Lecture 1 : Intro

Have to combine disparate information sources.

Alpha Go Zero: 100-0 vs Alpha Go. Learned entirely from self play.
- Always has an opponent at just the right level

Algorithms matter way more than data/compute... principled approach

- Label smoothing – Don't treat data from Humans as 100% accurate

History of NN's

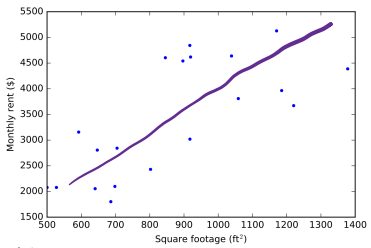
- Rosenblatt : Perceptron, fires >0, else doesn't fire
  - 1st a linear classifier
  - Multilayer perceptrons achieved non-linear behavior
    - Needed back prop to train

- Fully connected , Convolutional Neural Network
  - Why revolution post 2010? Datasets and compute
  - Image Net
    - 14 million Images
    - Bounding Boxes - localization
  - Alex Net - 1st to get <25% error on Image Net, kicked off CNN revolution

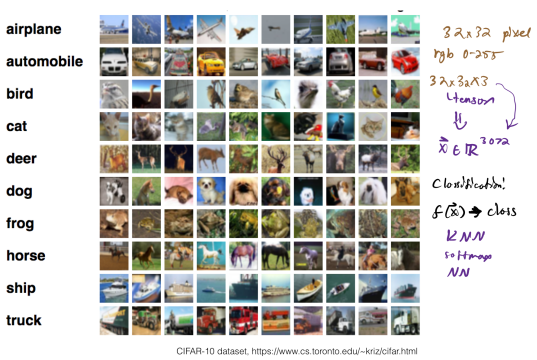
Basics of ML : LinearReg

Read Ch. 5 - 5.5 2.3

- Focus on supervised classification problems
- CIFAR-10 dataset used a lot
- Linear reg example



input : sq ft
output : rent



model? reasonably linear $y = ax + b$

parameters : choose a,b for best fit

- Min $J(a,b)$

- Could have done a polynomial : $y = b + a_{100}x^{100} + a_{99}x^{99} \cdots + a_1 x$

- Highly non-linear and overfit
- Degree of Freedom - # of parameters
  - Choose using cross validation, test set
- NN is a Universal Function Approx: Can emulate any function that exists
- Why sq' error?: Derivative calc is easier

Lec 2: Linear Reg 1/15

Linear Reg Example

Model: $y = ax+b = \theta^T x \quad \theta = \begin{bmatrix} a \\ b \end{bmatrix} \quad \hat{x} = \begin{bmatrix} x \\ 1 \end{bmatrix}$

Cost: $\mathcal{L}(\theta) = \frac{1}{2}\sum_{i=1}^{N} (y^{(i)} - \hat{y}^{(i)})^2 \quad \hat{y} = \theta^T x$

↑ scaling mag

Solve by setting derivatives/gradient $\frac{d\mathcal{L}}{d\theta} = 0$

Multivariate Derivatives / Gradients:

$x \in \mathbb{R}^n, \; y \in \mathbb{R}$

1) $\nabla_x y = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \vdots \\ \frac{\partial y}{\partial x_n} \end{bmatrix}$

- each number tells you Δx_i gives you Δy
- number in linear reg. function in general

Δy due to x_2 is $\{\frac{\partial y}{\partial x_2}\}\Delta x_2$ ($(\nabla_x y)_2$)

2) $f(x) = x^T A x, \quad \nabla_x f(x)?$ $\quad x \in \mathbb{R}^n \quad A \in \mathbb{R}^{n \times n}$

$A = \begin{bmatrix} A_{11} & \cdots \\ \vdots & A_{nn} \end{bmatrix} \quad f(x) = \sum_i \sum_j a_{ij} x_i x_j$

$i=j=1$: $\frac{\partial a_{11} x_1^2}{\partial x_1} = 2a_{11}x_1$, $i=1, j \neq 1$: $\sum_{j=2}^{n} a_{1j} x_1 x_j = \sum_{j=2}^{n} a_{1j} x_j$

$j=1, i=1$: $\sum_{i=2}^{n} a_{j1} x_i x_j$

$\Rightarrow \frac{\partial f(x)}{\partial x_i} = 2a_{ii}x_i + \sum_{j=1}^{n} a_{ij} x_j + \sum_{i=1}^{n} a_{i1} x_i$

$= \sum_{j=1}^{n} a_{ij} x_j + \sum_{i=1}^{n} a_{i1} x_i$

$(Ax)_i \quad (A^T x)$

$\nabla_x f(x) = (Ax + A^T x) = (A + A^T)x$

If A is symmetric ($A = A^T$) $= 2Ax$

Denominator Layout $\mathbb{R}^{denom \times num}$ dim = denom, num^T

Numerator Layout $\mathbb{R}^{num \times denom}$ dims = num, denom^T

Least Sqs Solution:

$$\theta = (X^TX)^{-1}X^Ty$$

$$\frac{\partial \mathcal{L}}{\partial \theta} = \tfrac{1}{2}\left[0 - 2X^Ty + (X^TX + X^TX)\theta\right]$$
$$= -X^Ty + X^TX\theta = 0$$
$$\theta = (X^TX)^{-1}X^Ty$$

X - features vector, y is values

For higher order:

$$\vec{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \theta = \begin{bmatrix} b \\ a_1 \\ \vdots \\ a_n \end{bmatrix}$$

More data - more complex models

Lec 3 — k fold / MLE — 1/10

θ - parameters

Hyper - choices prior to training

The standard for training, evaluating, and choosing models is to use different datasets for each step. (get θ)

- **Training data** is data that is used to learn the parameters of your model.
- **Validation data** is data that is used to optimize the hyperparameters of your model. This avoids the potential of overfitting to nuances in the testing dataset. (fit hyperparams)
- **Testing data** is data that is used to score your model. - unseen until testing

k-fold cross validation

k - folds in train/val not test

train on order n = [1 2 3]

val with $\mathcal{L}_1$ - $\mathcal{L}_5$

Max Likelihood Est. (MLE)

$$\mathcal{L} = \prod_{i=1}^{n} p(x^i, y^i \mid \theta)$$

$$\log \mathcal{L} = \sum \log p(x, y \mid \theta) = \sum \log [p(y)\, p(x \mid y)]$$

$$= N \log \tfrac{1}{3} + \sum \log \mathcal{N}(\mu_{y_i}, \Sigma_{y_i})$$

equiprobable — normal dist

$$\log p(\mathbf{x}^{(i)} \mid y^{(i)} = j) = \log \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma_j|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}^{(i)} - \mu_j)^T \Sigma_j^{-1} (\mathbf{x}^{(i)} - \mu_j)\right)$$
$$= -\frac{n}{2}\log 2\pi - \frac{1}{2}\log|\Sigma_j| - \frac{1}{2}(\mathbf{x}^{(i)} - \mu_j)^T \Sigma_j^{-1}(\mathbf{x}^{(i)} - \mu_j)$$
$$= k_2 - \frac{1}{2}\log|\Sigma_j| - \frac{1}{2}(\mathbf{x}^{(i)} - \mu_j)^T \Sigma_j^{-1}(\mathbf{x}^{(i)} - \mu_j)$$

Hence, we arrive at:

$$\log \mathcal{L} = k + \sum_{i=1}^{N} -\frac{1}{2}\log|\Sigma_{y^{(i)}}| - \frac{1}{2}(\mathbf{x}^{(i)} - \mu_{y^{(i)}})^T \Sigma_{y^{(i)}}^{-1}(\mathbf{x}^{(i)} - \mu_{y^{(i)}})$$

Lec 4 — image class / k-nearest — 1/10

- Plurality of k nearest (Euclidean Dist)
  1. Chose metric: Euclidean $\|x^i - x^j\|$
  2. # neighbors k
  3. $d(x^{new}, x^i)\ \forall i$
  4. $\{C_1 \dots C_k\}$ smallest distances, x^{new} is most-common class

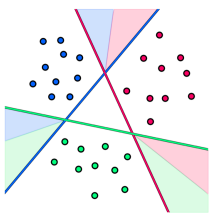
Pros: simple Cons: slow class, need every point in memory

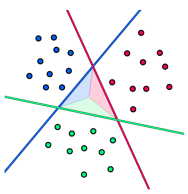
Distances are for less intuitive in higher dim space

What is linear class doing?

Further distance away on positive side is better!

max $w_i^T x$ if $w_j \leftarrow \frac{w_j}{\|w_j\|_2}$





https://jermwatt.github.io/machine_learning_refined/notes/7_Linear_multiclass_classification/7_2_OvA.html

Soft max $a_i = w_i^T x$

$$\frac{e^{a_i(x)}}{\sum_{j=1}^{c} e^{a_j(x)}}$$

$$\Rightarrow \Pr(y^i = i \mid x^j, \theta) = \text{softmax}_i(x^i)$$

Prob that x^i is in class j.

Lec 5 — -log softmax, Gradient Descent — 1/19

Softmax Classifier

$$\arg\max_\theta \prod_{i=1}^{m} p(x^i \mid \theta)\, p(y^i \mid x^i, \theta) = \arg\max_\theta \prod_{i=1}^{m} p(y^i \mid x^i, \theta)$$

equiprobable priors

Softmax log Likelihood:

$$\arg\min_\theta \sum_{i=1}^{m} \left(\log \sum_{j=1}^{c} e^{a_j(\mathbf{x})} - a_{y^{(i)}}(\mathbf{x}^{(i)}) \right)$$

$$\arg\max_\theta \sum_{i=1}^{m} \log p(y^i \mid x^i, \theta) = \arg\max_\theta \sum_{i=1}^{m} \log \text{softmax}_{y^i}(x^i)$$

$$= \arg\max_\theta \sum_{i=1}^{m} \log \left[\frac{e^{a_{y^i}(x^i)}}{\sum_{j=1}^{c} e^{a_j(x^i)}} \right]$$

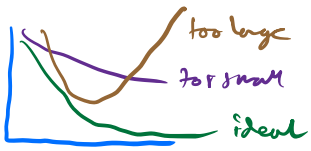
$$= \arg\max_\theta \sum_{i=1}^{m} \left[a_{y^i}(x^i) - \log \sum_j e^{a_j(x^i)} \right] \frac{1}{m}$$

make a loss

$$= \arg\min_\theta \frac{1}{m} \sum_{i=1}^{m} \left[\log \sum_j e^{a_j(x^i)} - a_{y^i}(x^i) \right]$$

Gradient Descent:

- 1st order approx $\nabla_x f \cdot \Delta x \approx f(x+\Delta x)$
  Linearization
- Step size:



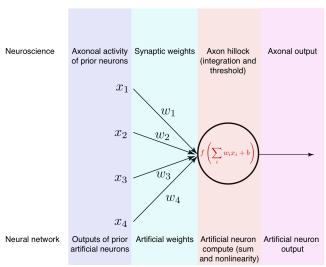
Remember, step size is using that approximation

- Batch vs Mini (Stoch) - • Batch is all at once (all in examples)
  • Mini batch GD - Use a small subset 12k<n
  • Stochastic — one example and step

Lec 6 : NN's : AE, Activation, etc 1/24

Neural Networks:

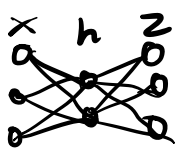
- Bio neuron
  - axon, axon hillock etc:



# of layers: hidden + output → hidden + 1

If dim(h) << dim(x): Low rank representation of data

Auto encoder: Lower dim representation of x



$\mathcal{L} = \|z - x\|^2$

PCA is linear, auto encoder doesn't have to be

Activation Functions (Pros + Cons):

Sigmoid:

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

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

Pros:
- Around $x = 0$, the unit behaves linearly.
- It is differentiable everywhere.

Cons:
- At extremes, the unit *saturates* and thus has zero gradient. This results in no learning with gradient descent.
- The sigmoid unit is not zero-centered; rather its outputs are centered at 0.5.

$x \xrightarrow{w_1} h_1, \xrightarrow{w_2} h_2 \xrightarrow{w} z$

Consider $f(\mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x} + b)$. Defining $z = \mathbf{w}^T\mathbf{x} + b$, the derivative with respect to w, the parameters, is:

$$\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} = \sigma(z)(1-\sigma(z))\mathbf{x}$$ $h_1 = \frac{\partial z}{\partial w_1}$ $\frac{\partial f}{\partial z}$

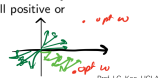
$f = \sigma(z)$

If $\mathbf{x} \geq 0$ (e.g., if the input units all had a sigmoidal output), then the gradient has all positive entries. Let's say we had some gradient, $\frac{\partial L}{\partial f}$, which can be positive or negative. Then $\frac{\partial L}{\partial \mathbf{w}}$ will have all positive or negative entries.

$\frac{\partial \mathcal{L}}{\partial f} \frac{\partial f}{\partial z} \frac{\partial z}{\partial w_1}$ $z \geq 0$

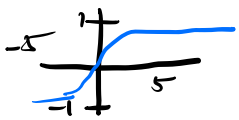
opt w

This can result in zig-zagging during gradient descent.



Hyperbolic Tangent:

$\tanh(x) = 2\sigma(x) - 1$



$\tanh'(x) = 1 - \tanh^2(x)$

Pros:

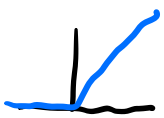
- Around $x = 0$, the unit behaves linearly.
- It is differentiable everywhere.
- It is zero-centered.

Cons:

- Like the sigmoid unit, when a unit saturates, i.e., its values grow larger or smaller, the unit saturates and no additional learning occurs.

ReLU

$\text{relu}(x) = \max(0, x)$



$\text{relu}'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$

Pros:

- In practice, learning with the ReLU unit converges faster than sigmoid and tanh.
- When a unit is active, it behaves as a linear unit.
- The derivative at all points, except $x = 0$, is 0 or 1. When $x > 0$, the gradients are large, and not scaled by second order effects.
- There is no saturation if $x > 0$.

Cons:

- $\text{ReLU}(x)$, like sigmoid, is not zero-centered.
- $\text{ReLU}(x)$ is not differentiable at $x = 0$. However, in practice, this is not a large issue. A heuristic when evaluating $\frac{d\text{ReLU}(x)}{dx}\Big|_{x=0}$ is to return the left derivative (0) or the right derivative (1); this is reasonable given digital computation is subject to numerical error.
- Learning does not happen for examples that have zero activation. This can be fixed by e.g., using a leaky ReLU or maxout unit.

ReLU variants:

Softplus: $\zeta(x) = \log(1 + \exp(x))$

Leaky ReLU / PReLU: $\max(\alpha x, x)$

Exp Relu: $\max(\alpha(\exp(x) - 1), x)$

Output activation / Loss

Use $\sigma(z)$

$MSE = \frac{1}{2}\sum_i^n [y^i - \sigma(z^i)]^2$

Cross Entropy (CE) $= -\sum_i^n y^i \log \sigma(z^i) + (1 - y^i) \log(1 - \sigma(z^i))$

$\frac{\partial MSE}{\partial z} =$

$\frac{\partial CE}{\partial z} =$

Lec 7: Bach Prop, Gates, NN Backprop layer 11/26

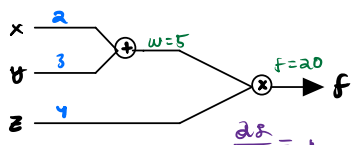
Backprop: Chain Rule, local gradient with upstream derivative

If you know local gradients, you can backprop anything

$f(x, y, z) = (x + y)z \quad w = x + y \quad f = wz$

Due to denominator - use reverse $\frac{\partial f}{\partial x} = \frac{\partial w}{\partial x}\frac{\partial f}{\partial w}\frac{\partial f}{\partial f}$

Example



$\frac{\partial f}{\partial f} = 1$

$\frac{\partial f}{\partial x} = 1 \cdot 4 \cdot 1 = 4$

$\frac{\partial f}{\partial y} = 1 \cdot 4 \cdot 1 = 4$

$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial z}\frac{\partial f}{\partial w} = (2+3) \cdot 1 = 5$

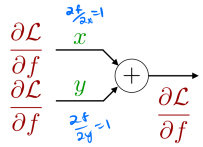
$x' = 2 - .2(4) = 1.2$

$y' = 3 - .2(4) = 2.2$

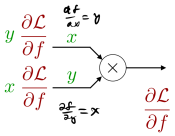
$z = 4 - .2(5) = 3$

Gradient Gates

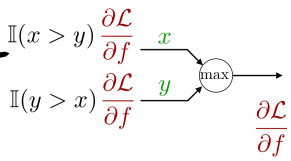
Add: Distribute gradient



Multiply: Switches gradient



Max: Routes gradient



Convergence: Lower total derivatives, sum all

Multivariate Grads + Chain Rule

Jacobian (vector-vector) $\nabla_x y = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_1} \\ \vdots & & \\ \frac{\partial y_1}{\partial x_m} & & \frac{\partial y_n}{\partial x_m} \end{bmatrix}$

$y \in \mathbb{R}^n \quad x \in \mathbb{R}^m$

$d_z = (\nabla_x y)^T$

$W \in \mathbb{R}^{h \times n} \quad x \in \mathbb{R}^n \quad y \in \mathbb{R}^h$

$\nabla_x x^T y = y$

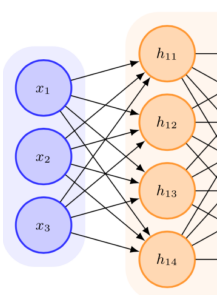
$\nabla_x W x = W^T$

$\nabla_W W x$ ought to be x^T

Correct (mostly), only due to sparse matrix

Back Prop NN-Layer

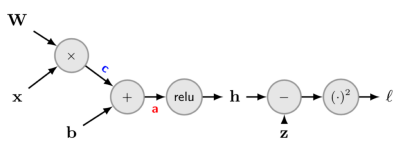
$y = Wx$



$\frac{\partial R}{\partial x} = W^T \frac{\partial L}{\partial y}$

$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial y} x^T$

Here, $\mathbf{h} = \text{ReLU}(\mathbf{W}\mathbf{x} + \mathbf{b})$, with $\mathbf{h} \in \mathbb{R}^h$, $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{W} \in \mathbb{R}^{h \times m}$, and $\mathbf{b} \in \mathbb{R}^h$. While many output cost functions might be used, let's consider a simple squared-loss output $\ell = (\mathbf{h} - \mathbf{z})^2$ where $\mathbf{z}$ is some target value.



$\nabla_h \ell = 2(h - z)$, $\frac{\partial \ell}{\partial a} = \mathbb{I}(a > 0) \odot \frac{\partial \ell}{\partial h}$,

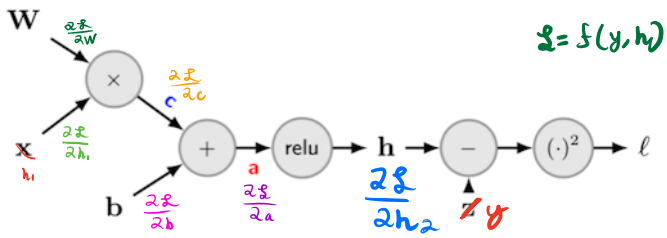
$\frac{\partial \ell}{\partial c} = \frac{\partial \ell}{\partial a}$, $\frac{\partial \ell}{\partial x} = \frac{\partial c}{\partial x}\frac{\partial \ell}{\partial c} = W^T \frac{\partial \ell}{\partial c}$

$\frac{\partial \ell}{\partial W} = \frac{\partial \ell}{\partial c} x^T$

Lec 8 : Init and Batch Norm

11/31

Backprop Basic NN Layer:



$\mathcal{L} = f(y, h)$

$\frac{\partial \mathcal{L}}{\partial a} = (x > 0)\frac{\partial \mathcal{L}}{\partial h_2}$

$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial a}\frac{\partial a}{\partial c} = \frac{\partial \mathcal{L}}{\partial a}$

$\frac{\partial \mathcal{L}}{\partial W} = h_1^T \frac{\partial \mathcal{L}}{\partial c}$ $\frac{\partial \mathcal{L}}{\partial h_1} = W^T \frac{\partial \mathcal{L}}{\partial c}$

Initializations:

- Too small - decline to zero, Too large - explode to infinity

Xavier Init:

$$w's \in \mathcal{N}\left(0, \frac{2}{n_{in} + n_{out}}\right)$$

Derivation:

$$Var(wh) = E^2[h]Var(h) + E^2[w]Var(w) + Var(w)Var(h)$$

$$Var(h_i) = Var(h_{i-1}) \cdot \sum_{i}^{n_{in}} Var(w_{ij}) \qquad h_i = \sum w_{ij} \cdot h_{i-1,j}$$

since $Var(h_i) = Var(h_{i-1})$, need $\sum Var(w) = 1$, thus

$$Var(w) = \frac{1}{n_{in}}$$

Same for backprop and n_{out}

Batch Norm:

For inputs, randomly sample to ensure similar distribution / mini batch.

For hidden layers, we use batch norm, solve for internal covariate shift

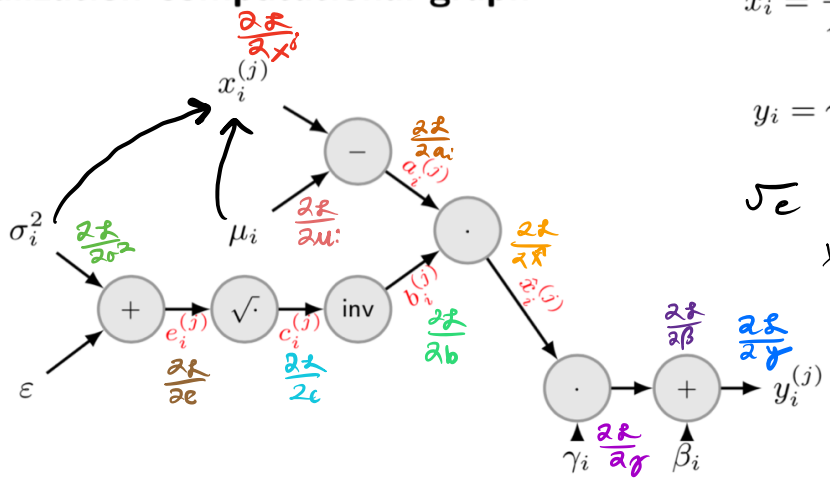
$$x'_i = \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}} \qquad \begin{cases} \mu_i = \frac{1}{m}\sum x_j^i \\ \sigma_i^2 = \frac{1}{m}\sum (x_i - \mu_i)^2 \end{cases}$$

across all samples (m) in mini batch

Scale + shift parameters $y_i = \gamma \hat{x}_i + \beta_i$

Batch Norm Backwards Pass

Batch normalization computational graph



$$\hat{x}_i = \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}}$$

$$y_i = \gamma_i \hat{x}_i + \beta_i$$

$\sqrt{e}$

$\frac{1}{2} e^{-\frac{1}{2}}$

$$\frac{\partial \ell}{\partial \beta} = \frac{\partial y}{\partial \beta} \frac{\partial \ell}{\partial y_i} = 1 \cdot \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

$$\frac{\partial \ell}{\partial \gamma} = \frac{dy}{d\gamma} \cdot \frac{\partial \ell}{\partial y_i} = \sum_{i=1}^{m} \hat{x} \cdot \frac{\partial \ell}{\partial y_i}$$

$$\frac{\partial \ell}{\partial \hat{x}} = \frac{\partial y}{\partial \hat{x}} \cdot \frac{\partial \ell}{\partial y_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial a} = \frac{\partial \hat{x}}{\partial a} \cdot \frac{\partial \ell}{\partial \hat{x}} = \frac{1}{\sqrt{\sigma^2 + \varepsilon}} \frac{\partial \ell}{\partial \hat{x}}$$

$$\frac{\partial \ell}{\partial u} = \frac{\partial a}{\partial u} \cdot \frac{\partial \ell}{\partial a} = -\frac{1}{\sqrt{\sigma^2 + \varepsilon}} \sum_{i=1}^{m} \frac{\partial \ell}{\partial \hat{x}}$$ u_i contributes to all j's

$$\frac{\partial \ell}{\partial b} = \frac{\partial \hat{x}}{\partial b} \cdot \frac{\partial \ell}{\partial \hat{x}} = (x - u_i) \frac{\partial \ell}{\partial \hat{x}}$$

$$\frac{\partial \ell}{\partial c} = \frac{\partial b}{\partial c} \cdot \frac{\partial \ell}{\partial b} = -\frac{1}{\sigma^2 + \varepsilon} \cdot \frac{\partial \ell}{\partial b}$$

$$\frac{\partial \ell}{\partial e_i} = \frac{\partial c}{\partial e_i} \cdot \frac{\partial \ell}{\partial c} = \frac{1}{2} \frac{1}{\sqrt{\sigma^2 + \varepsilon}} \cdot \frac{\partial \ell}{\partial c}$$

$$\frac{\partial \ell}{\partial \sigma^2} = \frac{\partial e_i}{\partial \sigma^2} \cdot \frac{\partial \ell}{\partial e_i} = 1 \cdot \sum_{j=1}^{m} \frac{\partial \ell}{\partial e_i}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial a}{\partial x} \cdot \frac{\partial \ell}{\partial a} + \frac{\partial \sigma^2}{\partial x_i} \cdot \frac{\partial \ell}{\partial \sigma^2} + \frac{\partial u}{\partial x_i} \cdot \frac{\partial \ell}{\partial u}$$

$$= 1 \cdot \frac{1}{\sqrt{\sigma^2 + \varepsilon}} \frac{\partial \ell}{\partial \hat{x}} + \frac{2}{m} (x_i - u) \frac{\partial \ell}{\partial \sigma^2} + \frac{1}{m} \frac{\partial \ell}{\partial u}$$

Lec 9 : 2/2

Regularization – any modification to reduce generalization error, but not training error.

- Increase estimator bias, but decrease variance
- Prevents overfitting

Parameter Norm –

$$J(\theta) + \alpha \Omega(\theta)$$

$\frac{1}{2} w^T w$ L2 Norm

$\|w - b\|$ Prior

$\|w_1 - w_2\|$ similar features

$\|w\|_1$ L_1 - sparse solutions

$\|h\|_1$ Sparse representation

Multi task learning: Composed NN's

Transfer learning: Add a head layer(s) with little training on existing model for similar task

Ensemble Methods: Train many models then avg

Bagging – Ic of M samples, k models, avg (M total sample size > N)

Dropout: Mask nodes with fixed prob. Approximates bagging in single model

Sample a binary mask for all units then do FW, BW pass.

New mask for every SGD step. 2^N configs

- Approximates bagging in one model
- Regularizing each hidden unit to work in many contexts
  - May encode redundancy
- Scale by 1/p in training so output is appropriately scaled.

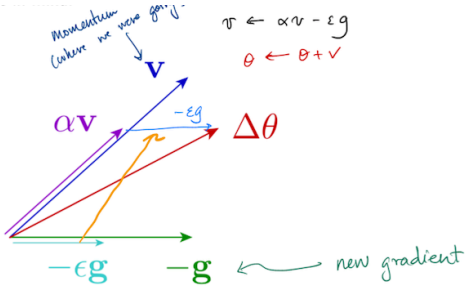
Optimization (GD variations):

Momentum

Momentum: $\alpha \in [0,1]$, learn rate ε

$$v \leftarrow \alpha v - \varepsilon g$$
$$\theta \leftarrow \theta + v$$

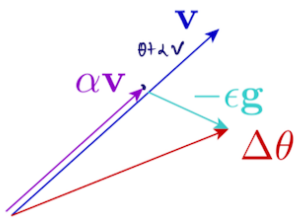
Pros: Avg's out back and forth, can pull us out of local optima



Nesterov Momentum

$$v \leftarrow \alpha v + \varepsilon \nabla_\theta J(\theta + \alpha v)$$
$$\theta \leftarrow \theta + v$$

Pros: Gradient computed after momentum step so more accurate.



Adagrad:

Scales grad down over time... drops

$$a \leftarrow a + g \odot g$$
$$\theta \leftarrow \theta - \frac{\varepsilon}{\sqrt{a}+v} \odot g$$

RMS Prop: Weighted version of adagrad (component wise)

$$a \leftarrow a\beta + (1-\beta)\, g \odot g$$
$$\theta \leftarrow \theta - \frac{\varepsilon}{\sqrt{a}+v} \odot g$$

Adam:

1st moment momentum $v = \beta_1 v + (1-\beta_1)\, g$

2nd moment grad norm $a = \beta_2 a + (1-\beta_2)\, g \odot g$

Bias correction $\tilde{v} = \frac{1}{1-\beta_1^t} v \qquad \tilde{a} = \frac{1}{1-\beta_2^t} a$

Step:

$$\theta \leftarrow \theta - \frac{\varepsilon}{\sqrt{\tilde{a}} + v} \odot \tilde{v}$$

Convolution

In practice, it's cross corr, or just dot (not flip kernel)

- typically depth matches

Parameters: $(h_f \times w_f \times d) \times \#$ of filters

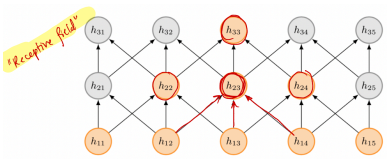
Sparsity:

CNN: $w_f \cdot h_f \cdot d$ weights / neuron

FC: $w \cdot h \cdot d$ weights / neuron

Receptive Field:

# of inputs a neuron 'sees'



Padding: Add zeros to maintain output size

Stride: Skip on drag of filter window

$$\left(\frac{w - w_f + 2\text{pad}}{\text{stride}} + 1, \frac{h - h_f + 2\text{pad}}{\text{stride}} + 1\right)$$

Pooling layer: Down sampling using specified operation + stride

No parameters, deal with spatial dimensions

$$\left(\frac{w - w_p}{\text{stride}} + 1, \frac{h - h_p}{\text{stride}} + 1\right)$$