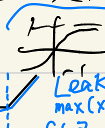
$\sigma(x) = \frac{1}{1+\exp(-x)}$ — Saturate grad, kill neuron

$\frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$

tanh

Relu



Leaky ReLu: $\max(x, 0.1x)$

Backprop

cuz ReLu: dead neurons with 0 grad

Q: 64×64 RGB to binary categories need $256^{3\times 64\times 64}$ bits.

He Initialization $(2/n^{[l-1]})$: ReLu

Xavier Initialization: tanh/sigmoid

$-W_{i,j}^{[l]} = N(0, \frac{1}{n^{[l-1]}})$

- assumes tanh activ.
- reduces vanishing/exploding gradients
- Xavier: $1/n_{prev}$
- He: $2/n_{prev}$
- Glorot: $2/(n_{prev}+n_{cur})$
- Derivation of Xavier:

$Var(a_i^{[l-1]}) = Var(a_i^{[l]})$

$\approx Var[z_i^{[l]}]$

$= Var(\sum_{j=1}^{n^{[l-1]}} w_{ij}^{[l]} a_j^{[l-1]})$

$= \sum_{j=1}^{n^{[l-1]}} Var(w_{ij}^{[l]} a_j^{[l-1]})$

$= \sum_{j=1}^{n^{[l-1]}} E(w_{ij}^{[l]})^2 Var(a_j^{[l-1]})$

$+ E[a_j^{[l-1]}]^2 Var(w_{ij}^{[l]})^2$

$+ Var(w_{ij}^{[l]}) Var(a_j^{[l-1]})$

$= n^{[l-1]} Var(w_{ij}^{[l]}) Var(a_j^{[l-1]})$

$\Rightarrow Var(W) = \frac{1}{n^{[l-1]}}$

Uniform $(-a, a) \sim U(0, \frac{a^2}{3})$

$(a,b) \sim U(\frac{a+b}{2}, \frac{1}{12}(b-a)^2)$

Optimization: ① GD (+d·GD)

② Momentum $V_{dw} = \beta V_{dw} + (1-\beta)dW$

$W = W - \alpha V_{dw}$

③ RMSProp $S_{dw} = \beta S_{dw} + (1-\beta)dW^2$

$W = W - \alpha \frac{dW}{\sqrt{S_{dw}}+\varepsilon}$

④ Adam $V_{dw} = \beta_1 V_{dw} + (1-\beta_1)dW$

$S_{dw} = \beta_2 S_{dw} + (1-\beta_2)dW^2$

$V_{corr\,dw} = V_{dw}/(1-\beta_1)^t$

$S_{corr\,dw} = S_{dw}/(1-\beta_2)^t$

$W = W - \alpha \frac{V_{corr\,dw}}{\sqrt{S_{corr\,dw}}+\varepsilon}$

Same for updating b!

Hyperparams:

① Batch size
② learning rate
③ learning rate decay

Mini batch/SGD better at avoiding saddle points

L_1 & L_2 Reg

$J_{L_1}(\vec{W}) = J(\vec{W}) + \lambda \sum_{i=1}^{l} |w_i|$

$J_{L_2}(\vec{W}) = J(\vec{W}) + \lambda \sum_{i=1}^{l} ||w_i||^2$

Update:

$w_i^{k+1} = w_i^k - \alpha\lambda\, sign(w_i) - \alpha \frac{\partial J}{\partial w_i}$

$w_i^{k+1} = \underbrace{w_i^k - 2\alpha\lambda w_i}_{decay} - \alpha \frac{\partial J}{\partial w_i}$

Goal: limit weight growth (weight decay)

L1 encourages sparsity

Dropout: "Inverted Dropout" ÷ keep-prob

- drop $a^{[l]}$ after activation

Test: switch off Dropout

Early Stopping: non-orthogonal approach to mitigate overfitting

Data augmentation:

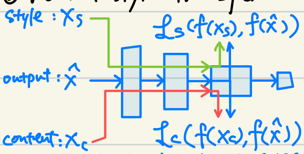
- crop, flip, rotate, blur, change color

Python Numpy/Tensorflow

- np.var, np.mean, np.exp, np.sqrt
- matrix operations:
  a.dot(b) = np.dot(a, b)
- np.empty_like(x): creates empty matrix w/ x's shape
- broadcasting

Algo and case studies:

① Neural Style Transfer



- NN: pre-trained network (VGG) for feature extraction

$-\mathcal{L}_c = \sum_{l\in\{l_c\}} ||F^l(I_c) - F^l(I)||^2$

$\mathcal{L}_s = \sum_{l\in\{l_s\}} ||G(F^l(I_s)') - G(F^l(I)')||^2$

- $\mathcal{L}_c$: content loss, tensor L^2 norm loss, F: intermediate features — extract content

$\mathcal{L}_s$: style loss, G: graham matrix for style extraction

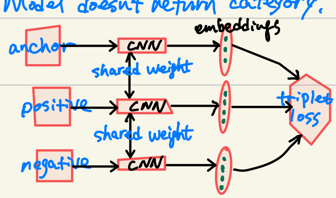
Face recognition/Triplet loss

X_A (anchor) X_P (pos) X_N (neg)

$\mathcal{L} = [||f(X_A) - f(X_P)|| - ||f(X_A) - f(X_N)|| + m]_+$

When $\mathcal{L}$ is minimized, $X_A \approx X_P$, X_A <<>> X_N, m is margin

- how to decide if should let car enter? run thru NN and get embedding, compare to other embeddings in your database. Model doesn't return category.



CNN

- 3D: • learnable params: $(f\times f\times n_c + bias) \times n_f$ (+1)
  • output shape: $(n-f+1)^2 \times n_f$
- "Valid": no padding
  "Same": pad s.t. output size = input size
- General formula for shap:

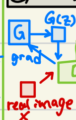
$\lfloor \frac{n-f+2p}{s} + 1 \rfloor^2_{Floor} \times n_f$

- Pooling layers:
  • no padding (normally) • usually: f=2, S=2
  • increases output sensitivity to position of image.
- Batchnorm layers
  • mini batch norm
  • apply before activation (on $z_i^{[l]}$)
  • Test time: use running avg of mean/var
  • $\mu = \frac{1}{m}\sum_i^m z^{(i)}$, $\sigma^2 = \frac{1}{m}\sum_i^m (z^{(i)}-\mu)^2$

$z_{norm} = \frac{z^{(i)}-\mu}{\sqrt{\sigma^2+\varepsilon}}$, $\tilde{z} = \gamma z_{norm}^{(i)} + \beta$

  • γ and β are learnable parameters
  • w/o γ and β: restrict network from learning larger values
  • why does batch norm work?
    - faster learning: each dim take similar values
    - mitigate covariance shift (make weights in later layers more robust to weight changes in earlier layers)
    - slight regularization effect: mean and var of mini-batch adds noise

GANs: $\min_{\theta_g} \max_{\theta_d} [E_x \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data } x} + E_z \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator for fake data } G(z)})]$



- θ_d wants to maximize objective s.t. D(x) is close to 1 (real) and $D(G(z)) \approx 0$ (fake)
- θ_g wants to minimize objective s.t. D(Gx) close to 1 (discriminator fooled into thinking G(z) is re

Denfense: ① safety net: network discerning fakes

② generate adversial examples and label correctly

③ adversial training: $\mathcal{L}_{new} = \mathcal{L}(w,b,x,y) + \lambda\mathcal{L}(w,b,x_{adv},y)$

④ logit pairing: $\mathcal{L}_{new} = \mathcal{L}(w,b,x,y) + \lambda||f(x|w,b) - f(x_{adv}|w,b)||_2^2$

Alternative cost function:

$J^{(D)} = \frac{1}{m_{real}} \sum_{i=1}^{m_{real}} y_{real}^{(i)} \log(D(x^{(i)})) - \frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} (1-y_{gen}^{(i)}) \log(1 - D(G(z^{(i)})))$

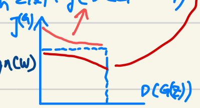
$J^{(G)} = -J^{(D)} = \frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} \log(1 - D(G(z^{(i)})))$ — saturating cost

Label: $y_{real} = 1$, $y_{gen} = 0$

non-saturating cost of $J^{(G)} = -\frac{1}{m}\sum_{i=1}^m \log(D(G(z^{(i)})))$

Methods of generating Adversarial examples:

- Fast gradient sign: $x^* = x + \varepsilon\, sign(w)$
- Input optimization



Loss functions:

- logistic:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{m}\sum_{j=1}^{n}|\theta_j| + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

- if output is probability, use logistic/softmax + cross entropy for loss.

$\log(\cdot)$ / $\log(1-\cdot)$ input $\in (0,1)$

- tanh has vanishing grad
- ReLu has dead neurons.

Softmax: $S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$

loss: $\sum_{i=1}^{n_y} -(y_i \log \hat{y}_i)$

Trigger Word Detection + Data acquisition

- need overfit / bias
- transition invariant
- covariante shift
- go over adam again
- backprop derivation

Recall: $\frac{TP}{TP+FN}$ ... Predicted

Precision: $\frac{TP}{TP+FP}$ ... Actual

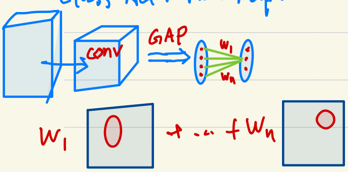
Accuracy: $\frac{TP+TN}{Total}$

F_1-Score: $2\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$

Saliency map: $\frac{\partial \hat{y}}{\partial x}$

Occlusion sensitivity: confidence of pred w/o area

Class Activation map:



Forward Prop:

$Z = W_1 x + b$, $A = \sigma(Z)$

$\hat{y} = W_2 A + b_2$

$L = \frac{1}{m}\|\hat{y} - y\|^2$

Back prop:

$$\frac{\partial L}{\partial W_2} = \frac{2}{m}(\hat{y}-y)A^T$$

$$\frac{\partial L}{\partial b_2} = \frac{2}{m}(\hat{y}-y)\vec{1}$$

$$\frac{\partial L}{\partial W_1} = \left(W_2^T\frac{2}{m}(\hat{y}-y)\odot A\odot(1-A)\right)x^T$$

$$\frac{\partial L}{\partial b_1} = \left(W_2^T\frac{2}{m}(\hat{y}-y)\odot A\odot(1-A)\right)\vec{1}$$