# Lecture 9: Multi-Layer Neural Networks, Error Backpropagation
# CS480/680 Intro to Machine Learning

2023-2-7

Pascal Poupart
David R. Cheriton School of Computer Science

UNIVERSITY OF WATERLOO

# Quick Recap: Linear Models

## Linear Regression

$y = w^T \bar{x}$

## Linear Classification

Logistic regression

Binary: $P(y|x) = \sigma(w^T \bar{x})$

Multi-class: $P(y_k|x) = \frac{e^{w_k^T \bar{x}}}{\sum_j e^{w_j^T \bar{x}}}$

Perceptron

threshold: $y = sign(w^T \bar{x})$

Sigmoid: $P(y|x) = \sigma(w^T \bar{x})$

# Quick Recap: Non-linear Models

Non-linear classification

Logistic regression

$P(y|x) = \sigma(w^T \phi(x))$

$P(y_k|x) = \frac{e^{w_k^T \phi(x)}}{\sum_j e^{w_j^T \phi(x)}}$

Perceptron

$y = sign(w^T \phi(x))$

$P(y) = \sigma(w^T \phi(x))$

Non-linear regression

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

multi-layer neural nets

# Non-linear Models

- **Convenient modeling assumption:** linearity
- **Extension:** non-linearity can be obtained by mapping $\boldsymbol{x}$ to a non-linear feature space $\phi(\boldsymbol{x})$
- **Limit:** the basis functions $\phi_i(\boldsymbol{x})$ are chosen a priori and are fixed

- **Question:** can we work with unrestricted non-linear models?

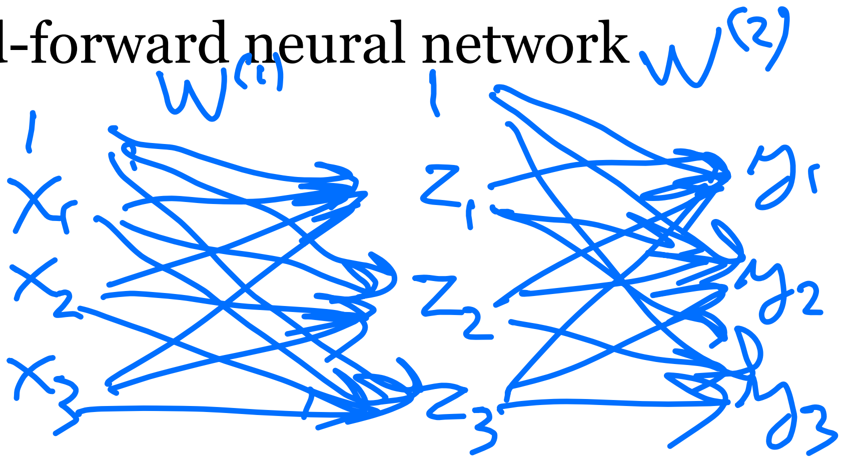
# Flexible Non-Linear Models

- Idea 1: Select basis functions that correspond to the training data and retain only a subset of them (e.g., **Support Vector Machines**)

- Idea 2: Learn non-linear basis functions (e.g., **Multi-layer Neural Networks**)

# Two-Layer Architecture

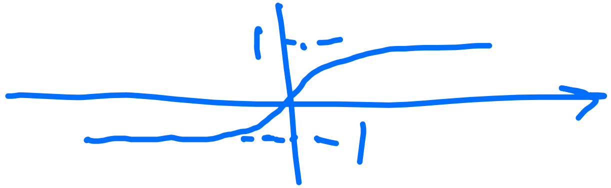
- Feed-forward neural network




- Hidden units: $z_j = h_1(\boldsymbol{w}_j^{(1)} \bar{\boldsymbol{x}})$
- Output units: $y_k = h_2(\boldsymbol{w}_k^{(2)} \bar{\boldsymbol{z}})$
- Overall: $y_k = h_2\left(\sum_j w_{kj}^{(2)} h_1\left(\sum_i w_{ji}^{(1)} x_i\right)\right)$

# Common activation functions $h$

- Threshold: $h(a) = \begin{cases} 1 & a \geq 0 \\ -1 & a < 0 \end{cases}$
- Sigmoid: $h(a) = \sigma(a) = \frac{1}{1+e^{-a}}$
- Gaussian: $h(a) = e^{-\frac{1}{2}\left(\frac{a-\mu}{\sigma}\right)^2}$
- Tanh: $h(a) = \tanh(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$
- Identity: $h(a) = a$

# Adaptive non-linear basis functions

- Non-linear regression
  - $h_1$: non-linear function and $h_2$: identity

$$y_k = \sum_j w_{kj}^{(2)} \sigma\Big(\sum_i w_{ji}^{(1)} x_i\Big)$$

linear combination — non-linear basis functions

- Non-linear classification
  - $h_1$: non-linear function and $h_2$: sigmoid

$$P(y_k) = \sigma\Big(\sum_j w_{kj}^{(2)} \sigma\Big(\sum_i w_{ji}^{(1)} x_i\Big)\Big)$$

linear combination — non-linear basis functions

# Weight training

- Parameters: $< W^{(1)}, W^{(2)}, \ldots >$
- Objectives:
  - **Error minimization**
    - **Backpropagation (aka “backprop”)**
  - Maximum likelihood
  - Maximum a posteriori
  - Bayesian learning

# Least squared error

- Error function

$$E(\boldsymbol{W}) = \frac{1}{2}\sum_{n} E_n(\boldsymbol{W})^2 = \frac{1}{2}\sum_{n} \left|\left|f(\boldsymbol{x_n}, \boldsymbol{W}) - y_n\right|\right|_2^2$$

- When $f(\boldsymbol{x}, \boldsymbol{W}) = \underbrace{\sum_j w_{kj}^{(2)}}_{\text{Linear combo}} \underbrace{\sigma\left(\sum_i w_{ji}^{(1)} x_i\right)}_{\text{Non-linear basis functions}}$

Linear combo    Non-linear basis functions

then we are optimizing a linear combination of non-linear basis functions

# Sequential Gradient Descent

- For each example $(\boldsymbol{x}_n, y_n)$ adjust the weights as follows:

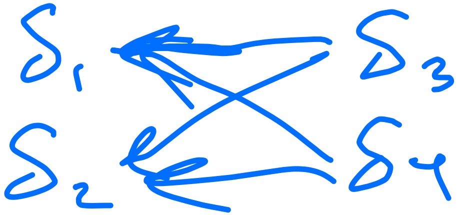
$$w_{ji} \leftarrow w_{ji} - \eta \frac{\partial E_n}{\partial w_{ji}}$$

- How can we compute the gradient efficiently given an arbitrary network structure?
- Answer: **backpropagation algorithm**
- Today: **automatic differentiation**

# Backpropagation Algorithm

- Two phases:
  - Forward phase: compute output $z_j$ of each unit $j$


  - Backward phase: compute delta $\delta_j$ at each unit $j$





UNIVERSITY OF WATERLOO

# Forward phase

- Propagate inputs forward to compute the output of each unit
- Output $z_j$ at unit $j$:

$$z_j = h(a_j) \quad \text{where} \quad a_j = \sum_i w_{ji} z_i$$

# Backward phase

- Use chain rule to recursively compute gradient
  - For each weight $w_{ji}$: $\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$
  - Let $\delta_j \equiv \frac{\partial E_n}{\partial a_j}$ then

$$\delta_j = \begin{cases} h'(a_j)(z_j - y_j) & \text{base case: } j \text{ is an output unit} \\ h'(a_j)\sum_k w_{kj}\delta_k & \text{recursion: } j \text{ is a hidden unit} \end{cases}$$

  - Since $a_j = \sum_i w_{ji} z_i$ then $\frac{\partial a_j}{\partial w_{ji}} = z_i$

# Simple Example

- Consider a network with two layers:
  - Hidden nodes: $h(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$
    - Tip: $tanh'(a) = 1 - (tanh(a))^2$
  - Output node: $h(a) = a$

- Objective: squared error

# Simple Example

- Forward propagation:
  - Hidden units: $a_j = \sum_i w_{ji} x_i$ $z_j = \tanh(a_j)$
  - Output units: $a_k = \sum_j w_{kj} z_j$ $z_k = a_k$
- Backward propagation:
  - Output units: $\delta_k = z_k - y_k$
  - Hidden units: $\delta_j = (1 - z_j^2) \sum_k w_{kj} \delta_k$
- Gradients:
  - Hidden layers: $\frac{\partial E_n}{\partial w_{ji}} = \delta_j x_i = (1 - z_j^2) \sum_k w_{kj} \delta_k x_i$
  - Output layer: $\frac{\partial E_n}{\partial w_{kj}} = \delta_k z_j = (z_k - y_k) z_j$

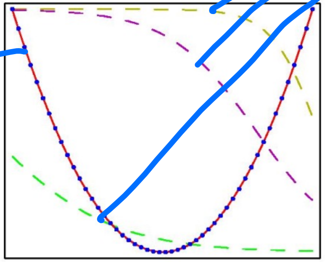
# Non-linear regression examples

- Two-layer network:
  - 3 tanh hidden units and 1 identity output unit

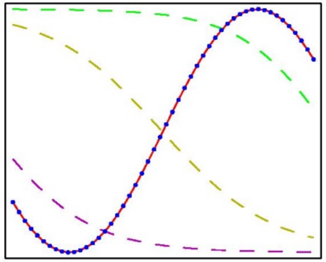
basis functions (tanh hidden units)

estimated function

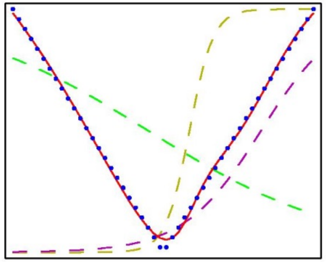
$y = x^2$



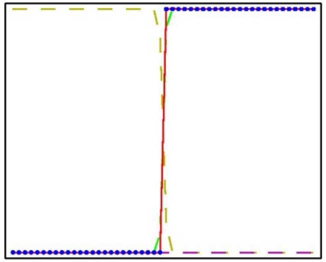
$y = \sin x$



$y = |x|$



$y = \int_{-\infty}^{x} \delta(t)dt$





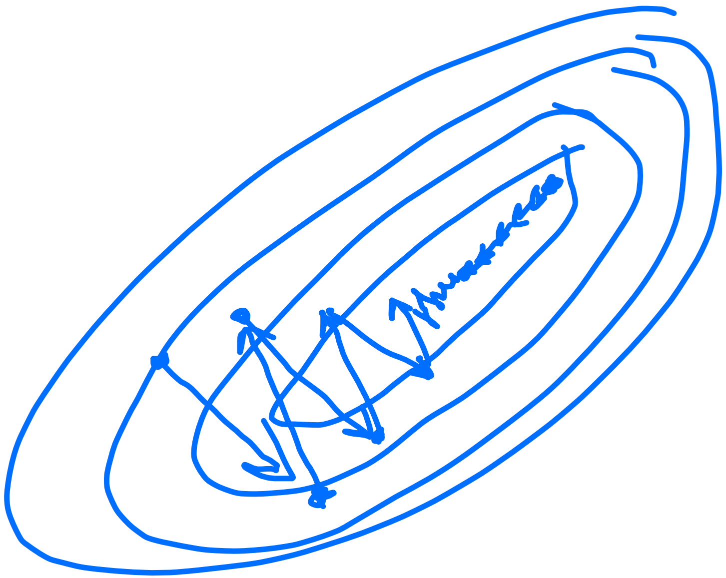
UNIVERSITY OF WATERLOO

# Analysis

- Efficiency:
  - Fast gradient computation: linear in number of weights
- Convergence:
  - Slow convergence (linear rate)
  - May get trapped in local optima
- Prone to overfitting
  - Solutions: early stopping, regularization (add $||w||_2^2$ penalty term to objective), dropout

# Slow convergence

- Gradient direction is not always ideal
- Picture



# Adaptive Gradients

- Idea: adjust the learning rate of each dimension separately
- **AdaGrad:**

$$r_t \leftarrow r_{t-1} + \left(\frac{\partial E_n}{\partial w_{ji}}\right)^2 \quad \text{(sum of squares of partial derivative)}$$

$$w_{ji} \leftarrow w_{ji} - \frac{\eta}{\sqrt{r_t}} \frac{\partial E_n}{\partial w_{ji}} \quad \text{(update rule)}$$

- Problem: learning rate $\frac{\eta}{\sqrt{r_t}}$ decays too quickly

# RMSprop

- Idea: divide by root mean square (RMS) (instead of root of the sum) of partial derivatives
- **RMSprop:**

$$r_t \leftarrow \alpha r_{t-1} + (1-\alpha)\left(\frac{\partial E_n}{\partial w_{ji}}\right)^2 \quad \text{(where } 0 \leq \alpha \leq 1\text{)}$$

$$w_{ji} \leftarrow w_{ji} - \frac{\eta}{\sqrt{r_t}} \frac{\partial E_n}{\partial w_{ji}} \quad \text{(update rule)}$$

- Problem: gradient lacks momentum

# Adaptive moment estimation

- Idea: replace gradient by its moving average to induce momentum
- **Adam**:

$$r_t \leftarrow \alpha r_{t-1} + (1-\alpha)\left(\frac{\partial E_n}{\partial w_{ji}}\right)^2 \quad \text{(where } 0 \le \alpha \le 1\text{)}$$

$$s_t \leftarrow \beta s_{t-1} + (1-\beta)\left(\frac{\partial E_n}{\partial w_{ji}}\right) \quad \text{(where } 0 \le \beta \le 1\text{)}$$

$$w_{ji} \leftarrow w_{ji} - \frac{\eta}{\sqrt{r_t}} s_t \quad \text{(update rule)}$$

# Empirical Comparison

- From Kingma & Ba (ICLR-2015):