CIS 522 - University Of Pennsylvania, Spring 2020

Lecture 4: Components of Deep Learning - Part I

28 January 2020

Lecturer: Konrad Kording *Scribe: Jianqiao, Nidhi, Kushagra*

1. Activation functions
2. Loss functions
3. Initializations

1 Activation functions

Activation functions are the atomic nonlinearities that make a deep neural network nonlinear and able to approximate arbitrary nonlinear functions as shown by the universal approximation theorem. We saw a few examples of activation functions in the previous lecture: (i) ReLU (ii) Leaky ReLU (iii) tanh sigmoid (iv) logistic sigmoid, and (v) softmax.

Recall that ReLU stands for Rectified Linear Unit and is defined by: $\sigma(z) = max(0, z)$. Here, we briefly discuss why ReLUs are most commonly used in current practice, say, compared to tanh or logistic sigmoidal activation functions. As noted in the previous lecture, the softmax activation function is not like the other three activation functions, in that its purpose is to convert a real valued array into probabilities (with range 0 to 1), rather than just introduce a nonlinearity.

1. ReLUs are simple and computationally efficient. As they consist of just a simple thresholding operation, that is checking whether the input z is greater than zero, ReLUs can be computed much faster than more nonlinear functions such as leaky ReLUs, tanh or other sigmoids. Analogously, the derivative of ReLU is piecewise constant: zero or one, depending again on whether the input z is greater than zero. This simplicity can provide a small constant-factor speed advantage.

2. The sigmoidal activation functions are susceptible to the so-called "vanishing gradient" problem. Training a neural network usually involves solving an optimization problem, for instance, using stochastic gradient descent, which needs good gradient information to make consistent progress. Because the sigmoidal activation functions have two mostly-flat regions and a relatively narrow region where the gradient is substantially non-zero, using sigmoids can result in almost-zero gradient values relatively easily — for instance, if the initialization of parameters were inappropriate, in a regime where the gradients are mostly small. Such vanishing gradients make it hard for gradient descent to make progress. ReLus are less susceptible to this vanishing gradient problem, as their gradient is non-zero for half the domain (a half space).

3. Presumably partly because of reason-2 above, ReLUs show better performance in practice as evaluated by current practice (e.g., see Krizhevsky et al. 2012, the AlexNet paper). Historically, most articles using neural networks used tanh or logistic sigmoidal activation functions, but the popularization of ReLUs by Krizhevsky et al. (2012) and others contributed to dramatically increased trainability and performance of deep neural networks, kick-starting the recent interest in deep learning.

4. Finally, to the extent that it useful to be inspired by biological neurons, the thresholding nature of a ReLU is more similar to (at least simple models of) biological neurons, which tend to fire beyond a threshold.

2 Loss functions

A deep neural network-based learning algorithm tries to find an optimal mapping from a sets of inputs to those outputs. The training problem is treated as an optimization problem, based on input training data. Popular choices for the optimization are variants of stochastic gradient descent algorithms [5]. Here the gradient refers to the gradients of training error, or the training loss, which is typically a surrogate function for mis-classification.

2.1 Statistics-Inspired Loss Function

There are many functions that could be used to estimate the error of a set of weights in a neural network. Past research on basic statistics and stochastic process give us strong tools in modelling the random events, e.g. the pixel-wise noise in an imaging sensor follows a Gaussian distribution by the statistical physics principle. Therefore for such a denoising application, we tend to build a Gaussian additive noise model and use ℓ_2 norm distance as the loss function. A statistically-inspired loss function helps the optimization algorithm converging to a consistent estimation for the weights.

The probability distribution should be set by the principle of data generating process. E.g. in neural science, the number of occurrences within a time interval can be modelled by Poisson distribution `https://www.cns.nyu.edu/~david/handouts/poisson.pdf`

$$f(k;\lambda) = P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}. \tag{1}$$

For some low-level (pixel-level) computer vision application like denoising, where the physics principle are known to additive, i.e. the observation approximately equals to the unbiased image plus Gaussian noise

$$P(X) = \mathcal{N}(X_0, \sigma^2 I_D) \tag{2}$$

where X_0 represents the images before noising effect, and σ is the variance of noise. Sometimes, there are random corruptions to those pixels, like salt-and-pepper noise, we could use loss function that are more robust to outliers, like Laplace distribution likelihood,

$$P(X; X_0, b) = \frac{1}{2b} \exp(-\|X - X_0\|/2b). \tag{3}$$

Taking logarithms of Laplace distribution likelihood gives a ℓ_1 distance. A denoising algorithm for such corruption noise is the median-filter, i.e. predicting each pixel to be the median value of all pixels in its neighborhood. This also motivates the view of l_1 distance being tolerant to outliers which can be thought of as being on extremes of the distribution.

To model the difference between two unit vectors, e.g. normalized embedding vector of words in NLP applications, we could use cosine similarity loss function.

$$L(X,Y) = \frac{1}{N}\sum_n \frac{X_n^T Y_n}{\|X_n\|\|Y_n\|}. \tag{4}$$

2.2 Information-theoretic Loss function

A popular loss function for multi-class classification problem is the cross-entropy loss function. Cross-entropy is a measure of the difference between two probability distributions for a given random variable or set of events. We recall that information entropy quantifies the number of bits required to encode and transmit an event. Lower probability events have more information, higher probability events have less information. A skewed distribution has a low entropy, whereas a distribution where events have equal probability has a larger entropy. The entropy of a random variable x with a continuous probability distribution $P(x)$ can be calculated as follows:

$$H(x) = \int P(x)\log(P(x))dx \tag{5}$$

The cross entropy measure two probability distribution p and q over the same set of events, about how many bits in expectation do we need to identify that an event is drawn from p rather than q . The analytical formula takes in two distributions, $p(x)$, the true distribution, and an estimated distribution $q(x)$, which are both defined over the variable x, and is given by

$$H(p,q) = -\int p(x)\log(q(x))dx. \tag{6}$$

$$\mathcal{L} = -y^T \log(\hat{y}). \tag{7}$$

2.3 Geometric Loss Function

The lecture also involves geometric loss for camera localization, which is a fundamental problem in robotics. As this may be unnecessarily complicated, we refer to the original paper [4]. This paper may takes some for the readers if you are not familiar with robotics or kinematics, i.e. quaternion. But as deep learning-based robotics are now a popular research direction, for those who are interested in vision or robotics, we strongly encourage the students to read the full paper.

The loss function measures the difference between two poses, i.e. position x and orientation q. The distance for position is ℓ_γ norm, where γ may be 1 or 2.

$$\mathcal{L}_x(I) = \|x - x\|_\gamma, \tag{8}$$

The orientation is described in quaternion, which a four dimensional vector that is easy to interpolate and differentiate,

$$\mathcal{L}_q(I) = \|q - \frac{\hat{q}}{\|\hat{q}\|}\|_\gamma, \tag{9}$$

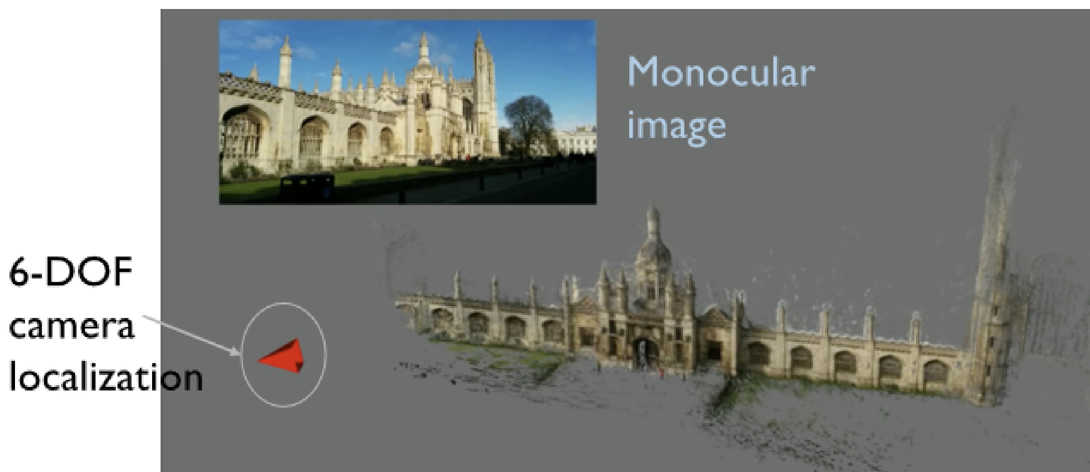
The total loss function combines the two loss functions above with an additional hyperparameter β, which is tuned to input data:

$$\mathcal{L}_\beta(I) = \mathcal{L}_x(I) + \beta\mathcal{L}_q(I). \tag{10}$$

While this seems to work well in some localization experiments, the paper [4] suggests to use statistically-theoretical hyperparameter β, which minimizes the uncertainty on some variance hyperparameter σ. Besides, they also suggested a hyperparameter-free loss function that uses reprojection loss function, which measures the distance between two reprojected points in 2D camera plane from the target camera pose (x, q) and the predicted camera pose

$$\mathcal{L}_g(I) = \frac{1}{|\mathcal{G}|}\sum_{g_i \in \mathcal{G}} \|\pi(x, q, g_i) - \pi(\hat{x}, \hat{q}, g_i)\|_\gamma, \tag{11}$$

where π is the projection operator that maps any 3D points to the image plane, and $\mathcal{G}$ represents all 3D points that are detected and registered.



(a) Geometric loss for camera localization

2.4 Perceptual Loss Function

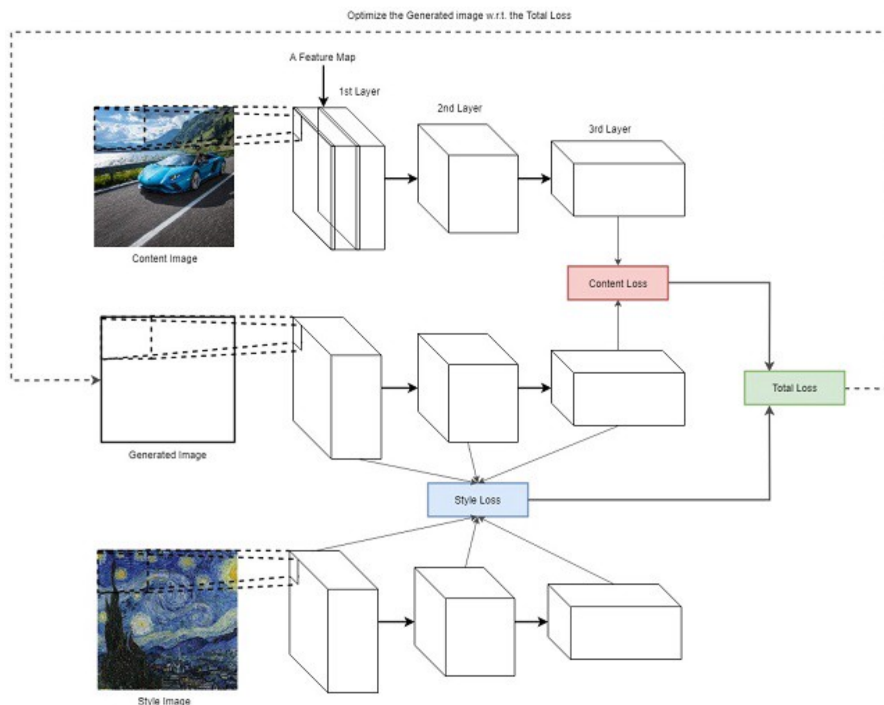
We discuss one recent innovation in loss function that is quite unique in deep learning applications, particularly in image synthesis [1] and artistic rendering[2]. We may refer to this loss function as perceptual loss, as it relates to high-level features As previous loss functions were mostly developed before the deep learning era, and can be used in non-deep learning algorithms.

The loss function defines the distance between two images in their high-level features that are extracted by a deep network. Assume that the feature map of a sample texture image I_s at layer l of a pre-trained deep classification network (on ImageNet classification task) is $F^l(I_s) \in \mathbb{R}^{C \times H \times W}$ where C is the number of channels, and H and W represent the height and width of the feature map $F^l(I_s)$.

How to define the similarly of visual appearance of two images? The original NST paper [2] uses the Gram-based representation can be obtained by computing the Gram matrix

$$G(F^l(I))_{i,j} = \sum_k F^l(I)_{i,k} F^l(I)_{j,k}, \tag{12}$$

over the feature map $(F^l(I_s))$ (a reshaped version of $F^l(I_s)$, where i, j are indices for channels and k is the index for position in layer l. This Gram-based texture representation from a CNN is effective at modelling wide varieties of both natural and non-natural textures.



By reconstructing representations from intermediate layers of the VGG network[6]. Observe that a deep convolutional neural network is capable of extracting image content from an arbitrary photograph and some appearance information from the well-known artwork. According to this observation, [2][1] build the content component of the newly stylised image by penalising the difference of high-level representations derived from content and stylised images, and further build the style component by matching Gram-based summary statistics of style and stylised images, which is derived from their proposed texture modelling technique. The details of their algorithm are as follows.

Given a content image I_c and a style image I_s, the algorithm tries to seek a stylised image I that minimise the following objective:

$$I^\star = \arg_I \alpha \min \mathcal{L}_c(I_c, I) + \beta\mathcal{L}_s(I_s, I) \tag{13}$$

where $\mathcal{L}_c$ compares the content representation of a given content image to that of the stylised image, and $\mathcal{L}_s$ compares the Gram-based style representation derived from a style image to that of the stylised image. α and β are used to balance the content component and style component in the stylised result. The content loss $\mathcal{L}_c$ is defined by the squared Euclidean distance between the feature representations F_l of the content image I_c in layer l and that of the stylised image which is initialised with a noise image:

$$\mathcal{L}_c = \sum_l \|\mathcal{F}^l(I_c) - \mathcal{F}^l(I)\|^2. \tag{14}$$

where $\mathcal{F}(I)$ denotes the set of VGG layers for computing the content loss. For the style loss $\mathcal{L}_s$, we exploit Gram-based visual texture modelling technique to model the style. Therefore the style loss is defined by the squared Euclidean distance between the Gram-based style representations of I_s and I:

$$\mathcal{L}_s = \sum_l \|\mathcal{G}(\mathcal{F}^l(I_c)) - \mathcal{G}(\mathcal{F}^l(I))\|^2. \tag{15}$$

where G is the aforementioned Gram matrix to encode the second order statistics of the set of filter responses. $\mathcal{F}(I)$ represents the set of VGG layers for calculating the style loss.

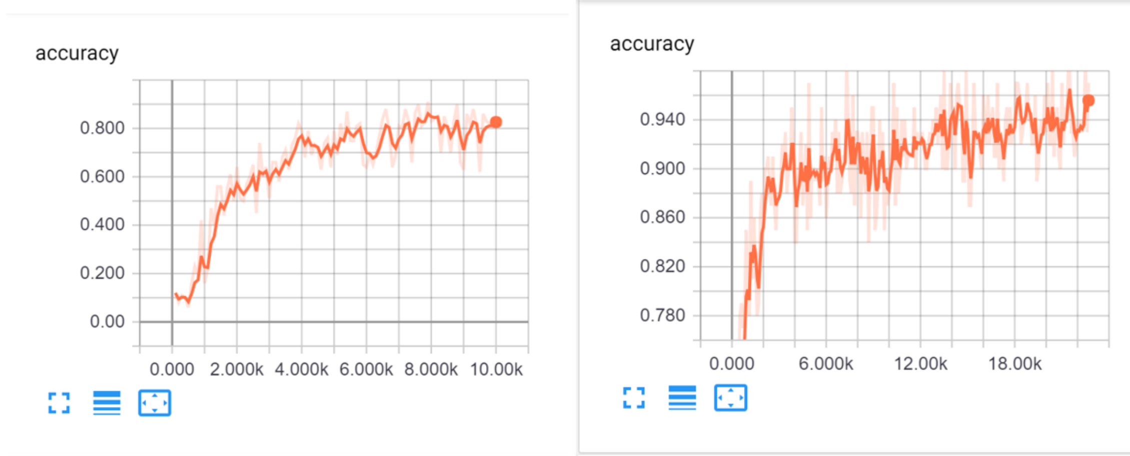
3 Initialization

The initialization step can be critical to the model's ultimate performance, and it requires the right method [7]. This is somehow easy to demonstrate. You could do some very basic simulation, using a VGG network architecture in PyTorch and initialize the weights with constant variables, and then see what is the range of the value of output variables. If the weights are initialized with zeros, what would happens in stead?

The empirical results in previous research and general practice indicates that initialization with constant variables generates very poor performance in terms of optimization. To analyze the reasons, we use a simple layer with two input variables, as in

$$relu(\alpha x_1 + \alpha x_2), \tag{16}$$

where we use α as the shared weights for this layer. Recall the update function for linear SVM and backpropagation, we know that the update direction is relates to the outputs and inputs in each layer, therefore if there are no difference between weights in the initialization, you may be fixed to one direction for all weights, therefore learning all similar filters.



(a) Training loss with random initialization and xavier initialization.

In addition to considering breaking symmetry, we should also initialize the weights with values either (i) not too small or (ii) not too large. An extreme value leads respectively to (i) slow learning or (ii) divergence.

From theoretical analysis as well as the empirical study on deep networks, we found that two cases will lead to inefficiency in training:
1): A too-large initialization leads to exploding gradients.
2): A too-small initialization leads to vanishing gradients.
The gradients of the cost with respect to the parameters are too small, leading to convergence of the cost before it has reached the minimum value.

A nice way to analyze the initialization behavior is to do some probabilistic problem calculation, suppose that all weights are initialized randomly, with n neurons in one layer, then the output y has a variance of

$$\begin{aligned} Var[y] =& Var(\sum_i w_i x_i) && (17) \\ =& \sum_i Var(w_i x_i) && (18) \\ =& \sum_i \mathbf{E}[x_i]^2 Var[w_i] + Var(w_i)Var(x_i) + \mathbf{E}[w_i]^2 Var[x_i] && (19) \\ =& \sum_i Var[x_i]Var[x_i] && (20) \\ =& \sum_i \frac{1}{12} \times Var[x_i] && (21) \\ =& \frac{n}{12} \times Var[x_i]. && (22) \end{aligned}$$

The goal is to derive a relationship between $Var(x^{l-1}) = Var(x^l)$. We want weight initialization methods where the variance of a single weight should scale inversely with width. A simple method

for initialization is as follows:

1), Xavier initialization (tanh, linear, sigmoid):

$$w_i \sim \mathcal{N}(0; \frac{1}{\sqrt{n}}), \quad | \quad Var(w_i) = \frac{1}{n} \tag{23}$$

2), He et. al. initialization (tanh, linear, sigmoid) [3]:

$$w_i \sim \mathcal{N}(0; \frac{1}{\sqrt{2n}}), \quad | \quad Var(w_i) = \frac{2}{n} \tag{24}$$

There are some recent papers on more theoretical analysis of initialization using kernel regime [8], the theorem may looks a bit of mathematically-heavy at first, and the proof takes some time to understand.

References

[1] Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *Advances in neural information processing systems*, pages 262–270, 2015.

[2] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[4] Alex Kendall and Roberto Cipolla. Geometric loss functions for camera pose regression with deep learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5974–5983, 2017.

[5] Quoc V Le, Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, and Andrew Y Ng. On optimization methods for deep learning. 2011.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[7] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[8] Blake Woodworth, Suriya Gunasekar, Jason Lee, Daniel Soudry, and Nathan Srebro. Kernel and deep regimes in overparametrized models. *arXiv preprint arXiv:1906.05827*, 2019.