

Neural Networks

Jan Chorowski
Instytut Informatyki
Wydział Matematyki i Informatyki Uniwersytet
Wrocławski
2017

Final exam

Please don't forget about it:

- **Monday 5.2.2018 14:00 in 119**
- **Friday 9.2.2018 14:00 in 119**

It should last for about 60-90 minutes (but we will have more time)

Projects

- Please finish by **9.2.2018**.
- Please consult with me or Adrian if you have problems with them.

What to do after Neural Nets?

- Artificial Intelligence course by P. Rychlikowski
- My seminar: Statistics and Neural Networks
- And a good summer school (if you get accepted I'll try to find you money from the University for it): <https://tmlss.ro/>
- My group (Pracownia Inteligencji Obliczeniowej, PIO) meets weekly to discuss papers and research ideas – let me know if you want to be notified about them.

Learning materials

Most lectures have accompanying Notebooks with explanations. Additional materials:

- For Linear Models, Learning Theory, SVMs, K-Means, EM and PCA you can consult Stanford's CS229 handouts by A. Ng: <https://see.stanford.edu/Course/CS229>
- For Deep Neural Nets and Convnets you can consult lecture notes for Stanford's CS231 <http://cs231n.stanford.edu/>
- For more info on LSTMs you can consult Chris Olah's blog and distill.pub: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
<https://distill.pub/2016/augmented-rnns/>
- Last but not least – the Goodfellow and Bengio Deep Learning book: <http://www.deeplearningbook.org/>

Topic 1 - Learning

- We speak about learning when we want to automatically determine the relations present in the data.
- Thus learning starts with **DATA**
 - Implementation of an algorithm is not learning
 - Choosing the parameters of a program to match the data is learning
- The other part of learning is choosing a **family of functions** (hypotheses) from which we will choose the one matching the data
 - The larger the hypothesis space, the more data we need to have to choose the correct hypothesis
 - We need to restrict the set of hypotheses (introduce bias based on our knowledge about the problem) – reliably learning a function from the set of all functions is impossible!

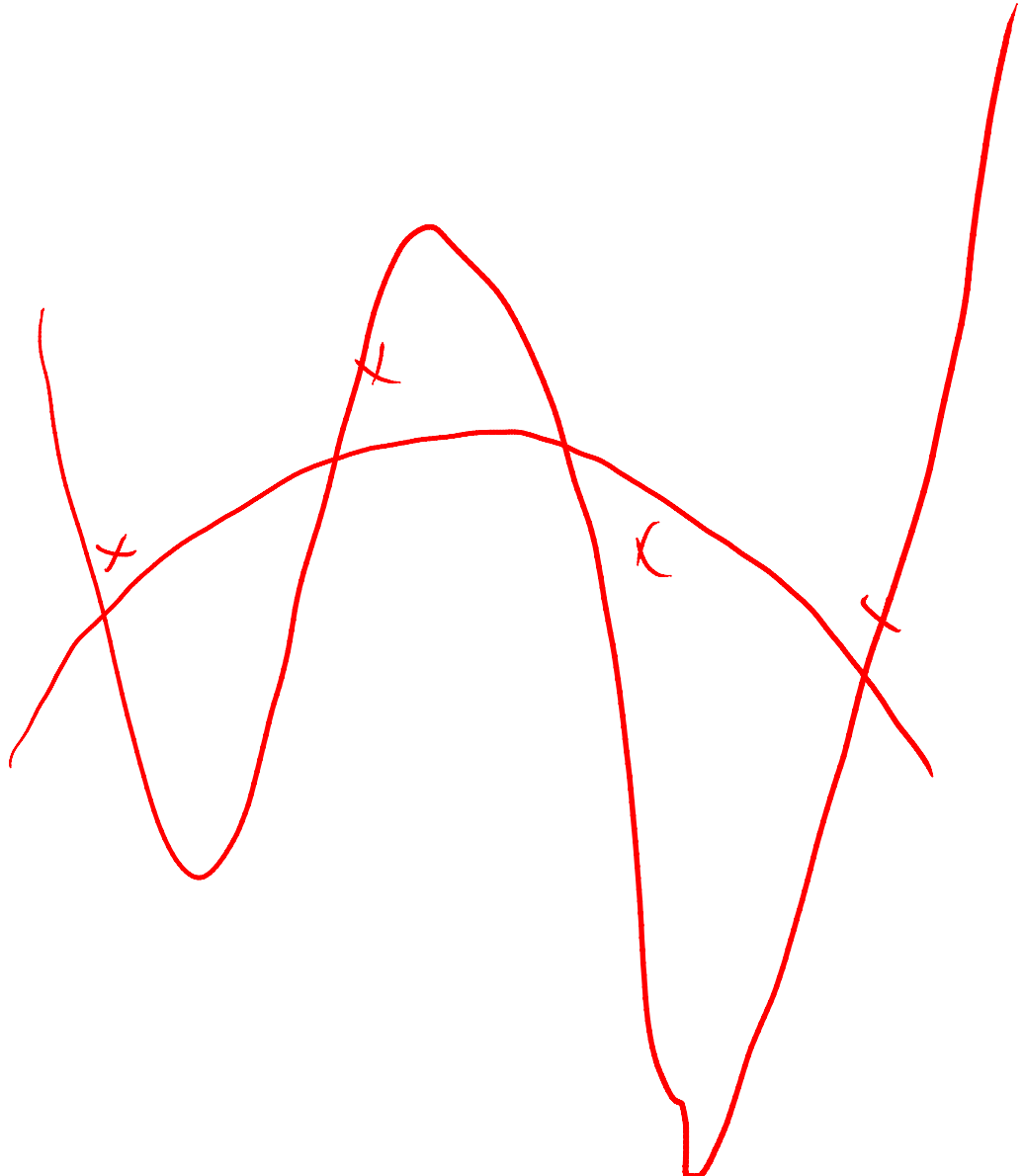
Learning – hypotheses

- During learning we choose a function (a model) from a family (the hypothesis space) based on a dataset.
- We choose it using **TRAINING DATA**, but we really want it to work on **UNSEEN TEST DATA**
- 2 sources of error:
 - **BIAS**: There is no function in the hypothesis space that faithfully represents the data
 - **VARIANCE**: The hypothesis space is so large and the data so scarce that we can't distinguish using the data a good function from a bad one.

Intuitive example: fitting polynomials.

Data:

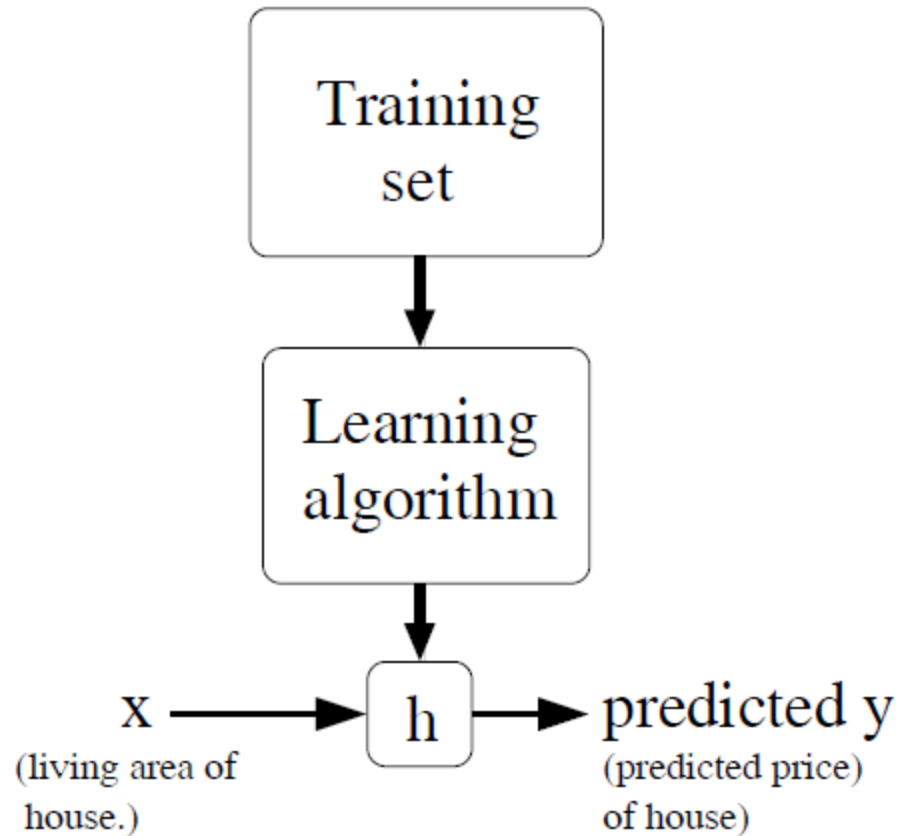
x	y
1	1
2	2
3	2
\vdots	\vdots



Please remember that...

- Learning from data is:
 - Choosing a hypothesis space (e.g. neural nets)
 - Choosing a hypothesis goodness criterion (e.g. log-likelihood)
 - Choosing the best hypothesis (i.e. optimization, e.g. SGD)
- Two major problems:
 - Mismatch between data and hypothesis space
 - Too large hypothesis space
- Learning Theory (PAC and Statistical Learning Theory):
 - Tells us a bound on the **error rate on unseen (test) data** that depends on the **error rate on the training data**, the **size of the hypothesis space**, and the **amount of training data**.
 - In other words: When one has sufficiently many data and a sufficiently small hypothesis space, the TRAINING and TESTING error will be similar

Learning



Types of learning

- Supervised:
 - The desired outputs (**labels**) are given
 - Data are (**input, output**) pairs
 - Goal is to learn the **input-output relation**
 - Examples:
 - Classification (discrete targets)
 - Regression (real-valued targets)
- Unsupervised:
 - No labels, just data points
 - Goal is to **describe** the data
 - Examples:
 - Clustering (find groups of closely related samples)
 - Dimensionality reduction
 - Component Analysis – PCA/ICA – express the data as a linear combination of basis functions
- Reinforcement (we have seen the policy gradient Pong example):
 - Feedback is given after a set of actions
 - E.g. learn to play a game based on its outcome only
 - Credit assignment problem: which actions were good, which were bad

Learners we know

- Least squares regression:

- Supervised learning

- Data are $\{(x^{(j)}, y^{(j)}), \in \mathbb{R}^n \times \mathbb{R}, j = 1..m\}$

- Hypothesis space:

$\Theta \in \mathbb{R}^n$ are the parameters

$$y \approx f(\mathbf{x}, \Theta) = \sum_{i=1}^n \Theta_i x_i = \Theta^T \mathbf{x}$$

- Training criterion (which hypothesis is the best):

$$\sum_{j=1}^m (f(\mathbf{x}^{(j)}, \Theta) - y^{(j)})^2$$

- Learning algorithm (how to choose the best hypothesis): mathematical optimization:

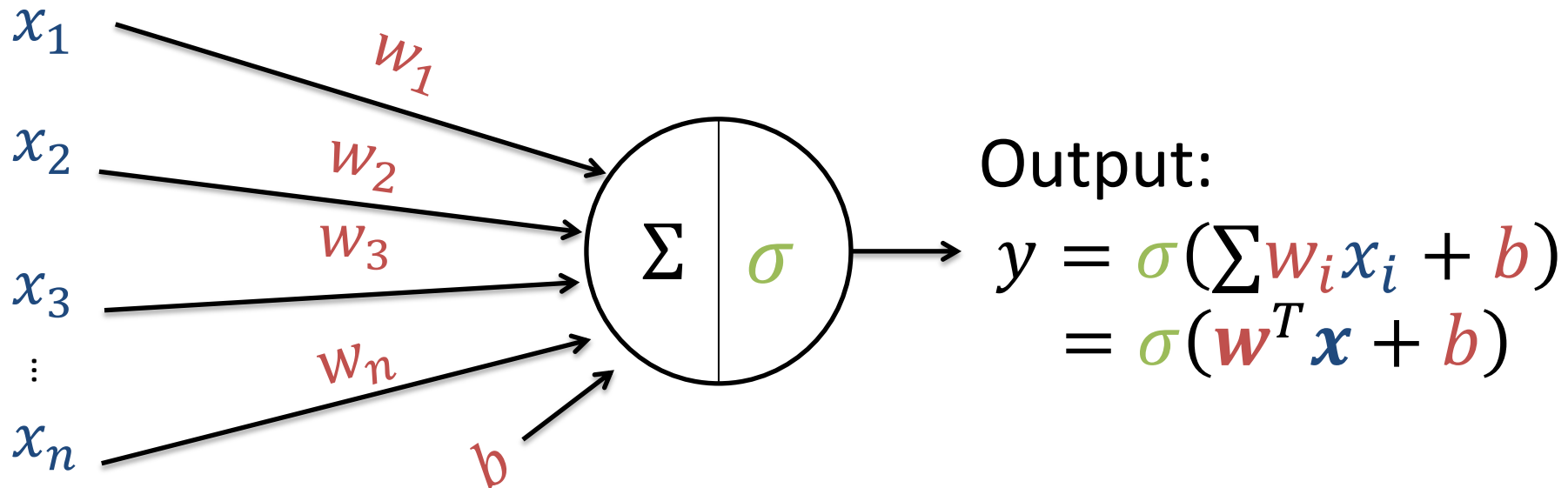
$$\Theta^* = \operatorname{argmin}_{\Theta} \sum_{j=1}^m (f(\mathbf{x}^{(j)}, \Theta) - y^{(j)})^2$$

x_1	x_2	x_3	y
0	0	0	0
1	10	3	1

Artificial Neural networks

- Are a family of functions that take real-valued vectors as inputs and produce real-valued vectors as outputs
- Are pictured as a **NETWORK** (directed graph) of simple computing nodes (the **NEURONS**)
- The function of the NN is stored in:
 - The architecture (which neurons are connected)
 - Weights (how strong the connections are)

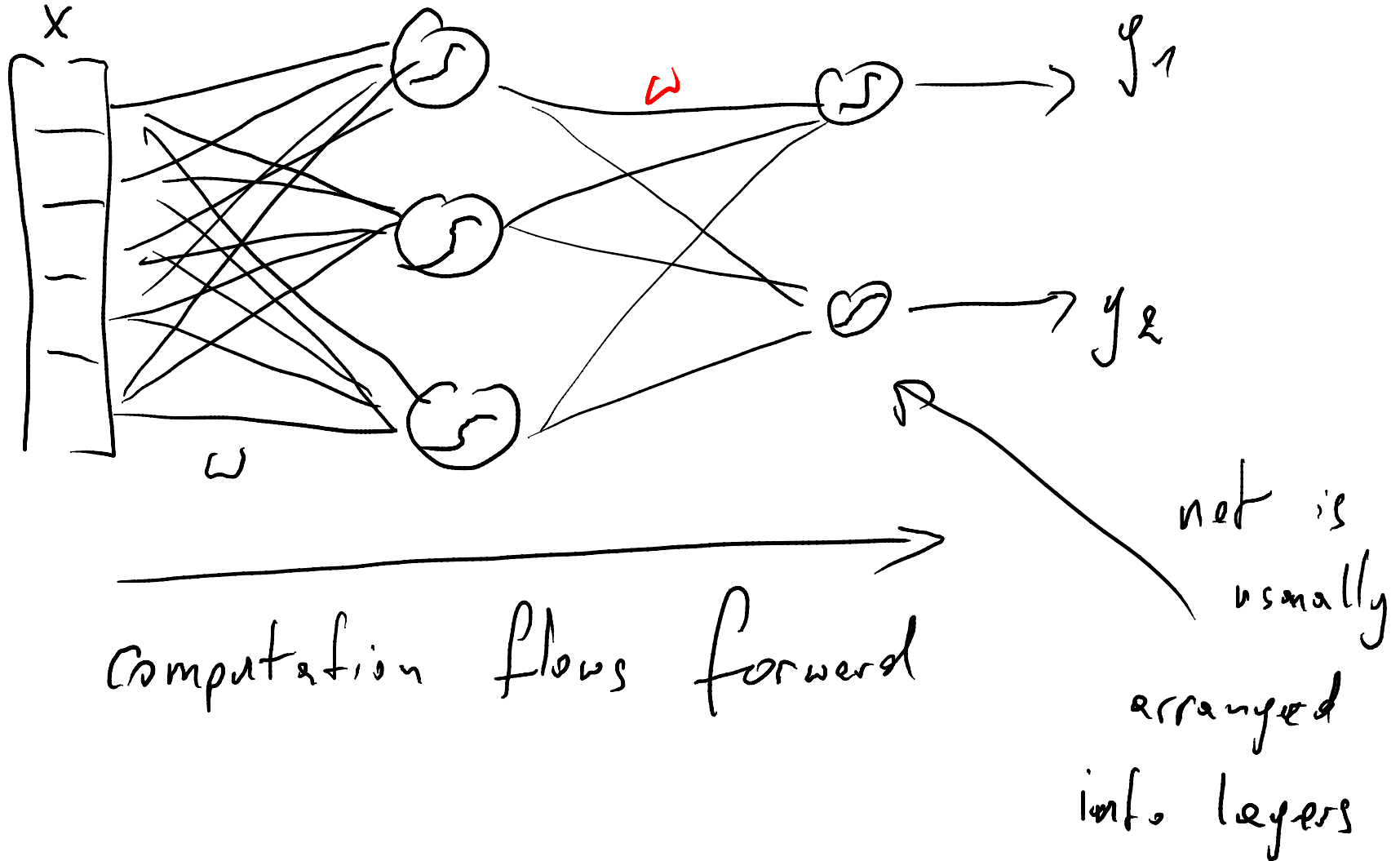
The artificial neuron (perceptron)



- x_i are the inputs
- w_i are the weights and b the bias
- Σ denotes the summation
- σ is a (possibly nonlinear) activation function

**w_i, b are
TUNABLE!!**

The Artificial Neural Network

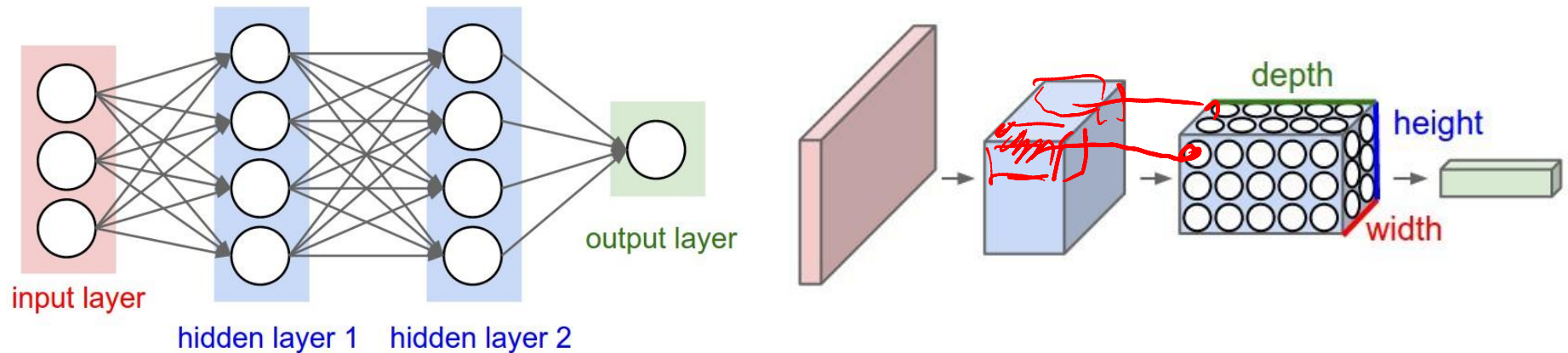


Sharing neurons - convolutions

Note: material from <http://cs231n.github.io/convolutional-networks/>

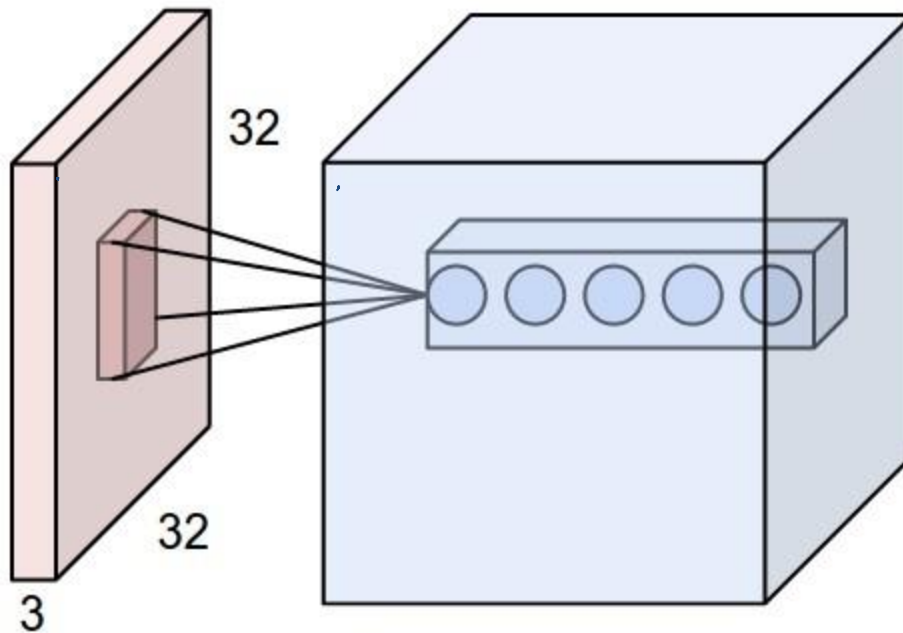
In a conv net we use a different connection pattern between layers:

- Typically we use an all-to-all scheme
- In a conv-net we use local connectivity!

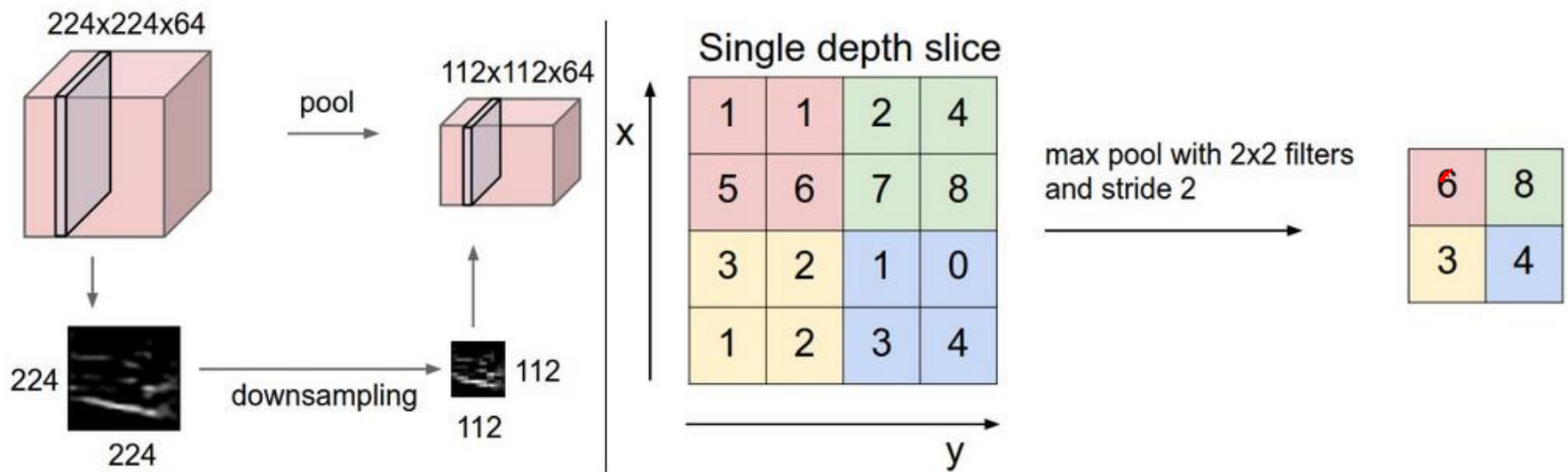


2D conv layer

- <http://cs231n.github.io/convolutional-networks/>

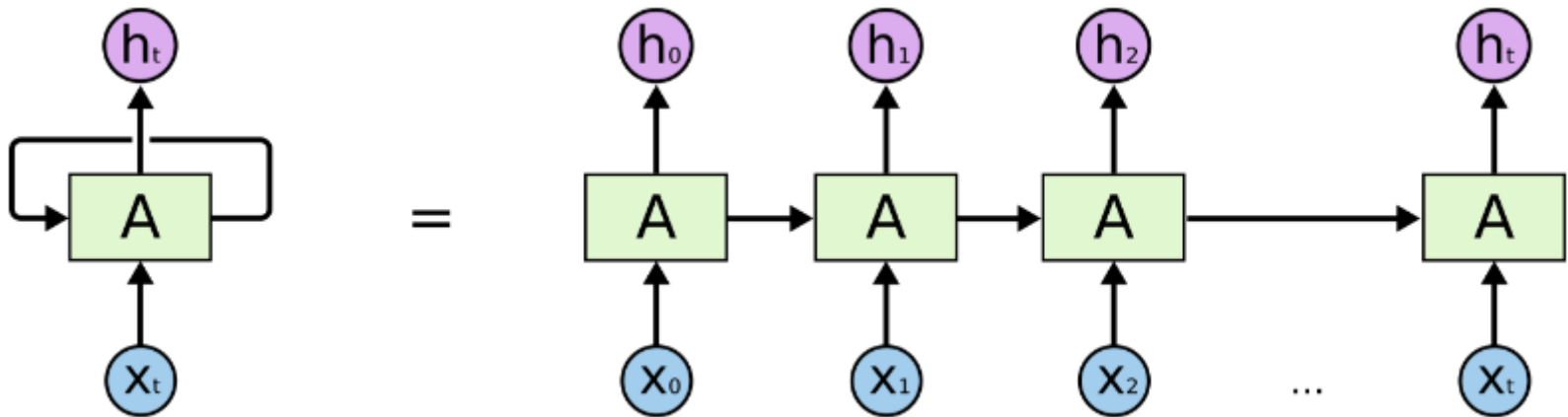


Pooling

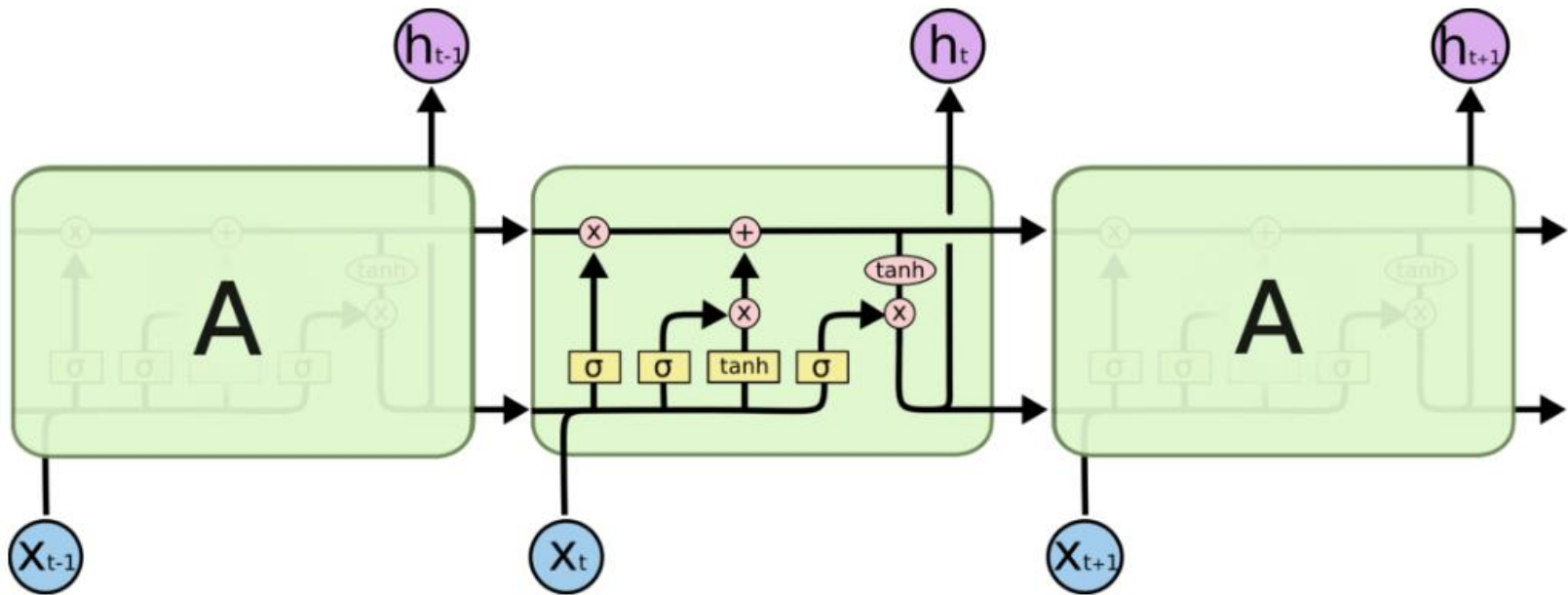


Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. **Left:** In this example, the input volume of size $[224 \times 224 \times 64]$ is pooled with filter size 2, stride 2 into output volume of size $[112 \times 112 \times 64]$. Notice that the volume depth is preserved. **Right:** The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2×2 square).

RNNs



LSTMs



Neural Net Uses

Please know about:

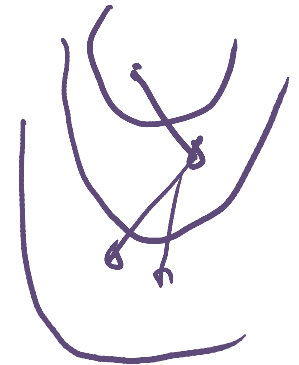
- Neural networks in image recognition
- Neural networks in language processing (language models, word vectors)
- Translation

How to train a net?

- Assume a certain architecture (#inputs, #outputs, connections, transfer functions)
- Then the network is fully specified by the weights
- Define a loss function – usually the negative of the logarithm of the likelihood (neg log-likelihood)
- Minimize the loss with respect to the weights
 - Initialize the weights to small random values – why?
 - Use gradient information to iteratively change weights – why?
 - Know how network architectural decisions impact gradient computations (e.g. activation function choice).
 - Backpropagation is a structured algorithm to compute the derivative of the loss wrt. weights. It is a direct consequence of the chain rule for differentiation.

Batch vs stochastic grad descent

- In batch gradient descent we compute the weight update on all (or a large subset) of available data:
 - Pros: the direction is reliable, can use second order methods and make large steps
 - Cons: many computations
- In on-line (stochastic) grad descent we compute the update using few (often just one) sample
 - Pros: very fast computations, good on large data sets
 - Cons: the weight update is „noisy” – must do small steps
 - Tricks:
 - Proper learning rate schedule e.g. $\alpha_t = \frac{b}{c+t}$
 - Typically want $\lim_{t \rightarrow \infty} \alpha_t = 0$ and $\lim_{k \rightarrow \infty} \sum_{t=1}^k \alpha_t = \infty$
 - Momentum – $\Delta \Theta_t = \alpha \nabla_{\Theta} (Loss) + \beta \Delta \Theta_{t-1}$



Practical aspects

- Neural Networks implement functions $\mathbb{R}^n \rightarrow \mathbb{R}^k$
- Need to encode inputs and outputs:
 - Discrete data is usually encoded using 1-of-N
e.g. Opt1 -> 100, Opt2 -> 010, Opt3 -> 001
 - Need to normalize inputs:
 - Zero mean, unit variance
 - Ideally decorrelate them (i.e. apply PCA or ZCA)
 - For classification – apply a sigmoid to limit the range of outputs, then treat the outputs as probabilities assigned by the net to a class
- For more see LeCun „Efficient Backprop”

Negative log likelihood

- Typically, we assume that the outputs of our model are probabilities of observing a data sample

– E.g. $P(y|x; \Theta) = \mathcal{N}(\mu = \Theta^T x, \sigma = 1)$

Then, under the assumption that samples are iid:

$$P(Y|X; \Theta) = \prod_{j=1}^m P(y^{(j)}|x^{(j)}; \Theta)$$
$$\ell(\Theta; Y, X) = - \sum_{j=1}^m \log \left(P(y^{(j)}|x^{(j)}; \Theta) \right)$$

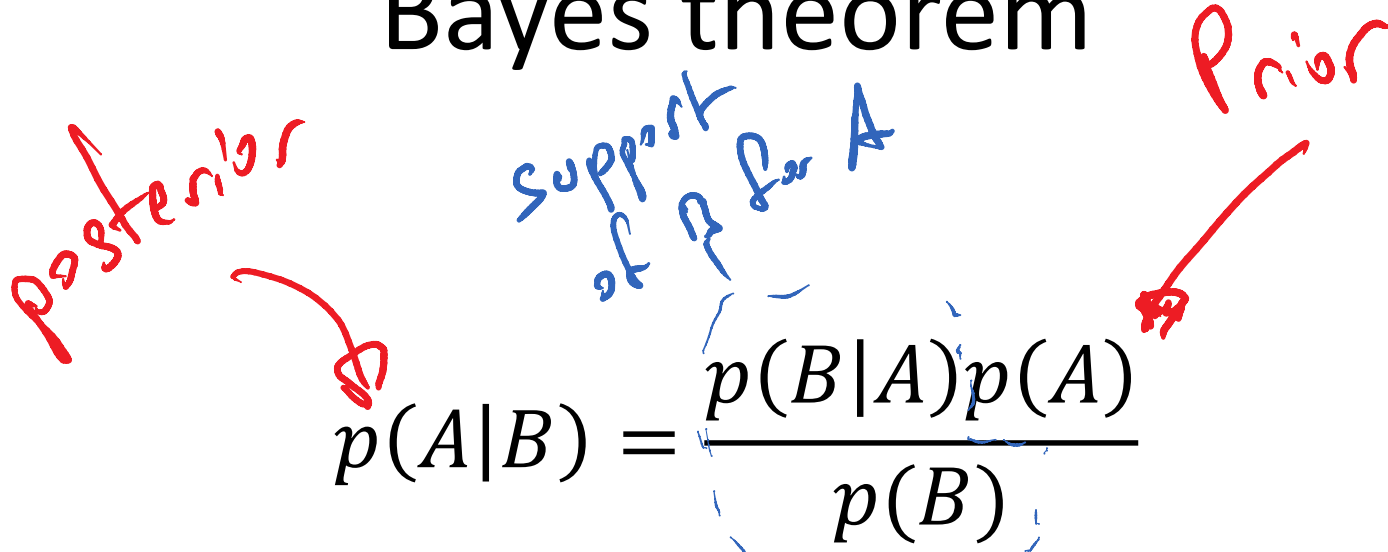
Training minimizes $\ell(\Theta; Y, X)$ over Θ

Regularization

- As we have seen, too „flexible” models are prone to overtraining.
- We need to prefer some hypotheses over others
 - Examples:
 - Linear models are simpler than polynomial
 - Small neural net is simpler than a large one
- Regularization serves to express our preferences about model simplicity
- Typically, we assign a **prior probability** to our models:

$$P(\Theta) = \prod_{i=1}^n \mathcal{N}(\Theta_i; \mu = 0, \sigma = \lambda)$$

Bayes theorem



Handwritten annotations on the Bayes' theorem formula:

- posterior** (red) with an arrow pointing to $p(A|B)$
- support of p for A** (blue) with a dashed blue circle around the numerator $p(B|A)p(A)$
- prior** (red) with an arrow pointing to $p(A)$

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$$

Interpretation: how our estimate of A changes after seeing B .

Why?

$$p(A, B) = p(A|B)p(B) = p(B|A)p(A)$$

Then divide by $p(B)$

Bayesian approach to ML

- What is the model probability after seeing the samples S ?

$$p(\Theta|S) = \frac{p(S|\Theta)p(\Theta)}{p(S)}$$

How to make predictions? Integrate over all models:

$$p(y|x, S) = \int_{\Theta} p(y|x, \Theta)p(\Theta|S)d\Theta$$

Then

$$E[y|x, S] = \int_y yp(y|x, S)dy$$

But computing $p(y|x, S)$ is often intractable :(

Maximum-a-posteriori

- Instead of integrating over all Θ
- Use the maximally probable Θ :

$$\begin{aligned}\Theta_{MAP} &= \arg \max_{\Theta} p(\Theta|S) \\ &= \arg \max_{\Theta} \left(\prod_{i=1}^m p(y^{(i)}|x^{(i)}, \Theta) \right) \underbrace{p(\Theta)}\end{aligned}$$

- It's like Max. Likelihood with the extra term (which is the regularization).

Gaussian model MAP

$$\arg \max_{\Theta} \prod_{i=1}^m p(y^{(i)} | x^{(i)}, \Theta) p(\Theta) =$$

$$\arg \max_{\Theta} \sum_{i=1}^m \underbrace{\log p(y^{(i)} | x^{(i)}, \Theta)}_{\text{ML term}} + \underbrace{\log(p(\Theta))}_{\text{REGULARIZATION}}$$

Now if Θ_j are Gaussian with zero-mean,

$$\log(p(\Theta)) \propto \sum_{j=1}^n (\Theta_j)^2$$

Thus our minimization criterion gets an extra term, whose derivative is:

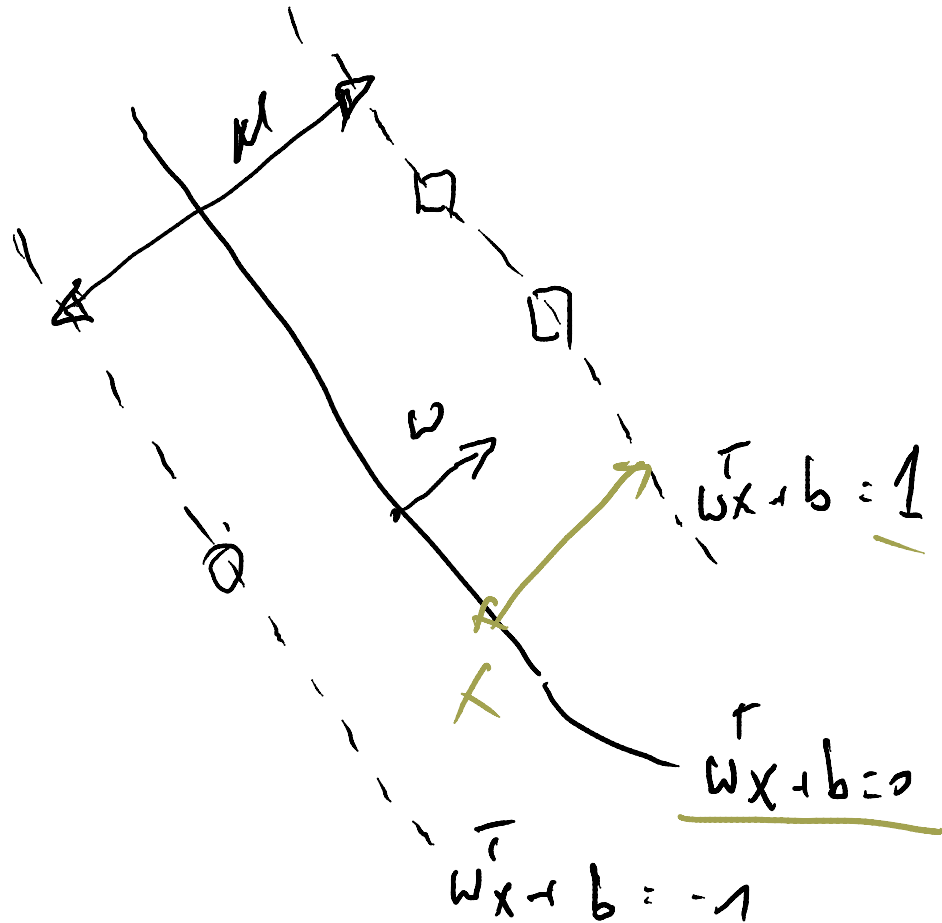
$$\nabla_{\Theta} \log(p(\Theta)) \propto \Theta$$

Longer example: SVM

- Task: 2-class classification
- Idea: find a hyperplane yielding max margin



The margin



$$\begin{aligned}
 w^T \left(x + \frac{Mw}{2\|w\|} \right) + b &= \\
 &= w^T x + b + \frac{M\|w\|^2}{2\|w\|} = \\
 &= \frac{M}{2} \|w\| = 1
 \end{aligned}$$

Thus:

$$M = \frac{2}{\|w\|}$$

Maximum margin => minimum weights!

Find a tradeoff between margin width and number of errors!

Solve:

$$\min \frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_i \xi_i$$

s. t.:

$$\underline{y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + b)} \geq 1 - \underline{\xi_i} \quad \forall i$$
$$\underline{\xi_i \geq 0} \quad \forall i$$

$$\begin{aligned} & \frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_i \xi_i \\ \text{s.t.: } & \underline{y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + b)} \geq 1 - \xi_i \quad \forall i \\ & \underline{\xi_i \geq 0} \quad \forall i \end{aligned}$$

Kernels – nonlinear SVM

- The SVM finds weights such to minimize

$$\frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_i \xi_i$$

$$\text{s. t. : } y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \forall i$$

It turns out we can express weights as a linear combination of training samples:

$$\underline{\mathbf{w}} = \sum_i \alpha_i \mathbf{x}^{(i)}$$

Where α_i are the Lagrange multipliers of the inequality constraints.

Kernels – nonlinear SVM

Map (nonlinearly) $x \rightarrow \phi(x)$

$$\mathbf{w} = \sum_i \alpha_i \phi(\mathbf{x}^{(i)})$$

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b = \sum_i \alpha_i \underbrace{\phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x})}_{K(\mathbf{x}^{(i)}, \mathbf{x})} + b$$

We only need dot-products in the feature $\phi(\cdot)$ space. Let $K(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T \phi(\mathbf{y})$

Then $f(\mathbf{x}) = \sum_i \alpha_i \underline{K(\mathbf{x}^{(i)}, \mathbf{x})} + b$

K is the **kernel function**. We never need to compute $\phi(\mathbf{x})$. We can always use it implicitly through the kernel function.

Only the α_i corresponding to errors and points inside the margin are nonzero:

$\mathbf{x}^{(i)}: \alpha_i \neq 0$ are called **support vectors**

Exemplary Kernels

- Gaussian: $K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}\right)$
 - Note: there is a family of Neural Networks (we haven't studied them), called Radial-Basis Function Networks that look like an SVM with Gaussian kernels.
- Polynomial: $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + c)^d$

Putting it all together

- MAP learning results in two terms:

$$\sum_{i=1}^m \log p(y^{(i)} | x^{(i)}, \Theta) + \lambda \log(p(\Theta))$$

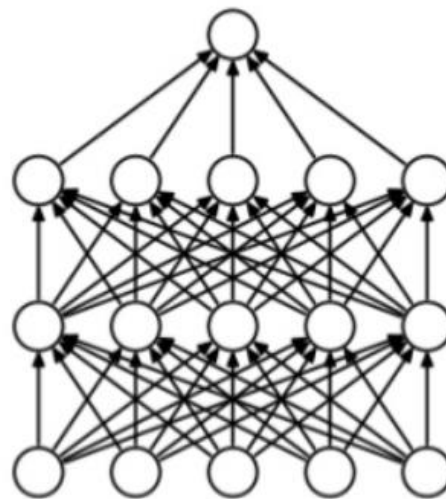
- The SVM similarly has two terms:

$$\frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_i \xi_i \quad \text{s.t.: } y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \forall i$$

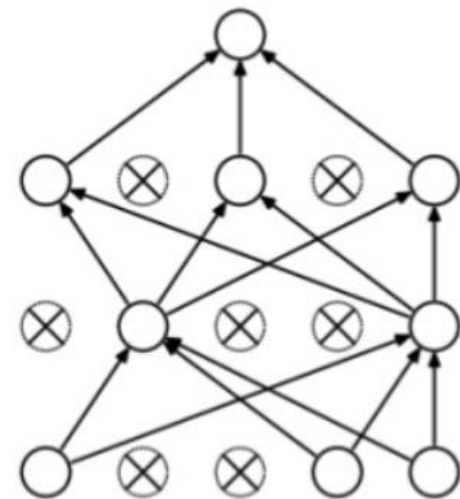
- The two terms (with their constants) allow us to balance MODEL COMPLEXITY and TRAINING LOSS
- Constants C , λ and other model parameters, such as number of neurons, type of kernel function, are set via CROSS-VALIDATION

Other regularization methods

- You can average many models – this nearly always boosts accuracy at the cost of making more computations.
- For neural networks try dropout:
 - For each sample remove some neurons (typically $\frac{1}{2}$)
 - This is like we were sampling a new net for each sample. However, all these networks share weights.
 - During testing use all neurons (need to divide their activations)
 - Net should overfit less



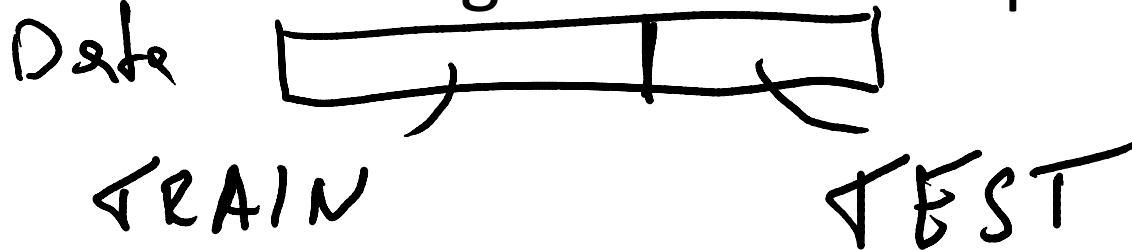
(a) Standard Neural Net



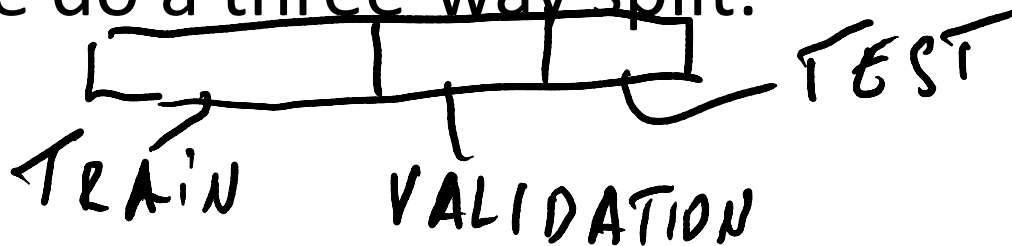
(b) After applying dropout.

Honest estimates: Hold-out set

- Split the training data into two parts:



- Train only on training, then test on testing.
- Often we do a three-way split:



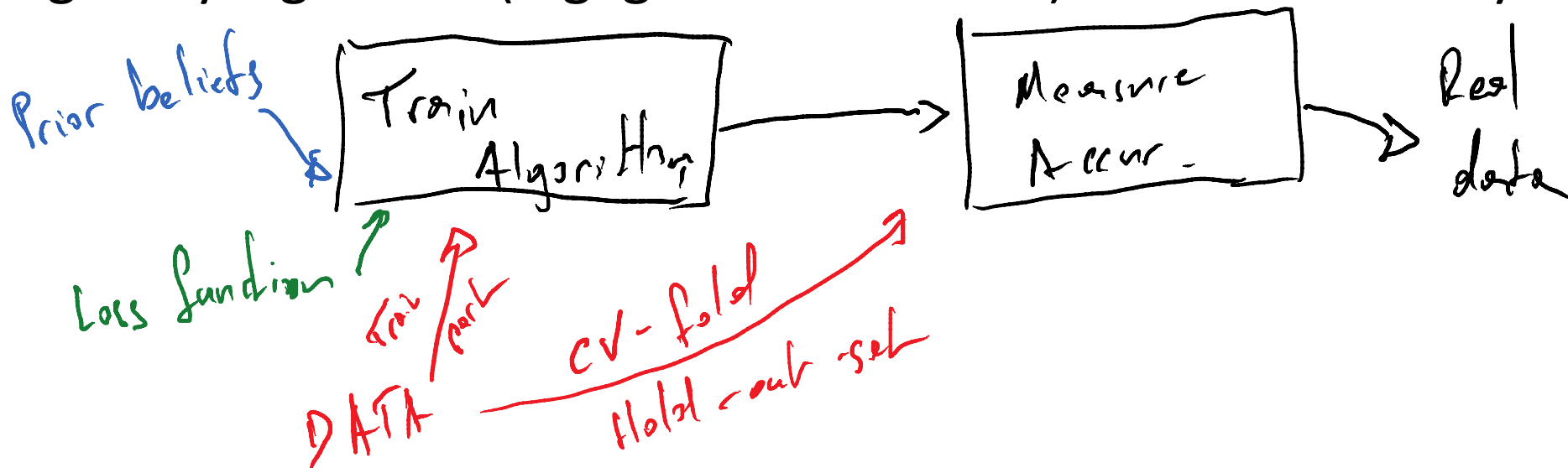
- Then:
 - Train many models on training (different algos, parameters)
 - Use validation to choose best model
 - Test on testing

Cross-validation



- Hold-out set makes inefficient data use
- Idea:
 - Divide the data into k sets ($\sim 5, 10$)
 - For $i=1..k$
 - Train on all but the i -th set
 - may further split to choose the model...
 - Test on the i -th set
 - Finally:
 - take the answers on the testing sets and use them to compute the performance measures
- Extreme case: leave-one-out (jackknife) – always use all but one sample to train!
- We also used the bootstrap – repeated sampling with replacement from the training set.

Approximations we take

- We want: accuracy on UNKNOWN TEST DATA
- Approximation: Cross-Validation, hold-out set
- But we can't directly optimize accuracy (non-differentiable, NP-hard...)
 - Also, criteria, like max. margin often enhance generalization
- Thus optimize a loss function as a proxy for accuracy
- This is often impossible to do exactly – usually use some greedy algorithm (e.g. gradient descent) started randomly



Errors can come at all stages

- Data:
 - Is it representative of the problem
 - Does it cover all possible variations (e.g. in France “z” is )
 - Can you get more of it? Generate? Transform? 
- Prior beliefs:
 - Does the architecture you choose match the problem?
 - Maybe you know something (e.g. invariants, predominating probability distribution...)
- Loss function:
 - Does it make sense? Is it for classification/regression? Do smaller loss correspond to better performance?
- Training algorithm:
 - Do you reach the minimum of what you optimize?
 - Intentionally? How about early stopping?
- Performance measures:
 - do you separate train from test data?
 - How do train and test errors compare?

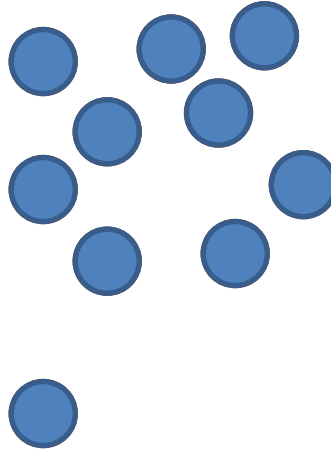
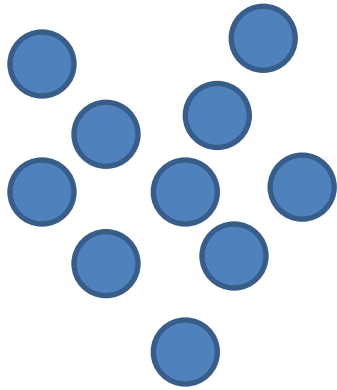
Example

Linear classifier makes 10% errors

Neural net with 1 hidden layer makes 20% 😞

- Use the same loss – e.g. cross-entropy, which one has the lowest?
(don't change training, just loss computation):
 - Linear classifier -> do you train the net correctly??
 - Maybe use a second order method or SGD?
 - Maybe the net is too small/too regularized?
 - Network -> how is your train and test error, do you over-fit?
 - If they are trained using a different loss, can you try the net with the loss of the linear classifier?
 - Try a smaller network
 - Do you use regularization? Early-stopping?
 - Can you get more training data?
 - Maybe the linear classifier is also over-fitting?

Unsupervised learning



In supervised learning we have labels
In unsupervised we don't have them!

Describe the data!:

- Find clusters (distinct groups of similar points)
- Reduce the dimensionality
- Find good features that describe the data
- Find and fit a probabilistic model that generated the data

K-Means – a basic algorithm

Divides the data into globular clusters according to some distance measure (typ. Euclidean)

Input: m input patterns $x^{(i)}$

1. Initialize K cluster centers $\mu_1 \dots \mu_K$ randomly, to some input patterns...

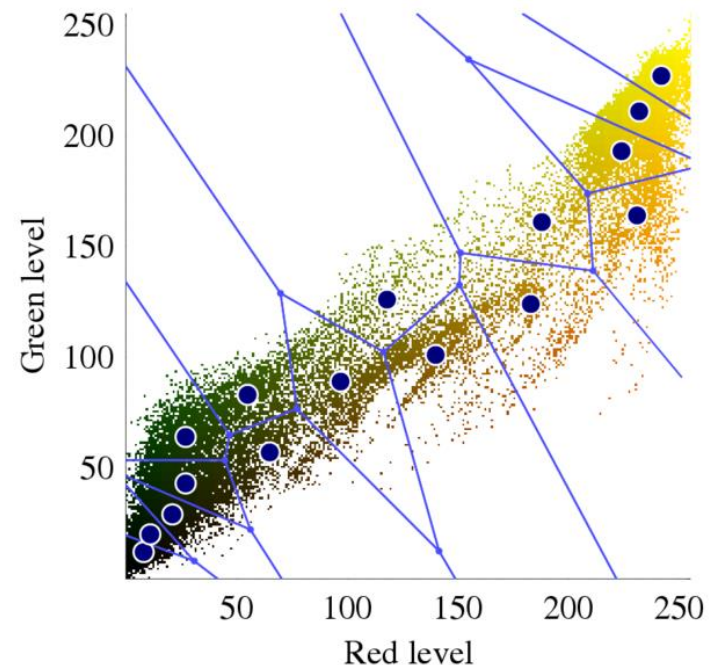
2. Loop until convergence:

1. For all i : set $c^i := \arg \min_j \|x^{(i)} - \mu_j\|^2$

2. For all j : set $\mu_j := \frac{\sum_i [c^{(i)}=j] x^{(i)}}{\sum_i [c^{(i)}=j]}$

The K-Means optimization problem

- $J(c, \mu) = \sum_i \|x^{(i)} - \mu_{c(i)}\|^2$
- The K-means algorithm repeatedly minimizes this over c , then over μ etc.
- Initialization:
 - Random
 - To some data samples
 - Bisecting:
 - Start with two clusters
 - Then divide them
 - Then repeat



Kohonen maps

K-means with topology:

- Assume a topology of units
- Iterate over data samples x^i
 - Find the Best Matching Unit
$$bmu = \arg \min_u \|x^i - w^u\|$$
 - Move the weights of the BMU and its neighbors in the chosen topology towards x^i :
$$\Delta w^j = \alpha N(j, bmu) x^i$$
- In a Kohonen map units close in the chosen topology point to similar data-space regions.

Gaussian mixtures and EM

- Assume the data comes from a mixture of Gaussian distributions.
- Probabilistic model for data:
 - First pick a cluster id $p(z^{(i)} = j) = \phi_j$
 - Then sample from the cluster
$$p(x^{(i)} | z^{(i)} = j) = \mathcal{N}(x; \mu_j, \Sigma_j)$$
- Thus the log-likelihood is:

$$\ell(\underbrace{\phi, \mu, \Sigma}) = \sum_i \log \left(\sum_j p(x^i | z^i = j) p(z^i = j) \right)$$

EM algorithm

Initialize randomly or from K-means

Iterate between:

- Estimate probability of $w_j^{(i)} = p(z^{(i)} = j)$

- From the Bayes rule

$$w_j^{(i)} = p(z^{(i)} = j) = \frac{p(x^{(i)} | z^{(i)} = j) p(z^{(i)} = j)}{\sum_l p(x^{(i)} | z^{(i)} = l) p(z^{(i)} = l)}$$

- Maximize log-likelihood:

$$\phi_j = \frac{1}{\#samples} \sum_i w_j^{(i)}$$

$$\mu_j = \frac{\sum_i w_j^{(i)} x^{(i)}}{\sum_i w_j^{(i)}}$$

$$\Sigma_j = \frac{\sum_i w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_i w_j^{(i)}}$$

PCA

- Idea: find a projection direction that will maximize the variance of the data
- X – data matrix (each column is a sample)
- v – projection direction
- $v^T X$ – projected data
- $\overline{v^T X}$ - projection mean $\overline{v^T X} = \frac{1}{N} \sum_{i=1}^N v^T x^{(i)}$
- $\frac{1}{N} (v^T X - \overline{v^T X}) (v^T X - \overline{v^T X})^T$ - projection variance
- Goal: find v maximizing variance such that $v^T v = 1$

PCA - implementation

PCA looks for eigenvectors of data covariance matrix:

- Normalize data – subtract mean
- Compute covariance $\Sigma = XX^T$

- Find eigendecomposition:

$$XX^T = V\lambda V^T$$

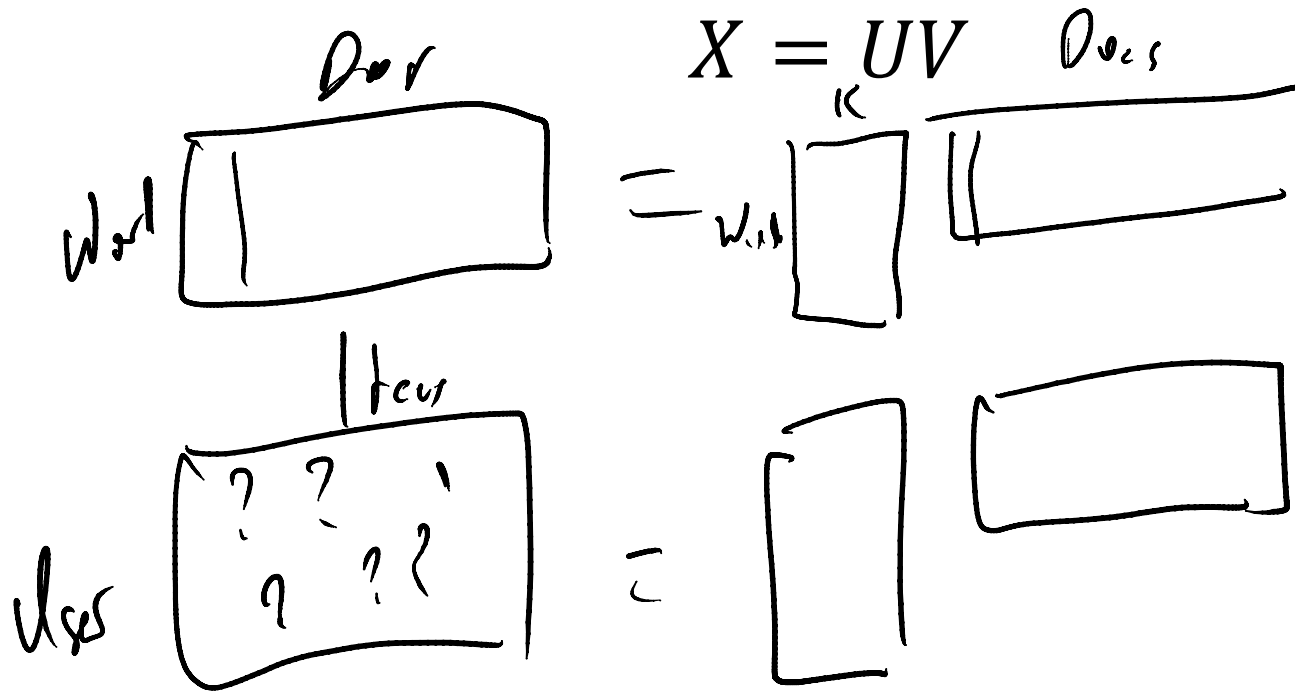
- Select the eigenvectors corresponding to the largest eigenvalues

PCA - interpretation

- PCA is a linear transformation that:
 - Maximizes the variation of the projection
 - Minimum amount of data variability lost, hopefully we lose only noise!
 - The projected data are:
 - Decorrelated
 - Normalized
- PCA is a good data preprocessing algorithm
 - It is quite common to do a PCA prior to training

Matrix factorization

- Express the data matrix as a product of two low-rank matrices



- Commonly used for: text representation, rating prediction

Important topics about learning

- Understand maximum log-likelihood and maximum a posteriori training rules
- Be able to write the negative-log likelihood for a small model (e.g. finding a population's mean)
- Be able to tell the probabilistic interpretation of a model (what is the interpretation of SoftMax, least squares etc.)?

Important topics for supervised learning

- Define the learning problem
- Linear classifiers:
 - Least-squares regression and logistic regression
 - Probabilistic interpretations
- SVM (need not know the derivation of the kernelized form, but need to know about kernels!)
- Neural network – define, compute derivatives – chain rule, backprop algorithm , how to train
 - Batch vs on-line training
 - Regularization, weight decay, dropout
 - Data preparation
 - Why random initialization
- Honest estimates: cross-validation

Important topics about neural networks

- Convolutional networks:
 - Know about convolution and pooling. What is their purpose?
 - Know for which data they are useful to use.
- Recurrent networks:
 - Be able to explain typical problems of gradient vanishing/exploding
 - Describe parts of LSTM cell, understand the operation.

Important topics – unsupervised learning

- K-Means
- PCA
 - Explain what it does and how to compute
 - Know when to use
- Matrix factorization – encoder and decoder networks