CS-667 Advanced Machine Learning

Nazar Khan

PUCIT

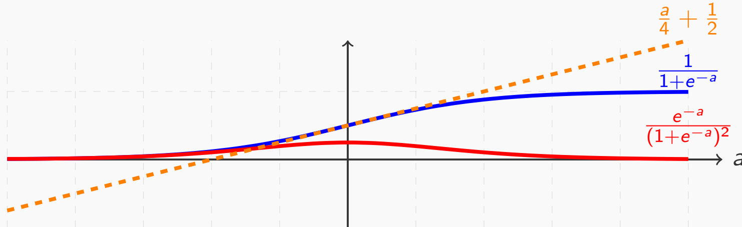
Deep Learning

Lecture slides assisted by Arbish Akram.

Deep Learning

- *Deep network*: A neural network with more than 1 hidden layers.
- Why use a fancy new name?
  - Standard methods of training neural networks are not useful when a network becomes deep.
  - *Main problem:* reduced flow of information through the layers (forward, backward or both).
  - Lead to slow or no learning at all, especially for the earlier layers.
- Three main innovations.
  1. Better activation functions.
  2. Better weight initializations.
  3. Better normalization of inputs.
- *Overall goal:* Maintain forward as well as backward flow of information.

Problems with sigmoidal neurons



- For large $|a|$, sigmoid value approaches either 0 or 1. This is called *saturation*.
- When the sigmoid saturates, the gradient approaches zero.
- Neurons with sigmoidal activations stop learning when they saturate.
- When they are not saturated, they are almost linear.
- There is another reason for the gradient to approach zero during backpropagation.

Vanishing Gradients

- Notice that gradient of the sigmoid is always between 0 and $\frac{1}{4}$.
- Now consider the backpropagation equation.

$$\delta_j = \underbrace{h'(a_j)}_{\leq \frac{1}{4}} \sum_{k=1}^{K} w_{kj}\delta_k$$

where δ_k will also contain *at least* one factor of $\leq \frac{1}{4}$.

- This means that values of δ_j keep getting smaller as we backpropagate towards the early layers.
- Since gradient $= \delta \times$input, the gradients also keep getting smaller for the earlier layers. Known as the *vanishing gradients* problem.
- *Therefore, while the network might be deep, learning will not be deep.*

1. Better Activation Functions

Name	Formula	Plot	Derivative	Comments
Logistic sigmoid	$\frac{1}{1+e^{-a}}$		$f(a)(1-f(a))$	Vanishing gradients
Hyperbolic tangent	$\tanh(a)$		$1-\tanh^2(a)$	Vanishing gradients
Rectified Linear Unit (ReLU)	$\begin{cases} a & \text{if } a>0 \\ 0 & \text{if } a \le 0 \end{cases}$		$\begin{cases} 1 \\ 0 \end{cases}$	Dead neurons. Sparsity.
Leaky ReLU	$\begin{cases} a & \text{if } a>0 \\ ka & \text{if } a \le 0 \end{cases}$		$\begin{cases} 1 \\ k \end{cases}$	$0<k<1$
Exponential Linear Unit (ELU)	$\begin{cases} a & \text{if } a>0 \\ k(e^a-1) & \text{if } a \le 0 \end{cases}$		$\begin{cases} 1 \\ f(a)-k \end{cases}$	$k>0$.

- Saturated sigmoidal neurons stop learning. Piecewise-linear units keep learning by avoiding saturation.
- ELU leads to better accuracy and faster training.
- *Take home message*: Use a member of the LU family. They avoid *i*) saturation and *ii*) the vanishing gradient problem.

2. Initialization of weights

- When all weights are initialized to the same value,
  - every neuron will output the same value,
  - all gradients will also be the same, and so
  - all neurons will learn the same thing.
- *Standard method*: initialize using $w \sim \mathcal{N}(0, 1)$. That is, every weight is initialized as a random value from the standard normal distribution.
- *Ideally*, for any neuron we would want the variance of output to be the same as the variance of inputs.
  - Why? So that the signal neither vanishes nor explodes after going through the neuron.
  - We would like this to be true for the forward as well as the backward pass.
- This can be approximately achieved via Xavier initialization.

Xavier Initialization

- Very useful for deep networks.
- Let n_i be the number of weights feeding into a neuron and n_o be the number of neurons that it feeds its output to.
- For weights of sigmoidal neurons, initialize[1] $w \sim \mathcal{N}(0, \frac{1}{n_i})$ or $w \sim \mathcal{N}(0, \frac{2}{n_i+n_o})$.
- Ensures variance of outputs is same as variance of inputs so that propagation through the network does not shrink or explode the signal.
- For weights of neurons from the LU family, initialize[2] $w \sim \mathcal{N}(0, \frac{2}{n_i})$.

[1] Glorot and Bengio, 'Understanding the difficulty of training deep feedforward neural networks'.

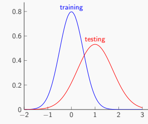
[2] He et al., 'Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification'.

So far . . .

- Better activation functions let the gradients remain alive.
- Better weight initializations let the signals as well as the gradients remain alive. But for how long?
- So it seems as if the main goal is to let forward and backward information remain alive.
- Can this be achieved in some other way?

3. Better normalization of inputs

- We have already seen the importance of normalizing inputs to shallow ML architectures.
- When distribution of input changes (covariate shift), weights need to be adapted.



- For deep architectures, the normalization problem becomes much more serious.
  - Change in $\mathbf{w}^{(l-1)}$ causes a change in distribution of inputs to layer l *and beyond*.
  - This forces $\mathbf{w}^{(l)}$ to adapt to the new distribution.
  - This leads to longer training times.
- It would be convenient if the input distribution of each neuron in each layer could remain stable.

Batch Normalization

- *Solution*: Inputs to every neuron in every layer must be normalized *in a differentiable manner*.
  - Normalization is useless if gradient ignores it.
- Batch normalization[3].
  - Avoids vanishing gradients for sigmoidal non-linearities.
  - Allows much higher learning rates and therefore dramatically speeds up training.
  - Reduces dependence on good weight initialization.
  - Regularizes the model and reduces the need for dropout.

[3] Ioffe and Szegedy, 'Batch normalization: Accelerating deep network training by reducing internal covariate shift'.

Normalization vs. Whitening

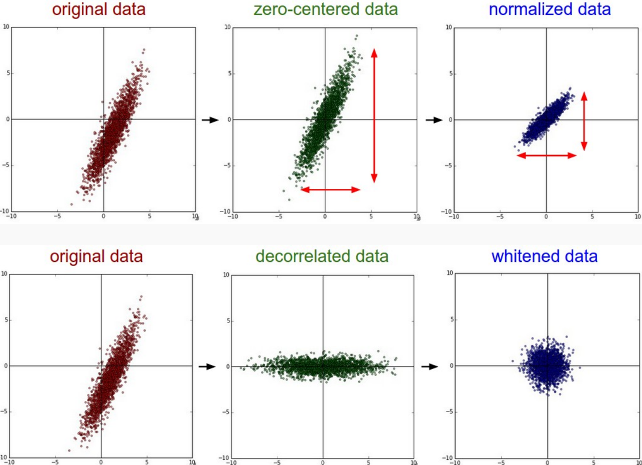


Figure: Normalization versus whitening. Taken from
`http://cs231n.github.io/neural-networks-2/`

Batch Normalization

- It is well-known that whitened inputs lead to faster convergence[4]. However, whitening can be expensive (due to inverse of covariance matrix).
- Normalization over a mini-batch $\mathcal{B}$ is a cheaper alternative.

$$\hat{x} = \frac{x - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}}} \tag{1}$$

where $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}$ are the mean and standard deviation of the neuron's activations over a batch $\mathcal{B}$.

[4] LeCun et al., 'Efficient backprop'.

Batch Normalization

- However, such normalization can cause $\hat{x}$ to lie close to the linear parts of a sigmoidal function.
- So we allow for scaling and shifting of $\hat{x}$ to obtain the identity transformation.

$$y = \gamma \hat{x} + \beta \tag{2}$$

where γ and β are trainable network parameters.

- The whole process is differentiable and therefore suitable for gradient descent.

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Please note that slightly different normalization is performed at test time where there are no batches.

Helpful Resources

- http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf
- http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf
- https://www.youtube.com/watch?v=nUUqwaxLnWs