Deep Learning for COMP6714 – Part I

September 17, 20

Outline

- ML basics
- Feed forward Network

Problem Definition

The standard *supervised* classification/regression setting:

- Input
  - Labelled data: $\{ \mathbf{x}_{(i)}, y_{(i)} \}_{i \in [n]}$
  - f

  -
    - $|C|$-class classification: $y_{(i)} \in$
    - Regression: $y_{(i)} \in \mathbb{R}$.

- Output: a function/mapping (typically *class*) from dom $\mathbf{x} \to$ dom y such t minimized.
- Assumption:
  - Training and test data are drawn i.i.d. from the same (unknown) distribution (defined over dom $\mathbf{X} \times$ dom $\mathbf{y}$).

Key Concepts

Ultimate goal:

- Generalization error: Errors (of the model) on unseen data

How to approximate it?

-
    -
    -
    - Test data:
- Use the errors on the test data

How to train a model?

- Minimize the loss function on the training data
- (Optionally) also considering some regularization measures.
    - To prevent overfitting

Loss Functions

Used to

- Characterize how bad a prediction is compared with the ground truth.
-

Commonly Used Loss Functions

Loss functions $L(\{\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots, \hat{\mathbf{y}}_n\}, \{\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_n\})$:

Typically, $L = \sum_{i=1}^{n} \ell(\hat{\mathbf{y}}_i, \mathbf{t}_i)$

- Classification:
  -
  - here j^*
  -

  classification problems.

- Regression:
  - MSE (Mean Squared Error): $\ell($

(Traditional) Machine Learning vs. Deep Learning

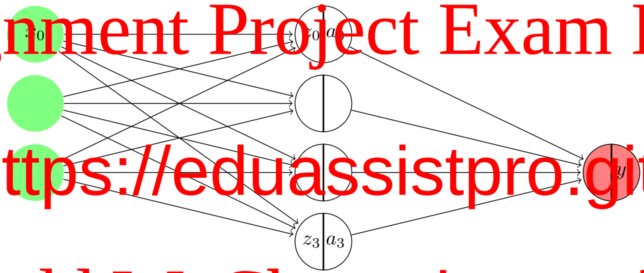
- ML: Features are defined/engineered.
- DL: Features are learned in an *end-to-end* fashion.

Examples

OCR

- ML: define *invariant* features. E.g., number of circles, number of (almost) horizontal strokes, . . .
  -
-
  - The final classifier is in fact a simple softma linear classifier).

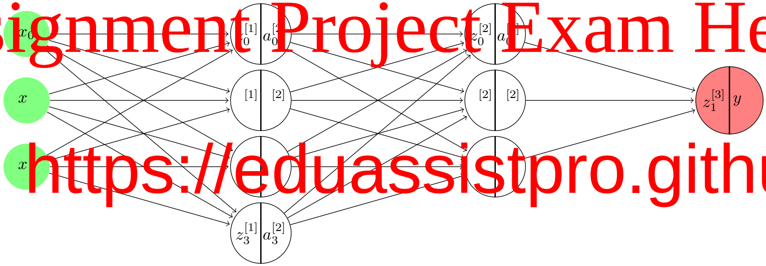
Feed Forward Network / Multilayer Perceptron (MLP)



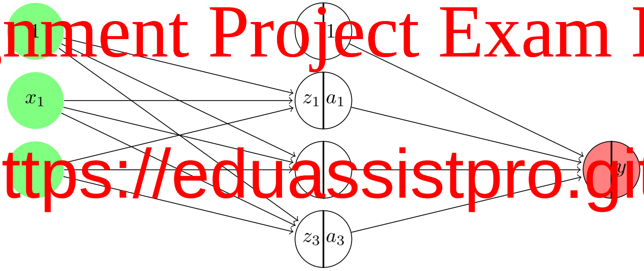
Concepts:

- Neurons
- Input / hidden / output layers
- Activitation function

NN with Multiple Hidden Layers



NN with One Hidden Layer and Biases



- $\mathbf{a}_n = \sigma_n(\underbrace{\mathbf{W}_n \mathbf{a}_{n-1} + \mathbf{b}_n}_{\mathbf{z}_n})$
- $\mathbf{y} = \mathbf{a}_n$ and $\mathbf{x} = \mathbf{a}_1$
- σ_ns are typically non-linear functions, applied element-wise to the input vector.

Non-linearalities /1

- sigmoid (aka. logistic): $\sigma(z) = \frac{1}{1+\exp(-z)}$
  - Special case of $\mathrm{Softmax}([z, 0])$, where $\mathrm{Softmax}([z_1, z_2, \ldots, z_m]) = [\frac{\exp(z_1)}{}, \frac{\exp(z_2)}{}, \ldots, \frac{\exp(z_m)}{}]$
  -
    -
    -
  - $\sigma\ (z) = \sigma(z)(1 - \sigma(z))$

Logit and Logistic Functions

Recall that $\mathrm{logit}(p) = \log \frac{p}{1-p}$. It follows th

$$\mathrm{logit}(p) = z \iff \mathrm{logistic}(z) = p$$

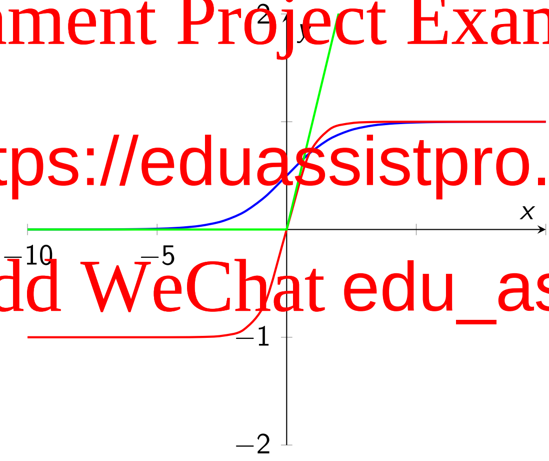
Non-linearalities /2

- tanh: $\tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$
  - It is a rescaled sigmoid: $\tanh(z) = 2\sigma(2z) - 1$
  - Squashing $\mathbb{R}$ to $[\ 1, 1]$, and differentiable every where.
  -

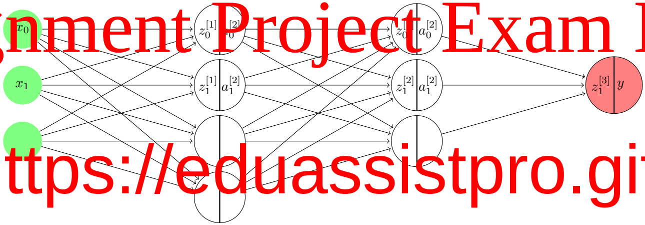
-
  - vanishing problems. Hence, popular for DL models.
  - There exist many slight variants.

  - $\text{ReLU}'(z) = \begin{cases} z & \text{, if } z \geq 0 \\ 0 & \text{, otherwise.} \end{cases}$

Illustration of Non-linearalities



Forward Computation



Notations: $w_{i,j}^{[l]}$: the weight on the edge from i-th layer $l-1$ to the j-th neuron in layer l

Things to ponder:

- Which weights influence $z_1^{[2]}$?
- What's the impact to y if x_1 increases by a tiny amount ϵ?

Function Approximation

- ANN can well approximate any function (despite potentially huge size requirement)
- Learning: find $\boldsymbol{\theta} = \arg\min \quad \ell(\mathbf{y} , \mathbf{t})$, where $\mathbf{y} = f(\mathbf{x}_i; \boldsymbol{\theta})$

Function Minimization

- Typically, NP-hard to minimize a general function.
- However, we can find a good-quality local minima instead of the global minimum.
- [illegible]
  1. [illegible]
  2. [illegible]

  3. Based on this approximation, find the be [illegible] $\mathbf{x}$ within the tiny neighborhood. Then, [illegible]

- Extending Taylor series to functions with v [illegible]

$$f(x_0 + \epsilon) \approx f(x_0) + f'(x_0)\epsilon$$

$$f(\mathbf{x}_0 + \boldsymbol{\epsilon}) \approx f(\mathbf{x}_0) + \langle \boldsymbol{\nabla} f(\mathbf{x}_0), \boldsymbol{\epsilon} \rangle$$

Which $\boldsymbol{\epsilon}$ can minimize $f(\mathbf{x}_0 + \boldsymbol{\epsilon})$ subject to $\|\boldsymbol{\epsilon}\| \leq$ some small constant?

Illustration of GD

Variants of GD

- Gradient descent (GD):

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \alpha \cdot \nabla_L(\boldsymbol{\theta}^{(t)})$$

-
    -
    -
- Mini batch SGD:
    - $\nabla_L(\boldsymbol{\theta})$ is evalauted only on a mini-batch o
    - Tuning minibatch sizes may achieve g
- SGD with momentum:
    - Think of the gradient as the velocity, and $\boldsymbol{\theta}$ as the position. Then this method keeps a portion of the last velocity value together with new gradient.
    - Helps to get over some difficult regions quickly (e.g., avoid too much oscillation).

Derivative

Let $y(x, a) = \sin(a \cdot x + 3\exp(x))$. Compute $\frac{\partial y}{\partial x}$.

- $\frac{\partial y}{\partial x} =$

Rewrite y in a verbose manner:

-
-
- $z_2 = a \cdot x$
- $z_3 = 3\exp(x)$

Then:

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial x} \qquad \frac{\partial z_1}{\partial x} = \frac{\partial z_2}{\partial x} + 3\frac{\partial z_3}{\partial x}$$

Rules

Important rules about (partial) derivatives (useful for NN):

- Chain rule: $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z_1}\frac{\partial z_1}{\partial z_2}\cdots\frac{\partial z_k}{\partial x}$
- $\underline{\partial(z_1+z_2)}$ $\underline{\partial z_1}$ $\underline{\partial z_2}$

Thes

Note

- We require that $\frac{}{\partial \mathbf{x}}$ has the same shape as $\mathbf{x}$.
- We can use this as a cue to work out which term ne transposition.

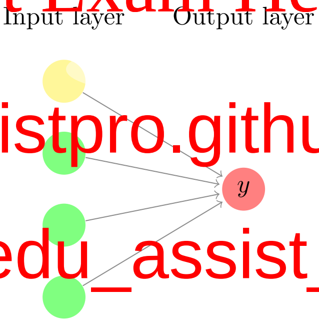
Computational Graph

$$y(x, a) = \sin(a \cdot x + 3\exp(x))$$

Baby Network

Model:

- For single **x** $\mathbb{R}^d$:
  y
- F
- **n**



Shapes:

- y is a *scalar*
- **x** is a row vector, $\mathbb{R}^{1\times d}$ ($d = 3$ here)
- **W** is a matrix, $\mathbb{R}^{d\times 1}$
- b (plot as x_0) is a *scalar*

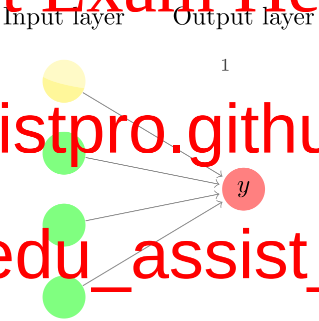
Simplifying the Bias Terms

Model:

- Extend $\mathbf{x}$ to $\mathbb{R}^{d+1}$ and let x_0 be t
- y
- i.e

Shapes:

- y is a scalar
- $\mathbf{x}$ is a row vector, $\mathbb{R}^{1\times(d+1)}$ ($d = 3$ here)
- $\mathbf{W}$ is a matrix, $\mathbb{R}^{(d+1)\times 1}$



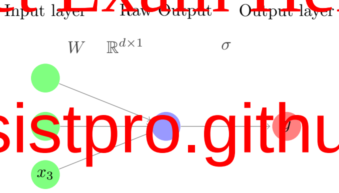
Exercise:

- $\frac{\partial y}{\partial W_{i1}} =$ $\frac{\partial y}{\partial \mathbf{W}} =$

Add the Non-linear Transformation

Model:

- For simplicity, ignore the bi
  th
- y
  z
- Let σ be the sigmoid function, then $\sigma'(\cdot) =$



Shapes:

Exercise:

- $\frac{\partial y}{\partial \mathbf{W}} =$

Add the Loss Function

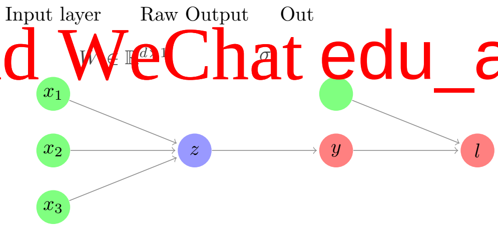
Model:

- $l = \ell(\sigma(\mathbf{Wx}), t)$

Exercise:

- $\frac{\partial l}{\partial \mathbf{W}} = \frac{\partial l}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial \mathbf{W}} =$

- $\ell($



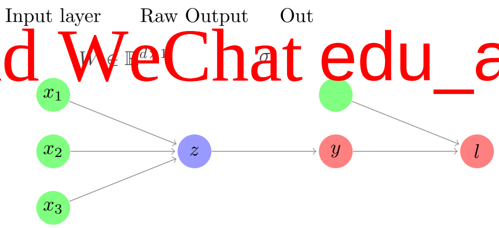
Vectorized Version

Model:

- $l = \ell(\sigma(\mathbf{WX}), \mathbf{t})$
- $\ell($

Exercise:

- $\frac{\partial l}{\partial \mathbf{W}} = \frac{\partial l}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial \mathbf{W}} =$



Computational Graph

Model:

- $l = \ell(\sigma(\mathbf{WX}), \mathbf{t})$
- $\ell($

Exercise:

- $\frac{\partial l}{\partial \mathbf{W}} = \frac{\partial l}{\partial y} \frac{\partial y}{\partial z} \frac{\partial z}{\partial \mathbf{W}} =$

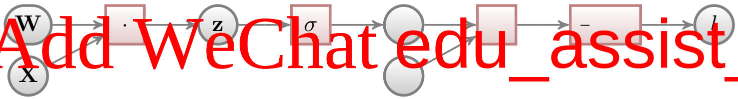


Figure: NN2

References

TBA