Indian Institute of Information Technology, Allahabad





Introduction to Machine Learning

Regularization



By

Dr. Shiv Ram Dubey
Associate Professor
Computer Vision and Biometrics Lab
Department of Information Technology
Indian Institute of Information Technology, Allahabad
Web: https://profile.iiita.ac.in/srdubey/

DISCLAIMER

The content (text, image, and graphics) used in this slide are adopted from many sources for academic purposes. Broadly, the sources have been given due credit appropriately. However, there is a chance of missing out some original primary sources. The authors of this material do not claim any copyright of such material.

Regularization

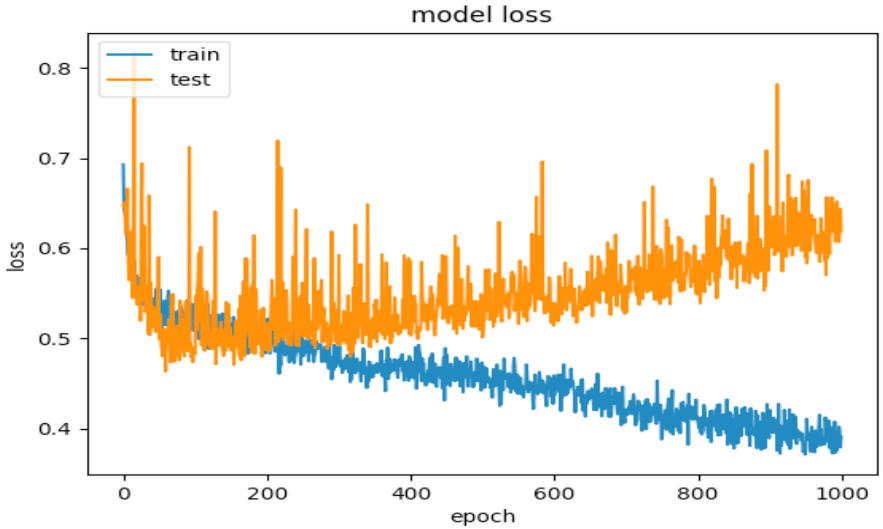
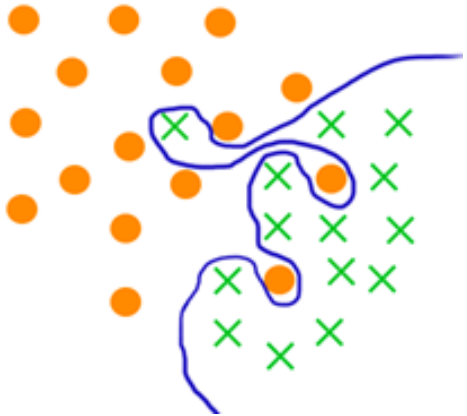
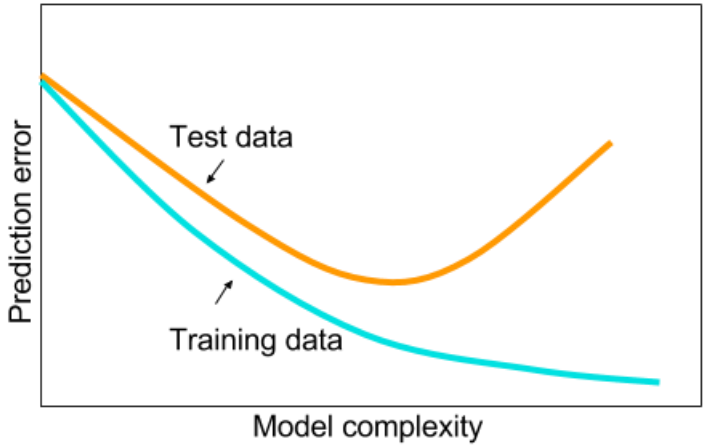


Image Source: https://stackoverflow.com/questions/44909134/how-to-avoid-overfitting-on-a-simple-feed-forward-network/44985765

Regularization

- Techniques for controlling the capacity of a neural network to prevent overfitting





Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}$$

Data loss: Model predictions should match training data

Source: cs231n

Regularization

λ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{} + \underbrace{\lambda R(W)}_{}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Source: cs231n

Regularization

λ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Simple examples

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Source: cs231n

Regularization

λ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Why regularize?

- Express preferences over weights
- Make the model *simple* so it works on test data
- Improve optimization by adding curvature

Source: cs231n

Regularization

$$x = [1,1,1,1]$$

$$w_1 = [1,0,0,0]$$

$$w_2 = [0.25,0.25,0.25,0.25]$$

Source: cs231n

Regularization

$$x = [1,1,1,1]$$

$$w_1 = [1,0,0,0]$$

$$w_2 = [0.25,0.25,0.25,0.25]$$

$$w_1 \,.x = w_2 \,.x = 1$$

Source: cs231n

Regularization

$$x = [1,1,1,1]$$

$$w_1 = [1,0,0,0]$$

$$w_2 = [0.25,0.25,0.25,0.25]$$

$$w_1 \,.x = w_2 \,.x = 1$$

Which W to consider?

Source: cs231n

Regularization

$$x = [1,1,1,1]$$

$$w_1 = [1,0,0,0]$$

$$w_2 = [0.25,0.25,0.25,0.25]$$

$$w_1 \,.x = w_2 \,.x = 1$$

L2 Regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

Source: cs231n

Regularization

$x = [1,1,1,1]$

$w_1 = [1,0,0,0]$

$w_2 = [0.25,0.25,0.25,0.25]$

$w_1 \,.x = w_2 \,.x = 1$

L2 regularization likes to "spread out" the weights

L2 Regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

Other types of regularization

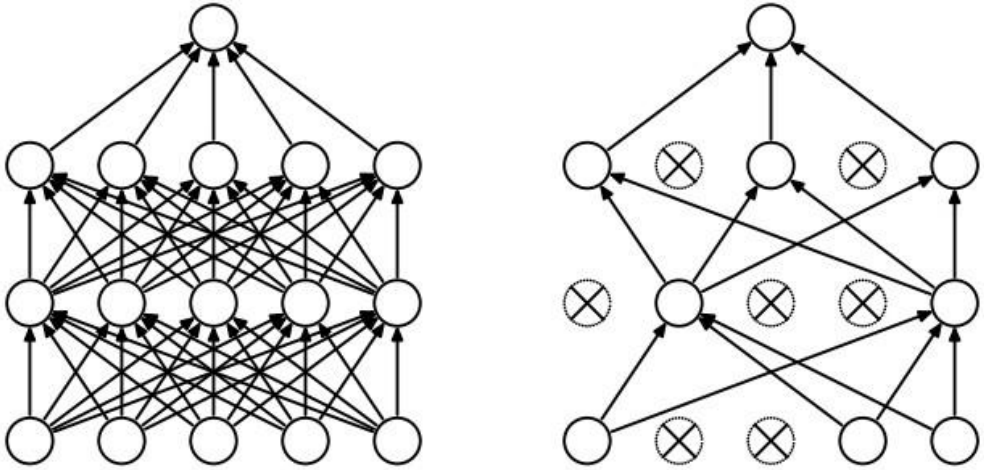
- Dropout
- Dropconnect
- Batch Normalization
- Data Augmentation

Dropout

Dropout

In each forward pass, randomly set some neurons to zero

Probability of dropping is a hyperparameter; 0.5 is common



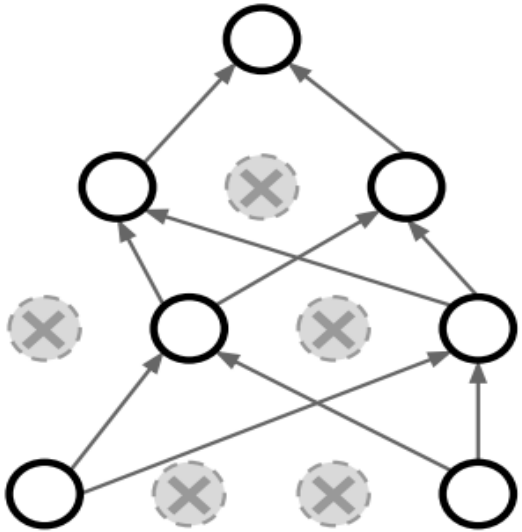
(a) Standard Neural Net

(b) After applying dropout.

Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

Dropout

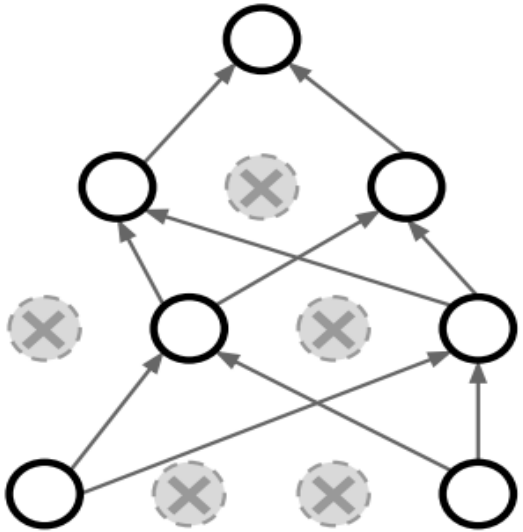
How can this possibly be a good idea?



Source: cs231n

Dropout

How can this possibly be a good idea?

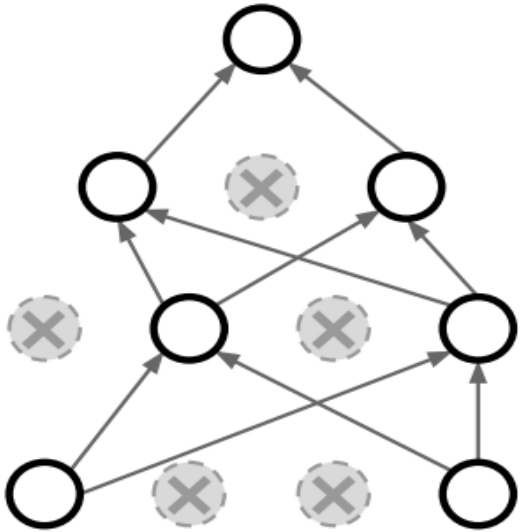


- Intuitions
  - Prevent "co-adaptation" of units, increase robustness to noise
  - Train *implicit ensemble*

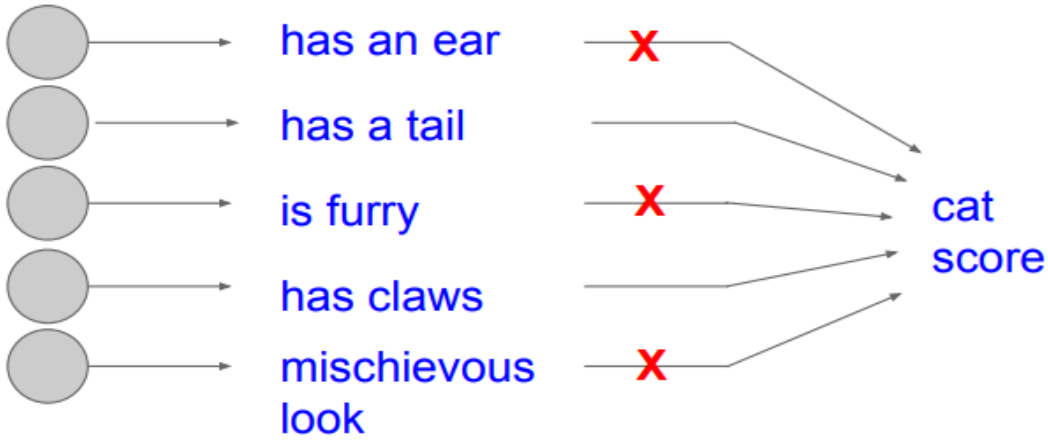
Source: cs231n

Dropout

How can this possibly be a good idea?



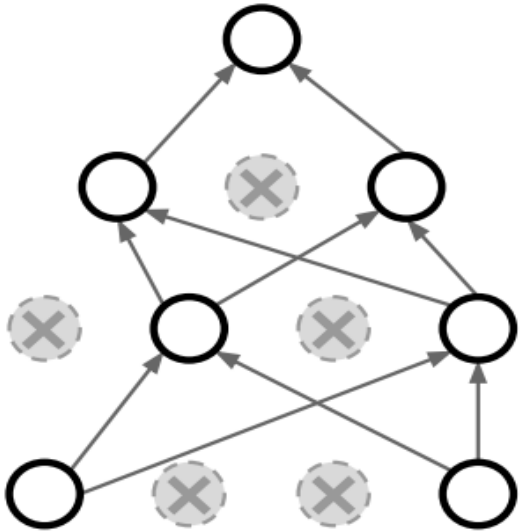
Forces the network to have a redundant representation;
Prevents co-adaptation of features



Source: cs231n

Dropout

How can this possibly be a good idea?



Dropout is training a large ensemble of models (that share parameters).

Intuition: successful conspiracies

- 50 people planning a conspiracy
- Strategy A: plan a big conspiracy involving 50 people
  - Likely to fail. 50 people need to play their parts correctly.
- Strategy B: plan 10 conspiracies each involving 5 people
  - Likely to succeed!

Source: cs231n

Dropout: Test Time

```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

At test time all neurons are active always
=> We must scale the activations so that for each neuron:
output at test time = expected output at training time

More common: "Inverted dropout"

Source: cs231n

Dropout: More common: “Inverted dropout”

We drop and scale at train time and don't do anything at test time.

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

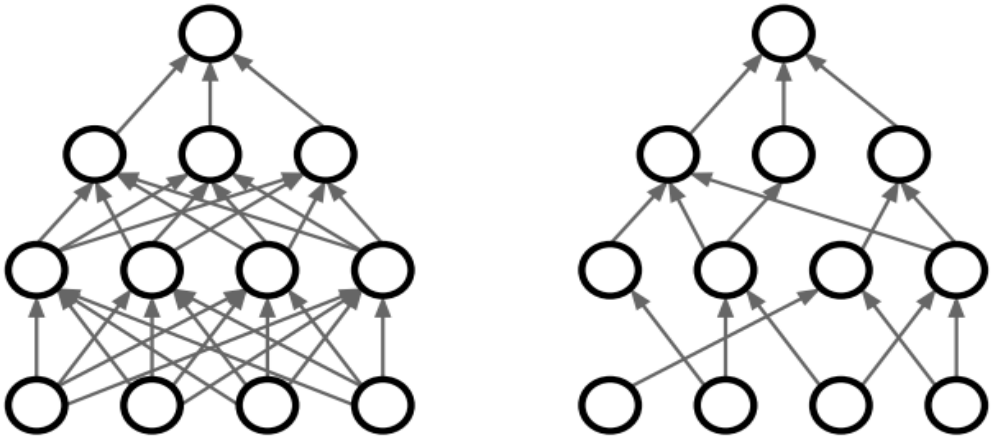
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

test time is unchanged!

Source: cs231n

DropConnect

Dropping some connections



Wan et al, "Regularization of Neural Networks using DropConnect", ICML 2013

Source: cs231n

Batch Normalization

Batch Normalization

"We want zero-mean unit-variance activations? lets make them so."

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]

Source: cs231n

Batch Normalization

"We want zero-mean unit-variance activations? lets make them so."

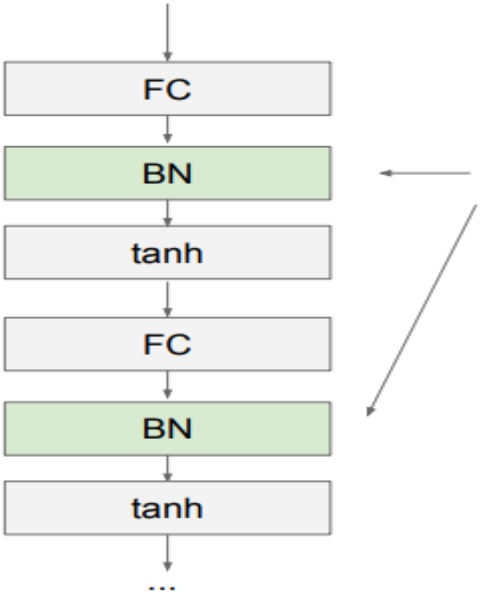
consider a batch of activations at some layer. To make each dimension zero-mean unit-variance, apply:

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

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]

Source: cs231n

Batch Normalization

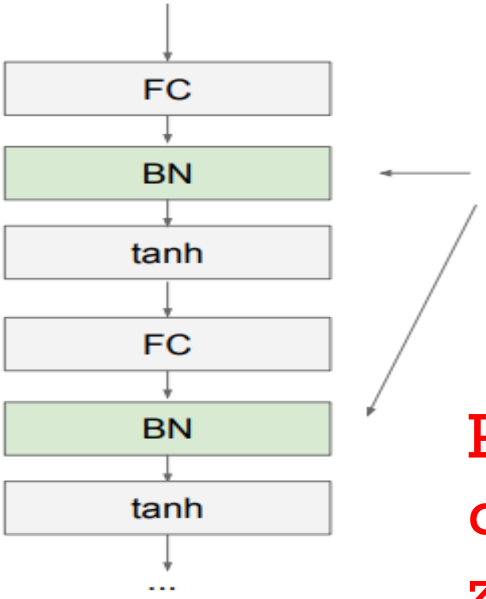


Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]

Source: cs231n

Batch Normalization



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

Problem:
do we necessarily want a
zero-mean unit-variance input?

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]

Source: cs231n

Batch Normalization

Normalize:

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

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

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

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]

Source: cs231n

Batch Normalization

Normalize:

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

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

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

Note, the network can learn:

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

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

to recover the identity mapping.

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]

Source: cs231n

Batch Normalization

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

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

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

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

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

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

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]

Source: cs231n

Batch Normalization

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)

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]

Source: cs231n

Batch Normalization

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

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

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

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

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

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

At test time (usually):

// mean training set

// variance training set

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]

Source: cs231n

Batch Normalization

Benefits

- Improves gradient flow through the network
- Allows higher learning rates and Accelerates convergence of training
- Reduces the strong dependence on initialization
- Acts as a form of regularization

Pitfalls

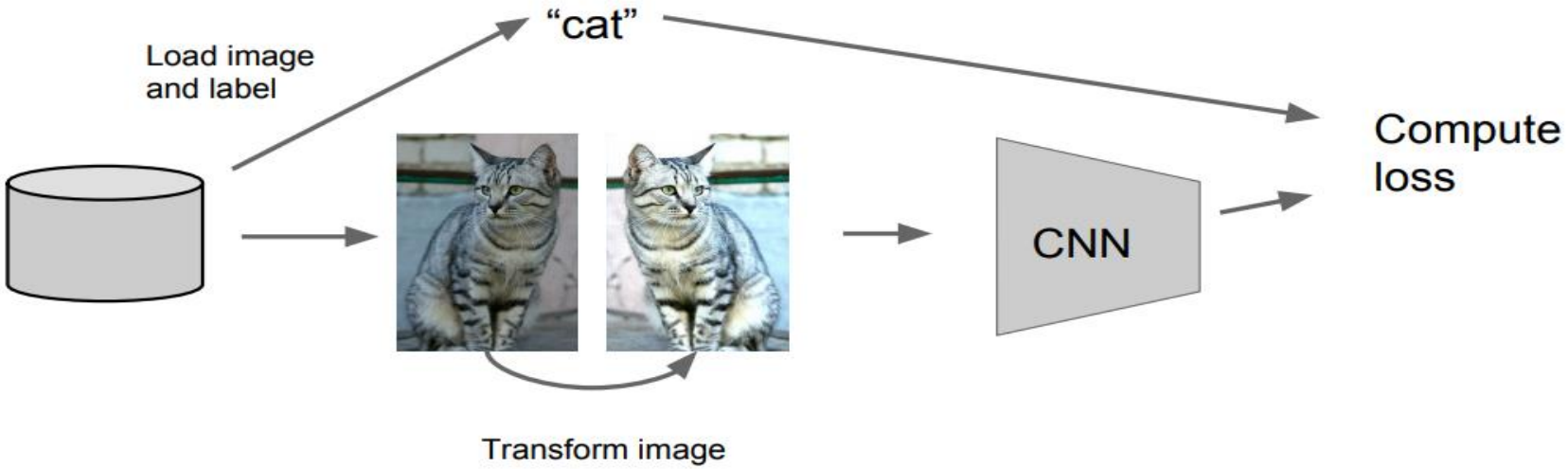
- Behavior depends on composition of mini-batches, can lead to hard-to-catch bugs if there is a mismatch between training and test regime (example)
- Doesn't work well for small mini-batch sizes
- Cannot be used in recurrent models

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]

Source: cs231n

Data Augmentation

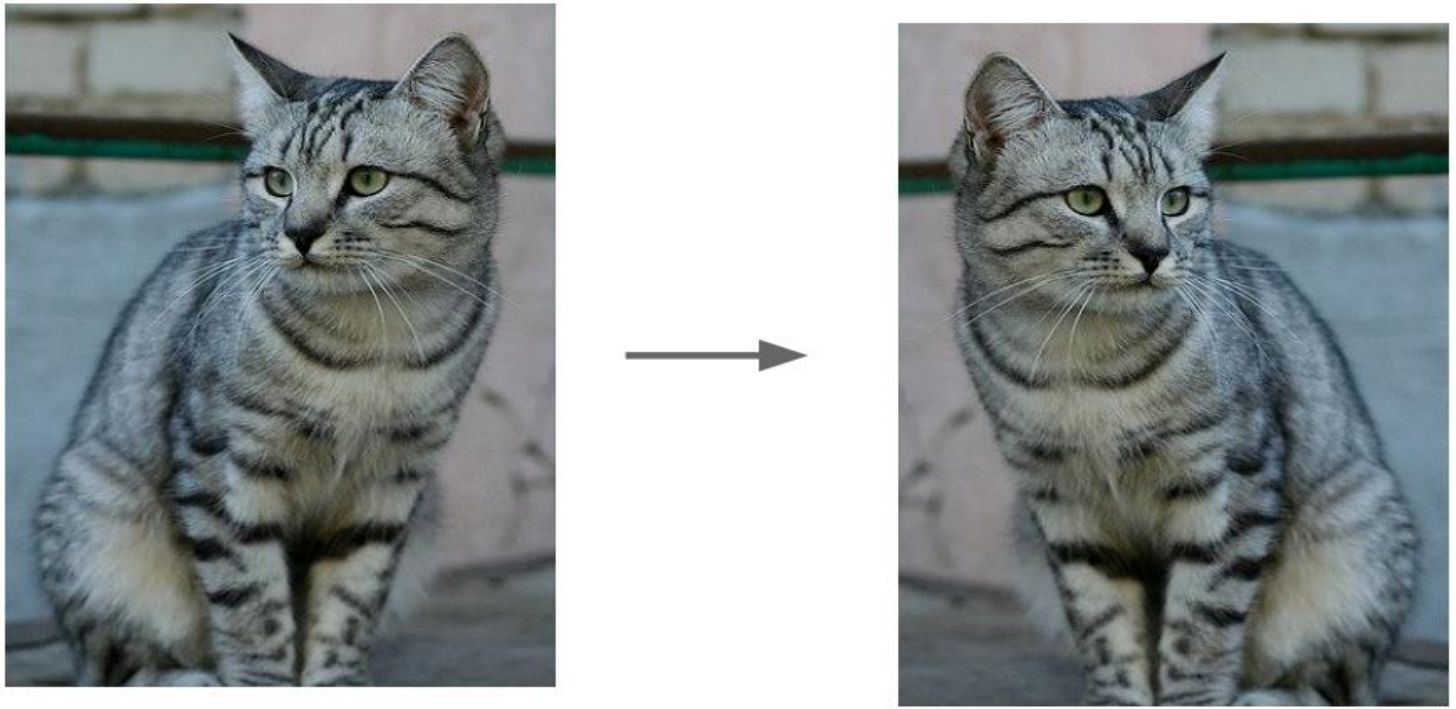
Data Augmentation (Jittering)



Source: cs231n

Data Augmentation (Jittering)

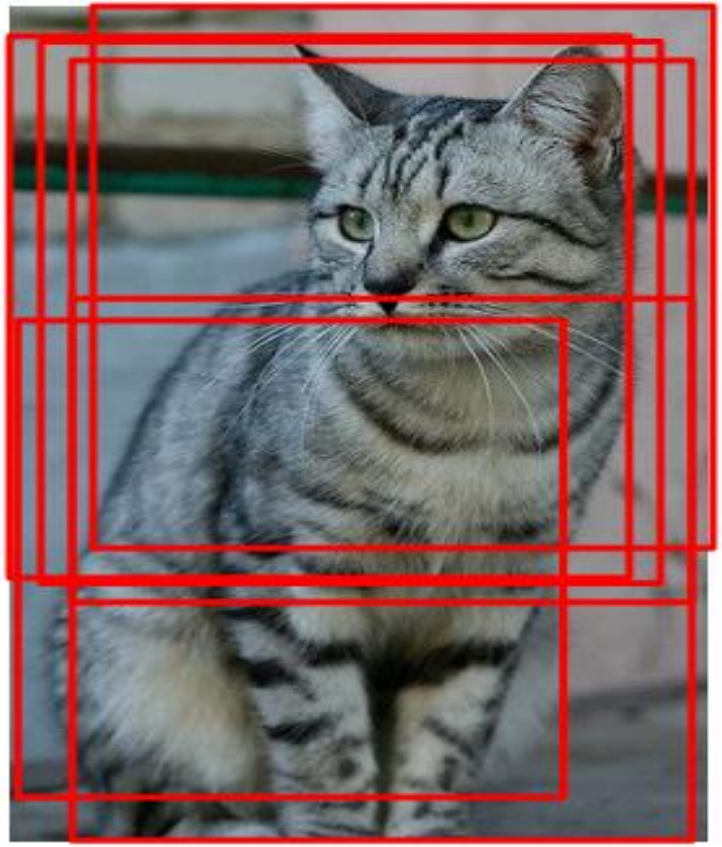
Horizontal Flips



Source: cs231n

Data Augmentation (Jittering)

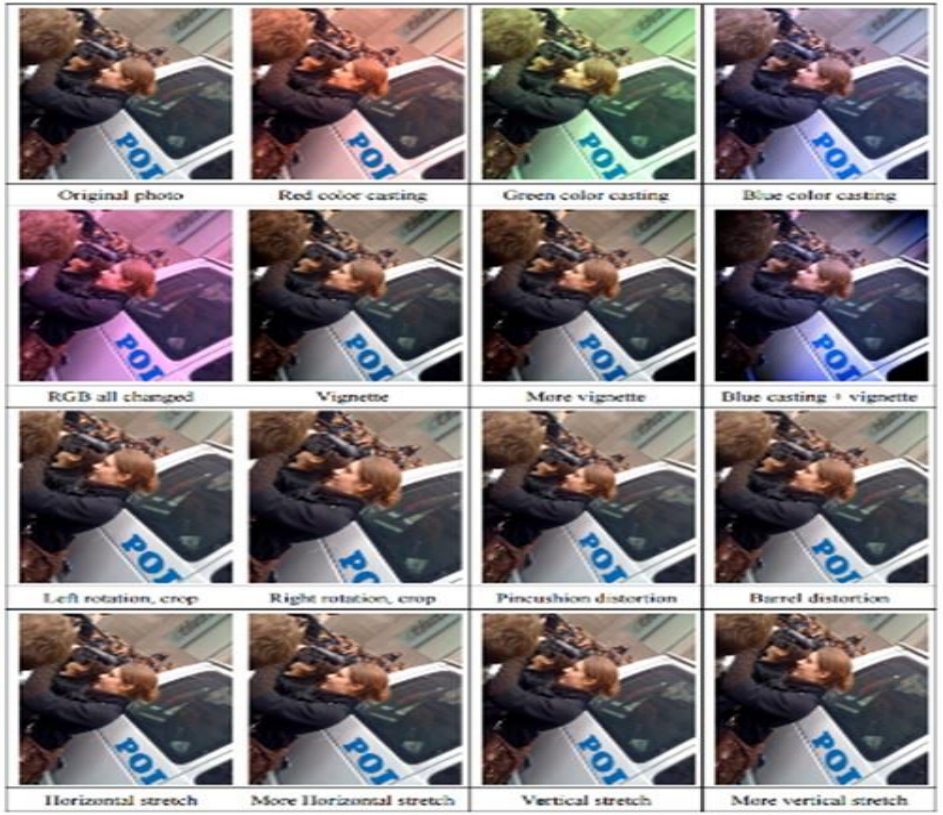
Random crops and scales



Source: cs231n

Data Augmentation (Jittering)

- Create *virtual* training samples
- Get creative for your problem!
  - Horizontal flip
  - Random crop
  - Color casting
  - Randomize contrast
  - Randomize brightness
  - Geometric distortion
  - Rotation
  - Photometric changes



Deep Image [Wu et al. 2015]

When to stop training?

- Monitor validation error to decide when to stop
  - "Patience" hyperparameter: number of epochs without improvement before stopping
  - *Early stopping* can be viewed as a kind of regularization

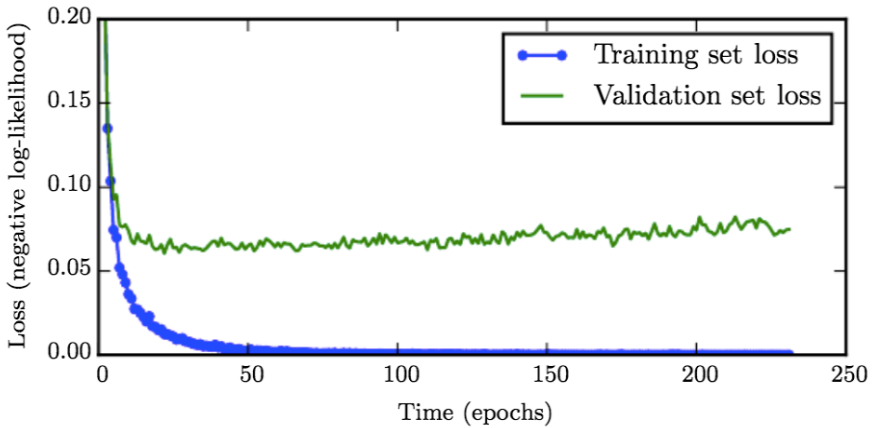


Figure from Deep Learning Book

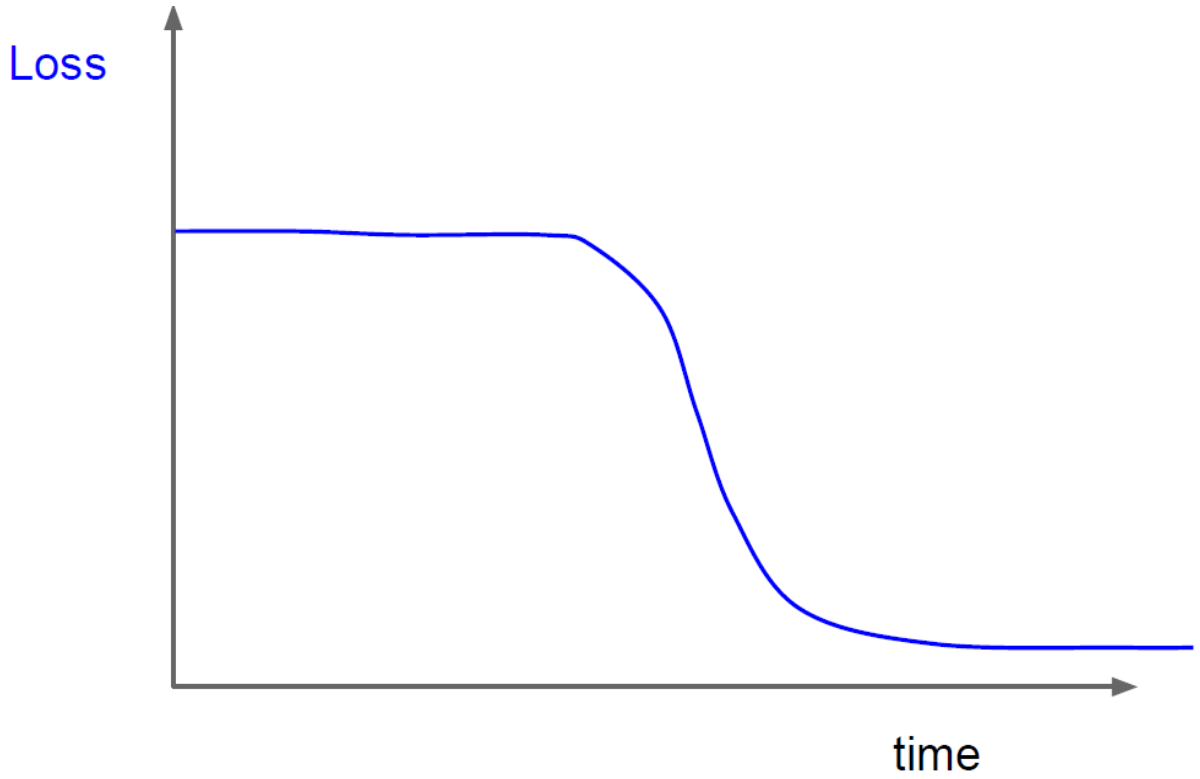
Setting the mini-batch size

- Larger mini-batches: more expensive and less frequent updates, lower gradient variance, more parallelizable

- SGD with larger batches may generalize more poorly (e.g., Keskar et al., 2017)
  - But can be made to work well by carefully controlling learning rate and addressing other optimization issues (Goyal et al., 2018)

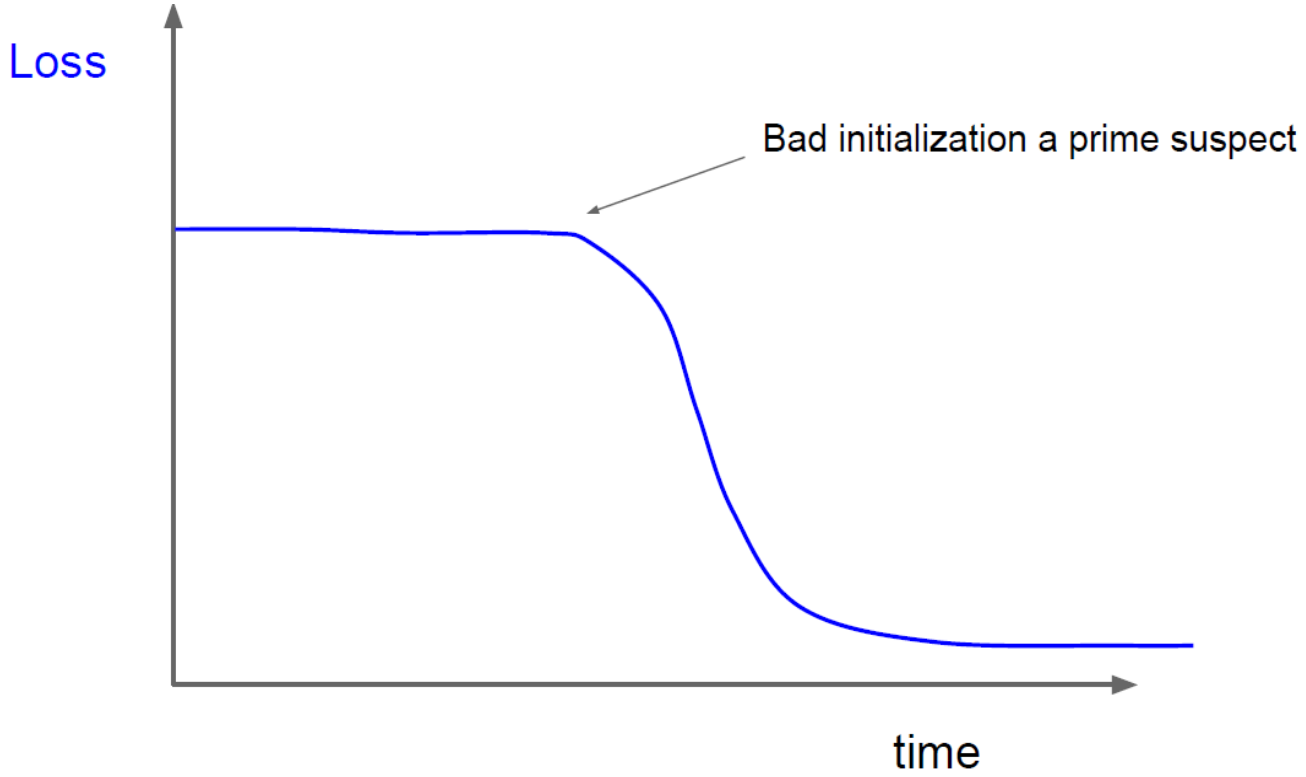
MONITOR AND VISUALIZE THE LOSS CURVE

Monitor and visualize the loss curve



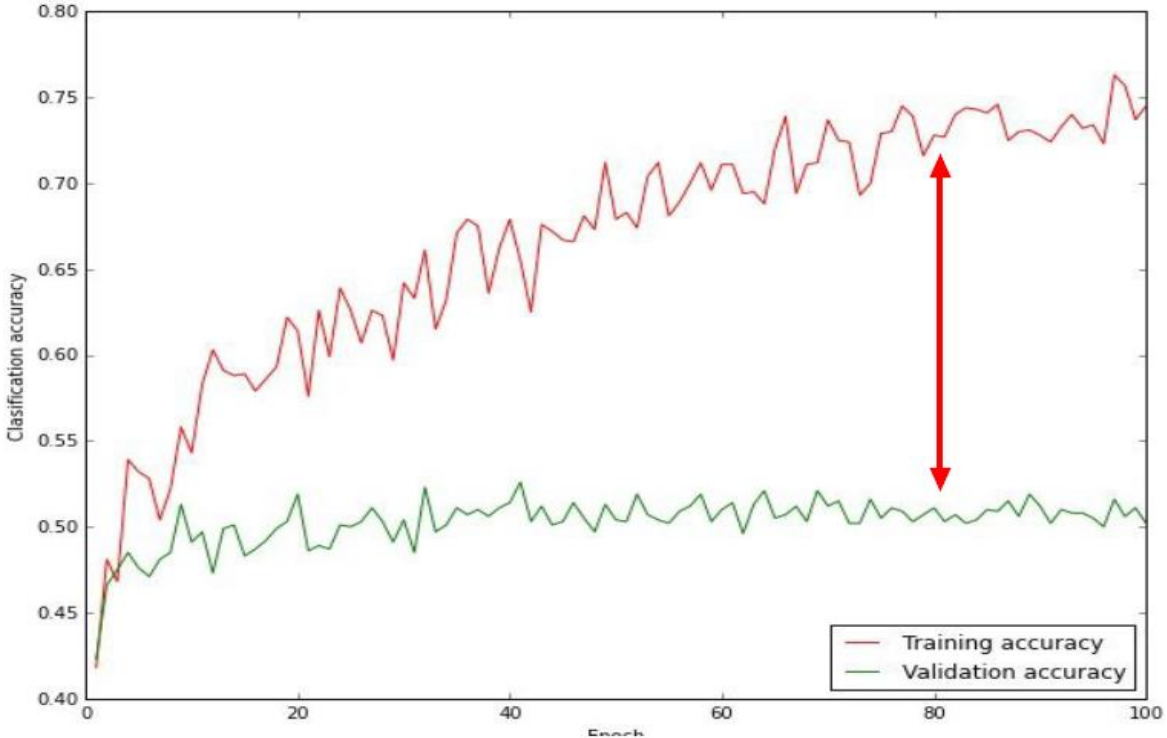
Source: cs231n

Monitor and visualize the loss curve



Source: cs231n

Monitor and visualize the loss curve

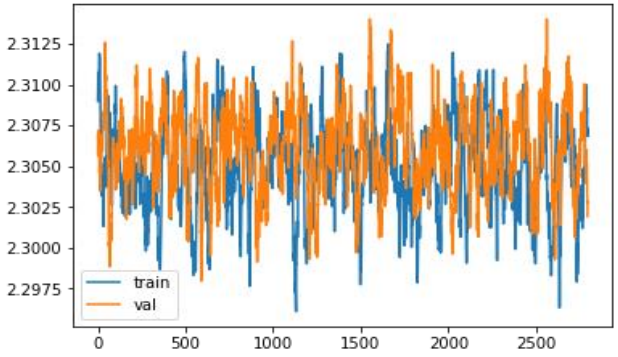


big gap = overfitting
=> increase
regularization strength?

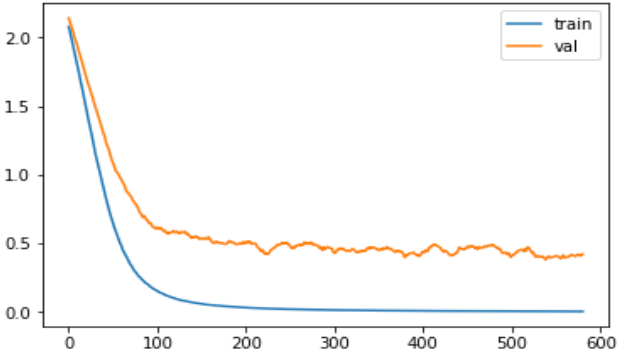
no gap
=> increase model
capacity?

Source: cs231n

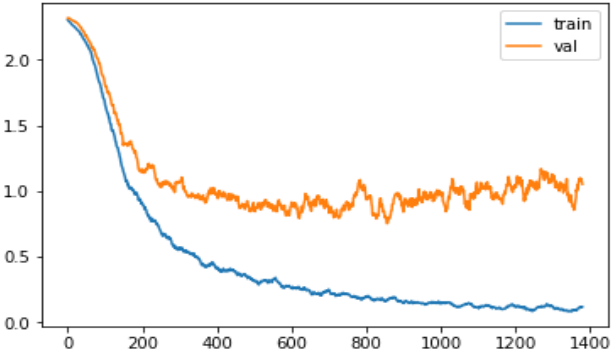
Monitor and visualize the loss curve



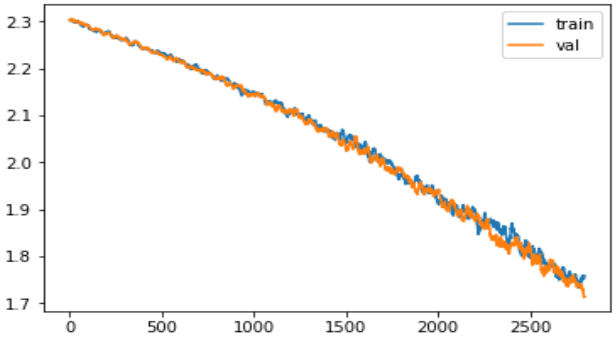
Not learning: gradients not applied to weights



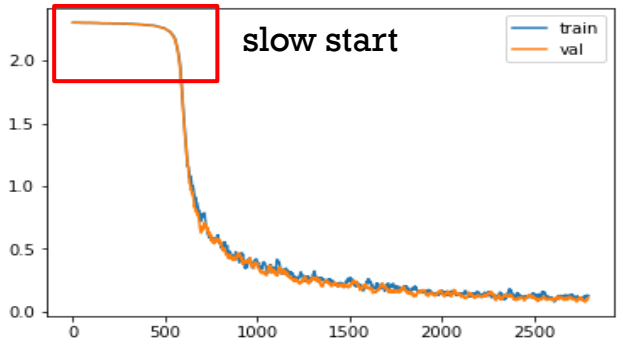
Overfit: model too large/dataset too small



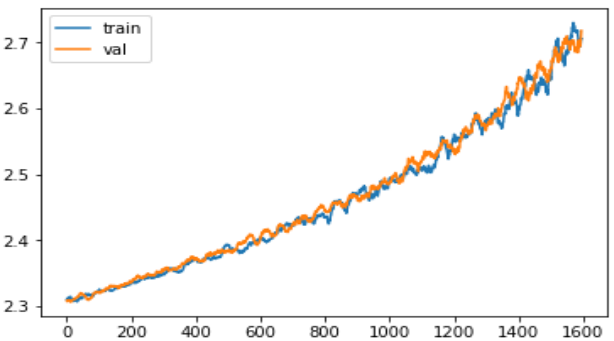
More extreme case of overfitting



Not converged yet: need longer training



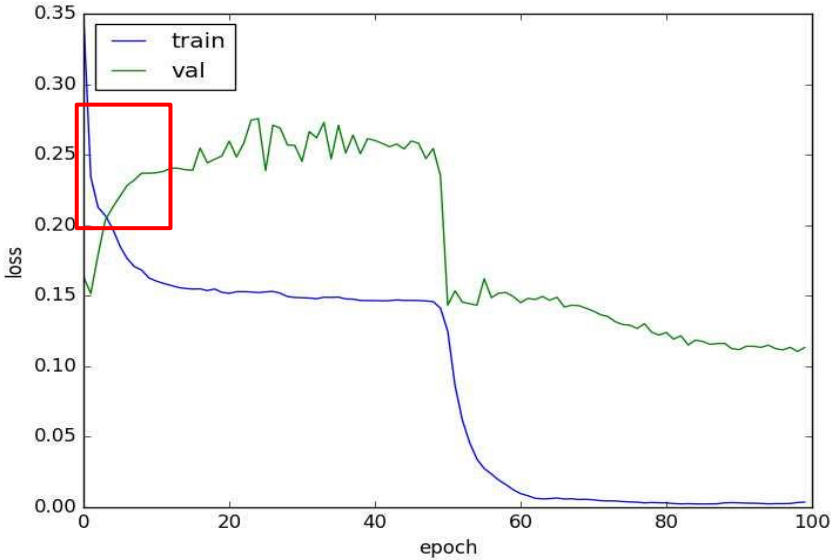
Slow start: initialization weights too small



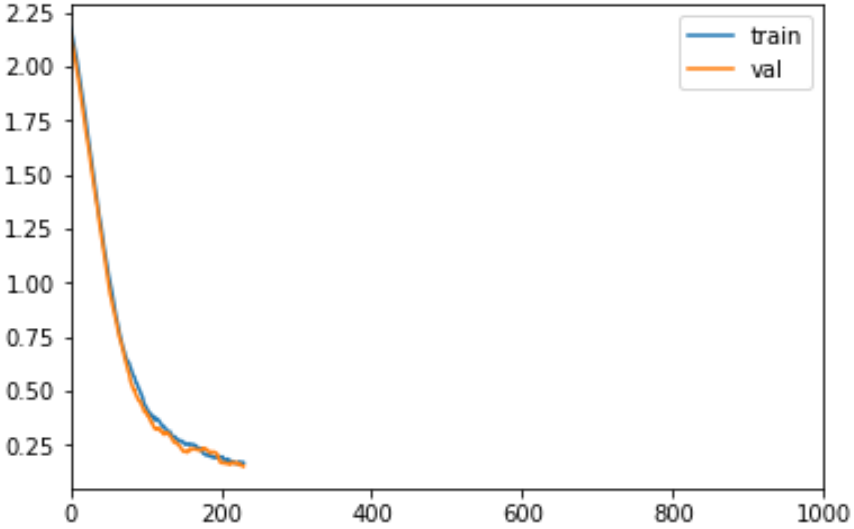
Applied the negative of gradients

Source: cs231n

Monitor and visualize the loss curve



Problem: val set too small, statistics not meaningful



Get nans in the loss after a number of iterations: caused by high learning rate and numerical instability in models

Source: cs231n

Attempt at a conclusion

- Training neural networks is still a black art
- Process requires close "babysitting"
- For many techniques, the reasons why, when, and whether they work are in active dispute – read everything but don't trust anything
- It all comes down to (principled) trial and error
- Further reading: A. Karpathy, A recipe for training neural networks



Acknowledgement

Thanks to the following courses and corresponding researchers for making their teaching/research material online

- Deep Learning, Stanford University
- Introduction to Deep Learning, University of Illinois at Urbana-Champaign
- Introduction to Deep Learning, Carnegie Mellon University
- Convolutional Neural Networks for Visual Recognition, Stanford University
- Natural Language Processing with Deep Learning, Stanford University
- And Many More .....

THANK YOU