**Deep Learning for Audio**

1. Machine Learning:
   1. y=f\_theta(x): Parameters theta to be chosen with the aid of data
2. Deep learning:
   1. y=f\_theta’’’’(f\_theta’’’(f\_theta’’(f\_theta’(f\_theta(x)))))
3. Loss functions:
   1. Also known as: cost function, objective function. “Defining how (un)happy you will be with the result”
4. Linear regression:
   1. Y = B0 + B1\*x
   2. Y = X\*B
5. Gradient descent:
   1. Pick random initial parameters theta
   2. Evaluate gradient (del L/del theta) at theta
   3. If gradient is of size zero, stop here.
   4. Update the parameters: thetai ← thetai – alpha \* (del L/del theta)
   5. Go to step 2.
6. Neural nets: [FIG: A neural network with three layers: input [4 nodes], hidden layer [4 nodes], output [1 node]. All nodes are connected to the nodes in the next layer.]
   1. Each node here is a scalar value.
   2. Non-linearites: [FIG: Multiple data points in a spiral formation on a x-y axis.]
7. Life without nonlinearity is unfulfilling
   1. Why?
   2. Every layer fn involves multiply-and-sum of the previous layer’s output.
   3. This linear part of the layer can be written as a matrix projection: Out\_n = Fn · Out\_n−1
   4. A sequence of matrix projections Y = F5 · F4 · F3 · F2 · F1 · X can equivalently be expressed by... (what?)
   5. For NNs, ‘Nonlinearities’ means pointwise scalar functions.
   6. Common nonlinear activation functions: Sigmoid, Tanh, Rectified Linear Unit (ReLU), Leaky ReLU, Parametric ReLU, Swish, Softplus, etc.
   7. For example: ReLU(x) = max(0, x)
8. Why deep learning?
   1. Multi-layer learning can do more than simple logistic regression.
   2. But still: why layers? Why not one cleverly-designed single function?
      1. Biological motivation:
         1. brains seem to do it
      2. Practical usefulness, e.g.:
         1. Train layer-wise
         2. Replace individual layers
9. Theoretical motivations:
   1. Universal approximation theorem (Cybenko 1989, Hornik 1991): “a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of R n , iff the activation function is not polynomial” (NB not about training)
   2. “Backward feature correction”, Allen-Zhu & Li (2020): provable that DL can learn some functions efficiently which NO known single-layer method can. Training: SGD (stochastic gradient descent), in poly(d) time using poly(d) samples.
10. Backpropagation of errors:
    1. “Backprop”
       1. Using chain rule and backward recurrence, we can update the parameters (‘weights’)  
            
          del L / del theta\_n = del Fn / del theta \* (out\_n-1, theta\_n) \* del L / del out\_n
11. Divide your data: training, test
    1. Data set preparation: Training data, testing data
    2. Feature processing: Spectogram, Spectogram
    3. Classification: Train classifier, Apply classifier.
    4. Decision
    5. Note: it’s vital that you do not use your test data at all, until all your training, tweaking, training, tweaking, ... is finished.
12. Divide your data even more
    1. Many problems are too hard for simple gradient descent.
    2. my GPU can’t hold it all
       1. ...so divide training data into ‘minibatches’
13. Minibatches and gradient
    1. Minibatch = a subsample from the training dataset   
         
       del L\_{minibatch} = an estimate of del L
    2. Q: Is it a good estimate? Biased? Highly variable?
14. Overfitting/underfitting
    1. A NN’s ‘capacity’ should be appropriate to the complexity of the problem
15. Number of parameters
    1. AlexNet (2012): 8 layers, 62 million params
    2. VGG16 (2015): 16 layers, 138 million params
    3. ResNet50 (2016): 50 layers, 23 million params
16. Training can easily fail
    1. As depth and/or num params increase, we risk:
       1. Local minima
       2. Vanishing/exploding gradients: Backpropagating through many layers → deeply-nested multiplications of ‘signal’ → param update can converge to 0 or diverge to infinity
17. Stabilising training
    1. Most of the important advances have been (in part) about stabilising the way NNs respond to training.
    2. rectifier nonlinearity (ReLU)
    3. batch norm
    4. dropout
    5. residual networks
    6. Adam
    7. RMSprop
    8. ...and CNNs?
18. CNNs match pieces of the images [FIG: a 9x9 grid with all -1’s except 1’s in a circle formation. If the 1’s do not match in the input image, then the classification outputs negatively.]
19. Going convolutional
    1. Shared weights among neurons
    2. Dramatically fewer free parameters → easier to train
    3. Shift-invariance built in
    4. Locality built in
    5. Axes preserved (e.g. frequency, time)
20. Going convolutional
    1. Who does convolution work on 2D data?
       1. (−1 \* −1) + ( 1 \* −1) + ( 1 \* −1) + . . .
       2. ( 1 \* −1) + ( 1 \* 1) + ( 1 \* 1) + . . .
       3. ( 1 \* 1) + ( 1 \* −1) + ( 1 \* −1) = 1 … 3 … 9 … 5 [feature map values]
21. Channels
    1. Convolutions usually computed for each channel and summed  
         
       (k \* g^c) = sum of (k^c \* g^c)
22. Convolutional layer parameters
    1. An image input 128 × 128 × 3 I 100 kernels of size 5 × 5 × 3
    2. Number of parameters of this layer?
       1. For every filter: 5 × 5 × 3 + 1 = 76 parameters.
       2. Total parameters: 100 × 76 = 7600 parameters.
    3. Think about how many parameters of a fully-connected layer with 100 and 1000 hidden units in this case?
       1. 100 hidden units: (128 × 128 × 3) × 100 + 100 = 4, 915, 300 parameters.
       2. 1000 hidden units: 49, 153, 000 parameters.
23. Dense layers
    1. Also known as: fully connected (FC) layers
    2. Typically 2 or 3 as end of CNN classifier. Why?
24. Max-pooling
    1. Downsampling. Shift invariance.
    2. Alternatives: mean-pooling
25. Invariance and equivariance
    1. For some operator a(.),
    2. Invariance: f (a(x)) = f (x) [max-pooling]
    3. Equivariance: f (a(x)) = a(f (x)) [equivariance]
    4. See also: Group theory
26. Typical CNN architectures for audio:
    1. Classify a spectrogram
       1. Spectrogram as input
       2. 2D real-valued matrix
       3. “as if it was an image”
       4. Many alternatives to spectrogram as input—to be discussed later—but this is overwhelmingly common and successful
       5. Many tasks “treated as classification” (of genre, speaker, species, ...)
    2. Classify each column of a spectrogram
       1. Automatic speech recognition (ASR)
       2. Tracking the ‘note’ (pitch) of musical instruments
    3. Classify each column of a spectrogram
       1. Automatic speech recognition (ASR)
       2. Tracking the ‘note’ (pitch) of musical instruments
27. Receptive fields
    1. “Receptive field” for a selected neuron: The subset of input features whose values affect that neuron’s value.
28. Receptive fields – in time-frequency
    1. The spectrogram settings (sample rate, frame length, hop) also affect the receptive field.
29. Receptive fields – the least bittern · ‘coo coo coo’ breeding call:
    1. 0.5 seconds, 500–600 Hz
    2. Recorded at 44.1 kHz
    3. Spectrogram: frame size 1024, 512 hop
    4. 3x3 conv →2x2 maxpool →3x3 conv
    5. Calculate: bandwidth; frame rate; receptive field needed...
30. To recap:
    1. We started with an extremely simple network
       1. Layers
       2. Nonlinearities, loss function, gradient descent
    2. Made it convolutional
       1. Constrains its flexibility
       2. Reduces num trained parameters
       3. Enforces some general principles
          1. Locality
          2. Shift invariance/equivariance
       4. Maxpooling
    3. Applied it to spectrogram data
       1. Typical architectures
       2. Receptive fields