COM4511 Speech Technology: Neural Networks

Anton Ragni

February 17, 2020

SPandH

Why Neural Networks?

- Previous lectures discussed:
  - Support Vector Machines
    - in the simplest, binary, case yields a theoretically optimal decision boundary
    - have been extended to linearly inseparable data, multiple classes and structured data
    - BUT manually designed features (kernel)
  - Gaussian Mixture Models
    - for large enough number of Gaussians can approximate any density
    - widely studied across many fields and areas (including speech technology!)
    - BUT manually designed features, inefficient use of parameters
- Other modules discussed:
  - Gaussian Processes
    - no more point estimates!
    - BUT same problems as with SVMs
- This lecture introduces neural networks
  - optimal, task-specific, feature engineering

Function Composition

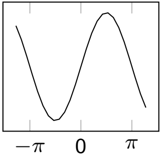
- Layer simple functions to yield complex functions
  - composition of two functions

$$y = f \circ g(x) = f(g(x))$$

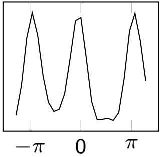
  - generalisation to L functions

$$y = f^{(L)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(x) = f^{(L)}(\cdots f^{(2)}(f^{(1)}(x))\cdots)$$

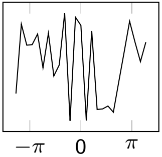
  - can be generalised to graph structures (later)
- Example of composition
  - $y = \sin(4\sin(3\sin(x) + 2) + 5)$
  - L?, $f^{(1)}, \ldots, f^{(L)}$?
  - $\min(y)$? $\max(y)$?
- Neural networks are examples of function composition



(a) $\sin(x)$



(b) $\sin(3\sin(x) + 2)$



(c) $\sin(4\sin(3\sin(x) + 2) + 5)$

Anton Ragni

Example of Functions

- Linear

$$g(x) = ax + b$$

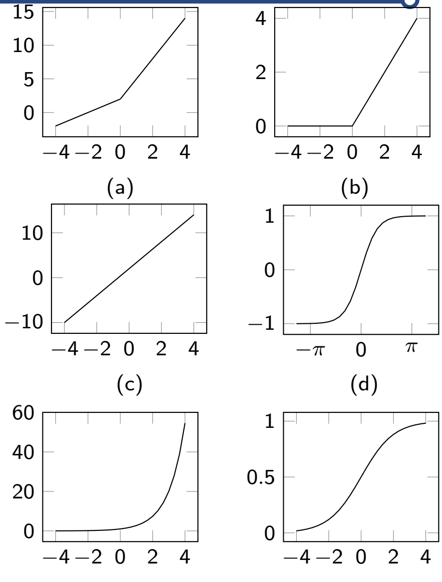
- Piece-wise linear

$$g(x) = \max(0, x)$$

- Non-linear

$$g(x) = \frac{1}{1 + e^{-x}} = \sigma(x), \quad g(x) = \tanh(x)$$

- Example
  - (a),...,(f)?



(a) (b) (c) (d) (e) (f)

Anton Ragni

Multivariate Extensions: Vector Functions

- Linear transformation

$$\boldsymbol{y}_{k\times 1} = \boldsymbol{A}_{k\times n}\boldsymbol{x}_{n\times 1} + \boldsymbol{b}_{k\times 1}$$

  - possible forms for $\boldsymbol{A}$: full, block, Toeplitz, diagonal (impact?)

- Element-wise

$$y_i = \phi^{(i)}(x_i)$$

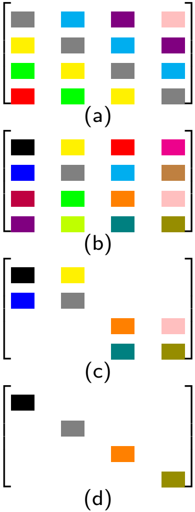
  - $\phi^{(1)}, \ldots, \phi^{(n)}$ are typically identical non-linear functions
  - what is the form of $\phi'(\boldsymbol{x})$?

- Softmax

$$y_i = \frac{\exp(x_i)}{\sum_{i=1}^{n}\exp(x_i)} \quad \text{for} \quad i = 1, \ldots, n$$

  - yields probability mass function $P(y = i|\boldsymbol{x})$ (why?)

Matrix shapes:



Anton Ragni

Multivariate Extensions: Matrix Functions

- Convolution
  - one-dimensional with filter $\boldsymbol{h} = \begin{bmatrix} h_1 & \dots & h_m \end{bmatrix}$

$$\boldsymbol{y} = \boldsymbol{h} * \boldsymbol{x}, \quad \text{where} \quad y_i = \sum_{j=1}^{m} h_j x_{i-j+1} \quad \text{and} \quad h_{j>m} = 0$$

    - need to decide how to handle $x_{i-j<0}$: pad $\boldsymbol{x}$, do not compute y_i
  - two-dimensional with filter $\boldsymbol{H} = \{h_{i,j}\}_{i=1\dots m, j=1\dots m}$

$$\boldsymbol{Y} = \boldsymbol{H} * \boldsymbol{X}, \quad \text{where} \quad y_{i,j} = \sum_{k=1}^{m} \sum_{l=1}^{m} h_{k,l} x_{i-k+1,j-l+1}$$

- Example
  - $\boldsymbol{H} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $\boldsymbol{X} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 3 \\ 4 & 2 & 1 \end{bmatrix}$, $\boldsymbol{Y}$?
  - matrix form $\mathbf{Y} = \mathbf{H} * \mathbf{X}$ is $\boldsymbol{f}(\boldsymbol{Y}) = \mathbf{C}(\mathbf{H})\boldsymbol{g}(\mathbf{X})$, what are $\boldsymbol{f}$, $\boldsymbol{g}$, $\mathbf{C}(\mathbf{H})$?

Feed-Forward Neural Network

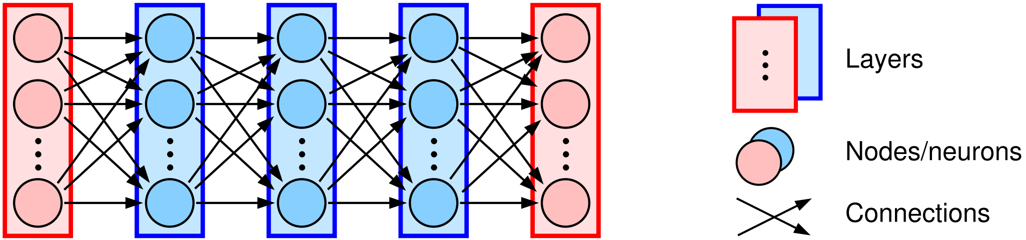


Diagram for $\boldsymbol{y} = \phi^{(4)}(\boldsymbol{A}^{(4)}\phi^{(3)}(\boldsymbol{A}^{(3)}\phi^{(2)}(\boldsymbol{A}^{(2)}\phi^{(1)}(\boldsymbol{A}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) + \boldsymbol{b}^{(2)}) + \boldsymbol{b}^{(3)}) + \boldsymbol{b}^{(4)})$

- Combines linear transformations with simple element-wise non-linearities
  - element-wise (non-)linearities are also called activations

Anton Ragni

Convolutional Neural Network

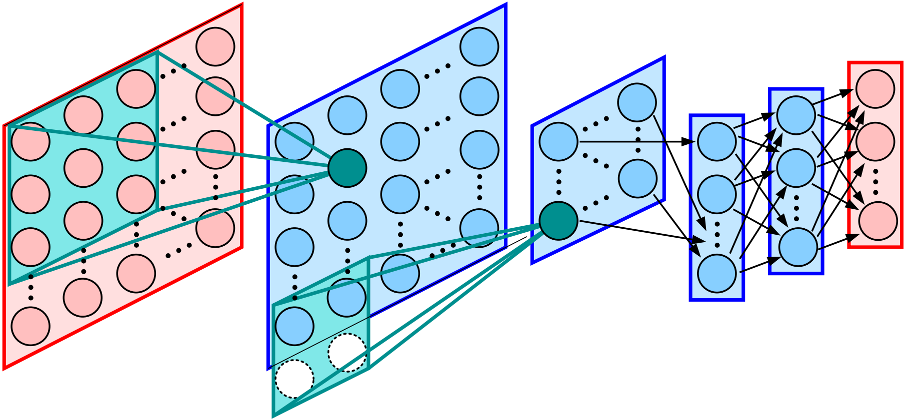


Diagram for $\boldsymbol{y} = \phi^{(4)}(\boldsymbol{A}^{(4)}\phi^{(3)}(\boldsymbol{A}^{(3)}\mathrm{vec}(\phi^{(2)}(\boldsymbol{H}^{(2)} *_2 \phi^{(1)}(\boldsymbol{H}^{(1)} * \boldsymbol{X}))) + \boldsymbol{b}^{(3)}) + \boldsymbol{b}^4)$

- Use convolutional layers to extract "optimal" input features
  - popular in image and speech recognition tasks

Properties of Composition

- Associativity

$$f \circ (g \circ h)(x) = (f \circ g) \circ h(x)$$

- Inversion

$$(f \circ g)^{-1}(x) = g^{-1} \circ f^{-1}(x)$$

  - which conditions f and g need to satisfy?
- Derivatives (chain rule)
  - two functions

$$(f \circ g)'(x) = (f' \circ g)(x) \cdot g'(x)$$

  - L functions

$$(f^{(L)} \circ \cdots \circ f^{(2)} \circ f^{(1)})'(x) = \prod_{l=L}^{1} (f^{(l)'} \circ \cdots \circ f^{(2)} \circ \cdots \circ f^{(1)})(x)$$

Anton Ragni

Automatic Differentiation

- Recall chain rule for compositions

$$(f \circ g \circ h)'(x) = (f' \circ g \circ h)(x) \cdot (g' \circ h)(x) \cdot h'(x)$$

- Options
  - forward propagation

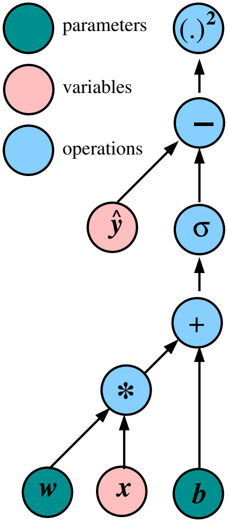
$$h'(x)|_{x \leftarrow x_0} \rightarrow (g' \circ h)(x) \rightarrow (f' \circ g \circ h)(x)$$

  - backward propagation

$$(f' \circ g \circ h)(x)|_{\hat{y} \leftarrow y_0} \rightarrow (g' \circ h)(x) \rightarrow h'(x)$$

- Can be extended to graphs (TensorFlow, PyTorch)
  - example: compute

$$\nabla \mathcal{L}(\hat{y}, x; \boldsymbol{\theta}) = \begin{bmatrix} \frac{d\mathcal{L}(\hat{y},x;\boldsymbol{\theta})}{dw} & \frac{d\mathcal{L}(\hat{y},x;\boldsymbol{\theta})}{db} \end{bmatrix}^{\mathsf{T}}?$$



Graph for $\mathcal{L}(\hat{y}, x; \boldsymbol{\theta}) = (\hat{y} - \sigma(wx + b))^2$

Objective Functions

Specify criterion for estimating parameters:

- squared error
  - scalar data

$$\mathcal{L}(\hat{y}, x; \boldsymbol{\theta}) = (\hat{y} - f(x; \boldsymbol{\theta}))^2$$

  - multi-dimensional data

$$\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{x}; \boldsymbol{\theta}) = (\hat{\boldsymbol{y}} - \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}))^{\mathsf{T}}(\hat{\boldsymbol{y}} - \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}))$$

- cross-entropy

$$\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{x}; \boldsymbol{\theta}) = -\sum_{i=1}^{n} \hat{y}_i \log(P(y = i|\boldsymbol{x}; \boldsymbol{\theta}))$$

  - typically $\hat{y}_i = 0$ for all classes but one
- Many other functions possible

Anton Ragni

Parameter Estimation

- Improve robustness by learning from multiple examples

$$\mathcal{L}(\mathcal{D};\boldsymbol{\theta}) = \frac{1}{|\mathcal{D}|}\sum_{(\hat{y},x)\in\mathcal{D}} \mathcal{L}(\hat{y},x;\boldsymbol{\theta})$$

  - issues: computational complexity, numerical stability and ...
- Stochastic approximation

$$\mathcal{L}(\mathcal{D}';\boldsymbol{\theta}) = \frac{1}{|\mathcal{D}'|}\sum_{(\hat{y},x)\in\mathcal{D}'} \mathcal{L}(\hat{y},x;\boldsymbol{\theta})$$

  - possible choices for $\mathcal{D}'$: single sample, M-length sample (mini-batch)
- Key elements:
  - optimisation: find solution as efficiently as possible
  - generalisation: ensure found solution is representative

Anton Ragni

Optimisation

Need to fit (typically highly non-convex) function to data — options

- Gradient methods (first-order)

If $\mathcal{L}(\mathcal{D};\boldsymbol{\theta})$ is differentiable in a neighborhood of $\boldsymbol{\theta}^{(i)}$ then

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \rho\,\nabla\mathcal{L}(\mathcal{D};\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(i)}}$$

for ρ small enough yields $\mathcal{L}(\mathcal{D};\boldsymbol{\theta}^{(i)}) \geq \mathcal{L}(\mathcal{D};\boldsymbol{\theta}^{(i+1)})$

  - gradient descent: pick $\boldsymbol{\theta}^{(0)}$ and ρ, iterate till convergence (local or global optimum?)

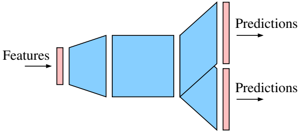
- Hessian methods (second-order)

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \left(\nabla^2\mathcal{L}(\mathcal{D};\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(i)}}\right)^{-1}\nabla\mathcal{L}(\mathcal{D};\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(i)}}$$

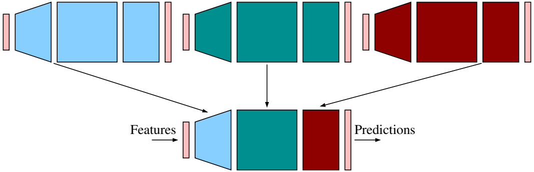
  - non-trivial to compute/manipulate Hessian (why?)

Regularisation

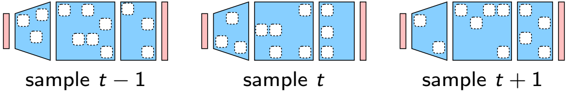
- Improve generalisation through architectural and procedural changes



(a) Multi-task



(b) Pre-training



(c) Dropout

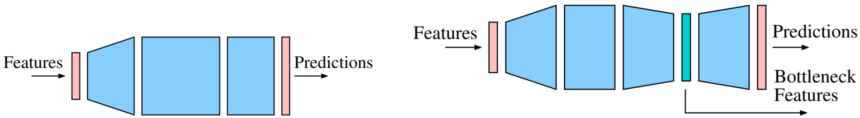
- And/or penalise solutions violating constraints set by regularisation term

$$\mathcal{F}(\mathcal{D};\boldsymbol{\theta}) = \mathcal{L}(\mathcal{D};\boldsymbol{\theta}) + C\mathcal{R}(\boldsymbol{\theta})$$

  - ℓ_p norms, stimulated patterns

Anton Ragni

Applications



(a) Classifier (b) Feature Extractor

- Prediction networks: classify speech into one of n classes using $P(y|\boldsymbol{x};\boldsymbol{\theta})$
  - also can model probability density functions via Bayes' rule (hybrid!)

$$p(\boldsymbol{x}|y;\boldsymbol{\theta}) = P(y|\boldsymbol{x};\boldsymbol{\theta})\frac{p(\boldsymbol{x};\boldsymbol{\theta})}{P(y;\boldsymbol{\theta})}$$

- Feature networks: extract speech features from one of intermediate layers
  - typically use a dedicated, small, bottleneck layer
  - feed extracted features into a different classifier (tandem!)

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{x}^{\mathsf{T}} & \boldsymbol{y}^{(l)^{\mathsf{T}}} \end{bmatrix}^{\mathsf{T}}$$

Anton Ragni

Summary

- Function composition
  - simple, conceptual and practical, framework for learning complex functions
  - mimics many natural process (including neurons)
- Applications
  - feature extraction, regression, classification, density estimation and many more!