DSME 6635: Artificial Intelligence for Business Research

Deep Learning Basics

Renyu (Philip) Zhang

1

Agenda

- Supervised Learning Model Training
- Deep Neural Nets
- Computations in Deep Learning

2

2

Supervised Learning

- Given the data observations: $(\mathrm{X}, \mathrm{Y})_i, i = 1,2, \ldots, n$
- Define your model
  - Linear regression: $\mathrm{Y} = a + b \cdot X + \epsilon$
- Define your loss function:
  - Regression: Squared error: $(Y - f(X))^2$
  - Classification: Cross entropy: $-\log(\mathrm{p})Y - \log(1-p)(1-Y)$
- Pick your optimizer
  - OLS estimator
  - Gradient descent
- Run your model on a CPU/GPU Cluster

Deep learning means the model family is Deep Neural Nets.

3

3

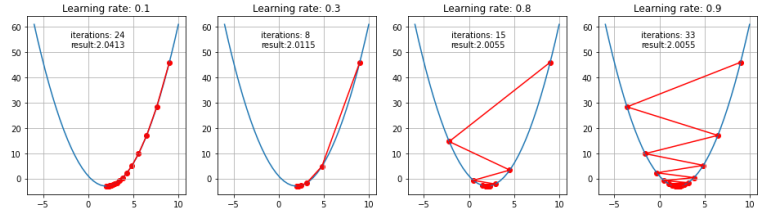
Training a Model

$$\hat{\theta} := argmin_\theta \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(Y_i, f(X_i, \theta))$$

- **Gradient Descent**: A first-order iterative optimization for finding a local minimum of a differentiable function.

Algorithm 1 Gradient Descent

1: Guess $\mathbf{x}^{(0)}$, set $k \leftarrow 0$
2: **while** $||\nabla f(\mathbf{x}^{(k)})|| \geq \epsilon$ **do**
3: $\quad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - t_k \nabla f(\mathbf{x}^{(k)})$
4: $\quad k \leftarrow k + 1$
5: **end while**
6: **return** $\mathbf{x}^{(k)}$



4

4

Example: OLS

- Loss function: $S(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left| y_i - \sum_{j=1}^{p} X_{ij}\beta_j \right|^2 = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2.$
- Closed-form Solution: $\hat{\boldsymbol{\beta}} = (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y}.$
- Gradient Descent: $w^{k+1} = w^k - \alpha_k \underbrace{X^T(Xw^k - y)}_{\nabla f(w^k)},$
- Statistics convergence vs. Optimization convergence:
  - **Convergence of estimator**: Is the estimator constructed by the loss function converging to the true underlying estimand? How fast? Bias and consistency?
  - **Convergence of optimization**: Will the optimization algorithm we use converge to the minimizer of the loss function given data? How fast?

Cost Function

$$J(\Theta_0, \Theta_1) = \frac{1}{2m}\sum_{i=1}^{m}[h_\Theta(x_i) - y_i]^2$$

($h_\Theta(x_i)$: Predicted Value; y_i: True Value)

Gradient Descent

$$\Theta_j = \Theta_j - \alpha\frac{\partial}{\partial\Theta_j}J(\Theta_0, \Theta_1)$$

(α: Learning Rate)

Now,

$$\begin{aligned}\frac{\partial}{\partial\Theta}J_\Theta &= \frac{\partial}{\partial\Theta}\frac{1}{2m}\sum_{i=1}^{m}[h_\Theta(x_i) - y]^2 \\ &= \frac{1}{m}\sum_{i=1}^{m}(h_\Theta(x_i) - y)\frac{\partial}{\partial\Theta_j}(\Theta x_i - y) \\ &= \frac{1}{m}(h_\Theta(x_i) - y)x_i\end{aligned}$$

Therefore,

$$\Theta_j := \Theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}[(h_\Theta(x_i) - y)x_i]$$

5

5

Another Example: Logistic Regression

- Model: $\Pr(y_t = 1|x_t) = \frac{\exp(x_t'\beta)}{1 + \exp(x_t'\beta)}.$
- MLE Estimator/Cross-Entropy Loss: $\hat{\beta} = \arg\max_\beta [\ln \mathcal{L}(\beta)] = \arg\max_\beta \left[\sum_t \left(y_t \ln\left(\frac{\exp(x_t'\beta)}{1 + \exp(x_t'\beta)}\right) + (1 - y_t)\ln\left(\frac{1}{1 + \exp(x_t'\beta)}\right)\right)\right].$
- Gradient Descent: What is the gradient of logistic regression? (HW)

6

6

Gradient Descent

- Reference: https://www.stat.cmu.edu/%7Eryantibs/convexopt-F13/scribes/lec6.pdf
- When does gradient descent converge? $f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) - \frac{1}{2L}\|\nabla f(\mathbf{x}_k)\|_2^2.$
- For Lipschitz continuous functions, you can use the Taylor expansion at a point x_k to show the descent lemma.
- You can use the descent lemma to show that, for a small enough learning rate, the function in period k+1 is smaller than that in period k.
- You leverage convexity to show the sequence decreases to the global minimum.

Theorem 6.1 *Suppose the function $f : \mathbb{R}^n \to \mathbb{R}$ is convex and differentiable, and that its gradient is Lipschitz continuous with constant $L > 0$, i.e. we have that $\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2$ for any x, y. Then if we run gradient descent for k iterations with a fixed step size $t \leq 1/L$, it will yield a solution $f^{(k)}$ which satisfies*

$$f(x^{(k)}) - f(x^*) \leq \frac{\|x^{(0)} - x^*\|_2^2}{2tk}, \tag{6.1}$$

where $f(x^*)$ is the optimal value. Intuitively, this means that gradient descent is guaranteed to converge and that it converges with rate $O(1/k)$.

7

7

Gradient Descent

- When and why does gradient descent converge?
- At what speed does it converge?

Theorem 3. *Let $f : S \to \mathbb{R}$ be a strongly convex function with parameters m, M as in the definition above. For any $\epsilon > 0$ we have that $f(\mathbf{x}^{(k)}) - \min_{\mathbf{x} \in S} f(\mathbf{x}) \leq \epsilon$ after $k^\star$ iterations for any $k^\star$ that respects:*

$$k^\star \geq \frac{\log\left(\frac{f(\mathbf{x}^{(0)}) - \alpha^\star}{\epsilon}\right)}{\log\left(\frac{1}{1 - m/M}\right)}$$

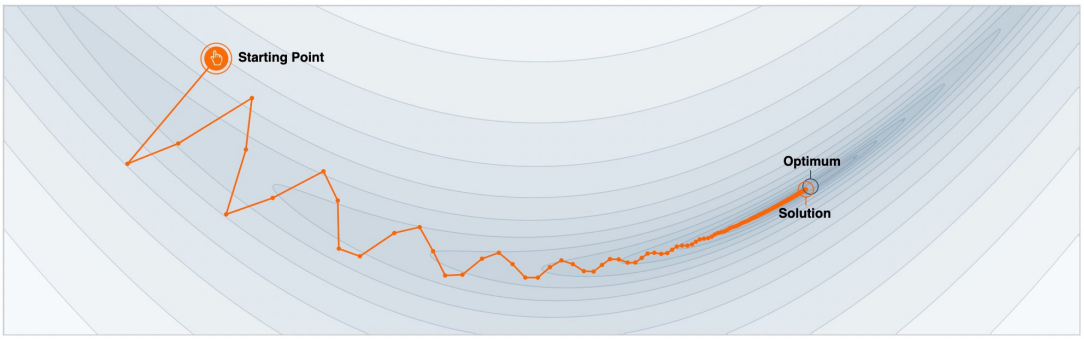
- Basically, you need O(1/epsilon) steps to converge for convex functions. For strongly convex functions, you need O(log(1/epsilon)) steps to converge.

8

8

Momentum

- https://distill.pub/2017/momentum/



Step-size α = 0.0033
0 0.003 0.006

Momentum β = 0.81
0.00 0.500 0.990

We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

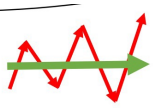
9

9

Momentum

- Reference: https://cs182sp21.github.io/static/slides/lec-4.pdf

Intuition: if successive gradient steps point in **different** directions, we should **cancel off** the directions that disagree



if successive gradient steps point in **similar** directions, we should **go faster** in that direction

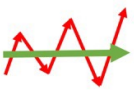
update rule:

$$\theta_{k+1} = \theta_k - \alpha g_k$$

before: $g_k = \nabla_\theta \mathcal{L}(\theta_k)$

now: $g_k = \nabla_\theta \mathcal{L}(\theta_k) + \underbrace{\mu g_{k-1}}$

"blend in" previous direction



10

10

Stochastic Gradient Descent

- When the data set is large (which is usually the case for deep learning), it is impossible to use all the data to update the gradient each time.
- Instead, we sample data to update gradient: SGD.

1. Sample $\mathcal{B} \subset \mathcal{D}$
2. Estimate $g_k \leftarrow -\nabla_\theta \frac{1}{B} \sum_{i=1}^{B} \log p(y_i|x_i, \theta) \approx \nabla_\theta \mathcal{L}(\theta)$
3. $\theta_{k+1} \leftarrow \theta_k - \alpha g_k$

- Each iteration is called a mini-batch.
- In practice, we shuffle data instead of randomly sampling.

11

11

Adaptive Gradient

Update rule:

$$\begin{aligned} V_t &= \mu V_{t-1} - \alpha \nabla L_t(W_{t-1}) \\ W_t &= W_{t-1} + V_t \end{aligned}$$

- $\alpha > 0$ – *learning rate* (typical choices: $0.01, 0.1$)
- $\mu \in [0, 1)$ – *momentum* (typical choices: $0.9, 0.95, 0.99$)

Update rule:

$$W_t = W_{t-1} - \alpha \frac{\nabla L_t(W_{t-1})}{\sqrt{\sum_{t'=1}^{t} \nabla L_{t'}(W_{t'-1})^2}}$$

Learning rate adapted per coordinate:

- Highly varying coordinate $\longrightarrow$ suppress
- Rarely varying coordinate $\longrightarrow$ enhance

12

12

Adam

- Adam: The most well-used algorithm to automatically tune momentum and learning rates.
- Adam combines AdaGrad and RMSP
- Adam is the default optimizer tuner in deep learning.

Adam

$M_0 = \mathbf{0}, R_0 = \mathbf{0}$ (Initialization)

For $t = 1, \ldots, T$:

$M_t = \beta_1 M_{t-1} + (1-\beta_1)\nabla L_t(W_{t-1})$ (1st moment estimate)

$R_t = \beta_2 R_{t-1} + (1-\beta_2)\nabla L_t(W_{t-1})^2$ (2nd moment estimate)

$\hat{M}_t = M_t / (1-(\beta_1)^t)$ (1st moment bias correction)

$\hat{R}_t = R_t / (1-(\beta_2)^t)$ (2nd moment bias correction)

$W_t = W_{t-1} - \alpha \frac{\hat{M}_t}{\sqrt{\hat{R}_t + \epsilon}}$ (Update)

Return W_T

Hyper-parameters:

- $\alpha > 0$ – learning rate (typical choice: 0.001)
- $\beta_1 \in [0,1)$ – 1st moment decay rate (typical choice: 0.9)
- $\beta_2 \in [0,1)$ – 2nd moment decay rate (typical choice: 0.999)

Adam: A method for stochastic optimization [PDF] arxiv.org

DP Kingma, J Ba

arXiv preprint arXiv:1412.6980, 2014 · arxiv.org

We introduce Adam, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning. Some connections to related algorithms, on which Adam was inspired, are discussed. We also analyze the theoretical convergence properties of the algorithm and provide a regret bound on the convergence rate that is comparable to the best known results under the online convex optimization framework. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods. Finally, we discuss AdaMax, a variant of Adam based on the infinity norm.

arxiv.org

收起 ^

☆ 保存 99 引用 被引用次数：164021 相关文章 所有 27 个版本 »

13

13

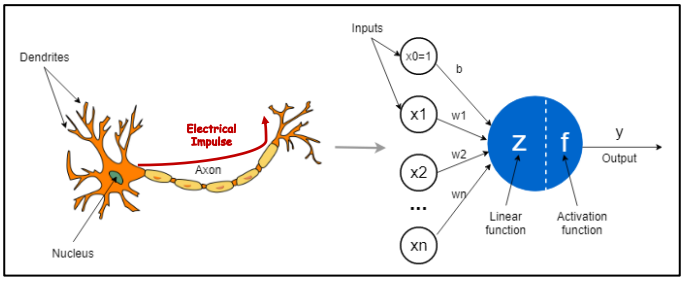
Agenda

- Supervised Learning Model Training
- **Deep Neural Nets**
- Computations in Deep Learning

14

14

Symbolic AI vs. Connectionist AI

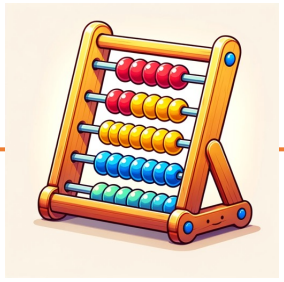


Neuron and Artificial Neuron

15

15

Computational Power of Human



10^{16} computations in the entire history

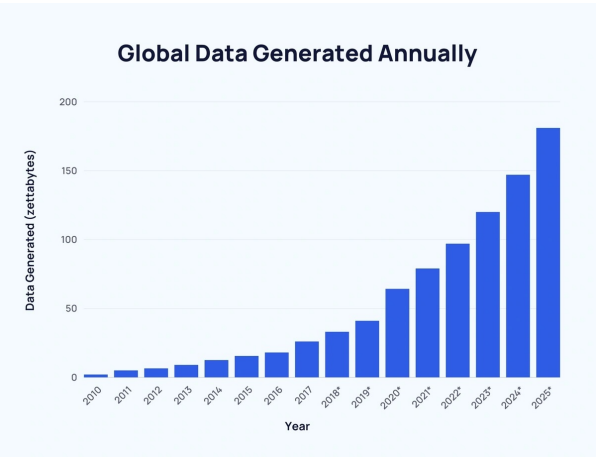
1945

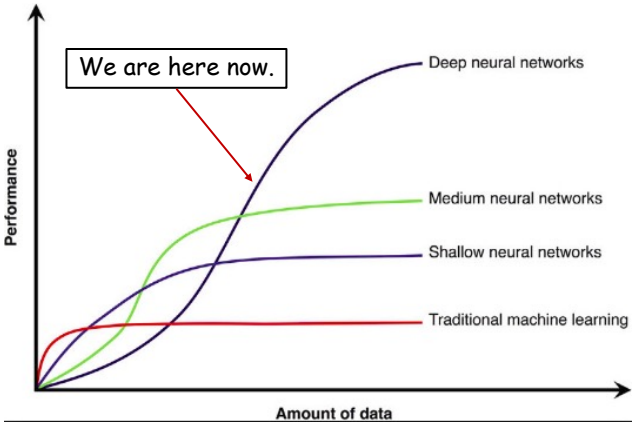
More than 10^{16} computations per second

16

16

Modern AI Relies on Big Data



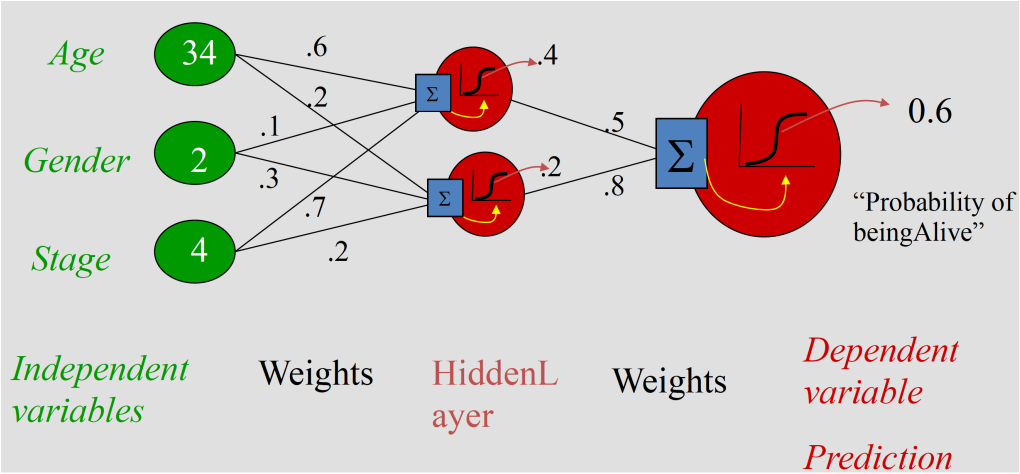


17

17

Neural Network Models

- We build a model to predict patient mortality based on their demographics.



- University Approximation Theorem: One hidden layer NN is enough to represent (not learn) an approximation of any function with an arbitrary degree of accuracy.

18

18

Are Simple Multilayer Perceptrons (MLP) Outdated?

nature > articles > article

Article | Open access | Published: 01 December 2021

Advancing mathematics by guiding human intuition with AI

Alex Davies, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, Marc Lackenby, Geordie Williamson, Demis Hassabis & Pushmeet Kohli

Nature **600**, 70–74 (2021) | Cite this article

266k Accesses | **128** Citations | **1624** Altmetric | Metrics

Abstract

The practice of mathematics involves discovering patterns and using these to formulate and prove conjectures, resulting in theorems. Since the 1960s, mathematicians have used computers to assist in the discovery of patterns and formulation of conjectures[1], most famously in the Birch and Swinnerton-Dyer conjecture[2], a Millennium Prize Problem[3]. Here we provide examples of new fundamental results in pure mathematics that have been discovered with the assistance of machine learning—demonstrating a method by which machine learning can aid mathematicians in discovering new conjectures and theorems. We propose a process of using machine learning to discover potential patterns and relations between mathematical objects, understanding them with attribution techniques and using these observations to guide intuition and propose conjectures. We outline this machine-learning-guided framework and demonstrate its successful application to current research questions in distinct areas of pure mathematics, in each case showing how it led to meaningful mathematical contributions on important open problems: a new connection between the algebraic and geometric structure of knots, and a candidate algorithm predicted by the combinatorial invariance conjecture for symmetric groups[4]. Our work may serve as a model for collaboration between the fields of mathematics and artificial intelligence (AI) that can achieve surprising results by leveraging the respective strengths of mathematicians and machine learning.

https://www.bilibili.com/video/BV1YZ4y1S72j

Note necessarily!

The key is to identify interesting and impactful applications.

Deep-Learning-Based Causal Inference for Large-Scale Combinatorial Experiments: Theory and Empirical Evidence

Zikun Ye[1], Zhiqi Zhang[2], Dennis J. Zhang[2], Heng Zhang[3], Renyu Zhang[4]
[1] University of Illinois Urbana-Champaign, Urbana, IL
[2] Washington University in St. Louis, St. Louis, MO
[3] Arizona State University, Tempe, AZ
[4] The Chinese University of Hong Kong, Hong Kong, China
zikunye2@illinois.edu, z.zhiqi@wustl.edu, denniszhang@wustl.edu, hengzhang24@asu.edu, philipzhang@cuhk.edu.hk

Large-scale online platforms launch hundreds of randomized experiments (a.k.a. A/B tests) every day to iterate their operations and marketing strategies. The combinations of these treatments are typically not exhaustively tested, which triggers an important question of both academic and practical interest: Without observing the outcomes of all treatment combinations, how does one estimate the causal effect of any treatment combination and identify the optimal treatment combination? We develop a novel framework combining deep learning and doubly robust estimation to estimate the causal effect of any treatment combination for each user on the platform when observing only a small subset of treatment combinations. Our proposed framework (called debiased deep learning, DeDL) exploits Neyman orthogonality and combines interpretable and flexible structural layers in deep learning. We prove theoretically that this framework yields efficient, consistent, and asymptotically normal estimators under mild assumptions, thus allowing for identifying the best treatment combination when observing only a few combinations. To empirically validate our method, we collaborated with a large-scale video-sharing platform and implemented our framework for three experiments involving three treatments where each combination of treatments is tested. When observing only a subset of treatment combinations, our DeDL approach significantly outperforms other benchmarks to accurately estimate and infer the average treatment effect of any treatment combination, and to identify the optimal treatment combination.

Key words: Deep Learning, Double Machine Learning, Causal Inference, Field Experiments, Experimentation on Online Platforms

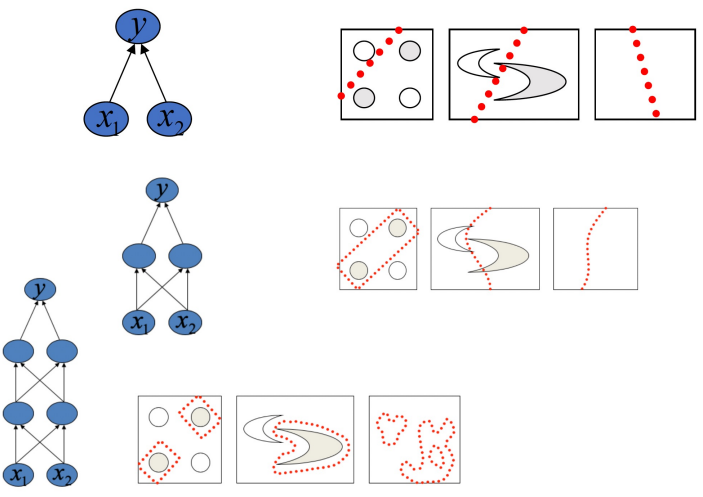
https://www.bilibili.com/video/BV19v4y1h7Ev

19

19

Neural Nets

- More layers, more complex functions, more powerful learning and more required data.
- 0 hidden layers: Linear classifier
  - Hyperplanes
- 1 hidden layer
  - Boundary of convex region
- 2 hidden layers
  - Combinations of convex regions



20

20

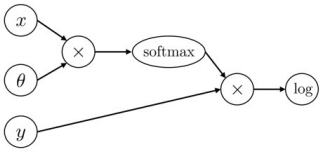
Neural Nets

- Logistic Regression:

$$f_\theta(x) = \begin{bmatrix} x^T\theta_{y_1} \\ x^T\theta_{y_2} \\ \dots \\ x^T\theta_{y_m} \end{bmatrix} \qquad f_\theta(x) = \theta x$$

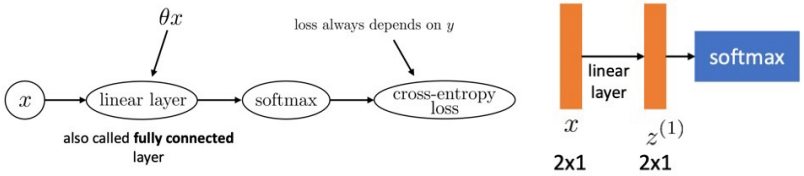
matrix

$$\begin{bmatrix} \theta_{y_1} \\ \theta_{y_2} \\ \theta_{y_3} \end{bmatrix} \times x = \begin{bmatrix} x^T\theta_{y_1} \\ x^T\theta_{y_2} \\ \dots \\ x^T\theta_{y_m} \end{bmatrix}$$



$$p_\theta(y = i|x) = \text{softmax}(f_\theta(x))[i] = \frac{\exp(f_{\theta,i}(x))}{\sum_{j=1}^{m}\exp(f_{\theta,j}(x))}$$

- A simpler representation:



21

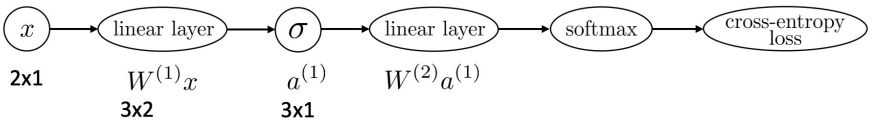
21

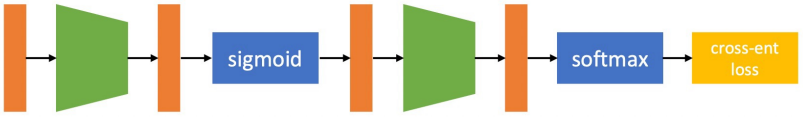
More Complex Neural Nets

- Add functions before SoftMax

$$\phi(x) = \begin{pmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \\ x_1x_2 \end{pmatrix}$$

$$\text{softmax}(\phi(x)^T\theta)$$



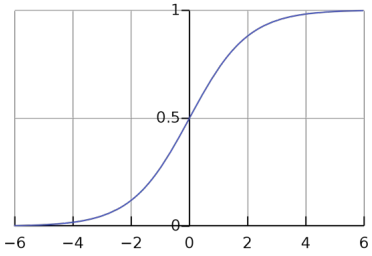


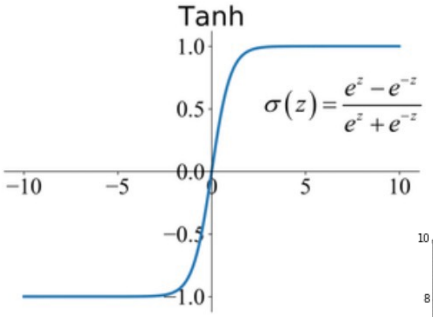
22

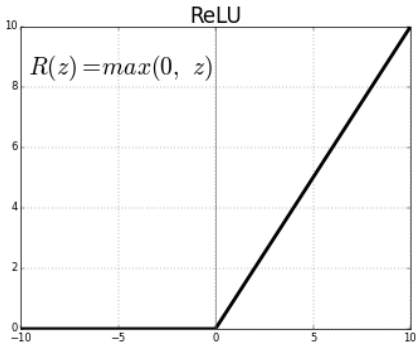
22

Activation Functions

- Activation functions:
  - Sigmoid
  - Tanh
  - ReLU (mostly used)







23

23

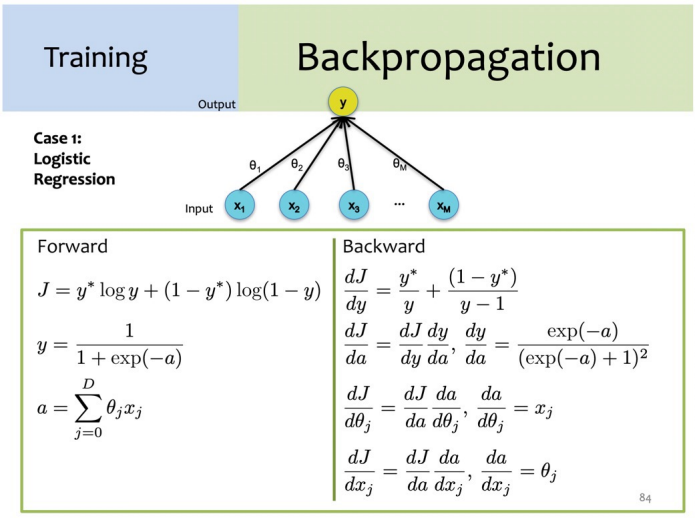
Estimation of Neural Nets

- Repeatedly apply the chain rule to obtain the gradient of the loss function with respect to the parameters and use Adam to update the parameters.

- Initialize network parameters randomly or semi-randomly.

- For each epoch:
  - Shuffle data
  - For each minibatch:
    - Use backpropagation to compute the gradient
    - Use Adam to update the parameters.

- Backpropagation: Chain rule.

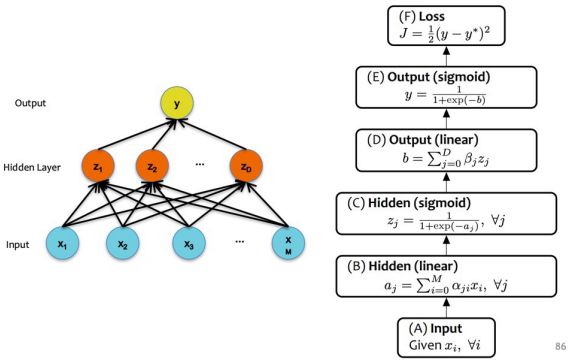


24

24

Backpropagation

Training | Backpropagation



86

Training | Backpropagation

Case 2: Neural Network

Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \frac{da_j}{dx_i} = \sum_{j=0}^{D} \alpha_{ji}$$

87

25

25

Gradient Descent for DNN

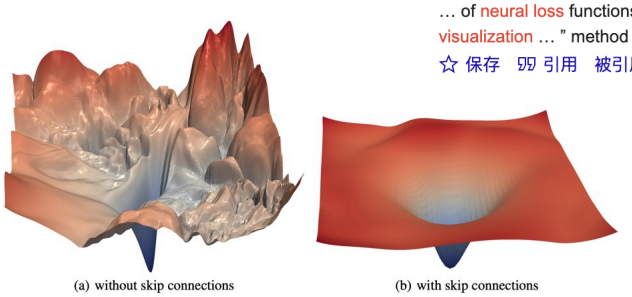
- How does the loss function landscape of DNN look like?
- In general, it's very complex and we don't know.

Visualizing the loss landscape of neural nets

H Li, Z Xu, G Taylor, C Studer… - Advances in neural …, 2018 - proceedings.neurips.cc

… of neural loss functions, and the effect of loss landscapes on generalization, using a range of visualization … " method that helps us visualize loss function curvature and make meaningful …

☆ 保存 99 引用 被引用次数：1746 相关文章 所有 16 个版本 »



(a) without skip connections (b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

26

26

Optimization for DNN

- Initialization matters.
- Normalization/standardization matters.

Understanding the difficulty of training deep feedforward neural networks
X Glorot, Y Bengio - Proceedings of the thirteenth …, 2010 - proceedings.mlr.press
Whereas before 2006 it appears that deep multi-layer neural networks were not successfully trained, since then several algorithms have been shown to successfully train them, with experimental results showing the superiority of deeper vs less deep architectures. All these experimental results were obtained with new initialization or training mechanisms. Our objective here is to understand better why standard gradient descent from random initialization is doing so poorly with deep neural networks, to better understand these recent …
☆ Save 99 Cite Cited by 17168 Related articles All 21 versions »

Batch normalization: Accelerating deep network training by reducing internal covariate shift
S Ioffe, C Szegedy - International conference on machine …, 2015 - proceedings.mlr.press
Abstract Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as internal covariate shift, and address the problem by normalizing layer inputs. Our method draws its strength from making …
☆ Save 99 Cite Cited by 39759 Related articles All 40 versions »

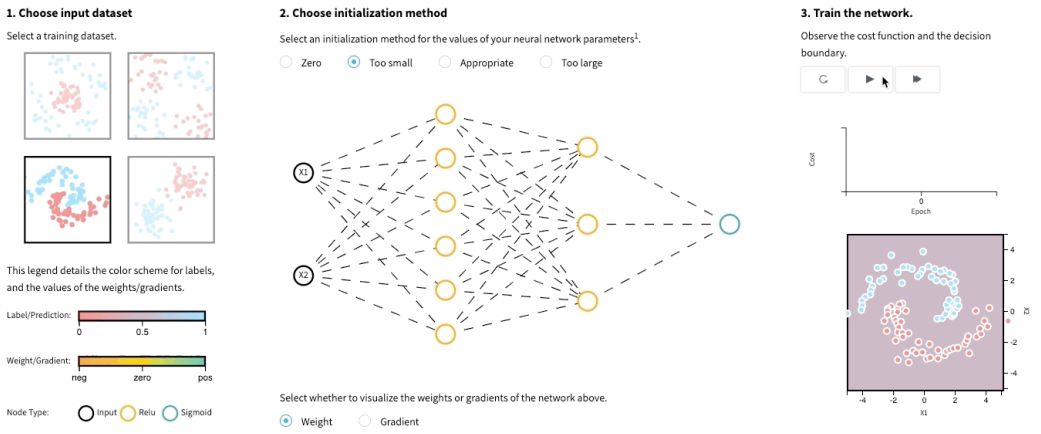
Delving deep into rectifiers: Surpassing human-level performance on imagenet classification
K He, X Zhang, S Ren, J Sun - Proceedings of the IEEE …, 2015 - openaccess.thecvf.com
Rectified activation units (rectifiers) are essential for state-of-the-art neural networks. In this work, we study rectifier neural networks for image classification from two aspects. First, we propose a Parametric Rectified Linear Unit (PReLU) that generalizes the traditional rectified unit. PReLU improves model fitting with nearly zero extra computational cost and little overfitting risk. Second, we derive a robust initialization method that particularly considers the rectifier nonlinearities. This method enables us to train extremely deep rectified models …
☆ Save 99 Cite Cited by 16784 Related articles All 19 versions »

27

27

Initialization

Reference: https://www.deeplearning.ai/ai-notes/initialization/index.html



28

28

DL Libraries

- Good news: "You will probably never implement backpropagation by yourselves unless for your homework."
- General parallel computing libraries:
  - TensorFlow (Google)
  - PyTorch (Facebook)
- Deep learning libraries:
  - Keras
  - HuggingFace
- TensorFlow/PyTorch is like numpy but
  - Much better documentation
  - Much better supported in parallel computing (GPUs)
  - Much better solver with large data
  - Much better for deployment (probably you don't care)
- The core function of DL libraries: Take derivatives and record them.

Operation	NumPy	PyTorch
Array/Tensor Creation	`numpy.array()`	`torch.tensor()`
Dimensions	`array.ndim`	`tensor.dim()`
Shape	`array.shape`	`tensor.size()`
Sum over all elements	`numpy.sum(array)`	`torch.sum(tensor)`
Mean	`numpy.mean(array)`	`torch.mean(tensor)`
Standard Deviation	`numpy.std(array)`	`torch.std(tensor)`
Element-wise Sum	`array1 + array2`	`tensor1 + tensor2`
Element-wise Product	`array1 * array2`	`tensor1 * tensor2`
Matrix Multiplication	`numpy.dot(a, b)`	`torch.matmul(a, b)`
Reshape	`array.reshape()`	`tensor.view()`
Transpose	`array.T`	`tensor.t()`
Max value	`numpy.max(array)`	`torch.max(tensor)`
Min value	`numpy.min(array)`	`torch.min(tensor)`
Concatenate	`numpy.concatenate([a, b], axis)`	`torch.cat([a, b], dim)`

29

29

Overfitting and Regularization

- Deep neural nets are over-parameterized tend to overfit.
  - Recall the bias-variance trade-off.
- Regularization: A standard way to reduce model complexity.
- Regularization for linear models:
  - Subset selection: Identify a subset of relevant features and throw away others (e.g., stepwise selection).
  - Shrinkage: Use all features but gradually shrink the parameters of some features to 0 (Lasso)
  - Dimension reduction: Projecting the features to a lower dimensional space (PCA).

Algorithm 6.2 *Forward stepwise selection*

1. Let $\mathcal{M}_0$ denote the *null* model, which contains no predictors.
2. For $k = 0, \ldots, p-1$:
   (a) Consider all $p-k$ models that augment the predictors in $\mathcal{M}_k$ with one additional predictor.
   (b) Choose the *best* among these $p-k$ models, and call it $\mathcal{M}_{k+1}$. Here *best* is defined as having smallest RSS or highest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

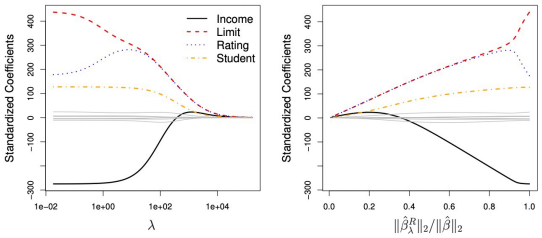
30

30

Ridge and Lasso

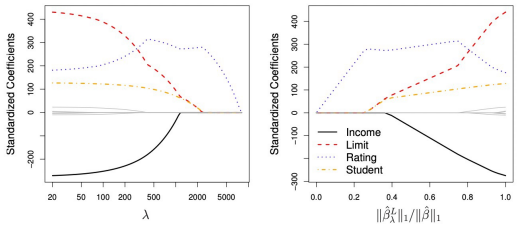
Ridge regression:

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2=\mathrm{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2, \qquad (6.5)$$



Lasso regression:

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j| =\mathrm{RSS}+\lambda\sum_{j=1}^{p}|\beta_j|. \qquad (6.7)$$



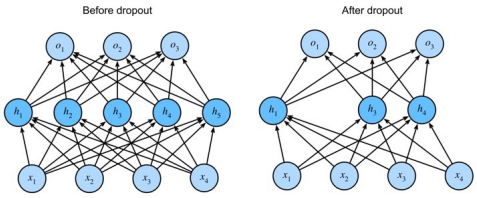
31

31

Dropout

- Training with noisy data is like regularization (Bishop 1995).
- Dropout provides a computationally efficient and effective way to add such noise into neural nets training.
- If you add a drop out layer with probability p, then each intermediate activation value h from the previous layer becomes:

$$h' = \begin{cases} 0 & \text{with probability } p \\ \frac{h}{1-p} & \text{otherwise} \end{cases}$$



[PDF] Dropout: a simple way to prevent neural networks from overfitting
N Srivastava, G Hinton, A Krizhevsky… - The journal of machine …, 2014 - jmlr.org
… This significantly reduces overfitting and gives major improvements over other regularization methods. We show that dropout improves the performance of neural networks on …
☆ 保存 99 引用 被引用次数：48060 相关文章 所有 36 个版本 Web of Science: 22354 »

32

32

Putting Everything Together

- **Model**: You can represent a complex function as a networks of computations.
- **Loss function**: Cross-entropy for classification and mean squared error for regression.
- **Estimation/optimization**: Use backpropagation to find the gradients fast; SGD + Adam to update the parameters with gradients.
- **Implementation**: Use PyTorch to code parallel vector computation; use Keras/Hugging Face to directly specify the model.

33

33

Agenda

- Supervised Learning Model Training
- Deep Neural Nets
- **Computations in Deep Learning**

34

34

Computing Equipment

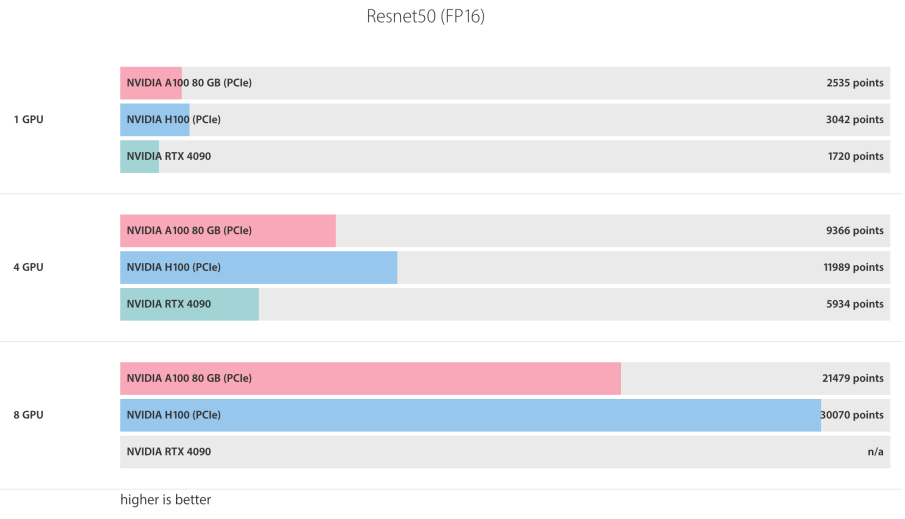
- **Self-made workstations/servers**:
  - CPU + Nvidia 4090 (~US$1,500, but very hard to get)
  - CUHK Business School DOT will deploy a server with 8 4090s soon.
- **Cloud**:
  - GCP https: https://cloud.google.com/compute/vm-instance-pricing#accelerator-optimized (US$ 4.05 per 1 A100 per hour)
  - Amazon: https://aws.amazon.com/ec2/instance-types/p4/ (US$32.77 per 8 A100 per hour)
- **Computing costs**: Proportional to the size of the network (the number of parameters) and the size of the training data.

35

35

GPU Comparison

- https://bizon-tech.com/gpu-benchmarks/NVIDIA-A100-80-GB-(PCIe)-vs-NVIDIA-H100-(PCIe)-vs-NVIDIA-RTX-4090/624vs632vs637



36

36

Model Size and Training Time

- Resnet-50, 12 million parameters, ImageNet Data, TF32, 30mins on an 8 A100s server (DGX).
  - ~0.5*8*1.5 = 6~10 hrs on 4090.
- BERT, 110 million parameters, 170GB BooksCorpus and Wikipedia Data, TF32, 5 hrs on DGX.
- BERT fine-tuning, Stanford Question Answering Data, 3~5 mins on DGX.
- GPT-3, 175B parameters (3.64e23 FLOPs, 300B training tokens), TF32, 128 DGX servers (A100: 80 TFLOP/s). How long it takes to pre-train GPT-3?
  - 3.64*e23/(80*e12)/128/8/24/3600 ~= 51 Days – 100 Days, which means 3 months.

37

37