Notes on INFO8010 - Deep Learning

Yuanzhe (Roger) Li

2020-04

Contents

This document contains notes and additional readings for self-study.

Lecture 4: Computer Vision

- Misc.
  - On cross-entropy
    - (Wiki page) Cross Entropy
    - A Gentle Introduction to Cross-Entropy for Machine Learning
- Classification
  - Image augmentation
  - Use pre-trained models for fine tuning and transfer learning
  - Large networks trained for classification are heavily re-used for object detection and semantic segmentation tasks.
- Object Detection
  - YOLO for object detection
    - EPFL EE-559, 8-3: Object Detection
  - R-CNN
    - Dive into Deep Learning - 13.8. Region-based CNNs (R-CNNs)
  - Takeaways
    - One-stage detectors (YOLO, SSD, RetinaNet, etc) are *fast* for inference *not as accurate.*
    - Two-stage detectors (Fast R-CNN, Faster R-CNN, R-FCN, Light head R-CNN, etc) are usually *slower* but are *more accurate.*
    - Both depend on engineering decisions.
- Segmentation
  - Task: partitioning an image into regions of different semantic categories at *pixel level.*
  - Fully convolutional network(FCN) and transposed convolution
    - CS231n, Lecture 11, 2018.
  - Mask R-CNN
    - Object detection combined with mask prediction enables instance segmentation.
    - Dive into Deep Learning - 13.8.4 Mask R-CNN

Lecture 5: Training Neural Networks

- Gradient descent

  - GD, SGD, mini-batch SGD
  - Rely on assumptions on 1) the magnitude of the local curvature to set the step size, and 2) *isotropy* in gradient so the step size makes sense in all directions
- Wolfe conditions ensures that both the loss function decreases sufficiently and the slope reduces sufficiently. However, line search will be too expensive for DL, and might lead to local minimum / overfitted solution.
- Momentem
  - Use momentum to add inertia in the choice of the step direction
  - Nesterov momentem
- Adaptive learning rate: without the assumption of istropic gradient
  - Per-parameter methods: AdaGrad, RMSProp, Adam
  - Scheduling
- Some additional reading on optimization: (Sebastian Ruder) An overview of gradient descent optimization algorithms
- Initialization
  - Principles
    * Break symmetry
    * Control variance of activation across layers during forward and backward pass
  - Xavier initialization
- Normalization
  - Batch normalization
  - Layer normalization

Lecture 6: Recurrent Neural Networks

- Some of the notes were added by reviewing EPFL EE-559, 12.1 – Recurrent Neural Networks, 12.2 LSTM and GRU
- Types of tasks
  - Classification: sequence to classes
  - Synthesis: real values to sequence
  - Translation: sequence to sequence
- Temporal convolutions
- Recurent neural networks
  - Structure
    * maintain a recurrent state updated at each time step (a function of state the previous step, input of the current step, and weights), $\mathbf{h}_t = \phi(\mathbf{x}_t, \mathbf{h}_{t-1}; \theta)$. So if $\mathbf{x} \in \mathbb{R}^D and h \in \mathbb{R}^Q$, then $\phi : \mathbb{R}^D \times \mathbb{R}^Q \to \mathbb{R}^Q$
    * Predictions can be computed at any step from the recurrent state $y_t = \psi(\mathbf{h}_t; \theta)$.

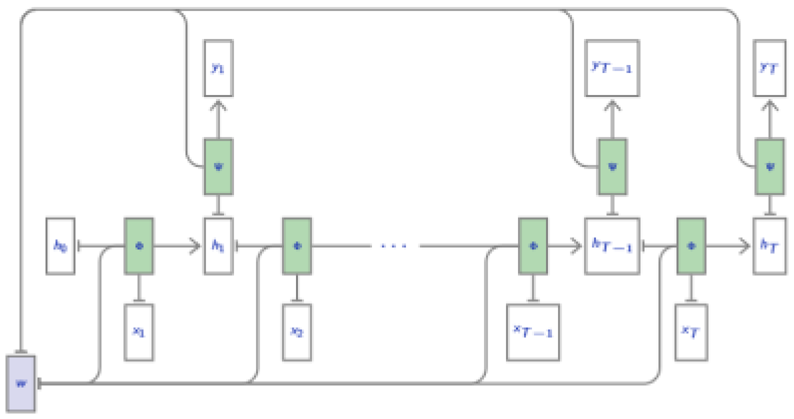
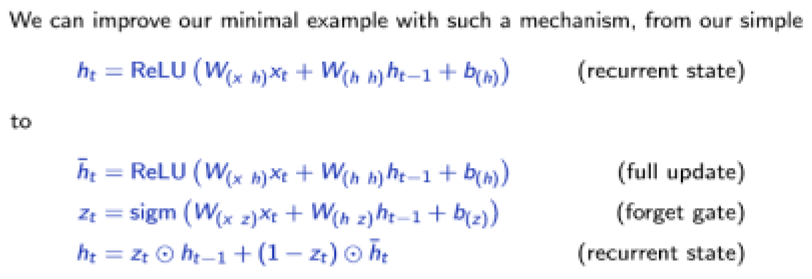


Figure 1: RNN

    * Elman netoworks apply non-linear activation functions as ϕ and ψ

$$h_t = \text{ReLU}(W_{x(\times h)}\mathbf{X}_t + W_{(h\times h)}h_{t-1} + b_{(h)}), \quad y_T = W_{(h\times y)}h_T + b_{(y)}$$

- **Stacked RNN**
  - Since RNNs can be viewed as layers producing sequences of activations, and can be stacked
- **Bidirectional RNNs**. RNNs can be made *bidirectional*. run the same single direction RNN twice from both end and concatenate the states.
- Gating
  - Similar to the skip connections in ResNet, RNN cells can inlude pass-throughs so recurrent state does not go repeatedly through a squashing non-linearity.
  - *forget gate*: current state update be a per-component weighted average of its previous value and a full update, with the weighting depending on input and the previous state.



We can improve our minimal example with such a mechanism, from our simple

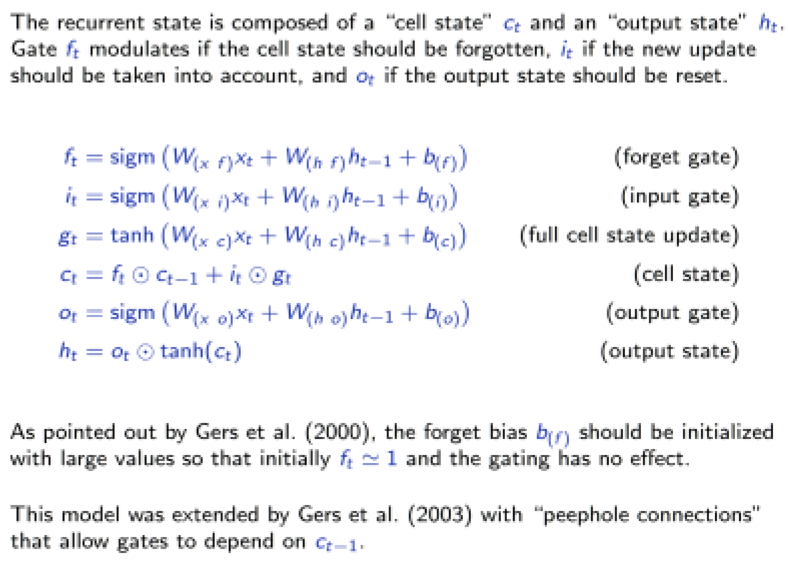
$$h_t = \mathrm{ReLU}\left(W_{(x\ h)}x_t + W_{(h\ h)}h_{t-1} + b_{(h)}\right) \qquad \text{(recurrent state)}$$

to

$$\begin{aligned} \bar{h}_t &= \mathrm{ReLU}\left(W_{(x\ h)}x_t + W_{(h\ h)}h_{t-1} + b_{(h)}\right) && \text{(full update)} \\ z_t &= \mathrm{sigm}\left(W_{(x\ z)}x_t + W_{(h\ z)}h_{t-1} + b_{(z)}\right) && \text{(forget gate)} \\ h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \bar{h}_t && \text{(recurrent state)} \end{aligned}$$

Figure 2: rnn: forget gate

- LSTM is able to learn long-term dependencies, and the core idea is to use cell state and erase/update/output gates for cell state information.
  - See Understanding LSTM Networks by Colah
  - Mathematical formulation of a LSTM cell



The recurrent state is composed of a "cell state" c_t and an "output state" h_t. Gate f_t modulates if the cell state should be forgotten, i_t if the new update should be taken into account, and o_t if the output state should be reset.

$$\begin{aligned} f_t &= \mathrm{sigm}\left(W_{(x\ f)}x_t + W_{(h\ f)}h_{t-1} + b_{(f)}\right) && \text{(forget gate)} \\ i_t &= \mathrm{sigm}\left(W_{(x\ i)}x_t + W_{(h\ i)}h_{t-1} + b_{(i)}\right) && \text{(input gate)} \\ g_t &= \tanh\left(W_{(x\ c)}x_t + W_{(h\ c)}h_{t-1} + b_{(c)}\right) && \text{(full cell state update)} \\ c_t &= f_t \odot c_{t-1} + i_t \odot g_t && \text{(cell state)} \\ o_t &= \mathrm{sigm}\left(W_{(x\ o)}x_t + W_{(h\ o)}h_{t-1} + b_{(o)}\right) && \text{(output gate)} \\ h_t &= o_t \odot \tanh(c_t) && \text{(output state)} \end{aligned}$$

As pointed out by Gers et al. (2000), the forget bias $b_{(f)}$ should be initialized with large values so that initially $f_t \simeq 1$ and the gating has no effect.

This model was extended by Gers et al. (2003) with "peephole connections" that allow gates to depend on c_{t-1}.

Figure 3: lstm: math

  - A visualization of a (two-layer) LSTM
- GRU (gated recurren unit) uses two (instead of three as in LSTM) gates (update/reset), and it performs similarly to LSTM but with fewer parameters (although LSTM is strictly stronger).
  - Mathematical formuation of a GRU cell
- Graident
  - Note that gated units prevent gradients from vanishing, but not from exploding, which can be solved using **gradient norm clipping** (scaling of the norm).

$$\tilde{\nabla f} = \frac{\nabla f}{||\nabla f||}\min(||\nabla f||, \delta)$$

Several such “cells” can be combined to create a multi-layer LSTM.

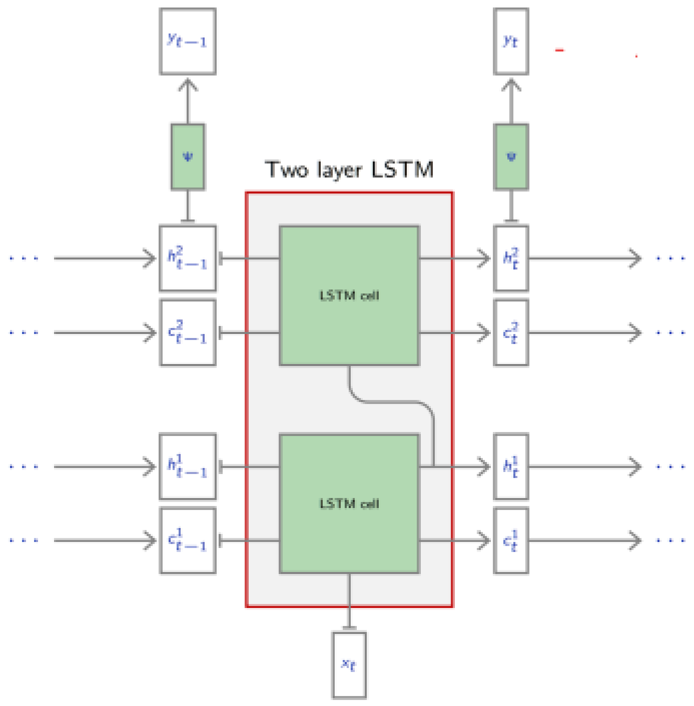


Figure 4: lstm: diagram

The LSTM were simplified into the Gated Recurrent Unit (GRU) by Cho et al. (2014), with a gating for the recurrent state, and a reset gate.

$$\begin{aligned}
r_t &= \mathrm{sigm}\left(W_{(x\ r)}x_t + W_{(h\ r)}h_{t-1} + b_{(r)}\right) && \text{(reset gate)}\\
z_t &= \mathrm{sigm}\left(W_{(x\ z)}x_t + W_{(h\ z)}h_{t-1} + b_{(z)}\right) && \text{(forget gate)}\\
\bar{h}_t &= \tanh\left(W_{(x\ h)}x_t + W_{(h\ h)}(r_t \odot h_{t-1}) + b_{(h)}\right) && \text{(full update)}\\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \bar{h}_t && \text{(hidden update)}
\end{aligned}$$

Figure 5: GRU: math

    - Orthogonal initialization (of the weight matrix) will guarantee that activations will neigher vanish nore explode.
- Applications
  - Sentiment analysis
    - Document-level modeling for sentiment analysis (= text classification), with stacked, bidirectional and gated recurrent networks. (Duyu Tang et al, 2015)
  - Language models
    - Language as Markov Chain $p(\mathbf{w}_t|\mathbf{w}_{1:t-1})$
    - An instance of sequence synthesis where predictions are computed at all time steps
    - Text generation (Max Woolf 2018)
  - Sequence synthesis
  - Neural machine translation (Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation)
  - Text-to-speech synthesis
- Beyond sequences
  - Neural computers
  - Programs as neural nets
  - Graph neural network
- Reference
  - Kyunghyun Cho, "Natural Language Understanding with Distributed Representation", 2015

Lecture 7: Auto-encoders and generative models

- An **auto-encoder** is a composite function made of
  - *encoder* f from the original space $\mathcal{X}$ to a latent space $\mathcal{Z}$
  - *decoder* g to map back to $\mathcal{X}$
  - such that $g \circ f$ is close to the identity on the data, i.e. $\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[||\mathbf{x} - g \circ f(\mathbf{x})||^2] \approx 0$
  - Training an auto-encoder consists of minimizing this loss function to find the best parameterization of f and g.
- Interpolation on latent space can be made to get an intuition of the learned latent representation.
- Denoising auto-encoders
  - The goal is to optimize $h = g \circ f : \mathcal{X} \to \mathcal{X}$ such that a perturbation $\tilde{\mathbf{x}}$ is restored to $\mathbf{x}$.
  - A weakness of denoising auto-encoder is that the posterior $p(\mathbf{x}|\tilde{\mathbf{x}})$ may be multi-modal.
- Generative models
  - a probabilistic model that can be used to simulate the data, $\mathbf{x} \sim p(\mathbf{x};\theta)$.
  - Applications
    - Supper-resolution, Compression, text-to-speech
    - Proteomics, drug discovery, astronomy
    - Planning, exploration, model-based RL
  - The decoder g can be assessed by introducing a density model q over the latent space $\mathcal{Z}$ for sampling and mapping back into the data space $\mathcal{X}$. (e.g., Gaussian $q(\mathbf{z}) = \mathcal{N}(\hat{\mu}, \hat{\Sigma})$)
  - Sampled and generated results are not satisfactory because the density model p on the latent space is too simple and inadequate.
- Variational inference (VI)
  - A prescribed latent variable model that defines a joint probability $p(\mathbf{x},\mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z})$
  - Bayes rule gives $p(\mathbf{z}|\mathbf{x}) = \dfrac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{x})}$, which is intractable for integrating over $\mathbf{x}$.
  - VI turns the posterior inference into an optimization problem that minimize the KL divergence between $p(\mathbf{z}|\mathbf{x})$ and the approximation $q(\mathbf{z}|\mathbf{x};\nu)$
    - See slides pp. 44 - pp.47 for details of the KL divergence, *evidence lower bound objective* (ELBO), and the optimization setups.
    - ELBO encourages distributions to place their mass on configurations of latent variables that explain the oberved data, and close to the prior.
- Variational auto-encoders
  - Variational auto-encoder is a deep latent model where
    - $p(\mathbf{x}|\mathbf{z};\theta)$ is parameterized with a **generative network** NN_θ (decoder) that takes input $\mathbf{z} \in \mathcal{Z}$ and outputs parameters $\phi = \text{NN}_\theta(\mathbf{z})$ to the data distribution, i.e.

$$\mu, \sigma = \mathrm{NN}_\theta(\mathbf{z}), \quad p(\mathbf{x}|\mathbf{z};\theta) = \mathcal{N}(\mathbf{x}; \mu, \sigma^2\mathbf{I})$$

  - * The approximate posterior $q(\mathbf{z}|\mathbf{x};\varphi)$ is parameterized with an **inference network** NN_φ (encoder) that takes as input $\mathbf{x}$ and outputs parameters $\nu = \mathrm{NN}_\varphi(x)$ to the approximate posterior. E.g.

$$\mu, \sigma = \mathrm{NN}_\varphi(\mathbf{x}), \quad q(\mathbf{z}|\mathbf{x};\varphi) = \mathcal{N}(\mathbf{z}; \mu, \sigma^2\mathbf{I})$$

  - We use variational inference to jointly optimize the generative and inference networks.
    - * Doing so involves Monte Carlo integration (for computing gradients of the ELBO w.r.t. θ) and reparameterization trick + Monte Carlo (for computing gradients of ELBO w.r.t. φ)

Lecture 8: Generative Adversarial Networks

- Generative adversarial networks (GANs)
  - Two-player game
    - * Generator network $g(\cdot;\theta) : \mathcal{Z} \to \mathcal{X}$ with prior $p(\mathbf{z})$ on latent space, thereby inducing a generative distribution

$$\mathbf{x} \sim q(\mathbf{x};\theta) \iff \mathbf{z} \sim p(\mathbf{z}), \mathbf{x} = g(\mathbf{z};\theta)$$

    - * Discriminative network $d(\cdot;\phi) : \mathcal{X} \to [0,1]$ as a classifier to distinguish between true samples $\mathbf{x} \sim p(\mathbf{x})$ and generated samples $\mathbf{x} \sim q(\mathbf{x};\theta)$
  - Objective of GANs
    - * Use cross-entropy loss for d and we have the *value function* (log-likelihood)

$$V(\phi,\theta) = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[\log d(\mathbf{x};\theta)] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[1 - d(g(\mathbf{z};\theta);\phi))]$$

    - * The ultimate goal is $\theta^* = \arg\min_\theta \max_\phi V(\phi,\theta)$
    - * Mathematically θ^* is minimum $\iff p(\mathbf{x}) = q(\mathbf{x};\theta)$, that is, the corresponding generative model can perfectly reproduce the true data distribution.
  - Learning process
    - * Alternating SGD

$$\theta \leftarrow \theta - \gamma\nabla_\theta V(\phi,\theta); \quad \phi \leftarrow \phi - \gamma\nabla_\phi V(\theta,\phi)$$

    - * For each step of θ we can take k steps on ϕ to make the classifier near optimal
    - * Computing ∇_θ requres backprop through d before computing the partial derivatives w.r.t. g's internals.
  - Open problems
    - * Training standard GAN often results in pathological behaviors due to
      - · Oscillations without convergence
      - · Vanishing gradients
      - · Mode collapse: g models well on a small sub-population concentrating on a few modes of the data distribution.
      - · Performance diffucult to assess in practice.
- Wasserstein GANs
  - Original GAN as defined above suffers from vanishing gradients, especially when initialy $\mathbf{x} \sim q(\mathbf{x};\theta)$ can be so bad that the response of d saturates, meaning d is nearly perfect there fore $V(\phi,\theta) = ...(definedabove)$ has near-zero gradients which halts the optimization.
    - * One of the reason for the setback is that Jensen-SHannon divergence poorly accounts for the metric structure of the space.
  - Wasserstein GAN uses Wasserstein-1 distance

$$\theta^* = \arg\min_\theta W_1(p(\mathbf{x}||q(\mathbf{x};\theta)) = \arg\min_\theta \max_{\phi:||d(\cdot;\phi)||_L \le 1} \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[d(\cdot;\phi)] - \mathbb{E}_{\mathbf{x}\sim q(\mathbf{x};\theta)[d(\mathbf{x};\theta)]}$$

    - * In this case, $d : \mathcal{X} \to \mathbb{R}$ is a critic function that satisfies 1-Lipschitzness.
    - * See Arjovsky et al (2017) and Gulrajani et al (2017) for details.
  - As a result, Wasserstein GANS benefit from
    - * A meaningful loss metric

        - Improved stability (no mode collapse is observed)
- Convergence of GANs
    - TODO: this section (from pp. 39 - 54) is skipped for now, with some brief notes as follows:
        - GANs suffer from saddle point, see Ferenc Huszár, GANs are Broken in More than One Way, 2017 for details
        - The loss function of Vanilla GANs and Wisserstein GANs can be unified to analyse the convergence.
        - Reference: Mescheder et al, 2018. Which Training Methods for GANs do actually Converge?
- State of the art
    - Progress: Wasserstein GANs as baseline + Gradient Penalty (Gulrajani 2017) + quite a few other tricks
    - BigGANs
        - Brock et al, 2018: Large Scale GAN Training For High Fidelit Natural Image Synthes
    - StyleGAN
        - V1: Karras et al, 2018
        - V2: Karras et al, 2019
- Applications
    - The prior $p(\mathbf{z})$ of the latent space need not be a random noise distribution
    - Image-to-image translation: CycleGANs (Zhu et al, 2017)
    - Nvidia: Stroke of Genius: GauGAN Turns Doodles into Stunning, Photorealistic Landscapes
    - Captioning
    - Text-to-image synthesis
        - Zhang et al, 2017: StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks
    - Music generation
        - MuseGAN (Dong et al, 2018)
    - Accelerating scientific simulators
        - Learning particle physics (Paganini et al, 2017)
        - Learning cosmological models (Rodriguez et al, 2018)
        - Brain reading (Shen et al, 2018)

Supplementary Notes

This section contains reading notes for lecture 12 - 13 of the EPFL EE-559 – Deep Learning course as supplementary materials to the INFO8010 course.

Recurrent models and NLP

Attention models

Resrouces

- EPFL EE-559 – Deep Learning - EE-559 "Deep Learning", taught by François Fleuret in the School of Engineering of the École Polytechnique Fédérale de Lausanne, Switzerland.
- Dive into Deep Learning: An interactive deep learning book with code, math, and discussions, based on the NumPy interface.
- Notes of deep learning specialization, good for reviewing the fundamentals of DL.