Lecture 11 – Convolutional Neural Networks (Part II)

Machine Learning

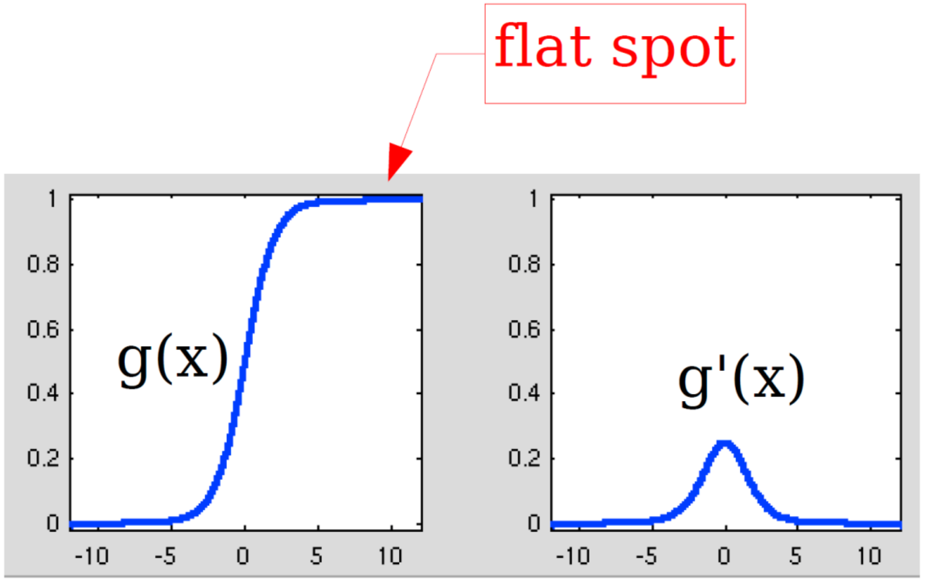
Spring Semester ‘2022

More on Convolutional Neural Networks

- Loss functions?
- What kind of output/hidden units?
- What neural network architecture to use?

Loss functions?

- MSE causes derivative of g() goes to zero if weights become large becomes large → vanishing gradient



Loss functions?

- Entropy = measure of uncertainty

Information:

$$I(x) = -\log P(x)$$

Shannon Entropy:

$$H(\mathrm{x}) = \mathbb{E}_{\mathrm{x}\sim P}[I(x)] = -\mathbb{E}_{\mathrm{x}\sim P}[\log P(x)]$$

$H(x) = -\sum_{i=1,k} P(x_i) \log_2 P(x_i)$ or $-\int_R P(x_i) \log_2 P(x_i)$

Loss functions?

- Cross entropy (P: label, Q: CNN output)

$$H(P,Q) = -\sum_{i=1,k} P(x_i)\log_2 Q(x_i)$$

If $P(0) = 1 - y, P(1) = y$

$Q(0) = 1 - o, Q(1) = o$

Then $L = -(y\log_2 o + (1-y)\log_2(1-o))$, where $o = \sigma(z)$, $z = wx + b$

And $\frac{\partial L}{\partial w} = x(o - y)$ and $\frac{\partial L}{\partial b} = (o - y)$ → no more vanishing gradients

→ compare with: $\frac{\partial E}{\partial w_{k,j}^{(2.1)}} = -2(d_k - o_k)S'\left(net_k^{(2)}\right)x_j^{(1)}$ of neural networks

In general,

$$L = -\sum_{i=1,N}(y_i\log_2 o_i + (1-y_i)\log_2(1-o_i))$$

Output units?

- **Softmax units (for multinoulli output)**
  - Consider *K*-class classification problem
  - We want to obtain probability distribution over *K* possible values
  - We want $\hat{y}_i = P(y = i|\boldsymbol{x})$ to be within 0 and 1, and to sum to 1.
  - Consider unnormalized probability distribution $\tilde{P}(y)$, which does not sum to 1.

$$\log \tilde{P}(y = i) = z_i$$

$$\tilde{P}(y = i) = \exp(z_i)$$

$$P(y = i) = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)} \triangleq softmax(\boldsymbol{z})_i$$

  - Loss function for maximum likelihood (cross entropy)

$$J(\boldsymbol{\theta}) = -\log P(y = i|\boldsymbol{x}) = -\log softmax(\boldsymbol{z})_i$$

$$\log softmax(\boldsymbol{z})_i = z_i - \log \sum_{j=1}^{K} \exp(z_j)$$

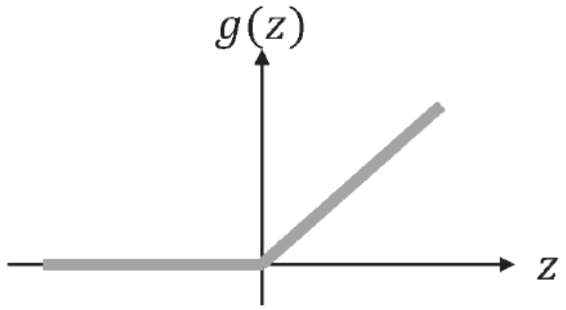
$$\approx z_i - \max_j z_j$$ = 0 if correct answer,

difference if wrong answer

Hidden units?

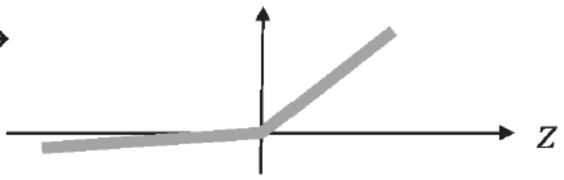
- Sigmoid units
- Rectified Linear Units (ReLU)

$$h = g(z) = \max\{0, z\}$$



  - Similar to linear units ➔ easy to optimize
  - Gradient is large (=1) whenever "active"
  - Not differentiable at 0

- **Generalization:** $g(z) = \max\{0, z\} + a \min\{0, z\}$
  - Absolute value rectification: $a = -1 \Rightarrow g(z) = |z|$
  - Leaky ReLU: small a →

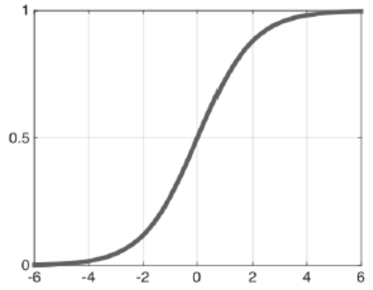


  - Parametric ReLU: training of a

Universal approximation theorem

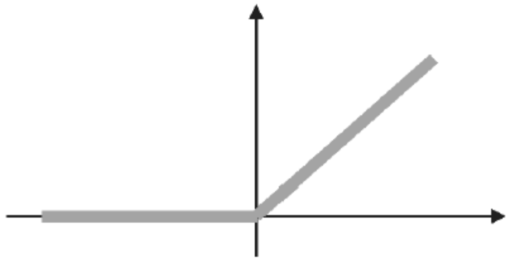
- [Hornick'89] A feedforward neural network with one hidden layer using a non-constant bounded continuous activation function can learn any continuous function arbitrarily well.

Sigmoid



- [Leshno'93] A neural networks with one hidden layer using a locally bounded piecewise continuous non-polynomial activation function can learn any continuous function arbitrarily well.

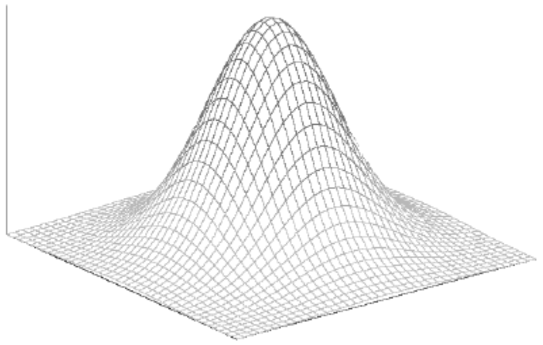
ReLU

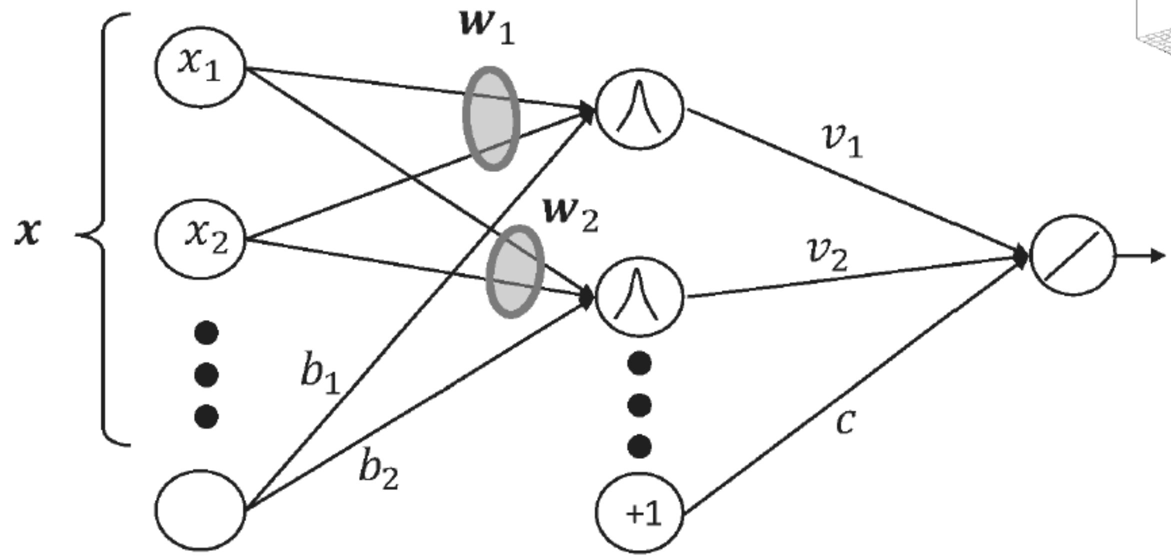


Hidden units?

- **Radial basis function (RBF) units**
  - Localized response similar to receptive field in biological sensory systems

$$h_i = \exp\left\{-\frac{\|\boldsymbol{x}-\boldsymbol{w}_i\|^2}{2b_i^2}\right\}$$





$$y = \boldsymbol{v}^T\boldsymbol{h} + c$$
$$= \sum_i v_i \exp\left\{-\frac{\|\boldsymbol{x}-\boldsymbol{w}_i\|^2}{2b_i^2}\right\} + c$$

In Deep Learning

- An overly complex model does not necessarily include the true data generating process
- We almost never have access to the true process, so we never can know for sure the right amount of regularization
- In DL, most applications are to domains where the true process is almost certainly outside of model family: DL is applied to extremely complicated domains, e.g., images, audio seq, text.
- That is, regularization in DL is not just finding the model of the right size with the right no. of parameters
  - In practice, we almost always find that the best fitting model is a large model that has been regularized appropriately.

Parameter Norm Penalty

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

where $\alpha \in [0, \infty)$ is a hyperparameter that weights the relative contribution

$\Omega(\theta)$ is the norm penalty term

- We penalize only the weights, leaving bias terms unregularized:
  - It’s often easy to fit bias terms
  - Not too much variance introduced by leaving bias terms
  - Regularizing bias terms may introduce significant underfitting

L^2 Regularization

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

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon \left(\alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})\right)$$

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

weight decay

Weight Decay as Constrained Optimization

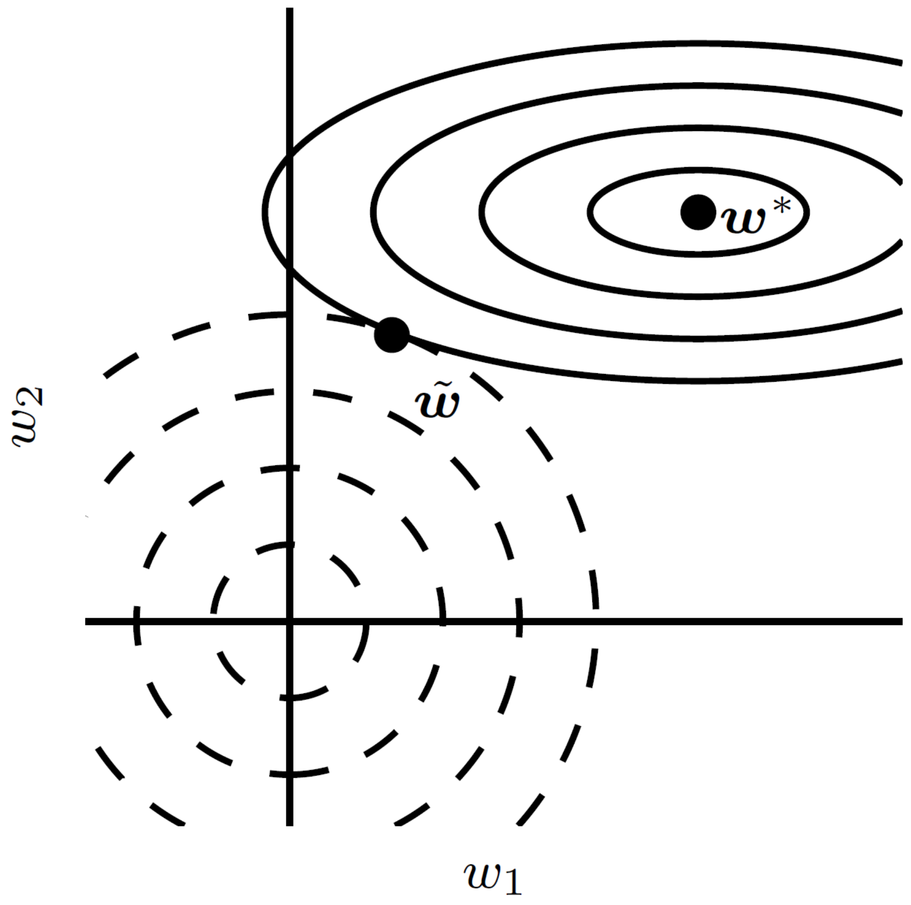


Figure 7.1

Contour of a quadratic objective

w1 direction:

- Corresponding eigenvalue of the Hessian is small
- No large change in objective moving away from w*

—> effect of regularization is large

w2 direction:

- Corresponding eigenvalue of the Hessian is large
- Large change in objective moving away from w*

—> effect of regularization is small

Early Stopping

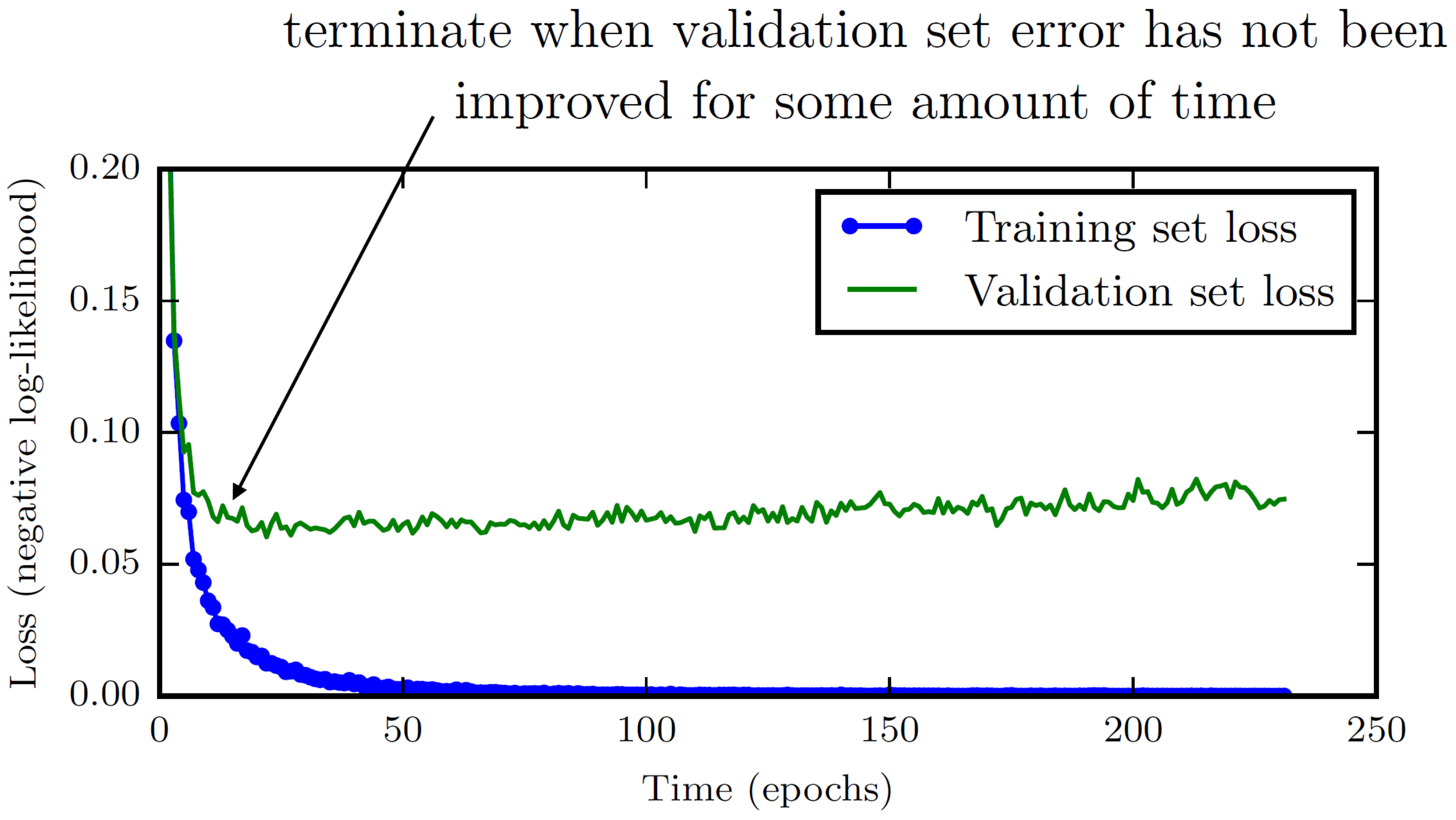
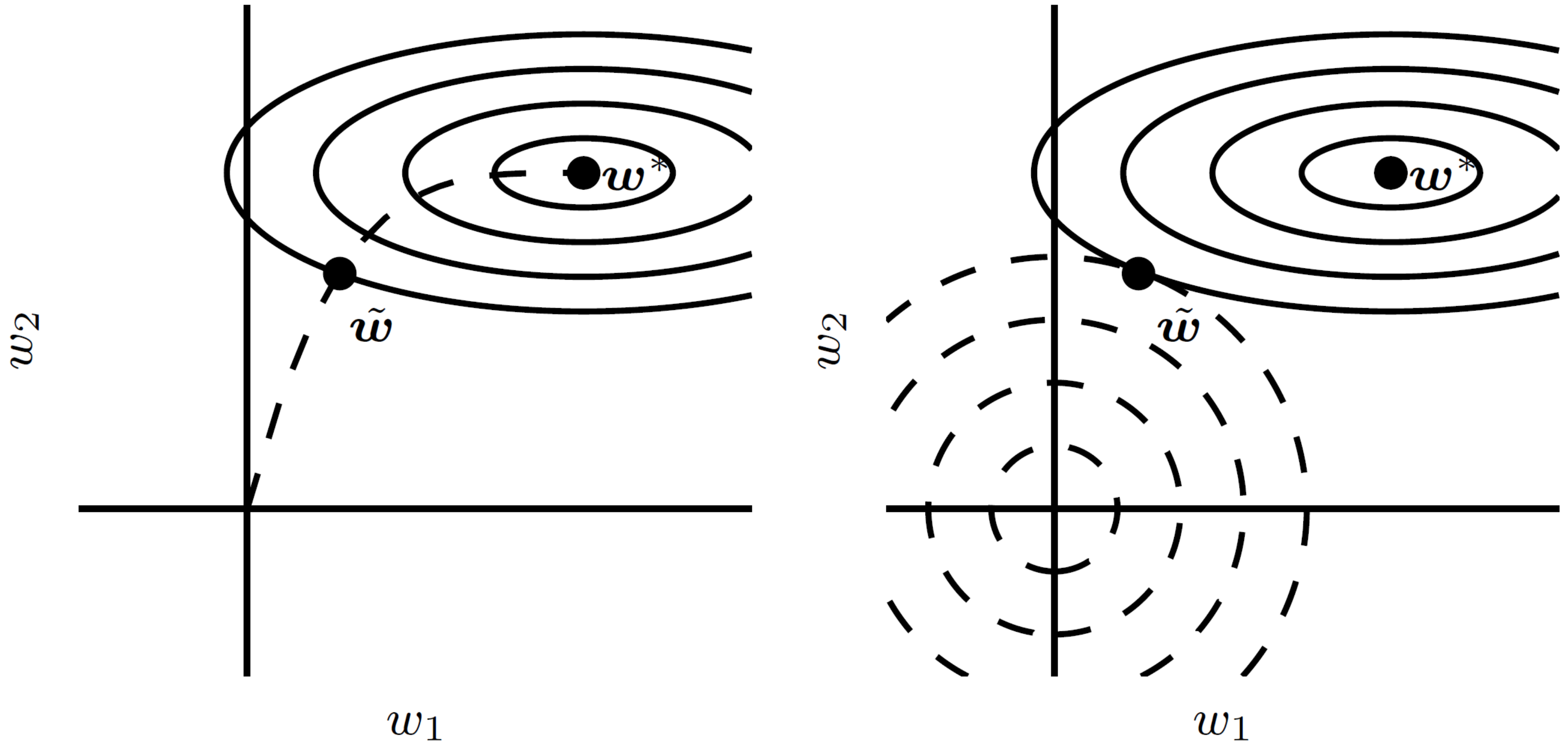


Figure 7.3

Early Stopping

- Most commonly used regularization strategy in DL
- Can be considered as a hyperparameter selection algorithm, regarding the training time
- Can be done in parallel
- Requires a validation set

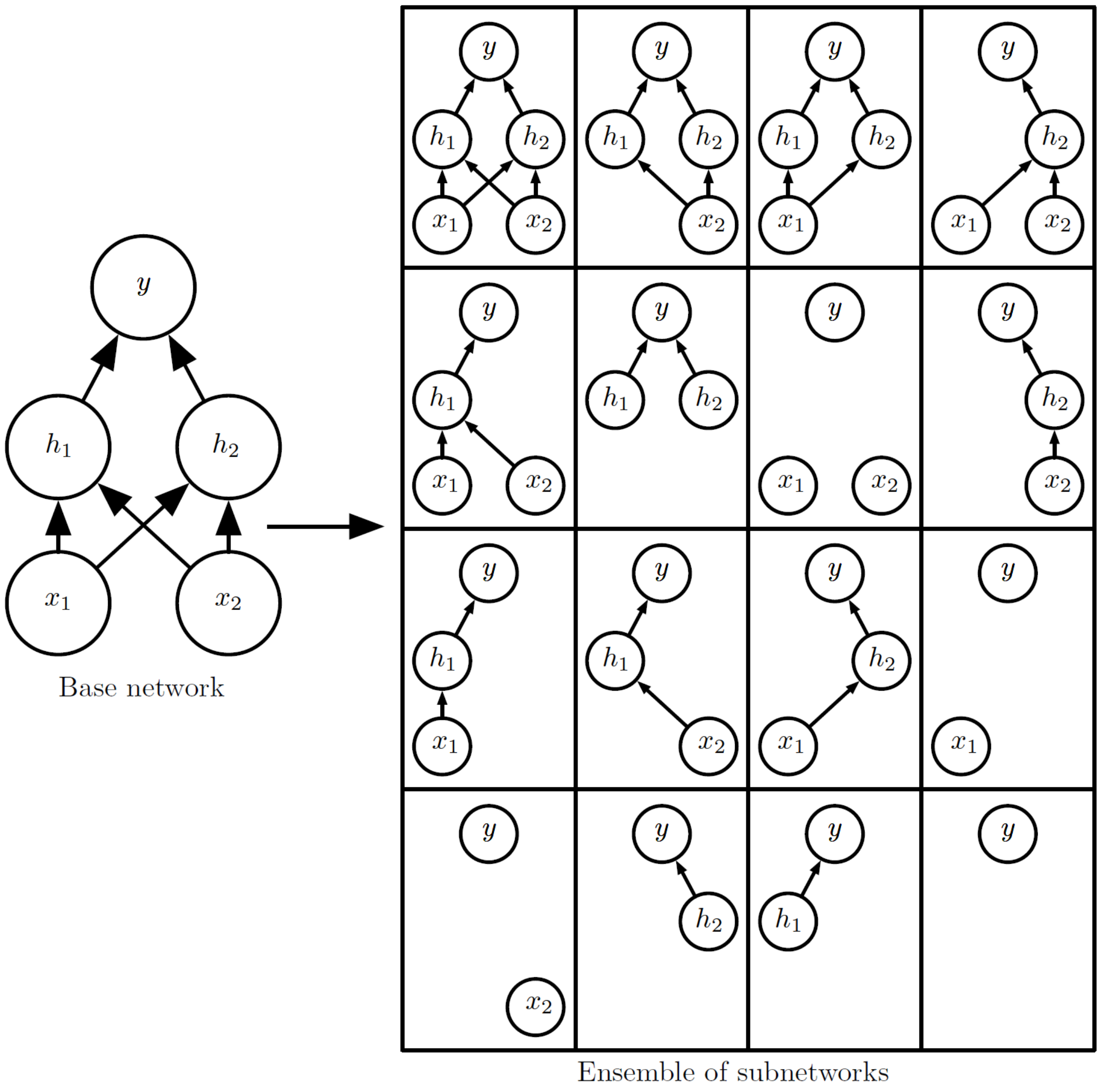
Early Stopping and Weight Decay



Dashed: trajectory of SGD

Figure 7.4

Dropout [Srivastavaet al., 2014]



Inexpensive but powerful method of regularization

Dropout trains the ensemble consisting of all sub-networks that can be formed by removing non-output units from an underlying base network

In each step of the SGD, a different binary mask is sampled to apply to all input and hidden units

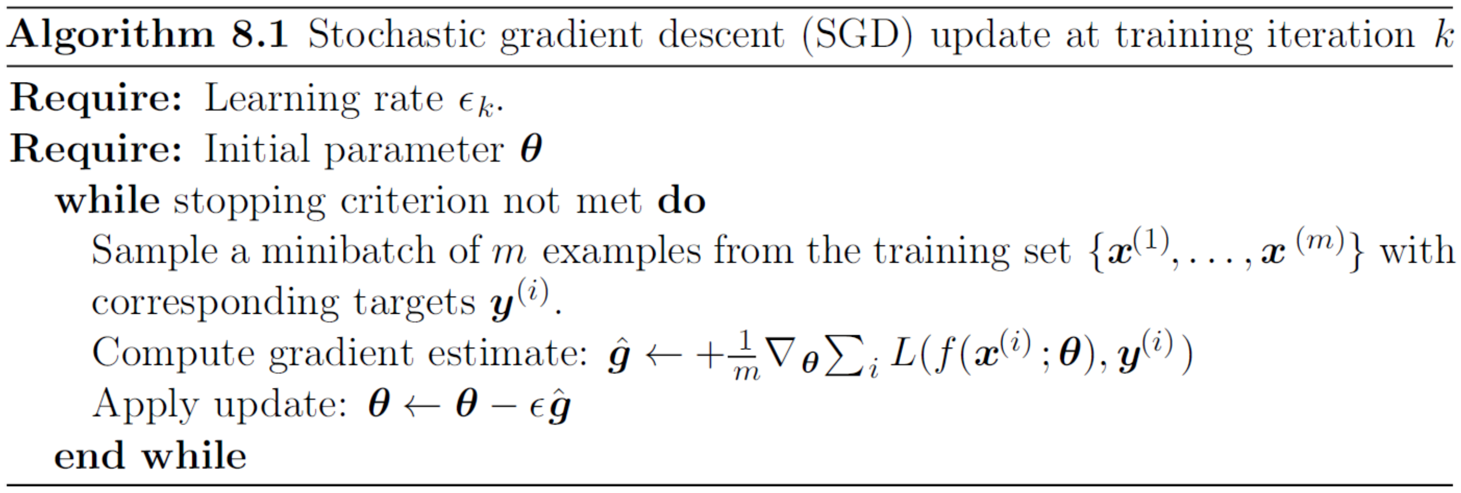
Large networks are preferred to apply dropout

Batch and Minibatch Algorithms (1)

- In practice, we can compute the expected gradients by randomly sampling a small number of examples from the data set and averaging only on those examples.
- For reasons such as:
  1. If the training set is huge, taking one step towards the minimum is very expensive
  2. Larger number of samples do not significantly reduce the standard error mean
  3. Redundancy in the training set

Batch and Minibatch Algorithms (2)

- Batch (Deterministic) Gradient Methods
  - Process all training examples simultaneously in a large batch
- Online Gradient Methods
  - Process a single example at a time
- Minibatch (Stochastic) Gradient Methods
  - Process more than one but less than all of the training examples



Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

```
Require: Learning rate $\epsilon_k$.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\hat{\boldsymbol{g}}$
  end while
```

Parameter (weight) initialization

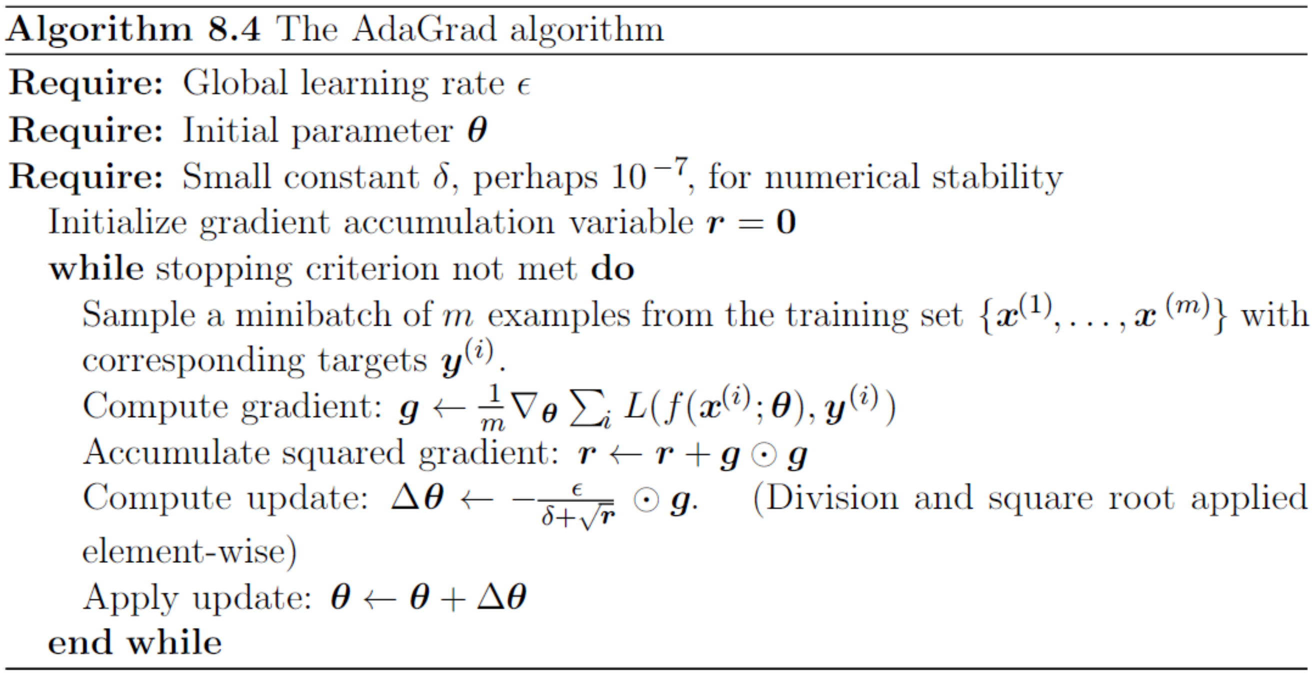
- The initial point can determine whether the algorithm converges at all.
- The only property known with complete certainty is that the initial parameters need to “break the symmetry” between different units.
  - Gaussian or a uniform distribution
  - Scale of the distribution

$$W_{i,j} \sim U\left(-\frac{6}{\sqrt{m+n}}, \frac{6}{\sqrt{m+n}}\right).$$

- Biases can be safely set to heuristically chosen constant

Optimization with adaptive learning rates (1)

- AdaGrad

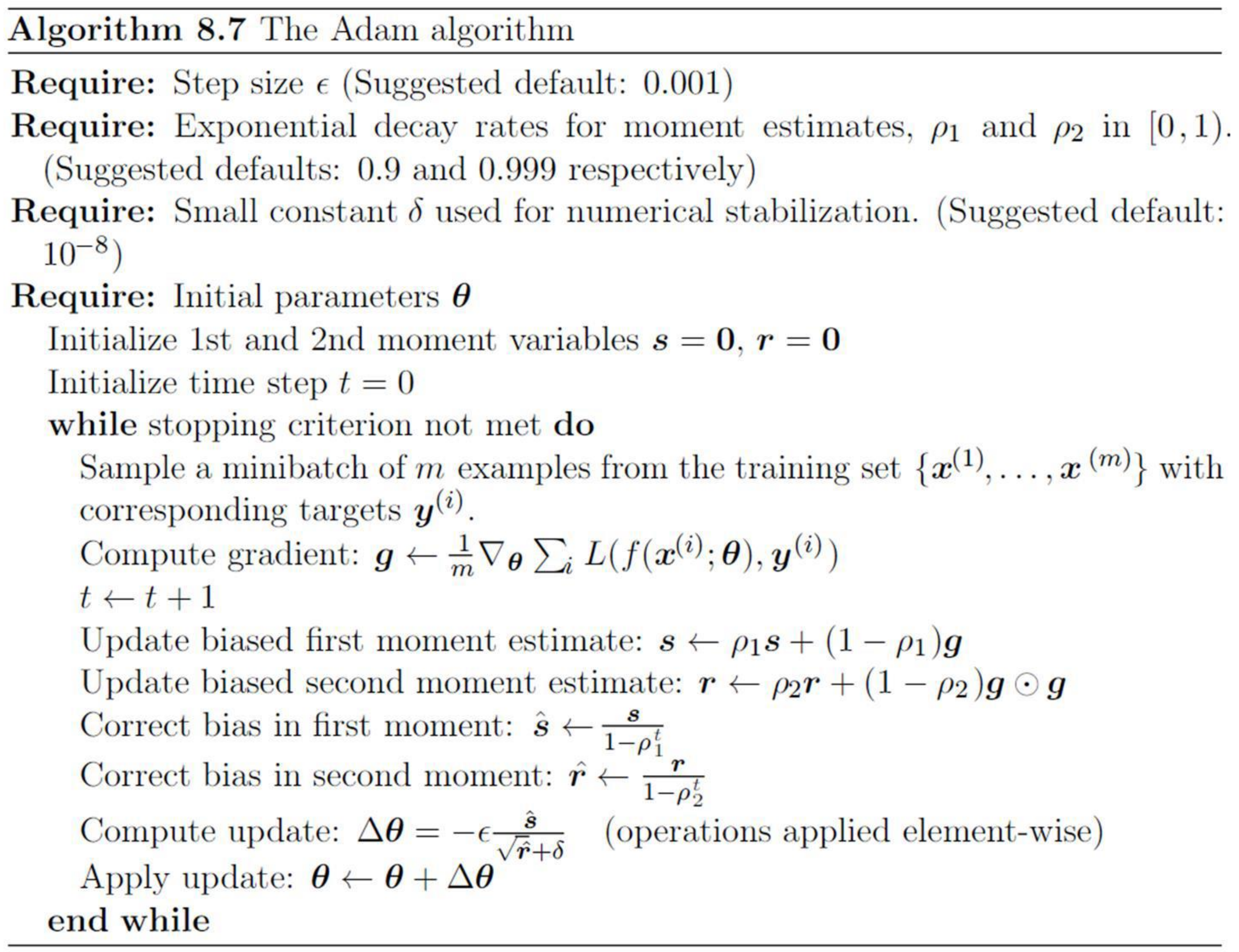


Algorithm 8.4 The AdaGrad algorithm

```
Require: Global learning rate $\epsilon$
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, perhaps $10^{-7}$, for numerical stability
  Initialize gradient accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
    Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta+\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$.   (Division and square root applied
    element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
```

  - weights that receive high gradients will have their effective learning rate reduced
  - weights that receive small or infrequent updates will have their effective learning rate increased

Optimization with adaptive learning rates (2)



Algorithm 8.7 The Adam algorithm

```
Require: Step size $\epsilon$ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, $\rho_1$ and $\rho_2$ in $[0, 1)$.
    (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant $\delta$ used for numerical stabilization. (Suggested default:
    $10^{-8}$)
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
  Initialize time step $t = 0$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta \boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$   (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
  end while
```

- Uses smooth gradient (s)