Sane's STATS Academy of Statistics

Training Deep Networks

Enhancing the Performance

Sane's STATS Academy of Statistics

Problems faced in Deep Neural Networks

Problems

- Vanishing / Exploding Gradient Descent
- Training speed extremely slow

Remedies Suggested

- Trying different initializations and activations
- Trying some fast optimizers

Vanishing / Exploding Gradient Descent

- Gradient Descent, as we know is necessary for calculating the change in weights across iterations
- In deep networks, the gradients get smaller and smaller as the algorithm progresses down to the lower (beginning) layers, as a result of which the weights of lower layers remain literally unchanged. This problem is called problem of *vanishing gradient descent*.
- In some cases, gradients grow bigger and bigger up to the lower layers making the weight updates horribly large. This problem is called problem of *exploding gradient descent*.

Sane's STATS Academy of Statistics

Different Initializations

- Glorot Initialization
- He Initialization
- LeCun Initialization

Glorot Initialization

- Glorot and Bengio, in their research paper recommended this algorithm
- They pointed out that the signal should flow properly in both the directions in the neural network, without gradient vanishing and also exploding
- The authors of this paper argued that for signal to flow properly, the variance of inputs should be equal to variance of outputs, also gradients should have equal variances before and after flowing through the layer

Glorot Initialization Function

- Glorot and Bengio defined one term $fan_{avg} = \frac{fan_{in}+fan_{out}}{2}$, where

  fan_{in} : No. of input neurons of the layer

  fan_{out}: No. of output neurons of the layer

- For logistic activation, they proposed initialization as:
  - Random numbers from Normal($\mu = 0, \sigma^2 = \frac{1}{fan_{avg}}$) or
  - Random numbers from Uniform(a=-r, b=r) with $r = -\sqrt{\frac{3}{fan_{avg}}}$

Sane's STATS Academy of Statistics

Popular Initializations

- Papers have provided similar strategies for initialization except the change in variance

Initialization	Activation Functions	σ^2 (Normal Distribution)
Glorot	None, tanh, logistic, softmax	$\frac{1}{fan_{avg}}$
He	ReLU and its variants	$\frac{2}{fan_{in}}$
LeCun	SELU	$\frac{1}{fan_{in}}$

Sane's STATS Academy of Statistics

Specifying initializations in keras

```
tf.keras.initializers.GlorotNormal(
    seed=None
)
```

default →

```
tf.keras.initializers.GlorotUniform(
    seed=None
)
```

```
tf.keras.initializers.he_uniform(
    seed=None
)
```

```
tf.keras.initializers.he_normal(
    seed=None
)
```

```
tf.keras.initializers.lecun_normal(
    seed=None
)
```

```
tf.keras.initializers.lecun_uniform(
    seed=None
)
```

Non-saturating Activation Functions

- Sigmoid function has been the first choice, but many a times it is known to saturate for positive values
- Better than sigmoid function, RELU definitely is better, but even RELU saturates with its output values as zero

Leaky RELU

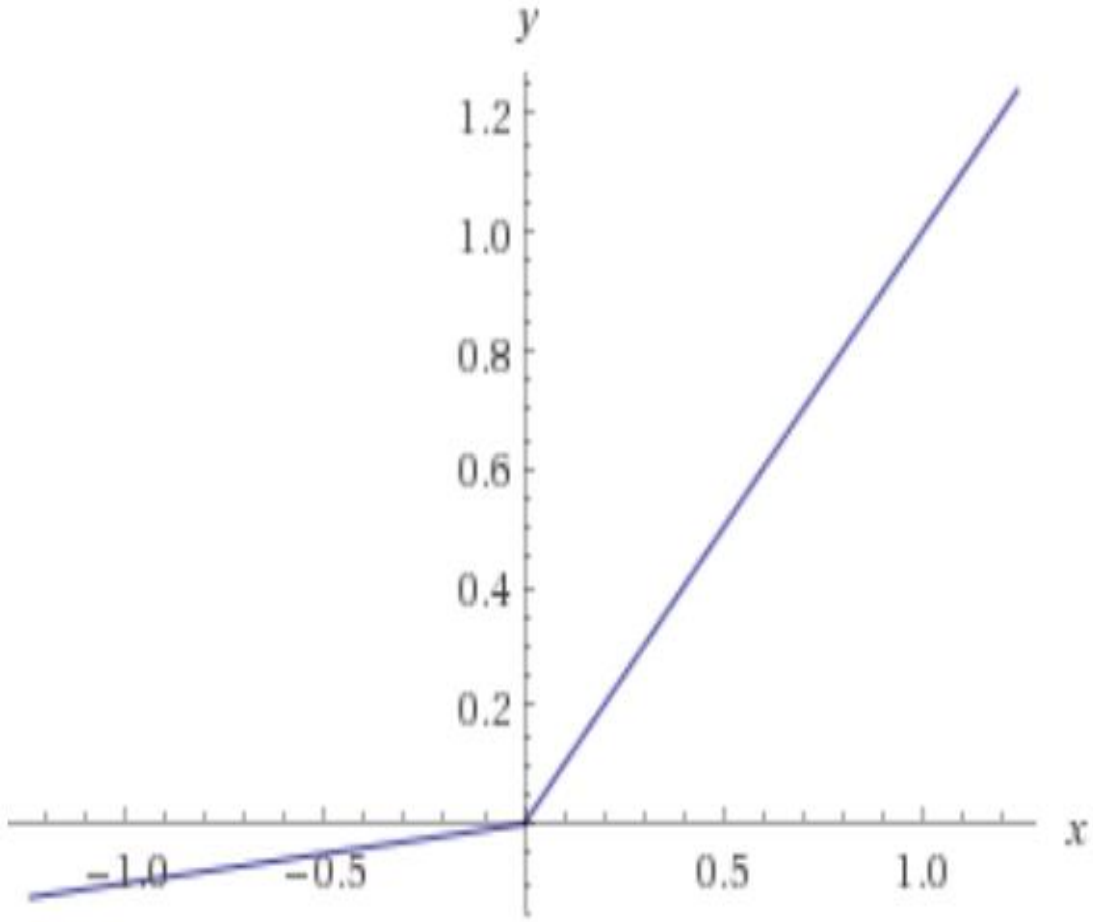
- For solving this problem, a variant of ReLU, Leaky ReLU can be used

$$LeakyReLU_{\alpha}(z) = \max(\alpha z, z)$$

- This ensures that the activation rarely evaluates to zero and if it gets less than zero then it does not go too far away from zero

- A variant of Leaky ReLU, called Randomised Leaky ReLU(RReLU), is there in which expression remains same, but α is chosen randomly
- Parametric Leaky ReLU (PReLU)is such that it uses the same expression as above but the α is learnt by backpropagation

(Proposed by Bing Xu, Naiyan Wang, Tianqi Chen, Mu Li in 2015)



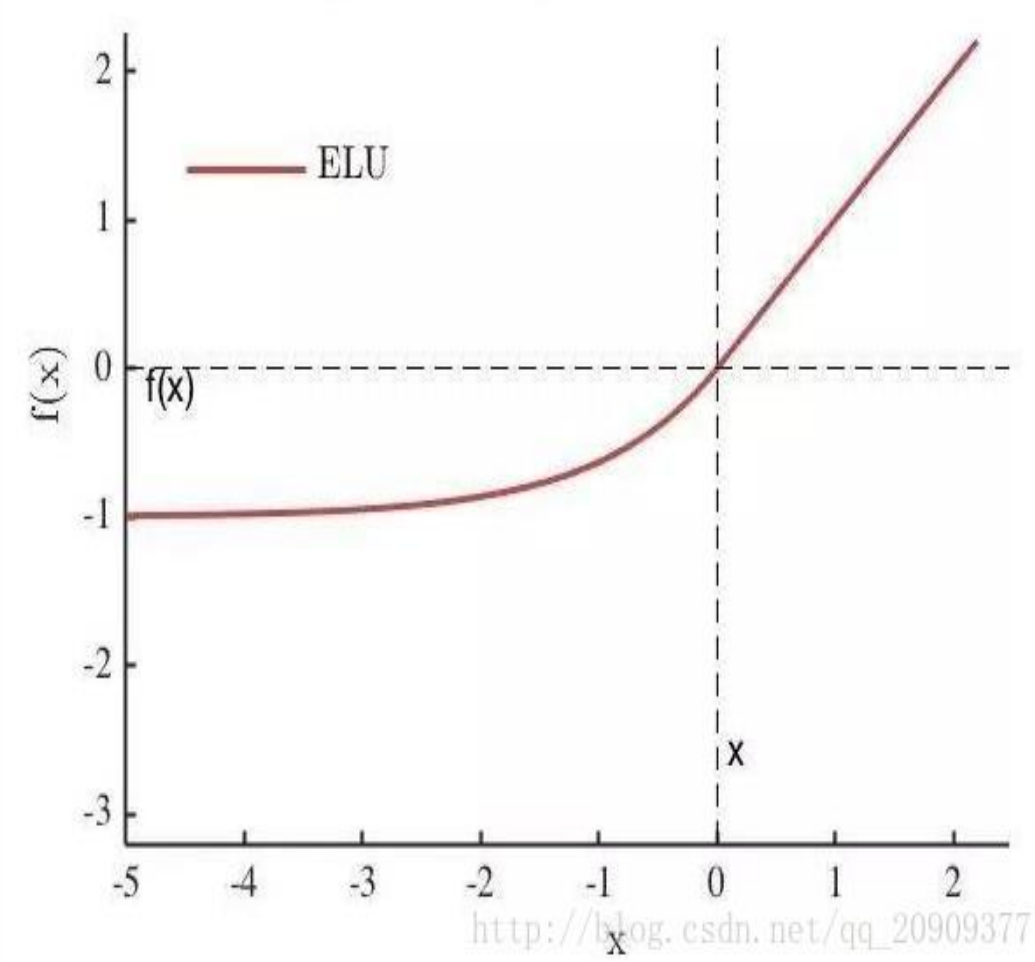
https://arxiv.org/pdf/1505.00853.pdf

ELU

- Djork-Arné Clevert, Thomas Unterthiner, Sepp Hochreiter in 2015 proposed a new activation function called exponential linear unit(ELU) that was better in performance than all ReLU variants
- The authors in their paper claimed that ELU reduces the training time, and performance is better on test set

$$ELU_\alpha(z) = \begin{cases} \alpha(e^z - 1), if\ z < 0 \\ z \qquad\qquad , if\ z \geq 0 \end{cases}$$

https://arxiv.org/pdf/1511.07289.pdf



SELU

- ELU is slower to calculate than ReLU
- Günter Klambauer, Thomas Unterthiner, Andreas Mayr, Sepp Hochreiter in 2017 proposed scaled exponential linear unit (SELU)
- The authors proposed that if all the hidden layers of deep neural network use SELU, then the network with self-normalize i.e. the output of each layer will tend to preserve a mean 0 and standard deviation 1 during training, which should solve the vanishing / exploding gradient problem.
- For self-normalization:
  - The input features must be standardized
  - Every hidden layer's weight must be initialized using LeCun normal initialization
  - Network must be sequential

https://arxiv.org/pdf/1706.02515.pdf

Sane's STATS Academy of Statistics

Fast Optimizers

- Apart from the regular gradient descent optimizer, there are popular optimizers:
  - Momentum Optimization
  - Nesterov Accelerated Gradient
  - AdaGrad
  - RMSProp
  - Adam

Momentum Optimization

- In this optimization, there is a term momentum (m), which is updated with learning rate and a constant β

$$m := \beta m - \eta \frac{\partial}{\partial w} J(w)$$

$$w := w + m$$

- This optimization is faster than ordinary gradient descent

Sane's STATS Academy of Statistics

Nesterov Accelerated Gradient

- This optimizer is a modification in momentum optimization
- Also known as Nesterov Momentum Optimization

$$m := \beta m - \eta \frac{\partial}{\partial w} J(w + \beta m)$$
$$w := w + m$$

AdaGrad

- AdaGrad algorithm Gradient Descent towards global minimum

$$s := s + \frac{\partial}{\partial w} \mathrm{J(w)} \otimes \frac{\partial}{\partial w} \mathrm{J(w)}$$

$$w := w - \eta \frac{\partial}{\partial w} \mathrm{J(w)} \oslash \sqrt{s + \epsilon}$$

- Here, $\otimes$ means element-wise multiplication and $\oslash$ means element-wise division.
- This is the vectorized form for all the parameters w_i
- This has been named as AdaGrad because of its property of adaptive learning rate

RMSProp

- AdaGrad has a risk of slowing down a bit too fast and never converging to the global minimum
- Root Mean Square Propagation(RMSProp) fixes this problem by accumulating only gradients from the most recent iterations

$$s := \beta s + (1-\beta)\frac{\partial}{\partial w}\mathrm{J(w)} \otimes \frac{\partial}{\partial w}\mathrm{J(w)}$$

$$w := w - \eta \frac{\partial}{\partial w}\mathrm{J(w)} \oslash \sqrt{s+\epsilon}$$

- β usually is set to 0.9 and rarely requires to be tuned

Adam

- Adaptive Moment Estimation (Adam) combines the ideas of momentum optimization and RMSProp

$$m \coloneqq \beta_1 m - (1-\beta_1)\frac{\partial}{\partial w}\mathrm{J(w)}$$

$$s \coloneqq \beta_2 s + (1-\beta_2)\frac{\partial}{\partial w}\mathrm{J(w)} \oslash \frac{\partial}{\partial w}\mathrm{J(w)}$$

$$\hat{m} \coloneqq \frac{m}{(1-\beta_1^t)} \quad , \quad \hat{s} \coloneqq \frac{s}{(1-\beta_2^t)}$$

$$w \coloneqq w + \eta\hat{m} \oslash \sqrt{\hat{s}+\varepsilon}$$

Sane's STATS Academy of Statistics

Batch Normalization

Need for Batch Normalization

- Although there are different options for initialization and optimizers present, still these options don't guarantee to solve the problem of vanishing / exploding gradient descent
- Batch Normalization (BN) technique may solve the problem
- The technique consists of a mathematical operation just before or after the activation function of each hidden layer
- BN zero-centres and normalizes each input using centring and scaling
- It does it in a type of $\frac{X-\mu}{\sigma}$ way over all the input of the hidden layer
- It calculates mean and standard deviation of the input over a batch of observations

Sane's STATS Academy of Statistics

Batch Normalization in Keras

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(90,activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(70,activation='relu'),
    tf.keras.layers.BatchNormalization(),
…
```