Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization

dzlabs

September 2018

Week 1

Practical aspects of Deep Learning

In this week, you've learned about:

- how to set up your train, dev, and test sets,
- how to analyze bias and variance and what things to do if you have high bias versus high variance versus maybe high bias and high variance.
- how to apply different forms of regularization, like L2 regularization and dropout on your neural network.
- So some tricks for speeding up the training of your neural network.
- And then finally, gradient checking.

You get to exercise a lot of these ideas.

Setting up your Machine Learning Application

Train / Dev / Test sets Traditionally dataset is split into 70% training and 30% test set, or 60%/20%/20% but now a days as datasest are in the order of millions or more the percentage of dev/test sets is reduced to smaller proportions (e.g. 1%).
Bias / Variance

- high Bias when the algorithm is under-fitting (e.g. linear regression)can be fixed by using a NN of large number of units.
- high Variance when the algorithm is over-fitting (e.g. complex logistic) can be fixed by using more data or using regularization.
- high Bias high Variance is the worst kind of algorithms, e.g. a linear regression that's too flexible in some region (e.g. middle).

Basic Recipe for Machine Learning

Regularizing your neural network

Regularization
Why regularization reduces overfitting? if λ is large then $w^{[l]}$ will be small, then as a result $z^{[l]} = w^{[l]} * a^{[l-1]} + b^{[l]}$ will be small. If z is close to zero then the activation function (e.g. $tanh$) will be in the 0 region where it looks like a line.
i.e. the learning function is close to a linear regression which will avoid overfitting.
Dropout Regularization Another technique of regularization. For each training example, you would train it using one of these neural based networks where you have removed some units and all it's links. Different possible implementations, like ***Inverted Dropout*** which is the most used technique.
Understanding Dropout A regularization technique used to avoid over-fitting, used a lot in Computer Vision as you never have enough data and your input is very large (number of pixels).
One downside is that the Cost function is no longer well defined, as nodes can go away randomly. As a result, debugging the learning algorithm by plotting the cost function by iterations is no longer effective. One way to avoid this, is to run the algorithm with a dropout threshold set to 1 in order to keep all nodes then plot the cost function, then to learn the parameters reset the dropout to usual values.
Other regularization methods

Data augmentation Apply some functions to the existent data set to create new samples e.g. flipping a picture or applying a distortion to a digit.

Early stopping as you run gradient descent, you plot the cost function by iterations or training error in the training set. Plus you plot the dev-set error. The latter usually goes down then from certain point it start going up. What early stopping does is to stop training the NN halfway at a moment where the NN was doing good.

Weight decay after each update, the weights are multiplied by a factor slightly less than 1. This prevents the weights from growing too large, and can be seen as gradient descent on a quadratic regularization term.

Orthogonolisation: think about when task at a time. e.g. optimize the cost function, then find ways to avoid overfitting. The downside of ***Early stopping*** is that it's combinig those two steps. One tool that mixes the two processes.

Setting up your optimization problem

Normalizing inputs
Vanishing / Exploding gradients deep networks suffer from the problems of vanishing or exploding gradients. Which was for a long time this problem was a huge barrier to training deep neural networks. If the w parameters are small then they will get smaller, if they are initialized to greater than 1 then the composition will be too large. In both cases, the learning becomes slower. A solution is to carefully initialize the training parameters w.
Weight Initialization for Deep Networks
Numerical approximation of gradients
Gradient checking
Gradient Checking Implementation Notes

QUIZ - Practical aspects of deep learning

1. If you have 10,000,000 examples, how would you split the train/dev/test set?

- 98% train . 1% dev . 1% test (X)
- 33% train . 33% dev . 33% test
- 60% train . 20% dev . 20% test

2. The dev and test set should:

- Come from the same distribution (X)
- Come from different distributions
- Be identical to each other (same (x,y) pairs)
- Have the same number of examples

3. If your Neural Network model seems to have high variance, what of the following would be promising things to try?

- Make the Neural Network deeper
- Get more training data (X)
- Get more test data
- Add regularization (X)
- Increase the number of units in each hidden layer

4. You are working on an automated check-out kiosk for a supermarket, and are building a classifier for apples, bananas and oranges. Suppose your classifier obtains a training set error of 0.5%, and a dev set error of 7%. Which of the following are promising things to try to improve your classifier? (Check all that apply.)

- Increase the regularization parameter lambda (X)
- Decrease the regularization parameter lambda
- Get more training data (X)
- Use a bigger neural network

5. What is weight decay?

- The process of gradually decreasing the learning rate during training.
- A regularization technique (such as L2 regularization) that results in gradient descent shrinking the weights on every iteration. (X)
- Gradual corruption of the weights in the neural network if it is trained on noisy data.
- A technique to avoid vanishing gradient by imposing a ceiling on the values of the weights.

6. What happens when you increase the regularization hyperparameter lambda?

- Weights are pushed toward becoming smaller (closer to 0) (X)
- Weights are pushed toward becoming bigger (further from 0)
- Doubling lambda should roughly result in doubling the weights
- Gradient descent taking bigger steps with each iteration (proportional to lambda)

7. With the inverted dropout technique, at test time:

- You apply dropout (randomly eliminating units) and do not keep the $\frac{1}{keep\_prob}$ factor in the calculations used in training
- You do not apply dropout (do not randomly eliminate units), but keep the $\frac{1}{keep\_prob}$ factor in the calculations used in training.
- You apply dropout (randomly eliminating units) but keep the $\frac{1}{keep\_prob}$ factor in the calculations used in training.
- You do not apply dropout (do not randomly eliminate units) and do not keep the $\frac{1}{keep\_prob}$ factor in the calculations used in training (X)

8. Increasing the parameter keep_prob from (say) 0.5 to 0.6 will likely cause the following: (Check the two that apply)

- Increasing the regularization effect (X)
- Reducing the regularization effect
- Causing the neural network to end up with a higher training set error (X)
- Causing the neural network to end up with a lower training set error

9. Which of these techniques are useful for reducing variance (reducing overfitting)? (Check all that apply.)

- Exploding gradient
- Dropout (X)
- Xavier initialization
- L2 regularization (X)
- Vanishing gradient
- Data augmentation (X)
- Gradient Checking

10. Why do we normalize the inputs xx?

- It makes it easier to visualize the data
- It makes the cost function faster to optimize (X)
- It makes the parameter initialization faster
- Normalization is another word for regularization–It helps to reduce variance

Week 2

Optimization algorithms

Algorithms to speedup Gradient Descent:

- Mini-Batch Gradient Descent
- Gradient Descent with Momentum
- RMSprop: Root Mean Square prop
- Adam optimization algorithm (Adaptive Moment Estimation)

Exponentially weighted averages

The formulas has an β as hyperparameters, θ_t temperature at day t, and V_{t-1} the average at day $t-1$:

$$V_t = \beta * V_{t-1} + (1-\beta) * \theta_t \tag{1}$$

V_t is approximatively averaged over last $\frac{1}{1-\beta}$ days. If we set β to

- 0.9 then the average is over 10 days.
- 0.98 the average is over 50 days and plot will be very smoothed to the right
- 0.5 then average over 2 days, and graph will adapt to current temperature

Bias correction

$$V_t^{corrected} = \frac{V_t}{1-\beta^t} \tag{2}$$

Gradient descent with momentum

Momentum, or Gradient descent with Momentum is almost always faster than the standard gradient descent algorithm. The basic idea is to compute an exponentially weighted average of your gradients, and then use that gradient to update your weights instead.
Implementation details, Hyperparameters α and β (most common value is 0.9, i.e. averaging over 10 days/iterations):

```
On iteration t:
  Compute dW, db on the current mini-batch
  v_{dW} = \beta v_{dW} + (1 - \beta) dW
  v_{db} = \beta v_{db} + (1 - \beta) db
  W = W - \alpha v_{dw}, b = b - \alpha v_{db}
```

RMSprop

The intuition about how this works. Recall that in the horizontal direction (i.e. the W direction) we want learning to go pretty fast. Whereas in the vertical direction (i.e the b direction), we want to slow down all the oscillations into the vertical direction.

So with this terms SdW an Sdb, we want SdW to be relatively small, so that here we're dividing by relatively small number. Whereas Sdb will be relatively large, so that here we're dividing yt relatively large number in order to slow down the updates on a vertical dimension.

Adam optimization algorithm

A combination of **Momentum** and **RMSprop**.

Learning rate decay

$$\alpha = \frac{1}{1 + decay\_rate * epoch\_num}\alpha_0 \tag{3}$$

Other learning rate decay methods:

The problem of local optima

In very high-dimensional spaces you're actually much more likely to run into a **saddle point** (like horse saddle) then the local optimum.

So it is unlikely to get stuck in a bad local optima. On the other hand, Plateaus can make learning slow. Algorithms like **Learning rate decay** can help avoiding getting stuck in plateaus.

QUIZ - Optimization algorithms

1. Which notation would you use to denote the 3rd layer's activations when the input is the 7th example from the 8th minibatch?

- $a^{[3]\{7\}(8)}$
- $a^{[8]\{7\}(3)}$
- $a^{[8]\{3\}(7)}$
- $a^{[3]\{8\}(7)}$ (X)

2. Which of these statements about mini-batch gradient descent do you agree with?

- Training one epoch (one pass through the training set) using mini-batch gradient descent is faster than training one epoch using batch gradient descent.
- You should implement mini-batch gradient descent without an explicit for-loop over different mini-batches, so that the algorithm processes all mini-batches at the same time (vectorization).
- One iteration of mini-batch gradient descent (computing on a single mini-batch) is faster than one iteration of batch gradient descent. (X)

3. Why is the best mini-batch size usually not 1 and not m, but instead something in-between?

- If the mini-batch size is m, you end up with stochastic gradient descent, which is usually slower than mini-batch gradient descent.
- If the mini-batch size is 1, you end up having to process the entire training set before making any progress.
- If the mini-batch size is 1, you lose the benefits of vectorization across examples in the mini-batch. (X)
- If the mini-batch size is m, you end up with batch gradient descent, which has to process the whole training set before making progress. (X)

4. Suppose your learning algorithm's cost JJ, plotted as a function of the number of iterations, looks like this: Which of the following do you agree with?

- If you're using mini-batch gradient descent, something is wrong. But if you're using batch gradient descent, this looks acceptable.
- Whether you're using batch gradient descent or mini-batch gradient descent, something is wrong.
- Whether you're using batch gradient descent or mini-batch gradient descent, this looks acceptable.
- If you're using mini-batch gradient descent, this looks acceptable. But if you're using batch gradient descent, something is wrong. (X)

5. Suppose the temperature in Casablanca over the first three days of January are the same:
Jan 1st: $\theta_1 = 10^oC$ Jan 2nd: $\theta_2 10^oC$
(We used Fahrenheit in lecture, so will use Celsius here in honor of the metric world.)

Say you use an exponentially weighted average with $\beta = 0.5$ to track the temperature: $v_0 = 0$, $v_t = \beta v_{t-1} + (1-\beta)\theta_t$. If v_2 is the value computed after day 2 without bias correction, and $v_2^{corrected}$ is the value you compute with bias correction. What are these values? (You might be able to do this without a calculator, but you don't actually need one. Remember what is bias correction doing.)

- $v_2 = 10$, $v_2^{corrected} = 7.5$
- $v_2 = 10$, $v_2^{corrected} = 10$
- $v_2 = 7.5$, $v_2^{corrected} = 10$ (X)
- $v_2 = 7.5$, $v_2^{corrected} = 7.5$

6. Which of these is NOT a good learning rate decay scheme? Here, t is the epoch number.

- $\alpha = \frac{1}{1+2*t}\alpha_0$
- $\alpha = \frac{1}{\sqrt{t}}\alpha_0$
- $\alpha = 0.95^t\alpha_0$
- $\alpha = e^t\alpha_0$ (X)

7. You use an exponentially weighted average on the London temperature dataset. You use the following to track the temperature: $v_t = \beta v_{t-1} + (1-\beta)\theta_t$. The red line below was computed using $\beta = 0.9$. What would happen to your red curve as you vary β? (Check the two that apply)

- Decreasing β will shift the red line slightly to the right.
- Increasing β will shift the red line slightly to the right. (X)
- Decreasing β will create more oscillation within the red line. (X)
- Increasing β will create more oscillations within the red line.

8. Consider this figure:
These plots were generated with gradient descent; with gradient descent with momentum ($\beta = 0.5$) and gradient descent with momentum ($\beta = 0.9$). Which curve corresponds to which algorithm?

- (1) is gradient descent. (2) is gradient descent with momentum (small β). (3) is gradient descent with momentum (large β) (X)
- (1) is gradient descent with momentum (small β), (2) is gradient descent with momentum (small β), (3) is gradient descent
- (1) is gradient descent with momentum (small β). (2) is gradient descent. (3) is gradient descent with momentum (large β)
- (1) is gradient descent. (2) is gradient descent with momentum (large β) . (3) is gradient descent with momentum (small β)

9. Suppose batch gradient descent in a deep network is taking excessively long to find a value of the parameters that achieves a small value for the cost function $\mathcal{J}(W^{[1]}, b^{[1]}, ..., W^{[L]}, b^{[L]})$. Which of the following techniques could help find parameter values that attain a small value for $\mathcal{J}$? (Check all that apply)

- Try initializing all the weights to zero
- Try mini-batch gradient descent (X)
- Try better random initialization for the weights
- Try tuning the learning rate α (X)
- Try using Adam (X)

10. Which of the following statements about Adam is False?

- The learning rate hyperparameter α in Adam usually needs to be tuned.
- Adam combines the advantages of RMSProp and momentum
- Adam should be used with batch gradient computations, not with mini-batches. (X)
- We usually use "default" values for the hyperparameters β_1, β_2 and ε in Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$

Week 3

Hyperparameter tuning, Batch Normalization and Programming Frameworks

Hyperparameter tuning

Tuning process some hyperparameters are more important than others, e.g. (alpha vs the rest). The two key takeaways are, use random sampling and adequate search and optionally consider implementing a coarse to fine search process.

Using an appropriate scale to pick hyperparameters It seems a bad idea to search for the hyperparameters in a linear scale, but more reasonable to search on a log scale. Where instead of using a linear scale, you'd have 0.0001 here, and then 0.001, 0.01, 0.1, and then 1. And you instead sample uniformly, at random, on this type of logarithmic scale. Now you have more resources dedicated to searching between 0.0001 and 0.001, and between 0.001 and 0.01, and so on. In python, it can be done as follows:

```
r = -4 * np.random.rand() // r in [-4, 0]
alpha = 10^r // randomly sampled
```

Hyperparameters tuning in practice: Pandas vs. Caviar panda: babysitting the model once at a time, or trying many models in parallel then picking the best one.

Normalizing activations in a network

Normalizing activations in a network normalize the hidden layer values as calculating the identity functions so instead of using $Z^{[l](i)}$ use $\tilde{Z^{[l](i)}}$ in future calculations. Where the later is calculated as follows:

$$\mu = \frac{1}{m}\sum_i Z^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_i (Z_i - \mu)^2$$

$$Z^{(i)}_{norm} = \frac{Z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$\tilde{Z^{(i)}} = \gamma Z^{(i)}_{norm} + \beta$$

α and β are learnable parameters of model.

Fitting Batch Norm into a neural network

Why does Batch Norm work?

Batch Norm at test time

Multi-class classification

Softmax Regression A generalization of the Logistic regression with more than two output classes. The unusual thing about the Softmax activation function is, because it needs to normalized across the different possible outputs, and needs to take a vector and puts in outputs of vector.

Training a softmax classifier You have to only implement forward propagation, tensorflow will know how to calculate derivities and perform backpropagation:

```
import numpy as np
import tensorflow as tf

# declare a parameter as a tensorflow variable
w = tf.Variable(0, dtype=tf.float32)
# declare the cost function that needs to be optimized
cost = tf.add(tf.add(w**2, tf.multiply(-10., w)), 25)
# define the training algorithm to be used,
# Gradient Descent (set the learning rate) with objective to minimize the cost
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)

```

```
init = tf.global_variables_initilizer()
# create tensorflow session
session = tf.Session()
session.run(init)
# at this moment nothing is run yet, so w is 0.0
print(session.run(w))

# run one step of Gradient Descent training
session.run()
print(session.run(w))

# run 1000 steps of Gradient Descent
for i in range(1000):
    session.run(train)
# now w should be close to 5 (which is the optimum)
print(session.run(w))
```

In the previous example, we were trying to minimize a fix function of w. The followning example shows how to get training data into tensor flow

```
import numpy as np
import tensorflow as tf

coefficient = np.array([[1.], [-10.], [25.]])
# declare a parameter as a tensorflow variable
w = tf.Variable(0, dtype=tf.float32)
x = tf.placehoder(tf.float32, [3, 1])
# declare the cost function that needs to be optimized
# replace cost = w**2 - 10 *w + 25 with
cost = x[0][0] * w ** 2 + x[1][0] * w + x[2][0]
# define the training algorithm to be used,
# Gradient Descent (set the learning rate) with objective to minimize the cost
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)

init = tf.global_variables_initilizer()
# create tensorflow session
session = tf.Session()
session.run(init)
# at this moment nothing is run yet, so w is 0.0
print(session.run(w))

# run one step of Gradient Descent training
session.run(train, feed_dict={x: coefficients})
print(session.run(w))

# run 1000 steps of Gradient Descent
for i in range(1000):
    session.run(train)
# now w should be close to 5 (which is the optimum)
print(session.run(w))
```

Introduction to programming frameworks

Deep learning frameworks Criteria for choosing a framework:

- ease of programming (development and deployment)
- running speed on large NN
- Truly open (open source with good governance), some companies govern an open source framework then gradually close the source in the future

TensorFlow

QUIZ - Hyperparameter tuning, Batch Normalization, Programming Frameworks

1. If searching among a large number of hyperparameters, you should try values in a grid rather than random values, so that you can carry out the search more systematically and not rely on chance. True or False?

- True
- False (X)

2. Every hyperparameter, if set poorly, can have a huge negative impact on training, and so all hyperparameters are about equally important to tune well. True or False?

- True
- False (X)

3. During hyperparameter search, whether you try to babysit one model ("Panda" strategy) or train a lot of models in parallel ("Caviar") is largely determined by:

- Whether you use batch or mini-batch optimization
- The presence of local minima (and saddle points) in your neural network
- The amount of computational power you can access (X)
- The number of hyperparameters you have to tune

4. If you think β (hyperparameter for momentum) is between on 0.9 and 0.99, which of the following is the recommended way to sample a value for beta?

- $r = np.random.rand(), beta = r * 0.09 + 0.9$
- $r = np.random.rand(), beta = 1 - 10 * *(-r - 1)$ (X)
- $r = np.random.rand(), beta = 1 - 10 * *(-r + 1)$
- $r = np.random.rand(), beta = r * 0.9 + 0.09$

5. Finding good hyperparameter values is very time-consuming. So typically you should do it once at the start of the project, and try to find very good hyperparameters so that you don't ever have to revisit tuning them again. True or false?

- True (X)
- False

6. In batch normalization as presented in the videos, if you apply it on the llth layer of your neural network, what are you normalizing?

- $a^{[l]}$
- $W^{[l]}$
- $b^{[l]}$
- $z^{[l]}$ (X)

7. In the normalization formula $z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$, why do we use epsilon?

- To avoid division by zero (X)
- In case μ is too small
- To speed up convergence
- To have a more accurate normalization

8. Which of the following statements about γ and β in Batch Norm are true?

- They can be learned using Adam, Gradient descent with momentum, or RMSprop, not just with gradient descent. (X)
- They set the mean and variance of the linear variable $z^{[l]}$ of a given layer. (X)
- There is one global value of $\gamma \in \Re$ and one global value of $\beta \in \Re$ for each layer, and applies to all the hidden units in that layer.
- The optimal values are $\gamma = \sqrt{\sigma^2 + \varepsilon}$, and $\beta = \mu$.
- β and γ are hyperparameters of the algorithm, which we tune via random sampling.

9. After training a neural network with Batch Norm, at test time, to evaluate the neural network on a new example you should:

- Perform the needed normalizations, use μ and σ^2 estimated using an exponentially weighted average across mini-batches seen during training.
- Skip the step where you normalize using μ and σ^2 since a single test example cannot be normalized.
- Use the most recent mini-batch's value of μ and σ^2 to perform the needed normalizations. (X)
- If you implemented Batch Norm on mini-batches of (say) 256 examples, then to evaluate on one test example, duplicate that example 256 times so that you're working with a mini-batch the same size as during training.

10. Which of these statements about deep learning programming frameworks are true? (Check all that apply)

- Deep learning programming frameworks require cloud-based machines to run.
- Even if a project is currently open source, good governance of the project helps ensure that the it remains open even in the long term, rather than become closed or modified to benefit only one company. (X)
- A programming framework allows you to code up deep learning algorithms with typically fewer lines of code than a lower-level language such as Python. (X)