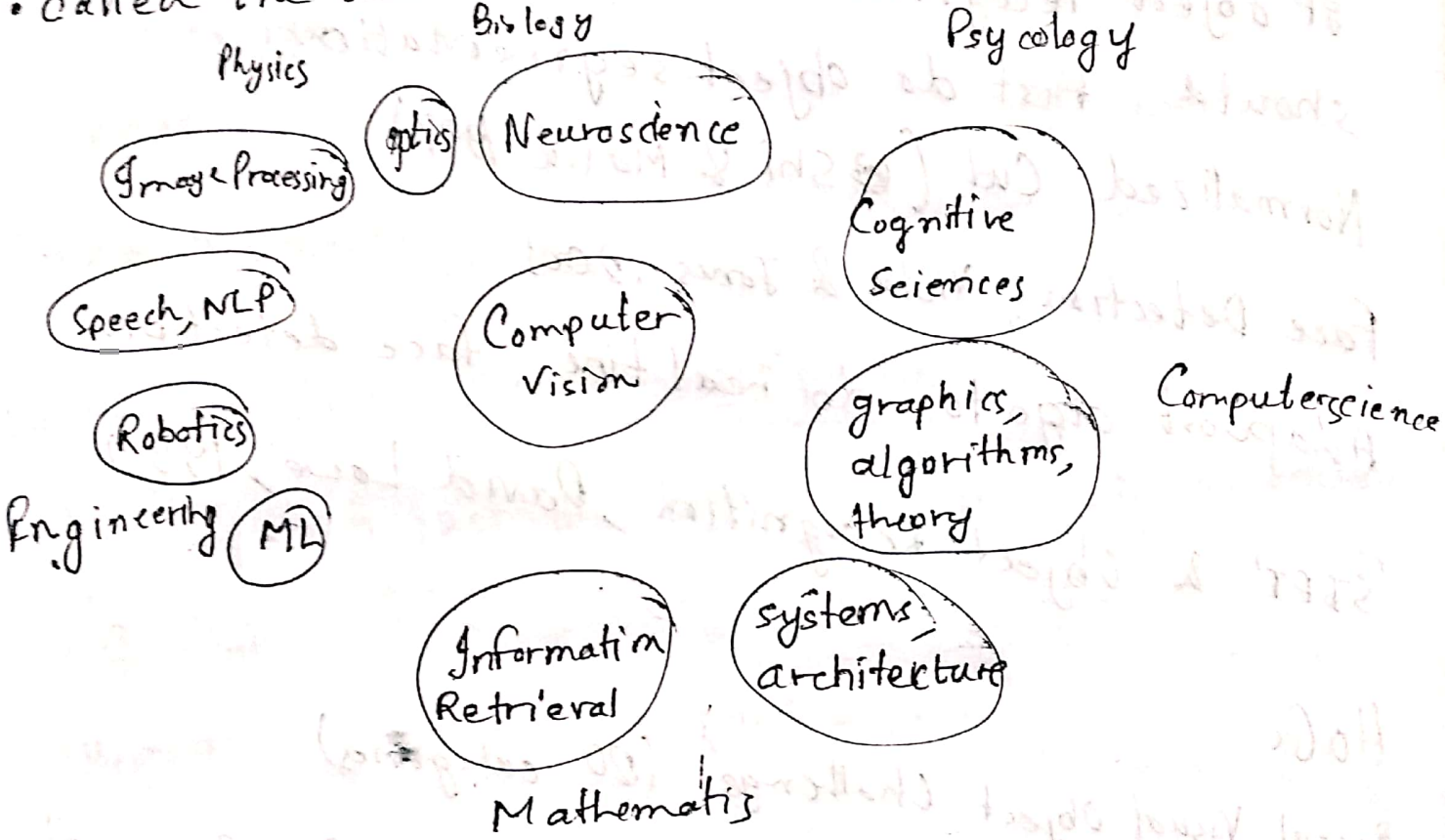
CS231n: Convolutional Nets for Visual Recognition

Fei-Fei Li; Justin Johnson; Serena Yueng

Computer vision is the study of visual data

- called the dark matter of internet



Hubel & Wiesel, 1959

Electrical signal from brain ⟷ mapping ⟷ stimulus

Block world [Larry Roberts, 1963]

1966 → Summer Vision Project

Generalized Cylinder
1979

Pictorial Structure
1973

Nonm.
If object recognition is too hard, may be we should first do object segmentation
Normalized Cut (Shi & Malik 1997)
Face Detection, Viola & Jones, 2001
Adaboost algo to do real time face detection
'SIFT' & Object Recognition, David Lowe, 1990

HoG
Pascal Visual Object Challenge (20 categories)
Imagenet (Deng, Dong, Socher, Li, & Fei-Fei, 2009)
Stanford & Princeton
(22K categories & 14M images)
Took 3 years (took help from Amazon Mechanical Turk)
Imagenet Classification Challenge

There is a number of visual recognition problems that are related to image classification, such as object detection, image captioning

Lecanet. al (conv-net)

Alexnet

Image → Captions

Lecture 2: Image Classification Pipeline

- An image is just a big grid of numbers between [0, 255]

Attempts: finding edges (not scalable)

Data Driven approach:

1) Collect data
2) Use ML to train
3) Evaluate the classifier

Distance Metric

L1 Distance ⇒ $d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$

L2 dist ⇒ $d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$

Nearest Neighbor $O(1)$

Memorize data

for each testimage
    find closest train image $O(N)$
    Predict label of nearest image

This is bad: we want classifiers that are fast at prediction; slow for training is ok

k-nearest neighbor (to get rid of noisy answer)

Hyperparameters: choices about the algo that we set rather than learn

Setting hyper params

K-NN on images never used.

- very slow at test time
- distance metric not informative (not good to check similarity)
- curse of dimensionality

Linear Classification

Parametric Approach

$f(x, W) \longrightarrow$ 10 numbers giving class scores

W ↑ Weights

$$f(x, W) = Wx + b$$

$f(x, W)$: 10×1 | W: 10×3072 | x: 3072×1 | b: 10×1

→ Linear classifier is learning only one template

Lecture 3 - Loss function & Optimization

Challenges of recognition

i) Viewpoint
ii) Illumination
iii) Deformation
iv) Occlusion
v) Clutter
vi) Intraclass variation

- Define a loss function that quantifies our unhappiness with the scores across the training
- Come up with a way of efficiently finding parameters that minimize the loss function

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Multiclass SVM loss

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

s_{y_i} : true value

$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \qquad s = f(x_i, W)$$

Then, $L = \frac{1}{N} \sum_{i=1}^{N} L_i$

What happen to loss if a score that's high is changed a bit?

→ Nothing happens because it still returns zero loss.

What is the min/max possible loss for SVM

→ min → 0
max → ∞

Q3| At initialization W is small so all $s \approx 0$.

What is the loss?

(number of classes - 1)

Q4| What if sum as ~~ad~~ over all classes?

The loss increases by 1

This is just for convention we omit the correct class so that our minimum loss is zero.

Q5| What if we used mean instead of sum?

→ answer would be same because we don't care about true scores

Q5| What if $\sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)^2$

→ This would end up ~~giving~~ to a different loss function that's not linear

Q6| If we find a W that gives $L=0$. Is this W unique?

No, there are many other Ws. Like $2W$

Regularization: Model should be simple so that it works on test data

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

Occam's Razor:

Among competing hypotheses the simplest is the best

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

Max norm regularization

Dropout

Batchnorm

Stochastic depth

If you're Bayesian: L2 regularization also corresponds to MAP inference using a Gaussian prior on W

Softmax Classifier (Multinomial Logistic Regression)

$$P(Y=k/X=x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{where} \quad s = f(x_i; W)$$

$$L_i = -\log P(Y=y_i | X=x_i) \qquad L_j = -\log\left(\frac{e^{s_{yi}}}{\sum_j e^{s_j}}\right)$$

Want to maximize log likelihood, or to minimize the negative log likelihood of the correct class

Q: What is the min and max loss?

min = 0
max = infinity

Q2: Usually at initialization W is small so all $s \approx 0$ what is the loss?

$$= -\log\left(\frac{1}{C}\right)$$

Softmax vs SVM

Q: Suppose I take a datapoint and jiggle a bit (changing its score slightly). What happens to the loss in both cases?

SVM doesn't care about the score
Softmax continuously try to improve the datapoints, like pushing the correct to inf and pushing the incorrect to -inf

$$\text{Softmax}, L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

$$\text{SVM}, L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$\text{Full Loss } L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$

Optimization:

Strategy #1 Random search

Strategy #2 follow the slope

In 1-dimension, the derivative of a function

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$ [slope]

In multiple dimensions, the gradient is the vector of partial derivatives along each direction is slope

- The slope in any direction is the dot product of the direction with the gradient
- The direction of steepest descent is the negative gradient.

→ This is super slow and super bad.

We can ~~use~~ compute anlytic gradient

Numberical gradient : approximate, slow, easy to write
Analytic gradient : exact, fast, error-prone

Gradient check: Using analytic gradient to find grads but checking with numerical gradient is
- This is an intersting debugging tool.

Gradient descent

i) find grads
ii) weights -= stepsize * grads
↑
hyperparamer
learning rate

Stochastic Gradient Descent:

Full sum is expensive when it's large
Aprrox sum using minibatch of examples
32/64/128 common

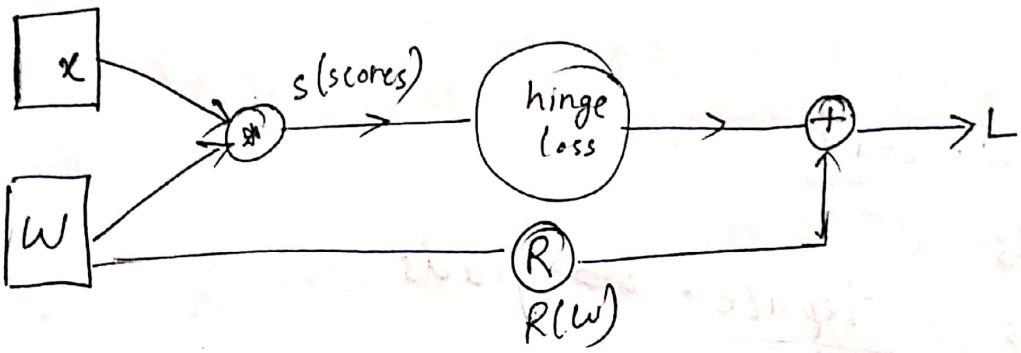
Image features:

Color Histogram
Histogram of Oriented Gradients
Bag of Words (Build codebook, Encode Images)

Image features vs ConvNets:

ConvNets (Krizhevsky 2012) AlexNet

Lecture 4: Intro to Neural Nets:



Leverage chain rule

$\frac{d}{dx}(e^x) = e^x$

$\frac{d}{dx}(ax) = a$

$\frac{d}{dx}\left(\frac{1}{x}\right) = -\frac{1}{x^2}$

~~$f(x) = c + x = a$~~

$\frac{d}{dx}(c+x) = 1$

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

$\frac{d\,\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2}$

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

$= [1-\sigma(x)]\,\sigma(x)$

If any problem with gradients, break down to computational graph

add gate: gradient distributor

Q: What is a max gate?

the highest one back is gonna get the max value. other will be zeroed out

mul gate: gradient switcher

local gradient is the value of the other variable

Neural Nets:

2-layer Network $f = W_2 \max(0, W_1 x)$

Biological Neurons are far more complicated

Activations

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

$$\nabla_W f = 2q \cdot x^T$$

$$\nabla_W f = 2W^T q$$

Lecture 5: Convolution Neural Networks

- Try to maintain spatial structure

History of NN and CNN

→ Mark I perceptron machine was the first implementation of the perceptron algorithm.

* no backprop $f(x) = \begin{cases} 1 & w \cdot x + b \geq 0 \\ 0 & \text{otherwise} \end{cases}$

Frank Rosenblatt (1957)

→ Widrow and Hoff (1960): Adaline/Madaline

Multilayer perceptron ⇔ no backprop

Rumelhar (1986) → Backprop

[Hinton and Salakhutdinov 2006] →

Reinvigorated research in Deep Learning

First strong results [2012] [Krizhevsky, 2012]

- Imagenet classification with deep CNN

AlexNet

Neurocognition [Fukushima, 1980]

[Lecun, Botto, Bengio, Haffner, 1998] →
Gradient-based learning applied to document recognition

[K

ConvNets for classification, retrieval, detection, segmentation, self driving car, pose estimation, gameplay
Image Captioning
Neural Style Transfer

NVIDIA Tesla line [GPU] ~~powers~~

Convolutional Neural Networks:

Convolutional layer → preserve spatial structure

→ We call the layer convolutional because it is related to convolution of two signals

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

↑ elementwise multiplication and sum of a filter and the signal (image)

Conv Layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparams
  - $K \rightarrow$ # of filters
  - $F \rightarrow$ # their spatial extent
  - $S \rightarrow$ stride
  - $P \rightarrow$ Amound of zero padding

*Produces a volume of size $W_2 \times H_2 \times D_2$ where

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$
- $D_2 = K$

→ with parameter sharing it introduces $F \cdot P \cdot D_1$ weights per filter for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases

* The output volume, the d-the depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S_2, and then offset by d-th bias.

Pooling layer

→ makes the representations smaller and more manageable

→ operates over each activation map independently

Pooling layer:

Accepts a volume of size $W_1 \times H_1 \times D_1$

- Requires three hyperparams
  - → their spatial extent f
  - → th strides
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - → $W_2 = (W_1 - f)/S + 1$
  - → $H_2 = (H_1 - f)/S + 1$
  - → $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for pooling layers

Fully connected Layer (FC-Layer)

→ Contains neurons that connect to the entire input volume, as in ordinary Neural Networks

Typical architectures

$[(Conv\text{-}ReLU)*N\text{-}POOL?]*M\text{-}(FC\text{-}ReLU)*K$, Softmax

- ResNet/GoogleNet challenge this paradigm

Lecture 6 : Training NNs I

Minibatch SGD

Loop:

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

P1

Activation Function

Data Preprocessing

Weight Initialization

Batch Normalization

Babysitting the Learning Process

Hyperparameter Optimization

Activation functions:

* Sigmoid => Squashes number to range [0,1]

→ historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

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

Problems:

1) Saturated neurons "kill" the gradients

2) Sigmoid outputs are not zero centered

3) exp() is a bit ~~me~~ computationally expensive

* tanh(x) • squashed numbers to range [-1, 1]
• zero-centered
• still kills gradients when saturated
[Krizhevsky et. al. 2012]

* ReLU (Rectified Linear Unit) $f(x) = \max(0, x)$
→ Does not saturate (in + region)
→ very computationally efficient
→ converges much faster than sigmoid/tanh in practice [6x]
→ actually more biologically plausible than sigmoid

Cons → not zero centered output
An annoyance:
what is the gradient when $x < 0$?

* Leaky ReLU → to ~~activate dead~~ activate dying ReLU
~~Does not s~~
• will not die $f(x) = \max(0.01x, x)$

Parametric ReLU $f(x) = \max(\alpha x, x)$

Exponential Linear Unit (ELU) [Clevert et. al. 2015]

- 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()

Maxout "Neuron" [Goodfellow et. al. 2013]

- Doesn't have the basic form of dot product -> nonlinearity
- Generalizes ReLU and Leaky ReLU
- Linear Regime! Doesn't saturate! Doesn't die!

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

Problem: doubles the number of params/neurons

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

Data Preprocessing

zero center and normalization so that all of them can contribute

PCA and whitened data (for high dimensional ones)

In practice for Images (center only)

→ Not common to normalize variance, to do PCA or whitening

Weight Initialization

What happens when W=0 init is used?

→ all the neuron start learning the same thing

First idea: small random numbers

(gaussian with zero mean and 1e-2 std deviation)

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

→ Works ~okay for small nets but problems with deeper networks

All activation become zero for bigger nets.

What do gradients look like?

things get smaller and smaller and later it's the same

Almost all neurons completely saturary either -1 and 1 and grads will be zero

Xavier initialization: [Glorot et. al. 2010]

Reasonable → mathematical derivation assumes linear activations

- but when using the ReLU non-linearity it breaks

He et. al. 2015 (note initialization/2)

- Proper initialization is an active area of research

Batch Normalization: [Ioffe and Szegedy, 2015]

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

- Consider a batch of activations at some layer.

Usually inserted after Fully connected or conv layers and before non-linearity

- do we necessorily want to a unit guassian inpnt to an activation layer

- $y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$ # scale and shift

network can squash the range if it wants to (learnable)

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

Note: at test time BatchNorm layer are not computed based on batch. Instead, a single fixed empirical mean of activations during training is used.

Babysitting Training Process

Check loss

Overfit on small portion

- take the first 20 examples from CIFAR-10
- turn off regularization
- use simple vanilla 'sgd'

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

→ loss not going down → learning rate too low

→ Learning rate increase → loss exploding
too high
Loss reaches NaN

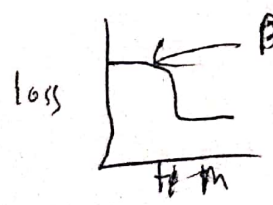
Hyperparam Optimization

Cross-validation strategy

coarse → fine

run coarse search for some epochs

Random search, Grid Search
[Bergsta 2012]



Lecture 7: Training NNs II

Fancier optimization
Regularization
Transfer learning

Optimization:

Vanilla gradient descent

Problems with SGD

What if loss changes quickly in one dimension and slowly in another? what does gradient do?

→ zigzag

What if the loss function has a local minima or saddle point?

Zero gradient, gradient descent gets stuck

- saddle point much common in high dimension

- Our gradients come from minibatches so they can be noisy!

- Full batch gradient descent doesn't solve these problems.

SGD + momentum

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

- Build up velocity as a running mean of gradients
- Rho gives friction: typically rho = 0.9 or 0.99

This solves all the problem is SGD

Nesterov momentum:

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

$$x_{t+1} = x_t + v_{t+1}$$

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

AdaGrad (don't choose in NN)

Added element-wise scaling of the gradient based on the historical sum of squares in each dimension.

step size gets smaller and smaller

RMSProp

step size is managed ~~to stay as~~ shrink slowly

Adam (almost)

Momentum + Adagrad/RMSProp

what happen at the first step

Full form

Momentum

- Bias correction → for the fact that first and second momentum estimates start at ~~zero~~ zero

AdaGrad / RMSProp

Default

beta1 = 0.9

beta2 = 0.999

~~learn~~ lr = 1e-3 or 5e-4

Which one of those learning rates is best to use?

step decay → decaying by half after each epoch

exponential decay: $\alpha = \alpha_0 e^{-kt}$

1/t decay => $\alpha = \alpha_0/(1+kt)$

~~Lear~~ decay is more common with SGD momentum less common with Adam

First order optimization

(1) use gradient form linear approx

(2) ~~Ha~~ step to minimize approx

Second order approximation:

(1) Use gradient and Hessian to for quadratic approx

(2) Step to the minima of the approx

$$J(\theta) = J(\theta_0) + (\theta - \theta_0)^T \nabla_\theta J(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T H (\theta - \theta_0)$$

$$\theta^* = \theta_0 - H^{-1} \nabla_\theta J(\theta_0)$$ [No learning rate]

Practically back for deep learning [impractical]

Hessian has $O(N^2)$ and inverting takes $O(N^3)$

N is large

In practice

→ Adam is a good default

→ L-BFGS can be tried out if we can afford full batch updates (can be good for style Transfer)

Beyond training error

Model Ensembles

- Train multiple independent models
- At train time average their results

Enjoy 2% extra performance

- Instead to training independent models, use multiple snapshots of a single model during training

→ keep a moving average of the params vector and use that at test time (Polyak averaging)

Improve single model performance

Regularization

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

~~$R(W)$~~

L2 Reg

L1

Elastic Net (L1+L2)

Dropout

Randomly set some neurons to zero. 0.5 is common

Forces the nework to have a redundant representation;

Prevents co-adeptim of features

Dropout: Test time

Dropout makes our output random

Want to 'average out' the randomness at test time

$$y = f(x) = E_z[f(x, z)] = \int p(z) f(x, z) dz$$

$E[a] = w_1 x + w_2 y$

During training we have $E[a] = \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(x w_1 x 0_y) + \frac{1}{4}(0 x_1 0_y) + \frac{1}{4}(0 x_1 w_2 y)$

$= \frac{1}{2}(w_1 x + w_2 y)$

At test time, multiply by dropout prob

Data Augmentation

~~Horiz~~ Flip
crop
occlusion
contrast and brightness

translation
rotation
stretching
shearing
lens distortions
(go crazy)

Drop Connect

we zero out the value of weight matrixes

almost like dropout

Fractional Max Pooling

Stocastic Depth

Randomly drop network

Transfer Learning

Reduce overfitting
Reduce need for huge data

Small dataset

freeze the taken ones and train.

Later try to train on the whole network

for different dataset

we need to try out ~~training~~ fine-tuning more layers