

lec6

Creation date: 09/01/2020 13:31

Last Modified: 10/01/2020 13:40

Lec 6: Training Neural Networks

Where we are now...

Mini-batch SGD

Loop:

1. **Sample** a batch of data
2. **Forward** prop it through the graph (network), get loss
3. **Backprop** to calculate the gradients
4. **Update** the parameters using the gradient

Stanford University April 20, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 9

Overview

1. **One time setup**
activation functions, preprocessing, weight initialization, regularization, gradient checking
2. **Training dynamics**
babysitting the learning process, parameter updates, hyperparameter optimization
3. **Evaluation**
model ensembles

Stanford University April 20, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 11

Part 1

- Activation Functions
- Data Preprocessing
- Weight Initialization
- Batch Normalization
- Babysitting the Learning Process
- Hyperparameter Optimization

Stanford
University
April 20, 2017

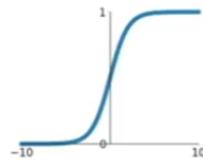
Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 12

Activation Functions

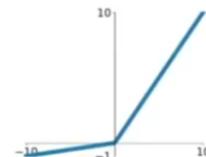
Sigmoid

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



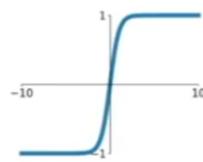
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

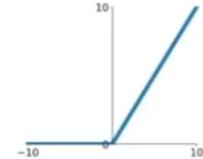


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

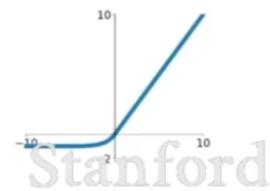
ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Fei-Fei Li & Justin Johnson & Serena Yeung

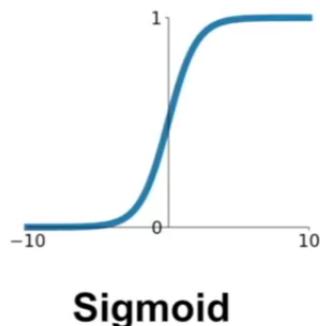
Lecture 6 - 15

Stanford
University
April 20, 2017

Activation Functions:

- Sigmoid

Activation Functions



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

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

3 problems:

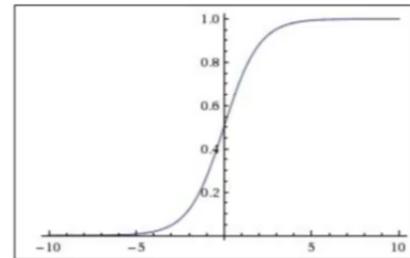
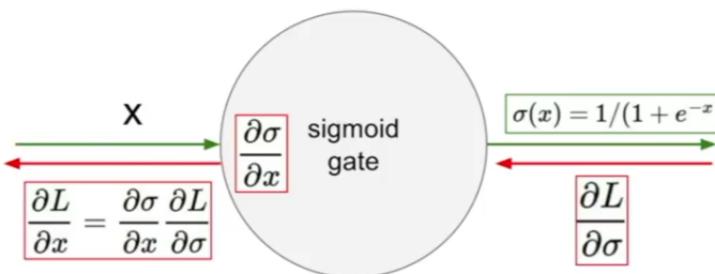
1. Saturated neurons “kill” the gradients

Stanford

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 17

University
April 20, 2017



What happens when $x = -10$?

What happens when $x = 0$?

What happens when $x = 10$?

Stanford

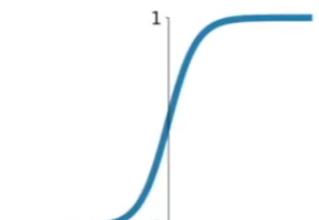
Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 18

University
April 20, 2017

- At 10 or -10 the gradient of the sigmoid layer will be 0 causing no gradient to be learnt.
- Only at the values near to $x=0$ we will have some gradients.

Activation Functions



Sigmoid

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

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

3 problems:

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered

Stanford

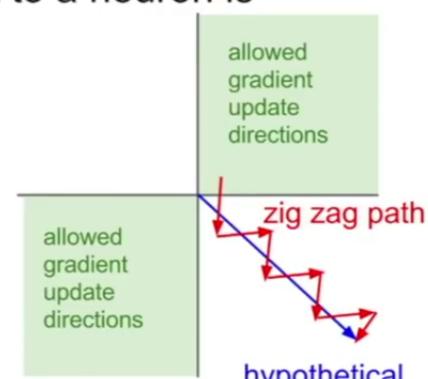
University
April 20, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 19

Consider what happens when the input to a neuron is always positive...

$$f \left(\sum_i w_i x_i + b \right)$$



What can we say about the gradients on w?

Always all positive or all negative :(

(this is also why you want zero-mean data!)

Stanford

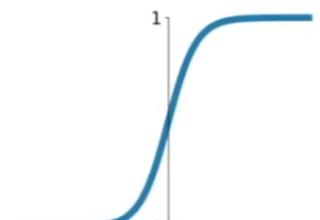
University
April 20, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 21

- Because the gradient for W will depend on x and if x is positive we will have only +ve gradients or when x is negative we'll have only -ve gradients.

Activation Functions



Sigmoid

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

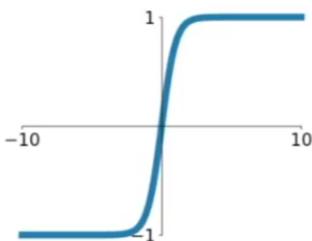
- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

3 problems:

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. $\exp()$ is a bit compute expensive

- TanH

Activation Functions

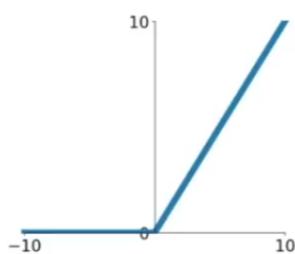


tanh(x)

- Squashes numbers to range [-1,1]
- zero centered (nice)
- still kills gradients when saturated :(

- ReLU

Activation Functions



ReLU
(Rectified Linear Unit)

- Computes $f(x) = \max(0, x)$
- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Actually more biologically plausible than sigmoid

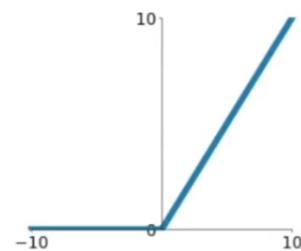
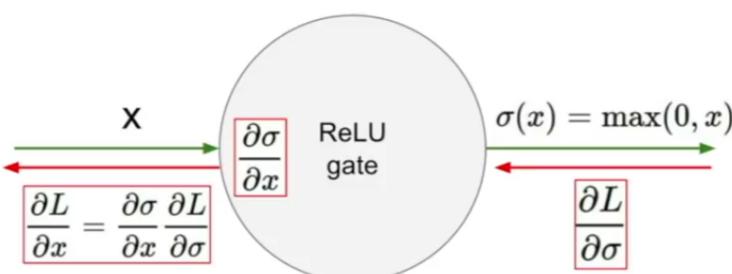
- Not zero-centered output
- An annoyance:

hint: what is the gradient when $x < 0$? 

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 25

University
April 20, 2017



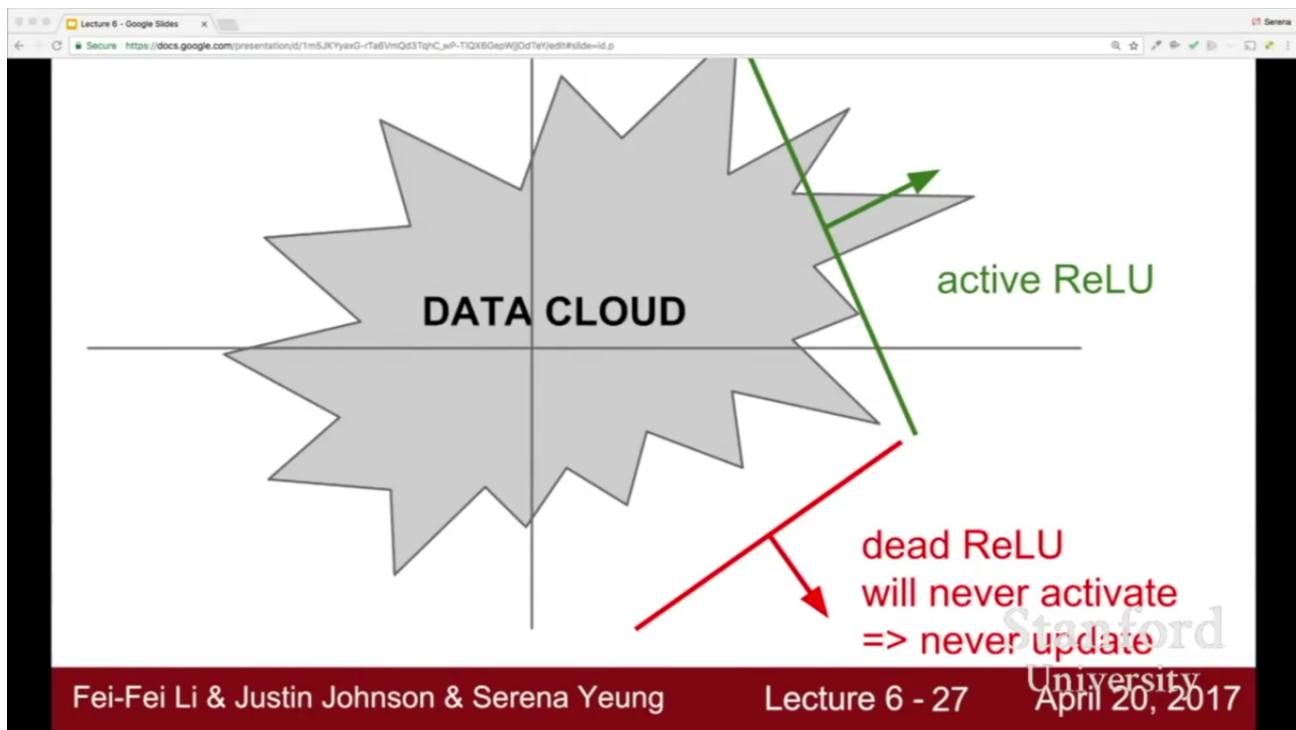
What happens when $x = -10$?
What happens when $x = 0$?
What happens when $x = 10$?

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 26

Stanford
University
April 20, 2017

- For $x = -10$, the gradient is 0.
- For $x = +10$, we have a linear function so we have a constant gradient value.
- For $x = 0$, it is undefined but for practical purposes we consider the gradient to be 0 at $x=0$.
- So in reality we have only half of the space available to find the gradients.

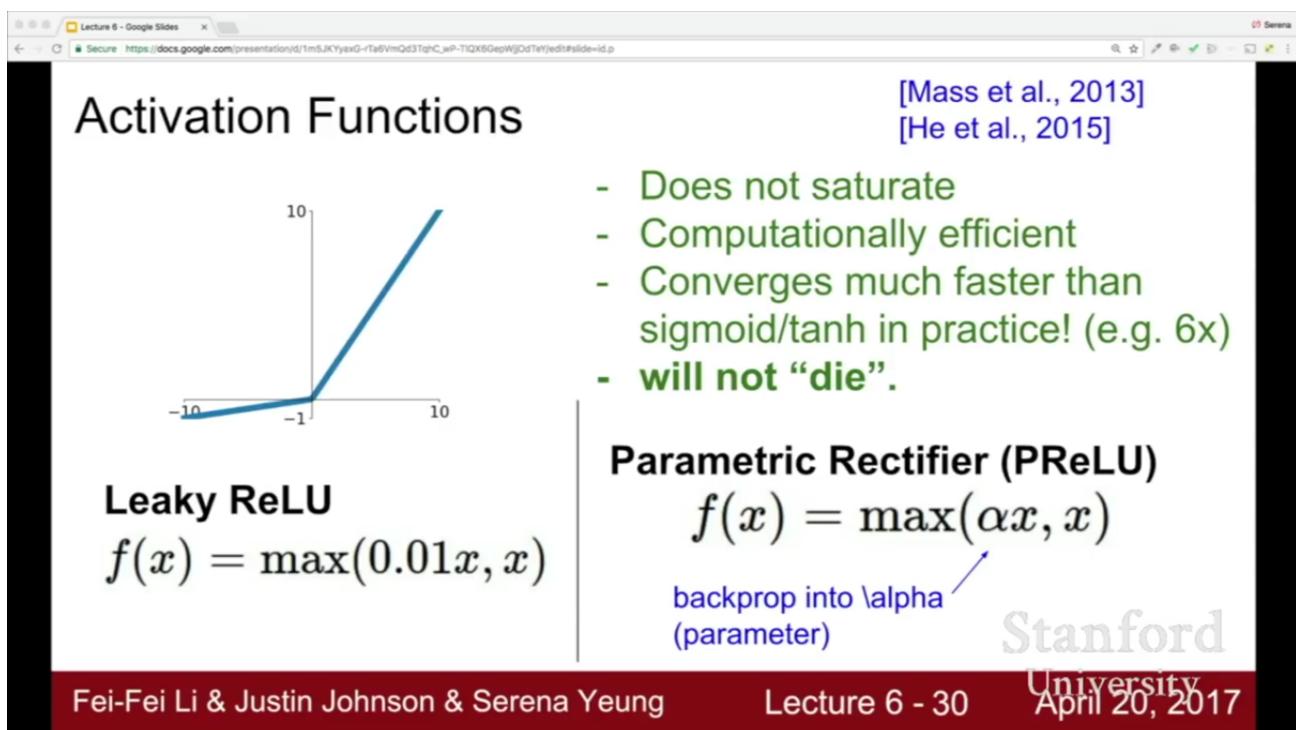


Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 27

Stanford University April 20, 2017

- So some region in the possible gradient region will never contribute to any gradients.
- **So it depends on the value of initialization also.**
- **Even if we have a very high learning rate, the update will oscillate and it can knock off the values of the gradient to the dead ReLU region and never update henceforth.**
- ***This problem is still existing and it is still an open research problem but never the less the training happens and it works but around 10% of the neurons will be in the dead ReLU region.***
- **People also initialize the ReLU with slightly positive biases (like 0.01).** So we have activate values. But initializing with 0 bias is also common.



Fei-Fei Li & Justin Johnson & Serena Yeung

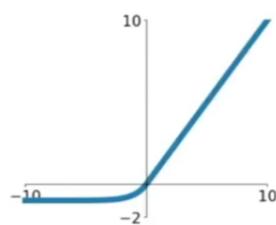
Lecture 6 - 30

Stanford University April 20, 2017

Activation Functions

[Clevert et al., 2015]

Exponential Linear Units (ELU)



- All benefits of ReLU
- Closer to zero mean outputs
- Negative saturation regime compared with Leaky ReLU adds some robustness to noise

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

- Computation requires $\exp()$

Stanford

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 31

University
April 20, 2017

- MaxOut "Neuron":

Maxout “Neuron”

[Goodfellow et al., 2013]

- Does not have the basic form of dot product -> nonlinearity
- Generalizes ReLU and Leaky ReLU
- Linear Regime! Does not saturate! Does not die!

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

Problem: doubles the number of parameters/neuron :(

Stanford

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 32

University
April 20, 2017

TLDR: In practice:

- Use **ReLU**. Be careful with your learning rates
- Try out **Leaky ReLU / Maxout / ELU**
- Try out **tanh** but don't expect much
- **Don't use sigmoid**

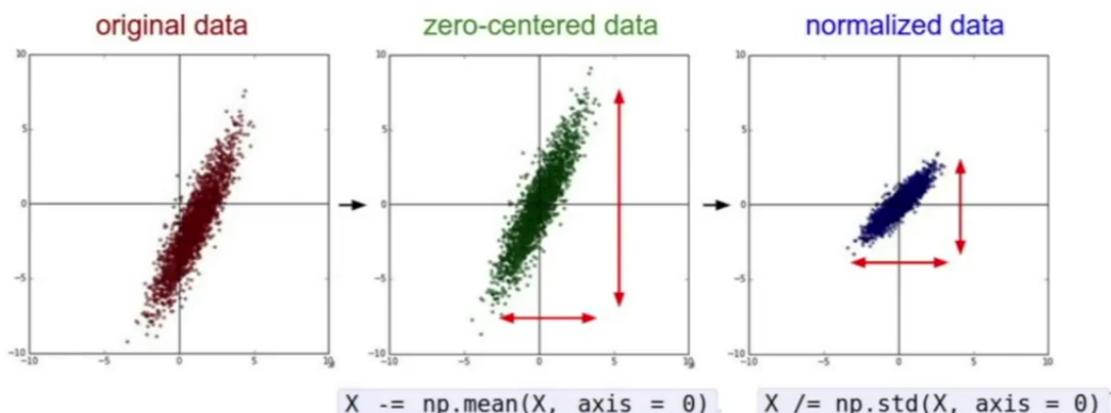
Stanford
University
April 20, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 33

Data Preprocessing:

Step 1: Preprocess the data



(Assume X [NxD] is data matrix,
each example in a row)

Stanford
University
April 20, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 35

- we normalize it by standard deviation for most of the cases.
- We do preprocessing to bring the values in the same range.
- For images we don't usually normalize the data, we just do zero-centering assuming that the values are normally distributed.

- But in other machine learning problems we do zero-centering and also perform normalization.

Step 1: Preprocess the data

In practice, you may also see **PCA** and **Whitening** of the data

Stanford University April 20, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 38

- We also perform **PCA** to **decorrelate** the data and **whitening** to **normalize** the zero-centered data.

TLDR: In practice for Images: center only

e.g. consider CIFAR-10 example with [32,32,3] images

- Subtract the mean image (e.g. AlexNet)
(mean image = [32,32,3] array)
- Subtract per-channel mean (e.g. VGGNet)
(mean along each channel = 3 numbers)

Not common to normalize variance, to do PCA or whitening

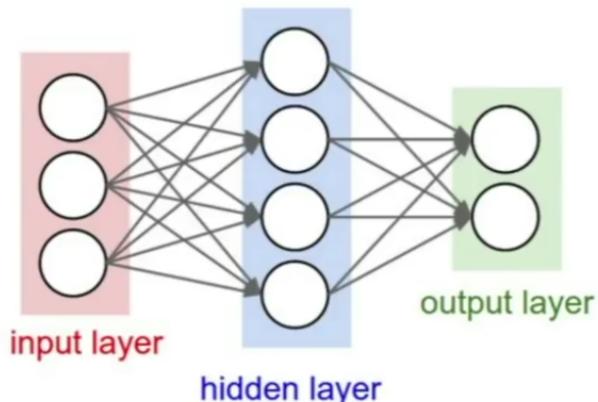
Stanford University April 20, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 39

- FOR IMAGE DATA WE DONT HAVE TO NORMALIZE THE VARIANCE, OR TO PCA OR WHITENING.
- IF TRAINING HAS CERTAIN PREPROCESSING DONE, WE HAVE TO PERFORM THE SAME PREPROCESSING WHILE TESTING.
- THE MEAN IS FOUND OVER THE ENTIRE TRAINING DATA.
- We can estimate the MEAN for a batch too if the batch is of considerable size.

Weight Initialization:

- Q: what happens when $W=0$ init is used?



Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 41

Stanford
University
April 20, 2017

- If $W=0$, all the neurons will have the same value since $\text{activation}(0 \cdot x + b) \Rightarrow \text{activation}(b)$
- So since all the neurons have the same value, the gradients will also be the same for each neuron. So they will update in the same way. Hence there will be no considerable learning because we want each neuron to learn different things.
- There will be **NO SYMMETRY** if we initialize with $W=0$
- Solutions:

- First idea: **Small random numbers**
(gaussian with zero mean and $1e-2$ standard deviation)

```
W = 0.01 * np.random.randn(D, H)
```

Works ~okay for small networks, but problems with deeper networks.

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 43

Stanford
University
April 20, 2017

input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532
hidden layer 6 had mean -0.000000 and std 0.000119
hidden layer 7 had mean 0.000000 and std 0.000026
hidden layer 8 had mean -0.000000 and std 0.000006
hidden layer 9 had mean 0.000000 and std 0.000001
hidden layer 10 had mean -0.000000 and std 0.000000

All activations become zero!

Q: think about the backward pass.
What do the gradients look like?

Hint: think about backward pass for a W^*X gate.

Stanford University April 20, 2017

- We can notice in the line graph that all the layers have 0 mean value and the std is reaching 0. Hence all the values of Wx will result in 0. The local gradient will be multiplied with the upstream gradient multiplied with the local gradient value. For W , it depends on X , so eventually all the gradients will collapse to 0.

`W = np.random.randn(fan_in, fan_out) * 1.0 # layer initialization`

*1.0 instead of *0.01

Almost all neurons completely saturated, either -1 and 1. Gradients will be all zero.

Stanford University April 20, 2017

- If we use a std of 1.0 instead of 0.01 and tanH the values will get saturated between -1 and +1.

input layer had mean 0.001800 and std 1.00131
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
hidden layer 6 had mean -0.000389 and std 0.292116
hidden layer 7 had mean -0.000228 and std 0.273387
hidden layer 8 had mean -0.000291 and std 0.254935
hidden layer 9 had mean 0.000361 and std 0.239266
hidden layer 10 had mean 0.000139 and std 0.228008

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

Xavier initialization
[Glorot et al., 2010]

Stanford University April 20, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 48

- For smaller networks Xavier initialization is acceptable but not for deeper networks since eventually the std deviation is collapsing to 0. And Xavier initialization assumes the activation to be linear.

input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.398623 and std 0.582273
hidden layer 2 had mean 0.272352 and std 0.403795
hidden layer 3 had mean 0.186976 and std 0.276912
hidden layer 4 had mean 0.136442 and std 0.198685
hidden layer 5 had mean 0.099568 and std 0.148299
hidden layer 6 had mean 0.072234 and std 0.103288
hidden layer 7 had mean 0.049775 and std 0.072748
hidden layer 8 had mean 0.035138 and std 0.051572
hidden layer 9 had mean 0.025404 and std 0.038583
hidden layer 10 had mean 0.018408 and std 0.026076

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

but when using the ReLU nonlinearity it breaks.

Stanford University April 20, 2017

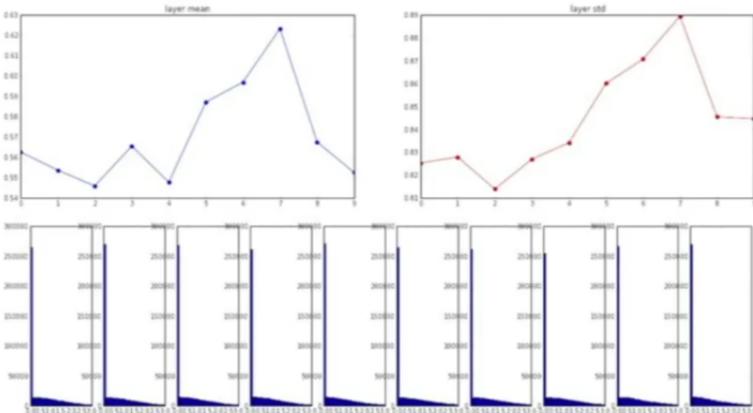
Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 49

- When we use ReLU activations, w.k.t half of the activation are 0 (for negative input values.).

```

input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.562488 and std 0.025232
hidden layer 2 had mean 0.553614 and std 0.827835
hidden layer 3 had mean 0.545867 and std 0.813855
hidden layer 4 had mean 0.565196 and std 0.826902
hidden layer 5 had mean 0.547678 and std 0.834692
hidden layer 6 had mean 0.587183 and std 0.860635
hidden layer 7 had mean 0.596867 and std 0.870618
hidden layer 8 had mean 0.623214 and std 0.889348
hidden layer 9 had mean 0.567498 and std 0.845357
hidden layer 10 had mean 0.552531 and std 0.844523

```



He et al., 2015
(note additional /2)

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 50

Stanford
University
April 20, 2017

- He initialization has the additional term of dividing by 2 for compensating for half of the activation being 0 when using ReLU.

Proper initialization is an active area of research...

Understanding the difficulty of training deep feedforward neural networks
by Glorot and Bengio, 2010

Exact solutions to the nonlinear dynamics of learning in deep linear neural networks by Saxe et al, 2013

Random walk initialization for training very deep feedforward networks by Sussillo and Abbott, 2014

Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification by He et al., 2015

Data-dependent Initializations of Convolutional Neural Networks by Krähenbühl et al., 2015

All you need is a good init, Mishkin and Matas, 2015

...

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 52

Stanford
University
April 20, 2017

Batch Normalization

Batch Normalization

[Ioffe and Szegedy, 2015]

"you want unit gaussian activations? just make them so."

consider a batch of activations at some layer.

To make each dimension unit gaussian, apply:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

this is a vanilla

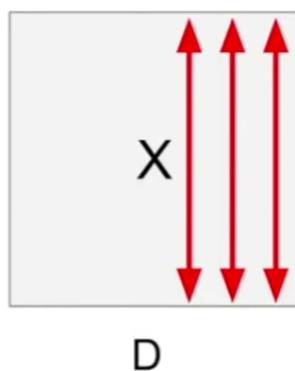
differentiable function...

Batch Normalization

[Ioffe and Szegedy, 2015]

"you want unit gaussian activations?
just make them so."

N



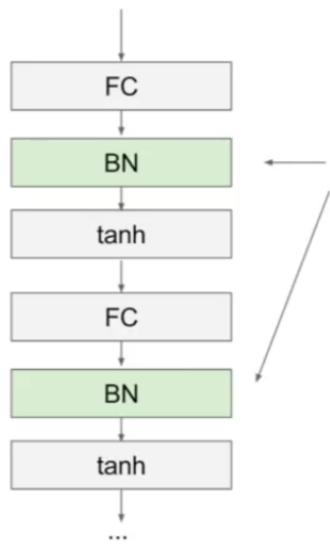
1. compute the empirical mean and variance independently for each dimension.

2. Normalize

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

Batch Normalization

[Ioffe and Szegedy, 2015]



Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

Stanford University
April 20, 2017

- We perform the Batch Normalization before applying any non-linearity and after Convolutional/FC layer.

Lecture 6 - Google Slides [Secure https://docs.google.com/presentation/d/1m5JKYyexQ-rtafVmQd3TqHC_wP-TIQX6DepWjOdTeY/edit#slide=id.p] [Serena]

Batch Normalization

[Ioffe and Szegedy, 2015]

Normalize:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\text{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \text{E}[x^{(k)}]$$

to recover the identity mapping.

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 58

Stanford University April 20, 2017

Lecture 6 - Google Slides [Secure https://docs.google.com/presentation/d/1m5JKYyexQ-rtafVmQd3TqHC_wP-TIQX6DepWjOdTeY/edit#slide=id.p] [Serena]

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$; Parameters to be learned: γ, β

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

```


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


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


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


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


```

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 59

Stanford University April 20, 2017

- Here γ and β are learned parameters.
- Advantages of BatchNorm:
 - Improves gradient flow through the network.
 - Allows higher learning rates.
 - Reduces the strong dependence on initialization.
 - Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe.
 - It acts like a regularizer because, each input now depends on itself and also on other samples (mean and variance calculation). So now it is not producing deterministic value for a

given training example and it is tying all of these input in a batch together.

- Since it is no longer deterministic, it kind of jitters your representation of X a little bit and in some sense gives a regularization effect.

Lecture 6 - Google Slides Serena

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;
 Parameters to be learned: γ, β

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

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

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

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

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

Note: at test time BatchNorm layer functions differently:

The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.
 (e.g. can be estimated during training with running averages)

Stanford University April 20, 2017

Babysitting the learning process

- Preprocess the data.

Lecture 6 - Google Slides Serena

Step 1: Preprocess the data

The figure consists of three side-by-side scatter plots. The first plot, labeled "original data", shows a cloud of red points centered around (0,0) with axes roughly aligned with the coordinate axes. The second plot, labeled "zero-centered data", shows the same points shifted so their center is at the origin (0,0), with red arrows indicating the shift. The third plot, labeled "normalized data", shows the points scaled so they have unit variance along the horizontal axis, with red arrows indicating the scaling. Below the plots are two lines of code: `X -= np.mean(X, axis = 0)` and `X /= np.std(X, axis = 0)`.

(Assume $X [NxD]$ is data matrix,
 each example in a row)

Stanford University April 20, 2017

- Choose the architecture.

Step 2: Choose the architecture:
say we start with one hidden layer of 50 neurons:

50 hidden neurons
CIFAR-10 images, 3072 numbers
input layer
hidden layer
output layer
10 output neurons, one per class

Stanford University April 20, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 63

- Double check the loss is reasonable.

Double check that the loss is reasonable:

```
def init_two_layer_model(input_size, hidden_size, output_size):
    # initialize a model
    model = {}
    model['W1'] = 0.0001 * np.random.randn(input_size, hidden_size)
    model['b1'] = np.zeros(hidden_size)
    model['W2'] = 0.0001 * np.random.randn(hidden_size, output_size)
    model['b2'] = np.zeros(output_size)
    return model
```

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
loss, grad = two_layer_net(X_train, model, y_train, 0.0) disable regularization
print loss
```

2.30261216167

loss ~2.3.
“correct” for 10 classes

returns the loss and the gradient for all parameters

Stanford University April 20, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 64

- Check the loss after regularization. It should increase

Double check that the loss is reasonable:

```
def init_two_layer_model(input_size, hidden_size, output_size):
    # initialize a model
    model = {}
    model['W1'] = 0.0001 * np.random.randn(input_size, hidden_size)
    model['b1'] = np.zeros(hidden_size)
    model['W2'] = 0.0001 * np.random.randn(hidden_size, output_size)
    model['b2'] = np.zeros(output_size)
    return model
```

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
loss, grad = two_layer_net(X_train, model, y_train, 1e-3) crank up regularization
print loss
```

3.06859716482 ← loss went up, good. (sanity check)

Stanford University April 20, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 65

- Use a small dataset to make it overfit

Lets try to train now...

Tip: Make sure that you can overfit very small portion of the training data

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
X_tiny = X_train[:20] # take 20 examples
y_tiny = y_train[:20]
best_model, stats = trainer.train(X_tiny, y_tiny, X_tiny, y_tiny,
                                  model, two_layer_net,
                                  num_epochs=200, reg=0.0,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = False,
                                  learning_rate=1e-3, verbose=True)
```

The above code:

- take the first 20 examples from CIFAR-10
- turn off regularization (reg = 0.0)
- use simple vanilla 'sgd'

Stanford University April 20, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 66

- If we overfit we should get a very low training loss.
- Make sure the training accuracy is very high. It should be since we need to overfit on the small amount of training data.

Lets try to train now...

Tip: Make sure that you can overfit very small portion of the training data

Very small loss,
train accuracy 1.00,
nice!

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 67

University
April 20, 2017

Lets try to train now...

Start with small regularization and find learning rate that makes the loss go down.

loss not going down:
learning rate too low

```

model = init two layer model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
X_tiny = X_train[:20] # take 20 examples
y_tiny = y_train[:20]
best_model, stats = trainer.train(X_tiny, y_tiny, X_tiny, y_tiny,
                                  model, two_layer.net,
                                  num_epochs=200, reg=0.8,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = False,
                                  learning_rate=1e-3, verbose=True)

Finished epoch 1 / 200: cost 2.302603, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 2 / 200: cost 2.302258, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 3 / 200: cost 2.301849, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 4 / 200: cost 2.301196, train: 0.650000, val 0.650000, lr 1.000000e-03
Finished epoch 5 / 200: cost 2.300044, train: 0.650000, val 0.650000, lr 1.000000e-03
Finished epoch 6 / 200: cost 2.297864, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 7 / 200: cost 2.293595, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 8 / 200: cost 2.285096, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 9 / 200: cost 2.268094, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 10 / 200: cost 2.234787, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 11 / 200: cost 2.173187, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 12 / 200: cost 2.076862, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 13 / 200: cost 1.974090, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 14 / 200: cost 1.895885, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 15 / 200: cost 1.820876, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 16 / 200: cost 1.737430, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 17 / 200: cost 1.642356, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 18 / 200: cost 1.535239, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 19 / 200: cost 1.421527, train: 0.600000, val 0.600000, lr 1.000000e-03

Finished epoch 195 / 200: cost 0.002694, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 196 / 200: cost 0.002674, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 197 / 200: cost 0.002655, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 198 / 200: cost 0.002635, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 199 / 200: cost 0.002617, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 200 / 200: cost 0.002597, train: 1.000000, val 1.000000, lr 1.000000e-03
finished optimization best validation accuracy: 1.000000

```

Loss barely changing: Learning rate is probably too low

Notice train/val accuracy goes to 20% though, what's up with that? (remember this is softmax)

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 71

University
April 20, 2017

Lets try to train now...

Start with small regularization and find learning rate that makes the loss go down.

loss not going down:
learning rate too low
loss exploding:
learning rate too high

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
    model, two_layer_net,
    num_epochs=10, reg=0.000001,
    update='sgd', learning_rate_decay=1,
    sample_batches = True,
    learning_rate=1e-6, verbose=True)

/home/karpathy/cs231n/code/cs231n/classifiers/neural_net.py:50: RuntimeWarning: divide by zero en
countered in log
    data_loss = -np.sum(np.log(probs[range(N), y])) / N
/home/karpathy/cs231n/code/cs231n/classifiers/neural_net.py:48: RuntimeWarning: invalid value enc
ountered in subtract
    probs = np.exp(scores - np.max(scores, axis=1, keepdims=True))
Finished epoch 1 / 10: cost nan, train: 0.091000, val 0.087000, lr 1.000000e+06
Finished epoch 2 / 10: cost nan, train: 0.095000, val 0.087000, lr 1.000000e+06
Finished epoch 3 / 10: cost nan, train: 0.100000, val 0.087000, lr 1.000000e+06
```

cost: NaN almost always means high learning rate...

Stanford University April 20, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 73

Hyperparameter Optimization

Cross-validation strategy

coarse -> fine cross-validation in stages

First stage: only a few epochs to get rough idea of what params work

Second stage: longer running time, finer search

... (repeat as necessary)

Tip for detecting explosions in the solver:

If the cost is ever > 3 * original cost, break out early

Stanford
University
April 20, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 76

For example: run coarse search for 5 epochs

```
max_count = 100
for count in xrange(max_count):
    reg = 10**uniform(-5, 5)
    lr = 10**uniform(-3, -6) ← note it's best to optimize
                                in log space!
    trainer = ClassifierTrainer()
    model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
    trainer = ClassifierTrainer()
    best_model_local, stats = trainer.train(X_train, y_train, X_val, y_val,
                                              model, two_layer_net,
                                              num_epochs=5, reg=reg,
                                              update='momentum', learning_rate_decay=0.9,
                                              sample_batches = True, batch_size = 100,
                                              learning_rate=lr, verbose=False)

    val_acc: 0.412000, lr: 1.405206e-04, reg: 4.793564e-01, (1 / 100)
    val_acc: 0.214000, lr: 7.231888e-06, reg: 2.321281e-04, (2 / 100)
    val_acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
    val_acc: 0.196000, lr: 1.551131e-05, reg: 4.374936e-05, (4 / 100)
    val_acc: 0.079000, lr: 1.753300e-05, reg: 1.200424e+03, (5 / 100)
    val_acc: 0.223000, lr: 4.215128e-05, reg: 4.196174e+01, (6 / 100)
    val_acc: 0.441000, lr: 1.750259e-04, reg: 2.110807e-04, (7 / 100)
    val_acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
    val_acc: 0.482000, lr: 4.296863e-04, reg: 6.642555e-01, (9 / 100)
    val_acc: 0.079000, lr: 5.401602e-06, reg: 1.599828e+04, (10 / 100)
    val_acc: 0.154000, lr: 1.618508e-06, reg: 4.925252e-01, (11 / 100)
```

nice →

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 77

Stanford
University
April 20, 2017

Lecture 6 - Google Slides C1 Serena

[Secure https://docs.google.com/presentation/d/1m5JKYyaxG-rTa6VmQd3TqhC_eP-TIQX6GepWjOdTeY/edit#slide=id.p](https://docs.google.com/presentation/d/1m5JKYyaxG-rTa6VmQd3TqhC_eP-TIQX6GepWjOdTeY/edit#slide=id.p)

Now run finer search...

```

max_count = 100
for count in xrange(max_count):
    reg = 10**uniform(-5, 5)
    lr = 10**uniform(-3, -6)

```

adjust range →

```

max_count = 100
for count in xrange(max_count):
    reg = 10**uniform(-4, 0)
    lr = 10**uniform(-3, -4)

```

val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562880e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)

53% - relatively good for a 2-layer neural net with 50 hidden neurons.

Lecture 6 - Google Slides C1 Serena

[Secure https://docs.google.com/presentation/d/1m5JKYyaxG-rTa6VmQd3TqhC_eP-TIQX6GepWjOdTeY/edit#slide=id.p](https://docs.google.com/presentation/d/1m5JKYyaxG-rTa6VmQd3TqhC_eP-TIQX6GepWjOdTeY/edit#slide=id.p)

Now run finer search...

```

max_count = 100
for count in xrange(max_count):
    reg = 10**uniform(-5, 5)
    lr = 10**uniform(-3, -6)

```

adjust range →

```

max_count = 100
for count in xrange(max_count):
    reg = 10**uniform(-4, 0)
    lr = 10**uniform(-3, -4)

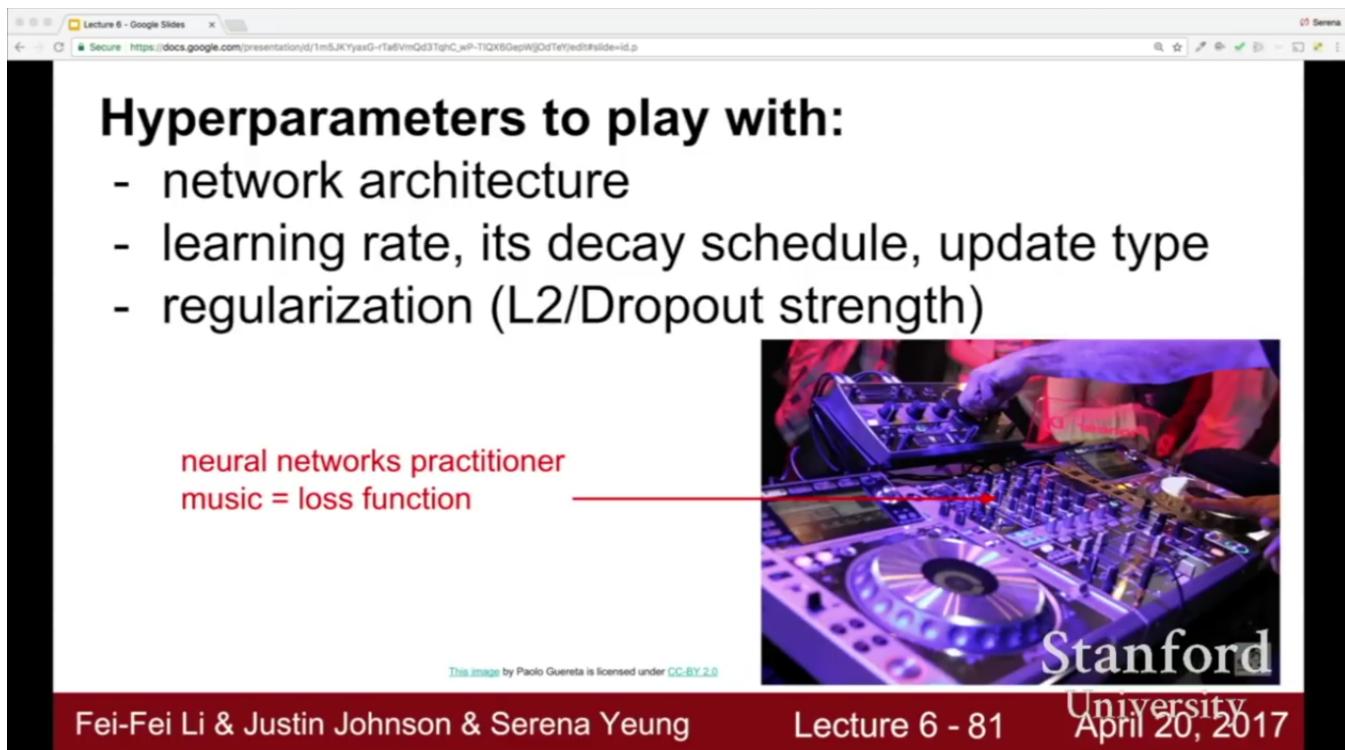
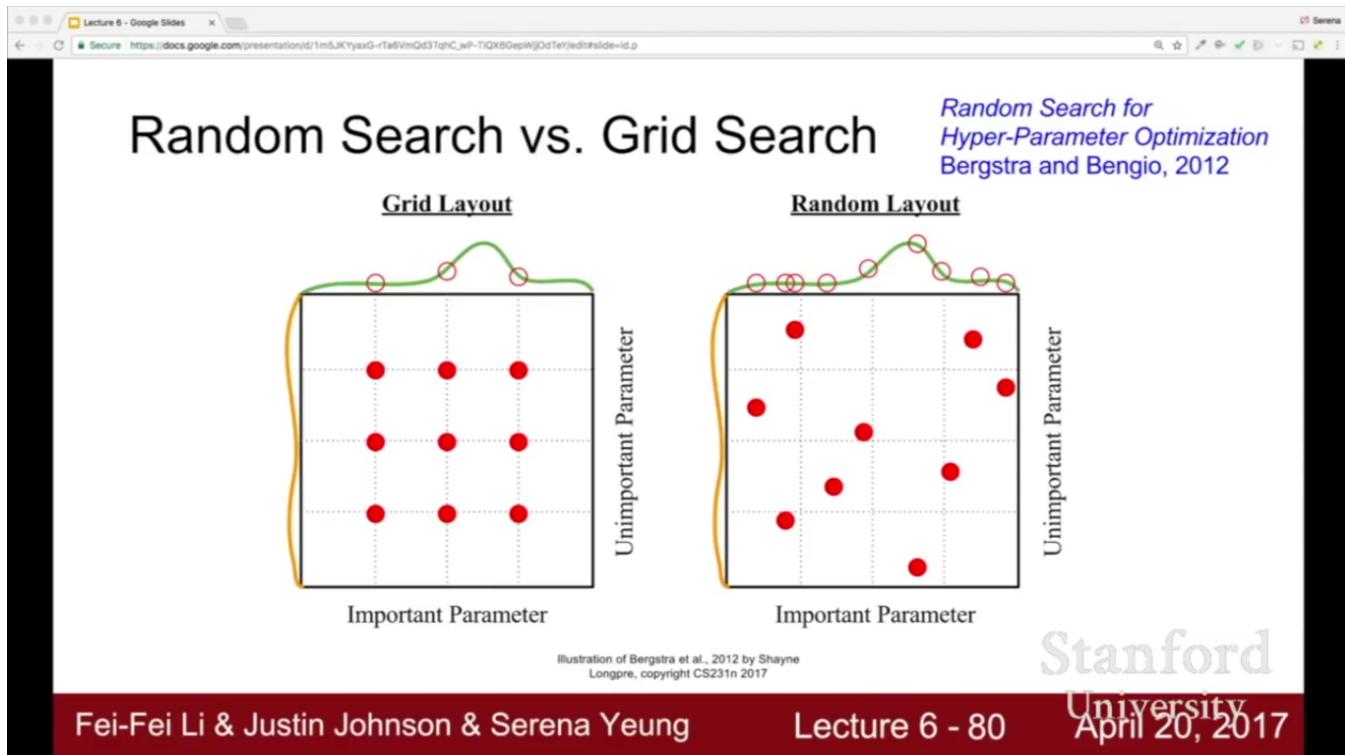
```

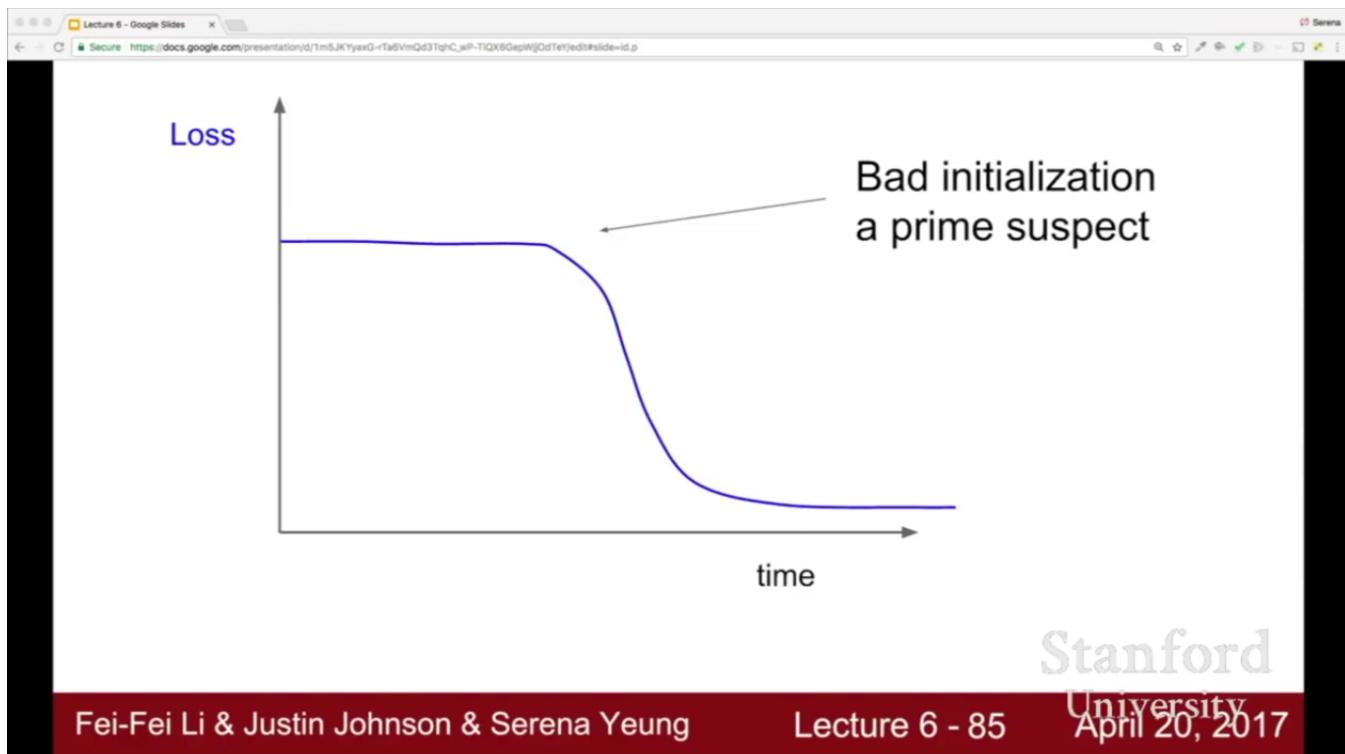
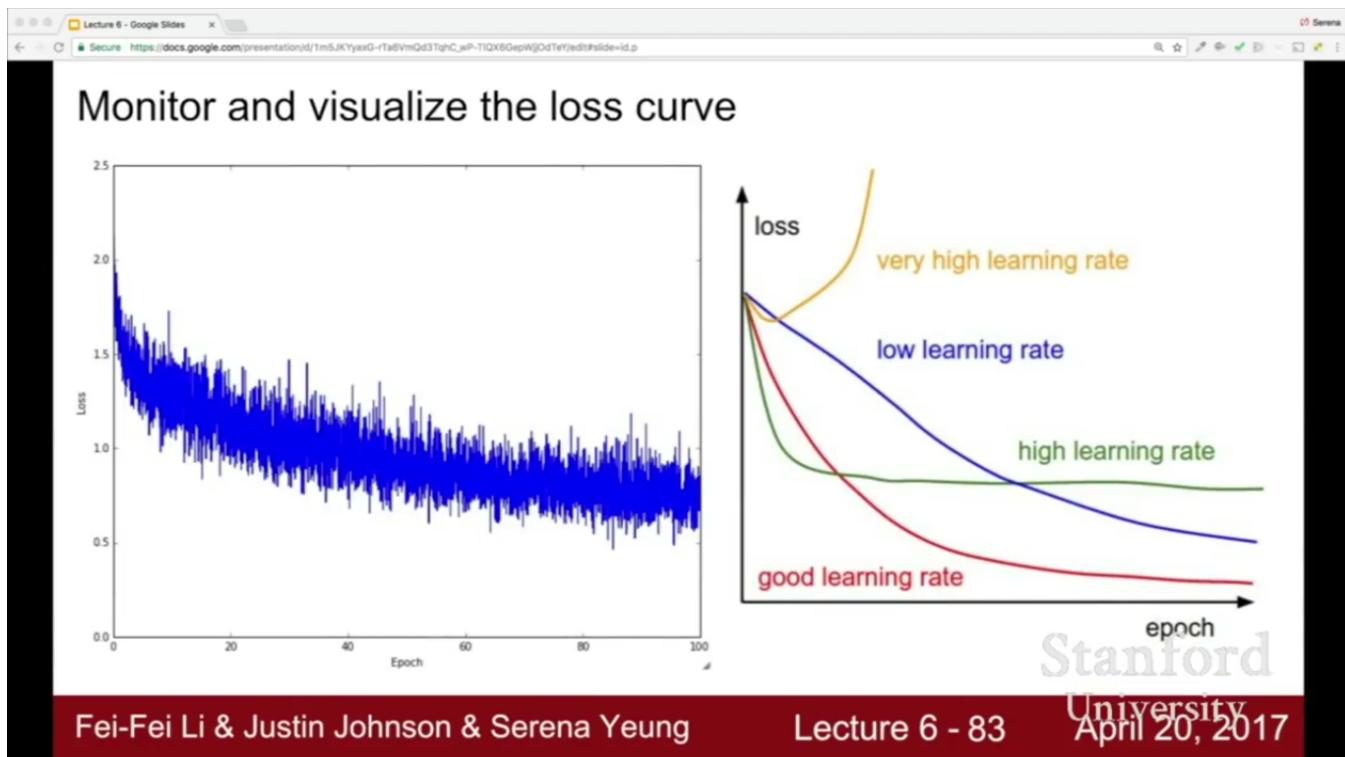
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562880e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)

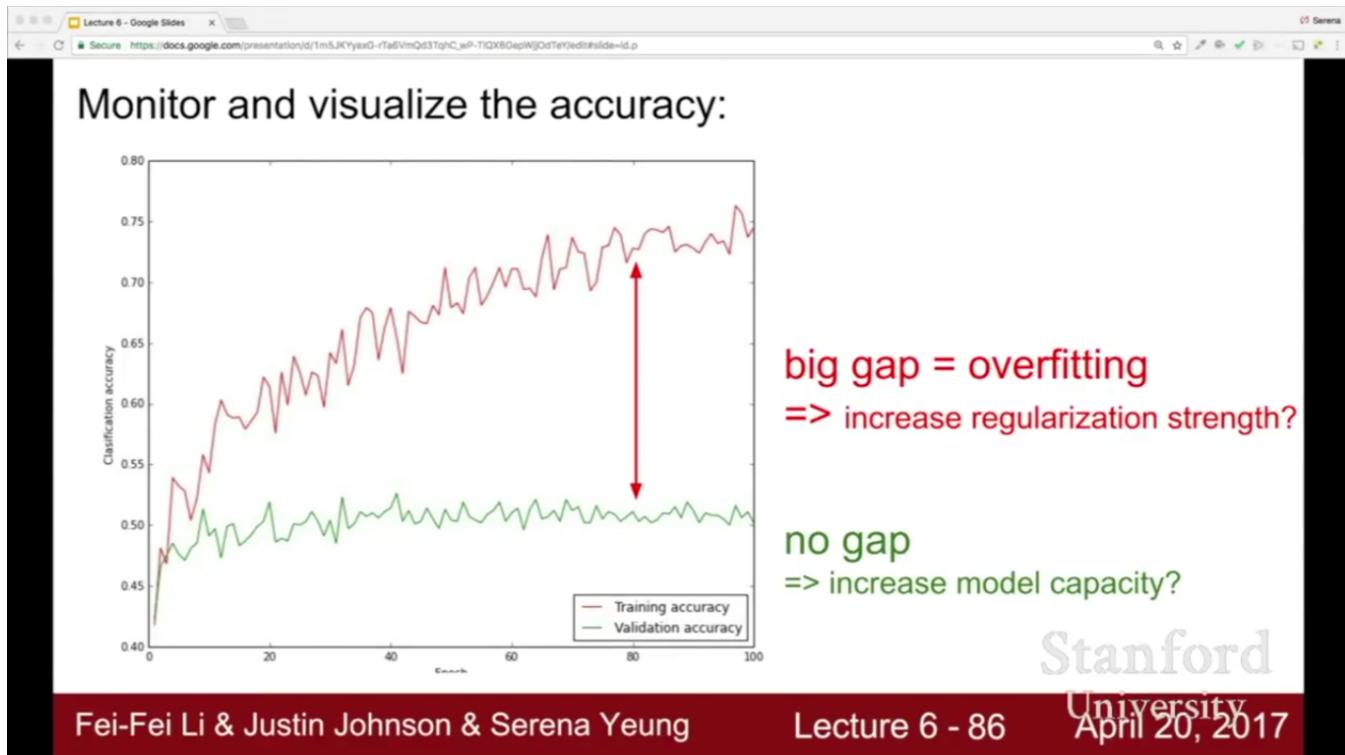
53% - relatively good for a 2-layer neural net with 50 hidden neurons.

But this best cross-validation result is worrying. Why? ←

- We can notice that the best result of 53.1% accuracy has a learning rate of almost 10^{-3} and we have considerably good result of 53% around 10^{-4} , this can be an issue since we might not have explored our learning rate space thoroughly, so better to keep a learning rate space from 10^{-6} to 10^{-1} .







Lecture 6 - Google Slides

Secure https://docs.google.com/presentation/d/1m5JKYyexQ-rtafIVmQd3TqhC_wP-TIQX6GeplWjjOdTeY/edit#slide=id.p

Track the ratio of weight updates / weight magnitudes:

```
# assume parameter vector W and its gradient vector dW
param_scale = np.linalg.norm(W.ravel())
update = -learning_rate*dW # simple SGD update
update_scale = np.linalg.norm(update.ravel())
W += update # the actual update
print update_scale / param_scale # want ~1e-3
```

ratio between the updates and values: $\sim 0.0002 / 0.02 = 0.01$ (about okay)
want this to be somewhere around 0.001 or so

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 6 - 87

Stanford University April 20, 2017

Summary

TLDRs

We looked in detail at:

- Activation Functions ([use ReLU](#))
- Data Preprocessing ([images: subtract mean](#))
- Weight Initialization ([use Xavier init](#))
- Batch Normalization ([use](#))
- Babysitting the Learning process
- Hyperparameter Optimization
([random sample hyperparams, in log space when appropriate](#))