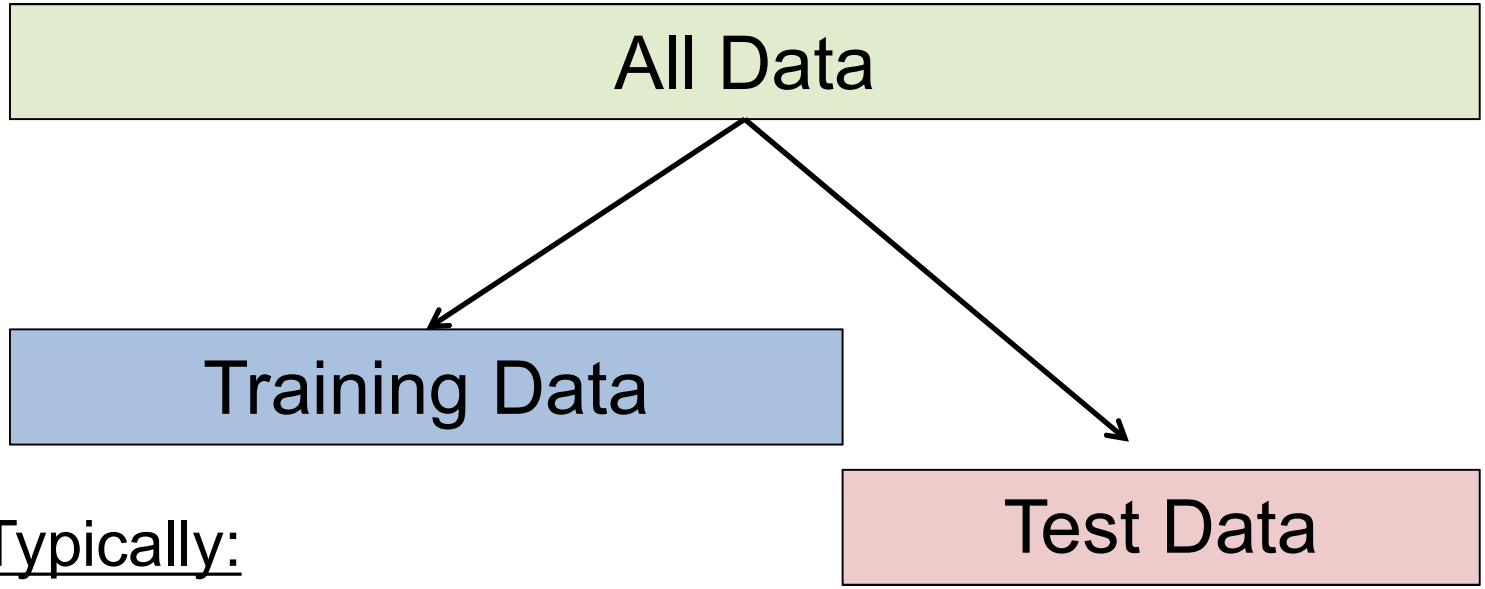
Feature selection

- Evaluate relevance of each feature separately and retain only top K features
  - Feature ranking, e.g. ANOVA, F-test
- Recursive feature elimination
  - Start with all feature in the model and recursiveley eliminate the ones not needed
- Bayesian variable selection
  - View model as whole and treat number of variables as additional parameter
  - Calculate probability of being the best model for all potential models given the data
  - Determine marginal probabilities that a variable should be in the model

A caveat

- Common methodological mistake in supervised machine learning
  - Learning the parameters of a prediction function and testing it on the same data
- Hold out part of the available data as a test set
  - Make sure test set does not “leak” into training
    - hyperparameters are optimised on separate validation set
    - Perform feature selection without looking at test data
    - Perform normalisation steps (standardisation etc) separately
- K-fold cross-validation
  - Split data in k folds
  - model is trained using k-1 of the folds as training data
  - Test models on kth fold

Training & Test Data



Typically:

- 75% : 25%
- Use stratification
- Consider cross-validation
  - 10-fold CV
  - Leave-one-out CV

Image source: py-con

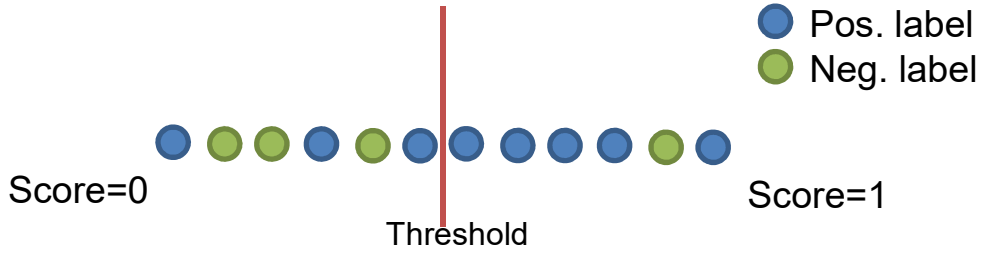
Evaluation and performance metrics

- Training objective (cost function) is only a proxy for real world objective.
  - Metrics help capture a business/diagnostic goal into a quantitative target (not all errors are equal).
- Useful to quantify the “gap” between:
  - Desired performance and baseline (estimate effort initially).
  - Desired performance and current performance.
- Useful for lower level tasks and debugging (like diagnosing bias vs variance)

Binary classifiers

- Two types of models
  - Models that output a categorical class directly (K Nearest neighbor, Decision tree)
  - Models that output a real valued score (SVM, Logistic Regression, NN)
    - Score could be margin (SVM), probability (LR)
    - Need to pick a threshold
    - We focus on this type

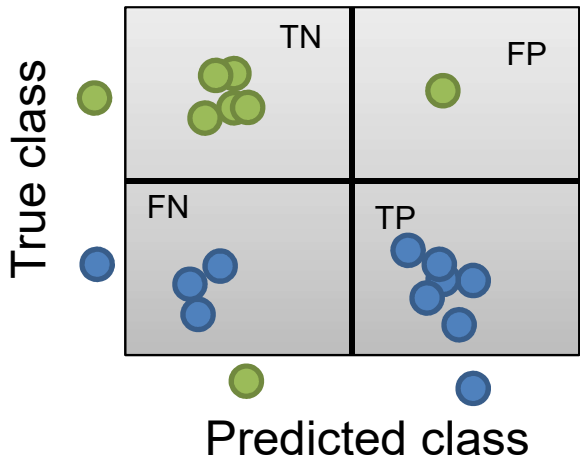
Score-based models

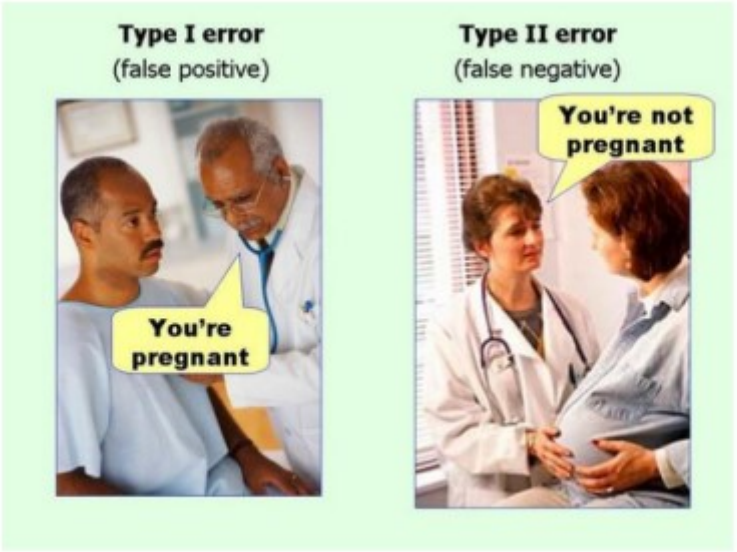


- For most metrics only ranks matter
- Set threshold to get classification
- Prevalence: (#pos. Examples)/(total # examples)
  - Class imbalance

Point-based metrics

- After thresholding, compute point-based metrics
  - Confusion matrix
    - TP, FP, TN, FN
    - Type I error, Type II error





Could not find true source of image to cite

Summary point metrics

- Accuracy: What overall fraction did we predict correctly?

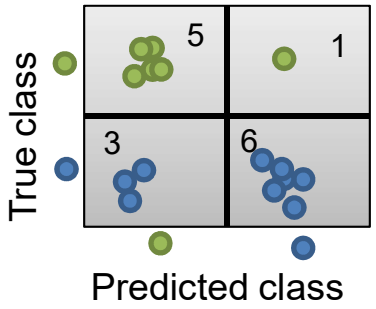
  Acc: (5+6)/(5+1+3+6) = 0.73

- Precision: Quality of positive predictions (how many are relevant?)

  Prec: 6/(1+6) = 0.86

- Recall (sensitivity): How many positives are identified? (How sensitive is the model for predicting disease?)

  Rec: 6/(3+6) = 0.66



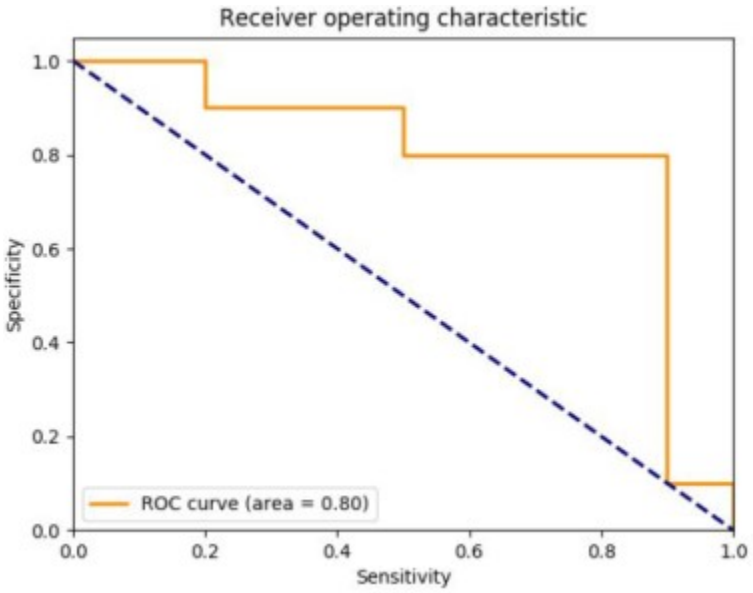
- Negative Recall (specificity): Proportion of actual negatives that are correctly identified as such? (percentage of healthy people who are correctly identified as not having the condition) spec: 5/(5+1) = 0.83
- F1 score: harmonic mean of rec and prec

Changing the threshold

- Depending on our actual (business/dignostic) goal, we can change the threshold to change precison/recall etc
- Scan through all thresholds and summarize tradeoff

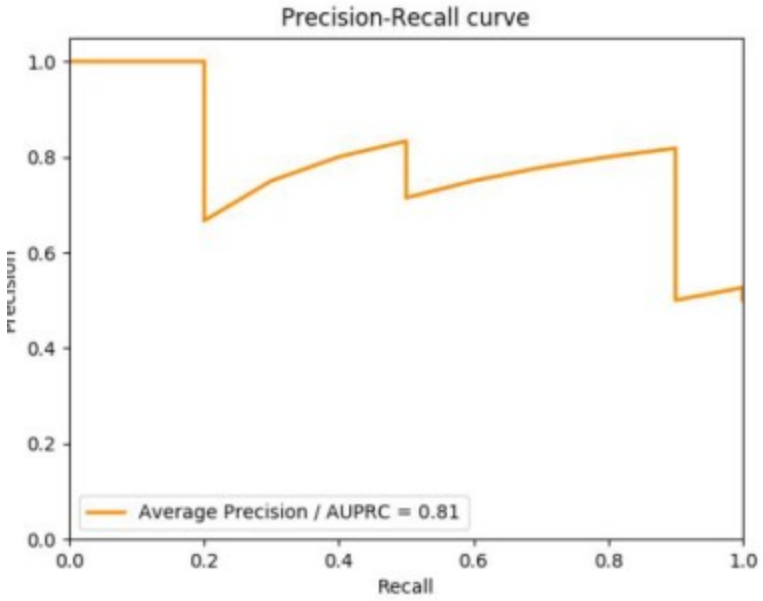
{Precision, Specificity} vs Recall/Sensitivity

ROC curve



- AUC = Area Under Curve. Also called C-Statistic (concordance score).
  - Represents how well the results are ranked.
- Thresholds are points on this curve. Each point represents one set of point metrics.
- Diagonal line = random guessing

PR curve



- Represents different tradeoff
  - More meaningful if TNs are not so important or low prevalence of relevant class (rare disease, search engine)
- Area under PRC = Average Precision
- End of curve at right cannot be lower than prevalence.
- Jaggedness (esp. for small sample sizes)
  - Sequence of positives: increase rec and prec – slow climb
  - Sequence of negatives: precision decreases, recall doesn't change – steep drop

Class imbalance

- For low prevalence (e.g. < 5%) many metrics are not meaningful (e.g. accuracy of 95% is trivial to achieve)
- Focus on PR and REC
  - High precision is required (search engine)
  - High recall is required (fraud detection)

Diagnostics

- Setting: test metrics of classifier are unacceptably bad
- Diagnostic:
  - High variance: Training loss will be much lower than test loss
  - High bias: Training loss will also be high
- Root cause
  - Overfitting (high variance)
  - Insuffcient information in data (e.g. bad features)

High variance

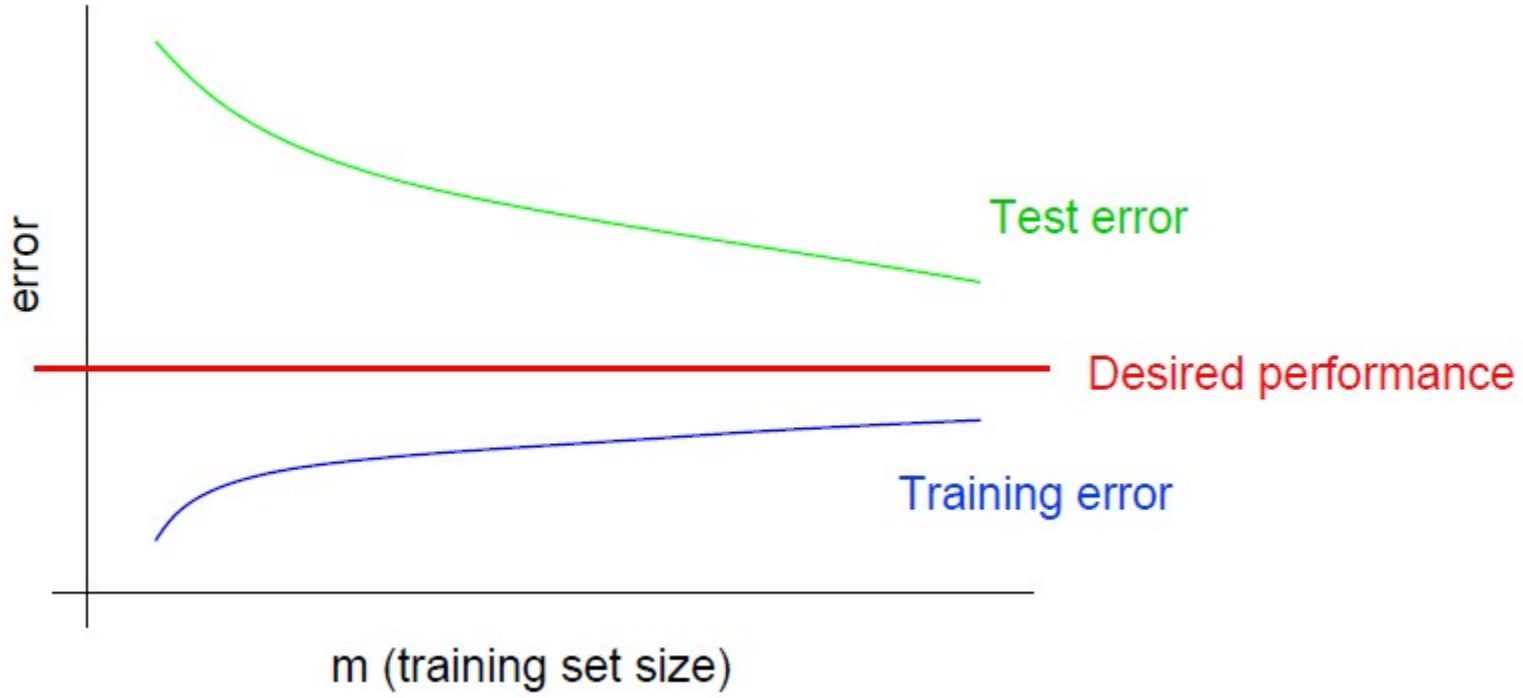


Image by Andrew Ng

High bias

Image by Andrew Ng

Fixes

- Increase number of training examples.
- Increase size of feature set
- Decrease size of feature set
- Train model for longer (more gradient steps)
- Tune model hyperparameters on validation set
- Try more complex model
- Try simpler model
- More reading
  - „Advice for applying Machine Learning” (Andrew Ng)
    - http://cs229.stanford.edu/materials/ML-advice.pdf

Math primer – probabilistic machine learning

Bayes' theorem

Everything follows from two simple rules:

Sum rule: $P(x) = \sum_y P(x, y)$

Product rule: $P(x, y) = P(x)P(y|x)$

$$P(\theta|\mathcal{D}, m) = \frac{P(\mathcal{D}|\theta, m)P(\theta|m)}{P(\mathcal{D}|m)}$$

$P(\mathcal{D}\|\theta, m)$	likelihood of parameters θ in model m
$P(\theta\|m)$	prior probability of θ
$P(\theta\|\mathcal{D}, m)$	posterior of θ given data $\mathcal{D}$

Slide adapted byZoubin Gharamani

Example for Bayes' theorem

- Exercise: Cancer Screening Example (~10 min)
- Mammograms:
  - Sensitivity: 80%
  - False Positive rate: 10%
  - Prevalence: 0.4%
- Q: Use Bayes Theorem to calculate the probability that you have cancer if you test positive!

$$P(y|x,H) = \frac{P(x|y,H)P(y|H)}{\sum_{y'} P(x|y',H)P(y'|H)}.$$

$$P(y \mid x, H) = \frac{P(x \mid y, H)P(y \mid H)}{\sum_{y'} P(x \mid y', H)P(y' \mid H)}.$$

Solution

Sensitivity: 80%
False Positive rate: 10%
Prevalence: 0.4%

- x=1: mammogram is positive
- y=1: you have cancer

$$p(x=1 \mid y=1) = 0.8$$

$$p(y=1) = 0.004$$

$$p(x=1 \mid y=0) = 0.1$$

$$P(y=1 \mid x=1) = \frac{p(x=1 \mid y=1)p(y=1)}{p(x=1 \mid y=1)p(y=1) + p(x=1 \mid y=0)p(y=0)}$$

Answer: The probability that you have cancer if you test positive is 0.031!

The Meaning of Probability

- Often used in two ways:
- 1st usage: Probabilities describe *frequencies* of outcomes in *random* experiments
  - Hard to give non-circular definitions of "frequency" and "random"
- 2nd (more general) usage: Probabilities describe *degrees of belief*
  - "probability that the email you just received is spam"
  - "probability that Oscar P. murdered his girlfriend, given the evidence"

Bayesian vs. frequentist viewpoint

- Frequentist:
  - Probabilities are restricted to frequencies in repeatable random experiments
- But: *Degrees of belief* can be mapped to probabilities (if they follow some rules of consistency: Cox Axioms)
- Bayesian viewpoint: Use probabilities to describe assumptions and inferences given those assumptions
  - Probabilities depend on assumptions
  - Bayesians: you cannot do inference without assumptions
- Bayesians: use probabilities to describe inferences
- Frequentist: use probabilities to describe random variables

Bayes' theorem - some terminology

- Common scenario:
  - Infer parameter theta given some data D: $P(\theta|D,H)$

$$P(\theta|D,H) = \frac{P(D|\theta,H)P(\theta|H)}{P(D|H)}$$

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}$$

Continuous random variables

- So far: discrete events/random variables
- Let *X* be some *uncertain*, *continuous* quantity
- Aim: Compute probability that $a \leq X \leq b$
  - Define event $A=(X \leq a)$, $B=(X \leq b)$, $W=(a<X \leq b)$
  - Then $p(B)=p(A)+p(W)$
  - $P(W)=p(B)-p(a)$
- Let $F(q)=p(X \leq q)$ be the *cumulative distribution function*: $p(a<X \leq b)=F(b)-F(a)$
- Define probability density function $f(x)=\frac{d}{dx}F(x)$

Working with continuous random variables

- Compute probability of a continuous variable being in a finite interval given a pdf f(x)

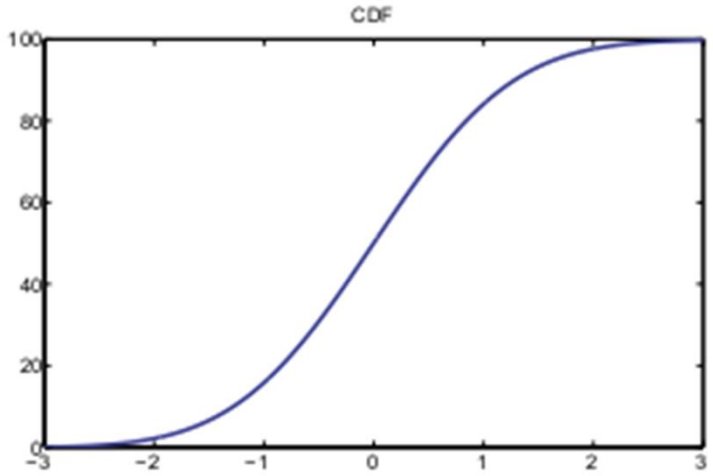
$$P(a < X \leq b) = \int_a^b f(x)\,dx$$

- Consequently, for small intervals:

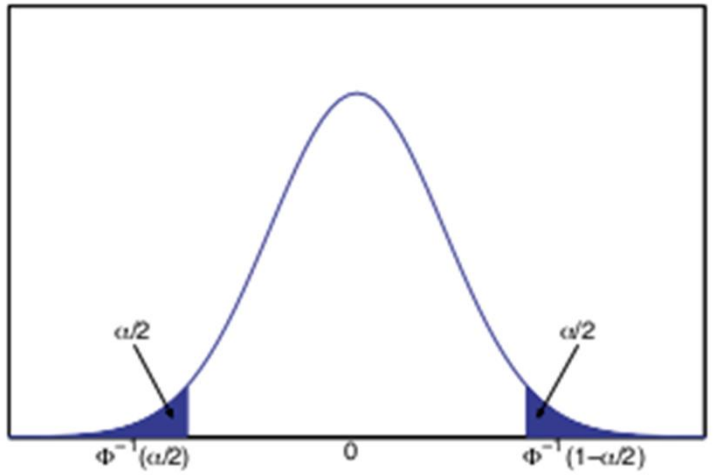
$$P(x \leq X \leq x + dx) \approx p(x)dx$$

- Note: f(x) needs to be positive but can be greater than 1 if it integrates to 1
- Uniform distribution: $Unif(x \mid a.b) = \frac{1}{b-a} I(a \leq x \leq b)$

Example



(a)



(b)

Some distributions

- Useful discrete distributions:
  - Binomial distribution:

$$Bin(k \mid n, \theta) := \binom{n}{k} \theta^k (1-\theta)^{n-k}$$

  - Bernoulli distribution: Special case of Binomial with n=1
  - Poisson distribution: $Poi(x \mid \lambda) = e^{-\lambda} \frac{\lambda}{x!}$

Continuous distributions

- Gaussian (normal) distribution
- pdf:

$$N(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$$

- cdf:

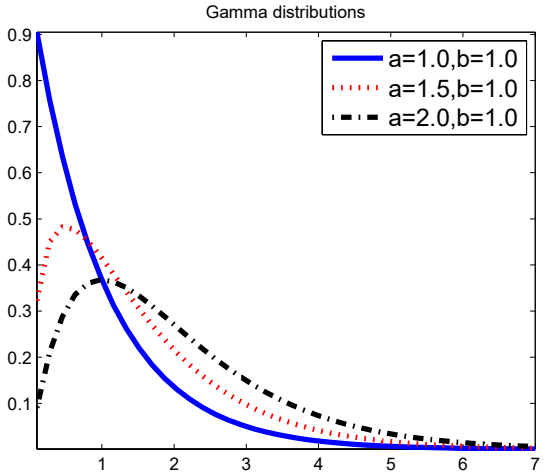
$$\phi(x; \mu, \sigma^2) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(z-\mu)^2} dz = \frac{1}{2}[1 + erf(z/\sqrt{2})]$$

- Most important distribution in stats/ML
  - Easy to interpret
  - Central limit theorem
  - simple mathematical form allows for effective inference methods

Continuous distribution with limited support

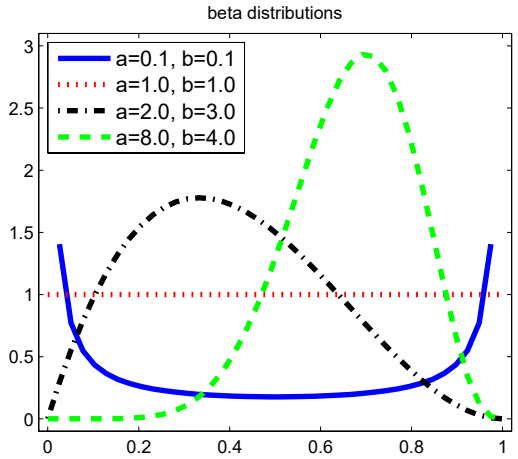
- Gamma distribution

$$f(x;k,\theta)=\frac{1}{\theta^k}\frac{1}{\Gamma(k)}x^{k-1}e^{-\frac{x}{\theta}}$$



- Beta distribution

$$f(x;\alpha,\beta)=\frac{1}{B(\alpha,\beta)}x^{\alpha-1}(1-x)^{\beta-1}$$



Example – Beta-Bernoulli model

- Toss a coin N times, obtain sequence of heads (N_1) and tails
- Questions:
  - What is the bias θ of the coin (fair coin: θ=0.5)?
  - What's the probability that the next toss will be head?
- In Bayesian terms:
  - What is the posterior $p(\theta \mid D)$?
- How to infer the posterior?

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}$$

Likelihood

- Data: N_1 heads in N trials

$$p(D \mid \theta) = \theta^{N_1}(1-\theta)^{N-N_1}$$

$$N_1 \sim Bin(N, \theta)$$

Prior

- Need prior with support over interval [0,1]
- If possible, same form as likelihood (makes maths easy)
  - Conjugate prior
  - Here: beta distribution!

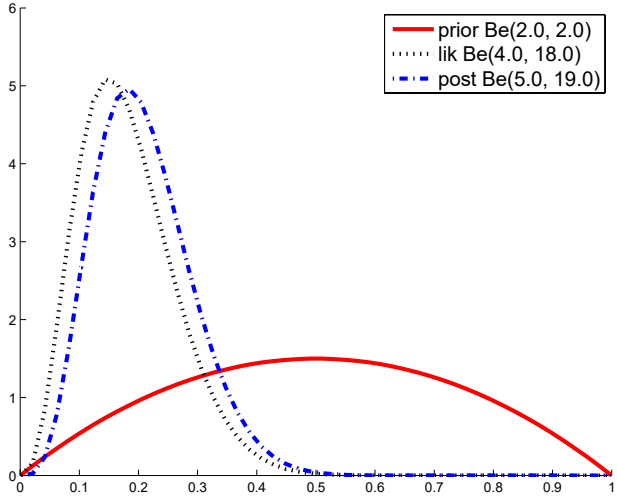
$$p(\theta) \propto \theta^{\gamma_1}(1-\theta)^{\gamma_2}$$

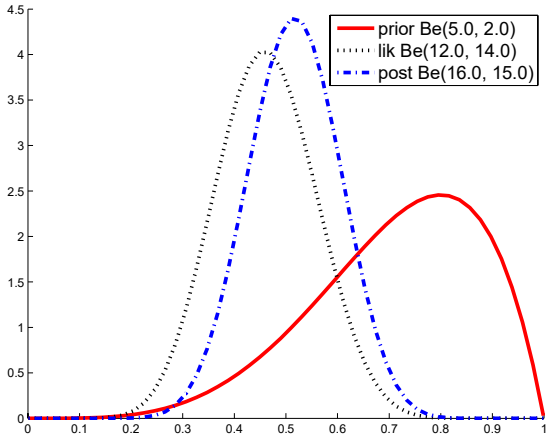
  - a and b are hyper-parameters - they encode our prior beliefs

$$Beta(\theta; a, b) = \frac{1}{B(a,b)}\theta^{a-1}(1-\theta)^{b-1}$$

Posterior

$$p(\theta \mid D) \propto Bin(N_1 \mid N, \theta) Beta(\theta \mid a, b) \propto Beta(\theta \mid N_1 + a, N_0 + b)$$





MAP and MLE

- MAP is mode of posterior
  - mode of Beta distribution with (a,b) is a-1/(a+b-2)
- If uniform prior is used MAP=MLE
- Mean of posterior is a/(a+b)

Linear algebra primer

- Provides a way of compactly representing and operating on sets of linear equations
- Example set of equations:

$$\begin{aligned} 4x_1 &- 5x_2 = -13 \\ -2x_1 &+ 3x_2 = 9. \end{aligned}$$

- With matrix notation:

$$Ax = b$$

$$A = \begin{bmatrix} 4 & -5 \\ -2 & 3 \end{bmatrix}, \quad b = \begin{bmatrix} -13 \\ 9 \end{bmatrix}.$$

Some concepts you should be familiar with

- Scalars, Vectors, Matrices and Tensors
- Multiplying Matrices and Vectors
- Identity and Inverse Matrices
- Eigendecomposition
- Singular value decomposition
- The Moore Penrose pseudo-inverse
- The trace operator
- The determinant
- You will need this in the 2nd tutorial, make sure to revise if needed

Eigendecomposition

- Factorization of a matrix such that it is represented in terms of its eigenvalues and eigenvectors
- Eigenvector **v** of square matrix **A**

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$$

- Each eigenvector has ist own equation

$$(\mathbf{A} - \lambda_i\mathbf{I})\,\mathbf{v} = 0$$

Eigendecomposition

- Factorise A as

$$\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}$$

- **Q** is NxN matirx, with columns being eigenvectors
- Lambda is diagonal matrix with eigenvalues on the diagonal

Eigendecomposition – fun facts

- Only diagonisable matrices can be eigendecomposed
- Real symmetric matrices can be decomposed so that EVs are orthogonal
- Useful for matrix inversion
  - A is invertible iff all EVs are non-zero
  - $\mathbf{A}^{-1} = \mathbf{Q}\mathbf{\Lambda}^{-1}\mathbf{Q}^{-1}$

Matrix calculus and Gradient

- Extension of calculus to the vector setting
- Let **f** be a function that takes as input a matrix *A* of size $m \times n$ and returns a scalar. Then the gradient of **f** (with respect to *A*) is the matrix of partial derivatives

$$\nabla_A f(A) \in \mathbb{R}^{m \times n} = \begin{bmatrix} \frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{12}} & \cdots & \frac{\partial f(A)}{\partial A_{1n}} \\ \frac{\partial f(A)}{\partial A_{21}} & \frac{\partial f(A)}{\partial A_{22}} & \cdots & \frac{\partial f(A)}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(A)}{\partial A_{m1}} & \frac{\partial f(A)}{\partial A_{m2}} & \cdots & \frac{\partial f(A)}{\partial A_{mn}} \end{bmatrix}$$

- As for derivatives, linearity, product rule and chain rule hold
  - $\nabla_x(f(x) + g(x)) = \nabla_x f(x) + \nabla_x g(x)$.
  - For $t \in \mathbb{R}$, $\nabla_x(t\ f(x)) = t\nabla_x f(x)$.

Hessian

- If gradient is the analogue of the first derivative for functions of vectors, the Hessian is the analogue of the second derivative

$$\nabla_x^2 f(x) \in \mathbb{R}^{n \times n} = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}.$$

- Some useful rules:

  - $\nabla_x b^T x = b$
  - $\nabla_x x^T A x = 2Ax$ (if A symmetric)
  - $\nabla_x^2 x^T A x = 2A$ (if A symmetric)

Jacobian

- Generalises gradient to functions that return vector
- Let $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ be a function which takes as input the vector $\mathbf{x} \in \mathbb{R}^n$ and returns as output the vector $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^m$. Then the Jacobian matrix $\mathbf{J}$ of $\mathbf{f}$ is an $m \times n$ matrix:

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

- Useful for transformations and variable changes
- Determinant at a given point gives important information about the behavior of **f** near that point
- If m is 1, Jacobian is transposed of gradient
- Hessian is Jacobian of gradient

Multivariate Gaussian

- Pdf of MVN in D dimensions

$$N(x|\mu, \mathbf{\Sigma}) \triangleq \frac{1}{(2\pi)^{D/2}|\mathbf{\Sigma}|^{1/2}} \exp\left[-1/2(\mathbf{x}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)\right]$$

- Covariance matrix: $\mathbf{\Sigma}$
- Mean vector: μ
- Eigendecomposition of $\mathbf{\Sigma}$: $\mathbf{\Sigma}^{-1} = \mathbf{U}^{-T}\mathbf{\Lambda}^{-1}\mathbf{U}^T = \sum_{i=1}^{D} \frac{1}{\lambda_i}\mathbf{u_i}\mathbf{u_i}^T$
- Mahalonobis distance: $(\mathbf{x}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu) = \sum_{i=1}^{D} \frac{y_i^2}{\lambda_i}$

$$y_i = \mathbf{u}_i^T(\mathbf{x}-\mu)$$

Application: Linear regression

- Model the response as a linear function of inputs

$$\mathbf{y} = \mathbf{w}^T \mathbf{x} + \epsilon = \sum_{j=1}^{D} w_j x_j + \epsilon$$

- Noise is normally distributed

$$p(y|\mathbf{x}, \theta) = \mathcal{N}\left(y|\mu(\mathbf{x}), \sigma^2(\mathbf{x})\right)$$

$$\mu(x) = \mathbf{w}^T \mathbf{x} = w_0 + w_1 x$$

Source: Machine Learning – A Probabilistic Perspective, KP Murphy

How to fit the model?

- Maximum likelihood
  - Common assumption: samples are independent and identically distributed (iid)
  - Minimize negative log likelihood
  - Minimize residuals

$$NLL(\theta) := -\log p(D|\theta) = -\sum_{i=1}^{N} \log p(y_i|x_i, \theta)$$

$$NLL(\theta) = -\sum_{i=1}^{N} \log \left[\left(\frac{1}{2\pi\sigma^2} \right) \exp\left(-1/2\sigma^2 (y_i - \mathbf{w}^T \mathbf{x}_i)^2 \right) \right]$$

$$\hat{\mathbf{w}}_{OLS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

What about the prior?

- We can put a normal prior on w
- Then use Bayes rule for Gaussians to compute the posterior

$$
\begin{aligned}
p(\mathbf{w}|\mathbf{X},\mathbf{y},\sigma^2) &\propto \mathcal{N}(\mathbf{w}|\mathbf{w}_0,\mathbf{V}_0)\mathcal{N}(\mathbf{y}|\mathbf{X}\mathbf{w},\sigma^2\mathbf{I}_N) = \mathcal{N}(\mathbf{w}|\mathbf{w}_N,\mathbf{V}_N) \\
\mathbf{w}_N &= \mathbf{V}_N\mathbf{V}_0^{-1}\mathbf{w}_0 + \frac{1}{\sigma^2}\mathbf{V}_N\mathbf{X}^T\mathbf{y} \\
\mathbf{V}_N^{-1} &= \mathbf{V}_0^{-1} + \frac{1}{\sigma^2}\mathbf{X}^T\mathbf{X} \\
\mathbf{V}_N &= \sigma^2(\sigma^2\mathbf{V}_0^{-1} + \mathbf{X}^T\mathbf{X})^{-1}
\end{aligned}
$$

How to implement it all?

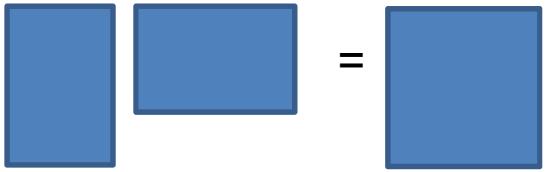
- Deep learning
  - **Tensorflow**
    - Keras
  - Theano
  - Caffe2
  - pyTorch
  - CNTK
- Linear Algebra
  - **NumPy**

Tensorflow vs NumPy

- NumPy
  - Library supporting
    - Multi-dimensional arrays and matrices
    - Large collection of high-level mathematical functions to operate on these arrays
- Tensorflow
  - Deep learning library open sourced by google
  - Provides primitives for defining functions on tensors
  - Automatically computes derivatives
  - GPU support

NumPy recap

```
In [1]: import numpy as np
        import tensorflow as tf

In [2]: A = np.ones((3,2))

In [3]: print(A)
        A.shape

        [[1. 1.]
         [1. 1.]
         [1. 1.]]

Out[3]: (3, 2)

In [4]: np.sum(A,1)

Out[4]: array([2., 2., 2.])

In [5]: B = np.ones((2,3))*2
        B.shape
        B

Out[5]: array([[2., 2., 2.],
               [2., 2., 2.]])

In [6]: np.dot(A,B)

Out[6]: array([[4., 4., 4.],
               [4., 4., 4.],
               [4., 4., 4.]])
```



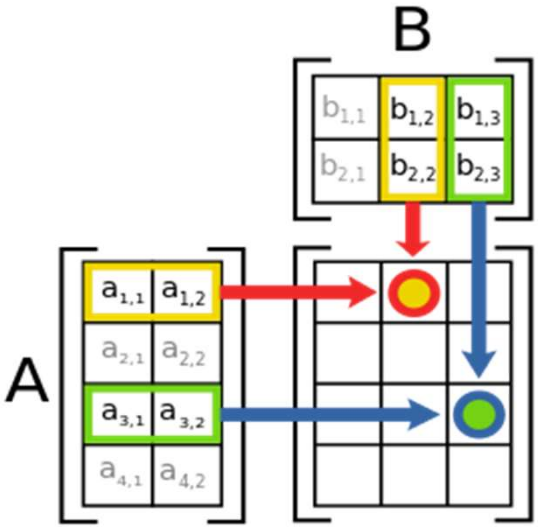
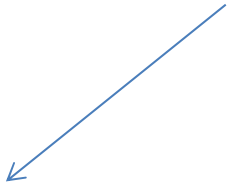


Image: https://en.wikipedia.org/wiki/Matrix_multiplication#/media/File:Matrix_multiplication_diagram_2.svg

Tensorflow

```
In [7]: tf.InteractiveSession()

Out[7]: <tensorflow.python.client.session.InteractiveSes

In [8]: A = tf.ones((3,2))

In [9]: print(A)
        A.get_shape()

        Tensor("ones:0", shape=(3, 2), dtype=float32)

Out[9]: TensorShape([Dimension(3), Dimension(2)])

In [10]: print(A.eval())

         [[1. 1.]
          [1. 1.]
          [1. 1.]]

In [11]: tf.reduce_sum(A, reduction_indices=1).eval()

Out[11]: array([2., 2., 2.], dtype=float32)

In [12]: B = tf.ones((2,3))*2
         B.get_shape()

Out[12]: TensorShape([Dimension(2), Dimension(3)])

In [13]: tf.matmul(A,B).eval()

Out[13]: array([[4., 4., 4.],
                [4., 4., 4.],
                [4., 4., 4.]], dtype=float32)
```

Interactive session keeps default session open



TensorFlow computations define a computation graph that has no numerical value until evaluated!

Matrix multiplication

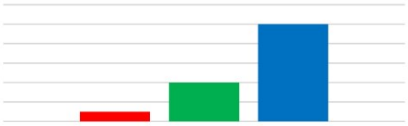
BREAK

Representation learning

x

Extract Features

Color histogram



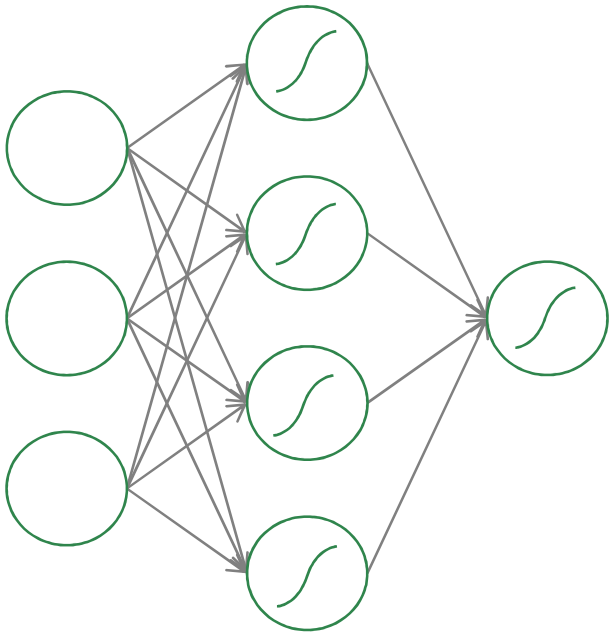
build model

$y = w^T \Phi(x)$

Why not learn $\Phi(x)$?

Feed-forward networks

View each dimension of $\Phi(x)$ as something that has to be learnt

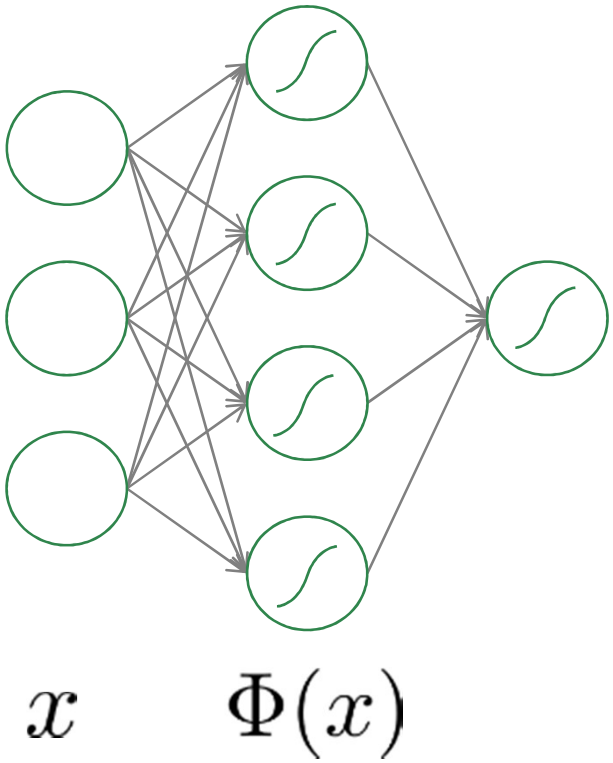


x $\Phi(x)$

$$y = w^T \Phi(x)$$

Feed-forward networks

Linear functions Phi don‘t work – we need **non-linearities**

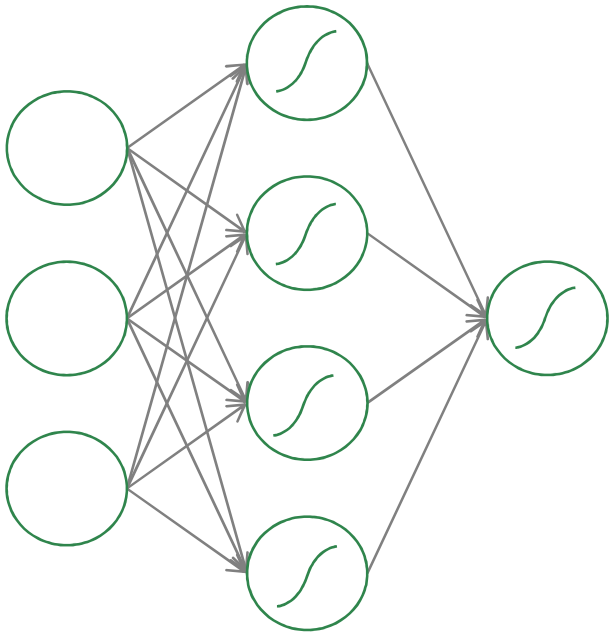


x $\Phi(x)$

$$y = w^T \Phi(x)$$

Feed-forward networks

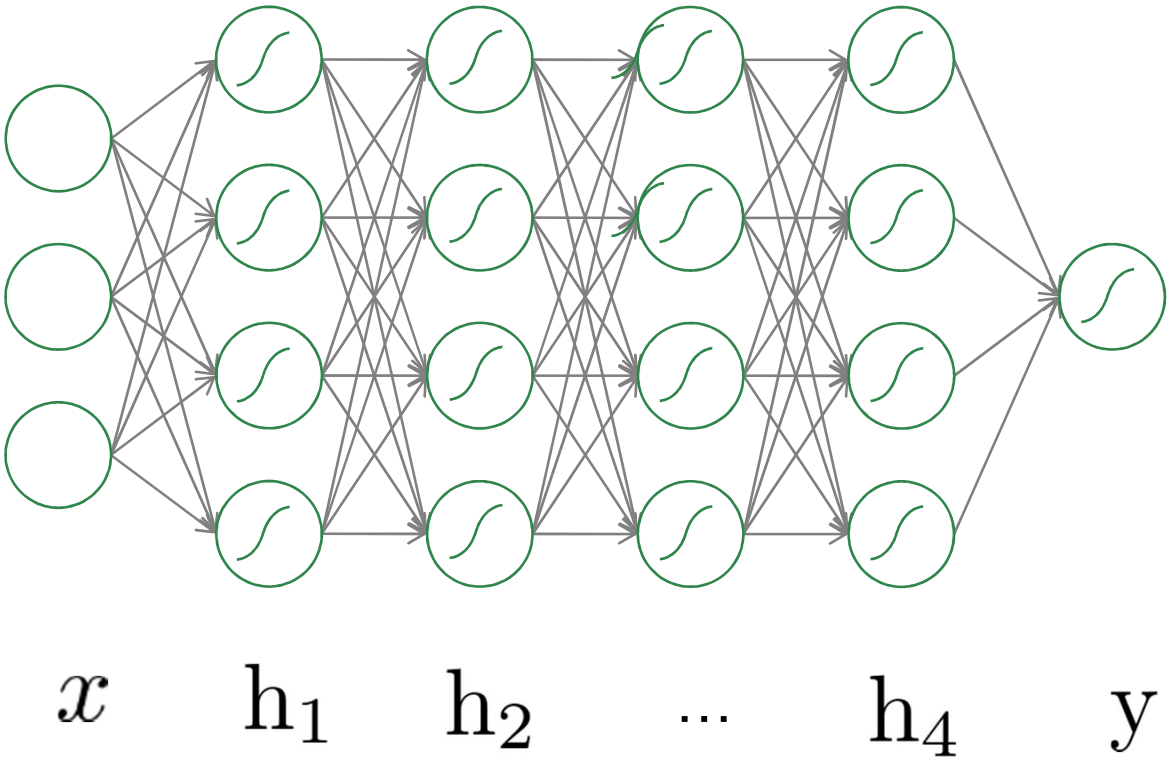
Typically use non-linear function r: $\Phi(x) = r(\theta^T x)$

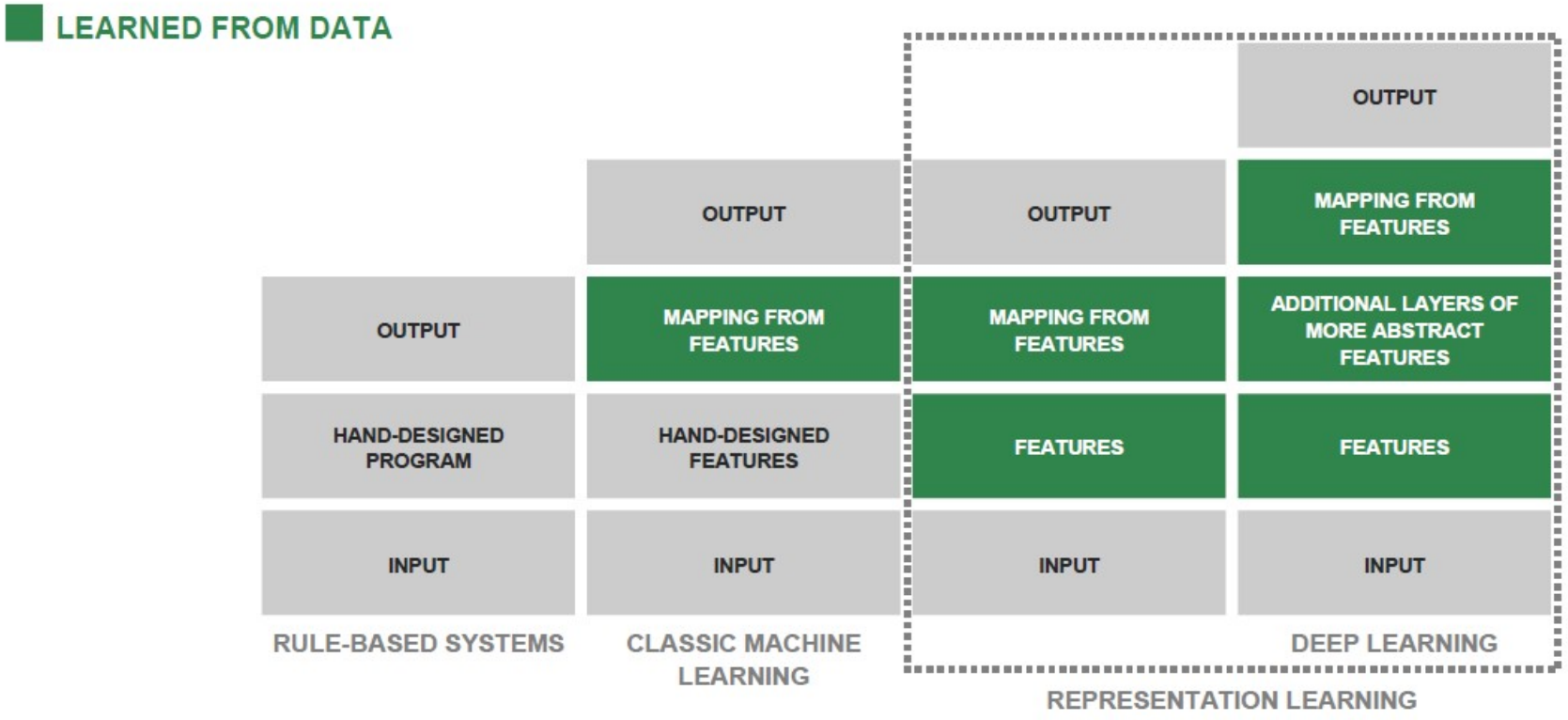


$y = w^T \Phi(x)$

x $\Phi(x)$

Deep neural networks

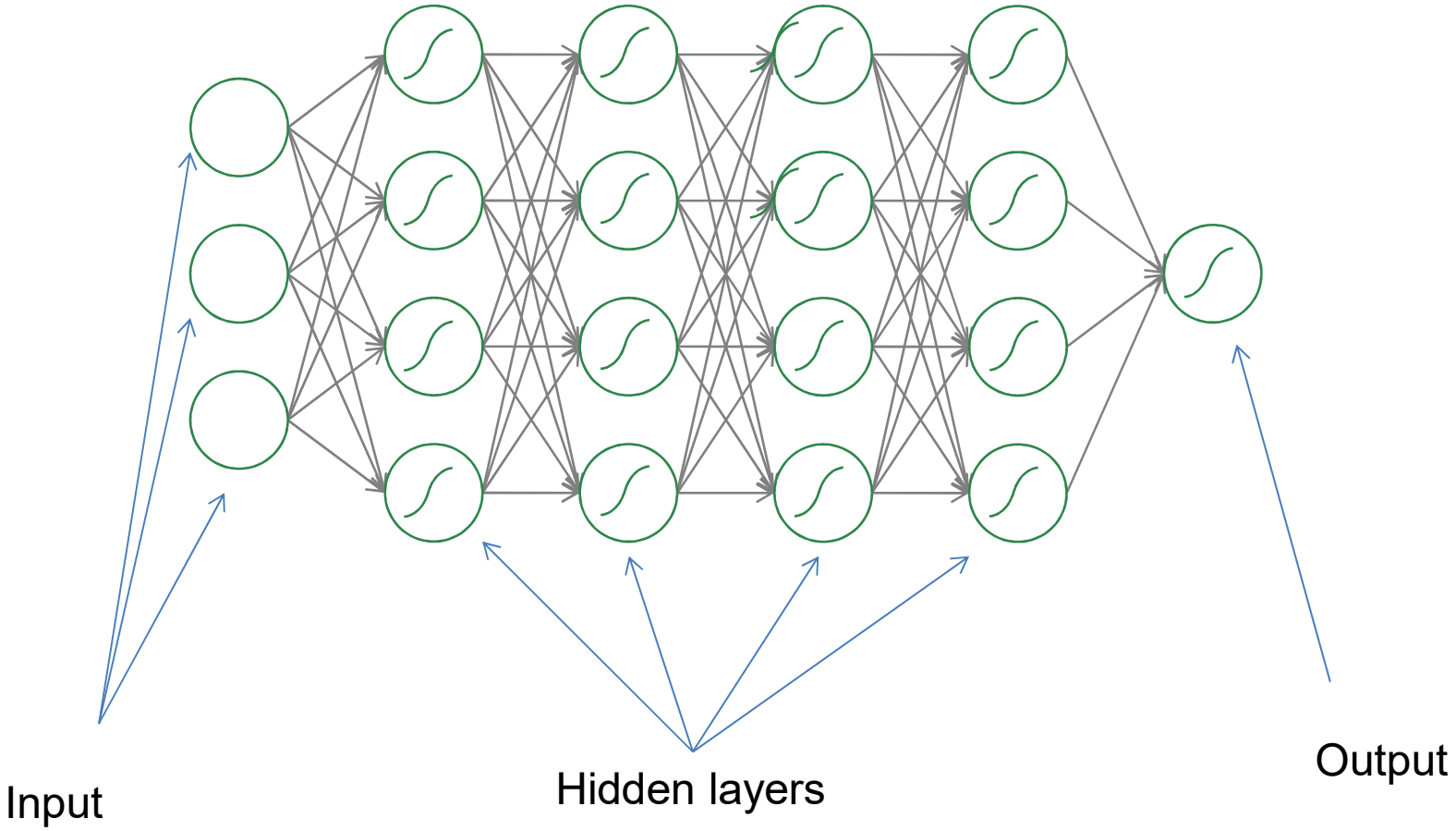



LEARNED FROM DATA
OUTPUT
OUTPUT
OUTPUT
MAPPING FROM FEATURES
OUTPUT
MAPPING FROM FEATURES
MAPPING FROM FEATURES
ADDITIONAL LAYERS OF MORE ABSTRACT FEATURES
HAND-DESIGNED PROGRAM
HAND-DESIGNED FEATURES
FEATURES
FEATURES
INPUT
INPUT
INPUT
INPUT
RULE-BASED SYSTEMS
CLASSIC MACHINE LEARNING
DEEP LEARNING
REPRESENTATION LEARNING

Components of basic neural networks

- Representations:
  - Input
  - Hidden variables
- Layers/weights:
  - Hidden layers
  - Output layer

Components



Input

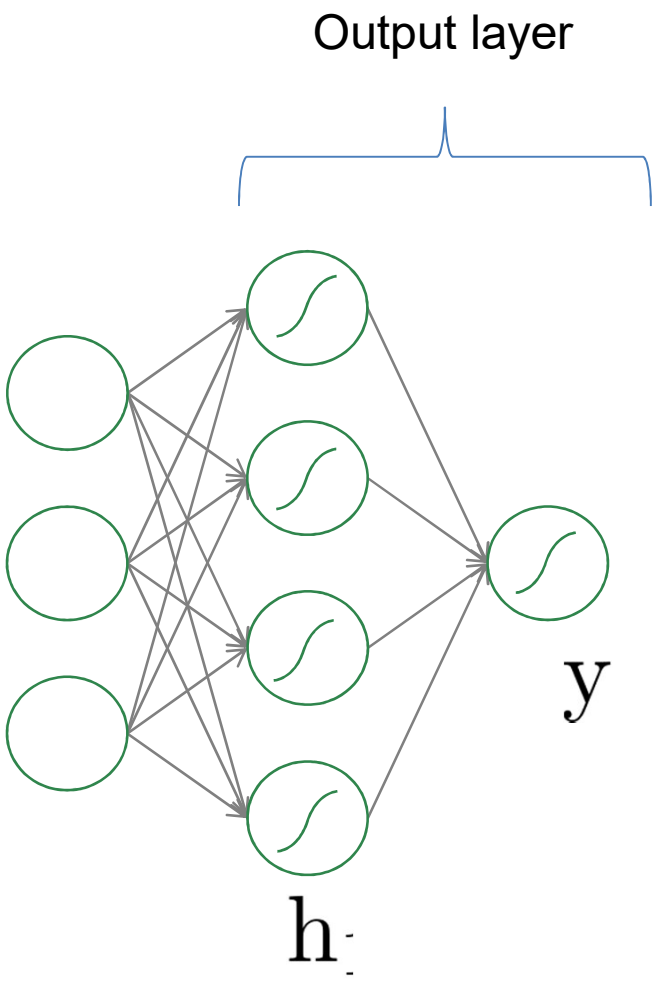
- Represented as a vector
- Sometimes require some preprocessing, e.g.,
  - Subtract mean
  - Divide by variance (standardise)
  - Normalize to [-1,1]

x

Expand

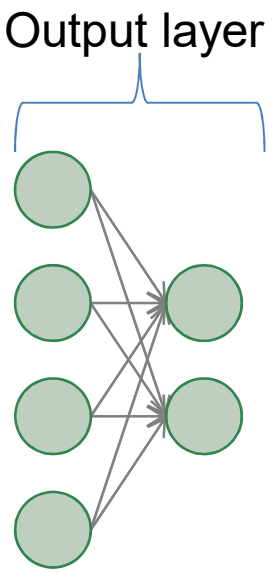
Output layer

- Regression: $y = w^T h + b$
- Linear units: no nonlinearity



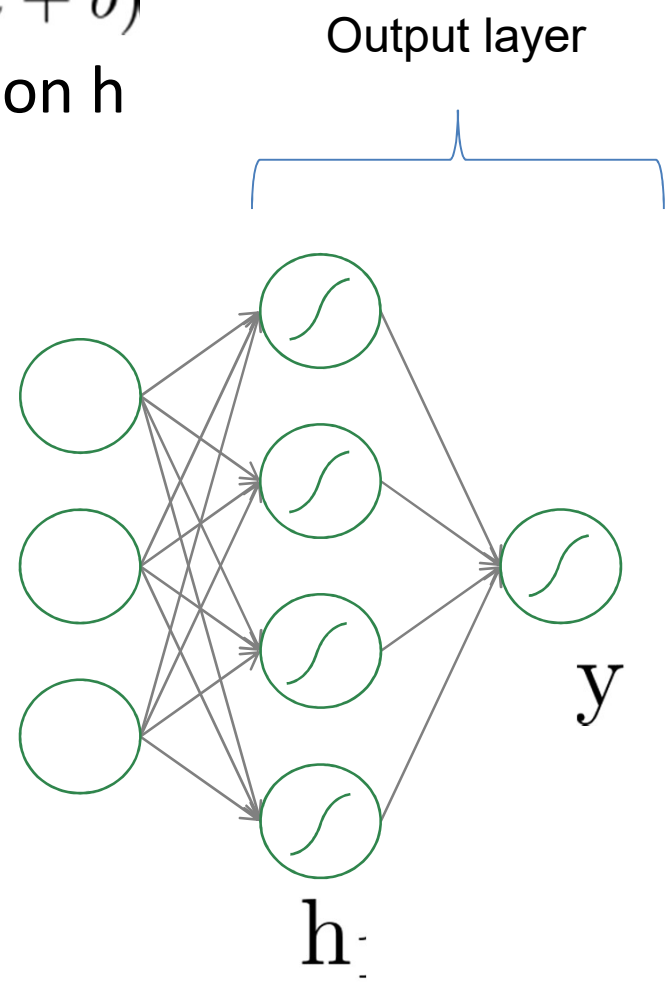
Output layer

- Regression: $y = W^T h + b$
- Linear units: no nonlinearity
- Multiple outputs



Output layer

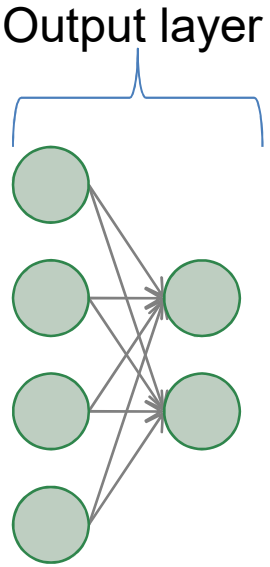
- Binary classification: $y = \sigma(w^T h + b)$
- Correponds to logistic regression on h



Output layer

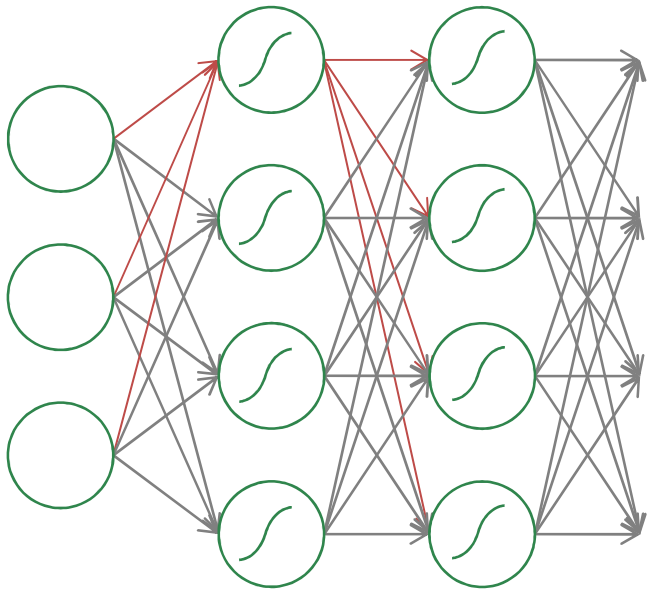
- Multi-class classification:
- $y = \text{softmax}(z) \quad z = W^T h + b$
- Correponds to multi-class logistic regression on h

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for} j = 1, \ldots, K$$



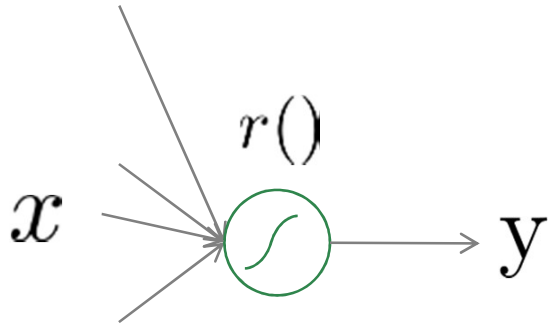
Hidden layers

- Neurons take weighted linear combination of the previous layer
- So can think of outputting one value for the next layer



Activations

- $y = r(w^T h + b)$
- Typical activation functions r
  - Threshold
    - t(z)=$\mathbb{I}$[z≥0]
  - Sigmoid
    - $\sigma(z)$=1/(1+exp(−z))
  - Tanh



Saturation

Small gradient!

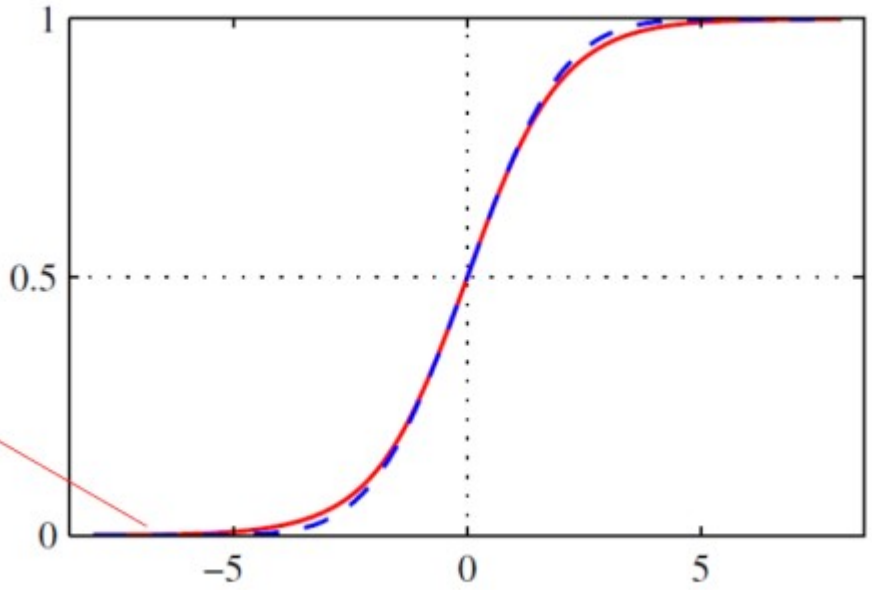


Figure borrowed from *Pattern Recognition and Machine Learning*, Bishop

ReLU

- $\mathrm{ReLU}(z) = \max\{z, 0\}$

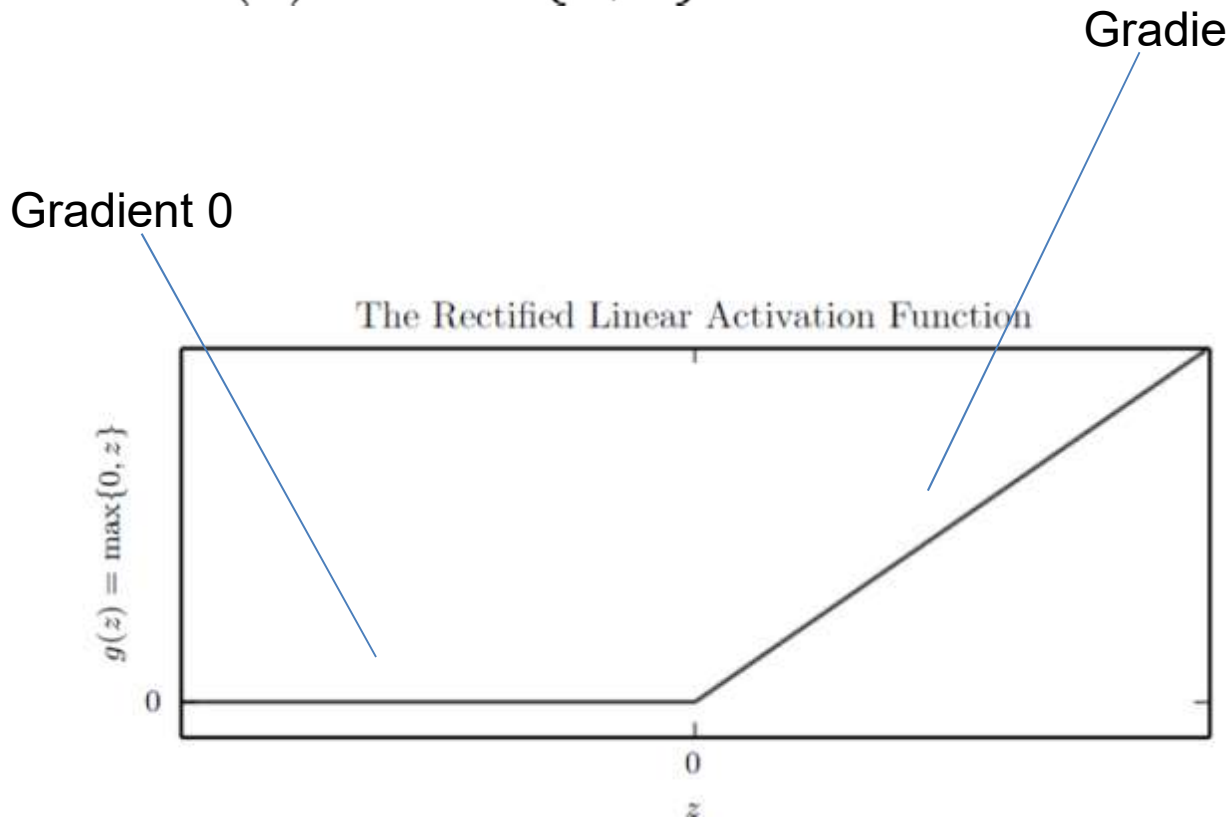


Figure from *Deep learning*, by Goodfellow, Bengio, Courville.

Fitting the NN

- Define a loss function that quantifies our unhappiness with the scores across the training data.
- Come up with a way of efficiently finding the parameters that minimize the loss function. (optimization)

Loss functions

- A loss function tells how good our current classifier is given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

- Where x is the input and y is the (scalar) label
- Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Regression model - MSE

- Mean squared error

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i, W))^2$$

- Optimised regression line minimizes the sum of distance of each point to the regression line
- Mean Squared Logarithmic Error
  - Used when large differences between actual and predicted value don‘t matter (for large values)
- Slow convergence for activation function used for classification

Softmax classifier – cross-entropy loss

- Let scores be unnormalised probabilities

$$\boxed{P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$

- Minimise NLL for correct class

$$\boxed{L_i = -\log P(Y = y_i|X = x_i)}$$

$$L = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(f(x_i W)) + (1 - y_i)\log(1 - f(x_i W))\right]$$

Other loss functions

- KL Divergence
  - measure of how one probability distribution diverges from a second expected probability distribution

$$
\begin{aligned}
\mathcal{L} &= \frac{1}{n}\sum_{i=1}^{n} \mathcal{D}_{KL}(y_i || f(x_i W)) \\
&= \frac{1}{n}\sum_{i=1}^{n} \left[y_i \cdot \log\left(\frac{y_i}{f(x_i W)}\right)\right] \\
&= \underbrace{\frac{1}{n}\sum_{i=1}^{n} \left(y_i \cdot \log(y_i)\right)}_{entropy} \underbrace{-\frac{1}{n}\sum_{i=1}^{n} \left(y_i \cdot \log(f(x_i W))\right)}_{cross-entropy}
\end{aligned}
$$

- Hinge loss
  - max-margin objective (used e.g. in SVMs)

Summary and outlook

- We now have all the ingredients to fit (deep) neural networks
  - Linear algebra+matrix calculus
  - Building blocks (input/hidden layers/outputs)
  - Activation functions
  - Loss
- In the next lecture you will learn how to bring this all together so that we can optimise the parameters of the neural network