11-785 – Deep Learning

Key things to understand.

A pedagogical tool to track the key aspects of the main lectures that the instructor emphasises, and as a checklist of things you have judged are important.

The recitation lectures focus on the development of skills involved in deep learning, and these are also included.

Some preliminaries

Recitation lecture 0A - Fundamentals of Python.

Refresher on Python. Brief intro on skills which are already familiar, such as: importing modules, loading data and data structures, 2D/3D array slicing, classes, debugging using pdb.

Recitation lecture 0B - Fundamentals of Numpy.

Refresher on Numpy. The key message is to keep code vectorised and implemented in Numpy. and to avoid loops where possible. Specific Numpy skills covered are: vector arrays, matrix arrays, vectorised functions, tensor arrays, padding tensor arrays. These are to assist with the homework assignments.

Recitation lecture 0C - Fundamentals of Jupyter Notebook.

- Provides information on AWS EC2 Instances and Sagemaker Instances – commercial cloud compute resources that will be required for the homework assignments.
- Brief discussion on costs/efficiency of available packages of AWS Instance types.
- Instruction on setting up Sagemaker Instances.
- IPython commands and Jupyter notebooks shortcuts.

Recitation lecture 0D/0E -Setting up AWS Deep Learning AMI

Detailed, comprehensive step-by-step tutorial covering:

- Configuration of AWS EC2 Instances, for part 2 of homework assignments which require GPU computing power for training.
- Connecting to, using, and managing AWS EC2 Instances.
- Use of Jupyter notebooks with AWS EC2 Instances.
- Use of a terminal multiplexer (TMUX), for training models without being connected to the Instance (extremely useful for training models overnight).

Week 1

Lecture 0 - Logistics

Lecture 1 – Introduction

- Understand what neural networks are
- Understand their neurophysiological basis.
- Understand the philosophical and psychological context of neural networks – associationism, proto-connectionism, connectionism.
- Understand recent historical developments – McCulloch-Pitts, Hebbian learning, Rosenblatt perceptrons, Minsky-Papert and XOR controversy.

- Understand the various capabilities of multilayer perceptrons (MLPs) as connectionist computational models, Boolean functions, Boolean machines.

Lecture 2 – Neural networks as universal approximators

- Understand that MLPs are networks of perceptrons.
- Understand the definitions of layer and depth of MLPs.
- Understand MLP capabilities in terms of universal Boolean functions, universal classifiers over real inputs, universal approximators, optimal depth-width.
- MLPs as universal Boolean functions:
    - Logic gates and perceptrons.
    - Truth tables, disjunctive normal form, Boolean functions.
    - Reduced DNF and Karnaugh maps.
    - Understand how these tools yield heuristic arguments regarding depth and width in this context.
    - Be acquainted with relevant research findings in this area.
    - Understand caveats and nuances of these findings.
- MLPs as universal classifiers over real inputs:
    - Understand the heuristic examples regarding the approximation of complex decision boundaries.
- MLPs as universal approximators.
- Optimal depth and width of MLPs:
    - Understand that there are requisite depth/width constraints on expressive power of MLPs.
    - Understand that the above feature as the "sufficiency of architecture".
    - Understand the interpretation of information propagation, loss.
    - Understand the role of appropriate activation functions in the depth-width tradeoff.
    - Be acquainted with various definitions of capacity/expressive power.

- *Make sure you understand the heuristic examples here, lots covered quickly.*

Recitation lecture 1 – Your first deep learning code

Week 2

Lecture 3 – Training the network (part 1)

- Understand feedforward architecture.
- Understand that learning a neural network corresponds to a statistical estimation problem (of weights and biases) to represent an unknown function.
- Understand that tuning parameters corresponds to minimisation of an integral over divergences.
- Understand the role of training data as noisy sampling from an unknown function we wish to represent.
- Understand the perceptron learning algorithm, and the Novikov's perceptron convergence theorem.
- Appreciate the theoretical difficulties of training a neural network using a series of perceptron learning algorithms as one of an NP-hard combinatorial optimisation problem.
- Understand the role of differentiable activation functions.

- Understand the relation between a perceptron with sigmoid activation and logistic regression.
- Understand the role of expected error, empirical estimates of expected error, empirical risk minimisation.
- Be fluent in scalar function optimisation using elementary calculus.
- Be fluent with tools for evaluating derivatives of multivariate scalar functions, such as gradients, Hessians, and their properties.
- Be fluent in unconstrained optimisation of a multivariate scalar function.
- Understand the role of non-convexity, numerical iterative methods, and gradient descent.

- *Appreciate the historical significance of ADALINE and MADALINE.*
- *Better understand the formal properties of the gradient in context of vector calculus, directional derivatives, and level sets.*

Recitation lecture 2 – How to compute a derivative

Week 3

Lecture 4 – Backpropagation

- Understand the details of the components of each element of a neural network:
    - Training data as input-output pairs.
    - The loss/error function.
    - Weight and bias parameters.
    - Affine combinations.
    - Activation functions and their derivatives.
    - Vector representation of the neural network.
    - Input representation.
    - Output representations for binary classification, multiclass classification.
- Understand the affinity between the problem setting, output activation function, and error/loss/divergence for regression; and the binary and multiclass classification settings.
- Understand the technique of label-smoothing.
- Understand gradient descent and its pseudocode.
- Understand that derivatives and the chain rule can be viewed as a route of influence from a source to a destination through a topological network with influence diagrams.
- Be fluent with the chain rule for univariate and multivariate functions.
- Be fluent with the forward and backward pass of the backpropagation/reverse-mode automatic differentiation algorithm.

Lecture 4.5 – Backpropagation (part 2)

- Understand a number of issues that have been backgrounded thus far:
- Understand the role of vector activations, and the consequence for derivative calculations.
- Understand that assuming affine combinations does not necessarily hold for other NN architectures.
- Understand the issue of non-differentiable activation functions, and the use of subgradients.
- Be fluent with the use of Jacobian matrices for derivatives of vector functions with respect to vector inputs.
- Understand their differing properties for scalar and vector activations functions.
- Be fluent with the chain rule cast in terms of gradients, Jacobians, and combinations.

- Be fluent with the vector representation of the NN, and the vector representation of forward and backward passes of the backpropagation algorithm.

- *Understand how techniques are adapted to accommodate multiplicative, rather than affine combinations.*

Lecture 5 – Convergence

- Understand the neural network training algorithm pseudocode.
- Understand heuristic comparisons of the performance of backpropagation and the perceptron learning algorithm when additional training instances are added.
- Understand the implications of these arguments for the MLPs.
- Understand how these are framed in terms of bias-variance.
- Understand comparisons of the backpropagation trained classifier and optimal classifiers.
- Understand visually that the loss or error surface can be visualised as a complex multi-dimensional manifold.
- Be aware that visualisation of an error surface for various neural networks is an open question.
- Understand that neural network error surfaces are highly non-convex.
- Understand the distinction between global, local minima and maxima; and saddle points.
- Understand the role of convex optimisation as well-studied and as providing a useful reference point.
- Understand the how convergence of an iterative algorithm is quantified theoretically through a convergence rate, and the properties of differing rates.
- Understand how contour plots in parameter space can help visualise convergence, oscillating convergence, and divergence.
- Understand how the learning rate or step-size parameter in gradient descent relates to convergence for quadratic error surfaces.
- Understand the relation between Taylor expansions, Newton's method, and the relation between the learning rate and 2nd derivative for generic differentiable, convex objectives.
- Understand the nuances and heuristic arguments on convergence for differing characterisations of convex functions, and as dimensions increase.
- Understand the role of rescaling axes to provide a normalised gradient descent rule, using Hessians.
- Understand the computational infeasibility of using Hessians.
- Be aware of research relating to 2nd order methods to approximate Hessians.
- Understand the use of a single learning rate for all parameters, and implications for convergence.
- Understand that heuristic arguments presented may not hold for non-convex error surfaces.
- Understand why decaying learning rates may be used.

- *Review paper on comparisons of backpropagation and the perceptron learning algorithm by Brady et al. (1989).*
- *Weight normalisation - Salimans, Kingma (2016)*

Recitation lecture 3 – Optimising the network

Week 4

Lecture 6 – Convergence in neural networks

- Understand the motivations for the use of resilient propagation (Rprop) and Quickprop algorithms.
- In particular, understand the motivation of the above as updating each component of the parameter separately.
- Understand the mechanics, properties, and pseudocode for Rprop and Quickprop.
- Understand that Rprop and Quickprop can be viewed as making use of approximate information from the gradient and Hessian respectively; and as derivative-inspired algorithms.
- Understand the issues related to making component-wise updates of the parameter, such as those of divergence and convergence in distinct directions.
- Understand the motivation for the use of momentum methods.
- Understand that momentum methods use running averages.
- Understand the mechanics, properties and pseudocode of momentum methods.
- Understand the motivation for the use of Nesterov's accelerated gradient method, together with mechanics, properties and pseudocode.
- Understand that the methods can be viewed as means of improve on the convergence of gradient descent.

- Understand the that there are design choices associated with the size of the training set to be processed before parameter updates are carried out.
- Have a heuristic understanding of the computational issues and algorithmic (mis)behaviour of choosing to do full-batch update of parameters and incremental updates as a means of motivating stochastic gradient descent (SGD).
- Understand the pseudocode for stochastic gradient descent.
- Understand heuristic reasons for why the learning rate should be reduced when implementing SGD.
- Understand formal results regarding sufficient conditions for the convergence of SGD for convex and non-convex loss functions.
- Understand formal convergence properties of batch gradient descent and SGD.
- Understand the statistical relation between the empirical and expected error i.e. through bias and variance of an estimator.
- Understand batch, single-instance SGD, and mini-batch SGD in terms of variance.
- Understand the motivations of, pseudocode for, and formal convergence properties of mini-batch gradient descent.
- *Related reading:*

<u>Lecture 7 – Stochastic gradient decent, overfitting, tricks</u>

- Understand the motivations for recent trend-based methods as building on mini-batch SGD.
- Understand how trend-based methods such as momentum and Nesterov's accelerated gradient can be combined with incremental updates to perform a variance smoothing function.
- Understand the motivations and design principles of more recent trend-based methods to smooth out the variation for mini-batch SGD.
- Understand the properties, mechanics and pseudocode for RMS Prop.
- Understand the properties, mechanics and pseudocode for ADAM (RMS Prop with momentum).

- Understand the properties of desirable loss functions, and that loss surfaces in deep learning will often be very complex.
- Understand the problem of covariate shift as a motivation for batch normalisation.
- Understand the procedure of batch normalisation.
- Understand in detail how batch normalisation affects backpropagation calculations.
- Understand the nuances of dealing with high-dimensional data in context of the number of training instances required to represent a function; and related issues such as over-fitting.
- Understand the motivations for the introduction of smoothing constraints.
- Understand, at the level of the individual unit in a simple neural network, how overfitting happens.
- Understand that L2 and L1 regularisation, or weight decay can be viewed as imposing smoothness constraints.
- Be aware of results that deeper neural networks may impose smoothness constraints, and the heuristic arguments for why this may be the case.

- *Review papers on ADAM, ADAGrad, batch-normalisation*
- *Related reading: DL Ch 8.5 – end of Ch 8*

Recitation lecture 4 – Tensorboard, t-SNE, visualisation

Week 5

Lecture 8 – CNNs

- Understand view of weights in NNs as a correlation filter.
- Understand the need for location/shift invariance of patterns in speech recognition and image classification.
- Understand how "scanning" using sliding windows over inputs can address this problem.
- Understand how scanning can be implemented using the view of a NN with each window corresponding to an identical sub-network.
- Understand how scanning can be implemented via parameter sharing.
- Understand how backpropagation and training of the NN is amended in the presence of parameter sharing, and the associated pseudocode.
- Understand how order of computation can be varied in context of scanning NNs.
- Understand how scanning can be distributed over multiple layers of the network in various ways.
- Understand viewing distributed scanning through the lens of a hierarchical build-up of features.
- Understand the convolution operation and the convolutional neural network (CNN) pseudocode.
- Understand the various benefits of using distributed scanning over layers.
- Understand the view of CNNs through hierarchical composition of features.
- Understand CNN-specific terminology such as receptive fields, filtering, pooling and max-pooling layers, stride, and
- Understand that CNNs are comprised of successive filtering and max-pooling layers.
- *Related reading: DL Ch. 9*

Lecture 9 – Guest lecture: Cascade Correlation Filters (Scott Fahlman, Dean Alderucci).

Recitation lecture 5 – CNNs: basics

Week 6

Lecture 10 – CNNs part 2

- Understand the cursory neurophysiological basis of Hubel and Wiesel's results on vision processing (1959)
- Understand the functional distinction between 'simple' S-cells, and 'complex' C-cells.
- Understand the instructor's narrative on the build-up of more complex patterns through successive layers of S-cells and C-cells.
- Understand the role of position invariance as a motivation for Fukushima's results on the neocognitron (1980).
- Understand how the neocognitron was inspired by Hubel and Wiesel's results, through the lens of S-cells and C-cells.
- Understand how the C-planes and the S-planes interact in the neocognitron.
- Understand the diagrams of the neocognitron, and the viewpoint that it is performing unsupervised learning/clustering.
- Understand how adding external supervision to the neocognitron motivated CNNs.
- Understand the roots of CNNs as computational models of biological phenomena.
- Understand the general architecture of CNNs as consisting of successive layers of convolutional and down-sampling layers.
- Understand how convolutional layers are engineered from linear maps, a learnable filter, a convolution operation, and an activation.
- Understand the illustration of the mechanics of the convolutional operation for a single map, multiple input maps, and be able to tie this together with the mathematical formulae.
- Understand the convention of illustrating CNNs using blocks as 3d filters which scan blocks of stacked arrangements of planes.
- Understand the pseudocode for CNNs in vector notation.
- Understand engineering considerations for the size of the output of the convolution operation, stride, and the rationale for zero-padding.
- Understand formal distinctions between convolution, correlation, and the computational costs of each.
- Understand the mechanics of max pooling in down-sampling layers.
- Understand the engineering considerations behind the size of the output of pooling.
- Be aware of engineering alternatives to max pooling e.g. mean pooling, p-norms, learnable filters.
- Understand that the function of down-sampling layers can be achieved with convolutional layers with stride greater than 1.
- Understand the pseudocode for a fully convolutional network with no pooling.
- Understand typical pre-processing steps for CNNs.
- Understand how the size of inputs/outputs vary according to type of layer being used.
- Understand the detailed schematic, and mathematical representation of a CNN, with particular focus on learnable parameters.

Lecture 11 – Guest lecture: Using CNNS to understand the neural basis of vision (Mike Tarr).

Week 7.

Lecture 12 – CNNs part 3.

- Reinforce understanding of the CNN specific components e.g. convolution operation, convolution layer, convolutive maps, pooling/downsampling, pseudocode.
- Reinforce understanding of forward propagation through a convolution layer (convolution + activation); and a down-sampling layer (pooling operation).
- Reinforce understanding that computation of the convolutive map sums convolutive outputs at all planes (combines maps); computation of pooling is performed independently for each map.
- Reinforce understanding of the familiar NN training setup – the loss function, individual per-instance loss/divergence function, and their derivatives; and gradient descent.
- Understand how the gradient descent update is modified for CNNs.
- Reinforce understanding that from perspective of parameter estimation, backpropagation is still used to compute gradients, and that CNNS are merely NNs with shared parameters.
- Understand that backpropagation for CNNS differs from conventional backpropagation in:
    - Convolutional layers (convolution + activation)
    - Downsampling layers (pooling)
- Understand that computing loss derivatives with respect to activation outputs (within a convolutional layer) involves simple component-wise application of backpropagation.
- Understand the computations occurring in a convolution operation in terms of "influence routes".
- Use consideration of influence routes to understand computing loss derivatives with respect to convolution outputs, and also with respect to weights.
- Understand the pseudocode for backpropagation for a convolutional layer, with strides.
- Understand at a fine-grained level how a single position in a convolution input map influences multiple positions in a single convolution output map.
- Understand how the loss derivative with respect to a single position on a convolutional input map is computed by convolving a left-right, top-bottom flipped filter with the back-propagated loss derivative from the next layer.
- Understand the necessity of zero-padding in this context.
- Understand how filter flipping and zero-padding modifies the CNN back-propagation pseudocode.
- Understand how loss derivatives are computed for downsampling layers with max-pooling and mean-pooling, and the corresponding pseudocode.
- Understand that upsampling layers increase the size of convolutive maps.
- Understand how upsampling layers are constructed using the transpose convolution operation.
- Understand how the weight matrices are adapted, and that stride can be viewed as a scale factor between maps, in this context.
- Understand how other forms of transformation invariance apart from shift invariance can be incorporated into CNNs conceptually, and in pseudocode.
- Understand how to adapt CNNS to adapt to tasks related to object position and location, such as pose estimation and finding bounding boxes.
- Understand how CNNs can be adapted for multi-task learning.
- Understand the distinction between depth-wise and conventional convolution, and associated parameter efficiencies.
- Understand the principle behind dataset augmentation.
- Have knowledge of contemporary CNN architectures, such as Le-Net 5, AlexNet, ZFNet, MobilNet, VGGNEt, GoogleNet, Resnet, Densenet.
- Understand that CNNs may be adapted for tasks outside of visual processing.

Lecture 13 – RNNs

- Understand the problem setup of modelling sequential data.
- Understand the conventions for representing RNNs diagrammatically (enfolded vs unfolded etc.)
- Reinforce understanding of CNNs applied to series data as time-delay neural networks.
- Understand the specification of a generic finite-response system (signal processing terminology).
- Understand the notion of long-term dependencies.
- Understand the specification of a generic infinite-response system.
- Understand various specifications of nonlinear autoregressive networks with exogenous input i.e. NARX networks.
- Understand how "one-tap", generic and "complete" NARX networks use feedback/recursion.
- Understand how "memory" can be engineered explicitly through the use of memory state variables, rather than implicitly through previous output.
- Understand the design of historically significant networks such as the Jordan and Elman networks, using "memory" and "context" units.
- Understand that they implement a form of simple recurrence in operation, but not in training.
- Understand the specification of a generic state-space model.
- Understand that state-space models retain information about the past through recurrent hidden states.
- Understand the various diagrammatic representations of recurrent neural networks (RNNs) architectures.
- Understand the necessity for, and consequence of, using deeper recurrence
- Understand the various formal specifications of RNNs using equations.
- Understand the problem setup for training an RNN – using labelled sequence data.
- Understand that divergence is defined between entire sequences.
- Understand that training an RNN uses backpropagation through time (BPTT).
- Be fluent with the BPTT equations, and pseudocode.
- Understand that BPTT differs from conventional backpropagation in:
    - Need to backpropagate through time, accounting for multiple temporal routes of influence.
    - Addition of recurrent weights
    - Multiple updating of the same gradient expressions.
- Understand that BPTT can be generalised to any variant on the architecture.
- Understand the variant of RNNs with forward recursion – bi-directional recurrent neural networks (BRNNs).
- Understand that BRNNs comprise a forward and backward net that process inputs in two temporal directions; and that computed states of both are used to compute the final output.
- Understand how BRNNs are trained using BPTT, and its implementation in pseudocode.
- *Related reading: DL Ch10*

Week 8.

Lecture 14 – RNNs part 2.

- Understand the comparative capabilities of iterated structures (e.g. time-delay NNs) vs recurrent structures (e.g. RNNs) in terms of short and long-term dependencies.
- Understand problem settings where NNs with recurrent architectures are advantageous.

- Understand the signal-processing notion of bounded input bounded output (BIBO) stability,
- Understand the NN notion of saturation.
- Understand the conditions under which time-delay NNs are BIBO.
- Understand the conditions for scalar linear recursion in RNNs to be BIBO.
- Understand how to generalise this to vector linear recursion using eigen-decomposition.
- Understand that long term stability of linear systems (e.g. simple RNNs) are sensitive to the properties of the hidden-layer weights matrix's eigenvalues.
- Understand that the capacity of RNNs to "remember" information in practice is contingent on the choice of the activation function.
- Understand the saturation properties of the sigmoid, tanh and ReLU activation functions.
- Understand that in context of "memory", tanh functions are more desirable in an RNN setting, but are still limited in their capacity to retain information (i.e. not saturate).
- Understand that similar results can be attained through formal tools such as convergence of Lyapunov functions, Routh's criterion.
- Understand the exploding/vanishing gradients problem for training deep NNs via backpropagation in general.
- Understand how this issue is related to RNN layer weight matrices' eigenvalues and Jacobians.
- Understand that the exploding/vanishing gradients issue applies to RNNs, as instances of deep NNs.
- Understand the consequences of this issue on RNN "memory".
- Understand that the additional design criterion of input-based determinations of "memory retention" motivate the design of long short-term memory (LSTM) architectures.
- Understand the distinction between RNNs and LSTMs.
- Understand the mathematical specification of and schematic diagram for the following components of the LSTM cell: constant error carousel, gates, forget gate, input gate, memory cell update, output gate, output, peephole connection.
- Understand the pseudocode of forward and backward passes of the LSTM cell.
- Understand the motivation for simplifying LSTMs and the gated recurrent unit (GRU) architecture.
- Understand the mathematical specification of and schematic diagram for the components of the GRU cell.
- Understand that LSTMs can, similar to RNNs, be bi-directional.

<u>Lecture 15 – RNNs part 3</u>

- Understand the distinction between time-synchrony and order-synchrony in RNNs.
- Understand the problem settings of the following RNN architectures:
    - One-to-one (conventional NN)
    - Many-to-many (time-synchronous outputs)
    - Many-to-one (sequence classification; order-synchronous time-asynchronous sequence-to-sequence generation).
    - Many-to-many (a posteriori sequence to sequence)
    - Many-to-one (single-input a posteriori sequence generation)
- Understand how a divergences over individual time instances in a sequence can be viewed as a special case of a divergence over an entire sequence.
- Understand how this is used in backpropagation for conventional one-to-one NNs.
- Understand BPTT for many-to-many time-synchronous output RNNs, with emphasis on the computation of divergence between sequences.

- Understand how the input/output representation, and loss function design choices for the above RNN applied to text and language modelling.
- Understand the benefits and costs of using one-hot encoded word vector representations – semantic-agnostic representations vs high-dimensionality.
- Understand how the high-dimensionality problem can be addressed via projections to a lower-dimensional subspace.
- Understand that projection is a simple linear transformation, which itself can be learnt/estimated through incorporation in an RNN.
- Understand the following architectures for text modelling, with emphasis on the learning of projections: TDNN, soft bag-of-words, skip-grams.
- Understand the how this leads to “word embeddings”.
- Be familiar with the literature surrounding word embeddings and language modelling.
-