**Machine Learning**

**Session 10**

1. Batch gradient descent:
   1. All N examples are fed to the network. Weights are updated only after all examples have been presented to the network (1 epoch of training).
   2. The batch-size is B = N. There are N/B = 1 iterations per epoch
   3. For each example: for each weight wi the corresponding gradient is computed. The weight wi is updated (by Delta wi) based on the average gradient over N examples.
   4. Weights are updated only per epoch.
2. Mini-Batch Gradient Descent:
   1. Mini-Batch: M randomly examples are fed to the network.
      1. Batch-size = M (M usually 32, 64, …, 512)
      2. M/B iterations per epoch
   2. For each example: for each weight wi the corresponding gradient is computed. The weights are updated (by Delta wi) based on the average gradient over M examples.
   3. Weights are updated M/B per epoch. Updates are noisy. This turns out to be good for training!
   4. This algorithm is called Stochastic Gradient Descent
3. Backprop stopping criteria
   1. When the gradient magnitude (or Delta w) is small, i.e. ssum of(modulus of Delta wi) less than epsilon
   2. When the maximum number of epochs has been reached
   3. When the error on the validation set does not decrease
      1. Error might decrease in the training set but increase in the validation set (overfitting!)  
           
         Fig: xy graph with error on y-axis and training iterations on x-axis with the training error and test set error displayed. Both are plotted as decreasing 1/x functions with the training set error being lower than the test set.
4. SGD with momentum:
   1. Standard SGD:  
        
      wi if and only if (wi + delta wi)  
        
      Delta wi = -n(del E / del wi)
   2. Problem with standard SGD: Same learning rate across axes
   3. SGD with momentum:  
        
      u = 0.9  
      Delta wi(t) = u Delta wi (t-1) + (1-u)(del E / del wi(t))  
      wi if and only if (wi – n Delta wi(t))
5. Learning rate decay
   1. Start with a high (depends on the problem/dataset) learning rate and decay it slowly.
   2. Typical values for initial learning rate, 0.1, 0.01 (for SGD)
   3. Step decay: Reduce the learning rate by some factor every few epochs, e.g., divide by 2 every 50 epochs
      1. Use a validation set to find when to decrease
   4. Keep learning rate constant for T epochs and then decrease as follows:   
        
      Lr\_t = Lr\_{t-1} \* scaling factor (e.g., 0.99)
   5. You can think your own way too!
6. Learning rate decay:  
     
   Fig: different decay rates plotted with the ideal one resembling a 1/x function.  
   1. If loss increases the learning rate is too high
   2. If loss goes down slowly the learning rate is low
   3. Adjust also learning rate based on validation accuracy
   4. Start with higher learning rate and reduce after several epochs.
7. Weight initialization:
   1. Common weight initialisation techniques
      1. Gaussian distribution with mean = 0, var = 0.02
      2. Xavier: Gaussian distribution with mean = 0, var = sqrt (2 / (n1+n2))
         1. n1, n2 are the number of neurons in the previous and next layers, respectively.
      3. He Initialisation: Gaussian distribution with mean = 0, var = sqrt (2 / n1)
         1. n1 is the number of neurons in the previous layer.
      4. There are many variants of the above
         1. (E.g. Gaussian could be replaced by Uniform)
8. Regularization
   1. Regularisation is a standard technique to reduce overfitting in Machine Learning. Regularisation techniques for NNs include:
      1. Early Stopping
      2. L2 regularization (weight decay)
      3. Dropout
      4. Augmentation
      5. More exotic ones (e.g. mix-up)
9. L2 Regularization
   1. Large weights dominate the network’s prediction
      1. because they are tuned to training set, this could lead to overfitting
   2. Better if all weights have “equal” contribution   
        
      E = E0 + 0.5 \* lambda \* sum of ( w^2 )
   3. E0 is the original loss function, e.g., L2 loss, CE
   4. We wish to minimise the original error function E0
   5. We also wish to penalise large weights, keep the weights small (second term)
      1. Small lambda: we prefer to minimise E0
      2. Large lambda: we prefer small weights
   6. This is also called weight decay
10. Dropout
    1. We don’t modify the error function but the network itself
    2. During training neurons are randomly dropped out
    3. A simple way to prevent neural networks from overfitting
    4. The probability that a neuron is “preserved” is p
    5. Dropout prevents overfitting because it prevents neurons from co-adapting too much.
    6. Each neuron should create useful features on its own without relying on other hidden units to correct its mistakes.
    7. Test time: outgoing weights of a neuron are multiplied by p
11. Data augmentation
    1. One of the best ways to avoid overfitting is more data
    2. We can artificially generate more data by introducing artificially more variations.
    3. For images: flip left-right, colour, rotate, random cropping, etc.
    4. At each batch: sample data, apply random augmentation, backprop
12. Summary
    1. Mini-Batch Gradient Descent (SGD) introduces noise that is beneficial for learning.
    2. Optimisation tricks are crucial.
       1. Momentum, Adam are often used. Often simple SGD works well.
       2. Set learning rate by monitoring the cost. Scheduling of learning rate
       3. Regularization is typically used. Dropout often.
       4. Data augmentation essential
13. Convolutional Neural Networks
    1. Convolutional Neural Networks (CNNs) have been very successful in computer vision
    2. Invented by LeCun in the 80s
    3. Reference paper: “Gradient-Based Learning Applied to Document Recognition”, Proc. IEEE, 1998
    4. The method of choice for extracting features from images, video and audio
    5. Image classification has been one of the most difficult problems in AI
    6. Computer Vision researchers have shown contempt for CNNs
    7. This paper changed everything: Krizhevsky, Sutskever & Hinton: “ImageNet Classification with Deep Convolutional Neural Networks”, NIPS 2012
14. ImageNet Competition – Object Classification:
    1. Classification of 1000+ objects
    2. State-of-the-art before 2012: error rate approx. 26%
    3. New state-of-the-art in 2012 with deep networks: error approx. 15%
    4. Tricks: Large dataset, GPUs for training, Augmentation, Dropout, ReLU
15. AlexNet
    1. It’s a deep network = many layers
    2. Each layer is either a convolutional layer or subsampling layer
    3. Final layers are fully connected layers
16. Convolution Motivation
    1. A standard —also called Fully Connected (FC) — layer is not suitable for images
    2. E.g.: Assume a grayscale image (1 channel) of resolution 256x256
    3. A single neuron connected to all pixels will require 2562 = 65K params
    4. If first layer: 128 of those neurons, in total 65\*128= 9M params !
    5. Convolution Layer:
       1. Divide the image into 3x3 windows.
       2. Apply the same FC layer for all windows
       3. A single neuron will have 3x3 params, and the whole layer 1K params!
17. Convolution – 1 input channel:
    1. FIG:
       1. Image: A 6x6 gird
       2. Zero-padded image: A 8x8 grid with the 1st and last row AND column blacked out.
       3. Neuron: A 3x3 grid
       4. Multiply each value in the neuron with corresponding value from image patch
       5. Sum over values
       6. Shift the neuron by one pixel and repeat to get the next value in the feature map
18. Convolution – 3 input channel:
    1. FIG: (same dimensions are before)
       1. Image: set of three grids
       2. Neuron (filter): set of three grids
       3. Same process per channel
          1. One immediate feature map per channel
       4. Final feature map:
          1. Sum over all channels
          2. Again a 2D matrix!
       5. Input to the next layer us a stack of N feature maps
19. Convolution: Why this works
    1. Image patches have similar statistical properties: we can analyse all of them using the same neuron (filter).
    2. CNNs have the equivariance property: if an object is shifted by some amount this results in a “shifted” output representation
    3. This makes a lot of sense for images (but also audio!)
    4. Not the case of FC layers
    5. An example from AlexNet
       1. Input layer has spatial resolution 27x27 and 128 channels (features).
       2. If input image layer is RGB: 3 channels
       3. The number of channels (128) is equal to the number of neurons of the layer L-1.
       4. A neuron from layer L to L+1 operates on 3x3 patches with 128 channels. So it has 3x3x128 weights
       5. Layer L+1 has 192 channels: hence, at layer L there are 192 neurons with 3x3x128 params each
       6. Each of them “scans” the input producing a feature map of 27x27
       7. Because of max-pooling this is reduced to resolution 13x13
       8. Since there are 192 neurons at layer L, there are 192 feature maps at L+1
20. AlexNet — in more detail:
    1. 5 Convolutional layers and 3 Fully Connected layers
    2. ReLU is applied after every Conv and FC layer.
    3. Dropout is applied before the first and the second FC connected layer.
    4. 90 epochs, SGD with momentum, wd=.0005, LR = 0.01— divided by 10 3 times when validation accuracy plateaus
21. CNNs for face recognition
    1. One of the most known applications of Computer Vision is face recognition
    2. With CNNs researchers reporting super-human accuracy
22. VGG-Net
    1. An extension to AlexNet which reduces the number of parameters
    2. This facilitates training
    3. Main idea: replace large kernels, e.g. 5x5 or 7x7 with a sequence of small ones
    4. For example one can show that a 5x5 convolution is equivalent to two 3x3 convolutions applied sequentially
    5. 5x5: 25C parameters vs 2 3x3: 18C parameters
    6. larger reductions for larger kernels (e.g. 7x7)
23. VGG-Net
    1. This enabled training deep networks (up to 19 layers)
24. Res-Net
    1. A simple method for training very deep networks
    2. This facilitates learning very powerful networks
    3. Deep residual learning for image recognition
    4. Main idea: add the input feature to the layer’s output
    5. output = F(input) + input; F: conv layer.
    6. called skip connection
    7. The advantage is that there’s a direct path for propagating gradients during back-prop
25. Res-Net, bottleneck layer
    1. Since the network is very deep now, the complexity is high.
    2. A bottleneck design is used to reduce the complexity
    3. ResNets greatly improved accuracy on ImageNet
    4. ResNet-152 still has lower complexity than VGG-19 !!
26. Summary
    1. Convolutional Neural Networks “filter” feature maps (e.g., multichannel “images”).
    2. Architectural choices, filter sizes, skip connections (residual networks), network depth lead to large differences in performance.
27. Fully Convolutional Architectures
    1. From image to label map
    2. No Fully Connected layers exist
    3. For example, assume we want to label every pixel with an object class label (semantic segmentation)
    4. The loss is applied in the end at each spatial location
28. Transfer Learning
    1. Pre-train a network on one dataset (usually large scale, with lot of classes)
    2. ImageNet is the best candidate
    3. Use the weights as initialisation for a new dataset
    4. If the new dataset is small then this works much better than initialising from scratch
29. Multi-task learning
    1. A train a single network to perform several tasks
    2. A single network is shared for all tasks
    3. Specialised networks (called heads) for each task
    4. In some cases multi-task learning improves accuracy (with lots of data better train a single network)
30. Unsupervised Learning
    1. ImageNet pretraining works well but we want to do better
    2. Collecting and labelling a bigger dataset very expensive (does not scale up)
    3. Solution: pretrain a network on large amounts of data without labels!
    4. We need to define a task that does not require human annotation.