Training Neural Networks

Outline

- Loss functions & Backpropagation
- Tricks of the trade:
  - Activation functions
  - Data preprocessing
  - Dropout
  - Batch normalization
  - Weight initialization
  - Hyperparameter optimization
  - Data augmentation

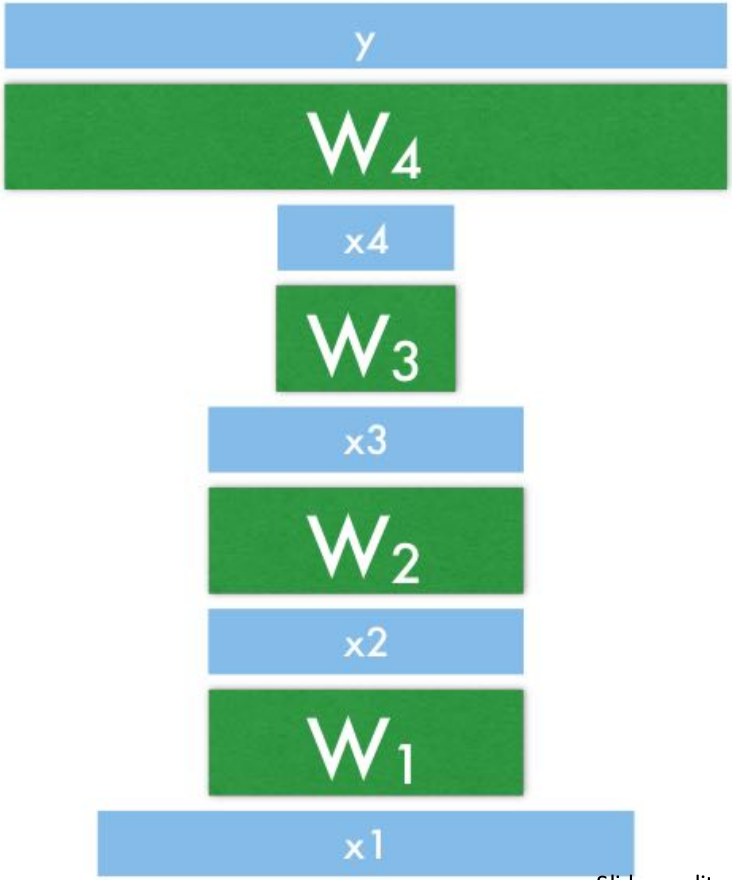
Multilayer Perceptron

- Layer representation

$$y_i = W_i x_i$$

$$x_{i+1} = \sigma(y_i)$$

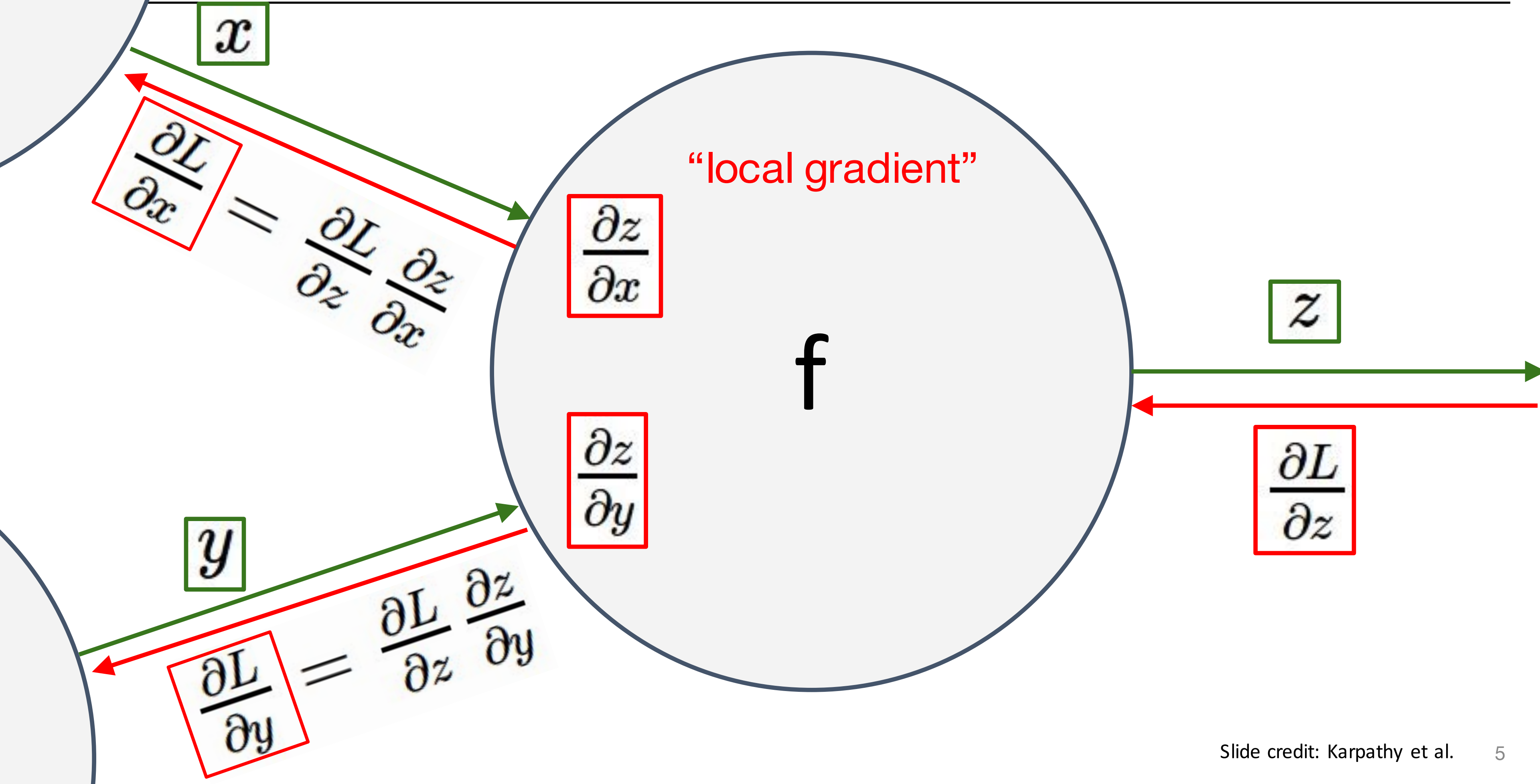
- Typically iterate between a linear mapping Wx and a nonlinear function
- Loss function L to measure the quality of the estimate so far



Slide credit: Alex Smola

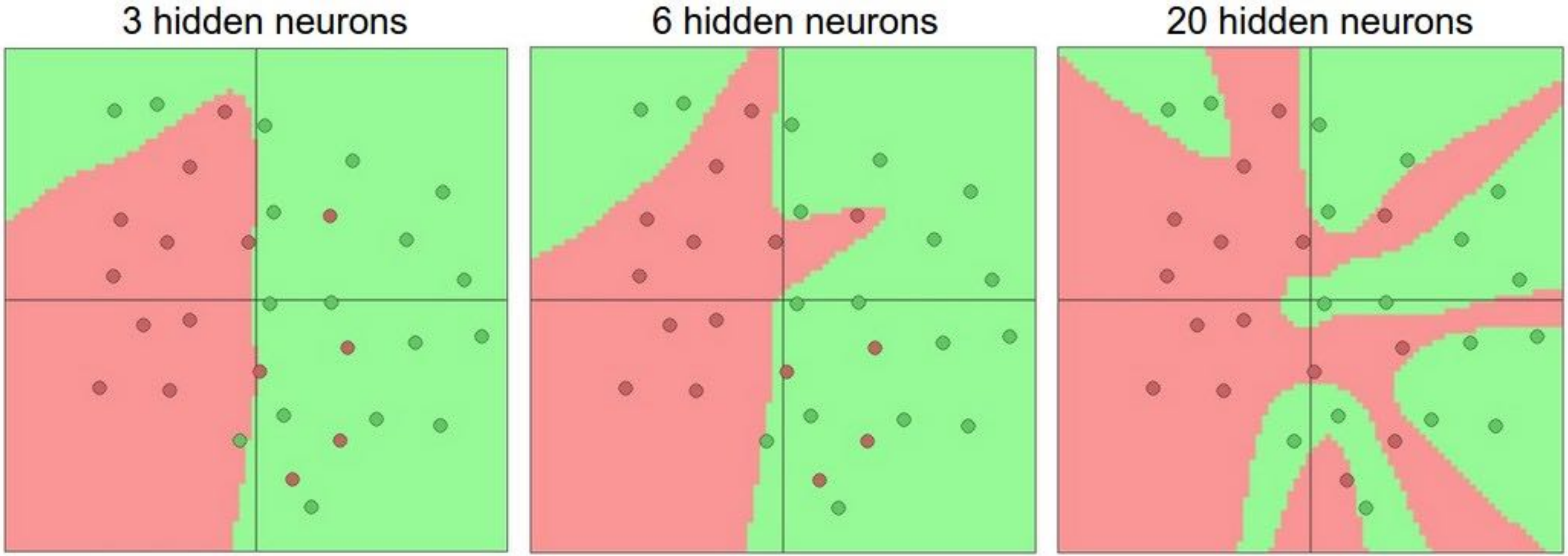
Loss function

- Loss = Data Loss + Regularization Loss
- Data loss measures the compatibility between a prediction and the ground truth label.
- Regularization loss penalizes the complexity of the model
- Ex: Regression data loss $L = \|f - y\|^2$
- L1 regularization loss: $\lambda|w|$
- L2 regularization loss: λw^2.
- Elastic net regularization: $\lambda_1|w|+\lambda_2 w^2$



Slide credit: Karpathy et al.

Setting the number of layers and their sizes

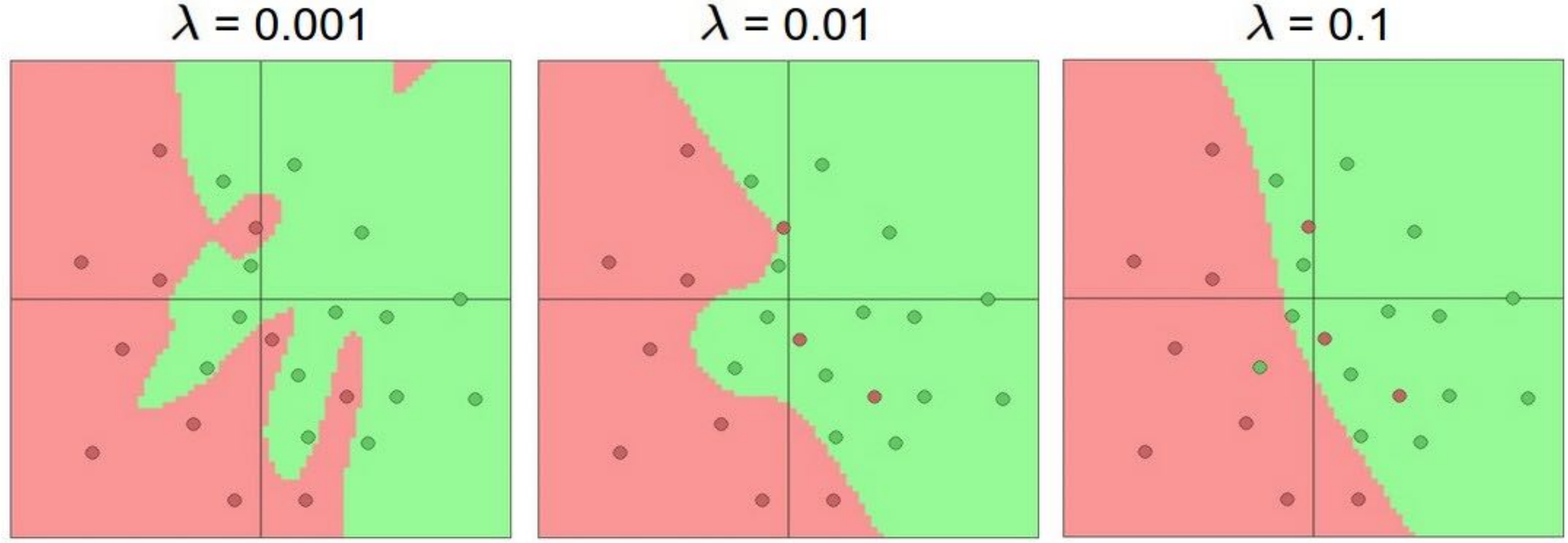


more neurons = more capacity

Slide credit: Karpathy et al.

Regularization: Penalizing large weights

- Do not use the size of the neural network as a regularizer.
- Use stronger regularization instead.



Slide credit: Karpathy et al.

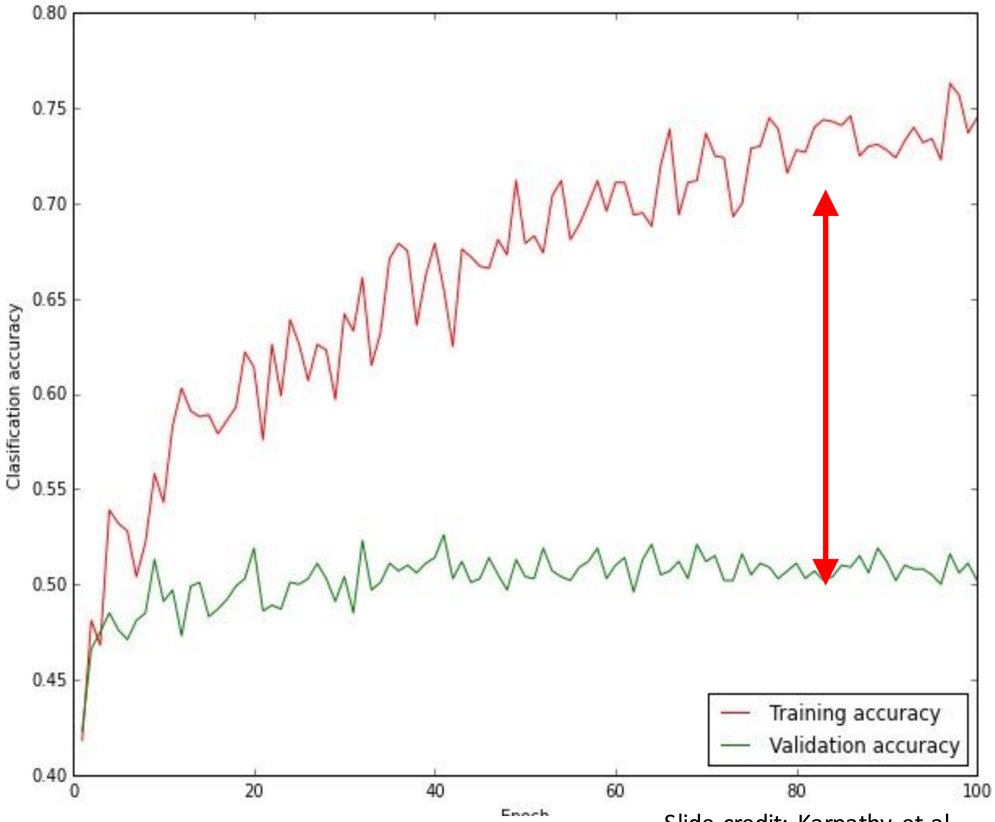
Monitoring the accuracy

big gap = overfitting

=> increase regularization strength?

no gap

=> increase model capacity?



Slide credit: Karpathy et al.

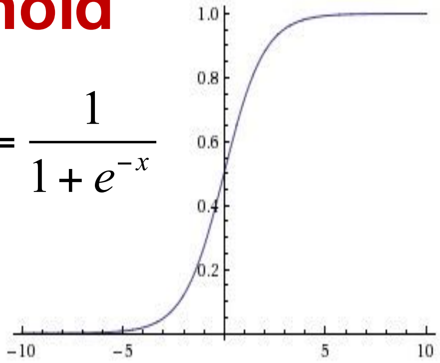
Outline

- Loss functions & Backpropagation
- Tricks of the trade:
  - Activation functions
  - Data preprocessing
  - Dropout
  - Weight initialization
  - Batch normalization
  - Hyperparameter optimization
  - Data augmentation

Activation Functions

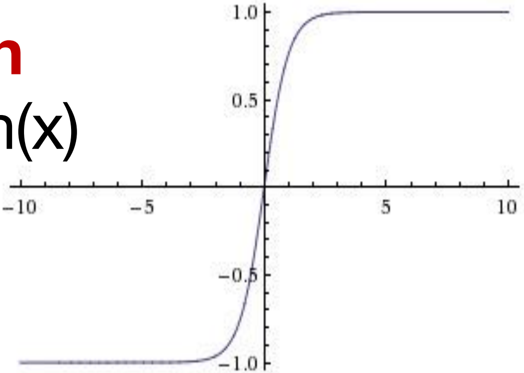
Sigmoid

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



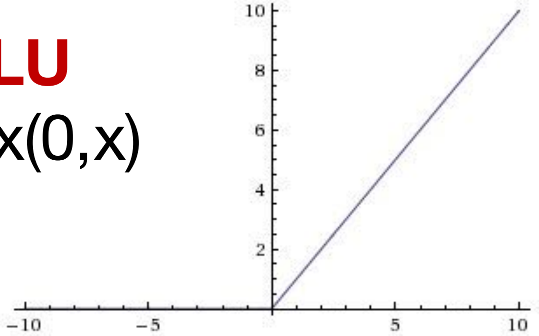
tanh

tanh(x)



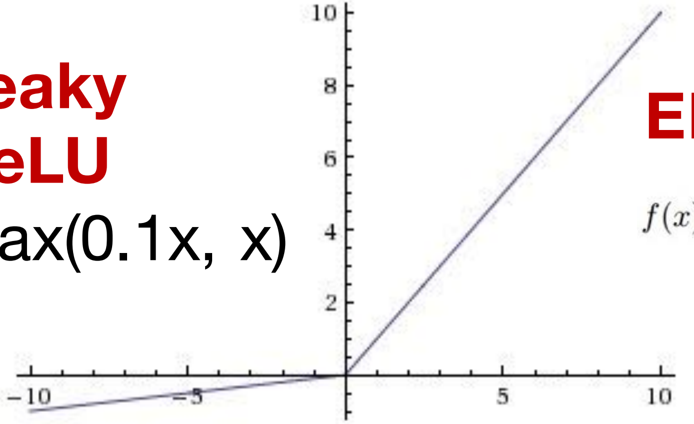
ReLU

max(0,x)



Leaky ReLU

max(0.1x, x)



Maxout

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

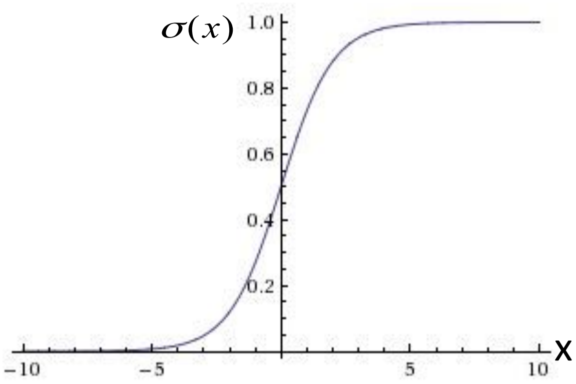
ELU

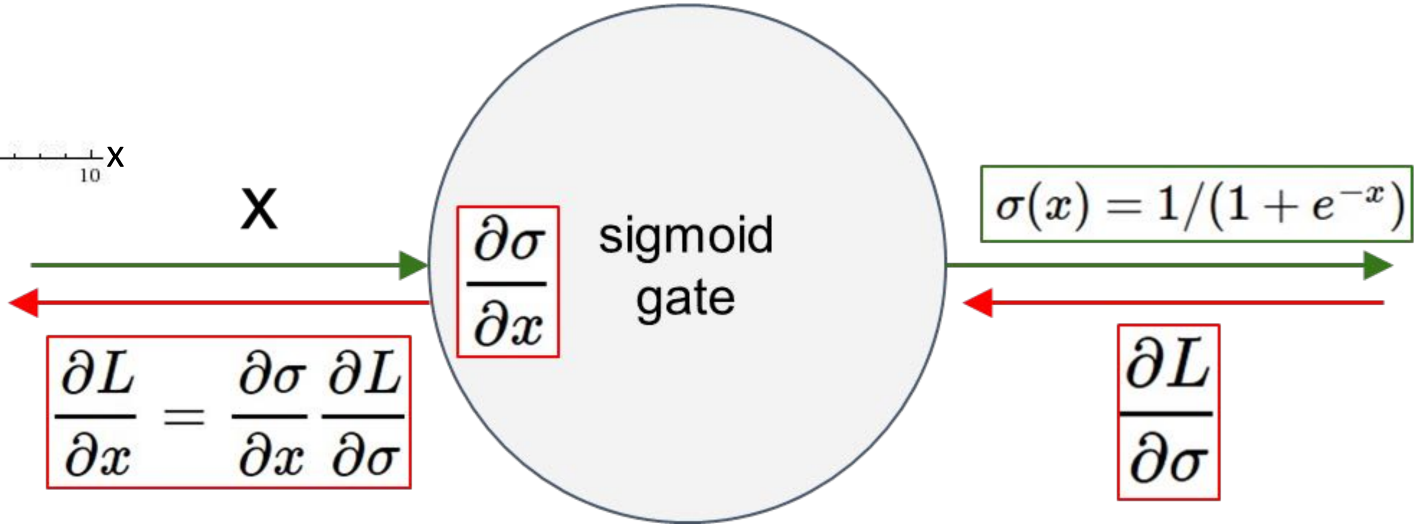
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$$

Slide credit: Karpathy et al.

Sigmoid

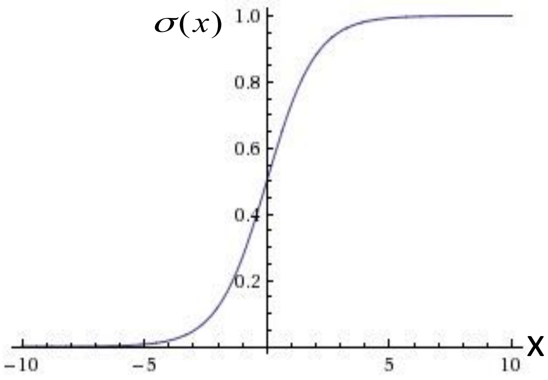
- Saturated neurons pull the gradients to zero
  - What happens when x = -10, x = 0, x=10 ?





Slide credit: Karpathy et al.

Sigmoid

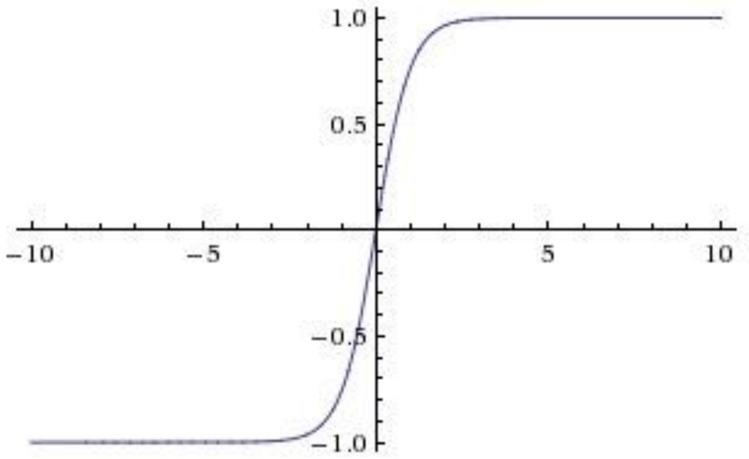


- Sigmoid outputs are not zero centered.
  - If the input to a neuron is always positive,
  - The gradients on **w** are always all positive or all negative.
  - This is why you want zero mean data!

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

tanh

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

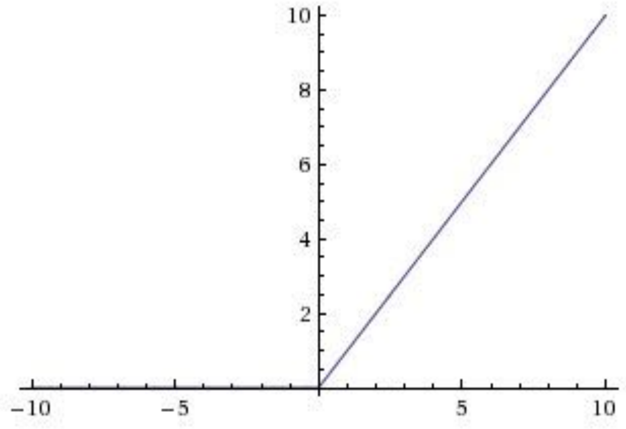


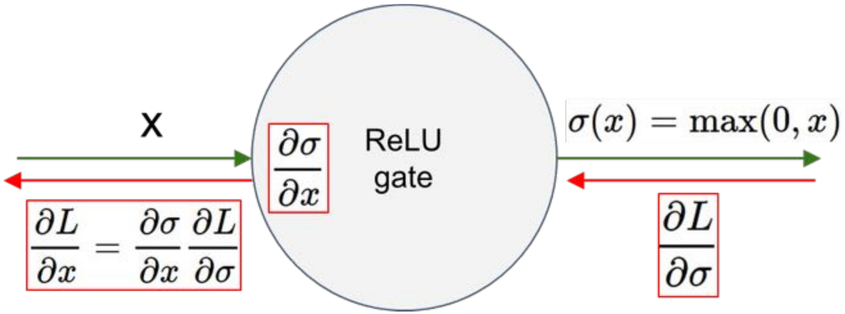
Slide credit: Karpathy et al.

[LeCun et al., 1991]

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)
- Not zero-centered output

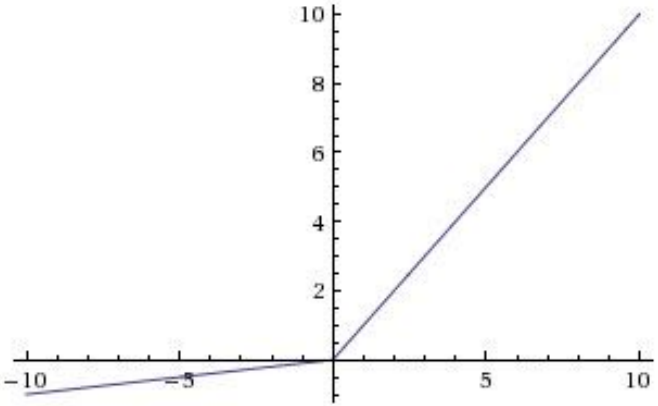




Slide credit: Karpathy et al.

[Krizhevsky et al., 2012]

Leaky ReLU



$$f(x) = \max(0.01x, x)$$

Does not saturate

TAKE- AWAY LESSONS:

- Use ReLU.
- Try out Leaky ReLU / Maxout / ELU / tanh
- Don't use sigmoid.
- Watch if gradients are dying. Be careful with your learning rates.
- Center your data.

Slide credit: Karpathy et al.

Tricks of the Trade: Data Preprocessing

- Center your images (zero mean)
- Subtract the mean image (e.g. AlexNet)
- Subtract per-channel mean (e.g. VGGnNEt)

An input image (256x256)

Minus sign

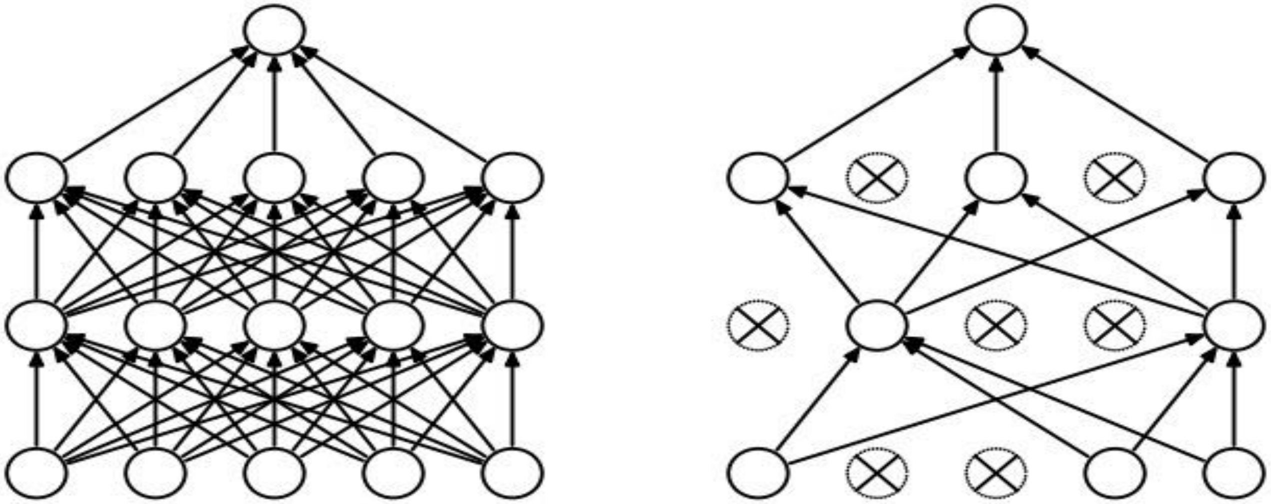
The mean input image

Tricks of the trade:

- Activation functions
- Data preprocessing
- Dropout
- Weight initialization
- Batch normalization
- Hyperparameter optimization
- Data augmentation

Regularization: Dropout

- Randomly set some neurons to zero in the forward pass
  - Multiply the output of the neuron by zero
  - So its gradient will be zero, so its weight will not get an update



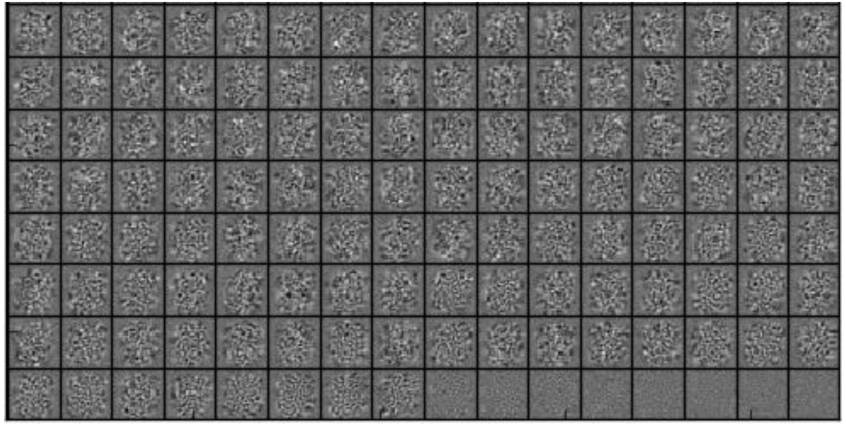
(a) Standard Neural Net (b) After applying dropout

[Srivastava et al., 2014]

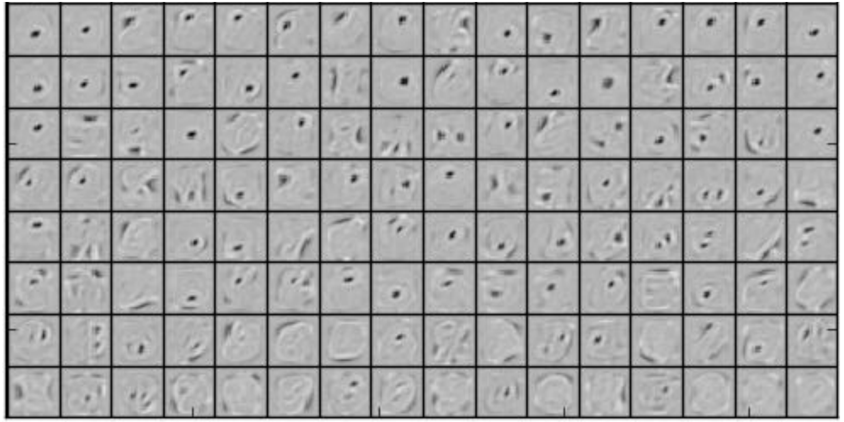
Slide credit: Karpathy et al.

Dropout

- Makes each and every hidden unit useful.
- At test time, use all the neurons, but scale the activations down by ½ (if you used 50% dropout).



(a) Without dropout



(b) Dropout with $p = 0.5$.

Slide credit: Karpathy et al.

Tricks of the trade:

- Activation functions
- Data preprocessing
- Dropout
- Weight initialization
- Batch normalization
- Hyperparameter optimization
- Data augmentation

Tricks of the trade: Weight initialization

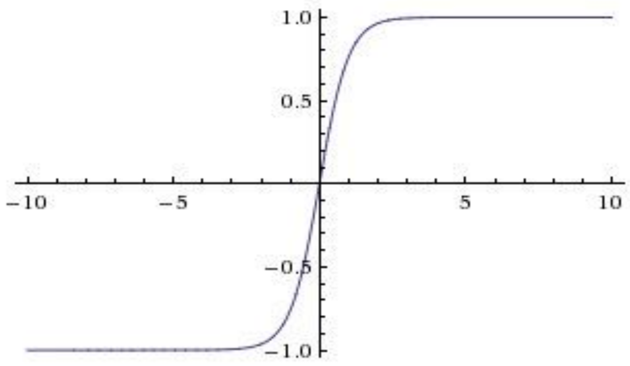
- If weights are initialized to very small numbers
- Assume tanh nonlinearity.
- What happens to the gradients for a W*X gate wrt. X?
- The gradients get multiplied through backpropagation
- All activations become zero!
- Called vanishing gradients

- Pretty important: Partly why NN's did not work for a long time!

Slide credit: Karpathy et al.

Weight initialization

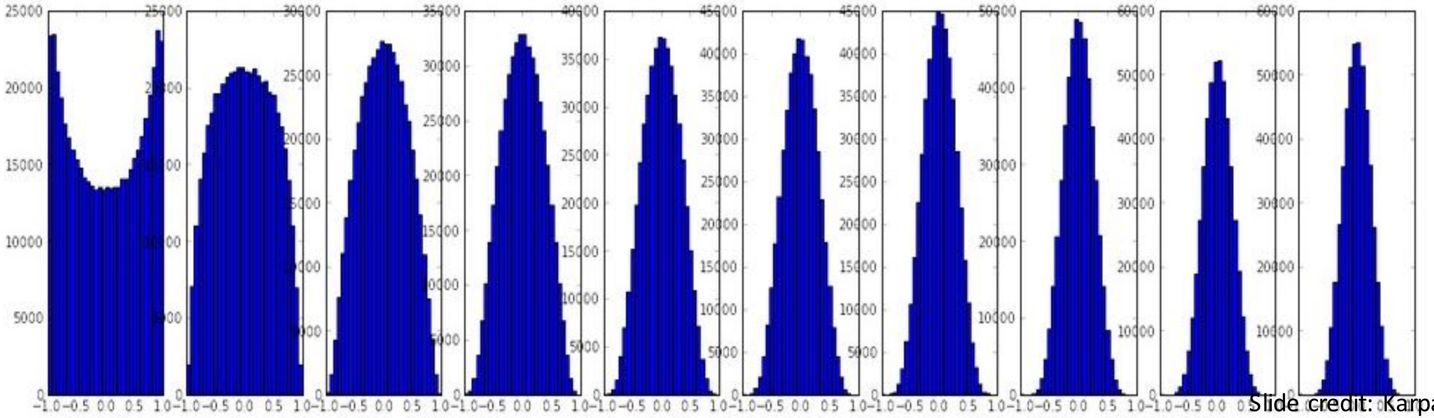
- If weights are initialized to large numbers W ~N(0, 1)
- Assume tahn nonlinearity.
- Almost all neurons completely saturate, either -1 and 1.
- Gradients will be all zero.



Slide credit: Karpathy et al.

Xaiver Weight Initialization

- W ~N(0, 1/ sqrt(fan_in)) (He et al., 2015)
- Lower weights if you have lots of inputs to a neuron.
- For a unit Gaussian input, you are in the active region of the tanh's.
- Distribution ends up being more sensible



Slide credit: Karpathy et al.

Tricks of the trade:

- Activation functions
- Data preprocessing
- Dropout
- Weight initialization
- Batch normalization
  - A technique that alleviates the problems of initialization
  - You want unit Gaussian activations!
  - Explicitly force the activations throughout a network to take on a unit Gaussian distribution at the beginning of the training.
- Hyperparameter optimization
- Data augmentation

Batch normalization

[Ioffe and Szegedy, 2015]

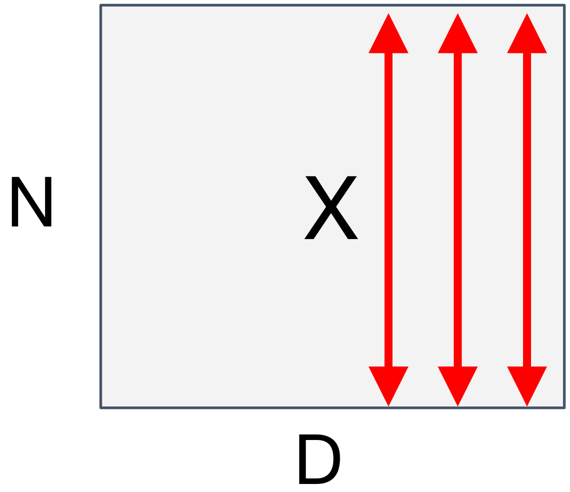
- Consider a batch of activations at some layer. To make each dimension unit gaussian, apply:
  - Compute the empirical mean and variance independently for each dimension.
  - Normalize

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

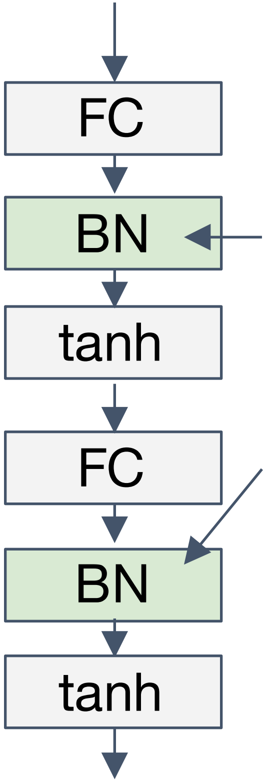
  - then allow the network to squash the range if it wants to:

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

parameters



Batch normalization



Have a BN layer after every convolution layer.

- Improves gradient flow through the network
- Your network learns faster!
- Reduces the strong dependence on initialization
- Acts as a form of regularization.

Slide credit: Karpathy et al.

Tricks of the trade:

- Activation functions
- Data preprocessing
- Weight initialization
- Batch normalization
- Hyperparameter optimization
  - Parameter updates, learning rate.
- Dropout
- Data augmentation

Parameter Updates

```
# Gradient descent update
x += - learning_rate * dx
```

```
# Momentum update
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```

```
# Adagrad update
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

```
# RMSProp
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

Slide credit: Karpathy et al.

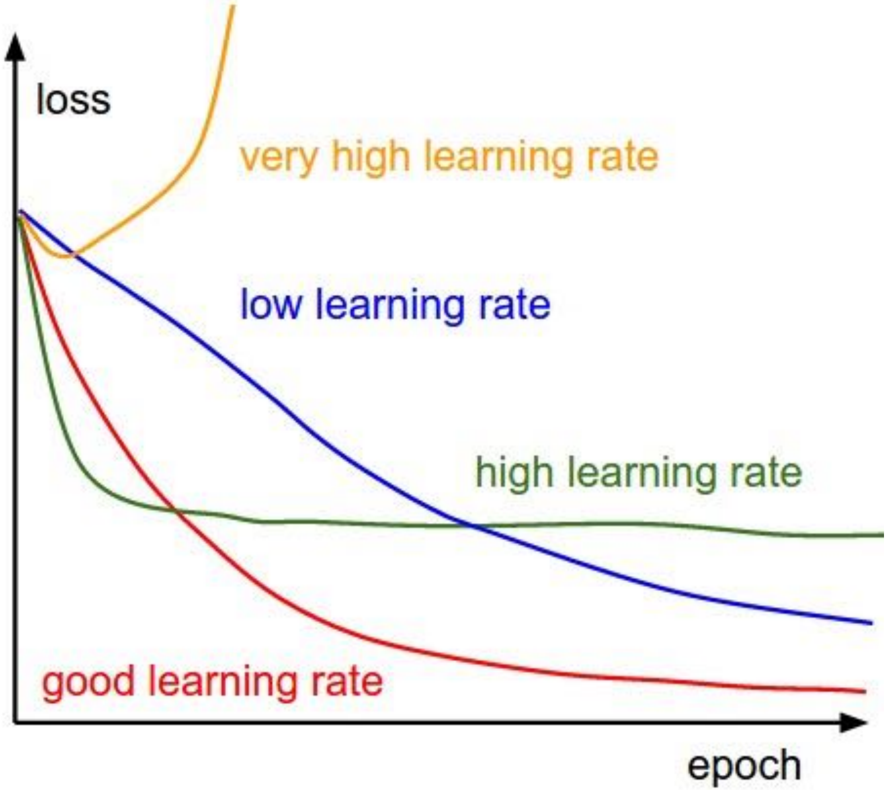
Parameter Updates: Adam update

```
# Adam
m,v = #... initialize caches to zeros
for t in xrange(1, big_number):
  dx = # ... evaluate gradient
  m = beta1*m + (1-beta1)*dx # update first moment
  v = beta2*v + (1-beta2)*(dx**2) # update second moment
  mb = m/(1-beta1**t) # correct bias
  vb = v/(1-beta2**t) # correct bias
  x += - learning_rate * mb / (np.sqrt(vb) + 1e-7)
```

Slide credit: Karpathy et al.

Monitoring the loss

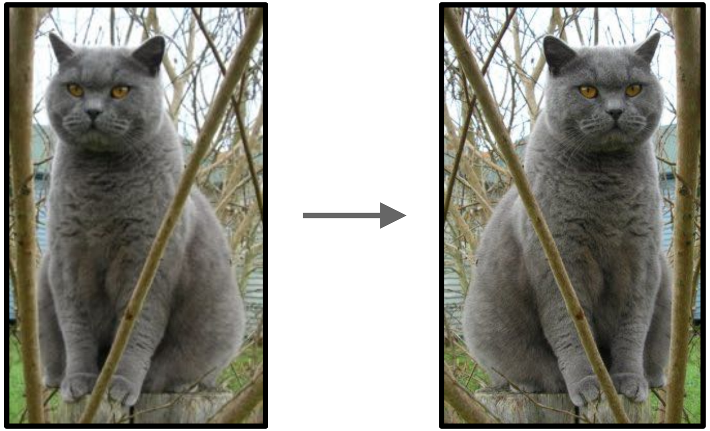
- GD, Momentum, AdaGrad, Adam etc. all have learning rates.
- Start with high learning rates
- Decay the learning rate by half every few epochs.



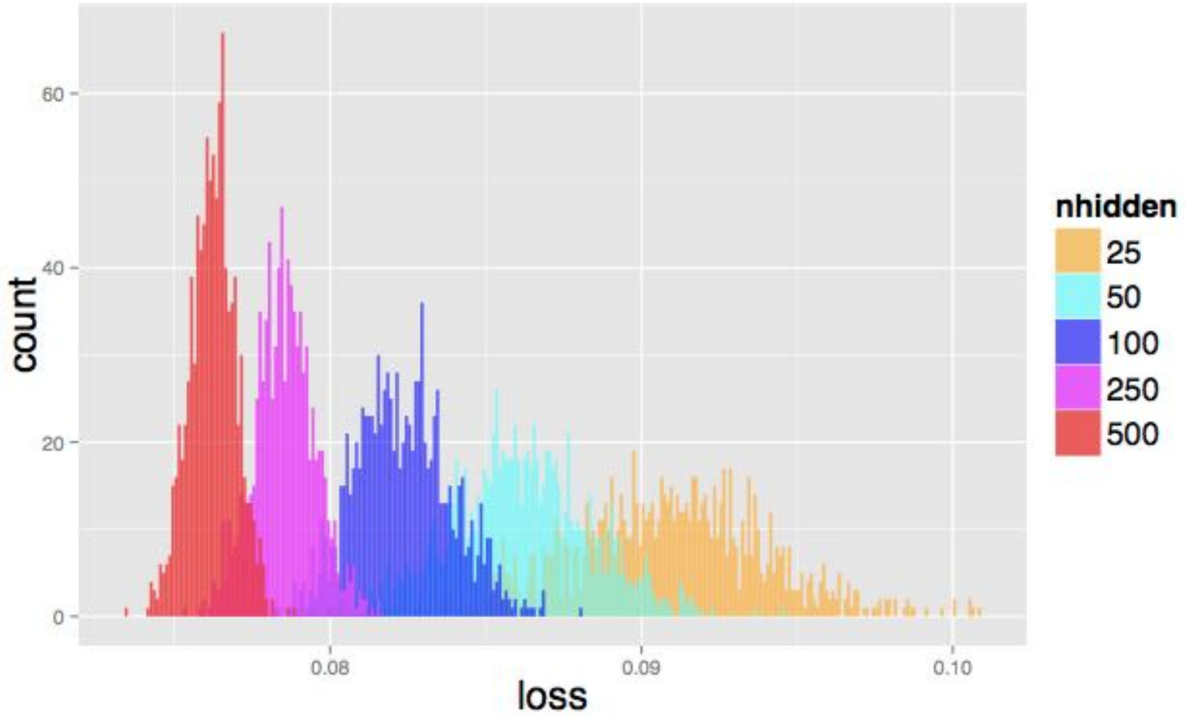
Slide credit: Karpathy et al.

Data Augmentation

- Providing a CNN with extra training examples can reduce overfitting.
- Perturb the existing training samples to create new ones.
  - Geometric transformation (translation, rotation, stretching, shearing)
  - Cropping
  - Contrast/brightness adjustment
  - Lens distortions
  - Horizontal flips



Local Minimum



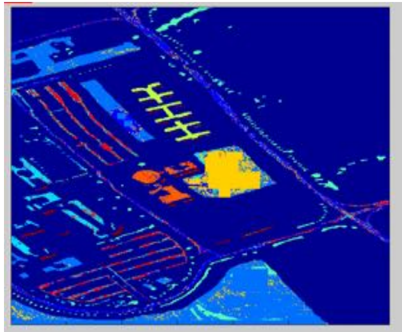
The Loss Surfaces of Multilayer Networks: A. Choromanska, M. Henaff, M. Mathieu G. B. Arous, Y. LeCun. In AISTATS 2015

Now that we have covered all the tricks

- Loss & Backpropagation
- Tricks of the trade:
  - Activation functions
  - Data normalization
  - Weight initialization
  - Dropout
  - Batch normalization
  - Hyperparameter optimization
  - Data augmentation

Deep Learning Research at

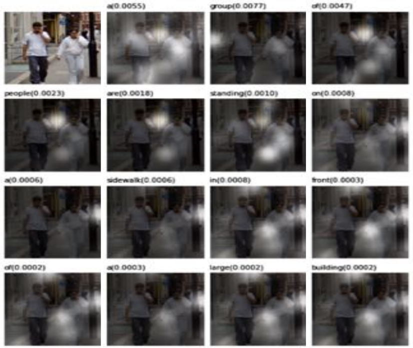
HACETTEPE UNIVERSITY COMPUTER VISION LAB & PARRSLAB



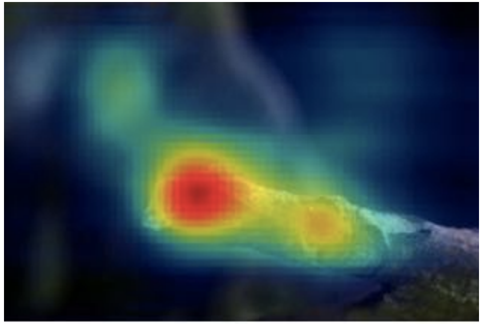
Hyperspectral Classification
Monday – Computer Vis. I

Visual Summarization
Tuesday – Special Ses. 7

Image Captioning in Turkish
Wednesday – Computer Vis. V



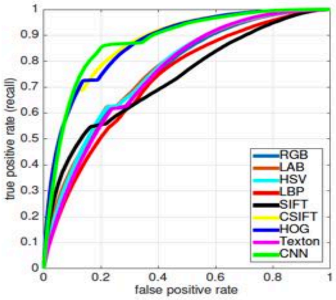
Attention-based
Image Captioning



Dynamic Saliency Prediction

Video Anomaly Detection

Scene Recognition



(a) a-Pascal

Visual Attribute Recognition

Human Interaction Recognition

Crowd Counting

Zero-shot Classification

If you would like to hear more on deep learning

- Please join us after a 10-min break
- And also join our talks in SIU

If you are interested, check out PARRSLAB and HUCVL at Hacettepe University! We are always looking for collaborators, motivated graduate as well as undergraduate students.

- parrslab.ee.hacettepe.edu.tr
- vision.cs.hacettepe.edu.tr