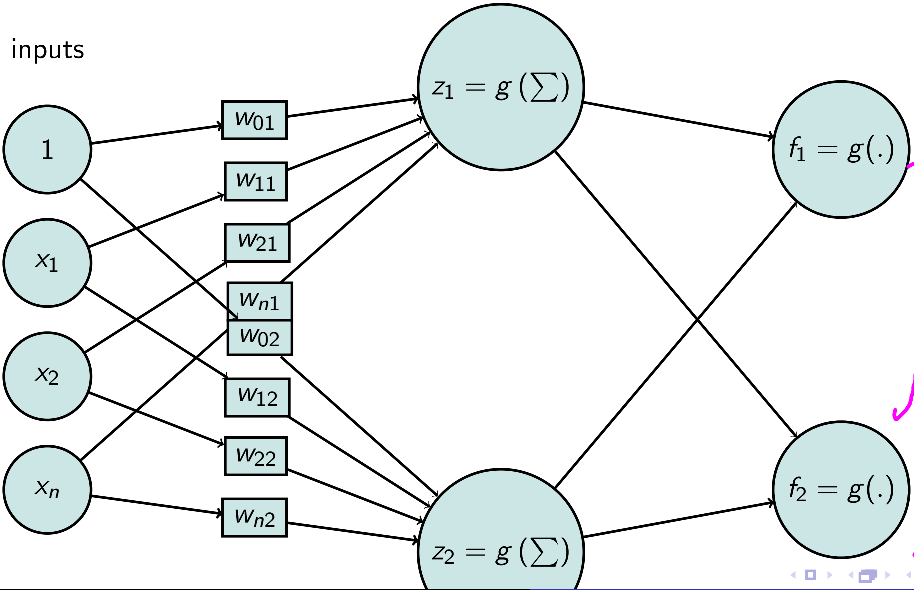
Lecture 22: Neural Networks, Back-propagation, etc

Instructor: Prof. Ganesh Ramakrishnan

Feed-forward Neural Nets



One hot encoding $\Rightarrow$ exactly one of the K nodes in (L^{th}) o/p layer expected to be a 1 & rest 0

Training a Neural Network

STEP 0: Pick a network architecture

- Number of input units: Dimension of features $\phi\left(\mathbf{x}^{(i)}\right)$.
- Number of output units: Number of classes.
- Reasonable default: 1 hidden layer, or if > 1 hidden layer, have same number of hidden units in every layer.
- Number of hidden units in each layer a constant factor (3 or 4) of dimension of x.
- We will use
  - the smooth sigmoidal function $g(s) = \frac{1}{1+e^{-s}}$: **We have now learnt how to train a single node sigmoidal (LR) neural network**
  - instead of the non-smooth step function $g(s) = 1$ if $s \in [\theta, \infty)$ and $g(s) = 0$ otherwise.

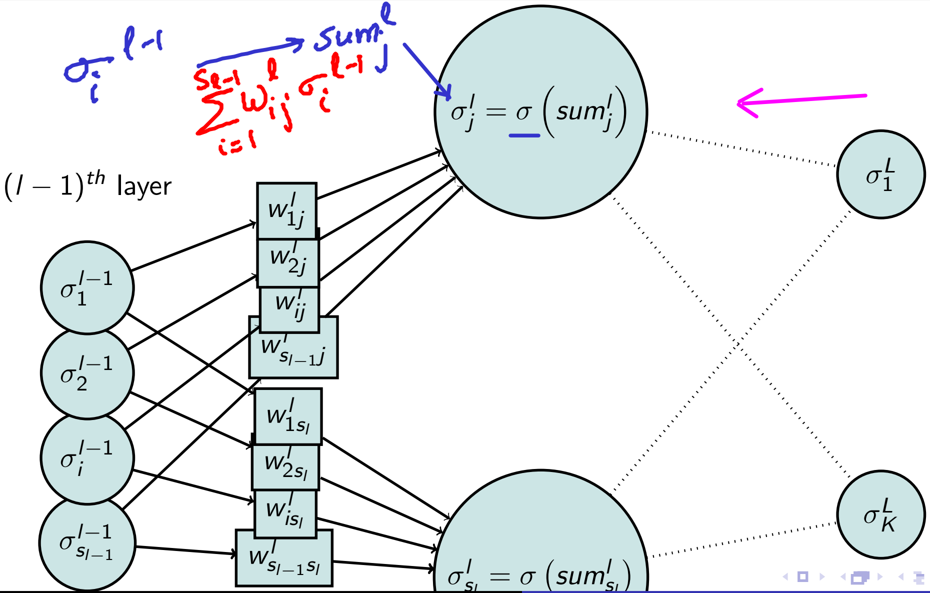
High Level Overview of Backpropagation Algorithm for Training NN

1. Randomly initialize weights w_{ij}^{l} for $l = 1, \ldots, L$, $i = 1, \ldots, s_l$, $j = 1, \ldots, s_{l+1}$.
2. Implement **forward propagation** to get $f_w(\mathbf{x})$ for any $x \in \mathcal{D}$.
3. Execute **backpropagation**
   1. by computing partial derivatives $\frac{\partial}{\partial w_{ij}^{(l)}} E(w)$ for $l = 1, \ldots, L$, $i = 1, \ldots, s_l$, $j = 1, \ldots, s_{l+1}$.
   2. and using gradient descent to try to minimize (non-convex) $E(w)$ as a function of parameters $\mathbf{w}$.

$$w_{ij}^{l} = w_{ij}^{l} - \eta \underline{\frac{\partial}{\partial w_{ij}^{(l)}} E(w)}$$

4. Verify that the cost function $E(w)$ has indeed reduced, else resort to some random perturbation of weights $\mathbf{w}$.

Setting Notation for Backpropagation



Gradient Computation

- The Neural Network objective to be minimized:

$$E(\mathbf{w}) = \underbrace{-\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\sigma_k^L\left(\mathbf{x}^{(i)}\right)\right) + \left(1 - y_k^{(i)}\right)\log\left(1 - \sigma_k^L\left(\mathbf{x}^{(i)}\right)\right)\right]}_{\text{Average cross entropy at } L^{th} \text{ layer}} + \underbrace{\frac{\lambda}{2m}\sum_{l=1}^{L}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(w_{ij}^l\right)^2}_{\text{Regularization on wts across all layers}} \tag{1}$$

Gradient Computation

- The Neural Network objective to be minimized:

$$E(\mathbf{w}) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\sigma_k^L\left(\mathbf{x}^{(i)}\right)\right) + \left(1 - y_k^{(i)}\right)\log\left(1 - \sigma_k^L\left(\mathbf{x}^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(w_{ij}^l\right)^2 \quad (1)$$

- $sum_j^l = \sum_{k=1}^{s_{l-1}} w_{kj}^l \sigma_k^{l-1}$ and $\sigma_i^l = \frac{1}{1+e^{-sum_i^l}}$
- $\frac{\partial \mathbf{E}}{\partial \mathbf{w_{ij}^l}} = \frac{\partial \mathbf{E}}{\partial \sigma_j^l} \frac{\partial \sigma_j^l}{\partial \mathbf{sum_j^l}} \frac{\partial \mathbf{sum_j^l}}{\partial \mathbf{w_{ij}^l}} + \frac{\lambda}{2m} w_{ij}^l$
- $\frac{\partial \sigma_j^l}{\partial \mathbf{sum_j^l}} = \left(\frac{1}{1+e^{-sum_i^l}}\right)\left(1 - \frac{1}{1+e^{-sum_i^l}}\right) = \boldsymbol{\sigma_j^l(1 - \sigma_j^l)}$
- $\frac{\partial \mathbf{sum_j^l}}{\partial \mathbf{w_{ij}^l}} = \frac{\partial}{\partial w_{ij}^l}\left(\sum_{k=1}^{s_{l-1}} w_{kj}^l \sigma_k^{l-1}\right) = \boldsymbol{\sigma_i^{l-1}}$

- For a single example $(\mathbf{x}, y)$:

$$-\left[\sum_{k=1}^{K} y_k \log\left(\sigma_k^L(\mathbf{x})\right) + (1-y_k)\log\left(1-\sigma_k^L(\mathbf{x})\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(w_{ij}^l\right)^2 \quad (2)$$

σ of current layers

sum of current layer

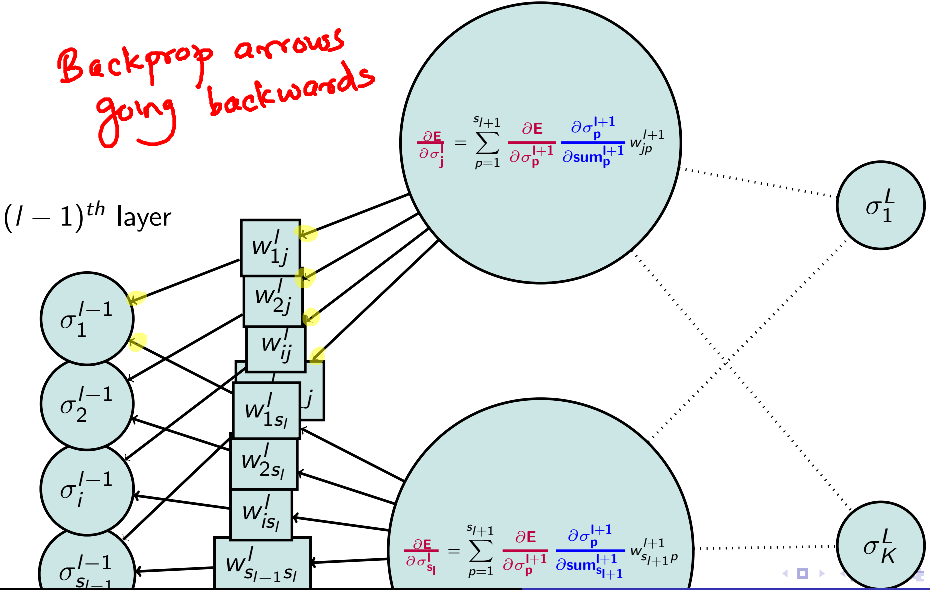
- $\frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{l}}} = \sum_{p=1}^{s_{l+1}} \frac{\partial E}{\partial sum_p^{l+1}} \frac{\partial sum_p^{l+1}}{\partial \sigma_j^l} = \sum_{p=1}^{s_{l+1}} \frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{p}}^{\mathbf{l+1}}} \frac{\partial \sigma_{\mathbf{p}}^{\mathbf{l+1}}}{\partial \mathbf{sum}_{\mathbf{p}}^{\mathbf{l+1}}} w_{jp}^{l+1}$ since $\frac{\partial sum_p^{l+1}}{\partial \sigma_j^l} = w_{jp}^{l+1}$

σ of prev layer

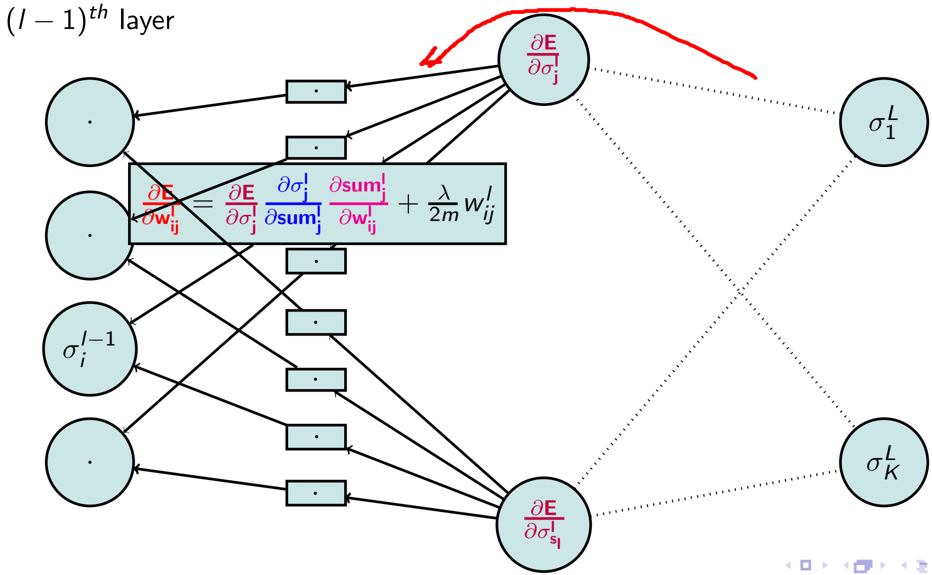
- $\frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{L}}} = -\frac{\mathbf{y_j}}{\sigma_{\mathbf{j}}^{\mathbf{L}}} - \frac{\mathbf{1-y_j}}{\mathbf{1}-\sigma_{\mathbf{j}}^{\mathbf{L}}}$

Backpropagation in Action

Backprop arrows going backwards



Backpropagation in Action

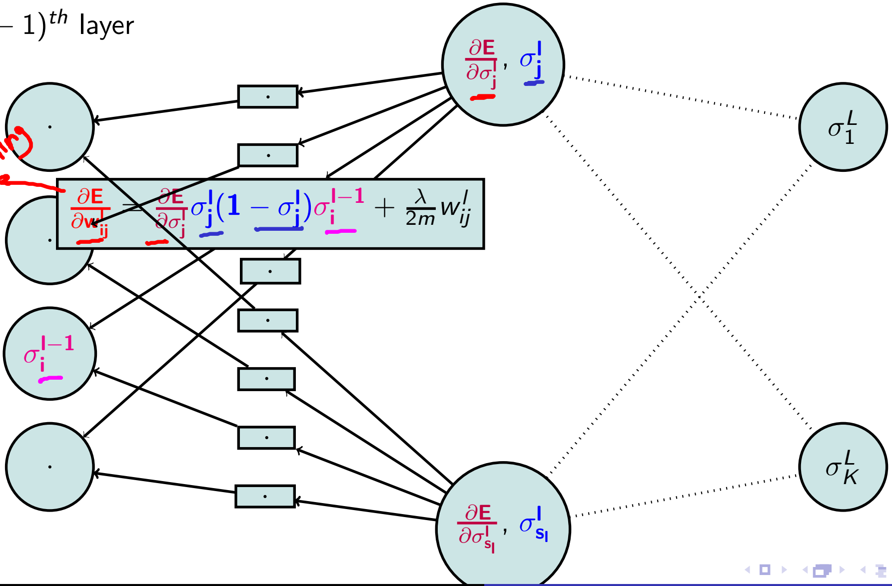


Recall and Substitute

- $sum_j^l = \sum_{k=1}^{s_{l-1}} w_{kj}^l \sigma_k^{l-1}$ and $\sigma_i^l = \frac{1}{1+e^{-sum_i^l}}$
- $\frac{\partial \mathbf{E}}{\partial \mathbf{w_{ij}^l}} = \frac{\partial \mathbf{E}}{\partial \sigma_\mathbf{j}^\mathbf{l}} \frac{\partial \sigma_\mathbf{j}^\mathbf{l}}{\partial \mathbf{sum_j^l}} \frac{\partial \mathbf{sum_j^l}}{\partial \mathbf{w_{ij}^l}} + \frac{\lambda}{2m} w_{ij}^l$
- $\frac{\partial \sigma_\mathbf{j}^\mathbf{l}}{\partial \mathbf{sum_j^l}} = \sigma_\mathbf{j}^\mathbf{l}(\mathbf{1} - \sigma_\mathbf{j}^\mathbf{l})$
- $\frac{\partial \mathbf{sum_j^l}}{\partial \mathbf{w_{ij}^l}} = \sigma_\mathbf{i}^\mathbf{l-1}$
- $\frac{\partial \mathbf{E}}{\partial \sigma_\mathbf{j}^\mathbf{l}} = \sum_{p=1}^{s_{l+1}} \frac{\partial \mathbf{E}}{\partial \sigma_\mathbf{j}^\mathbf{l+1}} \frac{\partial \sigma_\mathbf{j}^\mathbf{l+1}}{\partial \mathbf{sum_j^{l+1}}} w_{jp}^{l+1}$
- $\frac{\partial \mathbf{E}}{\partial \sigma_\mathbf{j}^\mathbf{L}} = -\frac{\mathbf{y_j}}{\sigma_\mathbf{j}^\mathbf{L}} - \frac{\mathbf{1-y_j}}{\mathbf{1}-\sigma_\mathbf{j}^\mathbf{L}}$

Backpropagation in Action

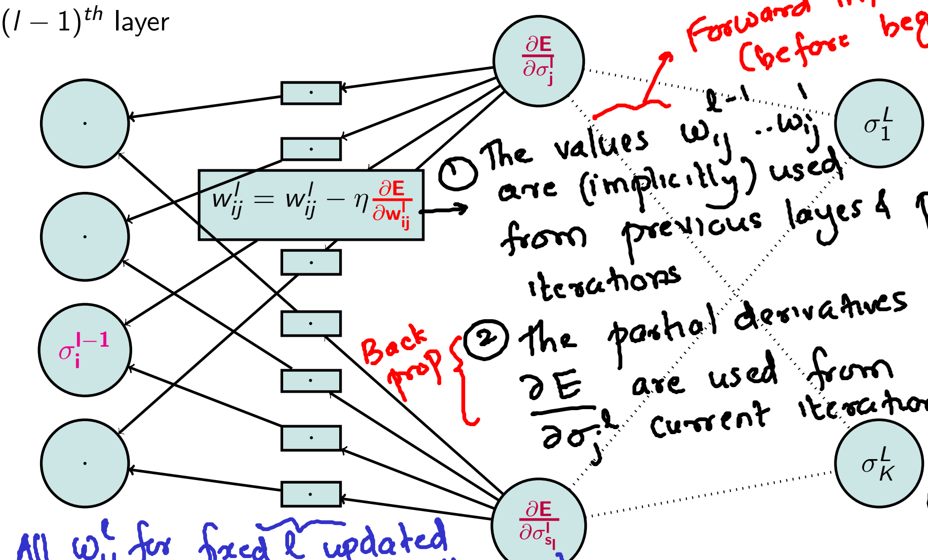
$(l-1)^{th}$ layer



$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \sigma_j^l} \sigma_j^l (1 - \sigma_j^l) \sigma_i^{l-1} + \frac{\lambda}{2m} w_{ij}^l$$

used for updating w_{ij}

Backpropagation in Action



Forward inference (before beginning backprop)

① The values $w_{ij}^{l-1} .. w_{ij}^{1}$ are (implicitly) used from previous layers & previous iterations

Back prop

② the partial derivatives $\frac{\partial \sigma_j^l}{\partial sum_j^l}$ $\frac{\partial E}{\partial \sigma_j^l}$ are used from current iterations but for $l' > l$ (following layers)

All w_{ij}^l for fixed l updated simultaneously

w_{ij}^{l}'s being updated simultaneously etc and other addition/multiplication operations are converted into efficient matrix/tensor operations in NN packages such as tensorflow, Theano, Torch etc.

The Backpropagation Algorithm for Training NN

Typically 0's, unless RELU is used

1. Randomly initialize weights w^l_{ij} for $l = 1, \ldots, L$, $i = 1, \ldots, s_l$, $j = 1, \ldots, s_{l+1}$.
2. Implement **forward propagation** to get $f_{\mathbf{w}}(\mathbf{x})$ for every $\mathbf{x} \in \mathcal{D}$. (using w^l_{ij} from previous iteration)
3. Execute **backpropagation** on any misclassified $\mathbf{x} \in \mathcal{D}$ by performing gradient descent to minimize (non-convex) $E(\mathbf{w})$ as a function of parameters $\mathbf{w}$.
4. $\frac{\partial E}{\partial \sigma^L_j} = -\frac{y_j}{\sigma^L_j} - \frac{1-y_j}{1-\sigma^L_j}$ for $j = 1$ to s_L. (on final/output layer)
5. For $l = L-1$ down to 2:
   1. $\frac{\partial E}{\partial \sigma^l_j} = \sum_{p=1}^{s_{l+1}} \frac{\partial E}{\partial \sigma^{l+1}_j} \sigma^{l+1}_j (1 - \sigma^{l+1}_j) w^{l+1}_{jp}$
   2. $\frac{\partial E}{\partial w^l_{ij}} = \frac{\partial E}{\partial \sigma^l_j} \sigma^l_j (1 - \sigma^l_j) \sigma^{l-1}_i + \frac{\lambda}{2m} w^l_{ij}$
   3. $w^l_{ij} = w^l_{ij} - \eta \frac{\partial E}{\partial w^l_{ij}}$

   Illustrated on prev slides

   Backpropagate on previous layers successively
6. Keep picking misclassified examples until the cost function $E(\mathbf{w})$ shows significant reduction; else resort to some random perturbation of weights $\mathbf{w}$ and restart a couple of times.

Note: If $\sigma_j^l(sum_j^l) = \dfrac{1}{1+e^{-sum_j^l}}$ is

replaced by RELU (Rectified linear unit)

$$relu_j^l(sum_j^l) = \max(0, sum_j^l)$$

Then $\dfrac{\partial \sigma_j^l}{\partial sum_j^l}$ becomes $\underbrace{\dfrac{\partial relu_j^l}{\partial sum_j^l}}_{\text{subgradient}} = 1$ (if $sum_j^l > 0$)

Therefore initializing $w_{ij}^l = 0$ can become meaningless. $\Leftarrow$ $\Big\{$ & $\dfrac{\partial relu_j^l}{\partial sum_j^l} = 0$ (if $sum_j^l \le 0$)

Efficient alternatives to Stochastic Gradient Descent (SGD)

SGD on (Strongly) Convex problems $\Rightarrow$ Pending error is $(O(\frac{1}{k}))$ $O(\frac{1}{\sqrt{k}})$ → k = # iterations

→ only indicative

- **Average (sigmoidal) gradient on a minibatch of m_b examples:**

$$\frac{\partial E}{\partial \sigma_j^L} = \frac{1}{m_b}\sum_{i=1}^{m_b} -\frac{y_j^{(i)}}{\underbrace{\sigma_j^L(x^{(i)})}} - \frac{1-y_j^{(i)}}{\underbrace{1-\sigma_j^L(x^{(i)})}} \text{ for } j = 1 \text{ to } s_L.$$

$\underbrace{\nabla E(i\in\{m\})}_{\text{True or batch}}$ $\quad \underbrace{\nabla E(i\in\{m_b\})}_{\text{Minibatch stochastic}} \cdots \underbrace{\nabla E(i)}_{\text{Stochastic}}$

Does not say much about our nonconvex avg cross entropy fn for NN's with hidden layers.

Necessary condition for optimality: $\nabla E(i\in\{m\}) = 0$

(Does not imply) $\Rightarrow$ $\nabla E(i\in\{m_b\}) = 0$ OR $\nabla E(i) = 0$

→ More probably 0 than for stochastic

$$w^{(k)} = w^{(k-1)} + \eta \nabla E(\underset{\sim}{m_b})$$

If $m_b = m$, then as $w^{(k-1)} \to w^*$

$\nabla_m E(w^{(k-1)}) \to 0$

$\Rightarrow w^{(k-1)} \to w^{(k)} \to w^*$ as $k\uparrow$

For $m_b < m$, can

η_k compensate for

a misbehaving

∇E_{m_b}

① η_k instead of η

② $\eta_k \to 0$ as $k\uparrow$

If $m_b < m$, then even as $w^{(k-1)} \to w^*$

No guarantee that $\nabla_{m_b} E(w^{(k-1)}) \to 0$

$\Rightarrow$ Even if $w^{(k-1)} \to w^{(*)}$ yet $w^{(k)}$ is updated

to drift away from w^*

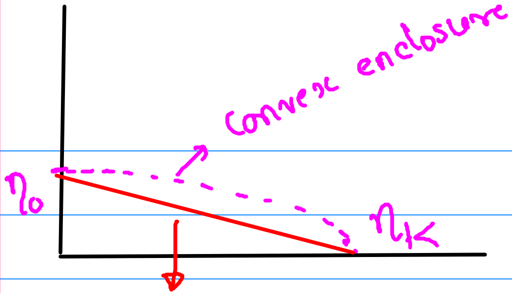
Efficient alternatives to Stochastic Gradient Descent (SGD)

SGD on (Strongly) Convex problems $\Rightarrow$ Pending error is $(O(\frac{1}{k}))$ $O(\frac{1}{\sqrt{k}})$

- **Average (sigmoidal) gradient on a minibatch of m_b examples:**

$$\frac{\partial E}{\partial \sigma_j^L} = \frac{1}{m_b}\sum_{i=1}^{m_b} -\frac{y_j^{(i)}}{\sigma_j^L(x^{(i)})} - \frac{1-y_j^{(i)}}{1-\sigma_j^L(x^{(i)})} \text{ for } j = 1 \text{ to } s_L.$$

- **Adaptive Learning Rate:** Expect at optimality $\nabla E = 0$. But not if gradient is approximated on a sample m_b! A sufficient condition for convergence of SGD is:
  - η_k (learning rate) vary across iterations $w_{ij}^l = w_{ij}^l - \eta_k \frac{\partial E}{\partial w_{ij}^l}$
  - $\sum_{k=1}^{\infty} \eta_k = \infty$ and $\sum_{k=1}^{\infty} \eta_k^2 \leq \gamma < \infty$ (eg: $\eta_k = \frac{1}{k}$, $\gamma = \frac{\pi^2}{6}$). Commonly $\eta_k = (1-\beta_k)\eta_0 + \beta_k\eta_K$, with $\beta_k = \frac{k}{K}$ that is convex combination of some max (η_0) and min (η_K) values.
- **Adagrad:** Individually adapts learning rates of all model parameters by scaling them inversely proportional to square root of sum of all their historical squared values $\eta_k = 1/\sqrt{\|\nabla E\|^2}$ over prev iterations averaged
- **RMSProp:** Modifies AdaGrad for non-convex setting by changing the gradient accumulation into an exponentially weighted moving average



Line segment for η_k

$$\eta_k = \underbrace{\beta}_{\frac{k}{K}}\eta_0 + (1-\beta)\underbrace{\eta_K}_{0.5}$$

Assume K is large enough & stopping criterion is that error on validation has stopped decreasing any further

Efficient alternatives to Stochastic Gradient Descent (SGD contd.)

Adagrad: Momentum through learning rate

Directly capture momentum for w_{ij}

- **Momentum[1] Accelerated SGD:** $w_{ij}^l = w_{ij}^l + v_{ij}^l$ where $v_{ij}^l = \alpha v_{ij}^l - \eta_k \frac{\partial \mathbf{E}}{\partial \mathbf{w_{ij}^l}}$
- **Different choice of Activation Function σ such as ReLU, tanh, etc**
- **Adam:** Best seen as a variant on the combination of RMSProp and Momentum

Associated with η_k.

associated with w_{ij}

The following part remains the same with sigmoidal functions (for $l = L-1$ down to 2):

1. $\frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{l}}} = \sum_{p=1}^{s_{l+1}} \frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{l+1}}} \sigma_{\mathbf{j}}^{\mathbf{l+1}}(\mathbf{1} - \sigma_{\mathbf{j}}^{\mathbf{l+1}}) w_{jp}^{l+1}$
2. $\frac{\partial \mathbf{E}}{\partial \mathbf{w_{ij}^l}} = \frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{l}}} \sigma_{\mathbf{j}}^{\mathbf{l}}(\mathbf{1} - \sigma_{\mathbf{j}}^{\mathbf{l}}) \sigma_{\mathbf{i}}^{\mathbf{l-1}} + \frac{\lambda}{2m} w_{ij}^l$

[1] In practice, $\alpha \in \{0.5, 0.9, 0.99\}$