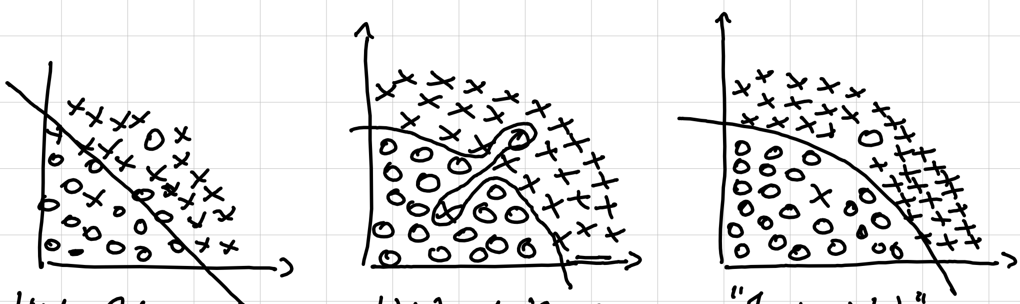
DeepLearning.AI Course on Coursera

- Bias / Variance / Overfit / Underfit



High Bias
= Underfit
- High error* on training set
- similar error* on dev set

High variance
= Overfit
- Low error* on training set
- High error* on dev set

"Just right"
- Lowish error* on training set
- Low-ish error* on dev set

*: Relative error to optimal error called "Bayes error"

What to do?

High bias :
- More powerful / bigger network / model
- train longer
- bigger network should be able to fit trainingset well (as long as somebody can do this well.)

High variance? (good train set → bad dev set perf.)

- Get more data?
- Regularization
- Better NN architecture

Knowing if we have high bias or high variance problem tells you what to do!

- E.g. high bias: more training data will not help (as much)
  → more powerful network or train longer / better.

Regularization

→ Helps fit overfitting / high variance

Ex: L_2 regularization for logistic regression
(add $\frac{\lambda}{2m} \|w\|_2^2$ to cost function)

For neural net: Frobenious norm of matrix

"weight decay" → makes weight update smoother in backprop.

Dropout

- Scale layer weights by inverse of dropout probability → <u>inverse dropout.</u>

Other techniques

- **Augment dataset** (e.g. by mirroring & distorting images, cropping)
- **Early stopping**

- Mixes hyperparam search and preventing overfitting
- Andrew Ng likes to separate the two.

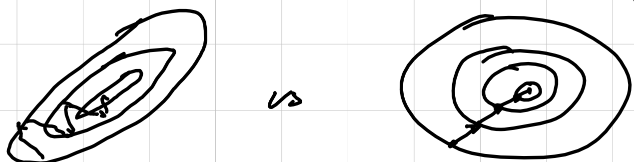
Why regularization works:

Moves weights so that network is closer to a linear function (ex. tanh activation, which is linear around 0)

(I understand

Normalizing Inputs

Inputs on different scales hurt learning speed



Need lower learning rate here

$\Rightarrow$ normalize all inputs e.g. $\mu = 0$, $\sigma = 1$.

Vanishing & Exploding Gradients

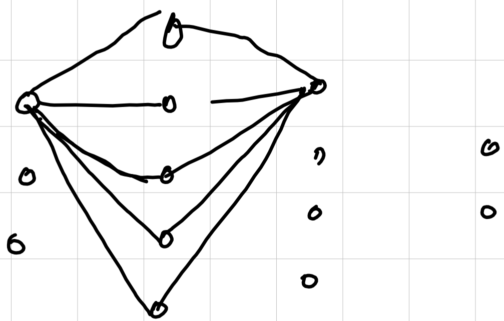
Deep networks: gradients a product of many weights
if everything at steps are > 1 $\rightarrow$ exploding
< 1 $\rightarrow$ vanishing

How to "solve" it: initialize weights "correctly":
$Var(w^i) = \frac{2}{n}$ $\rightarrow$ randomly initialize weights with this variance.

?

- Get the idea but don't really understand the details.

Network architecture recap



$x = z^0$ $\qquad$ z^1 $\qquad$ z^2 $\qquad$ $z^3 = y$

$$z^1 = g^1(z^0) = \phi\big(\underbrace{\overset{5\times3}{W^1}\,\overset{3\times1}{z^0} + \overset{5\times1}{b^1}}\big)$$

Weight dimensions: rows: # outputs

cols: # inputs

Numerical Approximation & Gradient Checking

Linear approx $\dfrac{f(\theta+\varepsilon) - f(\theta)}{\varepsilon}$

or $\dfrac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon}$

-> Check derivative computation against linear approx as debugging check.

$\theta = [W^1, b^1, W^2, b^2, \dots]$, compute $J(\theta)$

$d\theta = [dW^1, db^1, \dots]$

is $d\theta$ gradient of $J(\theta)$?

$$\forall i: \quad d\theta_{approx}^{(i)} \approx \frac{J(\theta_1, \dots, \theta_i+\varepsilon, \dots) - J(\dots, \theta_i-\varepsilon, \dots)}{2\varepsilon}$$

use $\varepsilon \approx 10^{-7}$

$$\frac{\|d\theta_{approx} - d\theta\|_2}{\|d\theta_{approx}\|_2 + \|d\theta\|_2} \approx 10^{-7} \rightarrow \text{great}$$

10^{-5} ? ok?

10^{-3} -> worry!

Notes

- Doesn't work with dropout
- Only use for debugging, not training
- To debug, try to identify faulty component (dW^i or db^i)
- Remember regularization
- Run at start & maybe after some training

① $20.000\cancel{.000} \cdot \frac{5}{\cancel{1000}}$ 100k obs to train ✓

② no test → no unbiased estimated value of variance.

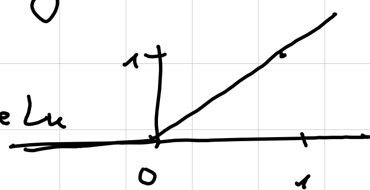
—

2023-08-30

First DeepLearning AI prog assignment.

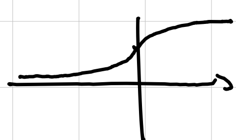
3 layer neural net: Lin → Relu → Lin → Relu → Lin - Sigmoid

ReLu



$\text{Relu}(x) = \max(x, 0).$

Sigmoid



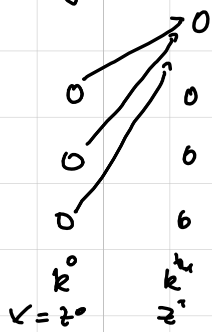
$\text{Sigmoid}(x) = \frac{1}{1+e^{-x}}$

$$\text{sigmoid}(-x) = 1 - \sigma(x) = 1 - \frac{1}{1+e^{-x}} \quad \frac{\cancel{1+}e^{-x} + 1}{1+e^{-x}}$$

$$= \frac{e^{-x}}{1+e^{-x}} = \frac{1}{e^{x} + 1} = \frac{1}{1+e^{x}} = \sigma(-x)$$

$\frac{1}{1+e^{-x}}$ is actually the logistic function.
Sigmoid is any function with this name.

Layer dims



$$z^1 = g(W^1 z^0 + b^1)$$

$$\dim(W^1) = \underbrace{k^1}_{\text{rows}} \times \underbrace{k^0}_{\text{cols}} \qquad \cancel{k^1 \times k} \quad \dim(b^1) = k^1 \times 1$$

This exercise: 2 for 2 dimensions

—

Maybe I should do the initial course again? Step through the different layers. Seems like that would not hurt...

And read my own post again....

—

Wow, the initialization really works wonders. σ^2

Matrix weights initialized to $\mathcal{N}\left(0, \frac{2}{\underbrace{k^{l-1}}_{\text{size of previous layer}}}\right)$

DeepLearning.AI Special - Week 1 Recap

What I learned

- Weight initialization: zero, large random, He ($Var(\frac{2}{n_l})$)
  → can have significant impact on training speed.

- Bias / Variance tradeoff

  High Bias: underfit

  · High error on training set

  → more powerful model
  → better training ← larger / opt-algo / initialization

  High Variance: overfit

  · High error on dev level set

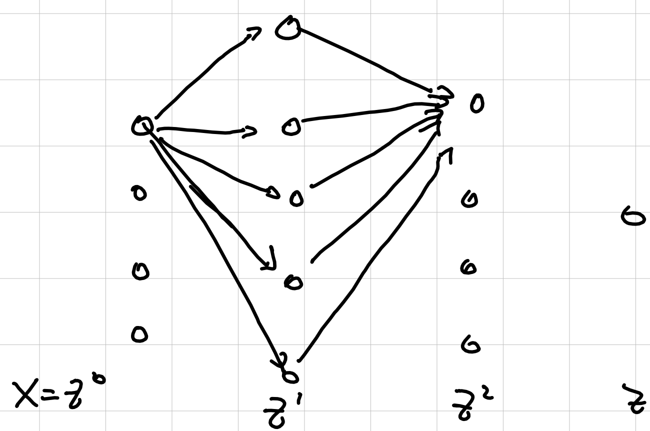
  → regularization ↔ L2 / Dropout / early stopping
  → more training data

- Train / dev / test sets
  - test: only once, unbiased estimate. can maybe be skipped
  - dev: evaluate variance / overfit

  Split doesn't have to be 70/30 or 60/20/20
  dev and test set just need to be large enough.

- gradient checking: Compare $\frac{J(\theta+\varepsilon_i) - J(\theta-\varepsilon_i)}{2\varepsilon}$ to $\frac{\partial J}{\partial \theta_i}$ to ensure gradient computation is correct.

Neural-net arch:



$$z^i = g(A^i) = g(\underbrace{W^i}_{\text{weight matrix } k^i \times k^{i-1}} \cdot z^{i-1} + \underbrace{b^i}_{\text{biases } k^i \times 1})$$

g: activation function, e.g. ReLu or logistic

Backprop:

Just re-read my post on backprop.
It is really good! I think I could produce
quality content on this stuff. Just write
up my journey, it will show some
good stuff and deepen my learnings.
How encouraging.
The details of backprop are a bit
involved, but understanding it once makes
me confident that I could again.

- Regularization: L_2 and Dropout
  - Adding L_2 and Dropout to simple neural nets.
  - Dropout: Inverted dropout: scale activations by $\frac{1}{\text{keep\_prob}}$ to have the same overall size.
    - Backprop: drop the same neurons on dA_i as on A_i

2023-09-01

Last Video: Exponentially weighted averages & bias correction

$V_t = (1-\beta) \cdot \theta_t + \beta V_{t-1}$ -> exp weighted avg

~ averages $\frac{1}{1-\beta}$ terms (e.g. $\beta = 0.02$ -> 50 terms)

Bias correction $V_t = \frac{1}{1-\beta^t} (\dots)$

corrects for initial v_0 being 0, goes to 1 for $t \to \infty$.

Gradient descent with momentum.

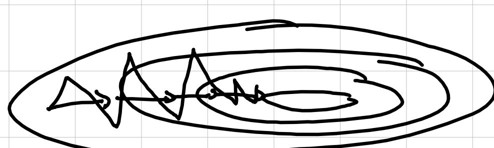
→ Apply exp. avg. to gradients:

$W^\ell := W^\ell + \alpha \cdot dW^\ell$ → $V_{dW^\ell} = (1-\beta) V_{dW^\ell} + \beta \, dW$

$W^\ell = W^\ell + \alpha \cdot V_{dW^\ell}$

and similar for b^ℓ

Intuition:



Smooth out unwanted oscillations in gradient descent.

RMSProp

- Another way of having a type of momentum

- Formula for updates:

$$S_{dW} = \beta_2 S_{dW} + (1-\beta_2) dW^2$$ ← squared

$$W = W - \alpha \frac{dW}{\sqrt{S_{dW} + \varepsilon}}$$

ε: small smoothing, e.g. 10^{-8}

Similar for b.

Why this formula for RMSprop?

Adam optimization

→ Combination of momentum and RMSprop

$$V_{dW} = \beta_1 V_{dW} + (1-\beta_1) dW$$

$$S_{dW} = \beta_2 S_{dW} + (1-\beta_2) dW^2$$

$$W = W - \alpha \frac{1}{1-\beta^t} \frac{V_{dW}}{\sqrt{S_{dW} - \varepsilon}}$$

Learning rate decay

→ different schemes for decreasing the learning rate over time. e.g.

$$\alpha = \frac{1}{1 + decay\_rate \times epoch\_num}$$

Local optima & saddle points

NN optimization unlikely to get stuck in a local minimum - with huge parameter spaces it is very unlikely that all parameters are sloped upwards
However much more likely to hit a saddle point - this can also slow down learning b/c gradients are small
Momentum / Adam / RMSprop helps with this.

Hyperparameter Search

Typical Hyperparameters

$\boxed{P0}$ α : learning rate

P2 β / $\beta_1, \beta_2, \varepsilon$: Parameters for momentum / Adam optimizer

P1 # Layers

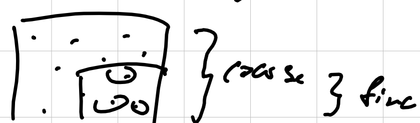
P1 # hidden units per Layer

P1 m: Minibatch-size

How to search:

1. Use random values, don't use a grid
   -> random values give many more values per HP.

2. Consider using a coarse-to-fine approach



Choose linear or logarithmic samples correctly

e.g. learning rate 0.0001 – 1 -> logarithmic, we want ~ same # points between $[10^{-4}, 10^{-3}]$ as between $[10^{-1}, 10^{0}]$.

=> Sample uniformly from the log space.

e.g. $x \in [-4, 0] \to 10^{x}$

For parameters close to 1, e.g. $\beta \in (0.9, 0.999]$ sample $1-\beta$ from logspace.

For some params, e.g. # layers and # neurons sampling from linear space should work well.

<u>Pandas vs Caviar strategy:</u>

Panda: Babysit a single network

Caviar: Many different networks in parallel.

=> Pandas if you must, caviar if you can.

Normalized Batch

→ Normalize linear units z^l in the hidden layers and add parameters β^l and γ^l as explicit scale & bias, which are also learned.

$$\tilde{z}^l = \frac{z^l - \mu^l}{\sqrt{\sigma^{l2} + \varepsilon}}$$ μ^l, σ^l computed per sample / minibatch

$$a^l = \underset{k\times 1}{\beta^l} \cdot \underset{k\times 1}{\tilde{z}^l} + \underset{k\times 1}{\gamma^l}$$ (elementwise product) replicate for minibatches.

→ eliminate b^l as we now have two bias terms

→ μ^l and σ^l normally computed for mini-batches

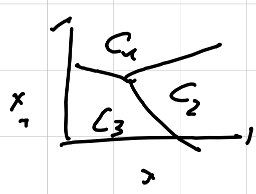
→ for test, usually a exp-weighted average of μ^l and σ^l is used that is computed across mini-batches.

→ Why it works? Explicitly controls ~~the~~ bias and scale of hidden layers. Makes their change more slowly which benefits later layers which have to adapt to these changes.

Softmax Regression

For multiclass classification
Last-layer activation function $g(z) = \frac{e^z}{\sum_{i=1}^{n} e^{z_i}}$

-> normalized e^z.
-> Bit unusual as it is across the full layer, not element-wise
-> Generalization of logistic regression across more classes.
-> Cost function $C(y, \hat{y}) = -\sum_i y_i \log \hat{y}_i$
-> Gives linear decision boundaries for single-layer networks, e.g.



$$\frac{\partial C}{\partial z^l} = \hat{y} - y = \frac{\partial C}{\partial a^l} \cdot \frac{\partial a^l}{\partial z^l} \;\ldots$$

ML Frameworks

- Many options to pick from
- Criteria:
  1) Code is easy to write + read
  2) Performance
  3) Truly open: OS + Good Governance

This course: TensorFlow.

Practical Aspects of ML Blog post

- Basically a recap of the different topics
- Links to copied lecture notes and copied notebooks + my handwritten notes
- Some thoughts on programming assignment and how these could be harder, yet doable
  - Provide step-by-step instructions, but no code
  - Provide successive hints, giving a very small amount of points for accessing them.
  - Remove the training wheels over time.
- Questions on the material that go deeper
  -> but don't overdo it.