Neizrazito, evolucijsko i neuroračunarstvo
Neuro-evolucijski sustavi

dr.sc. Marko Čupić

Fakultet elektrotehnike i računarstva
Sveučilište u Zagrebu

9. siječnja 2014.

Zašto neuro-evolucijski spoj? Problem 1.

- Postoji mnoštvo vrsta neuronskih mreža (vrlo gruba podjela):
  - unaprijedne
  - samoorganizirajuće
  - dinamičke (postoje povratne veze)
  - ...
- Postoji mnoštvo različitih vrsta neurona (prijenosnih funkcija):
  - identitet
  - funkcija skoka
  - linearna
  - linearna sa zasićenjem
  - logistička
  - polinomijalna
  - tangens-hiperbolni
  - radijalne bazne funkcije
  - ...

Zašto neuro-evolucijski spoj? Problem 1.

- Za svaku vrstu mreža razvijani su zasebni algoritmi učenja
- Razvijeni algoritmi rade uz jake pretpostavke na strukturu mreže i karakteristike neurona
  - Koje su to pretpostavke za algoritam *Backpropagation*?
- Postoji velika opasnost od lokalnih optimuma
  - Na zadanom problemu algoritam može biti praktički neprimjenjiv jer vrlo brzo zaglavi u neprihvatljivo lošem lokalnom optimumu
- Gradijentni postupci primjenjivi samo na poseban skup funkcija pogrešaka.

Zašto neuro-evolucijski spoj? Problem 1.

- Algoritmi evolucijskog računanja mogu se koristiti kao robusni (tipično populacijski) alati za rješavanje optimizacijskih problema
- Zadatak učenja neuronske mreže može se formulirati kao optimizacijski problem
  - Učenje s učiteljem: pronađi parametre mreže uz koje mreža ima minimalnu ocjenu greške
  - Grupiranje: pronađi centre uz koje se devijacija unutar grupa minimizira a ukupna devijacija između grupa maksimizira
  - Podržano učenje: pronađi parametre mreže uz koje mreža radi minimalne gubitke

Zašto neuro-evolucijski spoj? Problem 2.

- Pretpostavimo da problem rješavamo unaprijednom neuronskom mrežom (zašto?)
- Hoćemo li algoritmom *Backpropagation* trenirati općenitu unaprijednu mrežu od 10, 20 ili 50 neurona?
- Hoćemo li algoritmom *Backpropagation* trenirati slojevitu mrežu od 3, 4 ili 7 slojeva? I s koliko neurona po slojevima?
- Arhitektura neuronske mreže ima velik utjecaj na performanse neuronske mreže.
- Koja je optimalna arhitektura neuronske mreže?

Zašto neuro-evolucijski spoj? Problem 2.

- Optimalnost arhitekture može se ocjenjivati različitim kriterijima.
- Algoritme evolucijskog računanja moguće je primijeniti na zadatak pronalaska optimalne arhitekture neuronske mreže.

Zašto neuro-evolucijski spoj? Problem 3.

- Prethodno smo naveli niz prijenosnih funkcija koje mogu koristiti neuroni
- U optimalnoj arhitekturi neuronske mreže, koje nam prijenosne funkcije trebaju?
- Rješenje ne mora biti uniformno: zašto različiti neuroni ne bi koristili različite prijenosne funkcije?
- Mješavina: recimo da je broj neurona i način povezivanja fiksan i recimo da imamo unaprijed zadano da n neurona treba koristiti prijenosnu funkciju $\phi_1(x)$ a m neurona prijenosnu funkciju $\phi_2(x)$. Kojim neuronima dati koju prijenosnu funkciju?
- Problem možemo rješavati algoritmima evolucijskog računanja!

Zašto neuro-evolucijski spoj? Problem 4.

- Ovisno o problemu koji se rješava neuronskom mrežom, uobičajeni postupni učenja ne moraju biti najefikasniji.
- Može li se definirati kakvo bolje pravilo učenja uz koje će postupak treniranja neuronske mreže biti djelotvorniji i učinkovitiji?
- Problem možemo rješavati algoritmima evolucijskog računanja!

Zašto neuro-evolucijski spoj?

Algoritme evolucijskog računanja s neuroračunarstvom kombiniramo u sljedećim područjima:

1. Evolucija težinskih faktora
   $\Rightarrow$ problem 1
2. Evolucija arhitektura
   $\Rightarrow$ problem 2, problem 3
3. Evolucija pravila učenja
   $\Rightarrow$ problem 4

Pretpostavke

- Arhitektura mreže je u potpunosti zadana.
- Jedina su nepoznanica vrijednosti težinskih faktora uz koje mreža nad zadanim problemom radi minimalnu pogrešku.
- Klasični algoritmi učenja unaprijednih mreža:
  - *Backpropagation* [Rumelhart, Hinton i Williams (1986)]
  - *QuickProp* [Fahlman (1989), Craig Veitch i Holmes (1991)]
  - *Algoritam konjugiranog gradijenta* [Moller (1990)]
  - *Levenberg-Marquardt algoritam* [Hagan i Menhaj (1994), Suratgar, Tavakoli i Hoseinabadi]

Klasični algoritmi za unaprijedne mreže

- Prikazani klasični algoritmi za unaprijedne mreže temeljeni su na *gradijentu* i/ili višim derivacijama
  - Mogu biti dosta neefikasni $\rightarrow$ potreban velik broj iteracija algoritma
  - Postavljaju zahtjev na funkciju pogreške: mora biti derivabilna, inače algoritam ne radi
  - Kako izračun pogreške koristi izlaz mreže, zahtjev na derivabilnost se propagira u sve elemente mreže $\rightarrow$ prijenosne funkcije moraju biti derivabilne

Klasični algoritmi za unaprijedne mreže

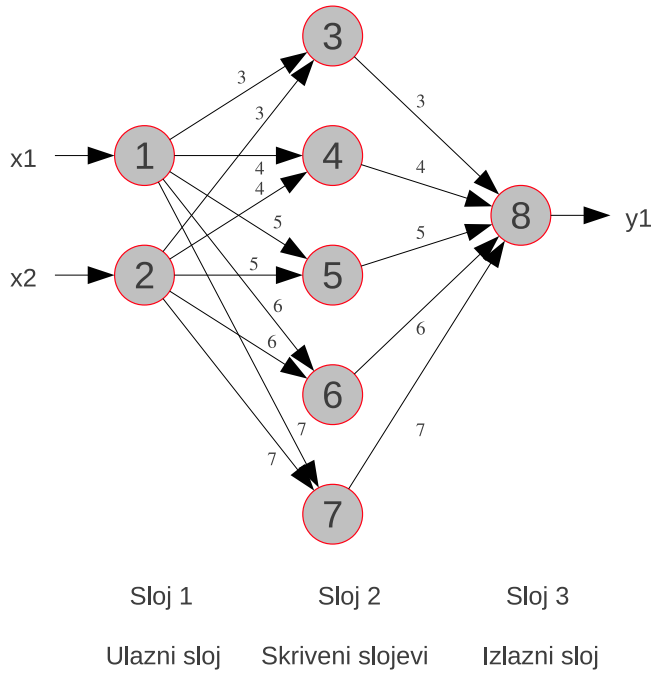
- Prikazani klasični algoritmi za unaprijedne mreže temeljeni su na *gradijentu* i/ili višim derivacijama
  - Algoritmi ove porodice su po prirodi algoritmi pronalaska najbližeg lokalnog optimuma – tamo pokazuje gradijent! Globalni optimum – ako nam se posreći
  - Otpornost na lokalne optimume pokušava se osigurati:
    - dodatnim modifikacijama pravila učenja (npr. dodavanje momenta inercije)
    - uporabom stohastičke procjene gradijenta umjesto korektne vrijednosti – kompromis:
      pravi gradijent $\rightarrow$ brža konvergencija (u najbliži optimum, najvjerojatnije lokalni);
      stohastička procjena $\rightarrow$ sporija konvergencija (ali veća šansa da izbjegnemo lokalne optimume)

Evolucija težinskih faktora

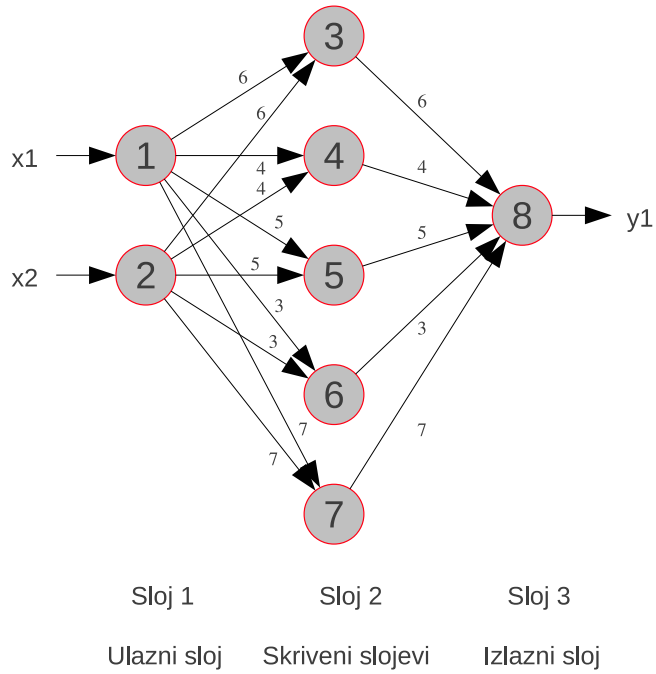
- Algoritmi evolucijskog računanja mogu se koristiti za pronalazak optimalnih težina
- Dosta robusni
- Kromosom
  - Binarni niz koji predstavlja sve težine u mreži
    - Svaka težina "troši" k bitova; ukupno za n težina kromosom sadrži $k \cdot n$ bitova
    - Potrebno odrediti način kodiranja / dekodiranja težina
    - Moguća uporaba uobičajenih evolucijskih operatora nad binarnim nizovima
    - Zahtjeva zadavanje granica prostora koji se pretražuje
  - Vektor realnih brojeva
    - Fleksibilnije rješenje
    - Granice pretraživanja nisu nužne (često niti ne znamo kako ih dobro postaviti)
    - Treba biti oprezan s evolucijskim operatorima

Problem permutacija

- Neuronske mreže su po prirodi vrlo simetrične strukture
  - Postoji više različitih zapisa neuronske mreže koja obavlja identično preslikavanje (koliko za ovaj primjer?)



(a) Originalna mreža



(b) Drugačija mreža za isto preslikavanje

Problem permutacija

- Kod populacijskih algoritama koji koriste operatore kombiniranja dva li više rješenja simetričnost rezultira problemom
  $\Rightarrow$ djeca su vrlo često bitno lošija od oba roditelja (zašto?)
- Problem je poznat pod nazivom problem permutacija (engl. *Permutation problem*, *Competing conventions problem*
- Zbog toga se često koriste samo algoritmi koji ne kombiniraju roditelje, već koriste samo operatore nad jednom jedinkom (npr. mutaciju)
- Dio znanstvenih publikacija ukazuje da ovaj problem ipak nedovoljno izražen da bi se odustalo od prednosti koje donosi operator križanja
- U praksi $\rightarrow$ oprez!

Mutacija za neuronske mreže

- Mutacija bez pristranosti
- Mutacija s pristranošću
- Mutacija neurona
- Mutacija najslabijeg neurona
- ...

(vidi skriptu za pojašnjenja)

Križanja za neuronske mreže

- Križanje težina
- Križanje neurona
- Križanje značajki
- ...

(vidi skriptu za pojašnjenja)

Poboljšanje performansi (1)

- Poznato je da algoritmi evolucijskog računanja relativno brzo uspjevaju pronaći obećavajući podprostor u prostoru koji pretražuju, a potom imaju dosta sporu konvergenciju.
- Međutim: dosta su otporni na problem lokalnih optimuma.
- Klasični algoritmi: izražen problem lokalnih optimuma; u praksi treba pokretati nekoliko puta prije no što se dobije zadovoljavajuće rješenje.

Poboljšanje performansi (2)

- Algoritmi evolucijskog računanja mogu se hibridizirati algoritmima lokalne pretrage
  - Algoritmi evolucijskog računanja rade grubu robusnu pretragu prostora
  - Lokalna pretraga zadužena je za brzu konvergenciju ka optimumu
- Lokalna pretraga može biti upravo klasičan algoritam, npr. *Backpropagation* odnosno nekoliko iteracija takvog algoritma
- Što se time gubi?

  $\Rightarrow$ algoritam više nije univerzalno primjenjiv, već funkcija mora biti takva da je lokalna pretraga na nju primjenjiva
- Oprez: koji kompromis radimo?

  $\Rightarrow$ više iteracija algoritma lokalne pretrage $\rightarrow$ konvergencija $\uparrow$
- Pitanja: Koje odluke još treba donijeti?

  $\rightarrow$ broja parametara $\uparrow$

Što želimo

- Za zadani zadatak pronaći arhitekturu optimalne neuronske mreže
- Tipično se pretražuje ograničeni podskup porodica arhitektura
- Zašto uopće postoji problem?
  - prejednostavna arhitektura – mreža ne može dobro modelirati zadano preslikavanje i radi veliku pogrešku
  - presložena arhitektura – mreža je podložna pretreniranju
    $\Rightarrow$ dobar štreber, ali loša sposobnost generalizacije
    $\Rightarrow$ postupak učenja će biti vrlo dugotrajan
- Ideja: pronaći optimalnu arhitekturu mreže
  $\Rightarrow$ primjerice, od svih slojevitih mreža s logističkim prijenosnim funkcijama

Izgrađujući i razgrađujući algoritmi

- Za potrebe pronalaska optimalne arhitekture postoje heuristički algoritmi:
  - izgrađujući – kreću od minimalne mreže pa je povećavaju
  - razgrađujući – kreću od vrlo općenite mreže pa uklanjaju nepotrebne dijelove
- Niz problema:
  - jaka ovisnost o upotrebljenim heuristikama
  - smislenost heuristike
  - problem globalne optimalnosti rješenja
- Ideja: formulirati problem kao optimizacijski pa primijeniti algoritme evolucijskog računanja koji su posebno pogodni za takvu vrstu problema

Karakteristike funkcije dobrote arhitekture

- Dobrotu arhitekture možemo mjeriti: jednostavnošću mreže, brzinom kojom klasični algoritam može naučiti takvu mrežu, kvalitetom uz koju mreža može rješavati zadani problem i slično, te kombinacijom tih elemenata
- Funkcija dobrote arhitekture ima sljedeće karakteristike:
  - beskonačna površina
  - nije derivabilna – gradijentne metode su neprimjenjive
  - kompleksna i sadrži veliku količinu šuma (problem mjerenja kvalitete)
  - decepcijska (slične arhitekture, bitno različita kvaliteta)
  - višemodalna (simetričnost mreža)

Evolucijski algoritam

- Odlučiti što se točno upisuje u kromosom (svaki kromosom predstavlja jednu mrežu)
- Potom radi sljedeće:
  1. generiraj početnu populaciju mreža
  2. dekodiraj svaki kromosom u odgovarajuću mrežu
  3. svaku mrežu treniraj puno puta
  4. temeljem rezultata treniranja svakoj mreži dodijeli pripadnu dobrotu
  5. u skladu s dobrotom biraj roditelje i generiraj djecu

Što se upisuje u kromosom

- Dvije krajnosti:
  - direktno kodiranje – u kromosom se upisuje sve, do na iznos svake pripadne težine
    $\Rightarrow$ istovremeno se evoluira i arhitektura i pripadne težine
  - indirektno kodiranje – u kromosom se upisuje samo grubi nacrt mreže, npr. broj slojeva i broj neurona po slojevima
    $\Rightarrow$ prilikom "instanciranja" mreže težine se dodjeljuju nekim pravilom ili se naprosto uče treniranjem mreže

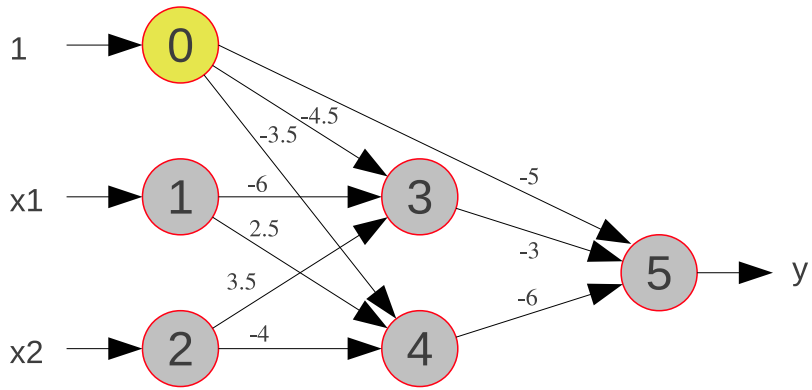
Direktno kodiranje

- Koristi se najčešće matrični prikaz: *matrica težina*

	0	**1**	**2**	**3**	**4**	**5**
0	0.0	0.0	0.0	$w_{0,3}$	$w_{0,4}$	$w_{0,5}$
1	0.0	0.0	0.0	$w_{1,3}$	$w_{1,4}$	0.0
2	0.0	0.0	0.0	$w_{2,3}$	$w_{2,4}$	0.0
3	0.0	0.0	0.0	0.0	0.0	$w_{3,5}$
4	0.0	0.0	0.0	0.0	0.0	$w_{4,5}$
5	0.0	0.0	0.0	0.0	0.0	0.0

	0	**1**	**2**	**3**	**4**	**5**
0	0.0	0.0	0.0	-4.5	-3.5	-5.0
1	0.0	0.0	0.0	-6.0	2.5	0.0
2	0.0	0.0	0.0	3.5	-4.0	0.0
3	0.0	0.0	0.0	0.0	0.0	-3.0
4	0.0	0.0	0.0	0.0	0.0	-6.0
5	0.0	0.0	0.0	0.0	0.0	0.0

Opći oblik unaprijedne mreže



Indirektno kodiranje

- Kodira se samo grubi nacrt mreže
- Više mogućnosti; dvije najčešće su:
  - *parametarski prikaz*
  - *razvojno pravilo*

Indirektno kodiranje: parametarski prikaz

- Pohranjuju se samo općenite informacije o mreži
- Primjerice, neku slojevitu unaprijednu mrežu možemo zapisati $2 \times 7 \times 3 \times 1$; to je mreža koja ima:
  - 4 sloja
  - 1. (ulazni) sadrži dva neurona
  - 2. (skriveni) sadrži sedam neurona
  - 3. (skriveni) sadrži tri neurona
  - 4. (izlazni) sadrži jedan neurona
  - mreža radi preslikavanje $\mathbb{R}^2 \rightarrow \mathbb{R}$
- sve ostalo se postavlja na pretpostavljene vrijednosti; npr.:
  - povezivost $\rightarrow$ potpuna (između slojeva)
  - prijenosne funkcije $\rightarrow$ logističke

Indirektno kodiranje: parametarski prikaz

- U kromosomu nemamo informaciju o vrijednostima težinskih faktora
- Da bismo došli do težina, trebamo napraviti postupak učenja mreže
- Da bismo procijenili kvalitetu arhitekture, trebamo težine kako bismo izračunali pogrešku
- Naučene težine ovise o tome koliko smo imali sreće prilikom učenja
  $\Rightarrow$ to je uzrok šuma u ovom optimizacijskom procesu
  $\Rightarrow$ ako nisam imao sreće, naučio sam loše težine iako je arhitektura dobra – dobit će slab iznos funkcije dobrote

Indirektno kodiranje: razvojna pravila

- Razvojna pravila definiraju kako izgraditi arhitekturu mreže
- Možemo na njih gledati kao na "program" koji kad izvršimo dobijemo arhitekturu
- Pretpostavimo da arhitekturu prikazujemo matricom veza
  - element matrice je 0 ako nema veze između para neurona, 1 ako veza postoji (iznos težine nije bitan)
- razvojno pravilo može biti rekurzivna jednadžba koja iz koraka u korak popunjava (ili gradi) konačnu matricu
  - u tom slučaju evoluiraju se koeficijenti u rekurzivnom izrazu i samo se oni upisuju u kromosom
- Matricu veza možemo graditi i produkcijskim sustavom (ako-onda pravilima)
  - u tom slučaju evoluiraju se pravila produkcijskog sustava
  - jedan kromosom predstavlja čitav produkcijski sustav

Indirektno kodiranje: razvojna pravila

$$S \longrightarrow \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

$$A \longrightarrow \begin{pmatrix} a & a \\ a & a \end{pmatrix} \quad B \longrightarrow \begin{pmatrix} i & i \\ i & a \end{pmatrix} \quad C \longrightarrow \begin{pmatrix} i & a \\ a & c \end{pmatrix} \quad D \longrightarrow \begin{pmatrix} a & e \\ a & e \end{pmatrix} \quad \cdots$$

$$a \longrightarrow \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \quad c \longrightarrow \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \quad e \longrightarrow \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \quad i \longrightarrow \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \cdots$$

$$S$$

(a)

$$\begin{matrix} A & B \\ C & D \end{matrix}$$

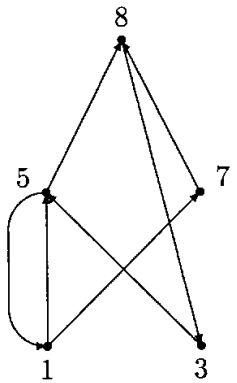
(b)

$$\begin{matrix} a & a & i & i \\ a & a & i & a \\ i & a & a & e \\ a & c & a & e \end{matrix}$$

(c)

$$\begin{matrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

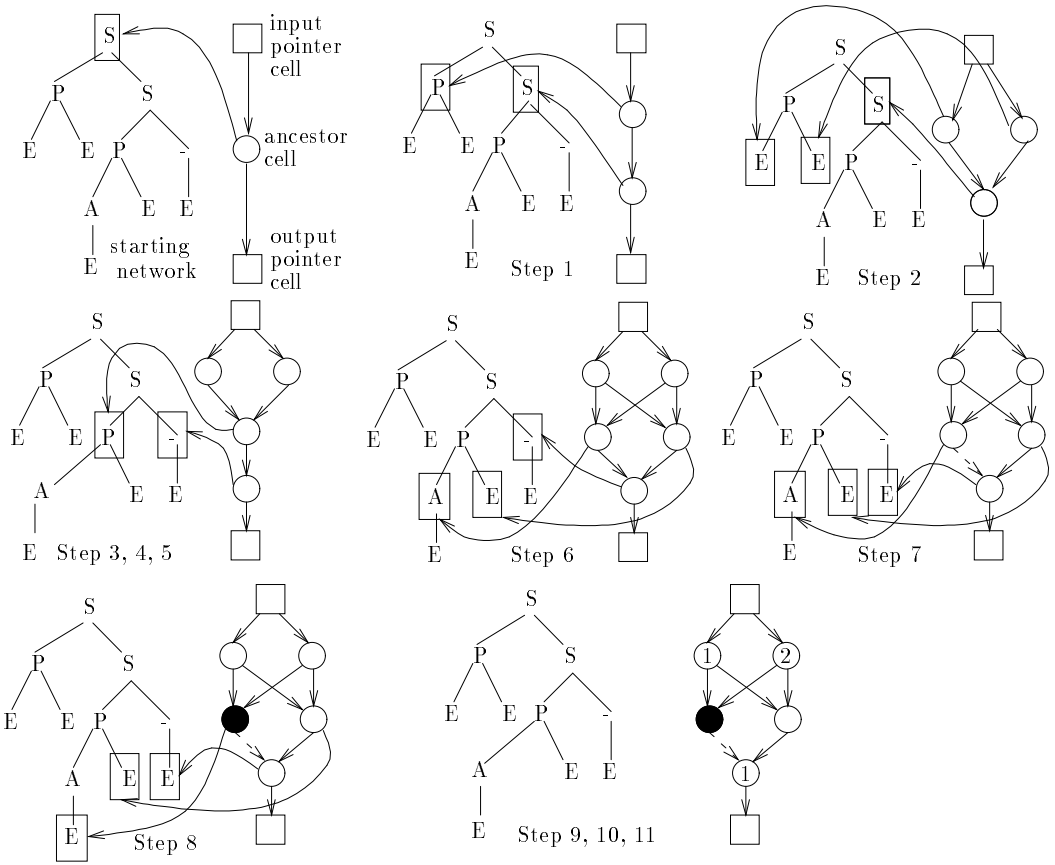
(d)



(e)

Indirektno kodiranje: razvojna pravila

- Zanimljiv pristup je i *stanični razvoj*



Što želimo

- Uz fiksno pravilo učenja: algoritmom evolucijskog računanja možemo tražiti optimalne parametre pravila (npr. kod algoritma *Backpropagation* stopa učenja i faktor inercije)
- Uz dostupne lokalne informacije svakog neurona (što mu je na ulazu, što mu je na izlazu i slično) neko drugo pravilo učenja koje radi bolje $\Rightarrow$ pogledati što piše u skripti!

Zaključak

- Vrlo interesantno područje koje je još uvijek u razvoju