Deep Learning Midterm Part II

April 13, 2020

Tejaishwarya Gagadam, tg1779

Please provide try to provide as concise solutions as possible, (long unnecessary answers won't be given any points)

1 Training a net from scratch [22 pts]

You are given the following neural net architecture:

$$\texttt{Linear}_1 \to f \to \texttt{Linear}_2 \to g$$

where $\texttt{Linear}_i(\boldsymbol{a}) = \boldsymbol{W}^{(i)}\boldsymbol{a} + \boldsymbol{b}^{(i)}$ is the i-th affine transformation (of a generic input activation $\boldsymbol{a}$), and f, g are element-wise nonlinear activation functions. When an input $\boldsymbol{x} \in \mathbb{R}^n$ is fed to the network, $\hat{\boldsymbol{y}} \in \mathbb{R}^K$ is obtained at the output.

1.1 Regression

We would like to perform *regression*. We choose $f(\cdot) = (\cdot)^+ = \texttt{ReLU}(\cdot)$ and g to be the identity function. To train this network, we choose a *mean squared error* (MSE) loss function $\ell_{\text{MSE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{2}(\hat{\boldsymbol{y}} - \boldsymbol{y})^2$, where $\boldsymbol{y}$ is the target output.

(a) Name/write the 5 programming steps you would take to train this model using *stochastic gradient descent* (SGD) on a single batch of data, writing **all** equations in mix of pseudocode and mathematical symbols. *Hint: the last step, called . . . , should look like* $\boldsymbol{W}^{(1)} \leftarrow \boldsymbol{W}^{(1)} + \ldots$, $\boldsymbol{b}^{(1)} \leftarrow \boldsymbol{b}^{(1)} + \ldots$, *and so on for all the net's parameters.* [10 pts]

- **Answer:**
  The 5 programming steps to train a model using SGD on a single batch of data in Pytorch would be:
  - To send the input data through the model output = model(input data)
  - Compute the loss using: criterion(output, original output data).
  - In reverse: Clear the gradient to make sure we do not accumulate the value: optimizer.zero_grad(). Back-propagation: loss.backward()
  - Step backwards: optimizer.step()
- **Step 1: Forward Pass**
  - The output to the first Linear layer can be given by
    $\boldsymbol{Z_1} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$
    where $\boldsymbol{x} \in n \times 1$, $\boldsymbol{W} \in m \times n$, $\boldsymbol{b} \in m \times 1$ and $\boldsymbol{Z_1} \in m \times 1$.
    Pseudo Code:
    $\boldsymbol{Z_1}[i]$ `= np.dot(`$\boldsymbol{W}[i], \boldsymbol{x}$`)` $+ \boldsymbol{b}[i]$
  - The non-linear activation used in this hidden layer is ReLU, which kills all the negative values.
    Mathematically represented by: $\boldsymbol{f(Z_1)} = max(0, \boldsymbol{Z_1})$
    We need to define the Activation function in Pseudo Code:
    ```
    def relu(X):
    return np.maximum(0,X)
    ```
  - The output to the second Linear layer can be given by
    $\boldsymbol{Z_2} = \boldsymbol{W_2 f(Z_1)} + \boldsymbol{b_2}$

where $\boldsymbol{f(Z_1)} \in m \times 1$, $\boldsymbol{W_2} \in k \times m$, $\boldsymbol{b} \in k \times 1$ and $\boldsymbol{Z_2} \in k \times 1$.
Pseudo Code:
$\boldsymbol{Z_2}[i]$ `=` `np.dot`$(\boldsymbol{W_2}[i], \boldsymbol{f(Z_1)}) + \boldsymbol{b_2}[i]$
$\boldsymbol{Z_2}[i]$ `=` `y_pred`

- The last activation layer is a simple identity function, which results in direct mapping. So, $\boldsymbol{Z_2}$ is directly mapped to the output $\hat{\boldsymbol{y}}$, so $\hat{\boldsymbol{y}} \in k \times 1$.

- **Step 2: Computing the Loss function**
  - We are using the Mean Squared Error: $\ell_{\text{MSE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{2}(\hat{\boldsymbol{y}} - \boldsymbol{y})^2$
  - Pseudo Code:
    `L = np.square(np.subtract(y_pred,y)).mean()`

- **Step 3: Backward Pass**
  - Here we need to compute gradients. We compute the partial derivatives of the Loss function with respect to the parameters $\boldsymbol{W_2}$, $\boldsymbol{b_2}$, $\boldsymbol{W_1}$ and $\boldsymbol{b_1}$. Using the chain rule, we back-propogate to the input.
  - Mathematically we get the following:

$$\frac{\partial L}{\partial W_2} = (\hat{y} - y)\frac{\partial \hat{y}}{\partial W_2} = f(Z_1)(\hat{y} - y)$$

$$\frac{\partial L}{\partial b_2} = (\hat{y} - y)$$

$$\frac{\partial L}{\partial W_1} = (\hat{y} - y)\frac{\partial (W_2 max(0, Z_1) + b_2)}{\partial W_1}$$

$$\frac{\partial L}{\partial b_1} = (\hat{y} - y)\frac{\partial (W_2 max(0, Z_1) + b_2)}{\partial b_1}$$

if $f(z_1) = 0$

$$\frac{\partial L}{\partial W_1} = 0$$

$$\frac{\partial L}{\partial b_1} = 0$$

else

$$\frac{\partial L}{\partial W_1} = W_2 x(\hat{y} - y)$$

$$\frac{\partial L}{\partial b_1} = W_2(\hat{y} - y)$$

  - Pseudo Code:
    `dL_dW2 = f(Z`$_1$`) *` $np.subtract$`(y_pred, y)`
    `dL_db2 = np.subtract`(y_pred, y)

`if f(`Z_1`) = 0 :`
`dL_dW1 = 0`
`dL_dW1 = 0`
`else:`
`dL_dW1 = np.dot(`$W2, X$`)*np.subtract(`y_pred, y`)`
`dL_dW1 = W2*np.subtract(`y_pred, y`)`

- **Step 4: Optimization**
  - Starting from a random initial point, taking a small batch of of data to perform stochastic gradient descent:
  - Pseudo code:
    `p = X.shape[1]+1`
    `w`$_0 = np.random.randn(p)$
    `step = 1e-6`
    $w_{i+1} = w_i$ `+ step*np.random.randn(p)`
    We measure the function and gradients at these steps and predict the amount the function should have changed based on the gradient.

(b) Let's say we'd like to predict a positive quantity. How would you or would you not change to reflect this new task? Explain your choice and what advantages/disadvantages it has. [2 pts]

- **Answer:** If we want to predict a positive output our final layer should have a ReLU function instead of an identity function. This way we can kill all the non-zero values and produce positive outputs. However, there is a chance that the affine transformations could yield negative output which are eliminated by the ReLU activation function - that is many such inut points could be mapped to 0.

1.2 Classification

We would like to perform binary *classification* (*i.e.* $K = 1, y \in \{0, 1\}$) by using a "binary network", so we set both $f, g = \sigma$, the logistic sigmoid function $\sigma(z) \doteq (1 + \exp(-z))^{-1}$.

(a) Should you want to train this network, what do you need to change in the equations of section 1.1 item (a), assuming we are using the same $\ell_{\text{MSE}}(\cdot)$. [2 pts]

- **Answer:** We need to change the equations of our activation layers, that were previously ReLU and an identity function, respectively, to two sigmoid functions.
- $\boldsymbol{f(Z_1)} = 1/(1 + np.exp(-\boldsymbol{z_1}))$
- $\boldsymbol{g(Z_2)} = 1/(1 + np.exp(-\boldsymbol{z_2}))$

(b) Now you think you can do a better job by using a *binary cross-entropy* (BCE) loss function $\ell_{\text{BCE}}(\hat{y}, y) = -\big[y \log(\hat{y}) + (1-y)\log(1-\hat{y})\big]$. What do you have to modify from the previous setup? Why could this improve over MSE? [4 pts]

- **Answer:**
- We need to change the loss function to Binary Cross Entropy loss.
- We need to calculate the partial derivatives of the new loss function with respect to $\boldsymbol{W_2}$, $\boldsymbol{b_2}$, $\boldsymbol{W_1}$ and $\boldsymbol{b_1}$.
- The Binary Cross Entropy Loss works better than MSE for Classification problems because we are aiming to get specific output, that is categorical classes as outputs, unlike Regression where we are predicting a number. This way we can maximize the likelihood of classifying the input data correctly.

$$\frac{\partial L}{\partial W_2} = -\left[\frac{y}{\hat{y}} + \frac{(1-y)}{(1-\hat{y})}\right] \frac{\partial L}{\partial W_2}$$

$$\frac{\partial L}{\partial b_2} = -\left[\frac{y}{\hat{y}} + \frac{(1-y)}{(1-\hat{y})}\right] \frac{\partial L}{\partial b_2}$$

$$\frac{\partial L}{\partial W_1} = -\left[\frac{y}{\hat{y}} + \frac{(1-y)}{(1-\hat{y})}\right] \frac{\partial L}{\partial W_1}$$

$$\frac{\partial L}{\partial b_1} = -\left[\frac{y}{\hat{y}} + \frac{(1-y)}{(1-\hat{y})}\right] \frac{\partial L}{\partial b_1}$$

(c) Things are getting better. You realize that not all intermediate hidden activations need to be binary (or soft version of binary). You decide to use $f(\cdot) = (\cdot)^+$ but keep g as σ. Explain why this choice of f can be beneficial for training a (deeper) network. [2 pts]

- **Answer:**
  - ReLUs are much more simpler to compute, for both forward and backward passes as compared to a Sigmoid function which has exponential terms in it. Using ReLU in the hidden layers, instead of Signmoid significantly reduce both training and evaluation times.
  - Sigmoid activations are prone to saturation. Since the Sigmoid function has a narrow non-zero window, if it reaches the left/right extreme - it will reduce the derivative to 0 when we backward pass through it. However, ReLUs only saturate when the input is less than 0 (this saturation can also be eliminated by using leaky ReLUs). For Deep Neural Networks, saturation is bad for learning, and since Sigmoids saturate and kill gradients, ReLUs are preferred in the hidden layers.

(d) You deploy the trained network and observe that the output is very unstable. You want now to generate some "hard inputs" $\tilde{\boldsymbol{x}} = \boldsymbol{x} + \delta\boldsymbol{x}$ and train on them. How would you pick / compute such $\delta\boldsymbol{x}$? [3 pts]

- **Answer:**
  - An unstable output would mean over-fitting, due to high variance - that is, if you perturb a single data point, the algorithm changes by a lot.
  - We can introduce noise $\delta\boldsymbol{x}$ with $\boldsymbol{x}$, to reduce overfitting. We can add guassian random noise to our data, with 0 mean and 0.1 standard deviation (or any other small value). The δ would be the measure of standard deviation that we give the noise.

2 Variational autoencoder [23 pts]

The encoder of a variational autoencoder maps an input $\boldsymbol{x} \in \mathbb{R}^n$ to a distribution $q(\boldsymbol{z}) = \mathcal{N}\big(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)\big)$, $\boldsymbol{z}, \boldsymbol{\mu}, \boldsymbol{\sigma}^2 \in \mathbb{R}^d$. A deterministic decoder can be used to can map a $\boldsymbol{z} \sim q(\boldsymbol{z})$ to $\hat{\boldsymbol{x}}$. To train both encoder and decoder, we can use a per-sample loss that has two additive terms: a squared reconstruction error and a relative entropy between $q(\boldsymbol{z})$ and a *prior* distribution $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{0}, \mathbb{I}_d)$, where $\mathbb{I}_d$ is the d-dimensional identity matrix. The relative entropy $D_{\mathrm{KL}}\big(q(\boldsymbol{z}) \,||\, p(\boldsymbol{z})\big)$ between distributions $q(\boldsymbol{z})$ and $p(\boldsymbol{z})$ is given by:

$$D_{\mathrm{KL}}\big(q(\boldsymbol{z}) \,||\, p(\boldsymbol{z})\big) \doteq -\mathbb{E}_{z\sim q(\boldsymbol{z})}\left[\log \frac{p(\boldsymbol{z})}{q(\boldsymbol{z})}\right]$$

where $q(\boldsymbol{z})$ is given by:

$$q(\boldsymbol{z}) = \frac{1}{(2\pi)^{d/2}\prod_i \sigma_i} \exp\left(-\frac{1}{2}\sum_{i=1}^{d}\frac{(z_i-\mu_i)^2}{\sigma_i^2}\right)$$

(a) Analytically compute (give a closed form expression for) the relative entropy $D_{\mathrm{KL}}\big(q(\boldsymbol{z}) \,||\, p(\boldsymbol{z})\big)$. [5 pts]

- **Answer:**
  let

$$p(\boldsymbol{z}) = \frac{1}{(2\pi)^{d/2}\prod_i \sigma_{1i}} \exp\left(-\frac{1}{2}\sum_{i=1}^{d}\frac{(z_i-\mu_{1i})^2}{\sigma_{1i}^2}\right)$$

  and,

$$q(\boldsymbol{z}) = \frac{1}{(2\pi)^{d/2}\prod_i \sigma_{2i}} \exp\left(-\frac{1}{2}\sum_{i=1}^{d}\frac{(z_i-\mu_{2i})^2}{\sigma_{2i}^2}\right)$$

  we know that,

$$D_{\mathrm{KL}}\big(q(\boldsymbol{z}) \,||\, p(\boldsymbol{z})\big) = -\int q(\boldsymbol{z}) \log \frac{p(\boldsymbol{z})}{q(\boldsymbol{z})} dz$$

$$= \int \left[\log q(\boldsymbol{z}) - \log p(\boldsymbol{z})\right] q(\boldsymbol{z}) dz$$

$$= \int [\log\left(\frac{1}{(2\pi)^{d/2}\prod_i \sigma_{2i}} \exp\left(-\frac{1}{2}\sum_{i=1}^{d}\frac{(z_i-\mu_{2i})^2}{\sigma_{2i}^2}\right)\right)$$

$$-\log\left(\frac{1}{(2\pi)^{d/2}\prod_i \sigma_{1i}} \exp\left(-\frac{1}{2}\sum_{i=1}^{d}\frac{(z_i-\mu_{1i})^2}{\sigma_{1i}^2}\right)\right)] q(\boldsymbol{z}) dz$$

$$= \int [-\frac{d}{2}\log(2\pi) + \sum_{i=1}^{d}\log\sigma_{2i} - \frac{1}{2}\sum_{i=1}^{d}\frac{(z_i-\mu_{2i})^2}{\sigma_{2i}^2}$$

$$+\frac{d}{2}\log(2\pi) - \sum_{i=1}^{d}\log\sigma_{1i} + \frac{1}{2}\sum_{i=1}^{d}\frac{(z_i-\mu_{1i})^2}{\sigma_{1i}^2}]q(\boldsymbol{z})dz$$

$$= \sum_{i=1}^{d}\log\sigma_{2i} - \log\sigma_{1i} - \frac{1}{2}\mathbb{E}\left[\sum_{i=1}^{d}\frac{(z_i-\mu_{2i})^2}{\sigma_{2i}^2} - \frac{(z_i-\mu_{1i})^2}{\sigma_{1i}^2}\right]$$

we know that $\sigma_{1i} = \mathbb{I}_d, \mu_{1i} = \mathbf{0}$

$$= \sum_{i=1}^{d}\log\sigma_{2i} - \frac{1}{2}\sum_{i=1}^{d}\mathbb{E}\left[\frac{(z_i-\mu_{2i})^2}{\sigma_{2i}^2} - {z_i}^2\right]$$

$$= \sum_{i=1}^{d}\log\sigma_{2i} - \frac{1}{2}\sum_{i=1}^{d}\mathbb{E}\left[\frac{{z_i}^2 - 2z_i\mu_{2i} + \mu_{2i}^2 - {z_i}^2\sigma_{2i}^2}{\sigma_{2i}^2}\right]$$

$$= \sum_{i=1}^{d}\log\sigma_{2i} - \sum_{i=1}^{d}\frac{1}{2\sigma_{2i}^2}\left[\mathbb{E}[{z_i}^2] - 2\mu_{2i}\mathbb{E}[z_i] + \mu_{2i}^2 - \sigma_{2i}^2\mathbb{E}[{z_i}^2]\right]$$

$$= \sum_{i=1}^{d}\log\sigma_{2i} - \sum_{i=1}^{d}\frac{1}{2\sigma_{2i}^2}\left[\mathbb{E}[{z_i}^2] - \mu_{2i}^2 - \sigma_{2i}^2\mathbb{E}[{z_i}^2]\right]$$

$$= \sum_{i=1}^{d}\log\sigma_{2i} - \sum_{i=1}^{d}\frac{1}{2\sigma_{2i}^2}\left[(1-\sigma_{2i}^2)\mathbb{E}[{z_i}^2] - \mu_{2i}^2\right]$$

$$= \sum_{i=1}^{d}\log\sigma_{2i} - \frac{1}{2\sigma_{2i}^2}\left[(1-\sigma_{2i}^2)\mathbb{E}[{z_i}^2] - \mu_{2i}^2\right]$$

$$= \sum_{i=1}^{d}\log\sigma_{2i} - \frac{1}{2\sigma_{2i}^2}\left[(1-\sigma_{2i}^2)\mathbb{E}[{z_i}^2] - (\mathbb{E}[{z_i}^2] - \sigma_{2i}^2)\right]$$

$$= \sum_{i=1}^{d}\log\sigma_{2i} - \frac{1}{2\sigma_{2i}^2}\left[\mathbb{E}[{z_i}^2] - \sigma_{2i}^2\mathbb{E}[{z_i}^2] - \mathbb{E}[{z_i}^2] + \sigma_{2i}^2)\right]$$

$$= \sum_{i=1}^{d}\log\sigma_{2i} - \frac{1}{2\sigma_{2i}^2}\left[\sigma_{2i}^2(1-\mathbb{E}[{z_i}^2])\right]$$

$$= \sum_{i=1}^{d}\log\sigma_{2i} - \frac{1}{2}\left[(1-\mathbb{E}[{z_i}^2])\right]$$

(b) Looking at this analytical solution, describe (using physics and intuition) two properties the encoder's $q(\cdot)$ distribution will have after minimising this term. [5 pts]

- **Answer:**
  The relative entropy calculated will give us the distance between the two distributions. Each z_i value can be visualized as a circle/a bubble in a d-dimensional space. The centre of the bubble is given by it's mean μ_i and the radius is given by the σ_i. When you minimize the analytical expression computed in part (a) - we can see that the variance is limited to 1, so the radii of these bubbles will be approximately 1.

(c) Now consider the full training loss, comprising the squared reconstruction and the relative entropy term. How do these two terms interact? [3 pts]

- **Answer:**
  When we consider the full training loss we have the squared reconstruction loss as well as the regularization term (which is the KL-Divergence) times β.

  $$l(\boldsymbol{x}, \hat{\boldsymbol{x}}) = l_{reconstruction} + \beta l_{KL}$$

  The reconstruction loss tries to eliminate any overlaps between the circles/bubbles - by increasing the distance between them. Overlaps are not favorable because it creates ambiguity in the mapping space. Since an overlap could map that z_i to multiple original inputs. When this reconstruction loss term pushes the bubbles away from each other, there is a chance it might explode if the distance between the them is too high.

  The penalty term restricts this explosion. $\mathbb{E}[{z_i}^2]$) minimizes the distance between the z_i circles, that keeps them close (without overlap) and prevents explosion. At the same time, it ensures that the radius of each circle is close to 1 (variance).

(d) *Support* of a distribution refers to the region that has non-zero probability (density). When we reach a local minimum of this loss, we look $q(\boldsymbol{z})$ for each $\boldsymbol{x}$. Are we *guaranteed* that the support of the q's don't overlap? If so, why are we guaranteed? If not, what architectural choices will help us achieve this? [5 pts]

- **Answer:**
  There is a possibility that the support's of $q's$ overlap. Which is why we use the reconstruction loss to ensure that these regions are far from each other. The $q's$ Gaussian distributions which appear to be "blobs" are architecturally more favorable to prevent overlaps. The reconstruction loss term and the penalty term make sure that overlaps don't occur, yet keeping the circles close, and each with a radius close to 1.

(e) Suppose you only want to use one of K possible latent codes, rather than any continuous vector. Describe one way to accomplish this. Which difficulties (regarding gradient-based optimisation) arise from this approach, if any? (Hint: most solutions to this will involve some gradient-related difficulty) [5 pts]

- **Answer:** A continuous latent variable describes a location in a subspace or manifold, which can be used for dimensionality reduction, where as discrete latent variables are useful for clustering data or for partition. If we choose to use only K possible latent codes, we could do this using a Sparse-Coding model.