Machine Learning Course

Lecture 10: Diving into Deep Learning

Outline

1. Previous lecture recap: backpropagation, activations, intuition.
2. Optimizers.
3. Regularization.
4. PyTorch practice.
5. Q & A.

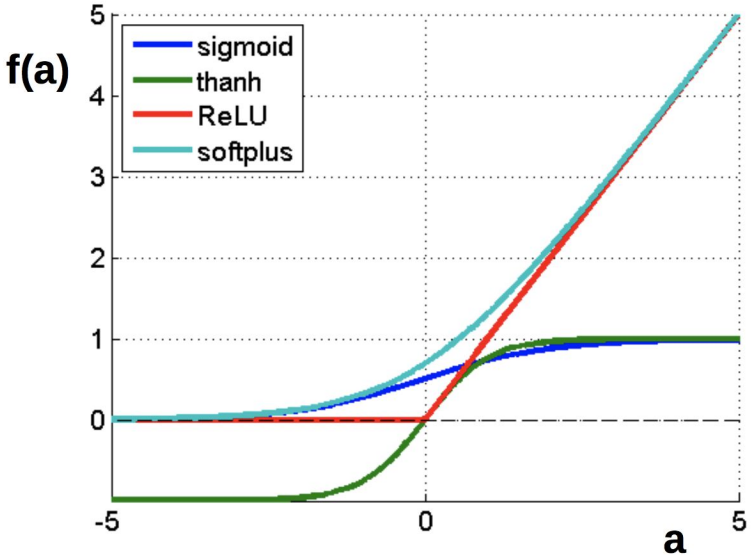
Once more: nonlinearities

$$f(a) = \frac{1}{1 + e^a}$$

$$f(a) = \tanh(a)$$

$$f(a) = \max(0, a)$$

$$f(a) = \log(1 + e^a)$$

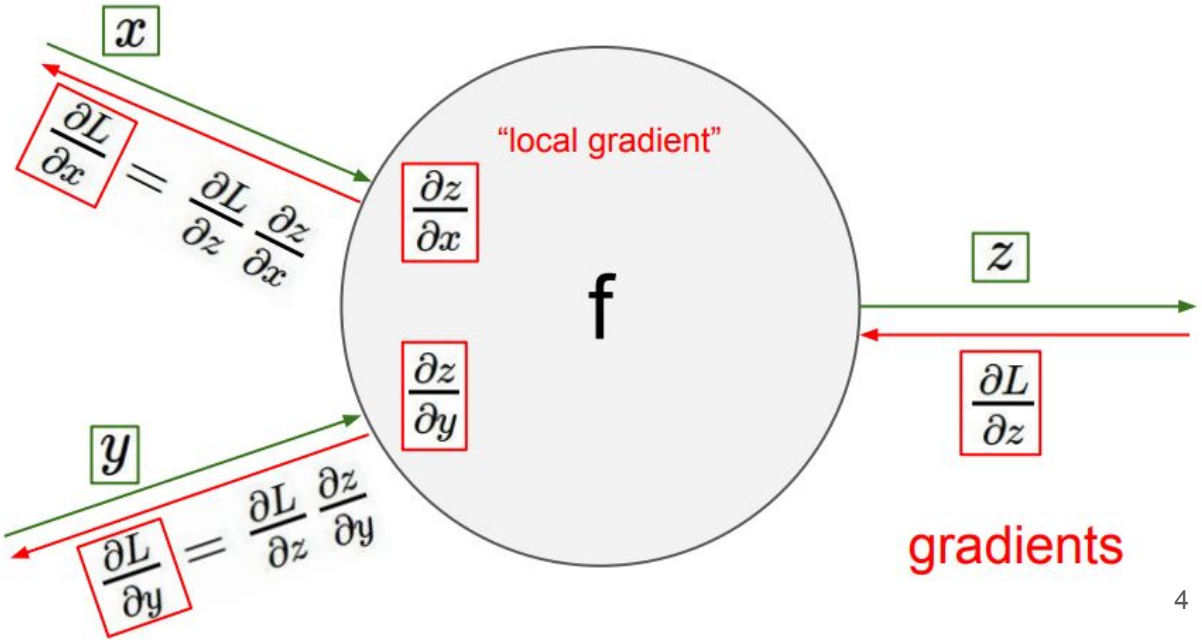


Backpropagation and chain rule

Chain rule is just simple math:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$$

Backprop is just way to use it in NN training.



source: http://cs231n.github.io

Different layers

- Layers
  a. Dense layer *(done)*
  b. Convolutional layer *(next lecture)*
  c. Pooling layer *(next lecture)*
  d. Dropout layer *(today)*
  e. Batchnorm layer (batch normalization) *(today)*
  f. Embeddings (aka word2vec, GloVe) *(previous lecture)*
  g. Recurrent layer~~s~~ *(last lecture)*

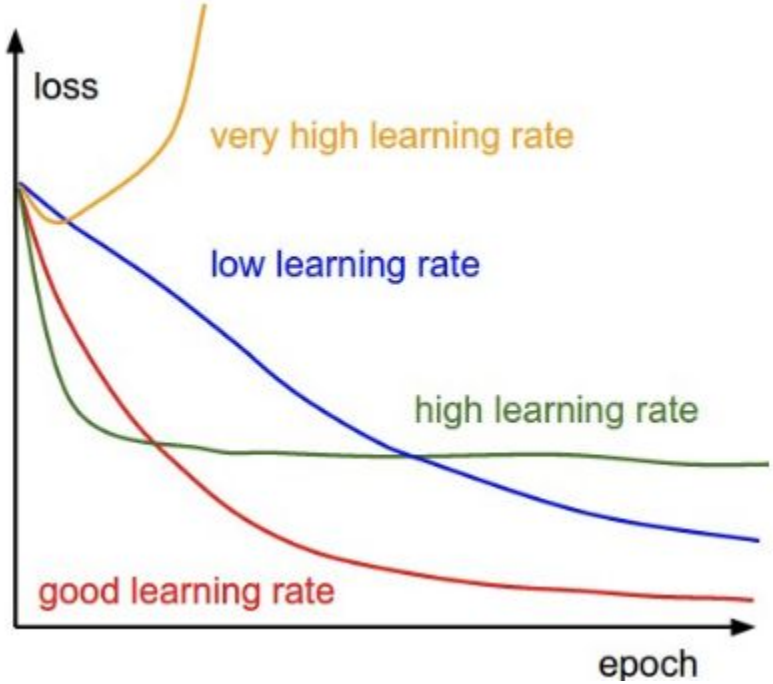
Different layers

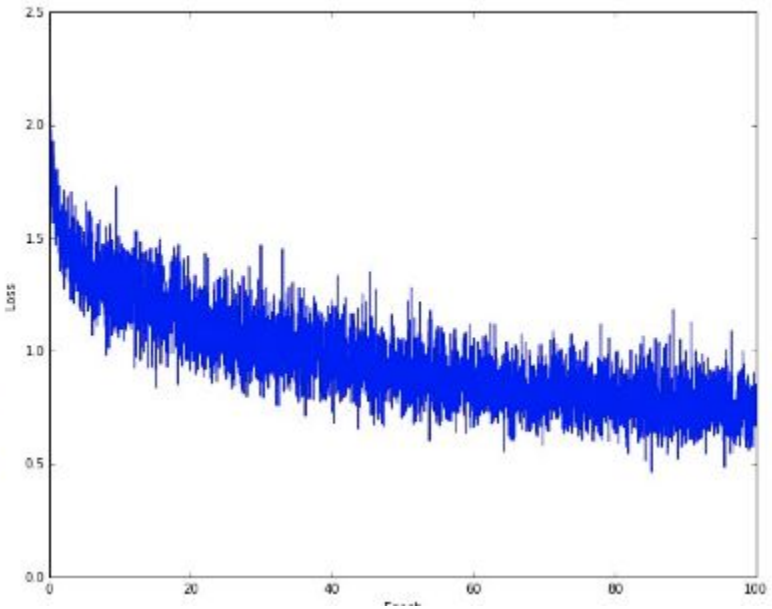
- Layers
  a. Dense layer *(done)*
  b. Convolutional layer *(next lecture)*
  c. Pooling layer *(next lecture)*
  d. Dropout layer *(today)*
  e. Batchnorm layer (batch normalization) *(today)*
  f. Embeddings (aka word2vec, GloVe) *(last lecture)*
  g. Recurrent layers *(last lecture)*

Optimizers

Stochastic gradient descent is used to optimize NN parameters.

$$x_{t+1} = x_t - \text{learning rate} \cdot dx$$



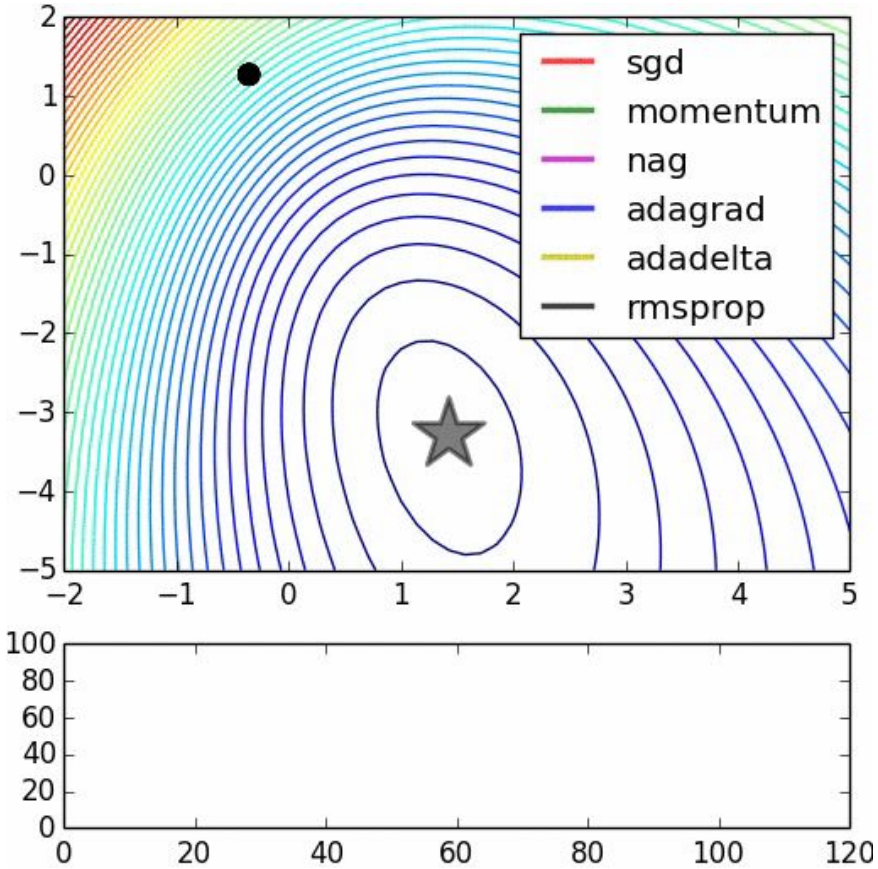


source: http://cs231n.github.io/neural-networks-3/

Optimizers

There are much more optimizers:

- Momentum
- Adagrad
- Adadelta
- RMSprop
- Adam
- …
- even other NNs

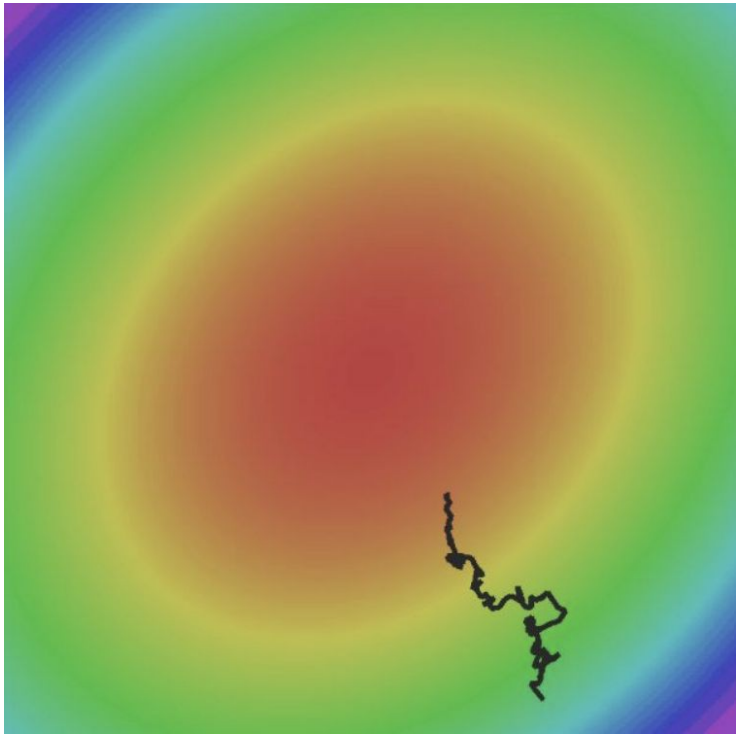


Optimization: SGD

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

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

Averaging over minibatches ---> noisy gradient



source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

First idea: momentum

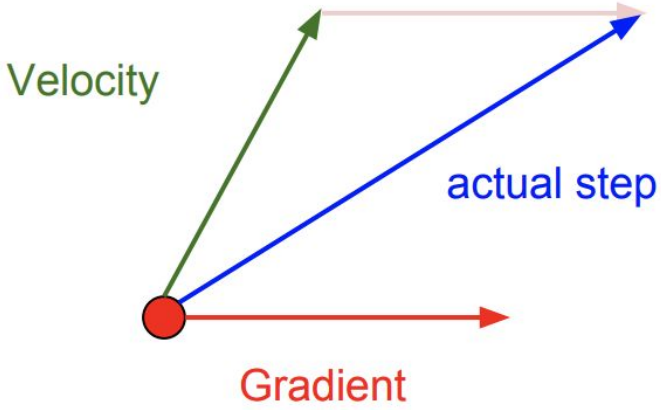
Simple SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

SGD with momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

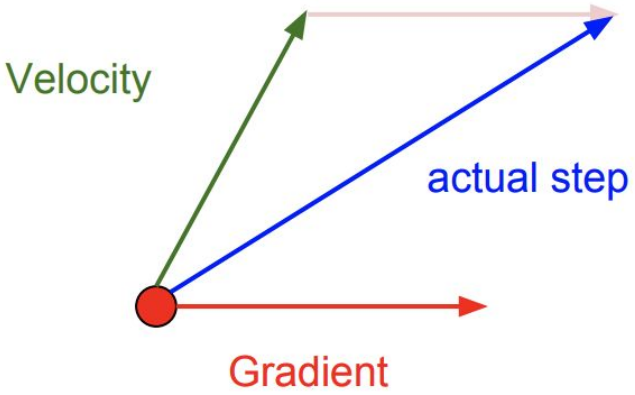
Momentum update:



source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

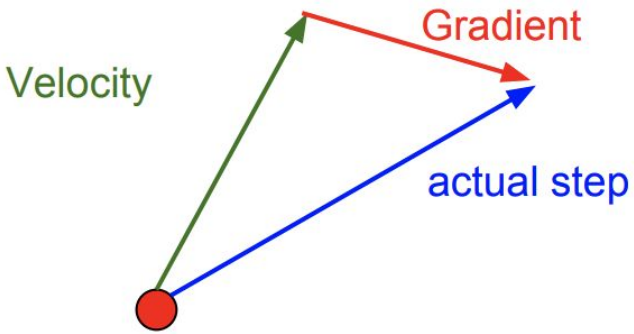
Nesterov momentum

Momentum update:



$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

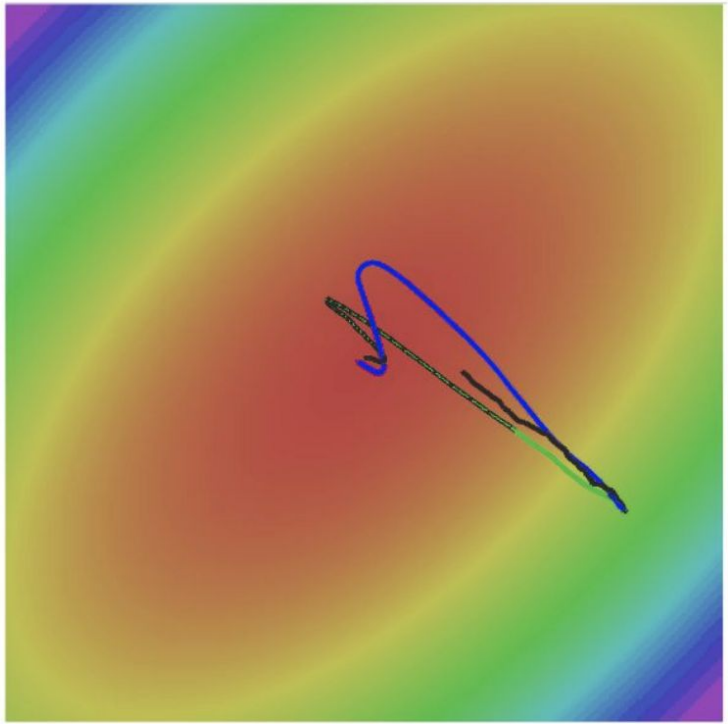
Nesterov Momentum



$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$
$$x_{t+1} = x_t + v_{t+1}$$

source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Comparing momentums



SGD

SGD+Momentum

Nesterov

source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Second idea: different dimensions are different

Adagrad: SGD with cache

$$\text{cache}_{t+1} = \text{cache}_t + (\nabla f(x_t))^2$$

$$x_{t+1} = x_t - \alpha \frac{\nabla f(x_t)}{\text{cache}_{t+1} + \varepsilon}$$

Second idea: different dimensions are different

Adagrad: SGD with cache

$$\text{cache}_{t+1} = \text{cache}_t + (\nabla f(x_t))^2$$

$$x_{t+1} = x_t - \alpha \frac{\nabla f(x_t)}{\text{cache}_{t+1} + \varepsilon}$$

Problem: gradient fades with time

Second idea: different dimensions are different

Adagrad: SGD with cache

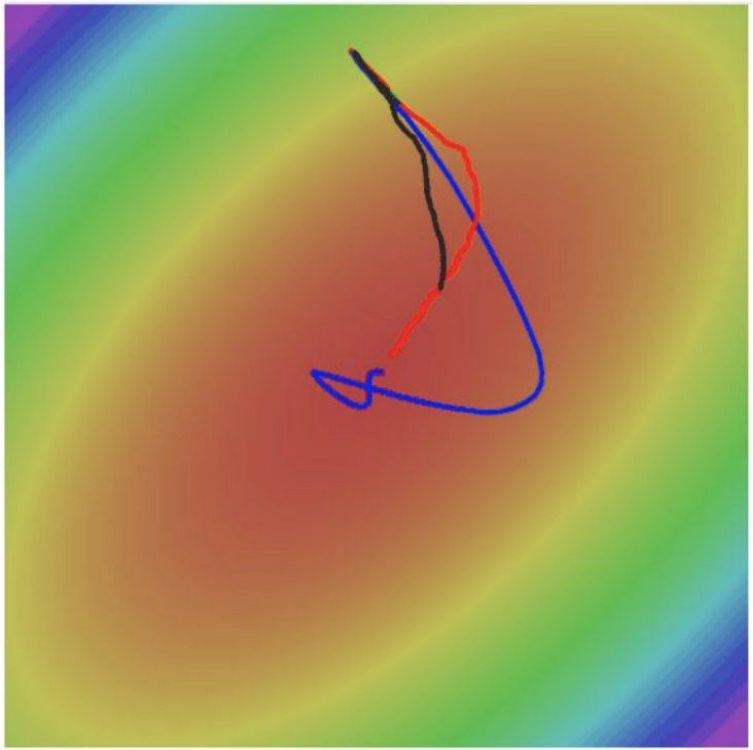
$$\text{cache}_{t+1} = \text{cache}_t + (\nabla f(x_t))^2$$

$$x_{t+1} = x_t - \alpha \frac{\nabla f(x_t)}{\text{cache}_{t+1} + \varepsilon}$$

RMSProp: SGD with cache with exp. smoothing

$$\text{cache}_{t+1} = \beta \text{cache}_t + (1-\beta)(\nabla f(x_t))^2$$

$$x_{t+1} = x_t - \alpha \frac{\nabla f(x_t)}{\text{cache}_{t+1} + \varepsilon}$$



SGD

SGD+Momentum

RMSProp

source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Adam

Let's combine the momentum idea and RMSProp normalization:

$$v_{t+1} = \gamma v_t + (1-\gamma)\nabla f(x_t)$$

$$\text{cache}_{t+1} = \beta \text{cache}_t + (1-\beta)(\nabla f(x_t))^2$$

$$x_{t+1} = x_t - \alpha \frac{v_{t+1}}{\text{cache}_{t+1} + \varepsilon}$$

Adam full form involves bias correction term. See http://cs231n.github.io/neural-networks-3/ for more info.

Adam

Let's combine the momentum idea and RMSProp normalization:

$$v_{t+1} = \gamma v_t + (1-\gamma)\nabla f(x_t)$$

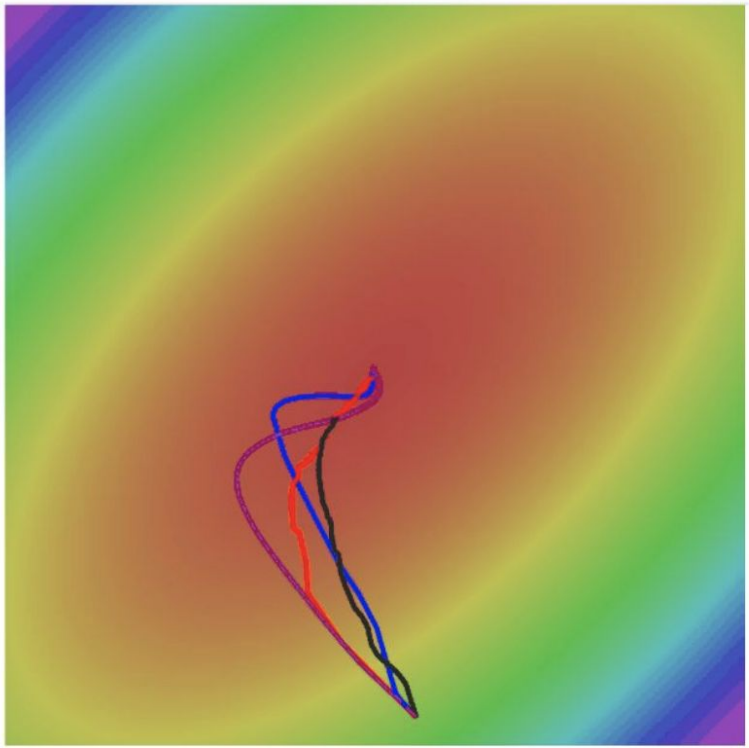
$$\text{cache}_{t+1} = \beta \text{cache}_t + (1-\beta)(\nabla f(x_t))^2$$

$$x_{t+1} = x_t - \alpha \frac{v_{t+1}}{\text{cache}_{t+1} + \varepsilon}$$

Actually, that's not quite Adam.

Adam full form involves bias correction term. See http://cs231n.github.io/neural-networks-3/ for more info.

Comparing optimizers



SGD

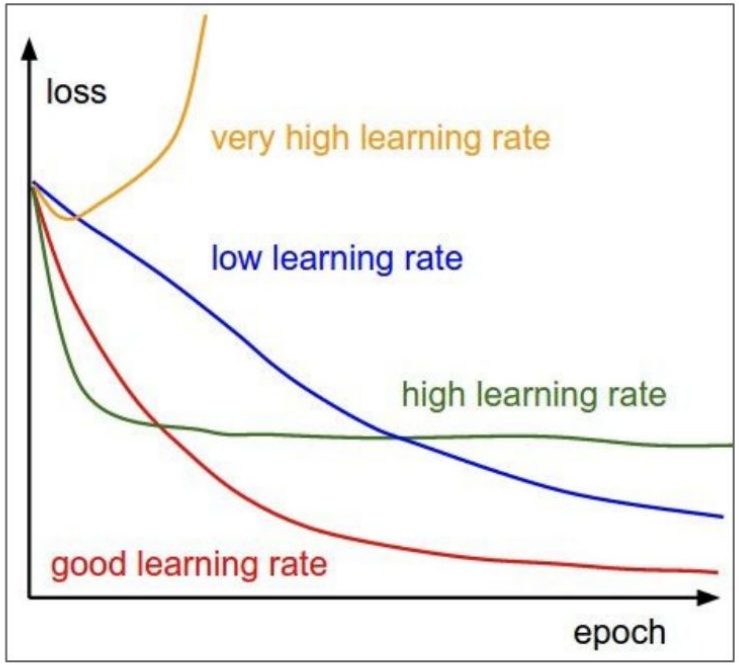
SGD+Momentum

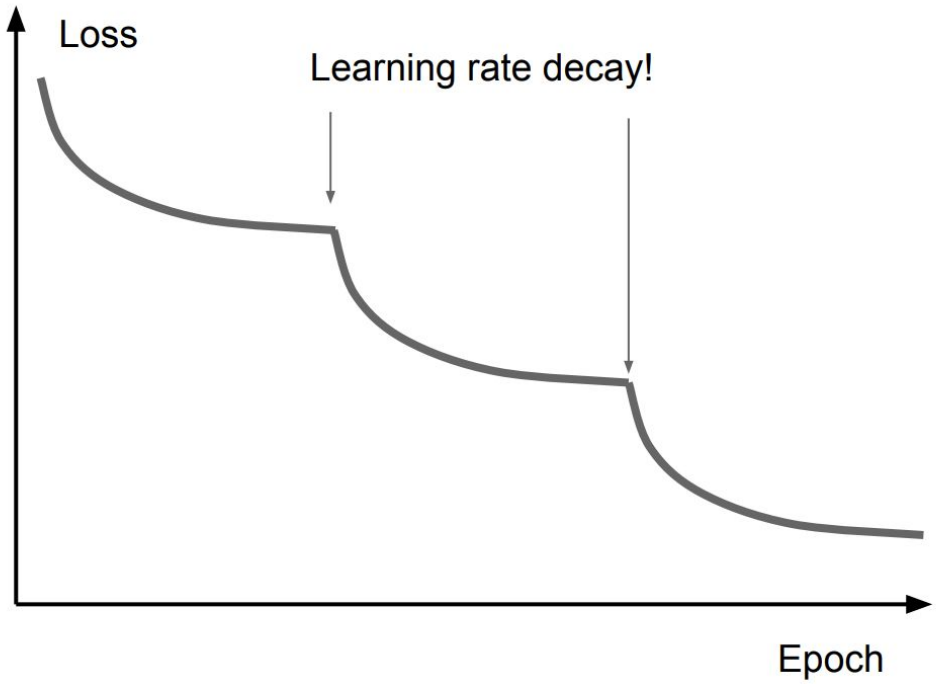
RMSProp

Adam

source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Once more: learning rate

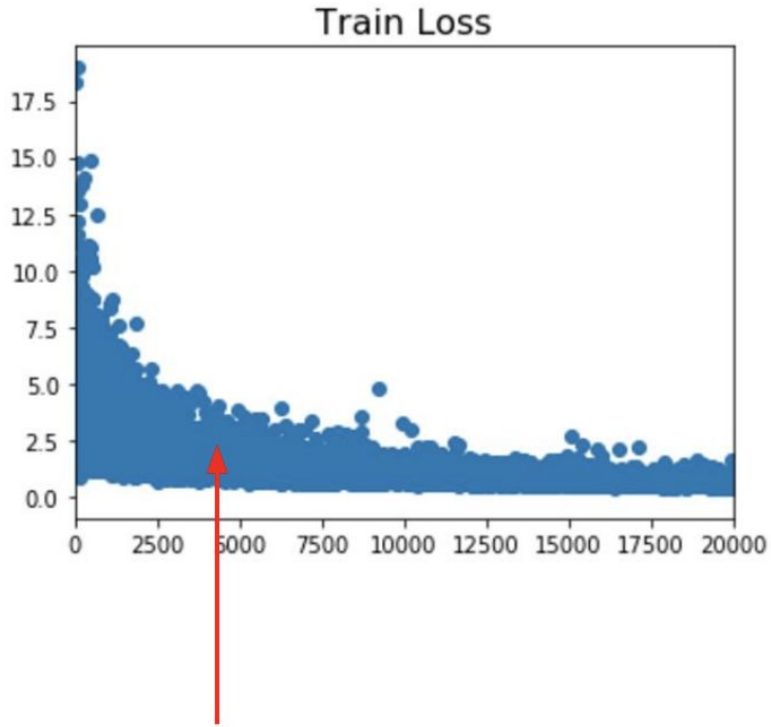


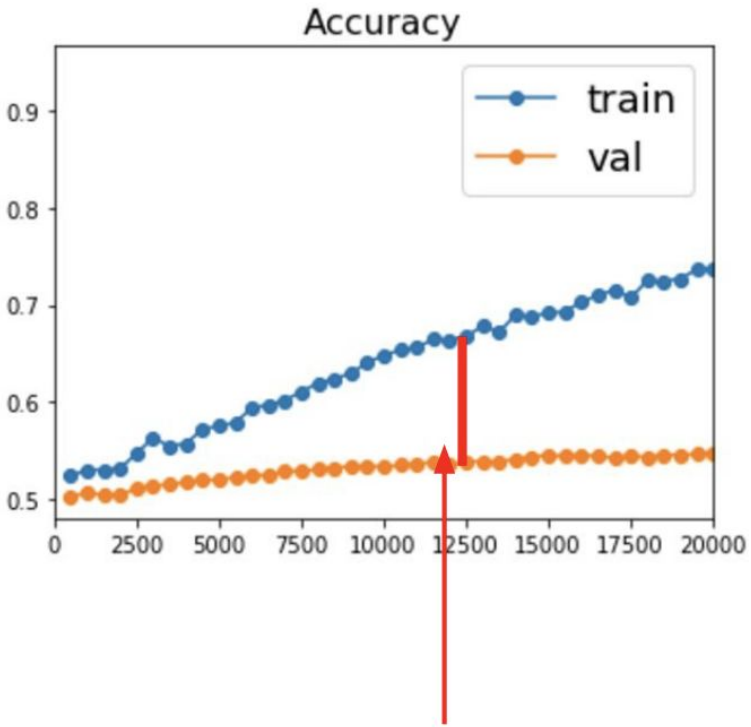


source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Sum up: optimization

- Adam is great basic choice
- Even for Adam/RMSProp learning rate matters
- Use learning rate decay
- Monitor your model quality





Better optimization algorithms help reduce training loss

But we really care about error on new data - how to reduce the gap?

source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Regularization

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

Adding some extra term to the loss function.

Common cases:

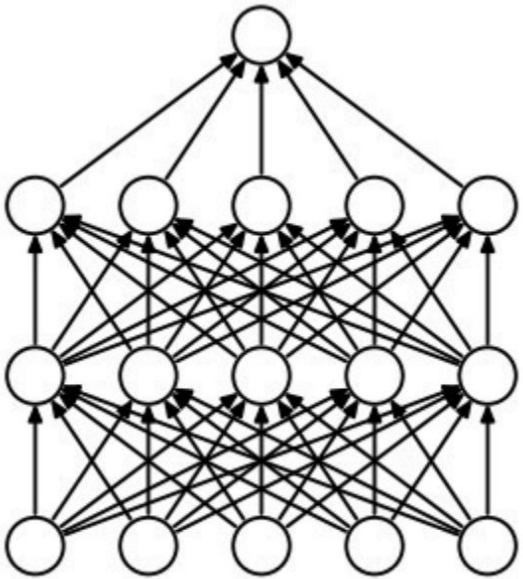
- L2 regularization: $R(W) = \|W\|_2^2$
- L1 regularization: $R(W) = \|W\|_1$
- Elastic Net (L1 + L2): $R(W) = \beta\|W\|_2^2 + \|W\|_1$

source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

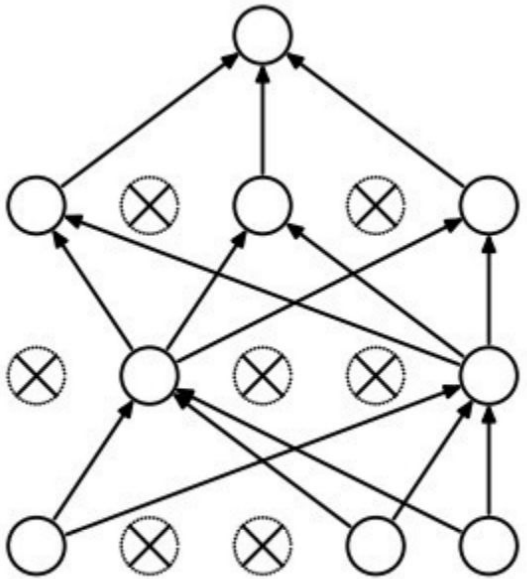
Regularization: Dropout

Some neurons are “dropped” during training.

Prevents overfitting.



(a) Standard Neural Net

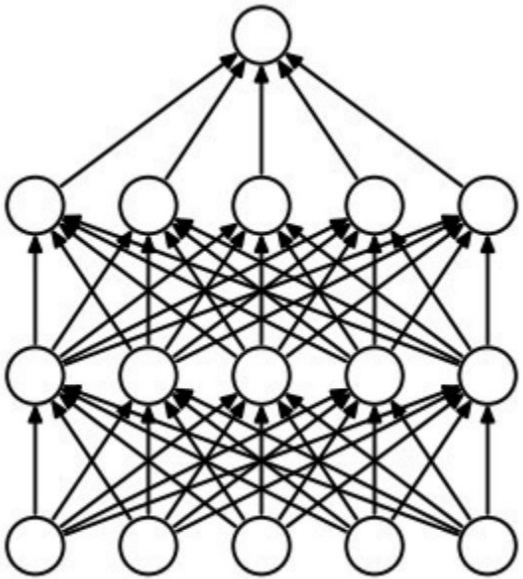


(b) After applying dropout.

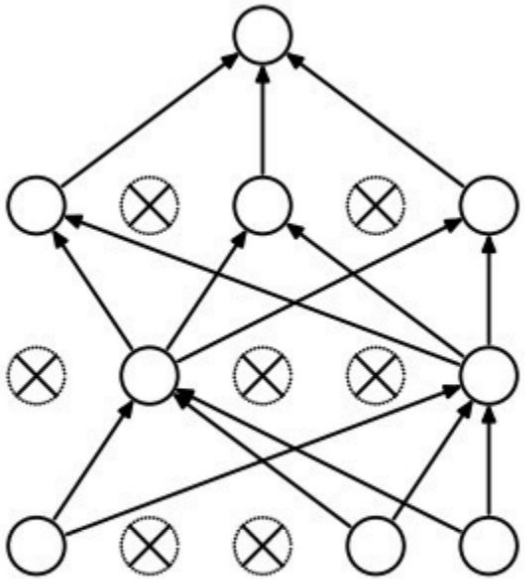
Regularization: Dropout

Some neurons are “dropped” during training.

Prevents overfitting.



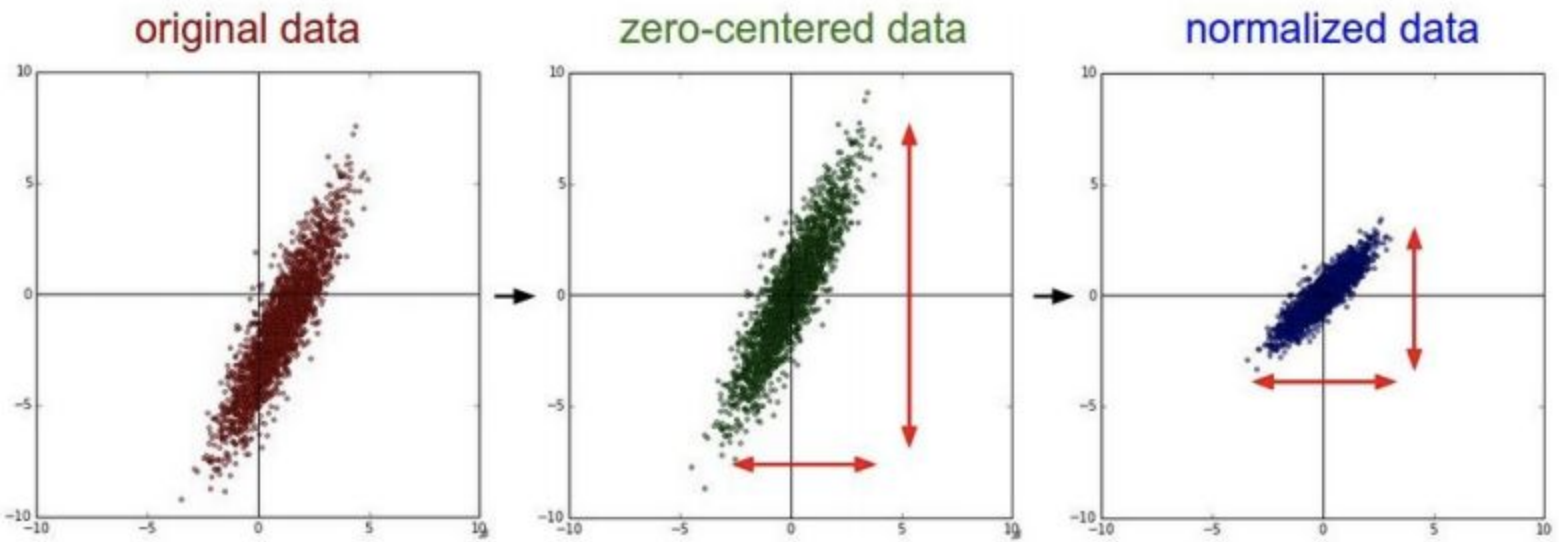
(a) Standard Neural Net



(b) After applying dropout.

Actually, on test case output should be normalized. See sources for more info.

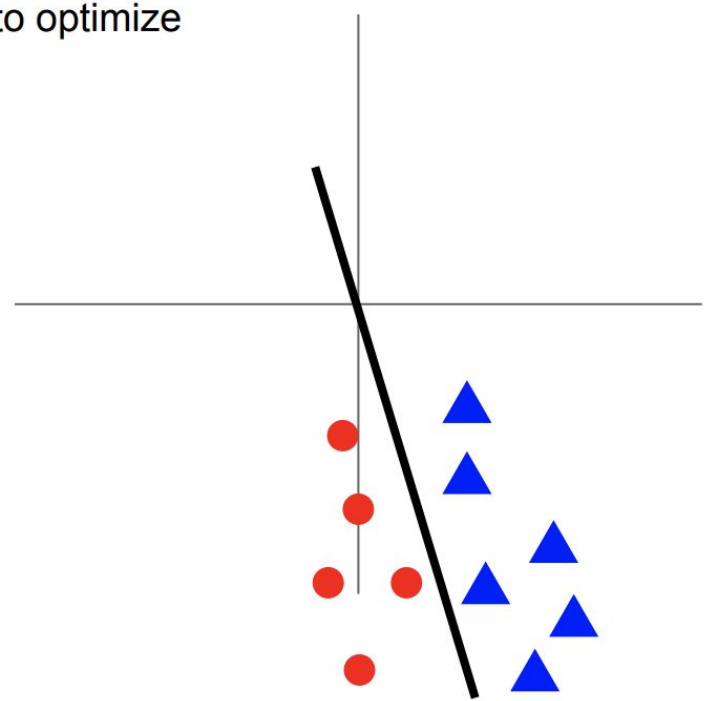
Data normalization



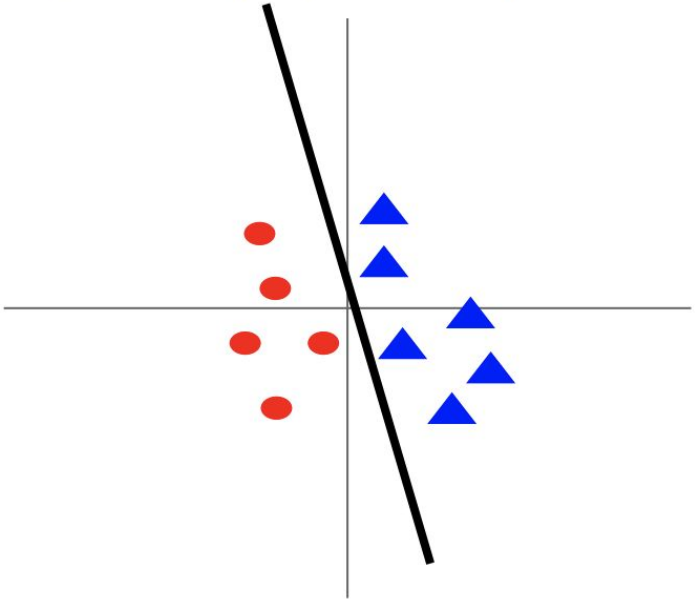
source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Data normalization

Before normalization: classification loss very sensitive to changes in weight matrix; hard to optimize



After normalization: less sensitive to small changes in weights; easier to optimize



source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Batch normalization

Problem:

- Consider a neuron in any layer beyond first
- At each iteration we tune it's weights towards better loss function
- But we also tune it's inputs. Some of them become larger, some – smaller
- Now the neuron needs to be re-tuned for it's new inputs

Batch normalization

TL; DR:

- It's usually a good idea to normalize linear model inputs

(c) Every machine learning lecturer, ever

Batch normalization

- Normalize activation of a hidden layer

  (zero mean unit variance)

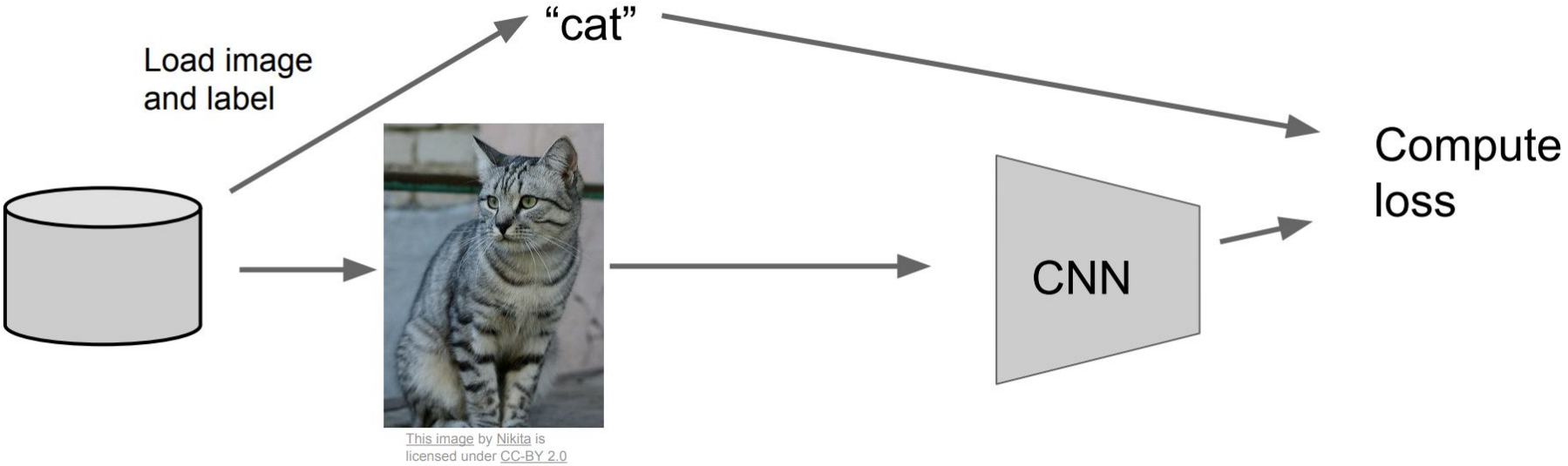
$$h_i = \frac{h_i - \mu_i}{\sqrt{\sigma_i^2}}$$

- Update μ_i, σ_i^2 with moving average while training

$$\mu_i := \alpha \cdot mean_{batch} + (1-\alpha) \cdot \mu_i$$

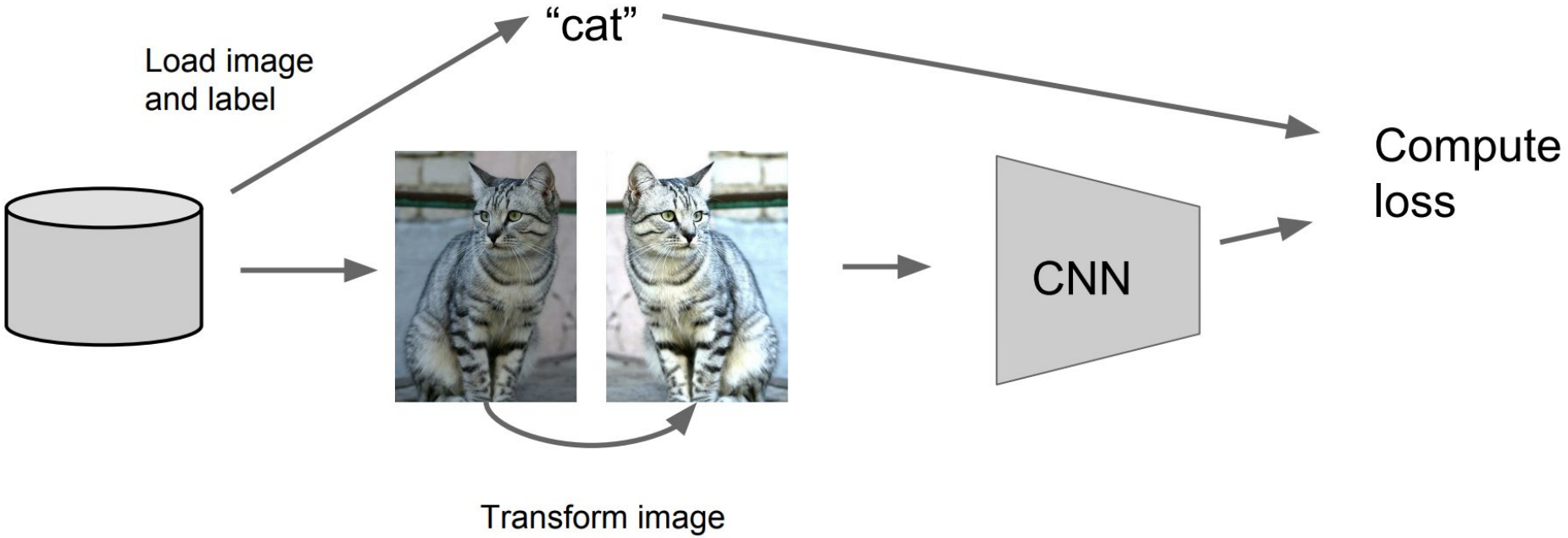
$$\sigma_i^2 := \alpha \cdot variance_{batch} + (1-\alpha) \cdot \sigma_i^2$$

Regularization: data augmentation



source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Regularization: data augmentation



source: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Sum up: regularization

Regularization:

- Add some weight constraints
- Add some random noise during train and marginalize it during test
- Add some prior information in appropriate form

Q & A

That’s all. Feel free to ask any questions.