ECE 598NSG/498NSU
Deep Learning in Hardware
Fall 2020

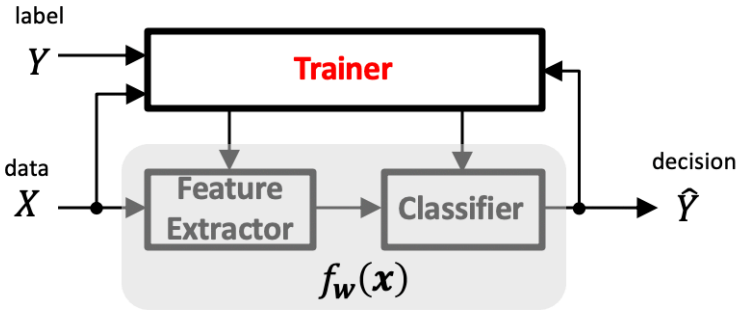
Training DNNs – SGD, Backprop and its Variants

Naresh Shanbhag
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign

http://shanbhag.ece.uiuc.edu

Today

- Overview of DNN training – the big picture
- The Stochastic Gradient Descent (SGD) training algorithm
- The back-prop algorithm

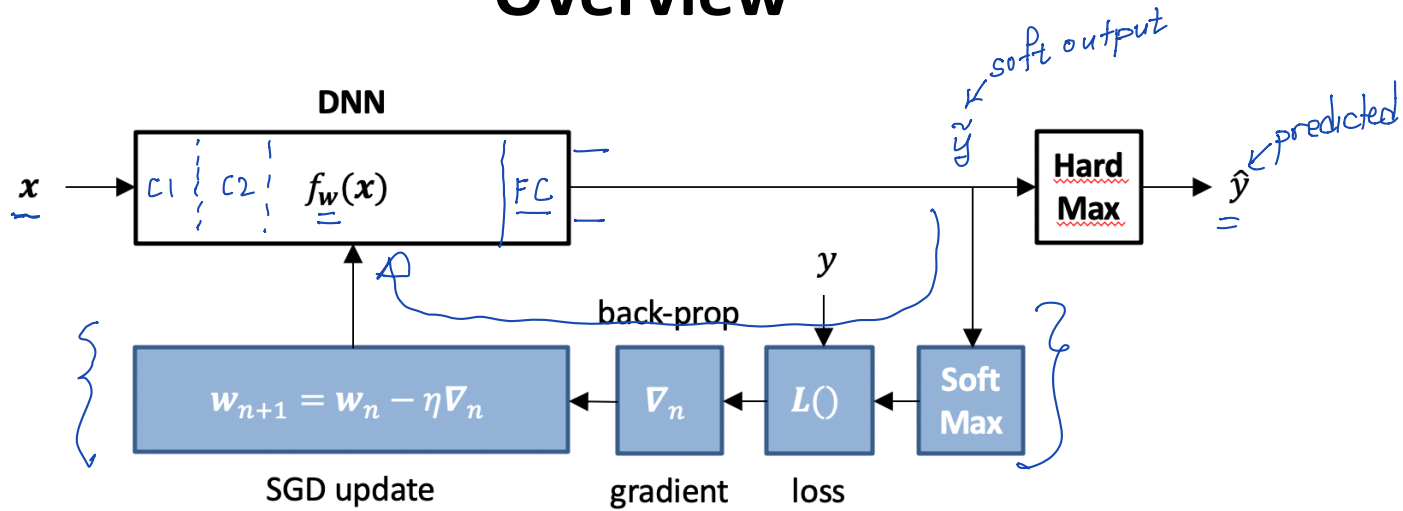


Supervised Training

- Sample: $z = (\boldsymbol{x}, y) = (\text{data}, \text{label})$;
- Training sample (example): samples used in training.
- Loss function: $L(\hat{y} = f_{\boldsymbol{w}}(\boldsymbol{x}), y)$; sample-specific value
- Family of functions parametrized by $\boldsymbol{w}$: $\mathcal{F}$: $f_{\boldsymbol{w}}(\boldsymbol{x}) = \hat{y}$
- Loss function evaluated on the training set: $Q(\boldsymbol{w})$
- learning (convergence) curves; learning rate; stability

DNN Training Set-up

Overview



- Forward pass – generates predicted label $\hat{y}$
- Backward pass – SGD-based back-prop to update weights

$f_w(x)$: multi-stage & non-linear.

Q: How to find w: we get accurate predictions?

Q: What is a measure of accuracy? → depends on the task.

Task → classification

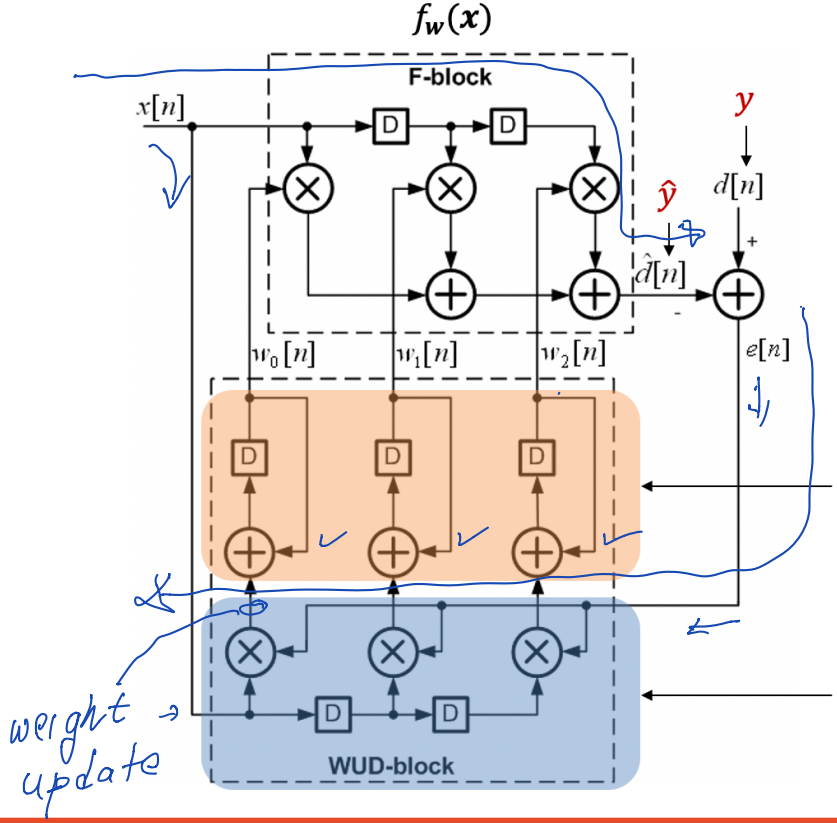
$P_e = Pr\{\hat{y} \neq y\}$: misclassification or error rate

(y: true label; $\hat{y}$: predicted label)

$1 - P_e$: accuracy

minimize P_e? How do we adjust w: P_e is minimized? → $\hat{y} \to y$

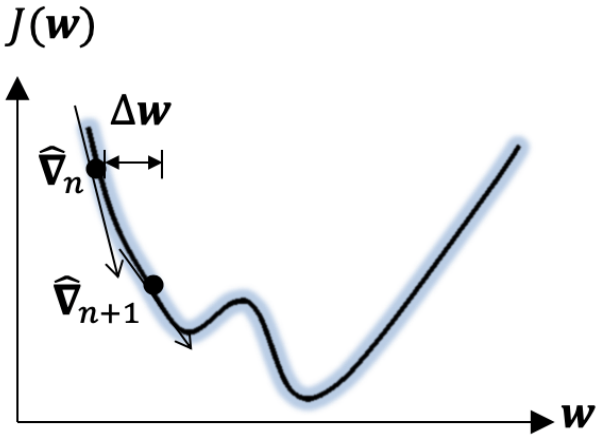
LMS via DNN Lens

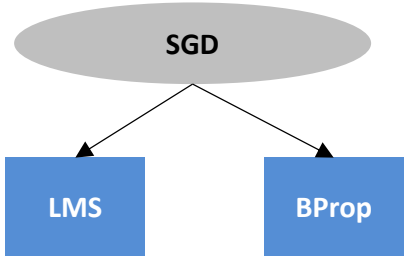


$$E[e^2[n]] = J(w)$$

$$\frac{\partial J(w)}{\partial w} \rightarrow 2e[n]x[n]$$

Stochastic Gradient Descent





- Calculate instantaneous gradient of loss function; update weight parameter by adding a small increment in the negative gradient

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \mu(-\hat{\boldsymbol{\nabla}}_n)$$

-ve of gradient of the loss fn. wrt. $\underline{w}$.

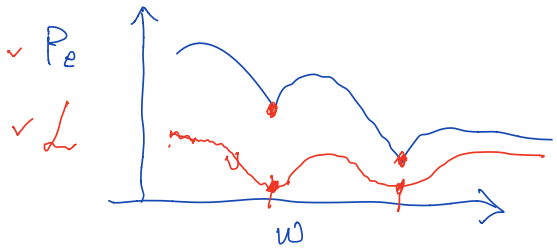
- $\hat{\nabla}_n = \dfrac{\partial \hat{J}(\boldsymbol{w}_n)}{\partial \boldsymbol{w}_n} \Rightarrow$ is the gradient of the *instantaneous loss function* wrt $\boldsymbol{w}_n$
- μ: step-size or learning rate

minimize P_e : directly is hard $\because$ $P_e(w)$

$$\frac{\partial P_e(w)}{\partial w} \longrightarrow \quad w_{n+1} = w_n + \mu\left(-\frac{\partial P_e(w)}{\partial w}\right)$$

Loss function : $\mathcal{L}(f_w(x), y) = \mathcal{L}(\hat{y}, y)$

↳ $\mathcal{L}$ is a surrgate/proxy for P_e



LMS: $MSE = \mathbb{E}\left[(\hat{y}-y)^2\right] \longrightarrow y, \hat{y} \in \mathbb{R}$

For DNNs : $\mathcal{L}$ is "cross entropy loss"

Entropy: of a RV X (discrete RV) → probability mass function $P_X(x)$

$$X \in \{-1, +1\} \rightarrow P_X(-1) = \frac{1}{2} \;;\; P_X(+1) = \frac{1}{2}$$

($-1 = x_1$, $+1 = x_2$)

Entropy of X : $H(X) = -\sum_i P_X(x_i) \log_2 P_X(x_i)$

$$H(X) = \sum_i P_X(x_i) \log_2\left(\frac{1}{P_X(x_i)}\right) = \mathbb{E}_{P_X}\left[\log_2\left(\frac{1}{P_X(x_i)}\right)\right]$$

↳ min. # of bits on average needed to represent X.

# of bits needed to rep. x_i

Cross-entropy: $X, Y \in \{-1, +1\}$, $X \sim P_X$, $Y \sim Q_X$

$$H(X,Y) = -\sum_i P_X(x_i) \log_2 (Q_X(x_i))$$

$$= +\sum_i P_X(x_i) \log_2 \left(\frac{1}{Q_X(x_i)}\right)$$

bits assigned

average # of bits needed to represent X when we assume Q_X as its distribution.

$H(X,Y) \geq H(X)$

In DNNs: $P_X \rightarrow$ true distribution of labels

$Q_X \rightarrow$ distribution generated network

Update $w \rightarrow$ nudge Q_X towards P_X, $\rightarrow$ "gap" or "distance" between Q_X & P_X is minimized.

↓

Kullback–Leibler divergence

$$D_{KL}(P_X \| Q_X) = \mathbb{E}_{P_X}\left(\log_2 \frac{P_X}{Q_X}\right)$$

$$= \sum_i P_X(x_i) \log_2 \frac{P_X(x_i)}{Q_X(x_i)}$$

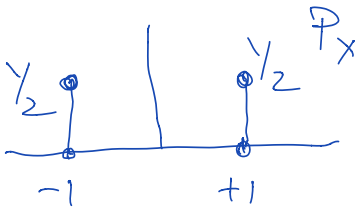
$$= \sum_i P_X \log_2 P_X - \sum_i P_X \log_2 Q_X = -H(X) + H(X,Y)$$

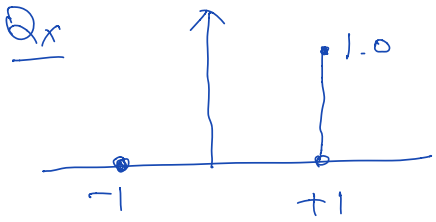
$$D_{KL}(P_x \| Q_x) = -H(X) + H(X,Y)$$

$$\min D_{KL}(P_x \| Q_x) = \min H(X,Y) \rightarrow \text{minimize CE loss.}$$

$$H(X,Y) \geq H(X) \rightarrow D_{KL}(P_x \| Q_x) \geq 0$$

Example: $x \in \{-1, +1\}$



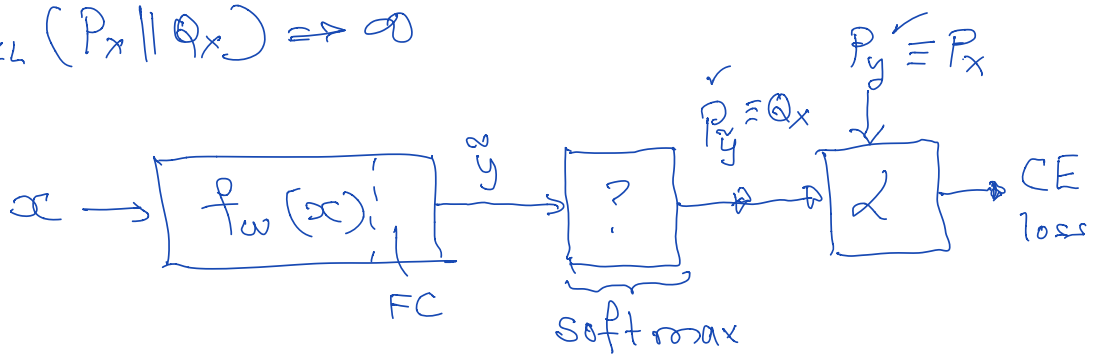


$$H(X) = -\left(\frac{1}{2} \times \log_2\left(\frac{1}{2}\right) + \frac{1}{2} \times \log_2\left(\frac{1}{2}\right)\right) = 1$$

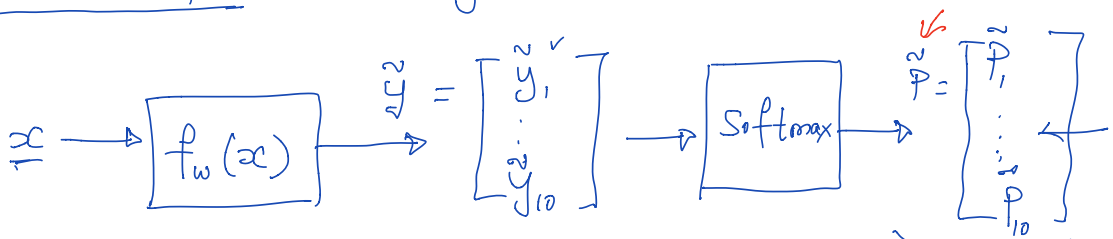
$$H(X,Y) = -\left(\frac{1}{2} \times \log_2(0) + \frac{1}{2} \times \log_2(1)\right) \rightarrow \infty$$

($\log_2(0) \rightarrow -\infty$, $\log_2(1) \rightarrow 0$)

$$D_{KL}(P_x \| Q_x) \Rightarrow \infty$$



DNN Example: 10-way classifier



$\hat{y} \leftarrow \max(\tilde{y})$

$$\sum_i \tilde{p}_i = 1$$

$$\tilde{p}_i = \frac{e^{\tilde{y}_i}}{\sum_i e^{\tilde{y}_i}}$$ → softmax → generates the predicted distribution of class labels.

$p = [0, 1, 0, \dots]$ — one-hot encoded vector

class 2 is true label.

$$\mathcal{L}(\hat{y}, y) = -\log_2 \tilde{p}_i \quad : i \leftarrow \text{true class}$$

$$= -\sum_i p_i \log_2 \tilde{p}_i$$

(p_i: 1-hot)

$$\frac{\partial(-\log_2 \tilde{p}_i)}{\partial \tilde{p}_i} = -\frac{1}{\tilde{p}_i \ln 2}$$

Ex: binary classifier

$\tilde{y}_1 = w_1 x_1 + w_2 x_2$; $\tilde{y}_2 = w_3 x_1 + w_4 x_2$

True class = 1

$\mathcal{L}(\hat{y}, y) = -\log_2 \tilde{p}_1$ ←

$\tilde{p}_1 = \frac{e^{\tilde{y}_1}}{e^{\tilde{y}_1} + e^{\tilde{y}_2}}$ ← soft max ; $\tilde{p}_2 = \frac{e^{\tilde{y}_2}}{e^{\tilde{y}_1} + e^{\tilde{y}_2}}$

$\left(\frac{\partial \mathcal{L}()}{\partial w_1}\right) = ? =$ — using chain rule

$\sum_i \tilde{p}_i = 1$; $\frac{\tilde{p}_1}{\tilde{p}_2} = e^{\tilde{y}_1 - \tilde{y}_2}$

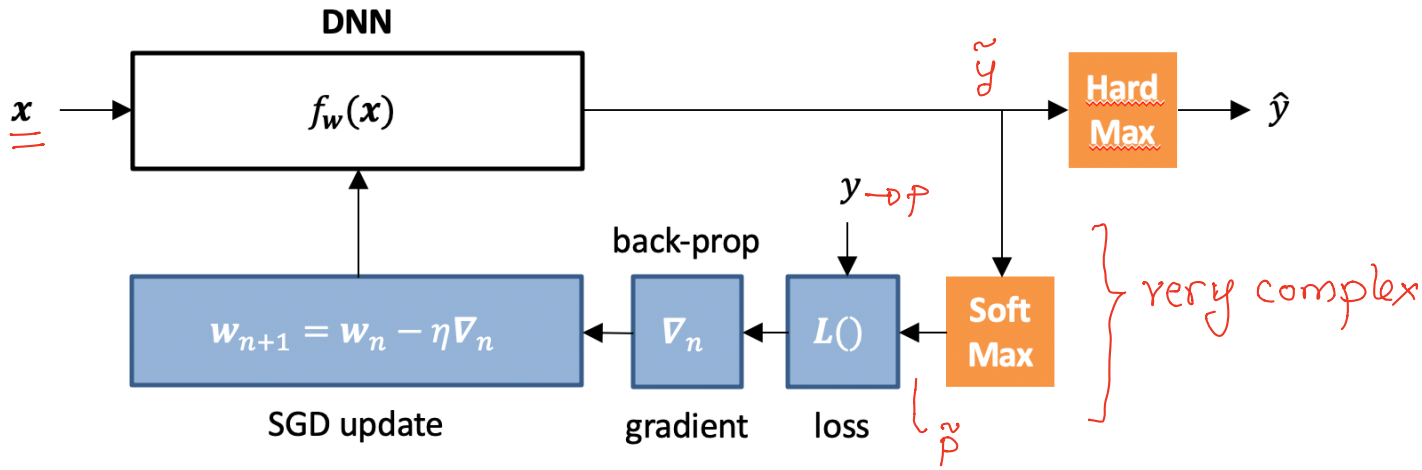
$\frac{\partial(-\log_2 \tilde{p}_1)}{\partial w_1} = ? = -\frac{1}{\tilde{p}_1 \ln(2)} \times \frac{\partial \tilde{p}_1}{\partial w_1} = -\frac{1}{\tilde{p}_1 \ln(2)} \left[\frac{\partial \tilde{p}_1}{\partial \tilde{y}_1} \times \underbrace{\frac{\partial \tilde{y}_1}{\partial w_1}}_{x_1}\right]$

$= -\frac{\tilde{p}_2 x_1}{\ln(2)}$

$\tilde{p}_1 \tilde{p}_2 = \frac{e^{\tilde{y}_1} \times e^{\tilde{y}_2}}{(e^{\tilde{y}_1} + e^{\tilde{y}_2})^2}$

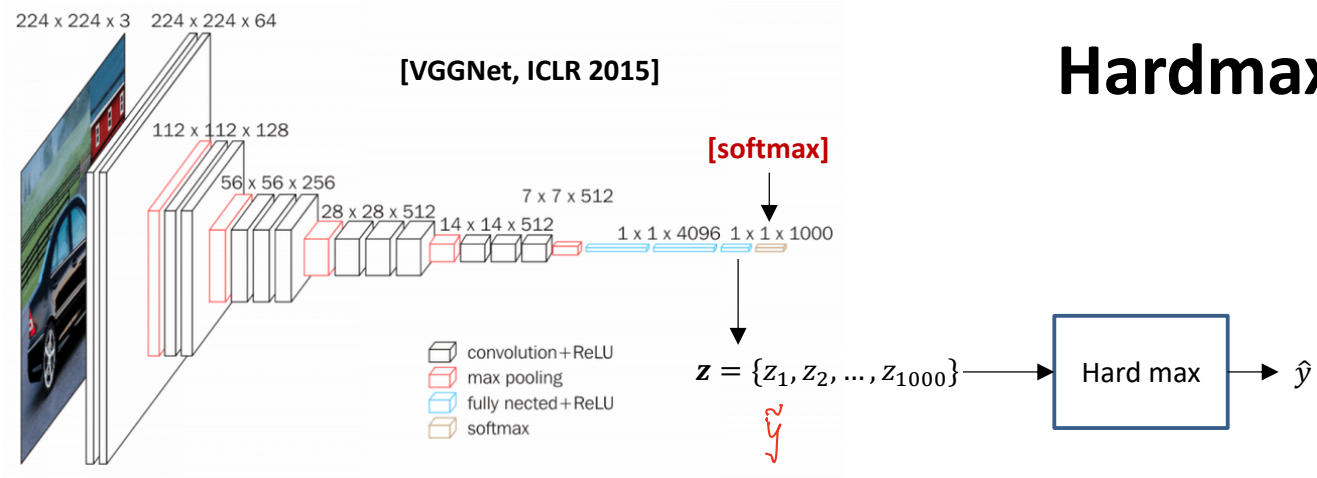
SGD update : $w_1 \leftarrow w_1 + \mu(-\nabla_{w_1}) = w_1 + \frac{\mu \tilde{p}_2 x_1}{\ln(2)}$

Overview



- Forward pass – generates predicted label $\hat{y}$
- Backward pass – SGD-based back-prop to update weights

Hardmax

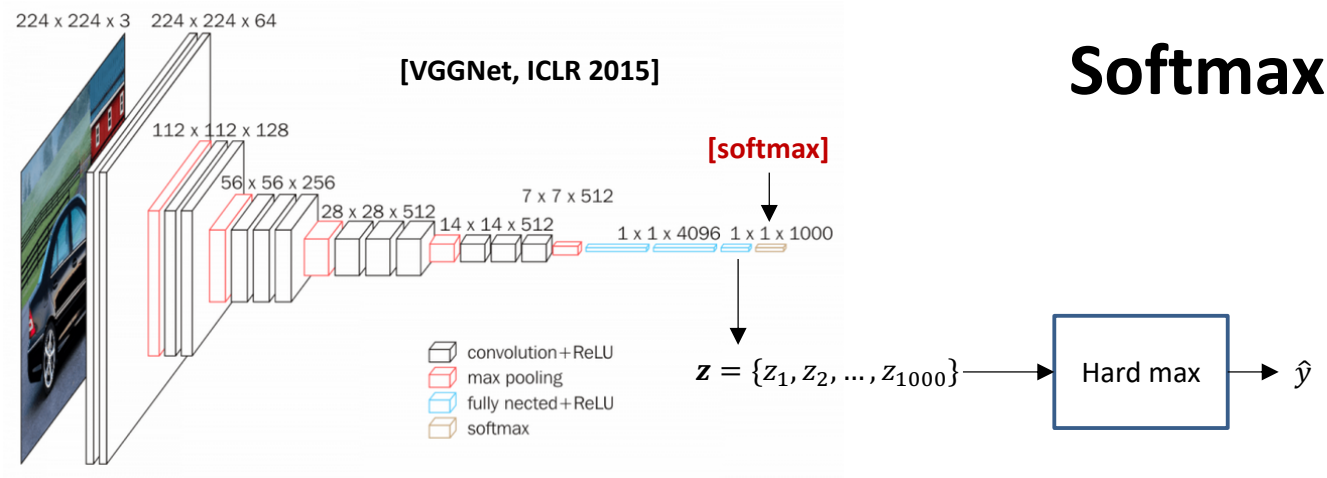


- N-way hard max: outputs are final decisions (Top-1, Top-5):

$$\hat{y} = \max(z_1, z_2, \ldots, z_N) \quad \text{or} \quad \hat{y} = \max(\sigma_1, \sigma_2, \ldots, \sigma_N)$$

- z_i: last FC layer outputs; σ_i: softmax outputs;

Softmax



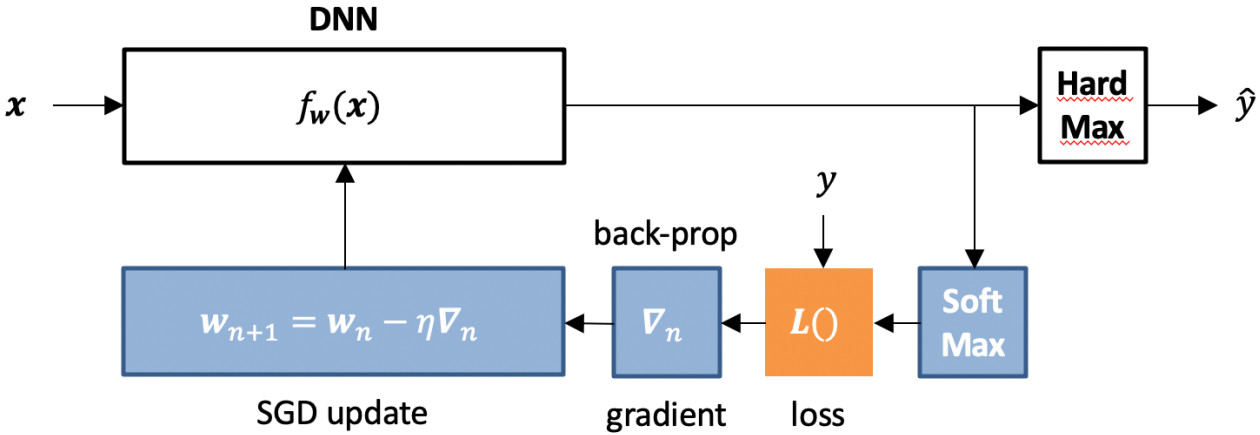
- N-way soft max outputs estimated class probabilities: (used for loss function computation)

$$\sigma_i = \frac{e^{z_i}}{\sum_{k=1}^{N} e^{z_k}}$$

- E.g.: binary case ($N = 2$):

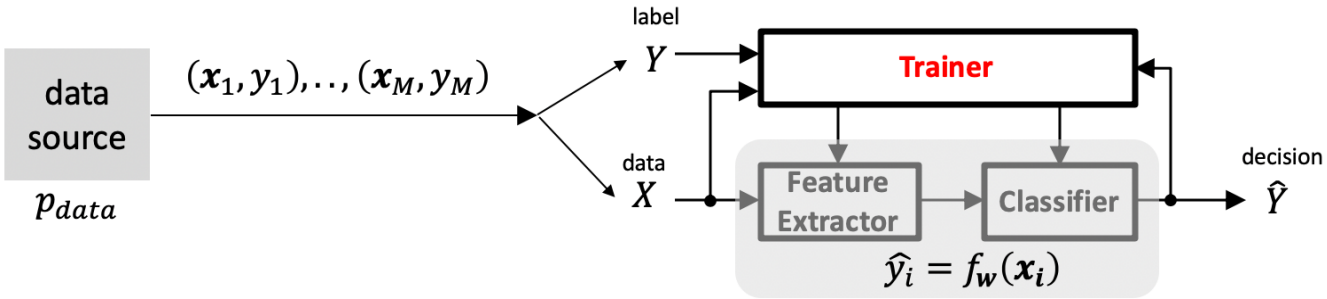
$$\sigma_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2}}; \qquad \sigma_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2}}$$

Overview



- Forward pass – generates predicted label $\hat{y}$
- Backward pass – SGD-based back-prop to update weights

Loss Function

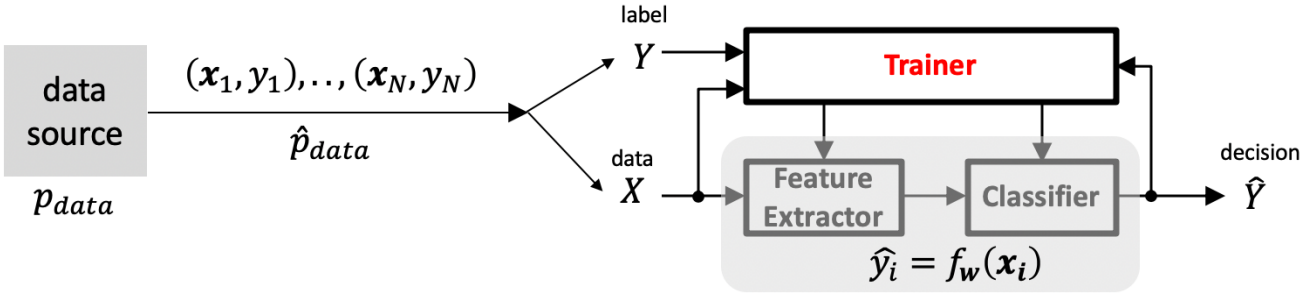


- measures generalization performance

$$J(\boldsymbol{w}) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[L(f_{\boldsymbol{w}}(\boldsymbol{x}), y)] = \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[L(\hat{y}, y)]$$

expectation is over the (unknown) data generating distribution p_{data}.

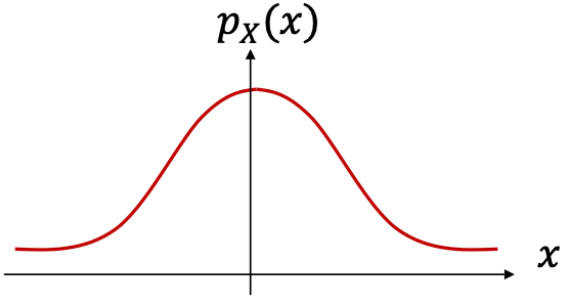
(Empirical) Loss Function



- True data distribution is unknown → empirical (sample) estimate of loss
- sample averaged loss over the training set.

$$\hat{J}(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N} L(f_{\boldsymbol{w}}(\boldsymbol{x}_i), y_i) = \mathbb{E}_{\boldsymbol{x}\sim\hat{p}_{data}}[L(f_{\boldsymbol{w}}(\boldsymbol{x}), y)]$$

Shannon Entropy



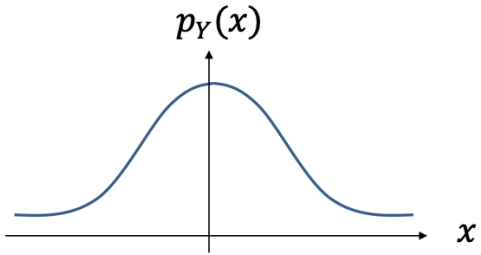
- Quantifies the uncertainty (information content) in a RV x

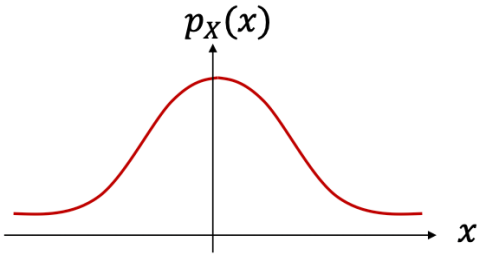
$$H(X) = -\sum_x p_X(x) \log_2\big(p_X(x)\big) = -\mathbb{E}_{\sim p_x}\left[\log_2\big(p_X(x)\big)\right] \quad \text{(in bits)}$$

- If $X \in \mathcal{F}$ is a uniformly distributed discrete RV then

$$H(X) = \log_2 |\mathcal{F}| \text{ bits}$$

Kullback-Leibler Divergence





- Distance between two distributions $p_X(x)$ & $p_Y(x)$ defined over the same RV x:

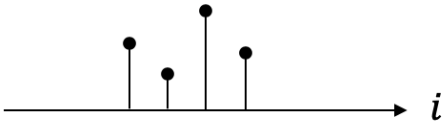
$$D_{KL}(p_X||p_y) = \mathbb{E}_{\sim p_X}\left[\log_2\left(\frac{p_X(x)}{p_Y(x)}\right)\right]$$
$$= \mathbb{E}_{\sim p_X}[\log_2(p_X(x))] - \mathbb{E}_{\sim p_X}[\log_2(p_Y(x))] = -H(X) + H(X,Y)$$

cross-entropy

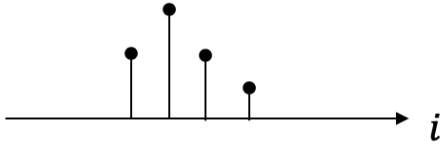
- $D_{KL}(p_X||p_y) = 0$ iff $p_X(x) = p_Y(x)$
- can be used to compare two probability distributions

Cross-entropy (Loss)

$p_Y(i)$ (class label)



$\sigma(i)$



- Cross-entropy: $H(X,Y) = -\sum_x p_X(x) \log_2\big(p_Y(x)\big)$
- minimizing $D_{KL}\big(p_X||p_y\big)$ → minimizing $H(X,Y)$
- Cross-entropy loss → differential function of its parameters

$$J(\boldsymbol{w}) = -\sum_i y_i \log_2(\sigma_i)$$

$\sigma_i = \frac{e^{z_i}}{\sum_{k=1}^{N} e^{z_k}}$; y_i: 1-hot encoded true class labels

Loss Function Example - MSE

- $L(f_{\boldsymbol{w}}(\boldsymbol{x}_i), y_i) = L(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2$
- true average loss:

$$J(\boldsymbol{w}) = \mathbb{E}_{\boldsymbol{x} \sim p_{data}}[L(f_{\boldsymbol{w}}(\boldsymbol{x}), y)]$$
$$= \sum_{\boldsymbol{x}} (f_{\boldsymbol{w}}(\boldsymbol{x}) - y)^2 p_{data}(\boldsymbol{x}, y)$$

- sample average loss

$$\hat{J}(\boldsymbol{w}) = \mathbb{E}_{\boldsymbol{x} \sim \hat{p}_{data}}[L(f_{\boldsymbol{w}}(\boldsymbol{x}), y)]$$
$$= \frac{1}{N} \sum_{i=1}^{N} (f_{\boldsymbol{w}}(\boldsymbol{x}) - y_i)^2$$

Regularization

- DNNs are over-parameterized → many solutions possible
- regularization guides the choice of a certain type of solution → intended to reduce generalization error but not training error
- includes additional constraints on parameter values
- many ways to regularize – norm-based, drop-out, soft weight sharing….
- Example: L_2 regularization (weight decay)

$$\min \quad \tilde{J}(\boldsymbol{w}) = \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} + J(\boldsymbol{w};\boldsymbol{x},y)$$

$$\boldsymbol{w}_{n+1} = (1-\eta\alpha)\boldsymbol{w}_n - \eta\nabla_{\boldsymbol{w}}J(\boldsymbol{w};\boldsymbol{x},y)$$

History

- SGD applied to a DNN (multi-layer network) is called **back-propagation** algorithm
- Based on the application of chain rule and SGD
- First proposed in Paul Werbos's doctoral dissertation [1974]

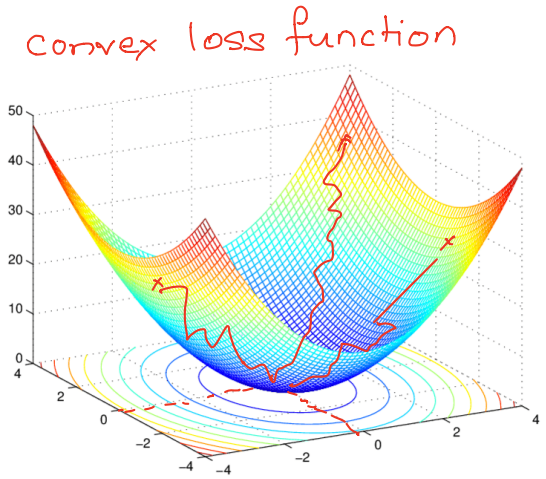
P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Committee on Appl. Math., Harvard Univ., Cambridge, MA, Nov. 1974.

- Rediscovered and popularized by Rumelhart, Hinton, and Williams [1986]

D. Rumelhart, D. Hinton, and G. Williams, "Learning internal representations by error propagation," in D. Rumelhart and F. McClelland, eds., *Parallel Distributed Processing, Vol. 1.* Cambridge, MA: M.I.T. Press, 1986.

Loss Landscape of DNNs

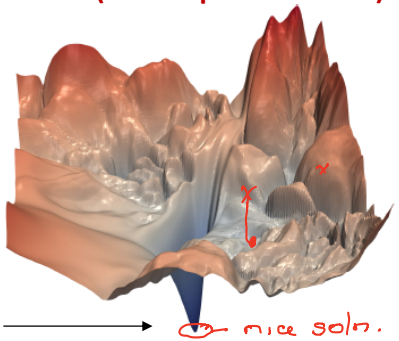
Linear regressor



Visualizing the Loss Landscape of Neural Nets

Hao Li[1], Zheng Xu[1], Gavin Taylor[2], Christoph Studer[3], Tom Goldstein[1]
[1]University of Maryland, College Park [2]United States Naval Academy [3]Cornell University
{haoli,xuzh,tomg}@cs.umd.edu, taylor@usna.edu, studer@cornell.edu

ResNet-56 (w.o. skip connections)



How to find this? →

- DNNs have non-convex loss function → loss landscape has many peaks and valleys

drone → $X_i = (\boldsymbol{x}_i, y_i)$: example

$\boldsymbol{x}_i$ y_i

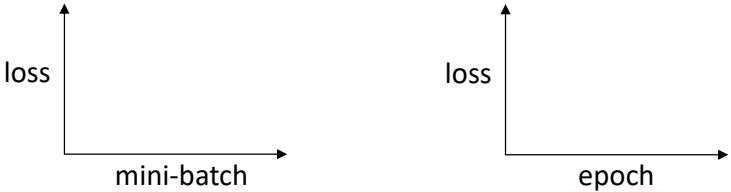
$X_1, X_2, \ldots, X_M$ → training set

random shuffle

$X'_1, X'_2, \ldots, X'_M$

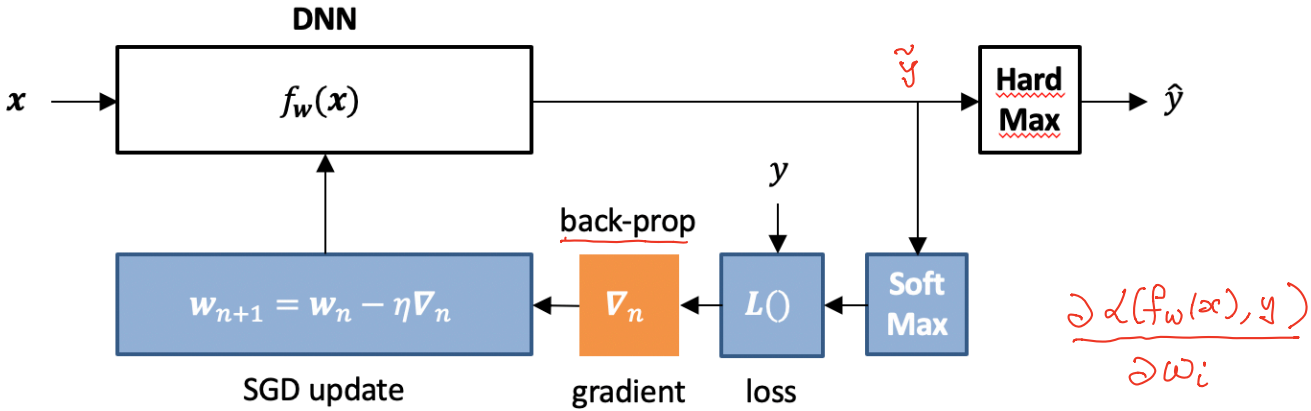
$[X'_1, X'_2, \ldots, X'_B]$ $[X'_{B+1}, X'_{B+2}, \ldots, X'_{2B}]$..$[X'_{M-B-1}, X'_{M-B}, \ldots, X'_M]$ epoch

mini-batch mini-batch mini-batch



The Back Propagation Algorithm

Overview



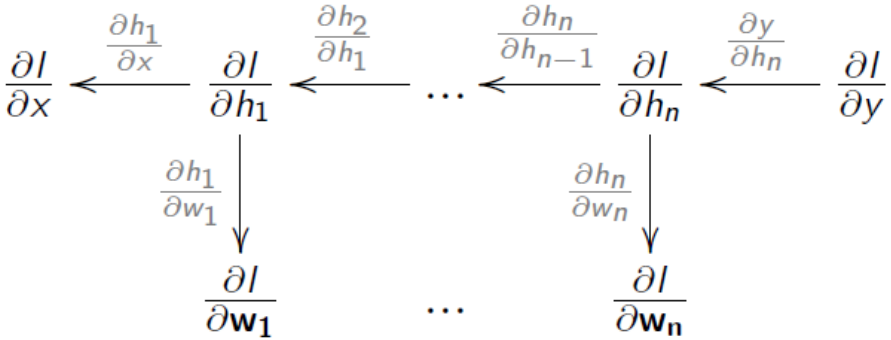
- Forward pass – generates predicted label $\hat{y}$
- Backward pass – SGD-based back-prop to update weights

Backpropagation

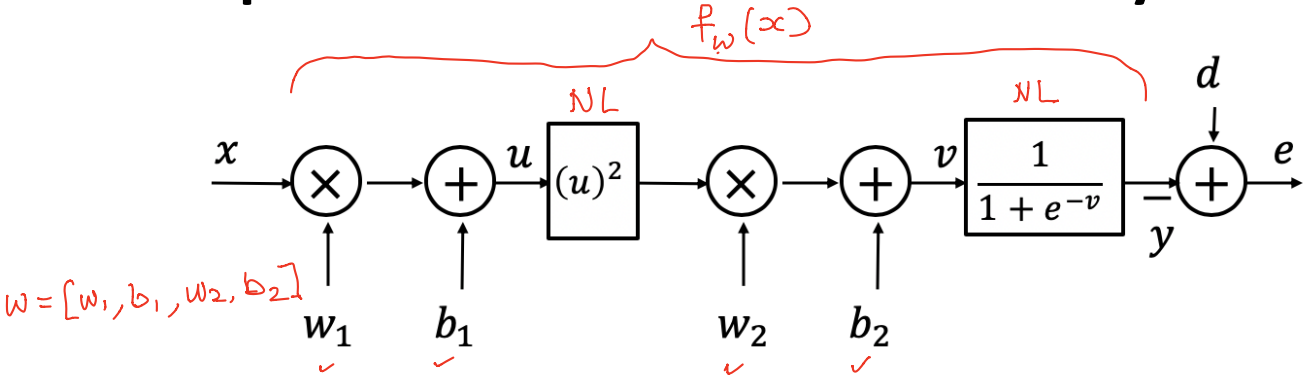
- determine $\mathbf{w}$ that minimizes loss function (empirical risk):

$$\frac{1}{n}\sum_{1}^{n} l(h(\mathbf{x}_i;\mathbf{w}), y_i)$$

- backpropagation = use of chain rule



Optimization with a Non-Linear System



- much like in LMS, we would like to use SGD in order to learn values of w_1, b_1, w_2, b_2 to minimize $\boldsymbol{E}[e^2]$
- need to find: $\frac{\partial e^2}{\partial w_1}, \frac{\partial e^2}{\partial b_1}, \frac{\partial e^2}{\partial w_2}, \frac{\partial e^2}{\partial b_2}$ → (stochastic) instantaneous gradients vs. true gradients → $\frac{\partial E(e^2)}{\partial w_1}, \ldots$
- to do so: use back-propagation (or chain rule)

True vs. Instantaneous (Stochastic) gradients

True gradient $\rightarrow \nabla_w \rightarrow w_{n+1} = w_n - \mu \nabla_w$ (GD)

Stochastic " $\rightarrow \hat{\nabla}_w \rightarrow w_{n+1} = w_n - \mu \hat{\nabla}_w$ (SGD)

$$y = w_1 x_1 + w_2 x_2 \; ; \; E[y] = w_1 \mu_{x_1} + w_2 \mu_{x_2}$$

True gradient: $\frac{\partial E[y]}{\partial w_1} = \mu_{x_1}$; $\frac{\partial E[y]}{\partial w_2} = \mu_{x_2}$

need distribution of inputs.

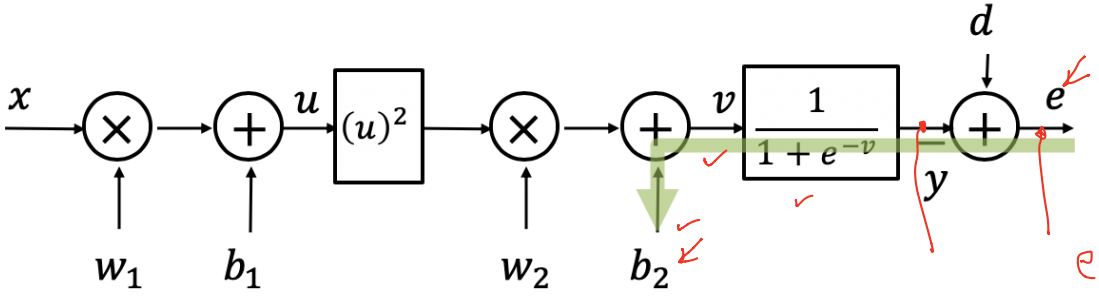
Stochastic gradient:

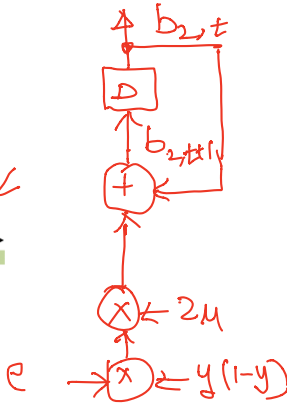
$$\frac{\partial E[y]}{\partial w_1} \longrightarrow \frac{\partial y}{\partial w_1} = x_1 \; ; \; \frac{\partial y}{\partial w_2} = x_2$$

Batch gradient:

$$\frac{\partial E[y]}{\partial w_1} \longrightarrow \frac{1}{M} \sum_i^M \left(\frac{\partial y}{\partial w_1}\right)_i \rightarrow \text{sample average}$$

Back-propagating to b_2





- $\frac{\partial e^2}{\partial b_2} = \frac{\partial e^2}{\partial y} \cdot \frac{\partial y}{\partial v} \cdot \frac{\partial v}{\partial b_2} = -2ey(1-y)$

$-2e$ $\quad y(1-y)$ $\quad 1$

$\hat{\nabla}_{b_2} =$

$e = d - y \rightarrow e^2 = d^2 - 2dy + y^2$

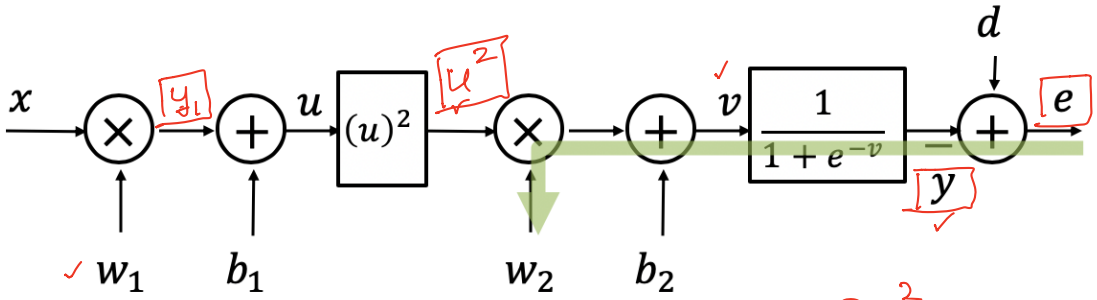
$\frac{\partial e^2}{\partial y} = -2d + 2y$

$= 2(y-d) = -2e$

$b_2 \leftarrow b_2 + \mu(-\hat{\nabla}_{b_2})$

- SGD update on b_2: $b_{2,t+1} \leftarrow b_{2,t} + 2\mu e_t y_t (1 - y_t)$

Back-propagating to w_2



weight gradient

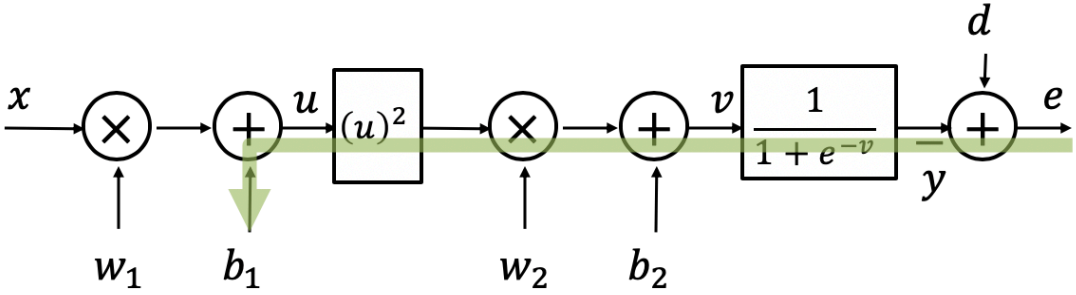
- $\frac{\partial e^2}{\partial w_2} = \frac{\partial e^2}{\partial y} \cdot \frac{\partial y}{\partial v} \cdot \frac{\partial v}{\partial w_2} = -2ey(1-y)u^2$

$\frac{\partial e^2}{\partial x}$ (activation gradient)

$-2e$ $\quad y(1-y)$ $\quad u^2$

- SGD update on w_2: $w_{2,t+1} \leftarrow w_{2,t} + 2\mu e_t y_t (1-y_t) u_t^2$

Back-propagating to b_1

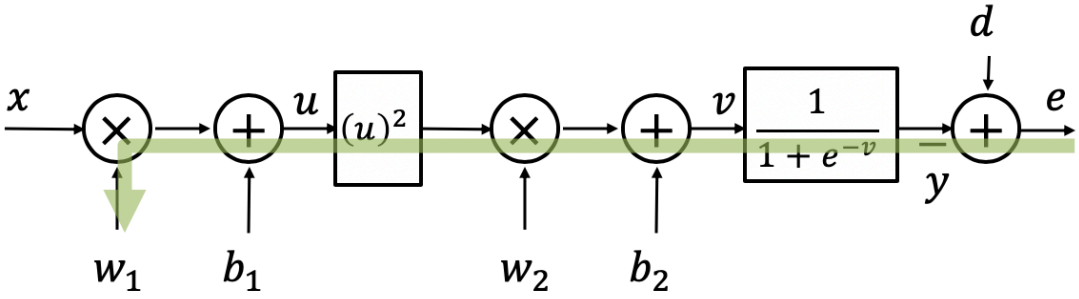


- $\frac{\partial e^2}{\partial b_1} = \frac{\partial e^2}{\partial y} \cdot \frac{\partial y}{\partial v} \cdot \frac{\partial v}{\partial u} \cdot \frac{\partial u}{\partial b_1} = -4ey(1-y)uw_2$

  $-2e$ $\quad y(1-y)$ $\quad 2uw_2$ $\quad 1$

- SGD update on b_1: $b_{1,t+1} \leftarrow b_{1,t} + 4\mu e_t y_t (1-y_t) u_t w_{2,t}$

Back-propagating to w_1

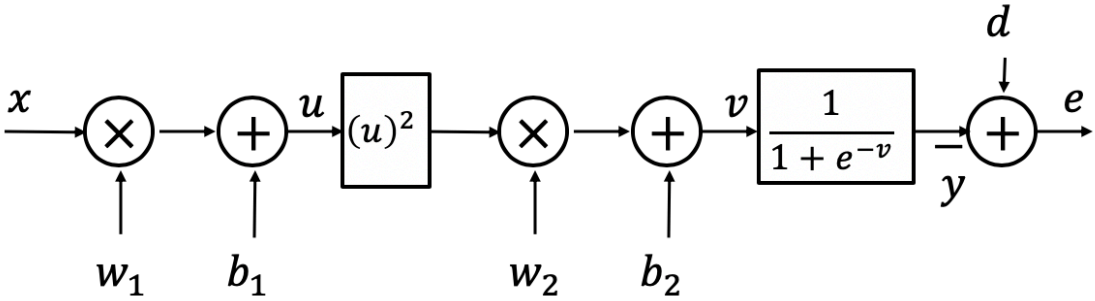


- $\frac{\partial e^2}{\partial w_1} = \frac{\partial e^2}{\partial y} \cdot \frac{\partial y}{\partial v} \cdot \frac{\partial v}{\partial u} \cdot \frac{\partial u}{\partial w_1} = -4ey(1-y)uw_2x$

  $-2e$ $y(1-y)$ $2uw_2$ x

- SGD update on w_1: $w_{1,t+1} \leftarrow w_{1,t} + 4\mu e_t y_t (1-y_t) u_t w_{2,t} x_t$

Summary



$$w_{1,t+1} \leftarrow w_{1,t} + 4\mu e_t y_t (1 - y_t) u_t w_{2,t} x_t$$

$$b_{1,t+1} \leftarrow b_{1,t} + 4\mu e_t y_t (1 - y_t) u_t w_{2,t}$$

$$w_{2,t+1} \leftarrow w_{2,t} + 2\mu e_t y_t (1 - y_t) u_t^2$$

$$b_{2,t+1} \leftarrow b_{2,t} + 2\mu e_t y_t (1 - y_t)$$

Gradient Computation in PyTorch

- deep learning packages provide a tool which allows you to compute derivatives effortlessly
- every computation graph is stored and its dual is computed
- the dual graph is used to compute derivatives automatically

using SGD with Pytorch: a few lines

```
outputs = net(inputs)
loss = criterion(outputs, targets)
loss.backward()
with torch.no_grad():
    for param in net.parameters():
        param.data.add_(-lr,param.grad.data)
        param.grad.data.zero_()
```

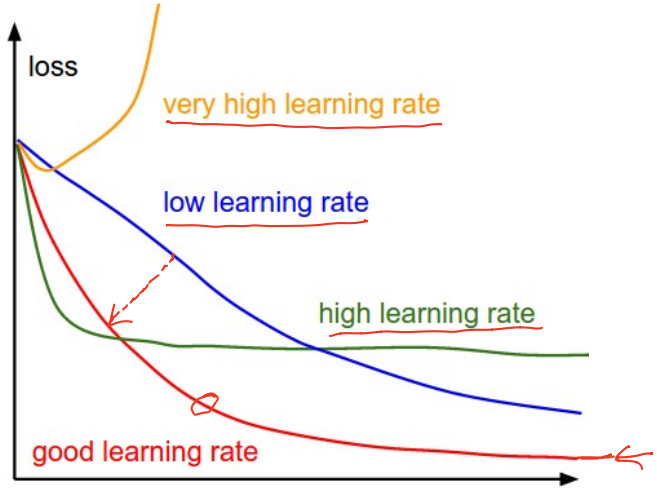
feedforward

automatic differentiation

weight update

Impact of Learning Rate on Convergence (μ)

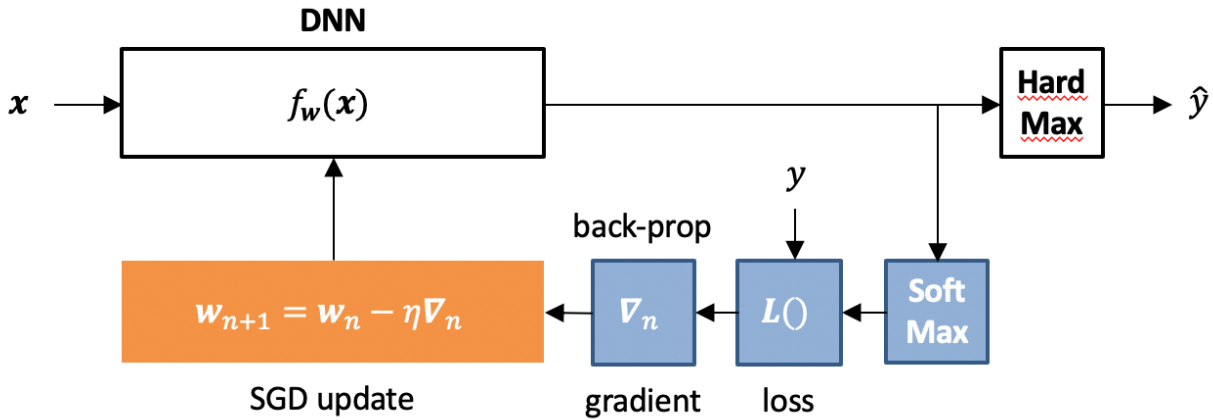
- a **very large** learning rate → instability
- a **large learning rate** wiggles at the minimum loss point
- a **small learning rate** causes slow convergence
- the learning rate should be neither too small nor too large
- gear shifting is a very popular trick to obtain the best of both worlds



[cartoon from http://cs231n.github.io/neural-networks-3/]

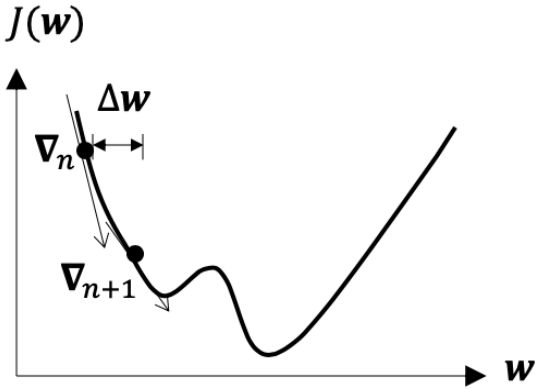
The Stochastic Gradient Descent (SGD) Algorithm

Overview



- Forward pass – generates predicted label $\hat{y}$
- Backward pass – SGD-based back-prop to update weights

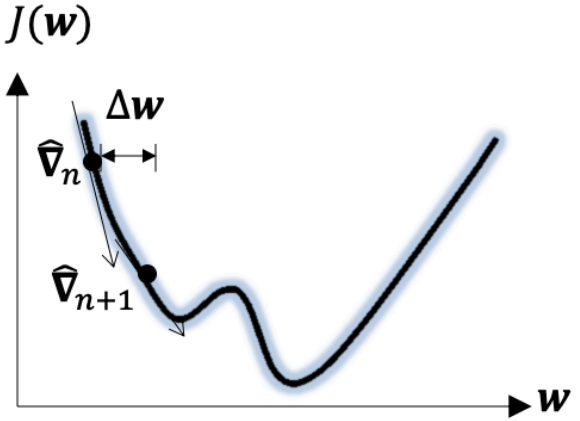
Gradient Descent



$$w_{n+1} = w_n + \mu(-\nabla_n)$$

- $\nabla_n = \frac{\partial J(w)}{\partial w}|_{w=w_n} \Rightarrow$ is the gradient of the true loss function wrt w evaluated at $w = w_n$
- μ: step-size or **learning rate**
- Loss function reduces if it is smooth. Can get stuck in a local minima.
- Gear-shift μ to jump over local minima.

Stochastic Gradient Descent

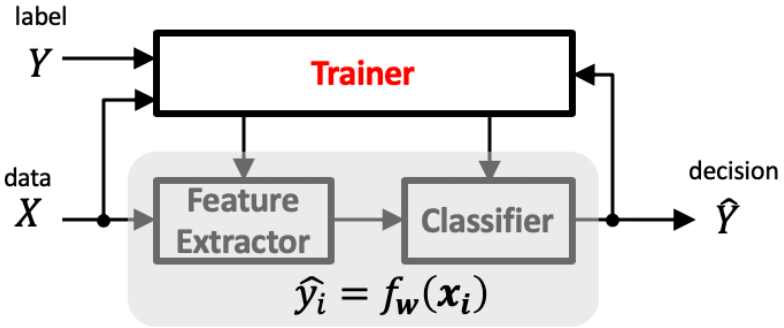


Replaces true gradient ∇_n with sample gradient $\hat{\nabla}_n$

$$w_{n+1} = w_n + \mu(-\hat{\nabla}_n)$$

- $\hat{\nabla}_n = \frac{\partial \hat{J}(w_n)}{\partial w_n} \Rightarrow$ is the gradient of the *sample loss function* wrt w_n
- μ: step-size or learning rate

SGD-based Learning Algorithms



- choose a predictor function: $\hat{y}_i = f_w(x_i)$ (w is unknown)
- choose a loss function: $L(\hat{y} = f_w(x), y)$ (differentiable)
- choose a data set: (x_i, y_i); initialize weights: $w = w_0$
- compute sample gradient of loss function: $\hat{\nabla}_i$
- determine update rule: $w_{i+1} = w_i + \mu(-\hat{\nabla}_i)$

History

- SGD applied to a DNN (multi-layer network) results in **back-propagation** algorithm
- Based on the application of chain rule and SGD
- First proposed in Paul Werbos's doctoral dissertation [1974]

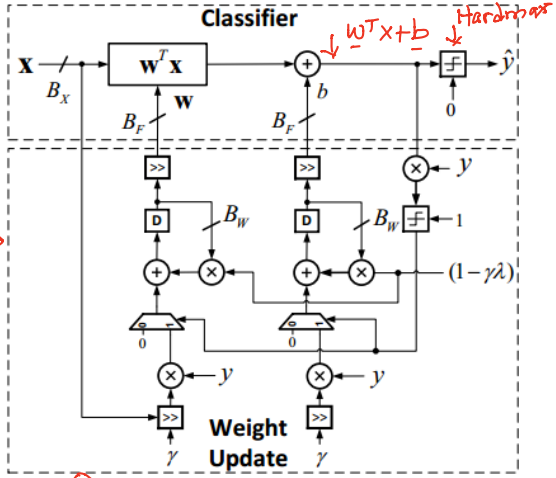
P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Committee on Appl. Math., Harvard Univ., Cambridge, MA, Nov. 1974.

- Rediscovered and popularized by Rumelhart, Hinton, and Williams [1986]

D. Rumelhart, D. Hinton, and G. Williams, "Learning internal representations by error propagation," in D. Rumelhart and F. McClelland, eds., *Parallel Distributed Processing, Vol. 1.* Cambridge, MA: M.I.T. Press, 1986.

Example - SVM (Support Vector Machine)

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma \hat{\nabla}_n$$



- sample loss function:

$$\hat{J}(\mathbf{w}_n) = \frac{1}{2}\lambda||\mathbf{w}_n||^2 + \max(0, 1 - y_n \mathbf{w}_n^T \mathbf{x}_n)$$

(w^Tw: regularizer; hinge loss: >1)

- sample gradient:

$$\hat{\nabla}_n = \frac{\partial \hat{J}(\mathbf{w}_n)}{\partial \mathbf{w}_n} = \lambda \mathbf{w}_n + \begin{cases} 0 & \text{if } y_n \mathbf{w}_n^T \mathbf{x}_n > 1 \\ -y_n \mathbf{x}_n & \text{otherwise} \end{cases}$$

$$\frac{\partial (1 - y_n w^T x_n)}{\partial w_n} = -y_n x_n$$

- SVM-SGD update rule:

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma \begin{cases} \lambda \mathbf{w}_n & \text{if } y_n \mathbf{w}_n^T \mathbf{x}_n > 1 \\ \lambda \mathbf{w}_n - y_n \mathbf{x}_n & \text{otherwise} \end{cases}$$

Example – K-means

- sample loss function:

$$\hat{J}(\mathbf{w}_{1,n}, \ldots, \mathbf{w}_{K,n}) = \min_{k=1\ldots K} \tfrac{1}{2}||\mathbf{x}_n - \mathbf{w}_{k,n}||^2$$

- Let $k^* = \arg \min_{k=1\ldots K} \tfrac{1}{2}||\mathbf{x}_n - \mathbf{w}_{k,n}||^2$, then

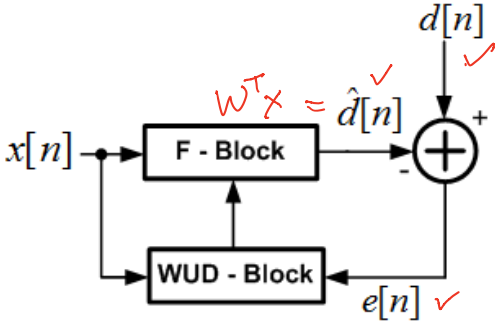
$$\hat{J}(\mathbf{w}_{1,n}, \ldots, \mathbf{w}_{K,n}) = \tfrac{1}{2}||\mathbf{x}_n - \mathbf{w}_{k^*,n}||^2$$

- sample gradient:

$$\nabla_{\mathbf{w}_{k,n}} \hat{J}(\mathbf{w}_{1,n}, \ldots, \mathbf{w}_{K,n}) = -\begin{cases} \mathbf{x}_n - \mathbf{w}_{k,n} & \text{if } k = k^* \\ 0 & \text{otherwise} \end{cases}$$

- K-means-SGD update rule:

$$\begin{cases} \mathbf{w}_{k,n+1} = \mathbf{w}_{k,n} + \gamma(\mathbf{x}_n - \mathbf{w}_{k,n}) & \text{if } k = k^* \\ \mathbf{w}_{k,n+1} = \mathbf{w}_{k,n} & \text{otherwise} \end{cases}$$



Example - LMS

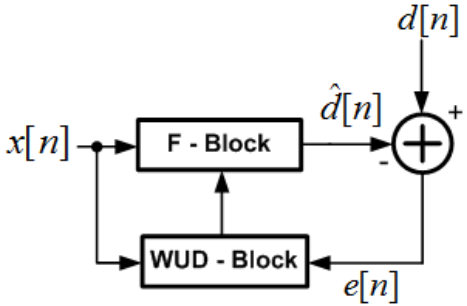
$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \mu(-\widehat{\boldsymbol{\nabla}}_n)$$ — SGD

- sample loss function: $\hat{J}(\boldsymbol{w}) = e^2(n) = (d(n) - \boldsymbol{w}_n^T \boldsymbol{x}_n)^2$;
- sample gradient:

$$\widehat{\boldsymbol{\nabla}}_n = \frac{\partial e^2(n)}{\partial \boldsymbol{w}_n} = 2e(n)\frac{\partial e(n)}{\partial \boldsymbol{w}_n} = 2e(n)\frac{\partial (d(n) - \boldsymbol{w}_n^T \boldsymbol{x}_n)}{\partial \boldsymbol{w}_n}$$
$$= -2e(n)\boldsymbol{x}(n)$$

- LMS update:

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + 2\mu e(n)\boldsymbol{x}(n)$$



Example – Sign-LMS

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \mu(-\hat{\boldsymbol{\nabla}}_n)$$

- Loss function: absolute error $E[|e_n|]$
- The instantaneous estimate of the gradient is

$$\hat{\nabla}_n = \frac{\partial |e_n|}{\partial \mathbf{w}_n} = \text{sign}(e_n) \cdot \frac{\partial e_n}{\partial \mathbf{w}_n}$$

- Use chain rule + derivative of the absolute function is the sign function.
- But,

$$\frac{\partial e_n}{\partial \mathbf{w}_n} = \frac{\partial (d_n - \mathbf{w}_n^T \mathbf{x}_n)}{\partial \mathbf{w}_n} = -\mathbf{x}_n$$

- So

$$\hat{\nabla}_n = \frac{\partial |e_n|}{\partial \mathbf{w}_n} = -\text{sign}(e_n) \cdot \mathbf{x}_n$$

- This gives the Sign-LMS equation:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu \cdot \text{sign}(e_n) \cdot \mathbf{x}_n$$

Vanilla SGD

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

LMS

$$e_n = y_n - \boldsymbol{w}_n^T x_n$$

$$\boldsymbol{w}_n = \boldsymbol{w}_n + \eta e_n \boldsymbol{x}_n$$

$$J(\boldsymbol{w}; \boldsymbol{x}_n; y_n) = e_n^2 = (y_n - \boldsymbol{w}_n^T \boldsymbol{x}_n)^2$$

$$\boldsymbol{\nabla}_n = \frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}}\Big|_{\boldsymbol{w}=\boldsymbol{w}_n} = -2e_n \boldsymbol{x}_n$$

SGD Variants used in DNN Training

An overview of gradient descent optimization algorithms*

Sebastian Ruder
Insight Centre for Data Analytics, NUI Galway
Aylien Ltd., Dublin
ruder.sebastian@gmail.com

(arxiv, June 2017)

Momentum

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)_t$$

$$\theta_{t+1} = \theta_t - v_t$$

g_t ← gradient

0.9

gradient averaging

$$v_t = \eta \sum_{i=0}^{t} g_{t-i}$$

- $\gamma \sim 0.9; \eta \sim 0.01$
- employs gradient/update history to generate current update
- dampens oscillations near local minima

Nestorov Acceleration

$$\approx g_{t+1}$$

$$v_t = \gamma\, v_{t-1} + \eta \nabla_\theta J(\theta_t - \gamma v_{t-1})$$

$$\theta_{t+1} = \theta_t - v_t \quad \Rightarrow \quad \theta_{t+1} \approx \theta_t - \gamma v_{t-1}$$

- $\gamma \sim 0.9; \eta \sim 0.01$
- similar to momentum except use predicted/future gradient to make update

AdaGrad

[Duchi et al., 2011]

$$g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i}) \quad \Rightarrow \quad \theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$ **Vanilla SGD**

$$G_{t,ii} = \sum_{k=1}^{t} g_{k,i}^2$$ **accumulate squared gradients from 1 to *t***

dimension

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$ **AdaGrad update**

η : large if G_t is large

- **adapts** learning rates **per dimension** and over time
- works well with sparse gradients

RMSProp

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

exponentially decaying average squared gradient

- includes a forgetting factor
- works well on-line and non-stationary settings

ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION

Diederik P. Kingma*
University of Amsterdam, OpenAI
dpkingma@openai.com

Jimmy Lei Ba*
University of Toronto
jimmy@psi.utoronto.ca

- ADAM - adaptive momentum estimation
- computes 1st and 2nd moments of gradient
- combines advantages of AdaGrad and RMSProp
- Advantages:
  - updates are invariant to scaling of gradient ✓
  - step-size is bounded by step-size hyperparameter ✓
  - naturally performs step-size annealing ✓
  - works with sparse gradients ✓

First & Second Moments

- gradient at sample index t: g_t

- first moment: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
- second moment: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

- bias correction: to make exponentially decaying averages approach true average, i.e.,$m_t \rightarrow \mathbb{E}[g_t]$

- bias-corrected 1st moment: $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$
- bias-corrected 2st moment: $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$

Bias Correction – for 2nd Moment

loop unroll

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \quad \Rightarrow \quad v_t = (1-\beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \cdot g_i^2$$

$$\begin{aligned} \mathbb{E}[v_t] &= \mathbb{E}\left[(1-\beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \cdot g_i^2 \right] \\ &= \mathbb{E}[g_t^2] \cdot (1-\beta_2) \sum_{i=1}^{t} \beta_2^{t-i} + \zeta \\ &= \mathbb{E}[g_t^2] \cdot (1-\beta_2^t) + \zeta \end{aligned}$$

- for stationary $g_t \rightarrow \zeta = 0$

Adam Update Rule

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

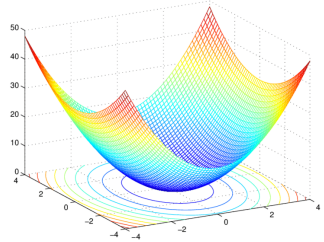
Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

SGD Tricks & Tweaks

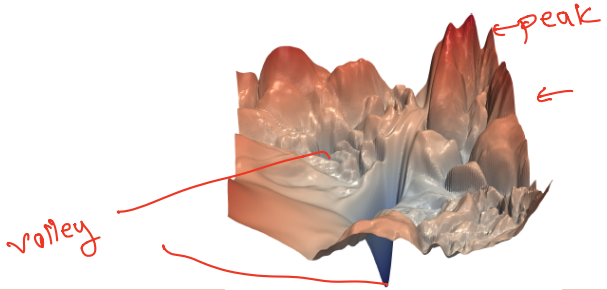
Initialization

- random initializing of linear regressor works → regressor converges to optimum regardless of initial conditions (convex loss function)

$$\boldsymbol{w}_n = \boldsymbol{w}_n + \eta e_n \boldsymbol{x}_n$$



- DNN loss function is non-convex → need to initialize properly: what does this mean?



Delving Deep into Rectifiers:
Surpassing Human-Level Performance on ImageNet Classification

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun

Microsoft Research

{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

- account for: 1) presence of ReLU, and 2) cascaded architecture

$$\mathbf{y}_l = \mathrm{W}_l \mathbf{x}_l + \mathbf{b}_l \qquad n_l = \dim(\mathbf{x}_l)$$

L-layer network

$$Var[y_L] = Var[y_1] \left(\prod_{l=2}^{L} \frac{1}{2} n_l Var[w_l] \right)$$

sufficient condition for convergence to a scalar

$$\frac{1}{2} n_l Var[w_l] = 1$$

- initialize weights as $w_l \sim N(0, \sigma_w^2)$ with $\sigma_w^2 = \frac{2}{n_l}$
- similar initialization in backward path with proper n_l

Batch Norm

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Arxiv,2015

Sergey Ioffe
Google Inc., *sioffe@google.com*

Christian Szegedy
Google Inc., *szegedy@google.com*

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

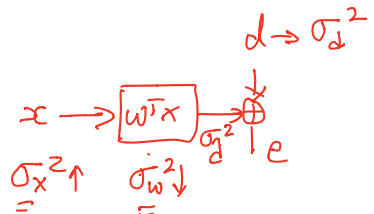
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

standardized

$E[\hat{x}] = 0$
$\mathrm{Var}(\hat{x}) = 1$



- ideally want to whiten (decorrelate the input) into a layer (why?) Because it speeds-up convergence
- Too expensive → standardize inputs into each layer:

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

- what is $\mathbb{E}[\hat{x}^{(k)}]$ and $Var(\hat{x}^{(k)})$?
- But, need to ensure the network's representational power is preserved:

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

- learn the βs and γs during training

Learning Schedule

- variable step-size (gear shifting) – tries to achieve fast convergence and high accuracy simultaneously
- Learning rate schedule: reduces μ as convergence proceeds
- large initial μ speeds up convergence
- small later μ later results in higher accuracy

- E.g., reduce μ by factor of 2 until precision limits are reached

Drop-out

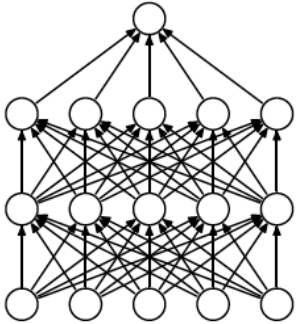
Journal of Machine Learning Research 15 (2014) 1929-1958 Submitted 11/13; Published 6/14

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

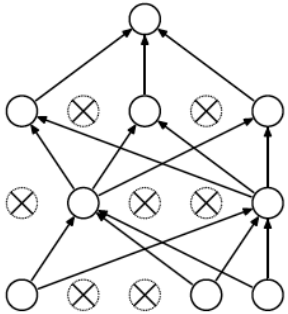
Nitish Srivastava NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU

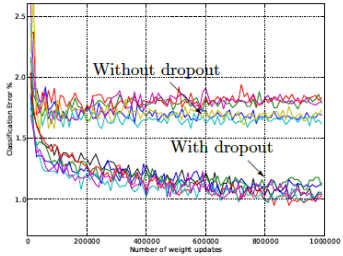
- a regularization technique
- ideal regularizer: average of predictions of an ensemble of networks (model combining)
- hard to realize → need multiple networks architectures or train with different data
- dropout is an efficient way to do this

standard network

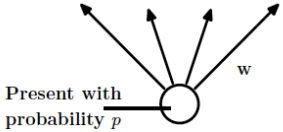


dropout thinned network

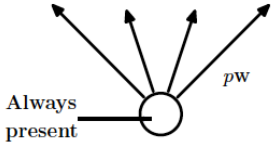




during training

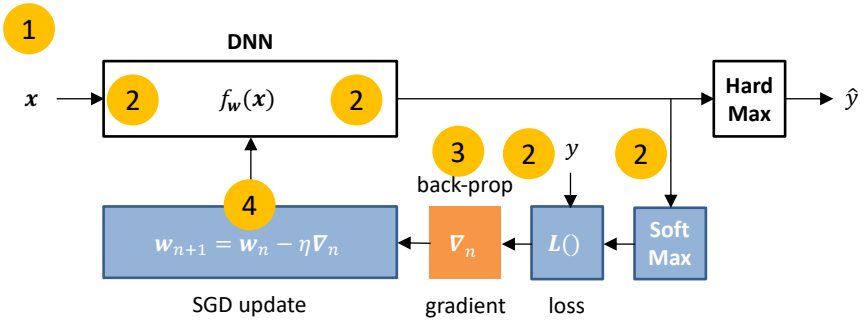


during test



- retain each node with probability p per sample
- p: retain probability of a unit/node ~ 0.5 to 1.0
- trains an ensemble of 2^N networks (N: number of nodes)

Summary- Key Steps in Training a DNN



- **Step 1**: fetch one batch of inputs
- **Step 2**: forward propagation and comparison of output with true label (loss function)
- **Step 3**: back-propagation of gradients with respect to loss function
- **Step 4**: weight update $w \leftarrow w - \mu \nabla_w$ (or other update rule)
- **Steps 1-4** are repeated until satisfactory convergence

Summary

- Trade-off between complexity and accuracy
- Based on design intuitions to preserve accuracy
- separable convolutions
- use of point-wise convolutions
- delaying max-pooling
- others

Course Web Page

https://courses.grainger.illinois.edu/ece598nsg/fa2020/
https://courses.grainger.illinois.edu/ece498nsu/fa2020/

http://shanbhag.ece.uiuc.edu