IE 398/CS 398 Deep Learning

University of Illinois at Urbana-Champaign

Spring 2019

Lecture 6

Project proposals due Thursday, February 21 at 5:00 PM.

Finite difference formula:

Let $g(v, z) : \mathbb{R} \times \mathcal{Z} \to \mathbb{R}$.

$$\frac{\partial g}{\partial v} \approx \frac{g(v + \Delta, z) - g(v, z)}{\Delta}. \tag{1}$$

For example, let $g = \rho(f(x; \theta), y)$, let $v = K_{i,j,p}$, and

$$z = \{\text{All other parameters}\} = \{b, W, K_{m,n,a \neq (i,j,p)}\}. \tag{2}$$

3-d convolutions:

Let the input image be $X \in \mathbb{R}^{d \times d \times d}$ and a filter $K \in \mathbb{R}^{k_y \times k_x \times k_z}$.

We define a 3-dimensional convolution of the matrix X with the filter K as the map

$$X * K : \mathbb{R}^{d \times d \times d} \times \mathbb{R}^{k_y \times k_x \times k_z} \to \mathbb{R}^{(d-k_y+1) \times (d-k_x+1) \times (d-k_z+1)}, \quad (3)$$

where

$$(X * K)_{i,j,q} = \sum_{m=0}^{k_y-1} \sum_{n=0}^{k_x-1} \sum_{r=0}^{k_z-1} K_{m,n,r} X_{i+m,j+n,q+r}. \quad (4)$$

A convolution network with a bias parameter:

$$\begin{aligned} Z &= b^1 + X * K, \\ H &= \sigma(Z), \\ U_k &= W_{k,:,:} \cdot H + b^2_k, \quad k = 0, \ldots, K-1, \\ f(x;\theta) &= F_{\mathrm{softmax}}(U). \end{aligned} \tag{5}$$

Multi-layer convolution networks:

- The input image is $X \in \mathbb{R}^{d \times d \times C^0}$.
- The ℓ-th convolution layer contains C^ℓ “feature maps”.
- The number of feature maps C^ℓ is often called the “number of channels” for layer ℓ.
- The ℓ-th hidden layer is $H^\ell \in \mathbb{R}^{d_y^\ell \times d_x^\ell \times C^\ell}$. The first feature map $H^0 = X$.
- The filters for the ℓ-layer are given by the variable $K^\ell \in \mathbb{R}^{d_y^\ell \times d_x^\ell \times C^\ell \times C^{\ell-1}}$.

$$H^\ell_{i,j,p} = \sigma\left(\sum_{p'=0}^{C^{\ell-1}-1} \sum_{m=0}^{k_y^\ell - 1} \sum_{n=0}^{k_x^\ell - 1} K^\ell_{m,n,p,p'} H^{\ell-1}_{i+m,j+n,p'} \right). \tag{6}$$

The height d_y^ℓ and width d_x^ℓ of the feature maps in the ℓ-th layer depend upon the height $d_y^{\ell-1}$ and width $d_x^{\ell-1}$ of the feature maps in the previous layer and the size of the filters $k_y^\ell \times k_x^\ell$:

$$\begin{aligned} d_y^\ell &= d_y^{\ell-1} - k_y^\ell + 1, \\ d_x^\ell &= d_x^{\ell-1} - k_x^\ell + 1. \end{aligned} \tag{7}$$

Padding

Expand the matrices $H^{\ell-1}_{:,:,p}$ by adding P zeros on all sides to form a larger tensor

$$\hat{H}^{\ell-1} \in \mathbb{R}^{(d_y^{\ell-1}+2P)\times(d_x^{\ell-1}+2P)\times C^{\ell-1}}. \tag{8}$$

$$H^{\ell}_{i,j,p} = \sigma\left(\sum_{p'=0}^{C^{\ell-1}-1}\sum_{m=0}^{k_y^{\ell}-1}\sum_{n=0}^{k_x^{\ell}-1} K^{\ell}_{m,n,p,p'}\hat{H}^{\ell-1}_{i+m,j+n,p'}\right). \tag{9}$$

H^{ℓ} therefore has dimensions

$$(d_y^{\ell-1} - k_y^{\ell} + 2P + 1) \times (d_x^{\ell-1} - k_x^{\ell} + 2P + 1) \times C^{\ell}. \tag{10}$$

Strides

A convolution layer with a stride s is

$$H^{\ell}_{i,j,p} = \sigma\left(\sum_{p'=0}^{C^{\ell-1}-1} \sum_{m=0}^{k^{\ell}_y-1} \sum_{n=0}^{k^{\ell}_x-1} K^{\ell}_{m,n,p,p'} H^{\ell-1}_{is+m,js+n,p'} \right). \tag{11}$$

H^{ℓ} therefore has dimensions

$$\left(\left\lfloor \frac{d^{\ell-1}_y - k^{\ell}_y}{s} \right\rfloor + 1 \right) \times \left(\left\lfloor \frac{d^{\ell-1}_x - k^{\ell}_x}{s} \right\rfloor + 1 \right) \times C^{\ell}. \tag{12}$$

Padding and Strides

$$H^{\ell}_{i,j,p} = \sigma\left(\sum_{p'=0}^{C^{\ell-1}-1} \sum_{m=0}^{k_y^{\ell}-1} \sum_{n=0}^{k_x^{\ell}-1} K^{\ell}_{m,n,p,p'} \hat{H}^{\ell-1}_{is+m,js+n,p'} \right). \tag{13}$$

H^{ℓ} has dimensions

$$\left(\left\lfloor \frac{d_y^{\ell-1} - k_y^{\ell} + 2P}{s} \right\rfloor + 1 \right) \times \left(\left\lfloor \frac{d_x^{\ell-1} - k_x^{\ell} + 2P}{s} \right\rfloor + 1 \right) \times C^{\ell}. (14)$$

Pooling:

- A form of downsampling
- Some invariance to local translation

Average pooling with size h and stride s:

$$V_{i,j,p} = \frac{1}{h^2} \sum_{m=0}^{h-1} \sum_{n=0}^{h-1} H_{is+m,js+n,p}. \tag{15}$$

Max pooling with size h and stride s:

$$V_{i,j,p} = \max_{0 \leq m,n \leq h} H_{is+m,js+n,p}. \tag{16}$$

- Example of overlapping pooling: $s = 2$, $h = 3$.

- Example of non-overlapping pooling: $s = 2$, $h = 2$.

- If the input to a pooling layer has dimensions $d_y \times d_x \times C$, then the output has dimensions

$$\left(\left\lfloor \frac{d_y - h}{s} \right\rfloor + 1\right) \times \left(\left\lfloor \frac{d_x - h}{s} \right\rfloor + 1\right) \times C. \tag{17}$$

Dropout

$$\begin{aligned} Z^1 &= W^1 X + b^1, \\ Z^\ell &= W^\ell H^{\ell-1} + b^\ell, \quad \ell = 1, \ldots, L, \\ H^\ell &= R^\ell \odot \sigma(Z^\ell), \quad \ell = 1, \ldots, L, \\ U &= W^{L+1} H^L + b^{L+1}, \\ f(X, R; \theta) &= F_{\mathrm{softmax}}(U) \end{aligned} \tag{18}$$

- $R^\ell \in \mathbb{R}^N$ is a vector of independent Bernoulli random variables with parameter p.
- Let $R = \{R^1, \ldots, R^L\}$. R is sometimes called a "mask".
- The vector R^ℓ removes a random subset of the hidden units in layer ℓ from the model. This is a form of **regularization** since it reduces the model complexity.

At each update step, a random data sample (X, Y) is drawn and a mask R is randomly generated. The stochastic gradient descent algorithm **with dropout** is:

- Randomly select a data sample (X, Y)
- Generate a random sample R
- Calculate the gradient

$$G^{(k)} = \nabla_{\theta}\rho\bigg(f(X, R; \theta), Y \bigg).$$

- Take a stochastic gradient descent step

$$\theta^{(k+1)} = \theta^{(k)} - \alpha^{(k)} G^{(k)}. \tag{19}$$

The dropout algorithm minimizes the objective function

$$\mathcal{L}(\theta) = \mathbb{E}_{(X,Y),R}\left[\rho\left(f(X,R;\theta),Y\right)\right]. \tag{20}$$

Note that, similar to (X,Y), the samples of R are i.i.d.

Therefore, under suitable technical conditions for the model f,

$$\mathbb{P}\left[\lim_{k\to\infty}\|\nabla_\theta\mathcal{L}(\theta)\| = 0\right] = 1. \tag{21}$$

- Dropout minimizes the "average loss" from a collection of models.

- The number of models grows exponentially with the total number of the hidden units $L \times d_H$.

- Therefore, directly optimizing over $\mathbb{E}_R\Big[\rho\Big(f(X, R; \theta), Y\Big)\Big]$ (for a single sample of X, Y) is not feasible.

- Instead, we also apply stochastic gradient descent to the collection of models, i.e. we sample 1 specific model at each training step.

What model to use for predictions (i.e., for the test dataset) ?

A heuristic algorithm is typically used to address this problem.

The random variable R^ℓ in (18) is replaced with $\mathbb{E}[R^\ell] = (p, \ldots, p)$.

$$\begin{aligned} Z^1 &= W^1 X + b^1, \\ Z^{\ell+1} &= W^\ell H^\ell + b^\ell, \quad \ell = 1, \ldots, L-1, \\ H^\ell &= p\sigma(Z^\ell), \quad \ell = 1, \ldots, L, \\ U &= W^L H^L + b^L. \\ f_{\text{prediction}}(X; \theta) &= F_{\text{softmax}}(U). \end{aligned} \tag{22}$$

This is heuristic, since it is equivalent to interchanging an expectation and a nonlinear function. The prediction rule (22) corresponds to a different objective function:

$$\mathcal{L}_{\text{prediction}}(\theta) = \mathbb{E}_{(X,Y)}\left[\rho\bigg(f(X, \mathbb{E}[R]; \theta), Y \bigg)\right]. \tag{23}$$

Since neural networks are nonlinear:

$$\begin{aligned} \mathcal{L}_{\text{prediction}}(\theta) &\neq \mathbb{E}_{(X,Y,R)}\left[\rho\bigg(f(X, R; \theta), Y \bigg)\right] \\ &= \mathcal{L}(\theta). \end{aligned} \tag{24}$$

Nonetheless, the prediction network (22) has proven effective in practice for many applications.

- Alternatively, a more mathematically correct approach would be Monte Carlo simulation.
- Monte Carlo simulation is more computationally costly.
- See Hinton's original paper on dropout for a comparison.

- (Stochastic) gradient descent is a first-order optimization method, i.e. it only uses the gradient and not the Hessian.

- Why not use second-order optimization such as Newton's method?

$$\theta^{(\ell+1)} = \theta^{\ell} - \left(H^{(\ell+1)}\right)^{-1} g^{(\ell)}. \tag{25}$$

- Second-order optimization builds a **quadratic** model of the local landscape, while first-order optimization builds a **linear** model.

- Consequently, second-order algorithms can take larger steps.

$$\theta^{(\ell+1)} = \theta^{\ell} - \left(H^{(\ell+1)}\right)^{-1} g^{(\ell)}.$$

There is no stochastic gradient descent scheme for second-order optimization..Why?

To accurately approximate $\left(H^{(\ell+1)}\right)^{-1} g^{(\ell)}$, a large mini-batch is required.

$$\theta^{(\ell+1)} = \theta^{\ell} - \left(H^{(\ell+1)}\right)^{-1} g^{(\ell)}. \tag{26}$$

Two interesting features that are absent from gradient descent:

- Adaptive learning rate
- Affine invariance

- AdaGrad
- RMSprop
- ADAM

RMSprop and ADAM are widely used in practice.

Data augmentation:

- Randomly flip images
- Randomly rotate images
- Random crop/ Resize (i.e., rescale)
- Randomly adjust brightness or contrast

- Dataset A: very large dataset.
- Dataset B: **smaller dataset** for which we want a predictive model.
- **Transfer learning**: Train the model on Dataset A, then “fine-tune” on Dataset B.
- This can help build large models with less overfitting!

Parameter initialization.

Let d^{ℓ} be the number of hidden units in the ℓ-th layer. The weight $W_{i,j}^{\ell}$ is typically initialized to

$$\frac{C}{\sqrt{d^{\ell-1}}} U_{i,j}, \tag{27}$$

where $U_{i,j}$ are independent standard normal RVs or uniform RVs.

If $H_i^{\ell-1}$ are mean zero and have variance 1, the input to the ℓ-th layer

$$Z_i = \sum_{j=0}^{d^{\ell-1}-1} \frac{C}{\sqrt{d^{\ell-1}}} U_{i,j} H_j^{\ell-1}. \tag{28}$$

Suppose $U_{i,j}$ are independent, standard normal RVs and $C = 1$. Then,

$$\begin{aligned} \mathbb{E}[Z_i] &= 0, \\ \mathrm{Var}[Z_i] &= 1. \end{aligned} \tag{29}$$

Linearization.

If the hidden units are tanh$(\cdot)$ and $W^\ell H^{\ell-1}$ is "small" (let $b^\ell = 0$ to simplify analysis),

$$\begin{aligned} H^\ell &= \sigma(W^\ell H^{\ell-1}) \\ &\approx W^\ell H^{\ell-1}. \end{aligned} \tag{30}$$

Let $C = 1$, $U_{i,j}$ are indep. standard normal RVs, and W^ℓ are initialized according to (27). Suppose X_i are mean zero and have variance 1. Then,

$$\begin{aligned} Z_i^\ell &= \sum_{j=0}^{d^{\ell-1}-1} W_{i,j}^\ell H_j^{\ell-1}, \\ \mathbb{E}[Z_i^\ell] &= 0, \\ \mathrm{Var}[Z_i^\ell] &= 1. \end{aligned} \tag{31}$$

Batch normalization: A normalization which is *learned* during training.

$$\begin{aligned} H^\ell &= \sigma\Big(\mathrm{BN}_\theta[Z^\ell]\Big), \\ Z^\ell &= W^\ell H^{\ell-1} + b^\ell. \end{aligned} \tag{32}$$

The goal is to learn a transformation such that

$$\begin{aligned} \mathbb{E}_X\Big[\mathrm{BN}_{\theta(k)}[Z^\ell]\Big] &= 0, \\ \mathrm{Var}_X\Big[\mathrm{BN}_{\theta(k)}[Z_i^\ell]\Big] &= 1. \end{aligned} \tag{33}$$

Consider

$$\hat{Z}_i^\ell = \frac{Z_i^\ell - \mathbb{E}_X[Z_i^\ell]}{\sqrt{\mathrm{Var}_X[Z_i^\ell]}}. \tag{34}$$

Then, we of course have that $\mathbb{E}_X[\hat{Z}_i^\ell] = 0$ and $\mathrm{Var}_X[\hat{Z}_i^\ell] = 1$.

However, expectations over the entire dataset are too computationally expensive, so we instead normalize over the mini-batch of data samples $x^1, \ldots, x^M$.

Let

$$\begin{aligned}\mu_B &= \frac{1}{M}\sum_{i=1}^{M} Z^{\ell,i}, \\ \sigma_B^2 &= \frac{1}{M}\sum_{i=1}^{M}(Z^{\ell,i} - \mu_{B,i})^2. \end{aligned} \tag{35}$$

Z^ℓ is then normalized using these mini-batch statistics:

$$\hat{Z}_i^\ell = \frac{Z_i^\ell - \mu_{B,i}}{\sqrt{\sigma_{B,i}^2 + \epsilon}}. \tag{36}$$

The final Batch Normalization transformation is

$$\mathrm{BN}_\theta(Z^\ell) = \gamma \odot \hat{Z}^\ell + \beta, \tag{37}$$

where (γ, β) are additional parameters.

- $\mathrm{BN}_\theta(\cdot)$ is a function of $Z^{\ell,1}, \dots, Z^{\ell,M}$, and therefore **depends upon both θ and $x^1, \dots, x^M$.**

- See original paper "Batch normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" by Ioffe and Szegedy (2015) for modified backpropagation rule.

- Batch normalization simultaneously learns (1) the best normalization and (2) the best network parameters.

- The normalization depends upon the network parameters.

RMSprop

$$\begin{aligned} r &= \rho r + (1-\rho) g \odot g, \\ \Delta\theta &= -\frac{\eta}{\sqrt{\delta + r}} \odot g, \\ \theta &= \theta + \Delta\theta. \end{aligned} \tag{38}$$

ADAM

$$\begin{aligned} s &= \rho^1 s + (1-\rho^1) g, \\ r &= \rho^2 r + (1-\rho^2) g \odot g \\ \hat{s} &= \frac{s}{1-(\rho^1)^k}, \\ \hat{r} &= \frac{r}{1-(\rho^2)^k}, \\ \Delta\theta &= -\eta \frac{\hat{s}}{\delta + \sqrt{\hat{r}}}, \\ \theta &= \theta + \Delta\theta. \end{aligned} \tag{39}$$

Overfitting.

- Dropout
- Transfer learning
- ℓ^2 penalty
- Ensembles
- Early stopping

Example of convolution network for CIFAR10:

- Convolution layer 1: 64 channels, $k = 4, s = 1, P = 2$.
- Batch normalization
- Convolution layer 2: 64 channels, $k = 4, s = 1, P = 2$.
- Max Pooling: $s = 2$, $k = 2$.
- Dropout
- Convolution layer 3: 64 channels, $k = 4, s = 1, P = 2$.
- Batch normalization
- Convolution layer 4: 64 channels, $k = 4, s = 1, P = 2$.
- Max Pooling → Dropout

(Continued)

- Convolution layer 5: 64 channels, $k = 4, s = 1, P = 2$.
- Batch normalization
- Convolution layer 6: 64 channels, $k = 3, s = 1, P = 0$.
- Dropout
- Convolution layer 7: 64 channels, $k = 3, s = 1, P = 0$.
- Batch normalization
- Convolution layer 8: 64 channels, $k = 3, s = 1, P = 0$.
- Batch normalization, Dropout
- Fully connected layer 1: 500 units.
- Fully connected layer 2: 500 units.
- Linear → Softmax function