CS231n - Stanford

Lesson 6

–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Lesson 5

–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Mark I Perceptron Machine, 1957, Frank Rosenblatt: First implementation of a Perceptron

CNN applications:

* Classification (Traditional)
* Object Detection (-> Bounding Boxes)
* Segmentation (-> Labelling every pixel -> show the outlines)

You can visualize what would produce the maximum activation for a given layer to find out what the neuron is looking for (e.g. horizontal, vertical edges)

Formula for output size: (N-K) / stride + 1 , where if you pad it just add 2p to N

Formula for no of params: (K \* K \* C(N) + 1) \* C(K) e.g. for 10 5x5 filters on 32x32x3 we get 5\*5\*3 + 1 = 76 -> 76 \* 10 = 760

Key difference from MLP: Neurons only have local connectivity, i.e. for each new pixel we do not look at all pixels available but just in the kernel size

5x5 Filter can be called a 5x5 receptive field the neuron is receiving

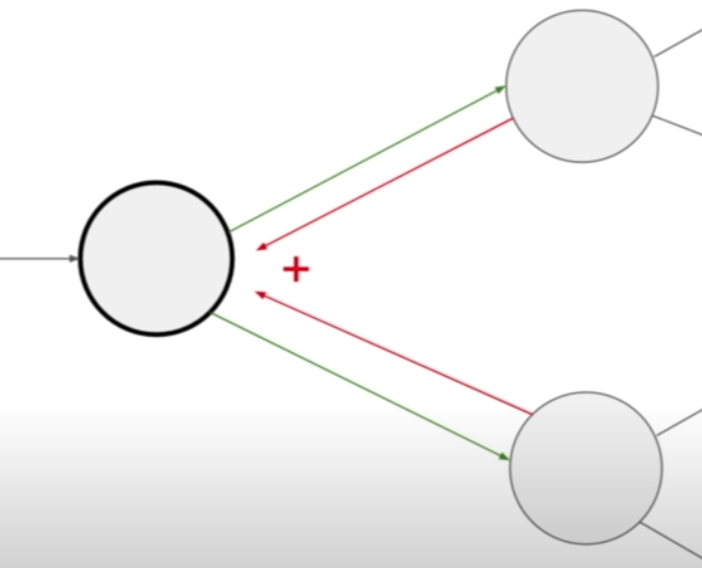
Maxpooling makes more sense than average pooling, as we’re pooling over activations. Hence the more a neuron fired in a certain location the more relevant that should be and we want to keep the max value, the spike.

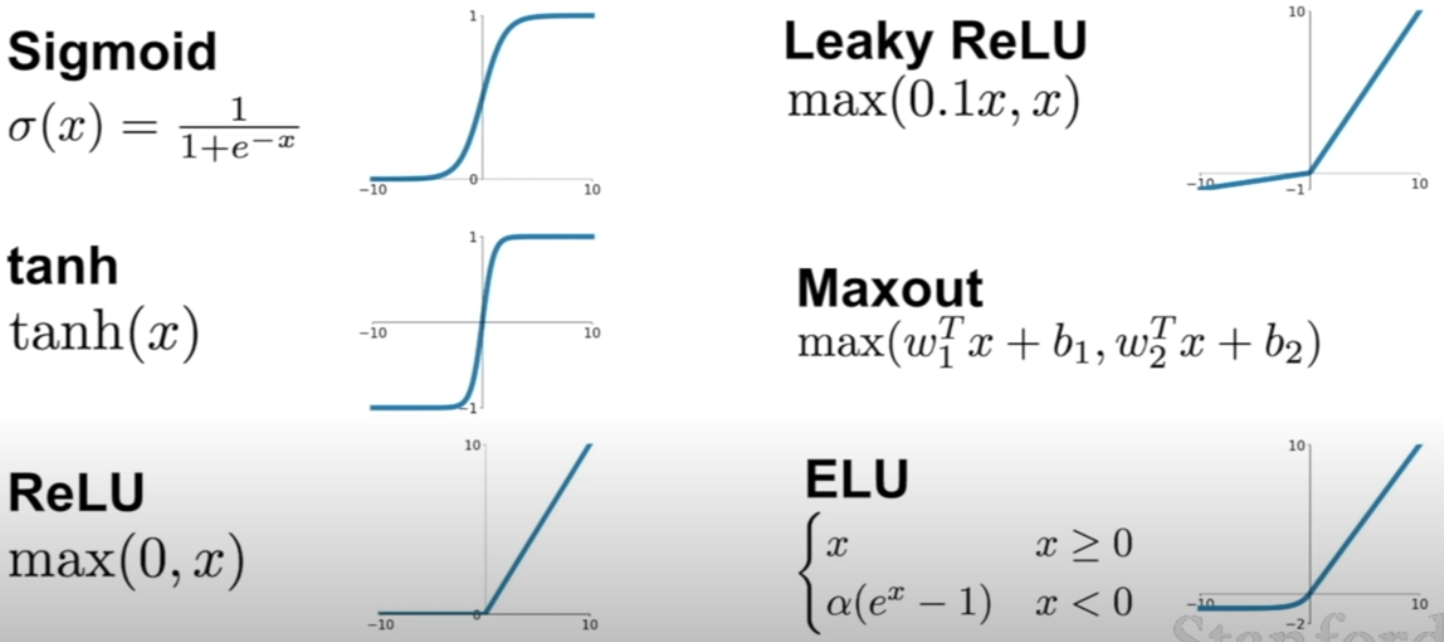
Visualizing Conv Layers:

<https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>

Lesson 4

–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––



* Anytime you’re troubled w/ finding gradients, just break it down into the most simple computation graph
* Gradients add at branches
* Jacobian matrix: Matrix of all first order partial derivatives
  + -> For an input of 4096 x’s & a output of 4096 x’s (in between e.g. a max(x, 0) function, the jacobian matrix will be 4096^2 in size (4096 rows, 4096 cols)
  + -> This is just for one item; if we have a minibatch w/ 100 examples it will be 409.600^2 …
    - -> We don’t need the whole matrix
* Good double-check: Does the gradient with respect to a variable have the same shape as the variable itself?
* ReLU is considered to be have most similar to real neurons firing /spiking rate (the sp/fi rate is considered to be the ‘activation function’ of neurons)

Lesson 3

–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

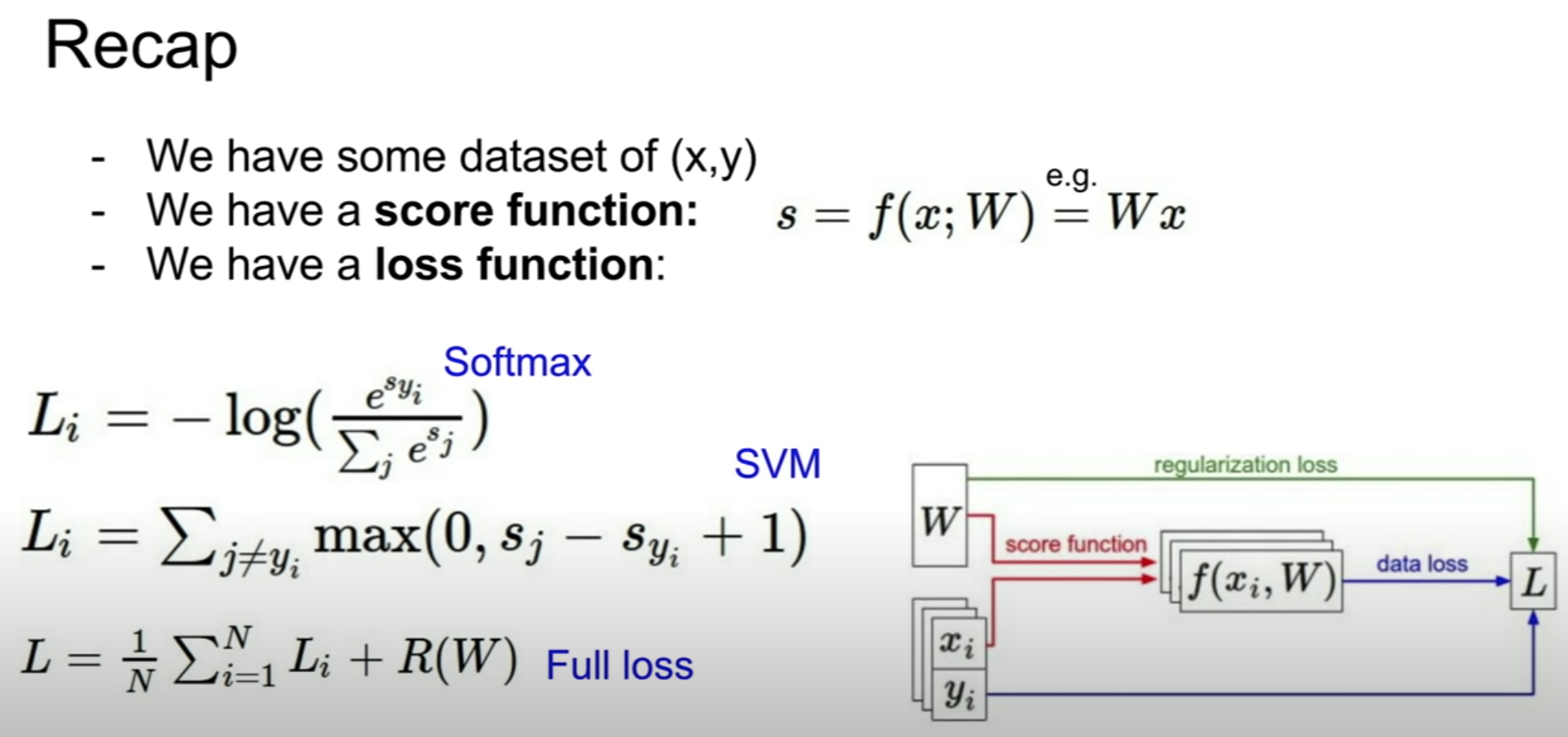
Feature Representation of Pictures:

* #1 Color Histogram:
  + Sort pixels into different buckets on Hue colored histogram
* #2 Histogram of oriented gradients (HoG):
  + Measure the local orientation of edges in image
* #3 Bag of Words for images:
  + Create ‘image words’ as different clusters
* #4 CNNs
  + Instead of writing down the features ahead of time, we’ll learn them

Optimization:

* #1 Random Search: Trying random Ws
  + 15.5% Accuracy on CIFAR10 (10% is worst case)
* #2 Use the geometry: Calculating the Gradient = Partial Derivatives along each Dim
  + Direction of steepest decent is the negative gradient
  + Slope in any direction is the dot product of the unit vector describing the direction & gradient
  + Numeric gradient (Gradient of finite differences via limit) is computationally to expensive, but can be used for grad checking (debugging) – w/ scaling down the Ws tho)
  + Analytical gradient (Derivative expression of function) is used in practice
  + <http://vision.stanford.edu/teaching/cs231n-demos/linear-classify/>

Loss:

* SVM / Hinge Loss (Because the curve is shaped like a hinge)
  + Intuition: We are happy if the true score is much higher than all the other scores (by some safety margin)
    - If not, we will incur some loss
    - The calc score for the other (wrong class) - The calc score for the correct class
      * -> If this is is lower than zero than we are happy
      * -> If this is higher than zero we are unhappy, i.e. this becomes the loss
    - For safety margin, we add +1 at the end
      * Whether we add +1 or else, does not matter since it will simply increase the scale of the W’s
  + If we initialize weights to 0, in the first iteration we should have loss of about N (classes) - 1 if 1 is our safety margin
    - If thats not the case, you probs have a bug!
  + If you loop over all classes instead (incl. the correct one) your loss will just be 1 higher and hence the optimum will be at Loss = 1
  + If you use mean instead of sum, it won’t change the progress, as you’re just rescaling the whole function
  + Squared Hinge Loss is also used – Changes the result
    - -> It will penalize ‘very wrong’ scores a lot more
    - Normal Hinge Loss penalizes very wrong and slightly wrong scores differently
* Occams Razor:
  + Among competing solutions the simplest is the best
    - Because it is more testable – For the complex solutions, there is probably an infinite amount of more complex alternatives to accomodate for each edge case
  + In order to get simpler models we add a Regularization term to our loss – It penalizes e.g. high degree polynomials
  + Various types of regularization:
    - L2 Regularization / Weight Decay: Square of Weight Vector (Maybe halved)
    - L1 Regularization / Weight Decay: Euclidean Norm of Weight Vector
    - Elastic Net (L1 + L2)
    - Max norm regularization
    - Specific to DL:
      * Dropout
      * Batch Norm
      * Stochastic Depth+
* L1 vs L2 – Two different notions of complexity:
  + L2: Prefers Weight vectors where the sum is spread across all values i.e. 0.25, 0.25, 0.25, 0.25 -> The Square of this W will be fairly low – Simplicity in spacing out
    - -> Spaces out to different W values
  + L1: Prefers where the Weight is more concentrated i.e. 0.25, 0.25, 0.25, 0.25 -> The Euclidean Norm will be higher than for the L2 (Higher penalty) – Prefers solutions such as 1, 0, 0, 0 with sparse distribution – Simplicity in concentration?
    - -> Drives many of your W values to 0 except for a couple
* Regularization works because both of them penalize high W’s // high complexity (Imagine the Ws are the a’s in a polynomial function f(x) = ax + ax^2 …) By driving the a’s down and deleting some polynomials you increase simplicity
* Softmax function (Multinomial Logistic Regression) to turn scores into probability distribution
  + L Function becomes: L = -log( e(y) / Sum(e(j) )
    - -> Negative log of the softmax function
    - Log, since easier to maximize (monotonic function)
    - Negative, since a higher softmax is better and this would give us a higher log, but loss function should measure how bad it is
  + Called Softmax Loss
  + Min Loss: 0; Max Loss: Infinity
    - You’ll never get a 0 loss though, as due to the normalization you’d need +infinity for the correct score and -infinity for all others
    - You’ll also neer get a infinity loss (i.e. -log (0)), as you’d need e(y) to become zero which needs -infinity
  + If initalize all W to 0 -> -log ( 1 / C) -> = **log (C) -> Check if this is true**
    - C = N classes
* SVM vs Softmax:
  + SVM will stop pushing after the threshold / bar is reached
  + Softmax continues to try to push the correct value to +infinity and the incorrect ones to -infinity

Lesson 2

–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

* Semantic Gap:
  + Diff btw. idea of a ‘cat’ and the pixel values
* Challenges for C Vision
  + Viewpoint
  + Illumination
  + Deformation
  + Occlusion
  + Background clutter
  + Intraclass variation (Different species of cats)
* K-nearest neighbor (<http://vision.stanford.edu/teaching/cs231n-demos/knn/>):
  + Hyperparams: K & Distance Metric
  + Training: Memorize all training data & labels
  + Prediction: Output label of the most similar image
    - Compare via L1/Manhattan distance: sum of Absolute differences of Pixel Values
      * Turns circle’s distance from center into square
      * Changes with changes in coordinate frame
      * -> Good if specific features are really important
    - Or: L2/Euclidean Distance: Root of the sum of the squared differences
      * Turns circle’s distance from center into circle
      * Robust to changes in coordinate frame
      * -> Good for more generic vectors
  + Using K neighbors, instead of just 1, smoothes out the decision boundaries (You always want to use K > 1, e.g. 3 or 5)
  + Speed: Train O(1); Test O(N)
    - Bad, as we want it the other way round; Fast at testing!
  + Does not differentiate different changes (i.e. adding 100 on the right or on the left = same)
  + Curse of dimensionality:
    - Data needed for result grows exponentially w/ dimensionality
    - -> For 1D Line you just need two separators -> 2D exponentially more
* Parametric Approach: Linear Classifier
  + f(W,x) = Wx + b
  + Restricted to learning only one weight template per category -> Not good to generalize to diff backgrounds repreentations
  + Difficulty of separating w/ just one line:
    - Parity Problem (separating odds from evens)
    - Multimodal situations (Category 1 has multiple different ‘islands’ in the space)
* Ideas for setting hyperparameters:
  + Train on one dataset -> K=1 will turn out to be the best; BAD
  + Train & Test set -> We tune acc. to test set -> No idea how performs on new data; BAD
  + Train, Val & Test set -> We tune acc val set & at the end! use test set; BETTER
  + Cross validate Train & separate Test set -> Great for small datasets; BEST

Lesson 1

–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

* Onset of vision might have lead to big bang in evolution
  + -> Vision makes life much more interactive (predators etc)
* Almost 50% of our neurons in our brain are involved in neural processing
  + 80% of our perceptions are visual
* Visual Processing of Humans/Mammals:
  + Acc to Hubel & Wiesel; it starts with edges and builds up
  + Block World – Larry Roberts:
  + David Marr – Vision: Input Image -> Edges/Blobs -> 2.5 D Sketch (Depth, Layers) -> 3D