Download the jupyter notebook from ILIAS
or
run the notebook in colab:
https://colab.research.google.com/drive/1eWA7jI4R7JMZFLzRMAXI5ckCKZyBKSRQ
(https://tinyurl.com/yx2xnook)

ATML Tutorial 03

Advanced Topics in Machine Learning
03.03.2020
Adam Bielski

Information

- First assignment now on ILIAS (due date: 24/03/2020)
- Bring your laptops next week

Last week

- Basic PyTorch (**tensors**, **CPU/GPU** computations, **gradients** with torch.autograd)
- Feeding data with PyTorch **Dataset** objects (__getitem__, __len__)
- Iterating datasets with **DataLoaders**
- Creating **models with nn.Module** subclasses (forward method) and linear layers (nn.Linear)
- Running the models

Today

- training neural networks in PyTorch
- debugging and analyzing overfitting/underfitting

Neural networks

- Input (features): **x** — *pixels of cat/dogs images*
- Output (labels/target values): **y** — *cat/dog label*
- We want to find a function ***f*** that will approximate ***f(x) = y*** and can be run on new data when ***y*** is unknown

Neural networks approximate the function ***f*** by composition of functions

$$f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$$

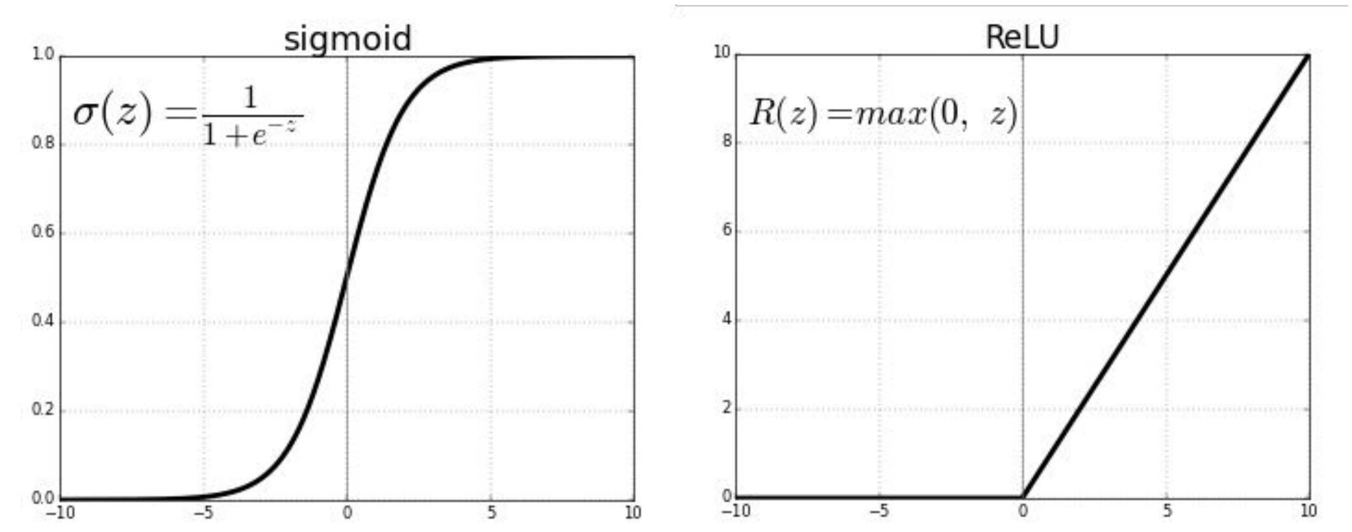
Multilayer Perceptron (MLP)

MLP: $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$

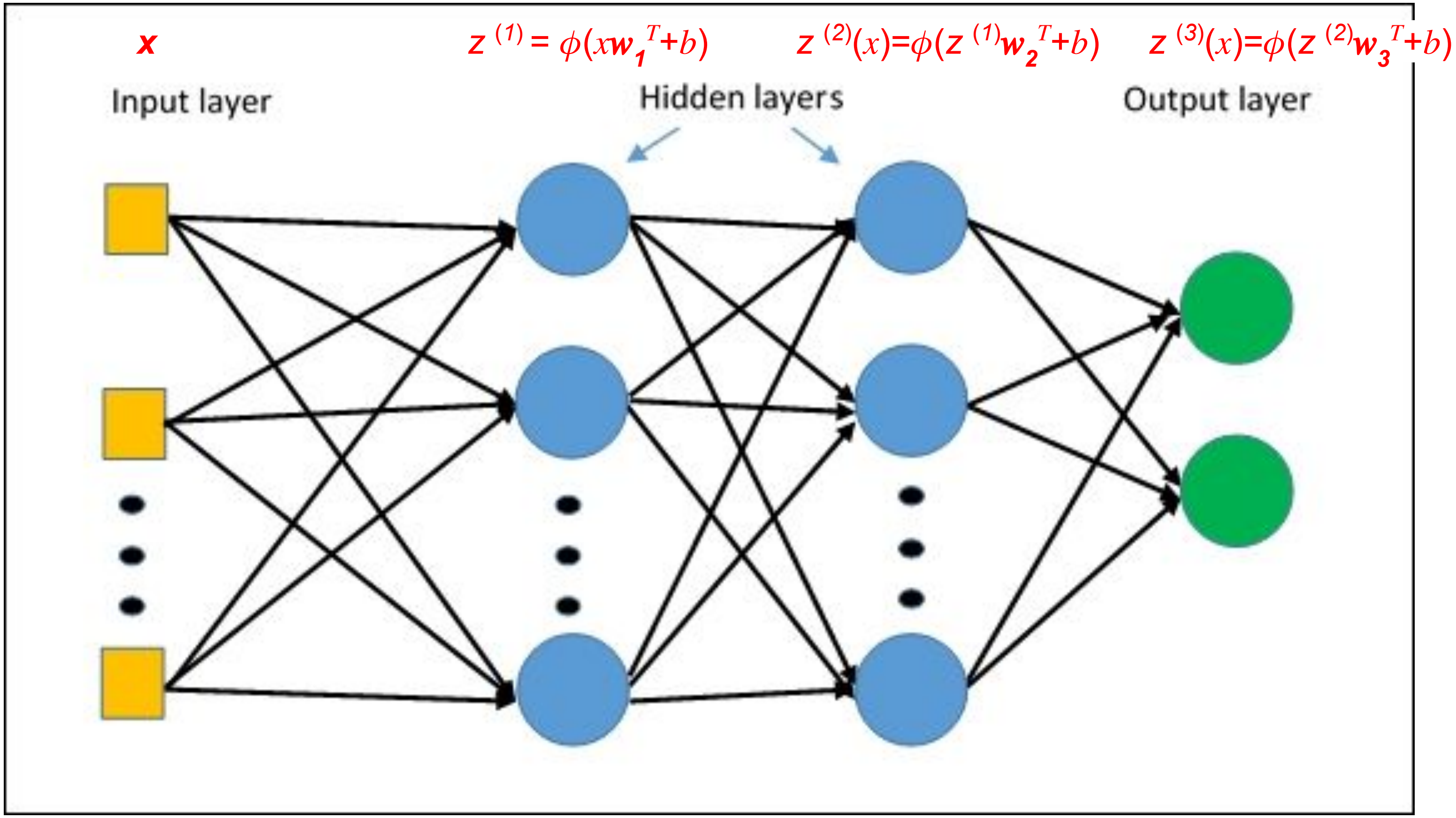
A class of neural networks, where $f^{(i)}(x) = \phi(x\boldsymbol{w}^T + b)$

$\boldsymbol{w}, b$ - parameters we we want to optimize

ϕ - non-linear function



Multilayer Perceptron (MLP)



Source: https://www.oreilly.com/library/view/getting-started-with/9781786468574/ch04s04.html

Loss functions - how different is prediction from target values?

We will **optimize** parameters of a neural network to **minimize** a loss function.

Problem	Output activation	Loss function
Regression y = real value	**Linear / no activation**	**Mean square error** **L = mean([y - y']^2)**
Regression (within range) y scaled to y =[-1; 1]	**Tanh** [-1; 1] / **Sigmoid** [0; 1]	**Mean square error** **L = mean([y - y']^2)**
Binary classification y = 0 or y = 1	**Sigmoid** 1/(1+exp(-x)) Probability of class y=1	**Binary cross-entropy** **L = -y*log(y') - (1-y)*log(1-y')**
Multi-class classification y=0..n	**Softmax** y'=exp(x_i)/sum(exp_j) Vector of probabilities for each class	**Cross-entropy** **L = -log(y'[y])** (maximize probability of true class)

Loss functions - PyTorch

PyTorch documentation - https://pytorch.org/docs/stable/nn.html#loss-functions

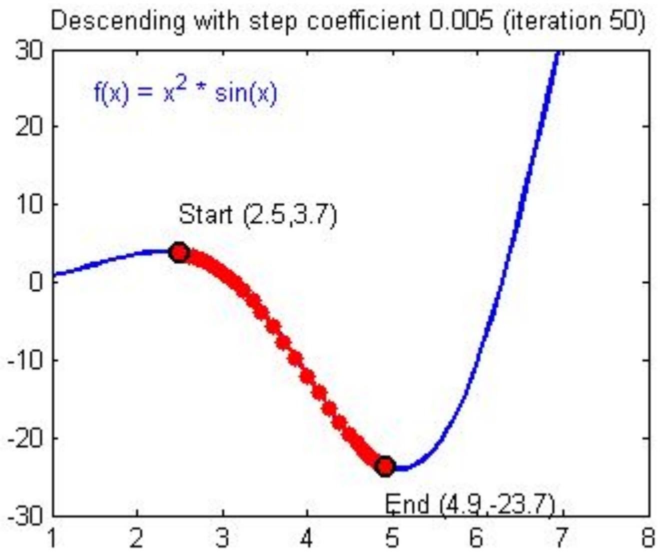
Problem	Output activation (in model)	Loss function
Regression y = real valued vector	**nn.Linear(h_dim, y_dim)**	**nn.MSELoss()**
Regression (within range) y scaled to y =[-1; 1]	**nn.Linear(h_dim, y_dim), nn.Tanh()**	**nn.MSELoss()**
Binary classification y = 0 or y = 1	**nn.Linear(h_dim, 1), nn.Sigmoid()**	**nn.BCELoss()**
	nn.Linear(h_dim, 1)	**nn.BCEWithLogitsLoss()** // combines Sigmoid and NLLLoss
Multi-class classification y=0..n	**nn.Linear(h_dim, n), nn.LogSoftmax()**	**nn.NLLLoss()**
	nn.Linear(h_dim, n)	**nn.CrossEntropyLoss()** // combines LogSoftmax and NLLLoss

Training neural networks (high-level description)
Stochastic Gradient Descent

Input: feature vectors ***x={x_i}***, targets ***y={y_i}***,
neural network model f(x; θ) with randomly initialized parameters θ
Output: optimized model parameters θ

for number of training iterations **do**
sample minibatch of m samples {(x_i, y_i)}
compute **predicted** y'=f(x; θ)
compute **loss** L=*loss_function(y', y)*
compute **gradients** $\nabla\theta$ L // backpropagation algorithm
update parameters θ <- θ - alpha*∇_θ L // update parameters to minimize loss

Training neural networks (high-level description)
Stochastic Gradient Descent



Source: https://nikcheerla.github.io/deeplearningschool/2017/09/08/ch2.1-linear-regression-sgd/

Let's train some networks!