Machine Learning/Advanced Machine Learning

Lecture 6.2:
More on Neural Networks

Sami S. Brandt

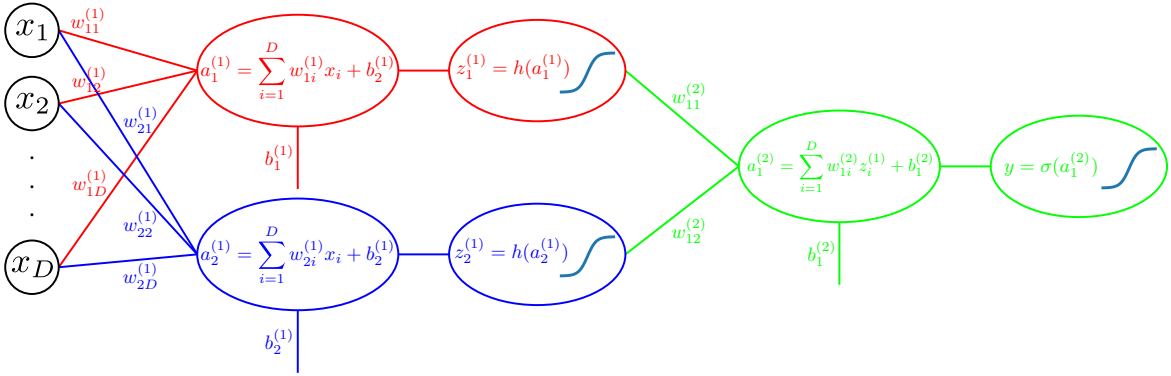
Department of Computer Science
IT University of Copenhagen

Based on slides originally made by Jes Frellsen

3 October 2019

IT UNIVERSITY OF COPENHAGEN

Recap: a two layer neural network



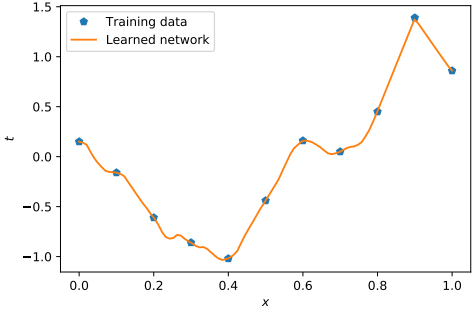
$\mathbf{x}$ $\quad \mathbf{a}^{(1)} = W^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \quad \mathbf{z}^{(1)} = h(\mathbf{a}^{(1)}) \quad \mathbf{a}^{(2)} = W^{(2)}z^{(1)} + b^{(2)} \quad \mathbf{y} = \sigma(\mathbf{a}^{(2)})$

$$y = \sigma(W^{(2)}h(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(1)})$$

Universal approximation theorem

Neural networks are universal approximators (Bishop):

"A two-layer network with linear outputs can uniformly
approximate any continuous function
on a compact input domain (compact subset of $\mathbb{R}^N$)
to **arbitrary accuracy**
provided the network has
sufficiently large number of hidden unites"



Example: Notebook 1

Example: Notebook 1

We can use better optimizers than gradient descent, e.g.:

- Adagrad
- RMSprop
- Adam

These make use of **adaptive** learning rate and/or **momentum**.

We see that the ReLU is easier to optimize.

Outline of lecture

Regularization in Neural Networks

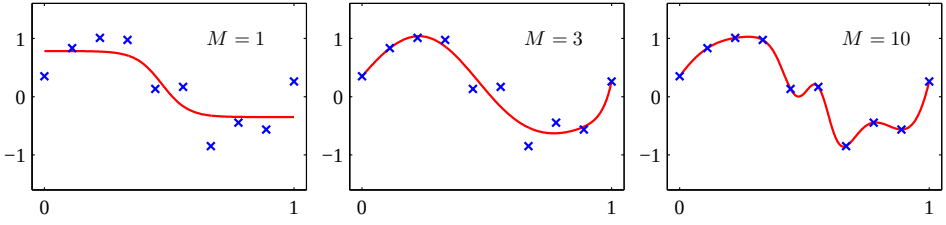
Backpropagation

Radial Basis Functions

A brief introduction to CNNs

Overfitting

As we increase the **complexity** of the model, we risk overfitting:



Two-layer neural network with M hidden units

Regularization: ℓ_2-norm

To avoid overfitting, we also add a **regularization term** to the error function:

$$\tilde{E}(\mathbf{W}, \mathbf{b}) = E(\mathbf{W}, \mathbf{b}) + \frac{\lambda}{2}\|\mathbf{W}\|_2$$

where λ is the regularization coefficient/factor[1] and

$$\|\mathbf{W}\|_2 = \sqrt{\sum_i W_i^2}.$$

[1] More generally the ℓ_p-norm is given by

$$\|\mathbf{W}\|_p = \left(\sum_i W_i^p\right)^{1/p}$$

Regularization: ℓ_2-norm

To avoid overfitting, we also add a **regularization term** to the error function:

$$\tilde{E}(\mathbf{W}, \mathbf{b}) = E(\mathbf{W}, \mathbf{b}) + \frac{\lambda}{2}\|\mathbf{W}\|_2$$

where λ is the regularization coefficient/factor[1] and

$$\|\mathbf{W}\|_2 = \sqrt{\sum_i W_i^2}.$$

How do we select the optimal value of λ?

[1] More generally the ℓ_p-norm is given by

$$\|\mathbf{W}\|_p = \left(\sum_i W_i^p\right)^{1/p}$$

Regularization: ℓ_2-norm

To avoid overfitting, we also add a **regularization term** to the error function:

$$\tilde{E}(\mathbf{W}, \mathbf{b}) = E(\mathbf{W}, \mathbf{b}) + \frac{\lambda}{2}\|\mathbf{W}\|_2$$

where λ is the regularization coefficient/factor[1] and

$$\|\mathbf{W}\|_2 = \sqrt{\sum_i W_i^2}.$$

How do we select the optimal value of λ?

We can try different values and selected the one with the lowest error on a validation set.

Example: Notebook 2

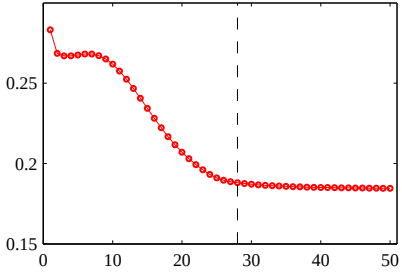
[1] More generally the ℓ_p-norm is given by

$$\|\mathbf{W}\|_p = \left(\sum_i W_i^p\right)^{1/p}$$

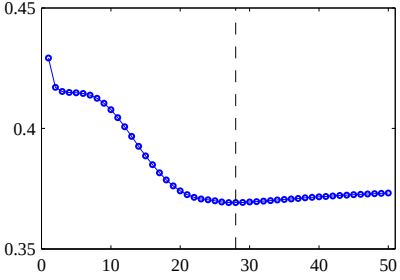
Regularization: early stopping

Stop the training when it is minimal on a independent validation set.

We test how well the model generalizes.



Training set



Validation set

There are heuristics for when to stop.

Example: Notebook 3

Outline of lecture

Regularization in Neural Networks

Backpropagation

Radial Basis Functions

A brief introduction to CNNs

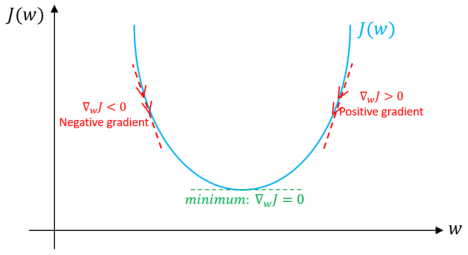
Parameter optimization and gradient descent

We want to find $\mathbf{W} = (\mathbf{W}, \mathbf{b})$ that **minimizes** $E(\mathbf{W})$, i.e. find $\mathbf{W}$ such that $\nabla E(\mathbf{W}) = 0$.

Gradient descent starts with an initial random point $\mathbf{W}^{(0)}$, and iteratively refines it by following the steepest descent direction:

$$\mathbf{W}^{(\tau+1)} = \mathbf{W}^{(\tau)} - \eta \nabla E(\mathbf{W}^{(\tau)})$$

where η is the called the **learning rate**.



Normally the gradient is calculate on the full dataset (*batch* optimization).

To avoid getting stuck in local minima, we can calculate the gradient on **mini-batches**:

$$\mathbf{W}^{(\tau+1)} = \mathbf{W}^{(\tau)} - \eta \nabla E_s(\mathbf{W}^{(\tau)})$$

where E_s is the error on a subset of the data.

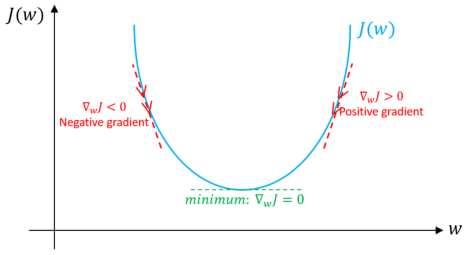
Parameter optimization and gradient descent

We want to find $\mathbf{W} = (\mathbf{W}, \mathbf{b})$ that **minimizes** $E(\mathbf{W})$, i.e. find $\mathbf{W}$ such that $\nabla E(\mathbf{W}) = 0$.

Gradient descent starts with an initial random point $\mathbf{W}^{(0)}$, and iteratively refines it by following the steepest descent direction:

$$\mathbf{W}^{(\tau+1)} = \mathbf{W}^{(\tau)} - \eta \nabla E(\mathbf{W}^{(\tau)})$$

where η is the called the **learning rate**.



Normally the gradient is calculate on the full dataset (*batch* optimization).

To avoid getting stuck in local minima, we can calculate the gradient on **mini-batches**:

$$\mathbf{W}^{(\tau+1)} = \mathbf{W}^{(\tau)} - \eta \nabla E_s(\mathbf{W}^{(\tau)})$$

where E_s is the error on a subset of the data.

How do we actually calculate $\nabla E_s(\mathbf{W}^{(\tau)})$?

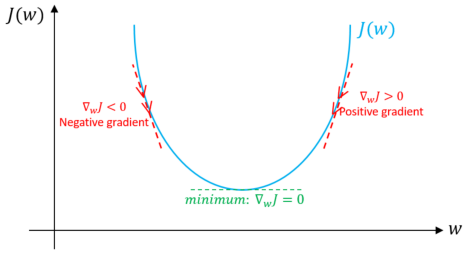
Parameter optimization and gradient descent

We want to find $\mathbf{W} = (\mathbf{W}, \mathbf{b})$ that **minimizes** $E(\mathbf{W})$, i.e. find $\mathbf{W}$ such that $\nabla E(\mathbf{W}) = 0$.

Gradient descent starts with an initial random point $\mathbf{W}^{(0)}$, and iteratively refines it by following the steepest descent direction:

$$\mathbf{W}^{(\tau+1)} = \mathbf{W}^{(\tau)} - \eta \nabla E(\mathbf{W}^{(\tau)})$$

where η is the called the **learning rate**.



Normally the gradient is calculate on the full dataset (*batch* optimization).

To avoid getting stuck in local minima, we can calculate the gradient on **mini-batches**:

$$\mathbf{W}^{(\tau+1)} = \mathbf{W}^{(\tau)} - \eta \nabla E_s(\mathbf{W}^{(\tau)})$$

where E_s is the error on a subset of the data.

How do we actually calculate $\nabla E_s(\mathbf{W}^{(\tau)})$?
Backpropagation!

Outline of lecture

Regularization in Neural Networks

Backpropagation

Radial Basis Functions

A brief introduction to CNNs

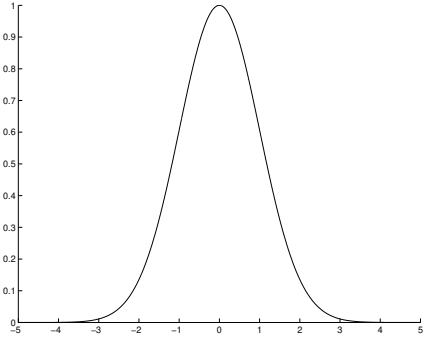
Radial Basis Function Networks

In MLP networks, hidden units use **dot product** and **sigmoid** as the non-linearity

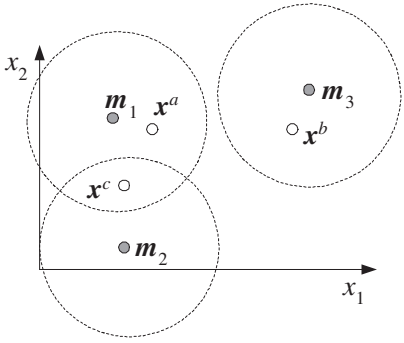
- This is a **distributed representation**

RBF networks, only a few units are active at a time

- They use a **local representation**

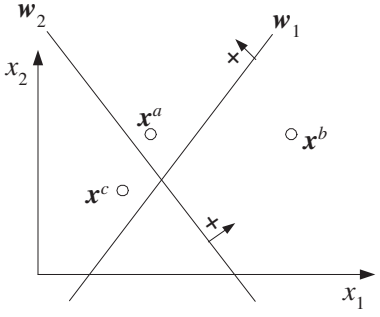


Local vs. Distributed Representation



Local representation in the space of (p_1, p_2, p_3)

$\boldsymbol{x}^a$: (1.0, 0.0, 0.0)
$\boldsymbol{x}^b$: (0.0, 0.0, 1.0)
$\boldsymbol{x}^c$: (1.0, 1.0, 0.0)



Distributed representation in the space of (h_1, h_2)

$\boldsymbol{x}^a$: (1.0, 1.0)
$\boldsymbol{x}^b$: (0.0, 1.0)
$\boldsymbol{x}^c$: (1.0, 0.0)

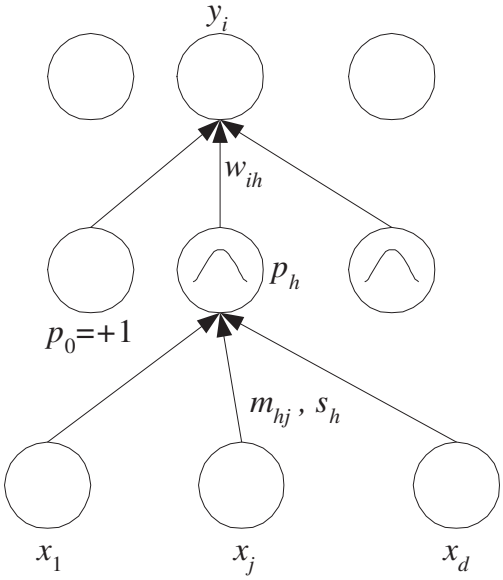
Radial Basis Functions

RBFs are locally tuned units, based on **Gaussian** kernels

$$p_h(\mathbf{x}) = \exp\left(-\frac{\mathbf{x}-\mathbf{m}_h}{2s_h^2}\right)$$

The response of the network

$$y_i(\mathbf{x}) = \sum_h w_{ih} p_h(\mathbf{x}) + w_{i0}.$$



Training of RBFs

Hybrid learning

- Find First layer centres and spreads by (unsupervised) k-means
- Second layer weights: (supervised) gradient decent

Fully supervised learning

- Backpropagation

Normalised Basis Functions

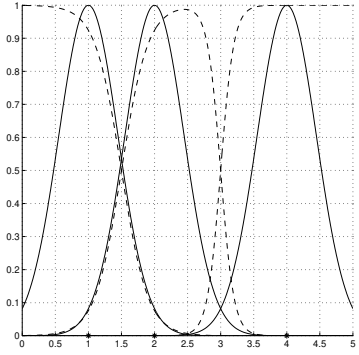
If a data point is outside the numerical range of the kernel functions, the network will have zero output

The **Normalisation** step circumvents the problem by

$$g_h(\mathbf{x}) = \frac{p_h(\mathbf{x})}{\sum_{h'} p_{h'}(\mathbf{x})}.$$

The response of the network is modified to

$$y_i(\mathbf{x}) = \sum_h w_{ih} g_h(\mathbf{x}).$$



Competitive Basis Functions

- So far, the RBF network output was a weighted sum of contributions
- Another approach is to use a competitive basis functions defined by the mixure model

$$p(\mathbf{r}|\mathbf{x}) = \sum_h p(h|\mathbf{x})p(\mathbf{r}|h,\mathbf{x})$$

- Regression: minimise the negative log-likelihood by gradient decent.
- Classification: maximise the likelihood by EM.

Outline of lecture

Regularization in Neural Networks

Backpropagation

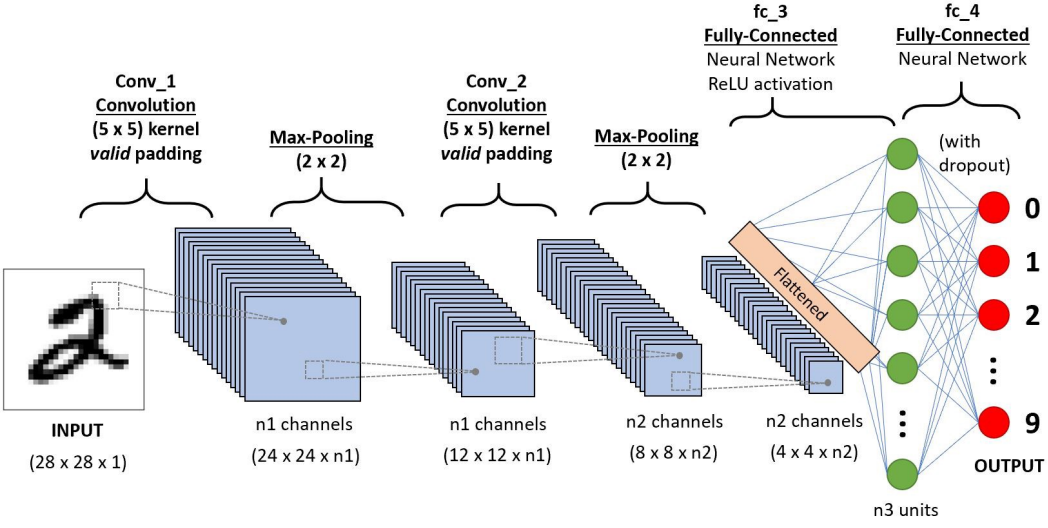
Radial Basis Functions

A brief introduction to CNNs

Brief Introduction to CNNs

https://youtu.be/YRhxdVk_sIs

A CNN for Handwritten Character Recognition



From https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Next week

Kernel Methods

Graphical Models

References I

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.