ECE C147 Midterm Cheat Sheet

$$\nabla_w\left(\frac{\alpha}{2}w^Tw\right)=\alpha w$$

L2 Regularization

$$\tilde{J}(w;X,y)=J(w;X,y)+\frac{\alpha}{2}w^Tw$$

$$w\leftarrow(1-\epsilon\alpha)w-\epsilon\nabla_wJ(w;X,y)$$

ML basics

K-nearest → too much memory, testing is expensive
→ distance measurements in high-D space do not reflect similarity

- calculate distance between x^{new} & $x^{(i)}\ \forall\ i$
- take the K nearest indices, classify via mode of $\{y^{(1)},y^{(2)},\dots,y^{(\alpha)}\}$

Pros: Simple

Cons: Slow/Scales w/ amount of training data, doesn't work if dimensionality is high, means neighbors are far away

L2 dist between rows: $\sqrt{(x_1-y_1)^2}\rightarrow\sqrt{(x_1-y_1)^2}=\sum x_i^2+\sum y_i^2-2\sum x_iy_i$

Broadcasting

class origin	one hot	Size 1
(2x3)	(3x1)	
(3)	(3)	
3x3	3x3	

Frobenius Norm

$$\|X\|_F^2=tr(X^TX)$$

Training data: learn model parameters
Testing data: Scoring the model
Validation set: select optimal hyperparameters (different from test set)
Test set: used as proxy to see how well the architecture and hyperparameters would generalize to new data.
Test set used once after cross-validation (each fold is used k-times for training/validation)
Everything prevents overfitting

Stats

$$E[\delta\delta^T]=\sigma^2,\ E[\delta]=0$$ (gaussian noise)

Softmax

$$\text{Softmax}_i(x)=\frac{e^{w_i^Tx+b_i}}{\sum_{j=1}^{c}e^{w_j^Tx+b_j}},\quad P(y^{(j)}=i\mid x^{(j)},\theta)=\text{softmax}_i(x^{(j)})$$

score: $a_j(x^{(j)})=w_j^Tx^{(j)}+b_j$

$$\theta^*=\underset{\theta}{\text{argmax}}\sum_{j=1}^{m}\log\text{softmax}_{y^{(j)}}(x^{(j)})$$

$$=\underset{\theta}{\text{argmax}}\sum_{j=1}^{m}\log\left[\frac{e^{w_{y^{(j)}}^Tx^{(j)}+b_{y^{(j)}}}}{\sum_{k=1}^{c}e^{w_k^Tx^{(j)}+b_k}}\right]$$

$$=\underset{\theta}{\text{argmax}}\frac{1}{m}\sum_{j=1}^{m}\left[a_{y^{(j)}}(x^{(j)})-\log\left(\sum_{k=1}^{c}e^{a_k(x^{(j)})}\right)\right]$$

$$=\underset{\theta}{\text{argmax}}\frac{1}{m}\sum_{j=1}^{m}\left[\log\left(e^{a_{y^{(j)}}(x^{(j)})}\right)-\log\left(\sum_{k=1}^{c}e^{a_k(x^{(j)})}\right)\right]$$

$\mathcal{L}=$ log likelihood

$\nabla_{w_i}\mathcal{L}$: ($y^{(j)}=i$: the class of the j^{th} sample)

$$\nabla_{w_i}\mathcal{L}=\nabla_{w_i}\left(\frac{1}{m}\sum_{j=1}^{m}\left(\log\left(\sum_{k=1}^{c}e^{w_k^Tx^{(j)}+b_k}\right)-\left(w_{y^{(j)}}^Tx^{(j)}+b_{y^{(j)}}\right)\right)\right)$$

$$=\frac{1}{m}\sum_{j=1}^{m}\left[\nabla_{w_i}\left(\log\left(\sum_{k=1}^{c}e^{w_k^Tx^{(j)}+b_k}\right)\right)-\nabla_{w_i}\left(w_{y^{(j)}}^Tx^{(j)}+b_{y^{(j)}}\right)\right]$$

(derivative of b w.r.t. a specific class $\in\{1,\dots,c\}$)

$$=\frac{1}{m}\sum_{j=1}^{m}\left[\nabla_{w_i}\left(\log\sum_{k=1}^{c}e^{w_k^Tx^{(j)}+b_k}\right)\right]-z$$

How many times does $y^{(j)}=i$? Let z represent the # of samples that belongs to class i

$$\nabla_{w_i}\left(\log\left(\sum_{k=1}^{c}e^{w_k^Tx^{(j)}+b_k}\right)\right)=\frac{\sum_{k=1}^{c}\nabla_{w_i}\left(e^{w_k^Tx^{(j)}+b_k}\right)}{\sum_{k=1}^{c}e^{w_k^Tx^{(j)}+b_k}}=\frac{e^{w_i^Tx^{(j)}+b_k}x^{(j)}}{\sum_{k=1}^{c}e^{w_k^Tx^{(j)}+b_k}}$$

$$\therefore\ \frac{\partial\mathcal{L}}{\partial w_i}=\frac{1}{m}\left[\sum_{j=1}^{m}\left(\frac{e^{w_i^Tx^{(j)}+b_i}}{\left(\sum_{k=1}^{c}e^{w_k^Tx^{(j)}+b_k}\right)}\right)x^{(j)}\right]-zx^{(j)}$$

$$\nabla_{b_i}\mathcal{L}=\nabla_{b_i}\left(\frac{1}{m}\sum_{j=1}^{m}\left(\log\left(\sum_{k=1}^{c}e^{w_k^Tx^{(j)}+b_k}\right)-\left(w_{y^{(j)}}^Tx^{(j)}+b_{y^{(j)}}\right)\right)\right)$$

$$=\frac{1}{m}\sum_{j=1}^{m}\left(\nabla_{b_i}\left(\log\sum_{k=1}^{c}e^{w_k^Tx^{(j)}+b_k}\right)-\nabla_{b_i}\left(w_{y^{(j)}}^Tx^{(j)}+b_{y^{(j)}}\right)\right)$$

(same rule)

Again, let z represent the # of samples that belong to class j

$$\nabla_{b_i}\left(\log\left(\sum_{k=1}^{c}e^{w_k^Tx^{(j)}+b_k}\right)\right)=\frac{\sum_{k=1}^{c}\nabla_{b_i}\left(e^{w_k^Tx^{(j)}+b_k}\right)}{\sum_{k=1}^{c}e^{w_k^Tx^{(j)}+b_k}}=\frac{e^{w_i^Tx^{(j)}+b_i}}{\sum_{k=1}^{c}e^{w_k^Tx^{(j)}+b_k}}$$

$$\therefore\ \frac{\partial\mathcal{L}}{\partial b_i}=\frac{1}{m}\left[\sum_{j=1}^{m}\left(\frac{e^{w_i^Tx^{(j)}+b_i}}{\left(\sum_{k=1}^{c}e^{w_k^Tx^{(j)}+b_k}\right)}\right)\right]-z$$

$$\nabla_x(w^Tx)=w^T$$

$$\frac{\partial\ell}{\partial W}=\frac{\partial\ell}{\partial Y}X^T\qquad Y=WX$$

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

$$-tr\left(W\sum_{i=1}^{K}x_iv_i^T\right)=-tr(WXY^T)$$ ✓ HW#1

$$=\frac{\partial}{\partial W}\left(-\frac{1}{m}\sum_{i=1}^{m}x^{(i)T}W^Ty^{(i)}\right)$$

$$=-\frac{\partial}{\partial W}(tr(X^TW^TY))$$

$$[x^{(1)}x^{(2)}\cdots x^{(m)}]\begin{bmatrix}w_{11}&\cdots&w_{1n}\\ \vdots&\ddots&\vdots\\ w_{m1}&\cdots&w_{mn}\end{bmatrix}\begin{bmatrix}y^{(1)}\\ \vdots\\ y^{(n)}\end{bmatrix}$$

$$\nabla_x\|x\|_2=\frac{x}{\|x\|_2}$$

$$\nabla_\theta\theta^Tg=g$$

$$\nabla_\theta(\theta^TH\theta)=(H+H^T)\theta$$

Linear Algebra

① $Av_i=\lambda_iv_i\rightarrow\|v_i\|^2=\|\lambda_iv_i\|^2=|\lambda_i|^2\|v_i\|_2^2$ norm of eigenvalues = 1

② $v_i^TAv_i=\lambda_iv_i^Tv_i=\lambda_i\|v_i\|^2,\ \forall A\succ0\ \forall v_i,\ \lambda_i>0$

③ $A=U\Sigma V^T$ (left singular eigenvectors of AA^T; right singular eigenvectors of A^TA)

$AA^T=U\Sigma\Sigma^TU^T$ (diagonal) $\lambda(AA^T)=\sigma^2A$ ✓ U is orthogonal

④ $Q^TQ=QQ^T=I$

⑤ $\|Qv\|=\|v\|$

⑥ $\det(Q^TQ)=\det^2(Q)=\det(I)$

$$Cov(Ax+b)=A\,Cov(x)\,A^T$$

Hinge Loss

$$\nabla_W \mathcal{L} = \frac{1}{K}\sum_{i=1}^{K} \mathbb{I}_{\{y^{(i)}(W^T x^{(i)}+b) \le 1\}} \odot (-y^{(i)} x^{(i)})$$

Gradient Descent

- don't use a numerical gradient ∴ # of params high, expensive
- batch: use all m training set examples
- Minibatch: use K examples
- Stochastic: use 1 example

Neural Networks

- Activation is usually RELU
  - maxout = $\max(W_1^T x + b_1, W_2^T x + b_2)$

Output Activation

"log loss"

$MSE = \frac{1}{2}\sum (y^{(i)} - \sigma(z^{(i)}))^2$ ↓ goes not in class 1

✓ cross entropy loss

$CE = -\sum [y^{(i)} \log \sigma(z^{(i)}) + (1-y^{(i)}) \log(1-\sigma(z^{(i)}))]$ → softmax

If z is large, the gradient is 0, therefore no learning

usually softmax

Activations - introduce nonlinearity

- without an activation, everything is linear and simplifies to $z = Wx+b$
- Sigmoid: $\frac{1}{1+e^{-x}} = \sigma$, $\frac{\partial \sigma}{\partial x} = \sigma(1-\sigma)$
  - around $x=0$, linear, differentiable everywhere
  - Saturates @ extremes, 0 gradient, no learning
  - could zigzag
- hyperbolic: $\tanh(x) = 2\sigma - 1$, $\frac{d(\tanh(x))}{dx} = 1 - \tanh^2(x)$
  - similar props to sigmoid, passes through 0
- Relu: $\max(x, 0)$
  - converges faster, linear if active, no saturation at large values
  - non-neg, zig-zags, no learning if activation is 0
- softplus: $\log(1+e^x)$
  - empirically worse than ReLU
- Leaky Relu: $x^*(x>0) + 0.1x^*(x>0)$
  - learning if $x<0$, if α is hyperparam, PReLU
- ELU: $\max(\alpha(e^x - 1), x)$
  - computation of exponential is expensive

Initialization

- if too small, $\sigma = 0.01$, decay → 0, last layer gradients are small
- if too large, $\sigma = 1$, explodes, tanh causes fluctuation
- Xavier initialization

$$Var(h_i) = Var(h_0) = Var(\nabla_{h_i} L) = Var(\nabla_{h_0} L)$$

Variance of units across all layers must be the same

$$Var(h_i) = Var\left(\sum_{j=1}^{n_{in}} w_{ij} h_{i-1,j}\right), \quad Var(wh) = E^2(w)Var(h) + E^2(h)Var(w) + Var(w)Var(h)$$

if $E[w] = E[h] = 0$,

$$Var(h_i) = Var(h_{i-1}) \cdot \sum_{j=1}^{n} Var(w_{ij})$$

$$Var(w) \times n_{in} = 1, \quad Var(w) = \frac{1}{n_{in}}$$ ✓ # of neurons in a layer

Both

$$\mathcal{U}\left(0, \frac{2}{n_{in} + n_{out}}\right)$$

2015:

$$\text{Uniform}\left(-\frac{\sqrt{6}}{n_{in}+n_{out}}, \frac{\sqrt{6}}{n_{in}+n_{out}}\right)$$

if relu

Regularization

Sacrifices higher training loss for lower testing loss.

- improves generalization

Stopping early

- return model with lowest validation error, train until the error is unchanged

param norm penalty

$\mathcal{L}(\theta) + \alpha \Omega(\theta)$

Dropout

approximates bagging

L2

$\Omega(\theta) = \frac{1}{2} W^T W = \frac{1}{2}\|W\|_F^2$

- Shrink weights before gradient update
- $\tilde{J} = J + \frac{\alpha}{2} W^T W$

Instead of sparse params, it may be beneficial to have small representations

$\nabla_W \tilde{J} = \nabla_W J + \alpha W$

$W = (1 - \epsilon\alpha) W - \epsilon \nabla_W J$

α causes weights to shrink

- W could be near 0 $\|W - 0\|_2^2$

L1

$\Omega(\theta) = \sum |W_j|$

- good for feature selection because gradient will have some weights go to exactly 0

$\Omega(h^{(i)}) = \|h^{(i)}\|_1$

dataset augmentation

Multi-task learning

transfer learning

- adds new layer at the end

introduces noise to model

ensemble methods - any multiple models

- bagging: K datasets/models small random size

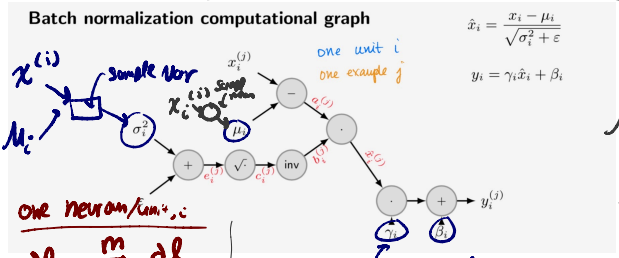
$$H(y,p) = -\sum_i y_i \log(p_i) = L$$

Batchnorm

makes each layer have unit statistics

- $\hat{x}_i = \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$, $\mu_i = \frac{1}{m}\sum_{j=1}^{m} x_i^{(j)}$, $\sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m}(x_i^{(j)} - \mu_i)^2$

Allows higher learning rates to be used.



$$\mu_i = \frac{1}{m}\sum_{j=1}^{m} x^{(i)} \qquad \frac{\partial \mu}{\partial x^{(i)}} = \frac{1}{m}$$

$$h_i = f(\text{batchnorm}(W_i h_{i-1} + b_i))$$

momentum in mini-batch gradient descent

- exponentially weighted moving average

Optimization

- Stochastic gradient descent / batch

$g = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta J(\theta)$ using m examples from training set

$\theta = \theta - \epsilon g$

Momentum

$v = 0$

while last_diff > tolerance:

$v = \alpha v - \epsilon \nabla_\theta J(\theta)$

$\theta += v$

pushes for local, shallow, flat, minima

Nesterov momentum

$\alpha = 0.9$ higher weight to more recent gradients

temp: p

$p = \alpha p - \epsilon g$

$\theta += p + \alpha(p - temp)$

also finds local minimum

$$\frac{\partial l}{\partial \beta} = \sum_{j=1}^{m} \frac{\partial l}{\partial y^{(j)}}$$

$$\frac{\partial l}{\partial \gamma} = \sum_{j=1}^{m} \frac{\partial l}{\partial y^{(j)}} \hat{x}^{(j)}$$

$$\frac{\partial l}{\partial \hat{x}^{(j)}} = \frac{\partial l}{\partial y^{(j)}} \gamma$$

$$\frac{\partial l}{\partial \sigma^{(j)}} = \frac{1}{\sqrt{\sigma^2 + \epsilon}} \frac{\partial l}{\partial \hat{x}^{(j)}}$$

$$\frac{\partial l}{\partial \mu} = -\frac{1}{\sqrt{\sigma^2 + \epsilon}} \sum_{j=1}^{m} \frac{\partial l}{\partial \hat{x}^{(j)}} - \left[\frac{2}{m}\sum_{j=1}^{m}(x^{(j)} - \mu)\right] \frac{\partial l}{\partial \sigma^2}$$

$$\frac{\partial l}{\partial b^{(j)}} = (x^{(j)} - \mu) \frac{\partial l}{\partial \hat{x}^{(j)}}$$

$$\frac{\partial l}{\partial c^{(j)}} = -\frac{1}{\sigma^2 + \epsilon}(x^{(j)} - \mu) \frac{\partial l}{\partial \hat{x}^{(j)}}$$

$$\frac{\partial l}{\partial e^{(j)}} = -\frac{1}{2} \frac{1}{(\sigma^2 + \epsilon)^{3/2}} (x^{(j)} - \mu) \frac{\partial l}{\partial \hat{x}^{(j)}}$$

$$\frac{\partial l}{\partial \sigma^2} = \sum_{j=1}^{m} \frac{\partial l}{\partial e^{(j)}} = \sum_{j=1}^{m} -\frac{1}{2} \frac{1}{(\sigma^2 + \epsilon)^{3/2}} (x^{(j)} - \mu) \frac{\partial l}{\partial \hat{x}^{(j)}}$$

$$\frac{\partial l}{\partial x^{(j)}} = \frac{\partial l}{\partial a^{(j)}} + \frac{\partial \sigma^2}{\partial x^{(j)}} \frac{\partial l}{\partial \sigma^2} + \frac{\partial \mu}{\partial x^{(i)}} \frac{\partial l}{\partial \mu}$$

$$= \frac{1}{\sqrt{\sigma^2 + \epsilon}} \frac{\partial l}{\partial \hat{x}^{(i)}} + \frac{2(x^{(i)} - \mu)}{m} \frac{\partial l}{\partial g^2} + \frac{1}{m} \frac{\partial l}{\partial \mu}$$

Adagrad

- learning rate decreased by grad norms

Let a denote running sum of squares of gradient norms

$a = a + g \odot g$

"if we've already stepped far in one direction, we'll step less in that direction in the future"

$\theta = \theta - \frac{\epsilon}{\sqrt{a} + \nu} \odot g$

RMS Prop

- augments adagrad by making the gradient accumulator an exponentially weighted moving average

"Let's consider recent history of grad norm squared, and place a little less emphasis on our grad norm so we don't shut off one direction too too much!"

$a = Ba + (1 - B)\, g \odot g$ or on $\beta = 0.99$

$\theta = \theta - \frac{\epsilon}{\sqrt{a} + \nu} \odot g$

RMS Prop + momentum

"Let's consider recent history of both gradient and grad norm squared. Let's try and step more in direction of recent gradients, & still place emphasis on recent norm squared, but not too much regarding our grad norm squared

$a = Ba + (1 - B)\, g \odot g$

$V = \alpha V - \frac{\epsilon}{\sqrt{a} + \nu} \odot g$

$\theta = \theta + V$ (can also use Nesterov)

Adam

- momentum-like + Adagrad/RMS prop-like, 2 moments + gradient

$-v = \beta_1 v + (1-\beta_1) g$

- bias correction amplifies 2nd moment

$\beta_1 = 0.9, \ \beta_2 = 0.999$

$v = \beta_1 v + (1-\beta_1) g$ ← momentum-like

$a = \beta_2 a + (1-\beta_2)\, g \odot g$ ← gradient normalization

$\hat{v} = \frac{1}{1-\beta_1^t} v$ ← bias correction

$\tilde{a} = \frac{1}{1-\beta_2^t} a$ ← "to the power of t"

all in one step: $\tilde{v}$

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

- First order method
  - doesn't use second derivatives
  - computes g, but not H

Newton's method: $\theta = \theta - H^{-1} g$

Bias Correction

- Fixes 1st & 2nd moments at the beginning of training of gradient descent. makes sure it starts off well
- First steps more effective

Miscellaneous

- exploding gradients
  - gradient clipping, $g = \frac{g}{\|g\|}$ clip
- vanishing gradients → architecture, regularization

$W = U \Lambda V^{-1}, \ W^T = U \Lambda^T U^{-1}$ eigenvector u_i has λ_i^2

CNN

$(W - W_{f+1}, h - h_{f+1})$

Padding + Stride $\left(\frac{W - W_{f+2*pad}}{Stride} + 1, \ \frac{h - h_{f+2*pad}}{Stride} + 1 \right)$

pooling: no padding

Dataset Augmentation

- flip, scale, rotate, crop images to generate more data, translation
- inject noise into network
- label smoothing

$\mathcal{L}^{(i)} = -\sum_{c=1}^{C} y_c \log(p_c)$

- subsampling
- brightness adjustments
- gaussian blurring

Multitask learning

- train to perform multiple tasks
- multiple tasks share common factors to explain variations in data

Transfer Learning

- freeze most parameters of original network

applicable even if you have a small amount of data!

Bootstrap aggregating

- produces different subdatasets based on randomly selecting N elements K times, creates K different models which makes your model more generalizable

Ensemble Methods

1. Train multiple different models
2. Average results at test time

K independent models, avg model error will decrease by $\frac{1}{K}$

expected mean error square error, $\frac{1}{K} \mathbb{E}[\epsilon_i^2]$

if dependent, $\frac{1}{K}\mathbb{E}[\epsilon_i^2] + \frac{K-1}{K}\mathbb{E}[\epsilon_i \epsilon_j] = \mathbb{E}[\epsilon_i^2]$ if $i=j$, perfect correlation

Bagging

- bootstrap aggregating
- method for regularization
- construct K datasets by setting a data size N, and drawing w/ replacement from the original dataset to get N samples, we do this K times
- train K different models using K datasets
- model averaging is expensive for large networks redundant features

Dropout

- approximates bagging procedure
- applies a mask of bernoulli r.v.'s
- each mask is like a different model
- N hidden units → 2^N model configurations each one good at predicting output
- Dropout regularizes each unit to many different contexts
- Dropout may cause units to encode redundant features

---

minimizing -log $\mathcal{L}$ ⟷ max $\mathcal{L}$, min CE_{loss}

① Gradients computed using SGD will be noisier than gradient computed using Batch gradient descent, because it is a single sample estimate

② SGD will take more steps than Batch GD, but converges faster in terms of clock time

"Each step is super fast in SGD, and each step is slower in batch GD"

③ SGD avoids the trap of poor local minima due to Zig Zagging. Every sample is unlikely to have the same minimum as the full batch.

④ Batch gradient descent is more expensive than SGD because it requires loading the entire data.

Difference between batchnorm in training VS. testing.

During training, it first computes the mean & variance of the mini-batch data in the unit-wise. Batch norm normalizes the layer's input based on the mean & variance. It then applies γ, β for scaling and shifting (learnable parameters). Keeps track of the running average for mean and variance through all batches. During testing, BatchNorm uses the fixed accumulated running average from training for normalizing test data. The learned scaling/shifting γ, β are applied to normalized data.

$f(x) = x^3$: +: non-linear / differentiable everywhere
−: exploding gradients, g can be large for inputs with large abs values
Small inputs close to 0 can lead to vanishing gradients

Increasing # of neurons: fits model more, can fix underfitting but can lead to overfitting

L1 regularization: Encourages model to optimize some feature weights to 0. This allows the model to learn simpler & sparser patterns by pushing less important feature weights to 0, which may prevent overfitting.
By observing trained weights, we can implement feature selection to simplify the model and save computation resource

Bias ∈ Adam: running averages are biased towards 0 at start of training. Without bias, the optimizer is likely to take large steps in the initial updates of the model that leads to unstable training and lower convergence. The bias correction help this initial estimations to be more accurated & stable.

$$L^{(i)} = L_{CE}(y^{(i)}, \hat{y}^{(i)}) = y^{(i)} \log(\hat{y}^{(i)}) + (1-y^{(i)}) \log(1-\hat{y}^{(i)})$$

↑ probability   ^ for singular unit

$$\frac{\partial L^{(i)}}{\partial \hat{y}^{(i)}} = \frac{y^{(i)}}{\hat{y}^{(i)}} - \frac{1-y^{(i)}}{1-\hat{y}^{(i)}}$$

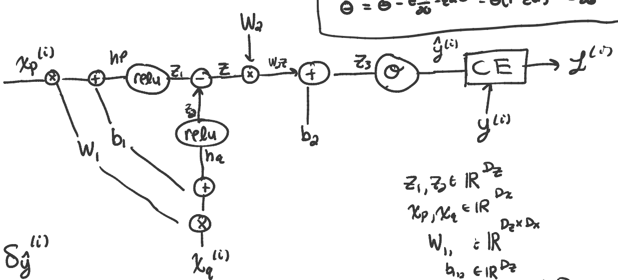
Mathematically show that ℓ_2 regularization shrinks the weight in gradient descent

$\tilde{\mathcal{L}}(\theta; X, y) = \mathcal{L}(\theta; x, y) + \frac{\alpha}{2}\|\theta\|_2^2$

$\frac{\partial \tilde{\mathcal{L}}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial \theta} + \alpha\theta$

$\theta = \theta - \epsilon\frac{\partial \mathcal{L}}{\partial \theta} - \epsilon\alpha\theta = \theta(1-\epsilon\alpha) - \epsilon\frac{\partial \mathcal{L}}{\partial \theta}$ (weight decay)

d) inverted dropout
training: generates binary mask ⊙ values after affine transform / p (prob of keeping)
test: nothing
/p causes output to have same expected value as if dropout was never performed



$z_1, z_3 \in \mathbb{R}^{D_z}$
$x_p, x_q \in \mathbb{R}^{D_x}$
$W_1 \in \mathbb{R}^{D_z \times D_x}$
$b_{1,2} \in \mathbb{R}^{D_z}$
$W_2 \in \mathbb{R}^{1 \times D_z}$

b) $\delta_{\hat{y}}^{(i)}$

c) $\hat{y}^{(i)} = \frac{1}{1+e^{-z_3}}$ $\frac{\partial \mathcal{L}^{(i)}}{\partial z_3} = \frac{\partial \hat{y}^{(i)}}{\partial z_3}\frac{\partial \mathcal{L}}{\partial \hat{y}^{(i)}}$

$\frac{\partial \hat{y}^{(i)}}{\partial z_3} = \frac{\partial}{\partial z_3}(1+e^{-z_3})^{-1} = -\frac{1}{(1+e^{-z_3})^2} \times -e^{-z_3} = \frac{e^{-z_3}}{(1+e^{-z_3})^2}$

$$\delta_{z_3} = \frac{e^{-z_3}}{(1+e^{-z_3})^2}\delta_{\hat{y}^{(i)}}$$

$D_z \times D_x$

d) $\delta_{b_2} = \delta_{z_3}$

$D_z \times D_x =$ $\swarrow \mathbb{R}^{D_z}$

$\frac{\partial \mathcal{L}^{(i)}}{\partial W_2} = \frac{\partial z_3}{\partial W_2}\frac{\partial \mathcal{L}^{(i)}}{\partial z_3}$

e) $z_3 = W_2 z + b_2$

$\frac{\partial z_3}{\partial W_2} \in \mathbb{R}^{D_z \times D_x \times D_z}$ $(D_z \times D_x) = (D_z \times 1) \times (1 \times D_z)$
↑ this is wrong, doesn't fit

$$\delta_{W_2} = \delta_{z_3} z^T$$

f) $\frac{\partial \mathcal{L}}{\partial z} = \delta_{z_3} W_2^T$ ($D_z \times 1$; 1×1; $D_z \times 1$)

g) $\delta_{z_1} = \delta_z$

h) $\delta_{z_2} = -\delta_z$

i) $\frac{\partial \mathcal{L}}{\partial h_q} = \frac{\partial z_2}{\partial h_q}\frac{\partial \mathcal{L}}{\partial z_2} = \mathbb{I}(h_q > 0) \odot \delta_{z_2} = \delta_{h_q}$

j) $\delta_{h_p} = \mathbb{I}(h_p > 0) \odot \delta_{z_1}$

k) $\delta_{b_1} = \delta_{h_p} + \delta_{h_q}$

l) $\delta_{h_p} x_p^T + \delta_{h_q} x_q^T = \nabla_{W_1}\mathcal{L}^{(i)}$

Nesterov Momentum

$V_{new} = \alpha V_{old} - \epsilon g(\Theta_{old} + \alpha V_{old})$

$\Theta_{new} = \Theta_{old} + V_{new}$

$\Theta'_{old} = \Theta_{old} + \alpha V_{old}$

$V_{new} = \alpha V_{old} - \epsilon g(\Theta'_{old})$

$\Theta_{new} = \Theta_{old} + V_{new}$

$\Theta'_{new} - \alpha V_{new} = \Theta'_{old} - \alpha V_{old} + V_{new}$

$\Theta'_{new} = \Theta'_{old} + V_{new} + \alpha(V_{new} - V_{old})$

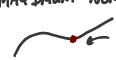
$\frac{\partial \mathcal{L}(\Theta'_{old})}{\partial \Theta'} = \frac{\partial \Theta}{\partial \Theta'}\frac{\partial \mathcal{L}(\Theta'_{old})}{\partial \Theta}$

$\frac{\partial \mathcal{L}(\Theta'_{old})}{\partial \Theta'} = \nabla_\Theta \mathcal{L}(\Theta_{old} + \alpha V_{old})$

We can take gradient w.r.t. same position $V_{new} = \alpha V_{old} - \epsilon \nabla_\Theta \mathcal{L}(\hat{\Theta}_{old})$ ↑ same thing

We can get either representation through change of variable. We don't have to calculate gradient at different value of $\Theta + \alpha V$.

Saddle point: curve decreases on one side of the saddle and increases on the other side of the saddle point. The gradient of a saddle point is equal to 0. NOT A LOCAL MAXIMUM NOR A LOCAL MINIMUM.

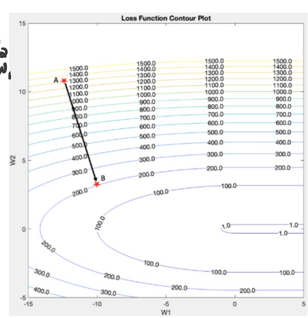


← poor local minima → minimum value of the function in its surrounding until a certain limit in all directions.
gradient is 0 @ local minimum.

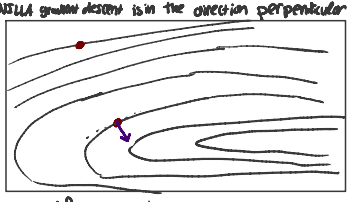
We need momentum to get out of a saddle point or local minima because the gradient is 0.

It is more likely to escape the saddle point than the local minima because after crossing local minima, the momentum decreases due to gradient being in opposite direction to momentum accumulated down the slope.

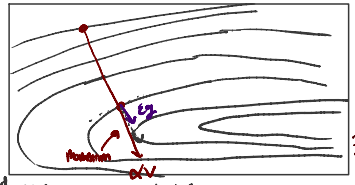
l) Gradient across W_2 is higher than along W_1.
Tighter lines mean higher gradient, lots of space means small gradient.



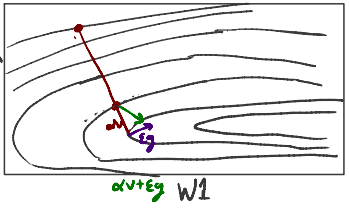
VANILLA gradient descent is in the direction perpendicular to contour line



Sgdf momentum



Sgdf momentum



Gradient descent + momentum scheme, find a general expression of V_t in terms of gradients $g_1, g_2, \cdots, g_t$ and ε, considering an initial value of momentum $V_0 = 0$.

$V_t = \alpha V_{t-1} - \varepsilon g_t$ $\quad \theta_t = \theta_{t-1} + V_t$

$V_1 = \alpha V_0 - \varepsilon g_1 = -\varepsilon g_1$

$V_2 = \alpha V_1 - \varepsilon g_2 = \alpha(-\varepsilon g_1) - \varepsilon g_2 = -\varepsilon(g_2 + \alpha g_1)$

$V_3 = \alpha V_2 - \varepsilon g_3 = \alpha(-\varepsilon(g_2 + \alpha g_1)) - \varepsilon g_3$

$= -\varepsilon(g_3 + \alpha g_2 + \alpha^2 g_1)$

$\Rightarrow V_t = -\varepsilon\left[g_t + \alpha g_{t-1} + \alpha^2 g_{t-2} + \cdots + \alpha^{t-1} g_1\right]$

$\mathbb{E}[V_t] = \mathbb{E}\left[-\varepsilon(g_t + \alpha g_{t-1} + \alpha^2 g_{t-2} + \cdots + \alpha^{t-1} g_1)\right]$

$= -\varepsilon\left[\mathbb{E}[g_t] + \alpha\mathbb{E}[g_{t-1}] + \cdots + \alpha^{t-1}\mathbb{E}[g_1]\right]$

Consider that the gradients above are i.i.d. random variables with μ, σ.

Find the expected value of weights θ_t at $t = 3$.

$\theta_1 = \theta_0 + V_1$

$\theta_3 = \theta_0 + V_1 + V_2 + V_3$

$= \theta_0 - \varepsilon\mathbb{E}[g_1] - \varepsilon\left(\mathbb{E}[g_2] + \alpha\mathbb{E}[g_1]\right) - \varepsilon\left(\mathbb{E}[g_3] + \alpha\mathbb{E}[g_2] + \alpha^2\mathbb{E}[g_1]\right)$

$= \theta_0 - \varepsilon\mu - \varepsilon(\mu + \alpha\mu) - \varepsilon(\mu + \alpha\mu + \alpha^2\mu)$

$= \theta_0 - \varepsilon\mu - \varepsilon\mu - \varepsilon\alpha\mu - \varepsilon\mu - \varepsilon\alpha\mu - \varepsilon\mu\alpha^2$

$= \theta_0 - 3\varepsilon\mu - 2\varepsilon\alpha\mu - \varepsilon\mu\alpha^2$

$= \theta_0 - \varepsilon\mu(3 + 2\alpha + \alpha^2)$

$= \theta_0 - \varepsilon\mu[1 + (1+\alpha) + (1+\alpha+\alpha^2)]$

$\mathbb{E}[V_t] = -\varepsilon\mu\left[\frac{1-\alpha^t}{1-\alpha}\right]$

$\mathbb{E}[\theta_3] = \mathbb{E}[\theta_0] + \sum_{t=1}^{3}\left[\frac{-\varepsilon\mu[1-\alpha^t]}{1-\alpha}\right]$

$= \mathbb{E}[\theta_0] - \frac{\varepsilon\mu}{1-\alpha}\left[\sum_{t=1}^{3}(1-\alpha^t)\right]$

$= \mathbb{E}[\theta_0] - \frac{\varepsilon\mu}{1-\alpha}\left[3 - \left(\frac{1-\alpha^3}{1-\alpha}\right)\right]$

$\theta_0 = 0$

$\theta_t = \theta_0 - \varepsilon\sum_{j=1}^{t}\sum_{i=1}^{j}\alpha^{j-i} g_i$

$\mathbb{E}[\theta_t] = -\varepsilon\sum_{j=1}^{t}\sum_{i=1}^{j}\alpha^{j-i}\mu$

$= -\varepsilon\mu\sum_{j=1}^{t}(1 + \alpha + \alpha^2 + \cdots + \alpha^{j-1})$

$= -\varepsilon\mu\sum_{j=1}^{t}\frac{1-\alpha^j}{1-\alpha}$

$= \frac{-\varepsilon\mu}{1-\alpha}\sum_{j=1}^{t}(1-\alpha^j)$

$\mathbb{E}[\theta_t] = \frac{-\varepsilon\mu}{1-\alpha}\left(t - \frac{\alpha(1-\alpha^t)}{1-\alpha}\right)$

Adagrad

The update equation:

$a \leftarrow a + g \odot g$

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

If gradient is accumulated more in a direction, then step $\frac{\varepsilon}{\sqrt{a+\nu}}$ is less in that direction

If gradient accumulated is less in a direction. Then the step $\frac{\varepsilon}{\sqrt{a+\nu}}$ is more in that direction

From plot, we can see that when it traversed from A to B, the gradients accumulated along W_2 direction is more and gradient along W_1 is smaller. Here, the step along W_1 is more and step across W_2 is less

Adagrad converges faster to minima w/o much Zig Zagging

$\|x\|_\infty = \max_i |x_i|$

$LSE(x) = \ln\left(\sum_{i=1}^{n} e^{x_i}\right)$

$\|x\|_\infty \le LSE(x) \le \|x\|_\infty + \ln(n)$

$p = \max_i\{|x_1|, \ldots, |x_n|\}$

$e^p \le e^{|x_1|} + e^{|x_2|} + \cdots + e^{|x_n|}$

$e^p \le \sum_{i=1}^{n} e^{|x_i|}$ (p doesn't appear every time)

$\sum_{i=1}^{n} e^{|x_i|} \le ne^p$

$\ln(e^p) \le \ln\left(\sum_{i=1}^{n} e^{|x_i|}\right) \le \ln(ne^p)$

$p \le \ln\left(\sum_{i=1}^{n} e^{|x_i|}\right) \le \ln(n) + p$

$\|x\|_\infty \le LSE(x) \le \|x\|_\infty + \ln(n)$

$\ln\left(\sum_{i=1}^{n} e^{x_i}\right) \ge \|x\|_\infty$

$\sum_{i=1}^{n} e^{|x_i|} > e^p$

$\ln\left(\sum_{i=1}^{n} e^{|x_i|}\right) > p$

$LSE(x) > \|x\|_\infty$

$\|x\|_1 \le \ln\left(\sum_{i=1}^{n} e^{|x_i|}\right) \le \|x\|_\infty + \ln(n)$

$\|tx\|_\infty \le \ln\left(\sum_{i=1}^{n} e^{|tx_i|}\right) \le \|tx\|_\infty + \ln(n)$

$t\|x\|_\infty \le \ln\left(\sum_{i=1}^{n} e^{|tx_i|}\right) \le t\|x\|_\infty + \ln(n)$

$\|x\|_\infty \le \frac{1}{t}\ln\left(\sum_{i=1}^{n} e^{|tx_i|}\right) \le \|x\|_\infty + \frac{\ln(n)}{t}$

Under what condition on x will

$\ln\left(\sum_{i=1}^{n} e^{|x_i|}\right) = p + \ln(n)$

$\sum_{i=1}^{n} e^{|x_i|} = ne^p$ (every $|x_i| = |x_j|$ for $i \ne j$)

K-nearest neighbors

error = $\frac{\#\text{num wrong}}{\#\text{total}}$

In order to minimize classification error on this test set, we don't necessarily need to choose a value of K that minimizes training, even if sampled from same distribution.

Small value of K ⟶ overfitting

Large value of K ⟶ underfitting | reduces effect of outliers

Use cross validation to choose K.

Activation functions where vanishing gradient is a problem.

- if it has zero gradient somewhere, then there's a problem
- RELU, σ, tanh

MORE ON BATCHNORM

- adds noise/regularization because computed over mini batches, not entire dataset
  each data distribution has a little bit of noise
- batch normalization accelerates training by requiring fewer iterations to converge to a given loss value
- batch norm has learnable parameters
- linear transformation @ test time
- introduces noise to hidden layer's activation because the mean and std are estimated w/ a mini-batch
- use running averages at test time

Adam combines effects of Adagrad & momentum

Learning rate decay may cause sudden drops in loss landscape

WEIGTS LEARNED IN L1 REGULARIZATION WILL BECOME more sparse than the weights learned using L_2 regularization

$$\mathbb{E}[v_t] = (1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}\,\mathbb{E}[g_i]$$

$$= (1-\beta_1)\mathcal{M}\sum_{i=1}^{t}\beta_1^{t-i} = \mathcal{M}(1-\beta_1)\sum_{i=0}^{t-1}\beta_1^{i} = \mathcal{M}(1-\beta_1)\left[\frac{(1-\beta_1^t)}{(1-\beta_1)}\right]$$

$$\mathbb{E}[v_t] = \mathcal{M}(1-\beta_1^t)$$

$$\gamma_i\,\mathbb{E}[v_t] = \frac{\mathcal{M}(1-\beta_1^t)}{1-\beta_1^t} = \mathcal{M}$$

Ensemble methods: if NN's make independent errors, then they are likely to make errors on diff trials.

We only get a trial wrong when several neural networks make the same errors. Hence, low correlation ⟶ better accuracy.

$$v_t = -\varepsilon\sum_{i=1}^{t}\alpha^{t-i}g_i$$

$$\text{Cov}(v_t) = \mathbb{E}[v_t v_t^T]$$

$$= \varepsilon^2\mathbb{E}\left[\sum_{i=1}^{t}\alpha^{t-i}g_i\left[\sum_{j=1}^{t}\alpha^{t-j}g_j\right]^T\right]$$

$$= \varepsilon^2\mathbb{E}\left[\sum_{i=1}^{t}\sum_{j=1}^{t}\alpha^{t-i}\alpha^{t-j}g_i g_i^T\right]$$

$$= \varepsilon^2\mathbb{E}\left[\sum_{i=1}^{t}\alpha^{2(t-i)}\sigma\right] = \varepsilon^2\sigma^2\left(\frac{1-\alpha^{2t}}{1-\alpha^2}\right)I$$

$$\sum_{k=m}^{n} ar^k = \frac{a(r^m - r^{n+1})}{1-r}$$

(d) (5 points) Consider a convolutional neural network (CNN) with N conv-pool layers. Each conv-pool layer uses five filters of size 3×3, stride $= 1$ and padding $= 1$, followed by relu(), followed by 2×2 max pooling with stride 2. The input image is of size $256 \times 256 \times 3$. What is the number of trainable parameters (including biases) in the first conv-pool layer?

- trainable params (filter width x filter height x 3) + 1) x (# of filters)
= 28 x 5 = 140

weight for class$_i$ = total samples / (#num samples in class$_i$ * num_classes)

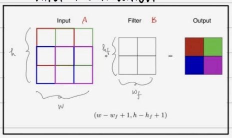
ratio of class samples is 10, pick $\alpha = 1$, $\beta = 10$

$$\frac{\lambda}{n} \sum_{i=1}^{n} \frac{\partial}{\partial W}\left(\max_{1 \le k \le P} \sum_{j=1}^{m} |W_{kj}|\right)$$

$$\frac{\lambda}{n} \sum_{i} |W_{j_1}| + |W_{j2}| + \cdots + |W_{jP}|$$

$$\frac{\partial a}{\partial W^{(\ell)}} = \begin{bmatrix} \frac{\partial a}{\partial W_{11}^{(\ell)}} & \frac{\partial a}{\partial W_{12}^{(\ell)}} & \cdots \\ \frac{\partial a}{\partial W_{21}^{(\ell)}} & & \end{bmatrix}$$

filter / convolution



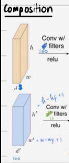
a VALID convolution: A and B fully overlapping

input: h = 32, w = 32, d = 3; filter: h = 5, w = 5, d = 3 } same depth, 3D → (w - wf + 1, h - hf + 1)

note: (5 x 5 x 3) + b = 75 + 1 = 76 params

each output matrix is a slice

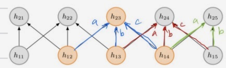
Composition



each layer has multiple filters, let us call that number nF, each of which has its own slice

outputs put together into tensor: (w - wf + 1, h - hf + 1, nF) → passed into ReLU → h_1, passed on

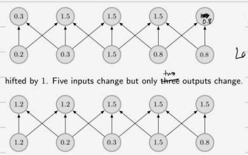
sparsely connected layers
- 1 output neuron only connected to values inside the convolution
- e.g above, each neuron only connected to 5 x 5 x 3
- sparse interaction means there should be more layers



same weights
- reduce memory (fewer params, same weights)
- reduce comp time O(mk) instead of O(mn)

eg: 4 filters are (4x4x3), so params = (4x4x3 + 1) x 4 = 196 params

CNN has (32 x 32x3 + 1) x neurons in layer ≈ Millions



use padding: (w - wf + 1 + 2pad, h - hf + 1 + 2pad)

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

use pooling: $\left(\frac{w - w_p}{stride} + 1, \frac{h - h_p}{stride} + 1\right)$ → no additional params, small spatial invariancies

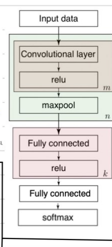
↳ sizing formulas for output size

C1 contains six 5x5 conv filters. stride = 1, pad = 0
Size of ~~feature maps~~ output at C1? 32 - 5 + 1 = 28 ⇒ (28 x 28 x 6)
Number of parameters in C1 layer? (5 x 5 + 1) x 6 = 156 params

S2 is a 2x2 pooling layer applied at stride 2.
Size of ~~feature maps~~ output at S2? (14 x 14 x 6)
Number of parameters in S2 layer? 0

C3 contains sixteen 5x5 x6 conv filters. ← depth of input to this layer
Size of ~~feature maps~~ output at C3? 14 - 5 + 1 = 10 (10 x 10 x 16)
params in C3? (5 x 5 x 6 + 1) x 16

Prof J.C. Kao, UCLA



- sizing: width, height from above, # of filters
- trainable params: ((filter width x filter height x 3) + 1) x # of filters (3 = RGB)
- pooling has 0 parameters

① LeNet 5
- neuron (5x5 + 1)
- # of neurons (28 x 28 x 6) } 122, 304

1. [28x28x6] **CONV**: 6 convolutional filters, 5x5 filter size, applied at stride 1.
2. [14x14x6] **POOL**: 2x2 pool with stride 2. (Adds all elems, multiplies them by trainable coefficient, then passes through sigmoid.)
3. [10x10x16] **CONV:** 16 convolutional filters, 5x5 x6.
4. [5x5x16] **POOL**: 2x2 pool with stride 2.
5. [120] **CONV**: 120 5x5 x16 convolutional filters. (w - wf + 1; 5 - 5 + 1 = 1)
6. [84] **FC**: FC layer: 84 x 120.
7. [10] **OUT:** MSE against a template for each digit.

② AlexNet
- input processing
- used pooling with overlapping
- augmentation
  - extract patches + reflections, ↓ 1.5% error
  - scale PCs of color - ↓ 1% error
  - dropout, p = 0.5

1. [55x55x96] **CONV**: 96 filters of size 11x11x3 with stride 4.
2. [27x27x96] **POOL**: 3x3 filters with stride 2.
3. [27x27x96] **NORM**: normalization layer
4. [27x27x256] **CONV**: 256 filters of size 5x5x96 with stride 1, pad 2. ($w_{out} = w_{in} - 5 + 2 \cdot 2 + 1 = w_{in}$)
5. [13x13x256] **POOL**: 3x3 filters with stride 2.
6. [13x13x256] **NORM**: normalization layer
7. [13x13x384] **CONV**: 384 filters of size 3x3 at stride 1, pad 1.
8. [13x13x384] **CONV**: 384 filters of size 3x3 at stride 1, pad 1.
9. [13x13x256] **CONV**: 256 filters of size 3x3 at stride 1, pad 1. (ZF: 512, 1024, 512)
10. [6x6x256] **POOL**: 3x3 filters at stride 2.
11. [4096] **FC**: Fully connected layer with 4096 units
12. [4096] **FC**: Fully connected layer with 4096 units
13. [1000] **FC**: Fully connected layer with 1000 units (class scores).
14. [1000] **OUT**: Softmax layer

③ ZFnet with different hyperparameters
- e.g: 7x7 had more freq coverage than 11x11
- they increased filters: 384 to 512/1024
- dropped from 16.4% to 11%

④ VGGnet
- CONV uniform: 3x3, stride 1, pad 1
- POOL uniform: 2x2 max, stride 2 } 11.7% → 7.3%
- smaller fields: need more layers, less params, but more non linearity
  mem intensive ← but (Win-4, hin-4) vs 3x (Win, hin), more than 3x activations
- 138M params, concentrated in FC layers (122M)

⑤ GoogleNet: deeper, Inception module, no FC, 5M params
- previous layers → 1x1, 3x3, 5x5, max pool 2x3 → concatenate → next layer