MIT 6.S191: Introduction to Deep Learning (2023)

Swarnim Jain

May 2023

Lecture 1: Intro to Deep Learning

ReLU is extremely popular nowadays because it is easy to compute and has a constant derivative.

Lecture 2: Deep Sequence Modeling

- In an RNN, at each time step, along with the output of that step, the state (memory), h, is also being passed as the output data. This state is updated at each time step.
- The weights which are applied to the input data x_t and h_{t-1} remain constant over all time steps.
- With each time step, the hidden state is being updated by applying a function f_W (activation function with weights) to the outputs of the previous state, i.e., x_t and h_{t-1}.
- There are two sets of weights, one for hidden states and one for input data.
- The final output vector, $\hat{y_t}$, is attained by applying a separate weight matrix to h_t which was computed in the above step.
- $\hat{y_t}$ is not passed on to the next time step, however h_t is. $\hat{y_t}$ is used for computing the loss at that particular time step, which is then summed up over all time steps to get the total loss for that pass of the NN.
- The weight matrices are W_{hh}, W_{xh}, and W_{yh} and are initialized randomly (or to zero).
- Embeddings for the inputs of an RNN are attained by training the vocabulary on a neural network to learn relationships.
- The vanishing gradient occurs during backpropagation when the weights matrix is multiplied to itself multiple times, leading to values ¡ 1 going to 0. Some solutions to this are the following:
  - Initialize the weights matrix to the identity matrix and biases to zero.
  - Use the ReLU activation function as for all $x > 0$, the derivative is 1.

Lecture 3: Deep Computer Vision

- In a normal feedforward fully connected neural network, the number of connections for an image as small as 100 x 100 pixels would be in the millions, which is computationally infeasible for larger image sizes. To solve this, convolution layers are applied to the image, thus the CNN exists.
- The convolution operation involves summing up the results of element-wise multiplication of the filter with the specific part of the image.
- Convolutions serve the purpose of extracting different features from the image. The filter matrices are also learnt as weights of the neural network.
- The three main steps for a classification CNN are (1) convolution - applying filters to generate the map of features, (2) non-linear - adding non-linearity to the input data through an activation function such as ReLU to capture more patterns, and (3) (max) pooling - downscaling the images (reducing their sizes) and selecting key features while discarding irrelevant information so that the model can scale well for larger images, increases the receptive field, as well as making the model less sensitive to translation as the pooling selects the maximum value within a pooling region.
- The receptive field is defined as the region of the input image that affects the activation of that neuron. Basically, the larger the patch of the image that is responsible for determining the result of that neuron, the larger the receptive field. As we go deeper into layers, the receptive field increases as the information from patches accumulates and the neuron is able to capture more global features.
- For each neuron in the hidden layer, a patch of the image is taken to which a convolution is applied (instead of dot product with weights), then a bias is applied to that, and finally is activated with a non-linear function.
- We may want to capture multiple features from a single image, so in this case, the result of a convolution operation would be a “volume” rather than an a 2D area. The depth of the volume is the number of features.

- ReLU is basically squashing all negative values (pixel values) to zero and keeping non-negative values unchanged as it is the identity function for values greater than zero.
- When we decrease the dimensionality of our features, we're essentially increasing the dimensionality of the filters. This is because that same filter on the smaller image is now capturing a larger receptive field and thus more global features. This is what pooling does.
- A common technique is max pooling (self-explanatory) over patches.
- The network learns features over a hierarchical progression as convolutional layers are stacked on top of each other.
- The structure of a CNN is: input → feature learning (conv + relu, pooling) → classification (flatten, fully connected, softmax). Through the conv and pool layers, the network learns high-level features and feeds it forward into a 1-dimensional output that is then passed through a fully connected layer from which we get probabilities (softmax).
- The part after feature learning can be changed depending on your use case, instead of classification.

Lecture 4: Deep Generative Modeling

- Generative modeling can have two forms: density estimation and sample generation. In density estimation, a probability distribution is generated to identify patterns in the underlying data set. In sample generation, this distribution is used to create new samples which are similar to the original but new.
- They are able to uncover underlying data and encode it in an efficient way.
- Latent variables are the hidden explanatory features of a dataset that cannot directly be visualized and is not explicitly present in the input data. The aim is to learn latent variables only from observed data.
- "Encoder" learns mapping from the data x to a lower-dimensional latent space z. Low-dimensional latent space is efficient (a vector). This model is trained by trying to reconstruct the original data to capture as much information as possible. This is all unsupervised learning.
- $L = ||x - \hat{x}||^2$
- Autoencoding is a form of compression.
- In traditional autoencoders, the latent layer z is deterministic, i.e., once it has been trained, it will return the same output on the same input. However, to have the model generate new data each time, we have variational autoencoders (VAEs).
- In this method, each latent variable is now associated with a mean and standard deviation that parameterizes the probability distribution of that variable.
- The encoder computes $q_\phi(z|x)$ and the decoder computes $p_\theta(x|z)$. Each process is associated with a different set of weights (θ, ϕ).
- $L(\phi, \theta, x)$ = (reconstruction loss) + (regularization term)
- Our aim while training the network is optimizing the network weights θ, ϕ that define weights of encoder and decoder.
- In regularization, an encoding of latent variables is compared with a "prior," i.e., a prior distribution that we compare our present encoding with. We do not want it to be too far from this distribution, and hence a term is added that represents the "distance" between both representations.
- A common prior is the Normal (Gaussian) distribution. This is used when we want our data to be relatively smooth and clustered around a central area. This prevents the model from cheating by simply memorizing the input data.
- A few things we want to achieve from regularization: continuity (similar content after decoding for points nearby) and completeness (meaningful decoded content for any point, hence should be well-defined).
- With the normal prior, all means and standard deviations of the latent variables are regularized to 0 and 1, respectively, so that the learned distribution of variables overlaps.
- Sampling random data from the distribution makes backprop not possible by introducing non-differentiability. To solve this, the mean and std are taken as fixed, and a fixed random variable is added from a fixed normal distribution that scales. In this case, the gradients can be computed for the network weights (latent variables).
- Ideally, we want our latent variables to be uncorrelated with each other. This is called disentanglement.
- β-VAEs are a type of VAE where an extra variable β is introduced which is multiplied with the regularization term in the loss function. A higher β ($\beta > 1$) encourages efficient latent encoding, i.e., disentanglement.
- We can interpret what the hidden latent variables represent by perturbing their values and seeing the result. This can be done with multiple latent variables together as well, to see the change in result on variation.

- In a generative adversarial network (GAN), the generator converts noise into an imitation of the data that tries to trick the discriminator. On the other hand, the discriminator tries to identify the real image from the fakes created by the generator.
- The loss functions of each of these NNs are at odds with each other.

  Loss function of the generator:

$$\arg\min_G \mathbb{E}_{\mathbf{z},\mathbf{x}}[\log D(G(\mathbf{z})) + \log(1 - D(\mathbf{x}))]$$

  Loss function of the discriminator:

$$\arg\max_D \mathbb{E}_{\mathbf{z},\mathbf{x}}[\underbrace{\log D(G(\mathbf{z}))}_{\text{Fake}} + \underbrace{\log(1 - D(\mathbf{x}))}_{\text{Real}}]$$

  Overall loss function:

$$\arg\min_G \max_D \mathbb{E}_{\mathbf{z},\mathbf{x}}[\log D(G(\mathbf{z})) + \log(1 - D(\mathbf{x}))]$$

- After the training process, the generator alone is used to create new instances. GANs learn to convert noise to some output in our target data space. We can now traverse through trajectories in the noise space that then map to traversals in the target data space. Now we can figure out all the steps required between an initial point and a target state. GANs are distribution transformers.

Lecture 5: Uncertainty and Bias

- Selection bias: the proportion of data in the dataset does not reflect the real world.
- Evaluation bias: originally, these models were not evaluated on subgroups.
- Class imbalance: some classes are more represented than others.
- Some ways to tackle class imbalance are:
  - Sample reweighting: sample more data points from underrepresented classes.
  - Loss reweighting: mistakes on underrepresented classes contribute more.
  - Batch selection: choose randomly from classes so that every batch has an equal number of points per class.
- It is important to capture variations within the same class so that we don't overgeneralize. This can be done by looking at the latent variables, which capture the elements of data.
- This needs to be done when there is a lot of variability of features in the same class and they are not represented well. The bias is in the latent features.
- To solve this, we use VAEs to learn the latent features of the dataset. Once we get the probability densities for these latent variables, we can oversample from the sparser areas and undersample for the denser areas of the dataset.

$$\underbrace{\hat{Q}(\boldsymbol{z}|X)}_{\text{Estimated joint distribution}} \propto \prod_i \underbrace{\hat{Q}_i(\boldsymbol{z}_i|X)}_{\substack{\text{Histogram for every latent variable } \mathbf{z}_i \\ \text{Independence to approximate}}}$$

$$\underbrace{W(z(x)|X)}_{\substack{\text{Probability of selecting datapoint}}} \propto \prod_i \frac{1}{\underbrace{\hat{Q}_i(z_i(x)|X)}_{\substack{\text{Histogram for every latent variable } \mathbf{z}_i}} + \underbrace{\alpha}_{\substack{\text{Debiasing parameter}}}}$$

- Uncertainty measure gives an estimate of the confidence of the prediction.
- There are two types of uncertainty: aleatoric uncertainty and epistemic uncertainty.
- Aleatoric uncertainty (data uncertainty):
  - It is irreducible and can be directly learned from the data.
  - We're asking about the uncertainty in the data itself, which can't be reduced even with a perfect model.
  - To estimate this type of uncertainty, we want (the model) to learn the variance corresponding to the input along with the prediction (output).
  - Higher variance $\rightarrow$ higher uncertainty.
  - $f_\theta(x) \rightarrow \hat{y}, \sigma^2$, i.e., variation is not constant and is a function of not only the distribution but of each input x.
  - Mean squared error learns parameters of a multivariate Gaussian with **constant variance**, thus we change the loss function to negative log likelihood (NLL), which accounts for non-constant variances.

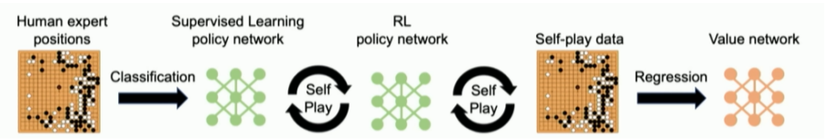
  - For example, in semantic segmentation, corners and boundaries in the image have high data or aleatoric uncertainty.

- Epistemic uncertainty (model uncertainty):
  - It is reducible by adding data and cannot be learned from the data.
  - We're asking if the model is uncertain about a prediction.
  - To estimate the uncertainty in such cases, we use ensembling, i.e., training multiple models using the same hyperparameters and data. Then, the model is fit on an input for each of the above models, and finally, the output of these models is used to calculate uncertainty (variance) and mean.
  - We can add stochasticity by adding dropout layers after every single layer in our model (which we used in regularization to prevent overfitting) instead of ensembling (high computational cost). This saves memory and compute.
  - We will keep dropout layers enabled in test time to estimate epistemic uncertainty (normally, we don't). To find the uncertainty, we calculate variance for the outputs of multiple forwards passes (with dropout enabled in test time).
  - An alternative way is using generative models such as VAEs. We take an input and it is then an output is then asked to be reconstructed to be similar to the input. If the model has not seen similar input images or is not confident, the output image will be different, and thus the reconstruction loss will be high.
  - However, this method requires training a decoder, which can be computationally expensive.

Lecture 6: Deep Reinforcement Learning

- An agent is a being that can take actions. Our algorithm is the agent.
- The environment is the world in which the agent operates.
- An action is a move that an agent can make in an environment.
- The set of all actions that it can take in an environment is called the action space A. It need not be a finite space.
- Observations are what the agent observes in the environment after taking actions.
- A state is a situation that the agent perceives at a particular moment.
- Reward is the feedback that measures the success or failure of an agent's particular action.
- The discounted sum of rewards from time t (till infinity) is given by $R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$
- The Q-function gives the expected total future rewards of a particular action (a_t) when it is at a particular state (s_t).
- $Q(s_t, a_t) = E[R_t | s_t, a_t]$
- The agent needs a policy $\pi(s)$ to infer the best strategy to take at its state s. The policy function is a function that determines the best action given a state.
- Hypothetically, if we had an accurate Q-function, we could find the best strategy by simply taking the action that gives the maximum future rewards at any given state. However, in reality, we don't have access to such a function and must find one.
- There are broadly two types of RL algorithms: value learning and policy learning.
- Value learning is when we learn to estimate the Q-function and then choose the action that maximizes the Q-function over all actions in that state.
- Policy learning is when we learn to choose which action to take when at a particular state instead of finding the Q-function.
- In value learning, our target return is $r + \gamma \max_{a'} Q(s', a')$.
- Our predicted return is $Q(s, a)$.
- Thus, the loss for our deep Q network is $L = E\left[\|(r + \gamma \max_{a'} Q(s', a')) - Q(s, a)\|^2\right]$ (Q-loss).
- After an action is taken, we can see the ground truth, which is the expected return.
- Our neural network returns a list of Q-functions for different actions in that state, and our corresponding policy function is $\pi(s) = \arg\max_a Q(s, a)$.
- This action is then sent to the environment, and we receive the next state.
- In Atari Breakout, the reward signal is very sparse, as the rewards for an action may become visible after multiple time steps.
- The downsides to Q-learning are:

  - The action space needs to be discrete (and small), i.e., not continuous action spaces.
  - The policy is deterministic in the sense that the action is chosen which has the maximum Q-value, and there is no stochasticity. This prevents the agent from exploring new potential actions.
- In policy learning, the network returns actions directly in the form of probabilities of a desirable value by taking that particular action. It skips the step of calculating the Q-function and hence Q-value for each action. It doesn't care about the exact value but rather which action gives the best value.
- It takes each action a_i with probability $P(a_i|s)$, and hence there is stochasticity in the model, which solves a drawback from value learning.
- Another advantage of value learning is that the network returns a probability distribution which need not be discrete and can be continuous as well.
- Because the action space is now continuous, it isn't possible to predict for every single action, thus we can parameterize our action space by a distribution (e.g., Gaussian).
- Because the distribution is a probability distribution, its integral must sum to 1.
- For a policy gradient neural network, $L = -\log P(a_t|s_t)R_t$.
- $\log P(a_t|s_t)$ is called the log-likelihood of action, and R_t is called the reward.
- Gradient descent update: $w' = w + \nabla \log P(a_t|s_t)R_t = w - \nabla \text{loss}$.
- AlphaGo's approach for go was as follows: (1) initial training human data, (2) self-play and reinforcement learning $\rightarrow$ super-human performance, (3) "intuition" about the board state.



- They later also tried to directly dive into self-play without looking at human games, and this technique outperformed previous methods, including the ones bootstrapped with human games.

Lecture 7: Limitations and New Frontiers

- Universal Approximation Theorem: A feedforward network with a single layer is sufficient to approximate, to an arbitrary precision, any continuous function.
- The caveats of this theorem are that the model may not generalize well (i.e., it may overfit) and that the number of hidden units may be infeasibly large.
- Neural networks are excellent function approximators in regions where we have training data.
- Similar to CNNs, Graph Convolutional Networks (GCNs) have a kernel matrix which goes to different parts of the graph and looks at what the neighbors of that node are and how it's connected to adjacent nodes.
- The kernel is used to extract features from the local neighborhood to then capture relevant information within the structure.
- The model learns weights of the edges between a node and its neighbors that define that local connectivity. The kernel is applied to each node in the graph.
- This local information is then collected together and aggregated to update the weights. The most common way of aggregation is to take the mean or sum of the output vectors from the kernel matrix, but there are other aggregation methods as well.
- The process of applying the kernel matrix and aggregating the results is repeated for multiple layers in the network, allowing the model to capture higher-order dependencies in the graph structure. This is analogous to CNNs.
- Graph neural networks can be extended to 3D data such as point clouds (unordered sets with spatial dependence between points, typically from LiDAR and depth cameras) by capturing local geometric features of point clouds while maintaining order invariance.
- The limitations of VAEs and GANs are as follows:
  - Mode collapse: these models can collapse into a phase where they produce a lot of outputs that are very similar, sometimes called regression to the average value (most common value, i.e., mode).
  - Generating OOD: they struggle to generate original samples that are more diverse and less similar to the training data.