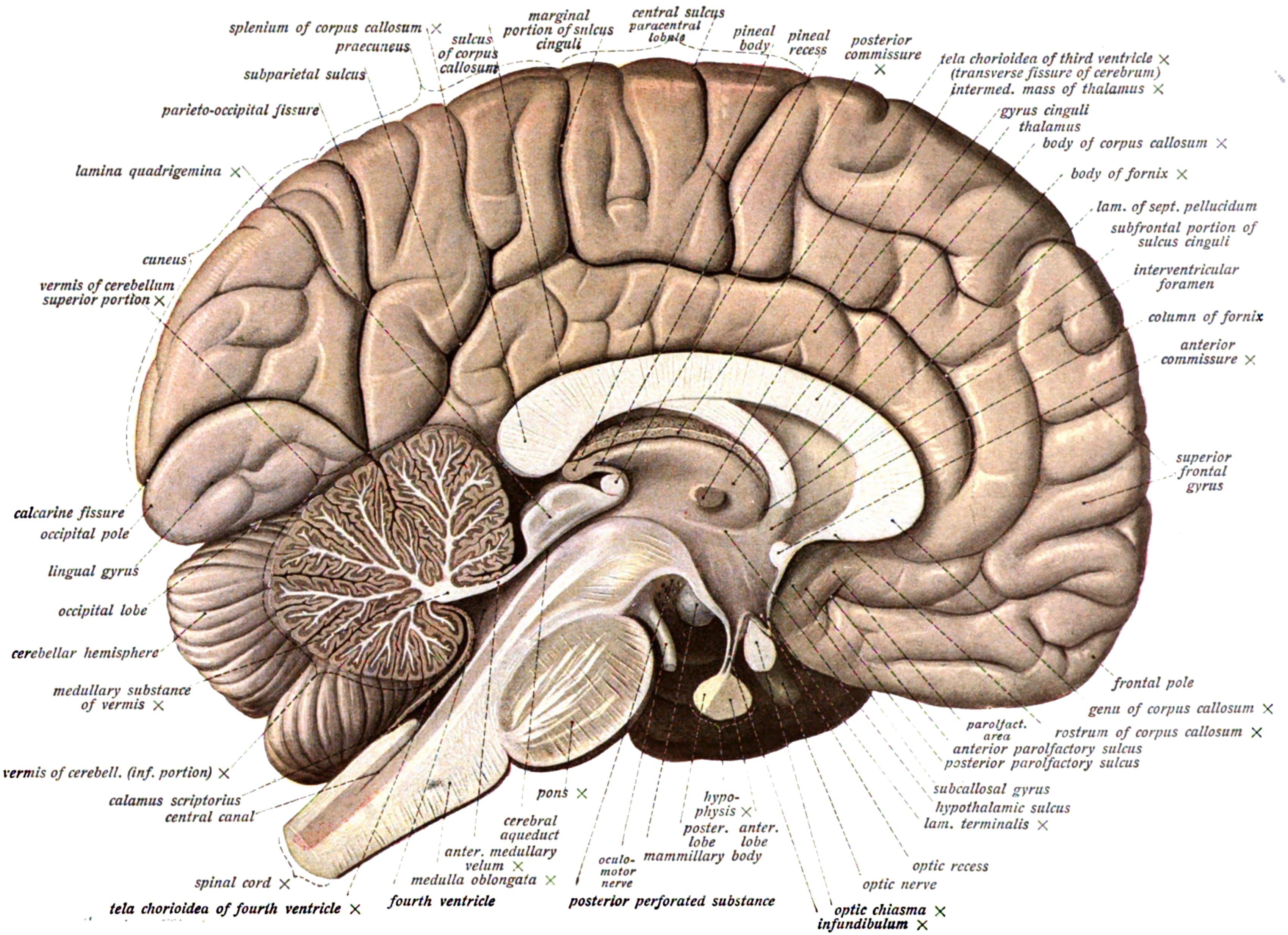
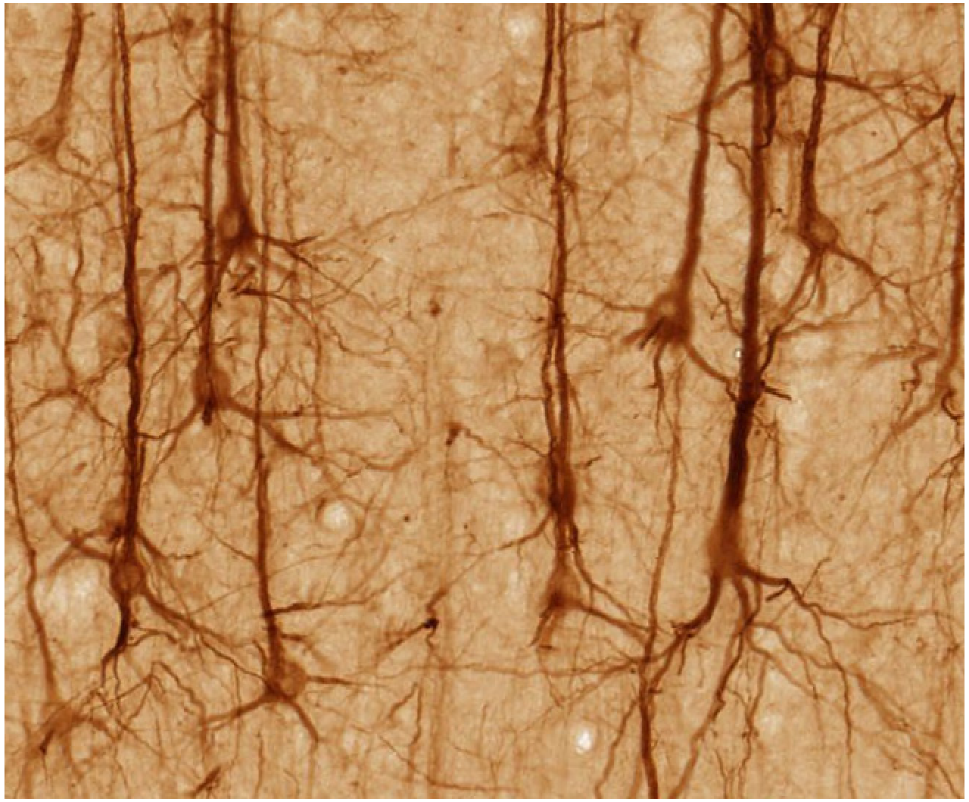
The Brain



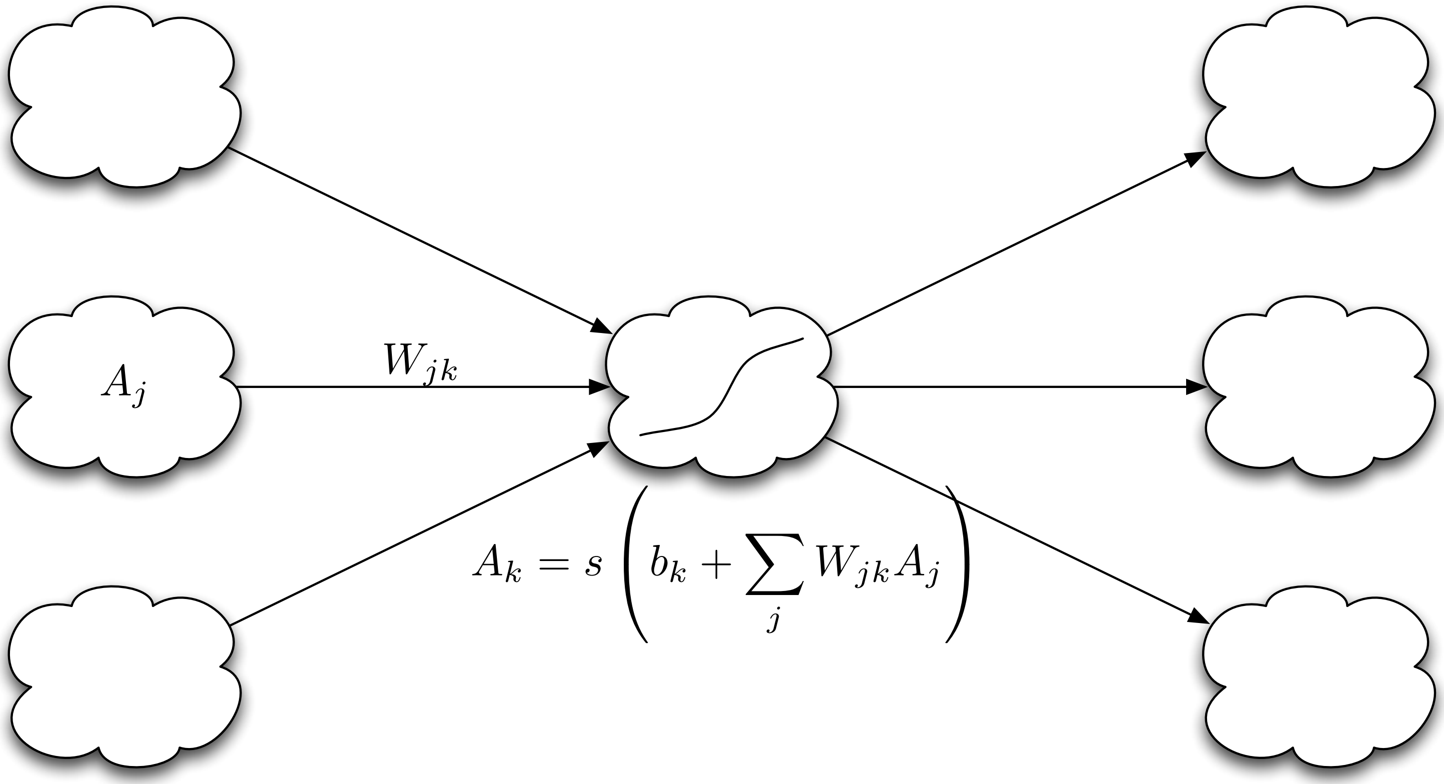
splenium of corpus callosum
praecuneus
subparietal sulcus
sulcus of corpus callosum
marginal portion of sulcus cinguli
central sulcus
paracentral lobule
pineal body
pineal recess
posterior commissure
tela chorioidea of third ventricle
(transverse fissure of cerebrum)
intermed. mass of thalamus
parieto-occipital fissure
gyrus cinguli
thalamus
body of corpus callosum
lamina quadrigemina
body of fornix
lam. of sept. pellucidum
subfrontal portion of sulcus cinguli
cuneus
vermis of cerebellum superior portion
interventricular foramen
column of fornix
anterior commissure
superior frontal gyrus
calcarine fissure
occipital pole
lingual gyrus
occipital lobe
cerebellar hemisphere
medullary substance of vermis
frontal pole
genu of corpus callosum
parolfact. area
rostrum of corpus callosum
anterior parolfactory sulcus
posterior parolfactory sulcus
vermis of cerebell. (inf. portion)
calamus scriptorius
central canal
pons
hypo-physis
subcallosal gyrus
hypothalamic sulcus
lam. terminalis
cerebral aqueduct
poster. lobe
anter. lobe
anter. medullary velum
oculo-motor nerve
mammillary body
optic recess
spinal cord
medulla oblongata
optic nerve
tela chorioidea of fourth ventricle
fourth ventricle
posterior perforated substance
optic chiasma
infundibulum

The Brain

- Basic computational elements: neurons.
- Receives signals from other neurons via dendrites.
- Sends processed signals via axons.
- Axon-dendrite interactions at synapses.
- $10^{10} - 10^{11}$ neurons.
- $10^{14} - 10^{15}$ synapses.
- Connectionist architecture: the network and its structure govern the computations performed.



A Simple Model of Neural Computations



Modelling Conditional Probabilities

- Data vectors $x_i \in \mathbb{R}^p$, binary labels $y_i \in \{0, 1\}$.
- **Inputs** $x_{i1}, \ldots, x_{ip}$
- **output** $\hat{y}_i = p(Y = 1 | X = x_i)$
- **hidden unit activations** $h_{i1}, \ldots, h_{im}$
  - Compute **hidden unit activations**:

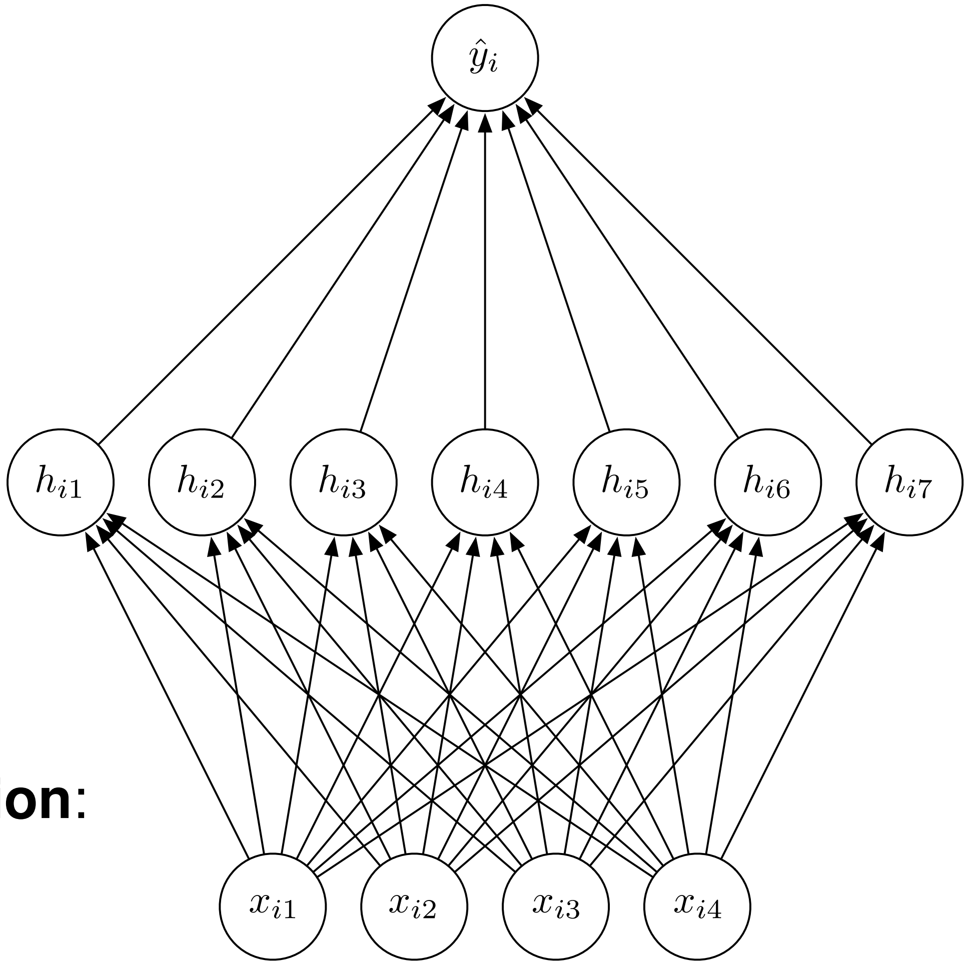
$$h_{ik} = s\left(b_k^h + \sum_{j=1}^{p} W_{jk}^h x_{ij}\right)$$

  - Compute **output probability**:

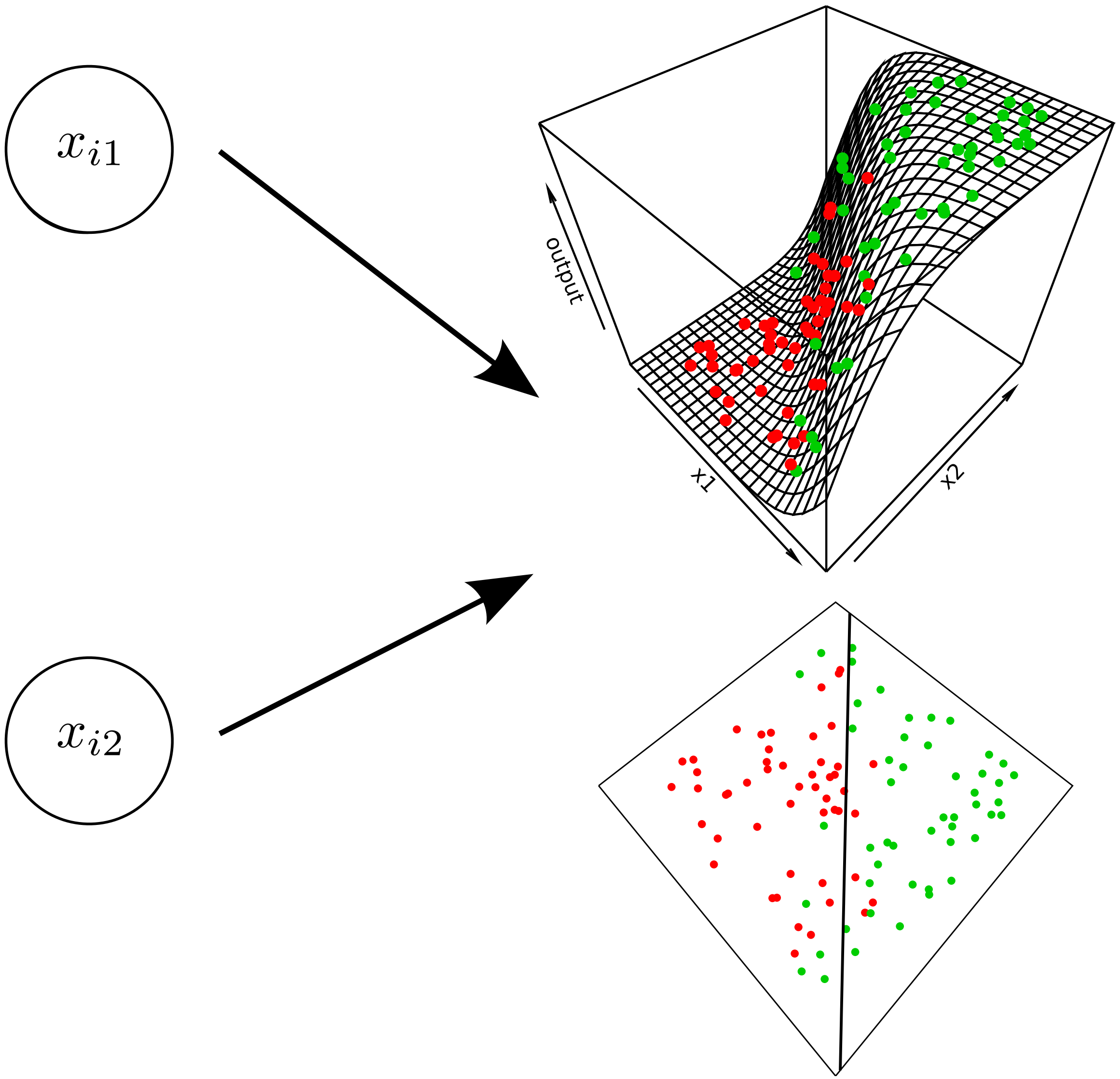
$$\hat{y}_i = s\left(b^o + \sum_{k=1}^{m} W_k^o h_{ik}\right)$$

- Common nonlinear **activation function**: the logisitic function

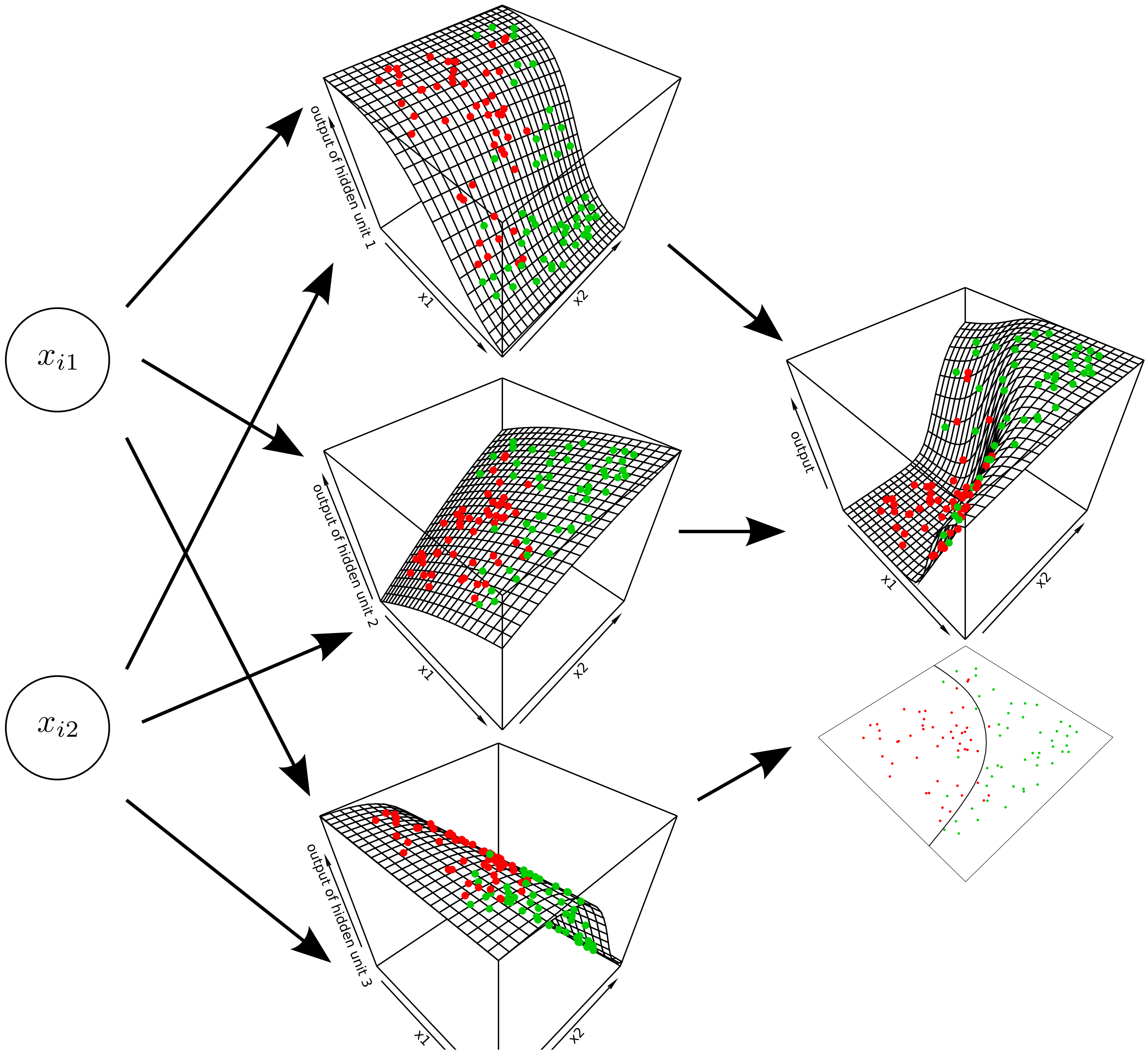
$$s(z) = \frac{1}{1 + \exp(-z)}$$



A Simple Model of Neural Computations



A Simple Model of Neural Computations



Training a Neural Network

- Objective function: L_2-regularized log loss

$$J = -\sum_{i=1}^{n} y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) + \frac{1}{2} \sum_{jk} C|W_{jk}^h|^2 + \frac{1}{2} \sum_{k} C|W_k^o|^2$$

where

$$\hat{y}_i = s\left(b^o + \sum_{k=1}^{m} W_k^o h_{ik}\right) \qquad h_{ik} = s\left(b_k^h + \sum_{j=1}^{p} W_{jk}^h x_{ij}\right)$$

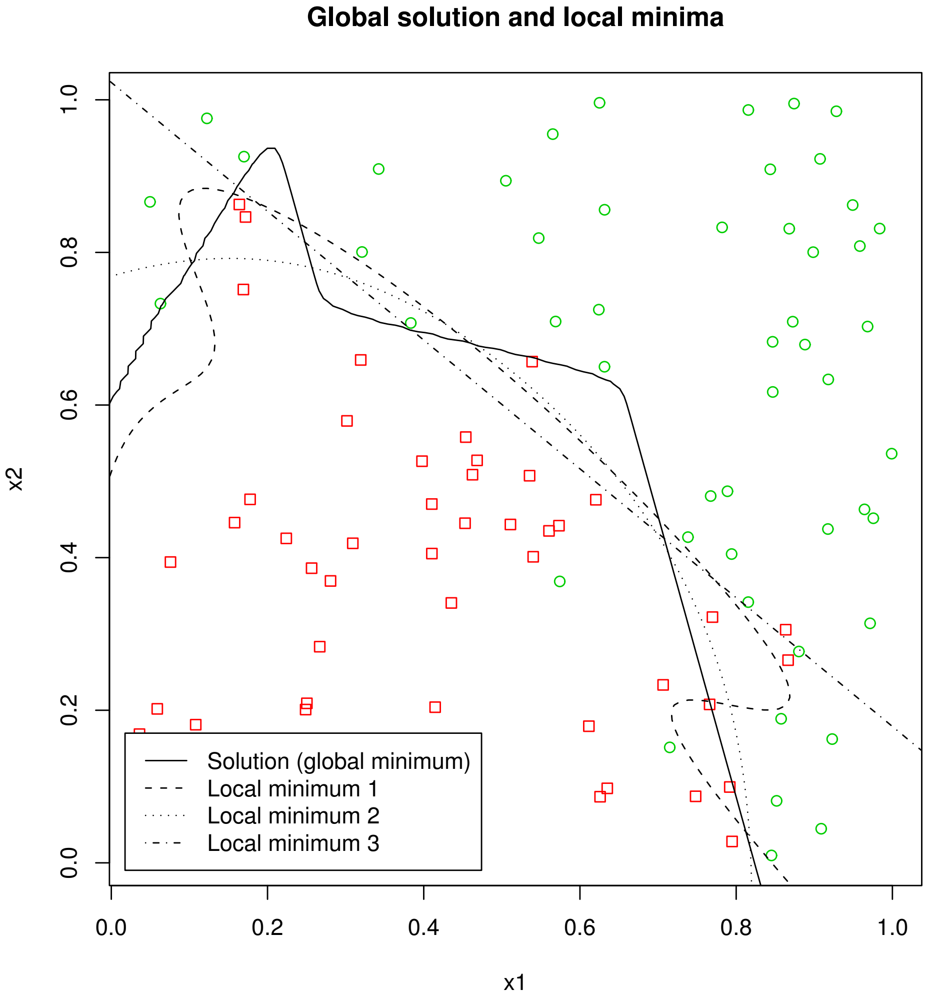
- Optimize parameters $\{b_k^h, b^o, W_{jk}^h, W_k^o\}$ by gradient descent.

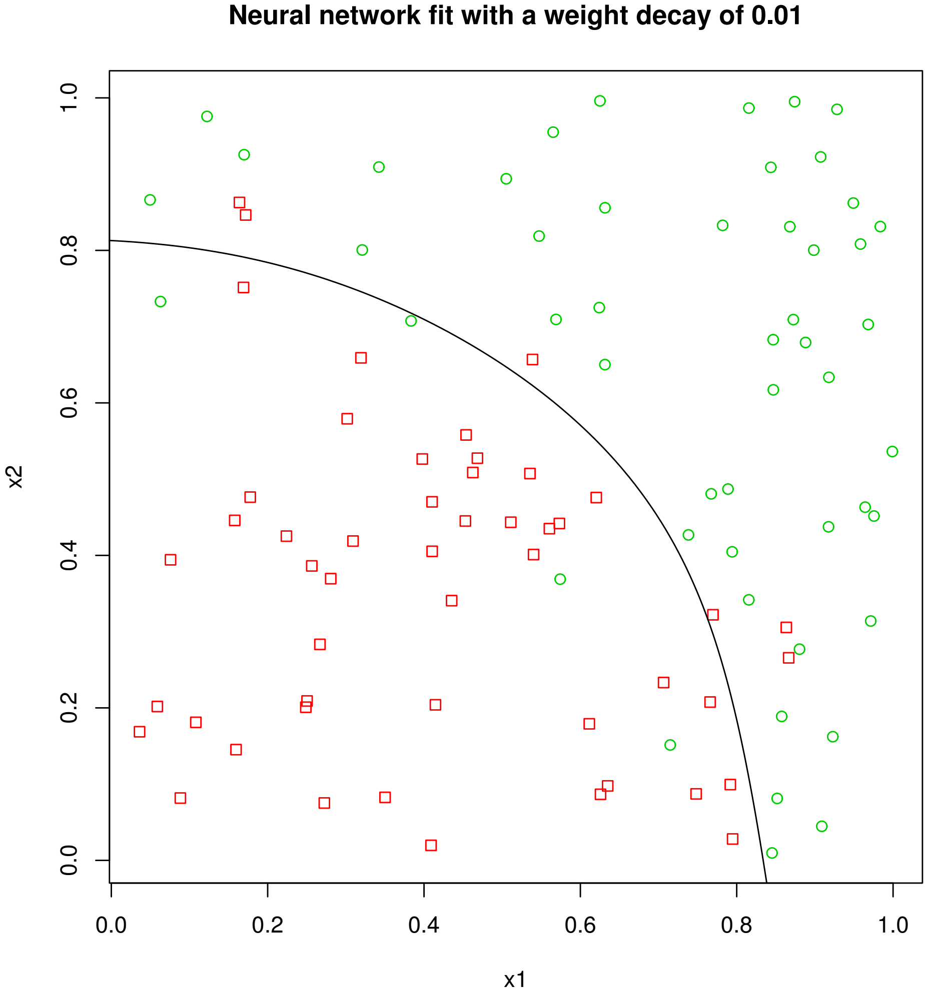
$$\frac{dJ}{dW_k^o} = CW_k^o + \sum_{i=1}^{n} \frac{dJ}{d\hat{y}_i} \frac{d\hat{y}_i}{dW_k^o} = CW_k^o + \sum_{i=1}^{n} (\hat{y}_i - y_i) h_{ik}$$

$$\frac{dJ}{dW_{jk}^h} = CW_{jk}^h + \sum_{i=1}^{n} \frac{dJ}{d\hat{y}_i} \frac{d\hat{y}_i}{dh_{ik}} \frac{dh_{ik}}{dW_{jk}^h} = CW_{jk}^h + \sum_{i=1}^{n} (\hat{y}_i - y_i) W_k^o h_{ik}(1 - h_{ik}) x_{ij}$$

- **Backpropagation**: gradients computed via chain rule, and propagated through the network backwards.
- L_2 regularization often called **weight decay**.

Neural Networks





R package implementing neural networks with a single hidden layer: `nnet`.

Neural Networks – Discussion

- Nonlinear hidden units introduce modelling flexibility.
- As opposed to user introduced nonlinearities, kernel methods, kNNs, features are global, and learnt to maximize predictive performance.
- Neural networks with a single hidden layer can model arbitrarily complex functions (with enough hidden units).
- Highly flexible framework, with many variations to solve different learning problems and introduce domain knowledge.
- Optimization problem is not convex, and objective function can have many local optima, plateaus and ridges.
- On large scale problems, often use stochastic gradient descent, along with a whole host of techniques for optimization, regularization, and initialization.
- Strengths of neural networks:
  - Flexibility and generalization ability.
  - Computational efficiency, parallelizability.
- Recent developments, especially by Geoffrey Hinton, Yann LeCun, Yoshua Bengio, Andrew Ng and others. See also `http://deeplearning.net/`.

Neural Networks – Variations

- Other loss functions can be used, e.g. for regression:

$$\sum_{i=1}^{n} |y_i - \hat{y}_i|^2$$

For multiclass classification, use **softmax** outputs:

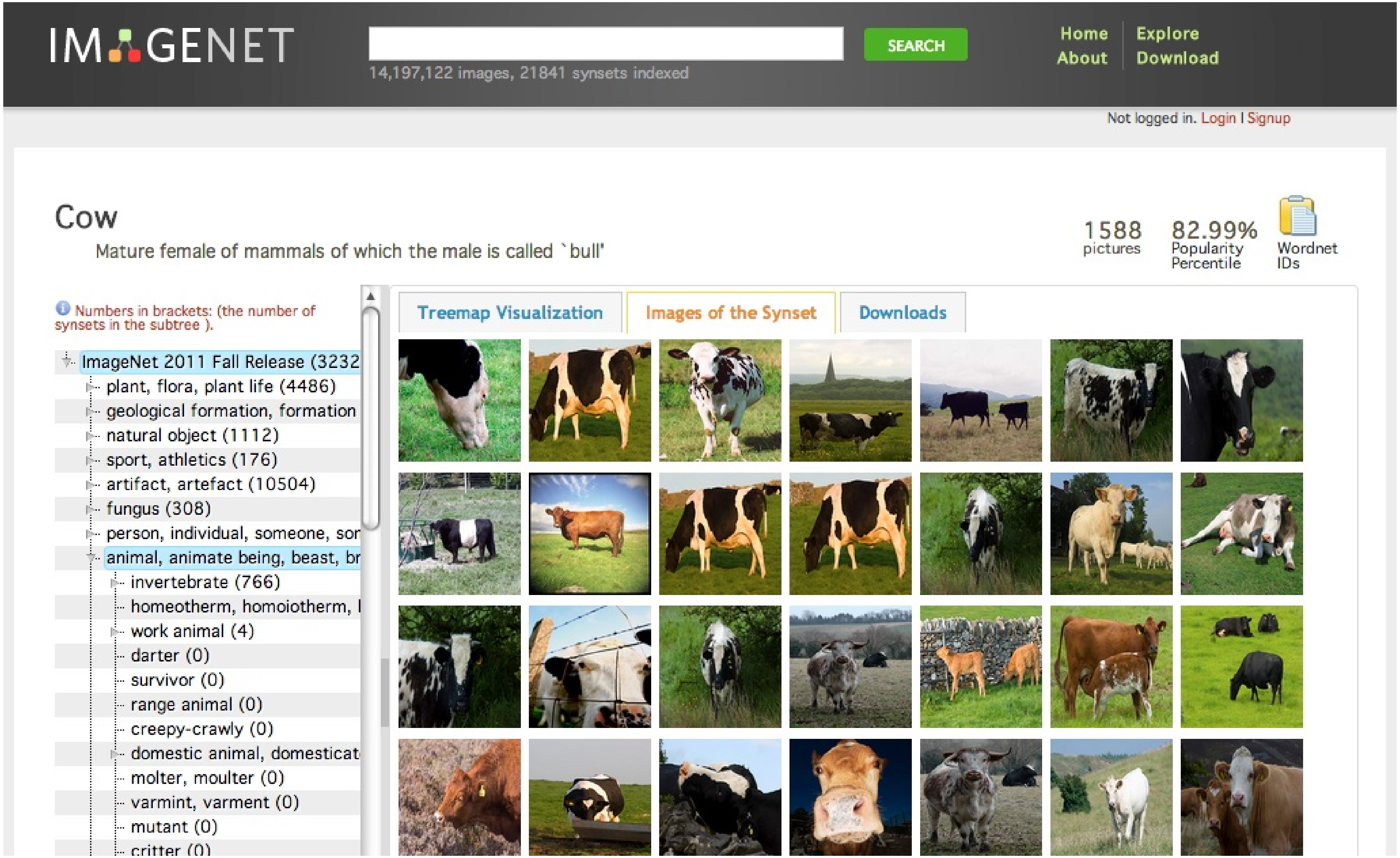
$$\hat{y}_{ik} = \frac{\exp(b_k^o + \sum_\ell W_{lk}^o h_{i\ell})}{\sum_{k'} \exp(b_{k'}^o + \sum_\ell W_{lk'}^o h_{i\ell})} \qquad L(y_i, \hat{y}_i) = \sum_{k=1}^{K} \mathbb{1}(y_i = k) \log \hat{y}_{ik}$$

- Other activation functions can be used, e.g. a recent popular one is called **rectified linear activation**:

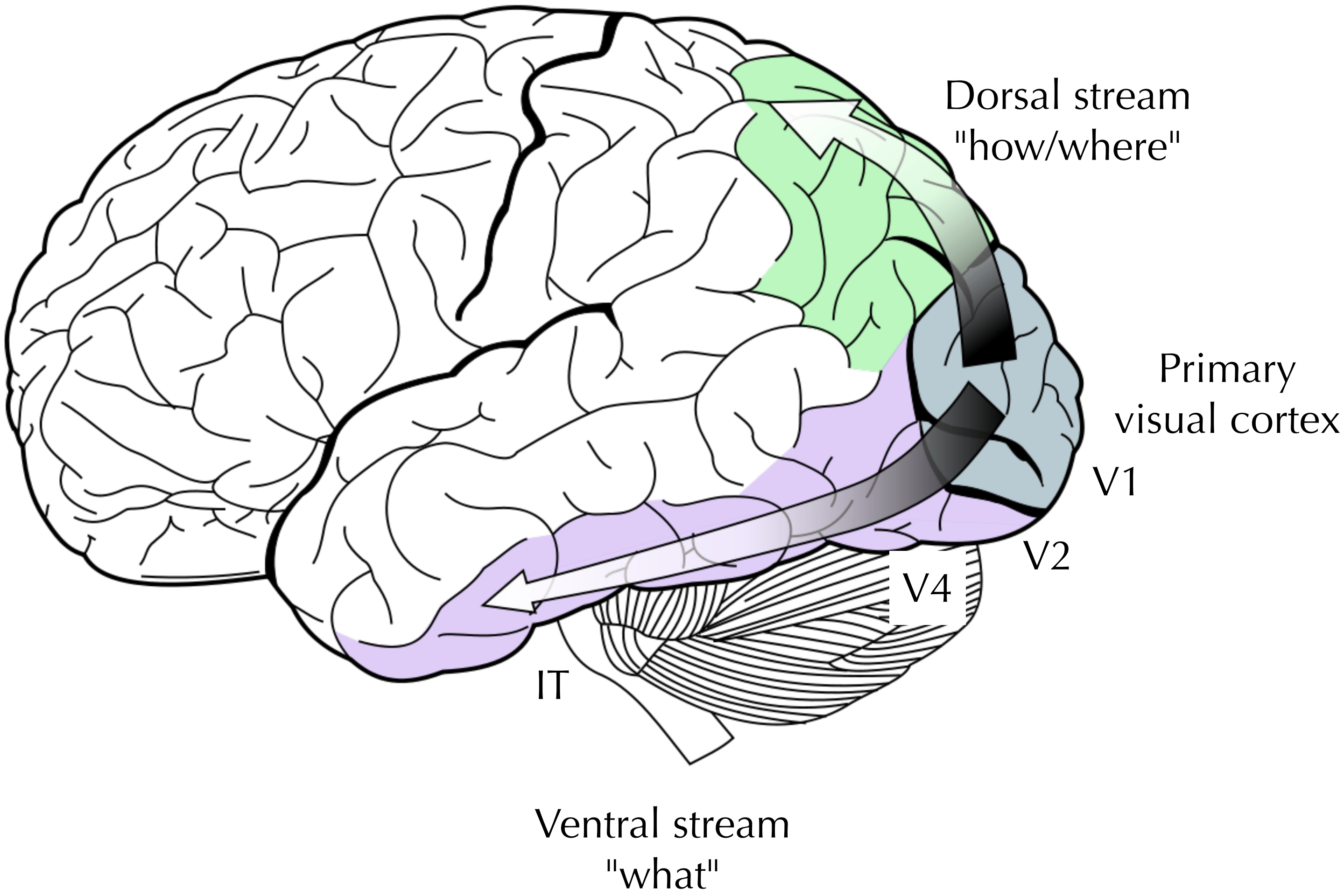
$$s(z) = \log(1 + \exp(z))$$

- Multiple layers of hidden units can be used, called **multilayer perceptrons** (MLP) or **deep networks**.

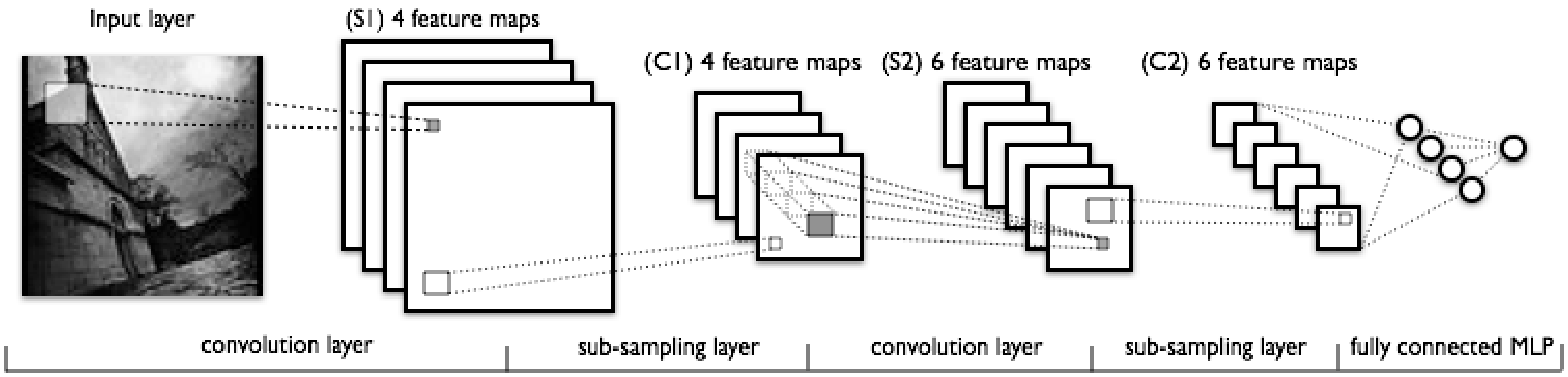
Visual Object Recognition



Visual Processing in the Brain



Deep Convolutional Neural Networks



- Input is a 2D image, $X \in \mathbb{R}^{p \times q}$.
- Convolution: detects simple object parts or features

$$A^m = s(X * W^m) \qquad\qquad A^m_{jk} = s\left(b^m + \sum_{fg} X_{j-f,k-g} W^m_{fg}\right)$$

- Sub-sampling: incorporates local translation invariance by max-pooling

$$B^m_{jk} = \max\{A^m_{fg} : |f-j| \le w, |g-k| \le h\}$$

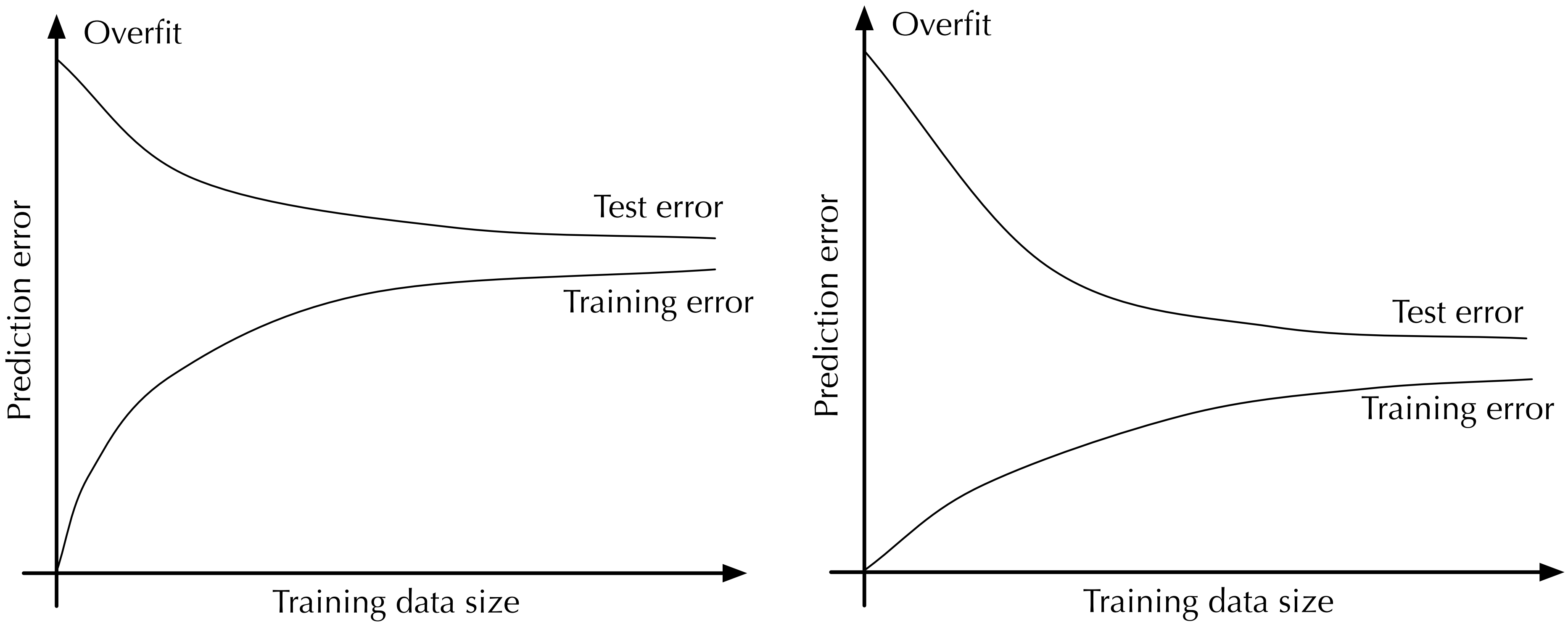
- Learn features/parts of increasing complexity over multiple layers.

LeCun et al, Krizhevsky et al

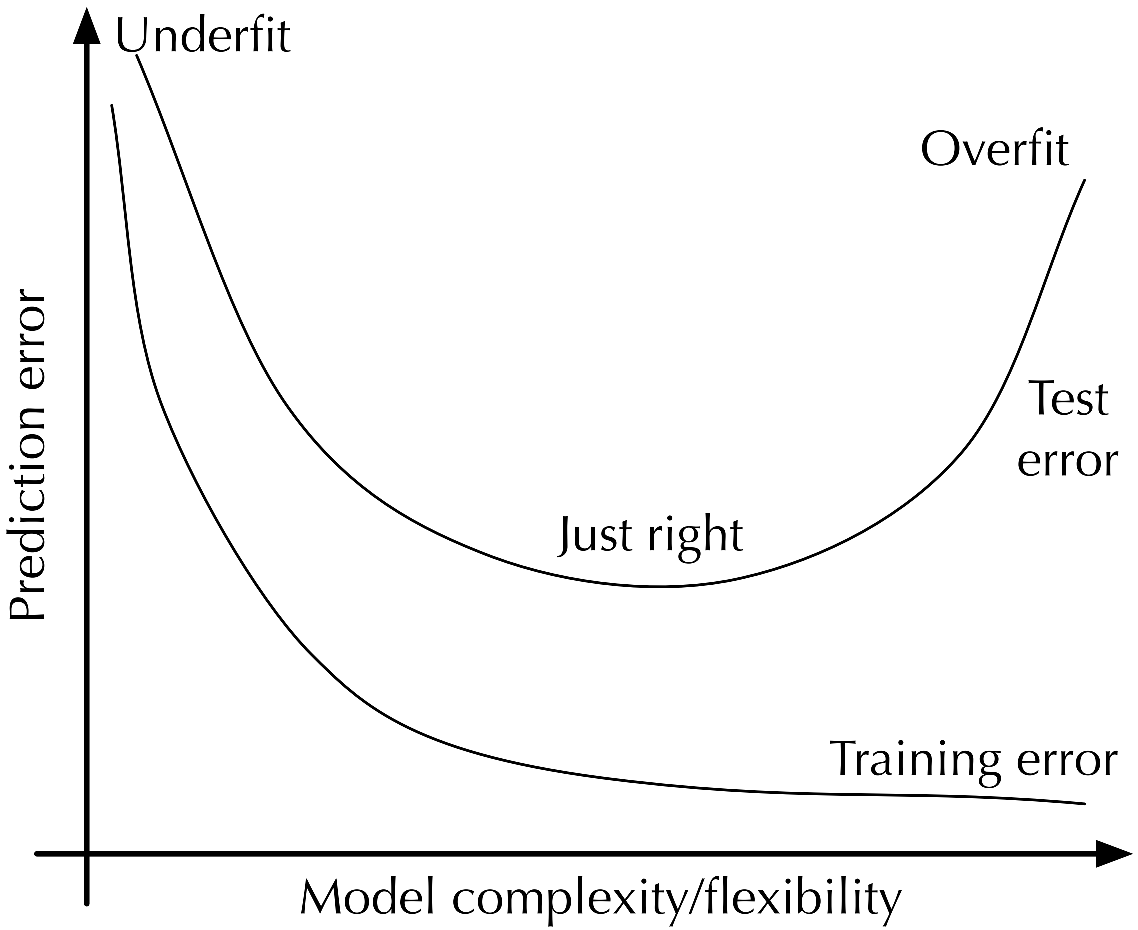
Revisiting Learning Generalization

- Generalization ability is a central concept in machine learning.
- Splitting data into training and test sets allows us to estimate how well our methods are generalizing.
- Two important factors determining generalization ability:
  - Model complexity
  - Training data size
- To control overfitting, we need to regularize learning.
- Can we learn the tuning parameters as well?

Learning Curves



Learning Curves



Bias-Variance Tradeoff

- A different perspective on generalization ability.
- Suppose we are in a regression setting, with

$$Y = f^*(X) + \mathcal{N}(0, \sigma^2)$$

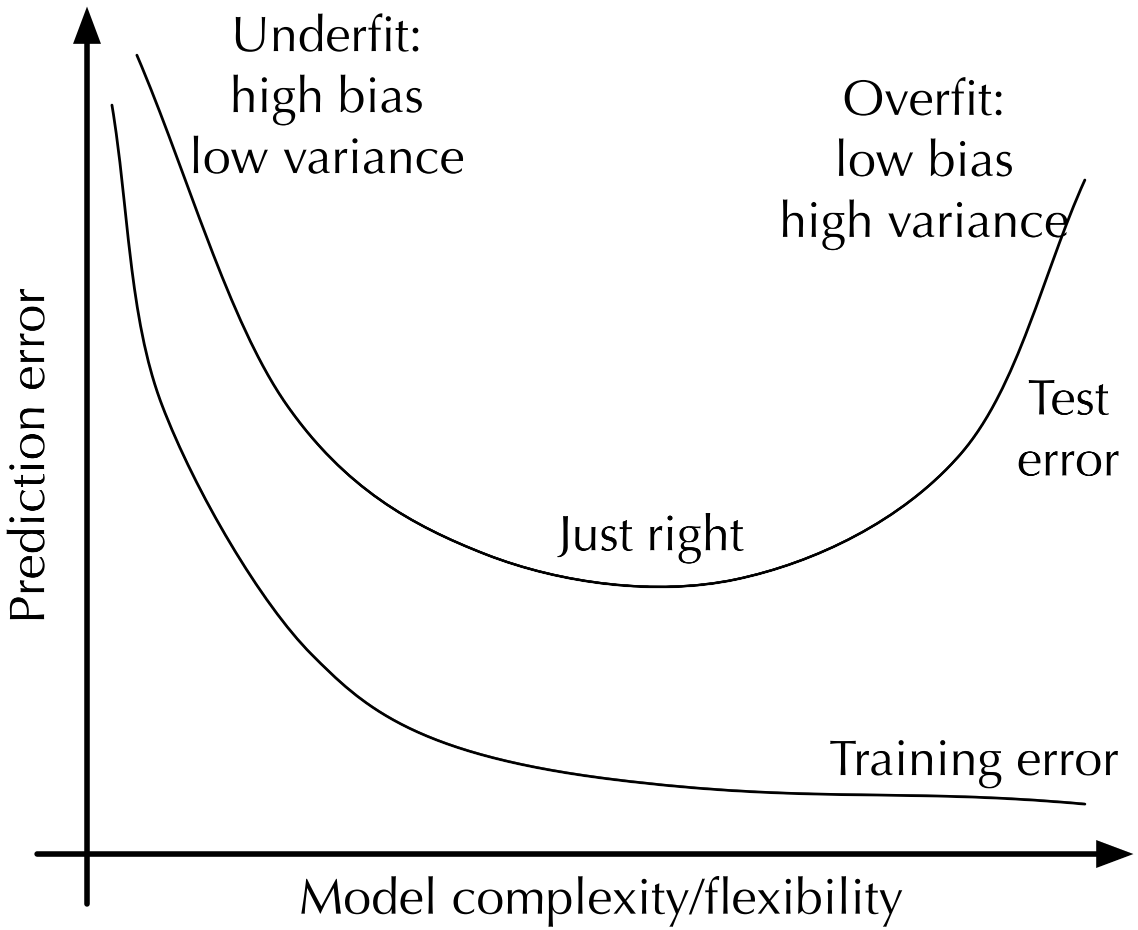
- Given a dataset $D = (x_i, y_i)_{i=1}^n$, train a model $f(x; \theta)$.
- Estimated $\hat{\theta}$ is a function of data set D.
- How will we do, averaging over data sets of size n?

$$\begin{aligned} & \mathbb{E}_D[(Y - f(X; \hat{\theta}(D)))^2] \\ = \; & \underbrace{(\bar{f}(X) - f^*(X))^2}_{\text{bias}^2} + \underbrace{E_D[(\bar{f}(X) - f(X; \hat{\theta}(D)))^2]}_{\text{variance}} + \underbrace{(Y - f^*(X))^2}_{\text{noise}} \end{aligned}$$

where $\bar{f}(X) = E_D[f(X; \hat{\theta}(D)]$ is average prediction (over data sets).

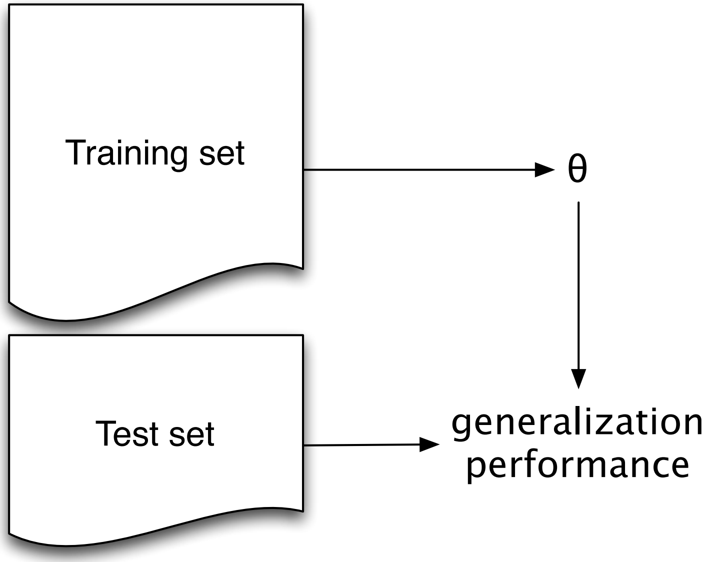
- **Noise**: intrinsic difficulty of regression problem.
-
- **Variance**: How variable is our method if given different datasets? **Bias**: How far is our average prediction away from the truth?

Learning Curve



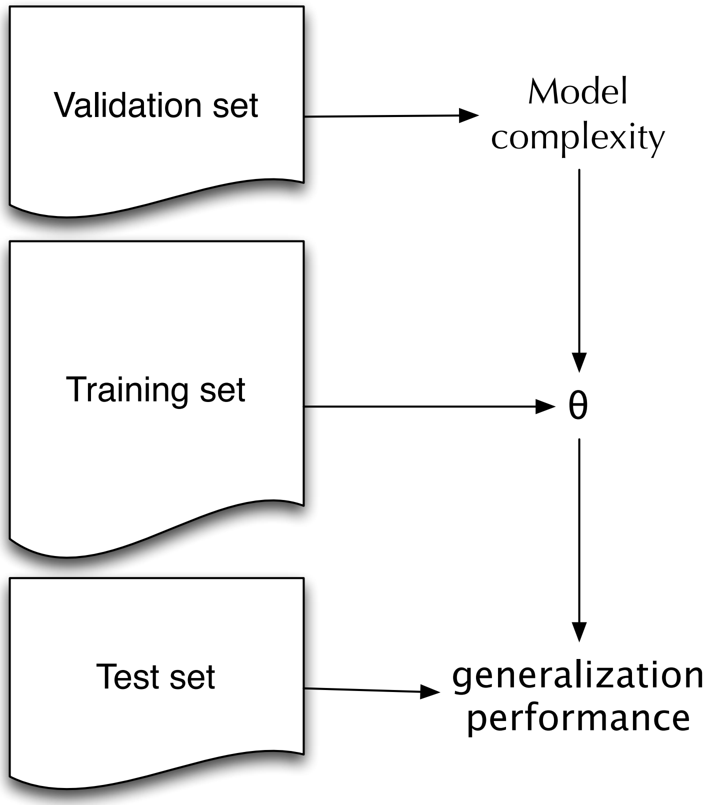
Optimizing Hyperparameters and Model Complexity

- How can we optimize generalization ability, via optimizing choice of tuning parameters, model size, and learning parameters?
- Suppose we have split data into training/test set.
- Test set can be used to determine generalization ability, and used to choose best setting of tuning parameters/model size/learning parameters with best generalization.
- Once these meta-parameters are chosen, still important to determine generalization ability, but cannot use performance on test set to gauge this anymore!
- Idea: split data into 3 sets: training set, test set, and **validation set**.

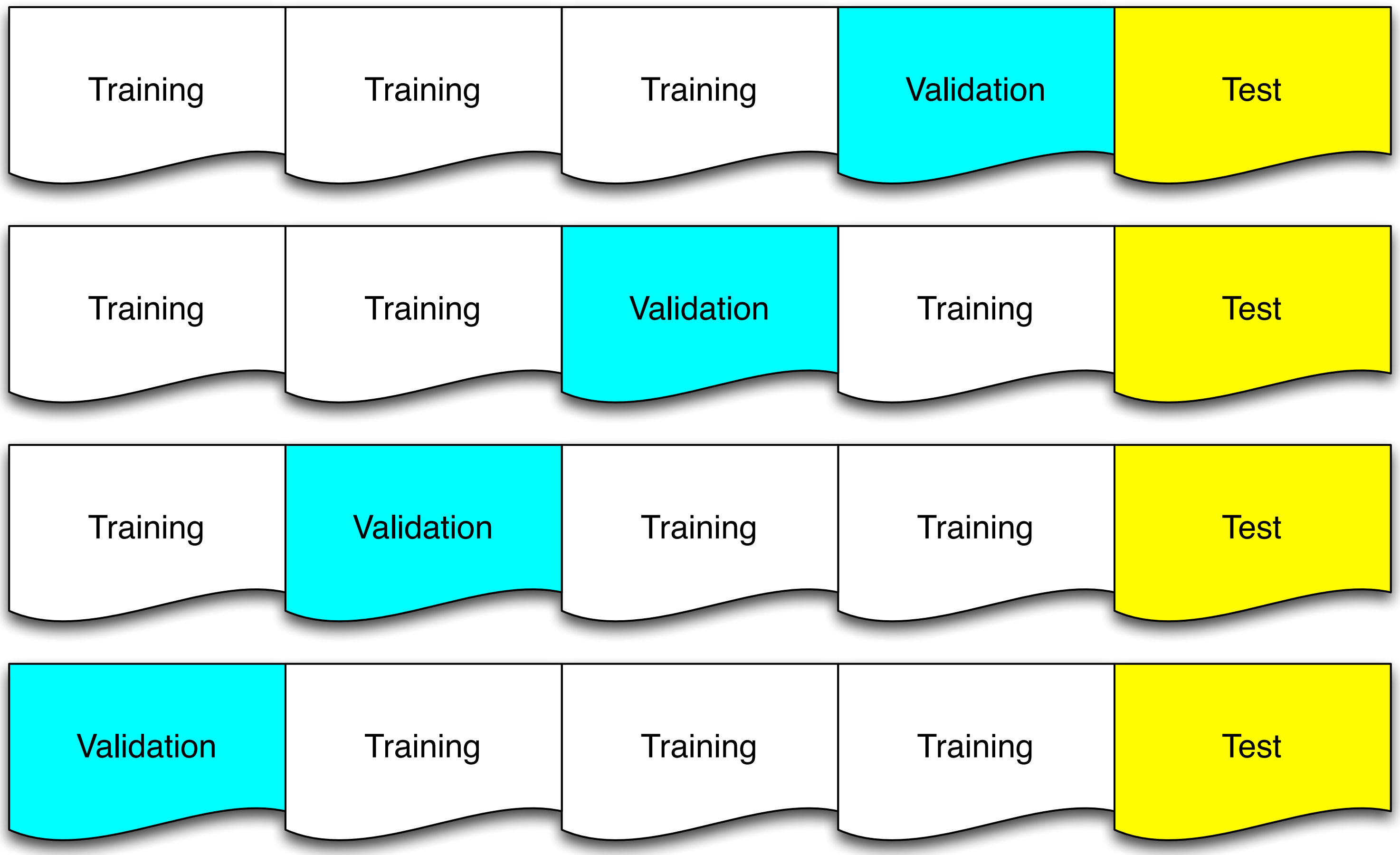


Validation Set

- For each combination of meta-parameters γ:
  - Train our model, obtaining model parameter $\theta(\gamma)$.
  - Evaluate $\theta(\gamma)$ on validation set.
- Pick γ^* with best performance on validation set.
- Using γ^*, train on both training and validation set (**fold** the validation set into the training set), to obtain optimal θ^*.
- Evaluate model with γ^*, θ^* on test set, reporting generalization performance.
- Problem: if we have insufficient data, very wasteful to split into 3 subsets, and estimated generalization performance on validation set may be too noisy to effectively choose meta-parameters.
- Solution: **cross-validation**.



Cross-Validation



Cross-Validation

- Basic approach:
  - Split training set into V folds.
  - For each γ and each $v = 1, \ldots, V$:
    - Use fold v as validation set and the rest to train the model parameters $\hat{\theta}_v$.

$$R_v^{\text{emp}}(\gamma) = \frac{1}{|\text{Fold}(v)|} \sum_{i \in \text{Fold}(v)} L(y_i, \hat{Y}(x_i; \hat{\theta}_v))$$

  - Choose γ^* which minimizes

$$\frac{1}{V} \sum_{v=1}^{V} R_v^{\text{emp}}(\gamma)$$

  - Train model with meta-parameter γ^* on all training set.
  - Report generalization performance on test set.
- Extreme case: **Leave-one-out (LOO)** cross validation: one data item per fold.
- Cross-validation can be computationally very expensive.